

acm International Collegiate Programming Contest

IBM®

event
sponsor

Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Biológicas
Departamento de Computação

Maratona de Programação
Algoritmos e Programação Avançada

Universidade Federal de Outro Planeta
Professor Marco Antonio Moreira de Carvalho
Professor Túlio Toffolo

Ouro Preto
Maio de 2012

Sumário

I Códigos	1
1 Default	1
1.1 C++	1
1.2 Java	2
1.3 C	3
2 Algoritmos	4
2.1 Arredondamento e Precisão	4
2.2 Binary Conversor	4
2.3 Bit Manipulation	5
2.3.1 C++	5
2.3.2 Java	7
2.4 Cálculo Numérico	9
2.4.1 Método da Bisseção	9
2.4.2 Método da Secante	9
2.4.3 Método de Newton	10
2.5 Conversor de Graus	10
2.6 Criação de um <i>set</i> com função de comparação:	10
2.7 Counting Sort	10
2.8 Impressão em 'mod x'	11
2.9 Ordenação (1 to N)	12
2.10 Heap Sort	12
2.11 qSort	13
2.12 Simplex	14
2.13 STL Algorithms	16
2.14 Torres de Hanoi	18
2.15 Varredura de Retângulos	20
3 Backtracking	21
3.1 8-Rainhas	21
3.2 Clique-Máximo	22
3.3 Select Three	23
3.4 TSP	24
4 Dividir para Conquistar	25
4.1 Busca Binária	25
4.2 Contador de Inversões	25
4.2.1 Versão Comum	25
4.2.2 Versão Alterada	26
4.2.3 Versão usando Árvore	27
4.3 Multiplicação de Matrizes	28
4.4 Exponenciação	30

5 Estruturas de Dados	30
5.1 BigInteger	30
5.1.1 C++	30
5.1.2 Java	35
5.2 Collections	38
5.3 Declaração de Vetores e Matrizes	38
5.4 Fenwick Tree	39
5.4.1 C++	39
5.4.2 Java	39
5.5 Hashtable	41
5.6 Heap / Fila de Prioridades	42
5.6.1 C++	42
5.6.2 Java	43
5.7 Map and Set	44
5.7.1 C++	44
5.7.2 Java	45
5.8 memset	46
5.9 Queue and Stack	46
5.9.1 C++	46
5.9.2 Java	47
5.10 Segment Tree	47
5.10.1 C++	47
5.10.2 Java	49
5.11 Set Insertion	51
5.12 Static Array Vs. Vector	51
5.12.1 C++	51
5.12.2 Java	52
5.13 Trie Tree	52
5.13.1 C++	52
5.13.2 Java	55
6 Grafos	60
6.1 Bellman-Ford	60
6.2 Detecção de Árvores e Ciclos	61
6.3 Detecção de Ciclo Euleriano	61
6.4 Dijkstra	61
6.4.1 C	61
6.4.2 C++	65
6.5 Dijkstra Guloso – 1 p/ 1	67
6.6 Edmonds Karp's	68
6.7 Euleriano	70
6.8 Flood Fill	71
6.9 Floyd Warshall	72
6.10 Fluxo Máximo + Corte Mínimo	73
6.11 Fluxo Máximo + Peso nos Vértices	75
6.12 Grafo Bipartido	77
6.13 BFS	78
6.14 DFS	80

6.15	Kruskal	85
6.15.1	Implementação do Livro (inc. Primm)	85
6.15.2	Outra	87
6.16	Menor Caminho em Árvores	89
6.16.1	DFS	89
6.16.2	Menor Ancestral Comum	91
6.17	MCBM (Max Cardinality Bipartite Matching)	92
6.18	Ordenação Topológica	94
6.19	Representações	94
6.20	Algoritmo de Tarjan	95
6.21	Union Find	97
6.22	Union Find	98
6.23	Union Find	99
6.24	Cordal	100
7	Grid	103
7.1	Order	103
7.2	Plates	105
8	Gulosos	109
8.1	Algoritmo de Huffman + Cláusulas de Horn	109
9	Math	113
9.1	Área da Esfera	113
9.2	Área do Triângulo, dados 3 pontos	113
9.3	Algoritmo de Briot-Ruffini	114
9.4	Algoritmo de Euclides	115
9.5	Ângulo entre duas Retas	116
9.6	Divisores de um número Natural	117
9.7	Exponenciação e Logaritmo	117
9.7.1	Regras da função exponencial	117
9.7.2	Propriedades da função logarítmica	117
9.8	Geometry	118
9.8.1	Estruturas de Pontos, Linhas e Círculos	118
9.8.2	Bola na Caixa	120
9.8.3	Caixa na Bola	121
9.8.4	Calcular $x^y \% n$	121
9.8.5	Circles Sample – Competitive Programming	121
9.8.6	Cycle-Finding - Pseudo-Random Numbers	122
9.8.7	Distância entre dois pontos (Esfera / 3D)	123
9.8.8	Raios do Círculo inscrito e circunscrito em um Triângulo	124
9.8.9	Triangle Area	125
9.8.10	Intersecting Circles	125
9.8.11	Intersecting Lines	126
9.8.12	Intersecting Segments	127
9.8.13	Teorema de Pitágoras Completo / Lei dos Cossenos	127
9.8.14	Polígonos	127
9.8.15	PI	131
9.8.16	Polygon in a circle	131

9.8.17 Triangles	131
9.8.18 Triângulo Acutângulo, Retângulo ou Obtusângulo	134
9.9 Infix to Post Sufix Equation	134
9.10 Mudança de Base	138
9.11 Multiplicação de Matrizes	138
9.12 Número de Euler	139
9.13 Números Primos – map impl.	139
9.14 Progressões	139
9.14.1 Aritméticas	139
9.14.2 Geométricas	140
9.15 Resto sem Divisão	140
9.16 Roman Numbers	141
9.17 Teste de Primalidade	141
9.17.1 Crivo de Eratóstenes com Fatoração	141
9.17.2 Fermat	146
9.17.3 \sqrt{n}	148
10 Programação Dinâmica	148
10.1 0-1 Knapsack	149
10.2 Bitmasks	150
10.3 Cutting Sticks (3.4.3)	151
10.4 Fishmonger (3.4.3)	152
10.5 Maximum Sum	153
10.6 Problema da Mochila	154
10.7 Subset Sum	156
11 Meta-Heurística: Roll and Improve	156
12 Strings	161
12.1 Distância de Edição	161
12.2 Highest-scoring Alignment	165
12.2.1 C++	165
12.2.2 Java	165
12.3 KMP	166
12.3.1 Implementação do Livro	166
12.3.2 Outra	167
12.4 Longest Common Substring	168
12.4.1 C++	168
12.4.2 Java	172
12.5 Longest Increasing Subsequence	177
12.5.1 C++	177
12.5.2 Java	177
13 Tutoriais	178
13.1 scanf	178
13.2 Java Scanner	179
II Exercícios Resolvidos	179

14 Ad-hoc	179
14.1 Ano Bissexto	179
14.2 Braile	179
14.3 Calculadora	179
14.4 Dobradura	180
14.5 Jogo da Vida	181
14.5.1 Especificação	181
14.5.2 Solução	182
14.6 Ones	187
14.7 Posição em permutações / anagramas	188
14.8 Roman Counting	190
14.9 Sudoku	191
14.10 Vetor para Reversão - decodificação	193
14.11 Xadrez	195
15 Busca Exaustiva	197
15.1 8-Rainhas	197
15.2 Crypt Kicker	198
16 Dividir para Conquistar	200
16.1 Roteadores	200
17 Estruturas de Dados	202
17.1 BigInteger – Krakovia	202
17.2 Pilha – Apaga	203
17.2.1 Especificação	203
17.2.2 Solução	204
17.3 Pilha – Estação	205
18 Grafos	206
18.1 BFS with Levels – Busca ao Tesouro	206
18.1.1 Especificação	206
18.1.2 Solução	209
18.2 BFS with Path	211
18.3 Contour Painting	212
18.3.1 Especificação	212
18.3.2 Solução	216
18.4 Dominó – BFS em grafo direcionado	219
18.5 Escalonamento Ótimo – Ordenação Topológica	221
18.5.1 Especificação	221
18.5.2 Solução	224
18.6 Floyd-Warshall	225
18.7 Geração de Valores com BFS	226
18.8 Knight Moves	227
18.9 Ordenação Topológica	230
18.10 Ordenação Topológica – Min e Max	232
18.10.1 Gabriel's	232
18.11 Pontos de Articulação	234
18.12 Pontos de Articulação em Grafos Desconexos	236

19 Gulosos	237
19.1 Packs 6x6	237
20 Matemática	240
20.1 Geometry	240
20.1.1 Coordenadas Geográficas	240
20.1.2 Distância de Manhattan	241
20.1.3 Furos	242
20.1.4 Interseção de Linhas	245
20.1.5 Interseção total de quadrados	247
20.1.6 Ponto → Círculo → Ponto	248
20.1.7 TV da Vovó	254
20.2 MDC (BigInteger)	256
20.3 Modular Fibonacci	256
20.3.1 Especificação	256
20.3.2 Solução	256
21 Programação Dinâmica	258
21.1 Altitude	258
21.2 Bottom-Up – Exemplo	259
21.3 Coin Change	259
21.4 Flight Simulator	260
21.4.1 Especificação	260
21.4.2 Solução	263
21.5 Geração de Valores com Recursos Limitados	264
21.5.1 Especificação	264
21.5.2 Solução	266
21.6 Geração de Valores com Recursos Ilimitados	268
21.6.1 Especificação	268
21.6.2 Solução	268
21.7 Maximum Sum 2D (Matrix)	269
21.8 Maximum Sum with Range	270
21.9 <i>nWays</i> - Ilha do Cubo	271
21.10 Série de Cantor / Count or Cantor	272
21.10.1 Especificação	272
21.10.2 Solução	274
21.11 Soma de dois == k	274
21.12 Stacking Boxes	275
21.12.1 Especificação	275
21.12.2 Solução	278
21.13 Supermarket	279
21.14 Top-Down – Exemplos	280
21.15 Wedding Shopping	284
21.15.1 Especificação	284
21.15.2 Solução	284
III UVa & Spoj	286

22 2-SAT	286
22.1 Spoj-BR - CARDAPIO (Algoritmo de Tarjan)	286
23 Ad-hoc	288
23.1 UVa - (120) Stacks of Flapjacks	288
23.2 UVa - (200) Rare Order	290
23.3 UVa - (11455) Behold my quadrangle	293
23.4 Spoj-BR - APAGA	294
23.5 Spoj-BR - CUBOS	295
23.6 Spoj-BR - LASERR	299
23.7 Spoj-BR - LOOPMUSI	299
23.8 Spoj-BR - PARIDADE	301
24 AGM	301
24.1 Spoj-BR - CIPO	301
24.2 Spoj-BR - INTERLMG	303
25 BFS	306
25.1 Spoj-BR - FORRO	306
25.2 Spoj-BR - NUMERDOS	310
25.3 Spoj-BR - TESOUR11	313
26 Busca Exaustiva	316
26.1 Spoj-BR - EUROVIMG	316
26.2 Spoj-BR - JUNINA (Clique Máximo)	318
27 Dijkstra	319
27.1 Spoj - SHPATH	319
28 Fluxo Máximo	323
28.1 Spoj-BR - CAVALOS	323
29 Game Theory	325
29.1 Spoj-BR - CHOC09	325
29.2 Spoj-BR - CHOCPJ09	326
29.3 Spoj-BR - MARIENBA	327
30 Grafo	328
30.1 Spoj-BR - TEOBALDO	328
31 Grafo de Intervalos	330
31.1 Spoj-BR - ATLETAMG	330
32 Grid	332
32.1 UVa - (121) Pipe Filters	332
33 Java BigInteger	333
33.1 Spoj - FCTRL2	333
33.2 Spoj - JULKA	334

34 Java GregorianCalendar	334
34.1 UVa - (11947) Cancer of Scorpio	334
35 Joseph	338
35.1 UVa - (151) Power Crisis	338
35.2 UVa - (305) Joseph	339
35.3 UVa - (10015) Joseph Cousin	341
36 Math	342
36.1 Spoj-BR - COMCAMEL	342
36.2 Spoj-BR - FATORIAL	345
36.3 Spoj-BR - MACACO	346
37 Median Generator + LIS	347
37.1 Spoj-BR - ACOES2MG	347
38 Ordenação Topológica	351
38.1 Spoj-BR - ESCALO11	351
38.2 Spoj-BR - TAREFMG	353
39 Percurso em Árvores	356
39.1 Spoj-BR - PREEMPOS	356
40 Pontos de Articulação	357
40.1 Spoj-BR - MANUT	357
41 Programação Dinâmica	359
41.1 Spoj-BR - DOCE	359
41.2 Spoj-BR - MINHOCA2	360
41.3 Spoj-BR - ROBUSMG	361
42 Segment Tree	363
42.1 Spoj - LITE (with Lazy Propagation)	363
42.2 Spoj-BR - BANFARAO	365
42.3 Spoj-BR - HOMEM (with Lazy Propagation)	370
43 Simulation	373
43.1 UVa - (10901) Ferry Loading III	373
44 String	376
44.1 UVa - (11221) Magic square palindromes	376
44.2 Spoj-BR - DICC11	377
44.3 Spoj-BR - PAL	379
44.4 Spoj-BR - PLAGIO	381
45 Union Set	383
45.1 Spoj-BR - FUSOES1	383

IV	Most Recent 2.0	385
45.2	all different	385
45.3	Almost	387
45.4	Tree DP	389
45.5	Campus – Articulation Points	390
45.6	Cads – MiniMax	392
45.7	Clock Angle	393
45.8	Couples – TLE – Gabriel	395
45.9	Couples – TLE – Leandro	397
45.10	Prefix Tree	401
45.11	Interval Product – Gabriel – 1	403
45.12	Interval Product – Gabriel – 2	405
45.13	Interval Product – Fenwick Tree / Binary Indexed Tree	407
45.14	Interval Product – Segmentation Tree	409
45.15	Kids – Gabriel	416
45.16	Kids – Leandro	418
45.17	Show sequence	421
45.18	Proportional [0,100]	423
45.19	Tree Simulation (Space Elevator)	425
45.20	Space Elevator – Internet Solution	429
V	Anexos	432
VI	Physical Cheat	433
VII	Math Cheat	434

Parte I

Códigos

1 Default

1.1 C++

```
/*
 * File :      main.cpp
 * Author :    ubuntulap
 *
5   * Created on February 28, 2008, 6:16 PM
 */

#include <stdlib.h>
#include <stdio.h>
10 #include <string.h>
#include <math.h>
#include <limits.h>
#include <assert.h>
#include <cctype>
15 #include <iomanip>
#include <iostream>
#include <string>
#include <algorithm>
#include <utility>
20 #include <iterator>
#include <numeric>
#include <sstream>
#include <fstream>
#include <bitset>
25 #include <map>
#include <vector>
#include <set>
#include <stack>
#include <queue>
30 #include <deque>
#include <list>

#define SWAP(a,b) ( (a)=(b), (b)=(a), (a)=(b) )
#define MIN(a,b) ( (a)<(b) ? (a) : (b) )
35 #define MAX(a,b) ( (a)>(b) ? (a) : (b) )
#define ABS(a) ( (a)<(0) ? (-1)*(a) : (a) )
#define ROUND(a) ( (a)>=(0) ? ((int)((double)(a) + 0.5)) : ((int)((
    double)(a) - 0.5)) )

#define FI(a) fastint(&a)
40 #define PL printf(" | %n")

#define MAXSTRSZ      100000
#define MAXNUMBER     200000
#define INFINITO      999999999
45 #define EPS          1e-9
#define PI            acos(-1.0)
```

```

using namespace std;

50  typedef long long ll;
typedef unsigned long long ull;
typedef pair<int,int> ii;
typedef vector< pair<int,int> > vii;
typedef vector<int> vi;

55  void fastint(register int *n)
{
    register char c;
    register int neg = 0;
60  *n = 0;

    while(c = getc(stdin), c < '0' || c > '9')
        neg |= c == '_';

65  do
{
    (*n) = (*n)*10 + (c - '0');
} while(c = getc(stdin), c >= '0' && c <= '9');

70  (*n) = neg?(*n)*(-1):(*n);
}

/*
 */
75  int main()
{
80  return 0;
}

```

1.2 Java

```

/*
 * File:   main.java
 * Author: ubuntulap
 *
5  * Created on February 28, 2008, 6:16 PM
 */

import java.math.*;
import java.util.*;

10 public class Main
{
    public static void main(String[] args)
    {
15     Scanner sc = new Scanner(System.in);

        }
}

```

1.3 C

```
/*
 * File : main.cpp
 * Author : ubuntulap
 *
5  * Created on February 28, 2008, 6:16 PM
 */

#include <stdlib.h>
#include <stdio.h>
10 #include <string.h>
#include <math.h>
#include <limits.h>
#include <assert.h>
#include <cctype>
15 #include <iomanip>
#include <iostream>
#include <string>
#include <algorithm>
#include <utility>
20 #include <iterator>
#include <numeric>
#include <sstream>
#include <fstream>
#include <bitset>
25 #include <map>
#include <vector>
#include <set>
#include <stack>
#include <queue>
30 #include <deque>
#include <list>

#define SWAP(a,b) ( (a)=(b), (b)=(a), (a)=(b) )
#define MIN(a,b) ( (a)<(b) ? (a) : (b) )
35 #define MAX(a,b) ( (a)>(b) ? (a) : (b) )
#define ABS(a) ( (a)<(0) ? (-1)*(a) : (a) )
#define ROUND(a) ( (a)>=(0) ? ((int)((double)(a) + 0.5)) : ((int)((
    double)(a) - 0.5)) )

#define FI(a) fastint(&a)
40 #define PL printf( "|n" )

#define MAXSTRSZ 100000
#define MAXNUMBER 200000
#define INFINITO 999999999
45 #define EPS 1e-9
#define PI acos(-1.0)

using namespace std;

50 typedef long long ll;
typedef unsigned long long ull;
typedef pair<int,int> ii;
typedef vector< pair<int,int> > vii;
typedef vector<int> vi;
55
```

```

void fastint(register int *n)
{
    register char c;
    register int neg = 0;
60    *n = 0;

    while(c = getc(stdin), c < '0' || c > '9')
        neg |= c == '_';

65    do
    {
        (*n) = (*n)*10 + (c - '0');
    } while(c = getc(stdin), c >= '0' && c <= '9');

70    (*n) = neg?(*n)*(-1):(*n);
}

/*
 */
75 int main()
{
80    return 0;
}

```

2 Algoritmos

2.1 Arredondamento e Precisão

```

double x = 0.085;
double m = pow(10.0, d);
x = (int)(x*m + 0.5) / m;
5 cout << fixed << setprecision(3) << x << endl;
printf("%.3lf\n", x);

```

2.2 Binary Conversor

```

int binary[106];

void toBinary(long long key, long long* firstDigit)
{
    long long i = 0, div = key;
    while(div > 0)
    {
        binary[i++] = div%2;
        div >>= 1;
10    }
    *firstDigit = i-1;
}

```

```

15   int main()
{  

    long long key, fd, tot, i;  

    while(1)  

    {  

        scanf( "%lld ", &key);  

        if(!key)  

            break;  

        toBinary(key, &fd);  

        tot = 0;  

        printf( "The parity of ");  

        for(i = fd; i>=0; i--)  

        {  

            printf( "%d", binary[i]);  

            tot+=binary[i];  

        }  

        printf( " is %lld (mod 2). |n", tot);  

    }  

    return 0;
}

```

2.3 Bit Manipulation

2.3.1 C++

```

#define isOn(S, j) (S & (1 << j))
#define setBit(S, j) (S |= (1 << j))
#define clearBit(S, j) (S &= ~(1 << j))
#define toggleBit(S, j) (S ^= (1 << j))
5 #define lowBit(S) (S & (-S))
#define setAll(S, n) (S = (1 << n) - 1)

#define modulo(x, N) ((x) & (N - 1))
#define isPowerOfTwo(x) ((x & (x - 1)) == 0)
10 #define nearestPowerOfTwo(x) ((int)pow(2.0, (int)((log((double)x) /
log(2.0)) + 0.5)))

void printSet(int _S)
{
    printf( "S = %2d = ", _S);
    stack<int> st;
    while (_S)
15     st.push(_S % 2), _S /= 2;
    while (!st.empty())
        printf( "%d", st.top()), st.pop();
    printf( " |n");
20 }
}

int main()
{
    int S, T;
25

```

```

printf("1. Representation (all indexing are 0-based and counted
      from right)|n");
S = 34; printSet(S);
printf("|n");

30 printf("2. Multiply S by 2, then divide S by 4 (2x2), then by 2|n
      ");
S = 34; printSet(S);
S = S << 1; printSet(S);
S = S >> 2; printSet(S);
S = S >> 1; printSet(S);
printf("|n");

printf("3. Set/turn on the 3-th item of the set|n");
S = 34; printSet(S);
setBit(S, 3); printSet(S);
printf("|n");

printf("4. Check if the 3-th and then 2-nd item of the set is on
      ?|n");
S = 42; printSet(S);
T = isOn(S, 3); printf("T = %d, %s|n", T, T ? "ON" : "OFF");
T = isOn(S, 2); printf("T = %d, %s|n", T, T ? "ON" : "OFF");
printf("|n");

50 printf("5. Clear/turn off the 1-st item of the set|n");
S = 42; printSet(S);
clearBit(S, 1); printSet(S);
printf("|n");

printf("6. Toggle the 2-nd item and then 3-rd item of the set|n")
      ;
S = 40; printSet(S);
toggleBit(S, 2); printSet(S);
toggleBit(S, 3); printSet(S);
printf("|n");

60 printf("7. Check the first bit from right that is on|n");
S = 40; printSet(S);
T = lowBit(S); printf("T = %d (this is always a power of 2)|n", T
      );
S = 52; printSet(S);
T = lowBit(S); printf("T = %d (this is always a power of 2)|n", T
      );
printf(|n);

65 printf("8. Turn on all bits in a set of size n = 6|n");
setAll(S, 6); printSet(S);
printf(|n);

70 printf("9. Other tricks (not shown in the book)|n");
printf("8 %c 4 = %d|n", '%', modulo(8, 4));
printf("7 %c 4 = %d|n", '%', modulo(7, 4));
printf("6 %c 4 = %d|n", '%', modulo(6, 4));
75 printf("5 %c 4 = %d|n", '%', modulo(5, 4));
printf("is %d power of two? %d|n", 9, isPowerOfTwo(9));
printf("is %d power of two? %d|n", 8, isPowerOfTwo(8));

```

```

    printf("is %d power of two? %d\n", 7, isPowerOfTwo(7));
    for (int i = 0; i <= 16; i++)
        printf("Nearest power of two of %d is %d\n", i,
               nearestPowerOfTwo(i));
    printf("\n");
}

return 0;
}

```

2.3.2 Java

```

class ch2_04_bit_manipulation
{
    private static int setBit(int S, int j) { return S | (1 << j); }

    private static int isOn(int S, int j) { return S & (1 << j); }

    private static int clearBit(int S, int j) { return S & ~(1 << j); }

    private static int toggleBit(int S, int j) { return S ^ (1 << j); }

    private static int lowBit(int S) { return S & (-S); }

    private static int setAll(int n) { return (1 << n) - 1; }

    private static int modulo(int x, int N) { return ((x) & (N - 1)); }

    private static int isPowerOfTwo(int x) { return (x & (x - 1)); }

    private static int nearestPowerOfTwo(int x) { return ((int) Math.
        pow(2.0, (int)((Math.log((double)x) / Math.log(2.0)) + 0.5))); }

    private static void printSet(int _S)
    {
        System.out.printf("S = %2d = ", _S);
        Stack<Integer> st = new Stack<Integer>();
        while (_S > 0)
        {
            st.push(_S % 2);
            _S /= 2;
        }
        while (!st.empty())
        {
            System.out.printf("%d", st.peek());
            st.pop();
        }
        System.out.printf("\n");
    }

    public static void main(String[] args)
    {
        int S, T;
    }
}

```

```

System.out.printf("1. Representation (all indexing are 0-based
                  and counted from right)|n");
S = 34; printSet(S);
System.out.printf("|n");

45      System.out.printf("2. Multiply S by 2, then divide S by 4 (2x2)
                  , then by 2|n");
S = 34; printSet(S);
S = S << 1; printSet(S);
S = S >> 2; printSet(S);
S = S >> 1; printSet(S);
System.out.printf("|n");

System.out.printf("3. Set/turn on the 3-th item of the set|n");
S = 34; printSet(S);
S = setBit(S, 3); printSet(S);
System.out.printf("|n");

55      System.out.printf("4. Check if the 3-th and then 2-nd item of
                  the set is on?|n");
S = 42; printSet(S);
T = isOn(S, 3); System.out.printf("T = %d, %s|n", T, T != 0 ? "
ON" : "OFF");
T = isOn(S, 2); System.out.printf("T = %d, %s|n", T, T != 0 ? "
ON" : "OFF");
System.out.printf("|n");

System.out.printf("5. Clear/turn off the 1-st item of the set|n
                  ");
65      S = 42; printSet(S);
S = clearBit(S, 1); printSet(S);
System.out.printf("|n");

System.out.printf("6. Toggle the 2-nd item and then 3-rd item
                  of the set|n");
70      S = 40; printSet(S);
S = toggleBit(S, 2); printSet(S);
S = toggleBit(S, 3); printSet(S);
System.out.printf("|n");

System.out.printf("7. Check the first bit from right that is on
                  |n");
75      S = 40; printSet(S);
T = lowBit(S); System.out.printf("T = %d (this is always a
power of 2)|n", T);
S = 52; printSet(S);
T = lowBit(S); System.out.printf("T = %d (this is always a
power of 2)|n", T);
System.out.printf("|n");

80      System.out.printf("8. Turn on all bits in a set of size n = 6|n
                  ");
S = setAll(6); printSet(S);
System.out.printf("|n");

85      System.out.printf("9. Other tricks (not shown in the book)|n");
System.out.printf("8 %c 4 = %d|n", '%', modulo(8, 4));
System.out.printf("7 %c 4 = %d|n", '%', modulo(7, 4));

```

```

90     System.out.printf("6 %c 4 = %d\n", '%', modulo(6, 4));
91     System.out.printf("5 %c 4 = %d\n", '%', modulo(5, 4));
92     System.out.printf("is %d power of two? %d\n", 9, isPowerOfTwo
93         (9));
94     System.out.printf("is %d power of two? %d\n", 8, isPowerOfTwo
95         (8));
96     System.out.printf("is %d power of two? %d\n", 7, isPowerOfTwo
97         (7));
98     for (int i = 0; i <= 16; i++)
99         System.out.printf("Nearest power of two of %d is %d\n", i,
100             nearestPowerOfTwo(i));
101    System.out.printf("\n");
102}

```

2.4 Cálculo Numéricico

2.4.1 Método da Bisseção

```

int p,q,r,s,t,u;

double f(double x){
    return p*exp(-x) + q*sin(x) + r*cos(x) + s*tan(x) + t*x*x + u;
}

double bisection(){
    double lo = 0, hi = 1;
    while (lo + EPS < hi){
        double x = (lo+hi)/2;
        if (f(lo) * f(x) <= 0){
            hi = x;
        } else {
            lo = x;
        }
    }
    return (lo+hi)/2;
}

int main(){
    while (scanf("%d %d %d %d %d %d", &p ,&q ,&r ,&s ,&t ,&u)!=EOF){
        if (f(0) * f(1) > 0){
            puts("No solution ");
        } else {
            printf("%.4lf\n", bisection());
        }
    }
}

```

2.4.2 Método da Secante

```

double secant(){
    if (f(0)==0) return 0;
    for (double x0=0, x1=1; ; ){
        double d = f(x1)*(x1-x0)/(f(x1)-f(x0));
        if (fabs(d) < EPS) return x1;
    }
}

```

```

        x0 = x1;
        x1 = x1 - d;
    }
}

```

2.4.3 Método de Newton

```

double fd(double x){
    return -p*exp(-x) + q*cos(x) - r*sin(x) + s/(cos(x)*cos(x)) + 2*t
           *x;
}

5   double newton(){
    if (f(0)==0) return 0;
    for (double x=0.5; ;){
        double x1 = x - f(x)/fd(x);
        if (fabs(x1-x) < EPS) return x;
10      x = x1;
    }
}

```

2.5 Conversor de Graus

$$K = C + 273 \quad (1)$$

$$\frac{C}{100} = \frac{F - 32}{180} \quad (2)$$

2.6 Criação de um *set* com função de comparação:

```

struct Cmp
{
    bool operator ()(const pair<int, int> &a, const pair<int, int>
                      &b)
    {
        return a.first < b.first;
    }
};

5   multiset< ii, Cmp > pq;

```

2.7 Counting Sort

- allocate an array $Count[0..k]$; initialize each array cell to 0;
- for each input item x :
 - $Count[\text{key}(x)] = Count[\text{key}(x)] + 1$;
- $\text{total} = 0$;

- for $i = 0, 1, \dots, k$:
 - $c = \text{Count}[i]$
 - $\text{Count}[i] = \text{total}$
 - $\text{total} = \text{total} + c$
- allocate an output array $\text{Output}[0..n-1]$;
- for each input item x :
 - store x in $\text{Output}[\text{Count}[\text{key}(x)]]$
 - $\text{Count}[\text{key}(x)] = \text{Count}[\text{key}(x)] + 1$
- return Output

2.8 Impressão em 'mod x'

```

int receita[TAM];

int main()
{
5   int inst, i, j, n, c;

  unsigned long long acc, aux;

  scanf("%d", &inst);
10
  for(c = 0; c<inst; c++)
  {
    scanf("%d", &n);

    for(i = 0; i<n; i++)
    {
      scanf("%d", &(receita[i]));
    }

    acc = 0;
    for(i = 0; i<n; i++)
    {
      if(i == 0)
    {
25      aux = 0;
      for(j = i; j<n; j++)
      {
        aux = (aux + receita[j])%1300031;
      }
    }
      else
    {
      aux = ((aux+1300031*1000) - receita[i-1]) % 1300031;
    }
30
      acc = (acc + receita[i]*aux)%1300031;
    }
  }
}

```

```

40     printf( "%lld\n", acc);
    }

    return 0;
}

```

2.9 Ordenação (1 to N)

Deve-se haver N elementos com valores de 1 a N, sem repetição.

```

int main()
{
    long long i, j, k;
    char* read = (char*)malloc((MAXSTRSZ+1)*sizeof(char));
    int* vetor = (int*)malloc(10050*sizeof(int));

    int n;
    scanf("%d", &n);
    for(int c = 0; c < n; c++)
    {
        scanf("%lld", &k);

        for(i = 0; i < k; i++)
        {
            scanf("%d", &(vetor[i]));
        }

        int cont = 0;

        for(i = 0; i < k; i++)
        {
            while(vetor[i] != i+1)
            {
                int pos1 = i;
                int pos2 = vetor[i]-1;

                int aux = vetor[pos1];
                vetor[pos1] = vetor[pos2];
                vetor[pos2] = aux;

                cont++;
            }
        }

        printf("%d\n", cont);
    }

    return 0;
}

```

2.10 Heap Sort

```

vector<int> x, y;

```

```

5      /*
6      */
7  int main()
8  {
9      int n;
10     int i, cont = 1, mid;
11
12     while(fastint(&n), n)
13     {
14         x.resize(n), y.resize(n);
15
16         for(i = 0; i <n; i++)
17             fastint(&(x[i])), fastint(&(y[i]));
18
19         mid = n >> 1;
20
21         partial_sort(x.begin(), x.begin()+mid+1, x.end());
22         partial_sort(y.begin(), y.begin()+mid+1, y.end());
23
24         printf("Teste %d | n%d %d | n | n", cont++, x[mid], y[mid]);
25     }
26
27     return 0;
28 }
```

2.11 qSort

```
5      typedef struct
{  
    int id;  
    int time;  
} TCar;  
  
10     TCar cars[106];  
  
15     int comp(const void * a, const void * b)  
{  
    return ( (*(TCar*)a).time - (*(TCar*)b).time );  
}  
  
20     int main()  
{  
    int n, m, i, j, val;  
  
    scanf( "%d%d ", &n, &m );  
  
25     cars[0].id = -1;  
    cars[0].time = -1;  
    for(i = 1; i<=n; i++)  
    {  
        cars[i].id = i;  
        cars[i].time = 0;  
    }  
}
```

```

10    for(i = 1; i<=n; i++)
11    {
12      for(j = 0; j<m; j++)
13      {
14        scanf( "%d ", &val);
15        cars[i].time+=val;
16      }
17    }
18
19    qsort(cars, n+1, sizeof(TCar), comp);
20
21    printf( "%d | %d | %d | %n", cars[1].id, cars[2].id, cars[3].id);
22
23    return 0;
24
25  }

```

2.12 Simplex

```

1 #include <iostream>
2 #include <vector>
3 #include <algorithm>
4
5 using namespace std;
6
7 #define fori(i,n) for(int i=0; i < (n); ++i)
8 #define forr(i,a,b) for(int i=(a); i <= (b); ++i)
9 #define ford(i,a,b) for(int i=(a); i >= (b); --i)
10 #define sz size()
11
12 const double EPS=1e-9; const int INF = 0x3f3f3f3f;
13
14 #define all(x) (x).begin(),(x).end()
15
16 int cmpD(double x, double y=0, double tol=EPS) {
17     return (x <= y+tol) ? (x+tol<y) ? -1 : 0 : 1;
18 }
19
20 struct simplex {
21     simplex( const vector< vector< double > > & A_, const
22             vector< double > & b_,
23             const vector< double > & c_ ) : A( A_ ), b(
24               b_), c( c_) {}
25     vector< vector< double > > A; vector< double > b, c, sol;
26     vector< bool > N; vector< int > kt; int m, n;
27     void pivot( int k, int l, int e ) {
28         int x = kt[l]; double p = A[l][e];
29         fori(i,k) A[l][i] /= p;
30         b[l] /= p; N[e] = false;
31         fori(i,m) if (i != l) {b[i] -= A[i][e]*b[l]; A[i][x]
32           ] = -A[i][e]*A[l][x];}
33         fori(j,k) if ( N[j] ) {
34             c[j] -= c[e] * A[l][j];
35             fori(i,m) if ( i != l ) A[i][j] -= A[i][e]
36                           * A[l][j];
37         }
38         kt[l] = e; N[x] = true; c[x] = -c[e] * A[l][x];
39     }
40 }

```

```

35     }
40     vector< double > go( int k ) {
45         vector< double > res;
50         while ( 1 ) {
55             int e = -1, l = -1;
60             fori(i,k) if ( N[i] && cmpD( c[i] ) > 0 ) {
65                 e = i; break;
70                 if ( e == -1 ) break;
75                 fori(i,m) if ( cmpD(A[i][e]) > 0 && ( l ==
80                     -1 || cmpD( b[i] / A[i][e],
85                         b[l] / A[l]
90                         ][e], 1e
95                         -20 ) <
0 ) ) l
100                     = i;
105                     if ( l == -1 ) return vector< double >();
110                     pivot( k, l, e );
115                 }
120                 res.resize( k, 0 );
125                 fori(i,m) res[kt[i]] = b[i];
130                 return res;
135             }
140             vector< double > solve() {
145                 m = A.sz; n = A[0].sz; int k = m+n+1;
150                 N = vector< bool >( k, true ); vector< double >
155                     c_copy = c;
160                     c.resize(n+m); kt.resize(m);
165                     fori(i,m) {
170                         A[i].resize(k); A[i][n+i] = 1; A[i][k-1] =
175                         -1;
180                         kt[i] = n+i; N[kt[i]] = false;
185                     }
190                     int l = min_element(all(b)) - b.begin();
195                     if( cmpD(b[1] ) < 0 ) {
200                         c = vector<double>(k,0);
205                         c[k-1] = -1; pivot(k, 1, k-1); sol=go(k);
210                         if( cmpD(sol[k-1])>0) return vector<double
215                           >();
220                         fori(i,m) if(kt[i] == k-1) {
225                             fori(j,k-1) if(N[j] && cmpD( A[i][j
230                               ] ) != 0 ) {
235                                 pivot( k, i, j); break;
240                             }
245                         }
250                         c=c_copy; c.resize(k,0);
255                         fori(i,m) fori(j,k) if(N[j]) c[j] -= c[kt[i
260                               ]]*A[i][j];
265                         }
270                         sol = go(k-1);
275                         if(!sol.empty()) sol.resize(n);
280                         return sol;
285                     }
290                 }
295             };
300             int main() {
305                 /* Exemplo: Maximize cx Subject to Ax <= b */
310                 vector<vector<double> > A(9);
315                 double Av[] [3] = {{1,1,0}, {0,0,-1}, {-1,-1,0},

```

```

{0,0,1}, {1,0,0},
{0,1,0},
{0,0,1}, {1,0,1},
{0,1,0}};

85   for(int i=0; i < 9; i++) {
     A[i].insert(A[i].begin(), &(Av[i][0]), &(Av[i][3]))
     ;
}

90   vector<double > c(3, 1);
double bv[] = {2,-1,-2,1,2,1,1,2,1};
vector<double > b(bv, bv+sizeof(bv)/sizeof(double));

95   simplex sim(A,b,c);
vector<double> s = sim.solve();
if (!s.size()) cout << "Impossible!" |n";
else
100  for(int i=0; i < s.size(); i++) {
      cout << s[i] << endl;
}
}

```

2.13 STL Algorithms

```

typedef struct
{
    int id;
    int solved;
    int penalty;
} team;

bool icpc_cmp(team a, team b)
{
10   if (a.solved != b.solved)
    return a.solved > b.solved;
   else if (a.penalty != b.penalty)
    return a.penalty < b.penalty;
   else
15     return a.id < b.id;
}

int main()
{
20   int *pos, arr[] = {10, 7, 2, 15, 4};
   vector<int> v(arr, arr + 5);
   vector<int>::iterator j;

   sort(v.rbegin(), v.rend());
25   for (vector<int>::iterator it = v.begin(); it != v.end(); it++)
     printf("%d ", *it);
   printf("|\n");
   printf("=====|\n");

30   sort(arr, arr + 5);
   reverse(arr, arr + 5);
}

```

```

    for (int i = 0; i < 5; i++)
        printf("%d ", arr[i]);
    printf("\n");
    printf("=====\n");

35   random_shuffle(v.begin(), v.end());
    for (vector<int>::iterator it = v.begin(); it != v.end(); it++)
        printf("%d ", *it);
    printf("\n");
    printf("=====\n");

    partial_sort(v.begin(), v.begin() + 2, v.end());
    for (vector<int>::iterator it = v.begin(); it != v.end(); it++)
40      printf("%d ", *it);
    printf("\n");
    printf("=====\n");

    sort(arr, arr + 5);
50   for (int i = 0; i < 5; i++)
        printf("%d ", arr[i]);
    printf("\n");
    sort(v.begin(), v.end());
    for (vector<int>::iterator it = v.begin(); it != v.end(); it++)
        printf("%d ", *it);
    printf("\n");
    printf("=====\n");

    team nus[4] = { {1, 1, 10},
60      {2, 3, 60},
      {3, 1, 20},
      {4, 3, 60} };

    for (int i = 0; i < 4; i++)
65      printf("id: %d, solved:%d, penalty: %d\n",
nus[i].id, nus[i].solved, nus[i].penalty);

    sort(nus, nus + 4, icpc_cmp);
    printf("=====\n");
70   for (int i = 0; i < 4; i++)
      printf("id: %d, solved:%d, penalty: %d\n",
nus[i].id, nus[i].solved, nus[i].penalty);
    printf("=====\n");

75   typedef pair < int, pair < string, string > > state;
state a = make_pair(10, make_pair("steven", "grace"));
state b = make_pair(7, make_pair("steven", "halim"));
state c = make_pair(7, make_pair("steven", "felix"));
state d = make_pair(9, make_pair("a", "b"));
80   vector<state> test;
test.push_back(a);
test.push_back(b);
test.push_back(c);
test.push_back(d);
85   for (int i = 0; i < 4; i++)
      printf("value: %d, name1 = %s, name2 = %s\n",
test[i].first, ((string)test[i].second.first).c_str(),
((string)test[i].second.second).c_str());
    printf("=====\n");

```

```

sort(test.begin(), test.end());
90   for (int i = 0; i < 4; i++)
    printf("value : %d, name1 = %s, name2 = %s | n", test[i].first, ((string)test[i].second.first).c_str(), ((string)test[i].second.second).c_str());
    printf("=====|n");

pos = lower_bound(arr, arr + 5, 7);
95   printf("%d | n", *pos);
j = lower_bound(v.begin(), v.end(), 7);
printf("%d | n", *j);

pos = lower_bound(arr, arr + 5, 77);
100  if (pos - arr == 5)
    printf("77 not found | n");
j = lower_bound(v.begin(), v.end(), 77);
if (j == v.end())
    printf("77 not found | n");
printf("=====|n");

next_permutation(arr, arr + 5);
next_permutation(arr, arr + 5);
110  for (int i = 0; i < 5; i++)
    printf("%d ", arr[i]);
printf(" | n");

next_permutation(v.begin(), v.end());
next_permutation(v.begin(), v.end());
115  for (vector<int>::iterator it = v.begin(); it != v.end(); it++)
    printf("%d ", *it);
printf(" | n");
printf("=====|n");

printf("min(10, 7) = %d | n", min(10, 7));
printf("max(10, 7) = %d | n", max(10, 7));

return 0;
}

```

2.14 Torres de Hanoi

```

void HanoiTower(int n, int orig, int dest, int temp, int* cont)
{
    if(n == 1)
        (*cont)++;
5     else
    {
        HanoiTower(n-1, orig, temp, dest, cont);
        (*cont)++;
        HanoiTower(n-1, temp, dest, orig, cont);
10    }
}

int hanoi[35];

```

```

15 | int main(int argc, char** argv)
{  

|   int nDiscs; /*, cont;  

|   int torres[4]; */  

|  

20 |   hanoi[1] = 1;  

|   hanoi[2] = 3;  

|   hanoi[3] = 7;  

|   hanoi[4] = 15;  

|   hanoi[5] = 31;  

25 |   hanoi[6] = 63;  

|   hanoi[7] = 127;  

|   hanoi[8] = 255;  

|   hanoi[9] = 511;  

|   hanoi[10] = 1023;  

30 |   hanoi[11] = 2047;  

|   hanoi[12] = 4095;  

|   hanoi[13] = 8191;  

|   hanoi[14] = 16383;  

|   hanoi[15] = 32767;  

35 |   hanoi[16] = 65535;  

|   hanoi[17] = 131071;  

|   hanoi[18] = 262143;  

|   hanoi[19] = 524287;  

|   hanoi[20] = 1048575;  

40 |   hanoi[21] = 2097151;  

|   hanoi[22] = 4194303;  

|   hanoi[23] = 8388607;  

|   hanoi[24] = 16777215;  

|   hanoi[25] = 33554431;  

45 |   hanoi[26] = 67108863;  

|   hanoi[27] = 134217727;  

|   hanoi[28] = 268435455;  

|   hanoi[29] = 536870911;  

|   hanoi[30] = 1073741823;  

50 |  

|   int teste = 1;  

|   while(1)  

| {  

|     scanf( "%d", &nDiscs);  

55 |  

|     if(!nDiscs)  

|       break;  

|  

|     /* cont = 0;  

|     torres[1] = nDiscs;  

60 |     torres[2] = torres[3] = 0;  

|  

|     HanoiTower(nDiscs, torres[1], torres[3], torres[2], &cont);  

|  

|     printf( " | hanoi[%d] = %d; | n ", nDiscs, cont); */  

|  

|     printf( " Teste %d | n%d | n | n ", teste++, hanoi[nDiscs]);  

}|  

|  

70 |   return 0;  

}

```

2.15 Varredura de Retângulos

```
int val[200][200];

void decode(int &key, int &n, int* x, int* y)
{
    *x = key/n;
    *y = key%n;
}

int soma(int &x, int &y, int &d1, int& d2)
{
    int i, j;
    int res = 0;
    for(i = 0; i<=d1; i++)
    {
        for(j = 0; j<=d2; j++)
        {
            res+=val[x+i][y+j];
        }
    }
    return res;
}

int main(int argc, char** argv)
{
    int i, j, k, n, x, y;

    while(scanf("%d", &n) == 1)
    {
        int lim = n*n;
        for(i = 0; i<lim; i++)
        {
            decode(i, n, &x, &y);
            scanf("%d", &(val[x][y]));
        }

        int maxSum = -1;

        for(int lim1 = 0; lim1 < n; lim1++)
        {
            for(int lim2 = 0; lim2 < n; lim2++)
            {
                for(i = 0; i<n-lim1; i++)
                {
                    for(j = 0; j<n-lim2; j++)
                    {
                        int res = soma(i,j,lim1,lim2);
                        if(res > maxSum)
                        {
                            maxSum = res;
                        }
                    }
                }
            }
        }
    }
}
```

```

        printf( "%d | n", maxSum);
    }

60   return 0;
}

```

3 Backtracking

3.1 8-Rainhas

```

int row[9], TC, a, b, lineCounter;

bool place(int col, int tryrow)
{
5   for (int prev = 1; prev < col; prev++)
     if (row[prev] == tryrow || (abs(row[prev] - tryrow) == abs(prev
           - col)))
       return false;
     return true;
}

10 void backtrack(int col)
{
   for (int tryrow = 1; tryrow <= 8; tryrow++)
     if (place(col, tryrow))
15   {
      row[col] = tryrow;
      if (col == 8 && row[b] == a)
      {
        printf( "%2d %d", ++lineCounter, row[1]);
        for (int j = 2; j <= 8; j++) printf( " %d", row[j]);
        printf( " | n");
      }
      else
        backtrack(col + 1);
25   }
}

30 int main()
{
  scanf( "%d", &TC);
  while (TC--)
  {
    scanf( "%d %d", &a, &b);
    memset(row, 0, sizeof row); lineCounter = 0;
    printf( "SOLN COLUMN|n");
    printf( "# 1 2 3 4 5 6 7 8 |n|n");
    backtrack(1);
    if (TC)
      printf( " |n");
40  }
  return 0;
}

```

3.2 Clique-Máximo

```
int apt[26][26];
int clique[26];
int tamClique = 0;

5 int addOK(int ind)
{
    int i;

    for(i = 0; i < tamClique; i++)
        if(!(apt[clique[i]][ind] && apt[ind][clique[i]]))
            return 0;

    return 1;
}

15 void montaClique(int ind, int* maxClique, int n)
{
    int i;

    if(tamClique > *maxClique)
    {
        *maxClique = tamClique;
    }

    20 for(i = ind; i < n; i++)
    {
        if(addOK(i))
        {
            clique[tamClique++] = i;
            montaClique(i+1, maxClique, n);
            tamClique--;
        }
    }
}

25 int main()
{
    int n, i, j, v1, cont = 1, maxVal;

    30 while(scanf("%d", &n), n)
    {
        for(i = 0; i < n; i++)
        {
            for(j = 0; j < n; j++)
            {
                35 apt[i][j] = 1;
            }
            apt[i][i] = 0;

            40 while(scanf("%d", &v1), v1)
            {
                apt[i][v1-1] = 0;
            }
        }
    }

    45 maxVal = 0;
```

```

        montaClique(0, &maxVal, n);

60    printf( "Teste %d | n%d | n\n", cont++, maxVal);
}

return 0;
}

```

3.3 Select Three

```

int entrosamento[MAXELEM][MAXELEM];

void backtracking(int ind, int n, vector<int> &solution, vector<int>
                  > &melhor, int* foMelhor)
{
5   if(solution.size() == 3)
   {
      int tot = entrosamento[solution[0]][solution[1]] + entrosamento
                  [solution[0]][solution[2]] + entrosamento[solution[1]][
                  solution[2]];

      if(tot > *foMelhor)
10
         {
            *foMelhor = tot;
            melhor = solution;
         }
         return;
15
   }

   for(int i = ind; i<=n; i++)
   {
      solution.push_back(i);
20   backtracking(i+1, n, solution, melhor, foMelhor);
      solution.pop_back();
   }
}

25 int main()
{
   int n, m, v1, v2, w, i;

   scanf("%d%d", &n, &m);

30   memset(entrosamento, 0, (n+1)*(n+1)*sizeof(int));

   for(i = 0; i<m; i++)
   {
35     scanf("%d%d%d", &v1, &v2, &w);

      entrosamento[v1][v2] = entrosamento[v2][v1] = w;
   }

40   vector<int> solution;
   vector<int> melhor;
   int foMelhor = -INFINITO;

```

```

    backtracking(1, n, solution, melhor, &foMelhor);

45   printf("%d %d %d\n", melhor[0], melhor[1], melhor[2]);

    return 0;
}

```

3.4 TSP

Small instances.

```

int i, j, TC, xsize, ysize, n, x[11], y[11], dist[11][11], memo
[11][1 << 11];

int tsp(int pos, int bitmask)
{
5   if (bitmask == (1 << (n + 1)) - 1)
      return dist[pos][0];
   if (memo[pos][bitmask] != -1)
      return memo[pos][bitmask];

10  int ans = 2000000000;
   for (int nxt = 0; nxt <= n; nxt++)
       if (nxt != pos && !(bitmask & (1 << nxt)))
          ans = min(ans, dist[pos][nxt] + tsp(nxt, bitmask | (1 << nxt)));
   return memo[pos][bitmask] = ans;
15 }

int main()
{
20   scanf("%d", &TC);
   while (TC--)
   {
      scanf("%d %d", &xsize, &ysize);
      scanf("%d %d", &x[0], &y[0]);
      scanf("%d", &n);
25   for (i = 1; i <= n; i++)
      scanf("%d %d", &x[i], &y[i]);

      for (i = 0; i <= n; i++)
         for (j = 0; j <= n; j++)
30            dist[i][j] = abs(x[i] - x[j]) + abs(y[i] - y[j]);

      memset(memo, -1, sizeof memo);
      printf("The shortest path has length %d\n", tsp(0, 1));
   }
35   return 0;
}

```

4 Dividir para Conquistar

- Definir a parada de recursão, apesar de não influenciar diretamente na complexidade assintótica, altera o desempenho real (análise empírica de desempenho).

4.1 Busca Binária

```
int buscaBinaria(int key, int* vec, int limEsq, int limDir, int*
comp)
{
    if(limEsq == limDir)
    {
        if(vec[limEsq] == key)
            *comp = 0; /* O elemento procurado foi encontrado; */
        else if(vec[limEsq] < key)
            *comp = 1; /* O elemento procurado é maior que o encontrado;
                         */
        else
            *comp = -1; /* O elemento procurado é menor que o encontrado
                         ; */
    }
    return limEsq;
}

int mid = (limEsq + limDir)/2;

if(vec[mid] == key)
{
    *comp = 0;
    return mid;
}

if(vec[mid] < key)
    return buscaBinaria(key, vec, mid+1, limDir, comp);
25
return buscaBinaria(key, vec, limEsq, mid, comp);
}
```

4.2 Contador de Inversões

4.2.1 Versão Comum

```
void merge(int* vec, int vecSize, int* contador)
{
    int mid;
    int i, j, k;
5
    mid = vecSize / 2;

    i = 0;
    j = mid;
10
    k = 0;

    while(i < mid && j < vecSize)
```

```

15     {
16         if(vec[i] <= vec[j])
17         {
18             tmp[k] = vec[i];
19             ++i;
20         }
21         else
22         {
23             (*contador)+=mid-i;
24             tmp[k] = vec[j];
25             ++j;
26         }
27         ++k;
28     }
29     if (i == mid)
30     {
31         while (j < vecSize)
32         {
33             tmp[k] = vec[j];
34             ++j;
35             ++k;
36         }
37     }
38     else
39     {
40         while(i < mid)
41         {
42             tmp[k] = vec[i];
43             ++i;
44             ++k;
45         }
46     }
47     for (i = 0; i < vecSize; ++i)
48         vec[i] = tmp[i];
49     }
50
51 void mergeSort(int* vec, int vecSize, int* contador)
52 {
53     int mid;
54
55     if(vecSize > 1)
56     {
57         mid = vecSize / 2;
58         mergeSort(vec, mid, contador);
59         mergeSort(vec + mid, vecSize - mid, contador);
60         merge(vec, vecSize, contador);
61     }
62 }

```

4.2.2 Versão Alterada

```

#include <iostream>
#include <cstdlib>
#include <cstdio>
#include <vector>
5

```

```

using namespace std;

int main(){
    int n, entrada;
10
    while(scanf("%d", &n) && n){
        vector<int> jogo(n);
        for(int i=0; i<n; i++)
            scanf("%d", &jogo[i]);
15
        int trocas = 0;
        int i=0, aux;
        while(i<n){
            while(jogo[i] != i+1){
20
                aux = jogo[jogo[i]-1];
                jogo[jogo[i]-1] = jogo[i];
                jogo[i] = aux;
                trocas = 1-trocas;
            }
            i++;
        }

        if(trocas == 0) printf("Carlos\n");
        else printf("Marcelo\n");
30
    }

    return 0;
}

```

4.2.3 Versão usando Árvore

```

long int ans = 0;

struct Node
{
5    int n, right;
    Node * child[2];
    Node(int _n)
    {
        child[0] = child[1] = 0;
        n = _n, right = 1;
    }
};

void insert(Node *& node, int a)
15
{
    if(node)
    {
        if(a >= node->n)
            node->right++;
        else
            ans += node->right;
        if(a - node->n)
            insert(node->child[a - node->n > 0], a);
20
    }
    else
25

```

```

        node = new Node(a);
    }

int main()
{
    int N, a;

    Node *r = 0;
    while (scanf("%d", &N) == 1 && N && !(ans = 0) && !(r = NULL))
    {
        while (N-- && scanf("%d", &a) == 1)
            insert(r, a);

        printf("%ld\n", ans);
    }
    return 0;
}

```

4.3 Multiplicação de Matrizes

$$T(n) \Theta = O(n^{2.82})$$

$$(n^2) \oplus + \left(\frac{n}{2} \right)_T = \left(n \right)_T$$

$$\begin{aligned} P_4 &= D(G-E) \\ P_3 &= (C+D)(E+F) \\ P_2 &= (A+B)(H+G) \\ P_1 &= A(F-H) \\ P_5 &= (A+D)(E+H) \\ P_6 &= (B-D)(G+H) \\ P_7 &= (A-C)(E+F) \end{aligned}$$

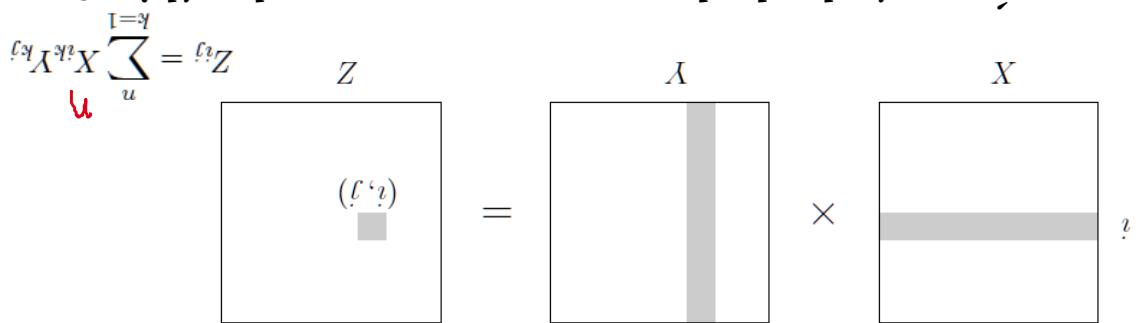
$$\text{Strassen (1969)} \quad XY = \begin{bmatrix} P_5 + P_4 - P_2 + P_6 & P_3 + P_4 \\ P_1 + P_2 & P_1 + P_3 - P_7 \end{bmatrix}$$

D&C: Multiplicação de Matrizes



$$\begin{array}{c} (n^2) \oplus + \left(\frac{n}{2} \right)_T = \left(n \right)_T \\ X = \begin{bmatrix} A & B \\ C & D \end{bmatrix}, Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix}, Z = \begin{bmatrix} CE + DG & CF + DH \\ AE + BG & AF + BH \end{bmatrix} \end{array}$$

- É possível calcular $Z = XY$ em tempo sub-cúbico?



Multiplicação de Matrizes



4.4 Exponenciação

```
x^n em O(log n) passos:  
x^n = n par ? x^(n/2) * x^(n/2) : x * x^((n-1)/2) * x^((n-1)/2);
```

5 Estruturas de Dados

5.1 Big Integer

5.1.1 C++

```
#define MAXDIGITS 100 /* comprimento máximo do bignum */  
#define PLUS 1 /* bit de sinal positivo */  
#define MINUS -1 /* bit de sinal negativo */  
  
5 typedef struct  
{  
    char digits[MAXDIGITS]; /* representa o número */  
    int signbit; /* 1 se positivo, -1 se negativo */  
    int lastdigit; /* índice do dígito mais significativo */  
} bignum;  
  
10 void print_bignum(bignum *n)  
{  
    int i;  
    if (n->signbit == MINUS) printf("- ");  
    for (i=n->lastdigit; i>=0; i--)  
        printf("%c", '0'+n->digits[i]);  
    printf("|\n");  
}  
  
15 void int_to_bignum(int s, bignum *n)  
{  
    int i; /* contador */  
    int t; /* int a ser trabalhado */  
  
    if (s >= 0) n->signbit = PLUS;  
    else n->signbit = MINUS;  
  
    20 for (i=0; i<MAXDIGITS; i++) n->digits[i] = (char) 0;  
    n->lastdigit = -1;  
    t = abs(s);  
  
    while (t > 0)  
    {  
        n->lastdigit++;  
        n->digits[n->lastdigit] = (t % 10);  
        t = t / 10;  
    }  
    30 if (s == 0) n->lastdigit = 0;  
}  
  
35 void initialize_bignum(bignum *n)
```

```

45  {
46      int_to_bignum(0, n);
47  }
48
49  int max(int a, int b)
50  {
51      return a>b?a:b;
52  }
53
54  /* c = a +/* b; */
55  void subtract_bignum(bignum *a, bignum *b, bignum *c)
56  {
57      int borrow;           /* algo a ser emprestado? */
58      int v;                /* dígito placeholder (apenas para
59                             preenchimento do comprimento) */
60      int i;                /* contador */
61
62      initialize_bignum(c);
63
64      if ((a->signbit == MINUS) || (b->signbit == MINUS))
65      {
66          b->signbit = -1 * b->signbit;
67          add_bignum(a,b,c);
68          b->signbit = -1 * b->signbit;
69          return;
70      }
71      if (compare_bignum(a,b) == PLUS)
72      {
73          subtract_bignum(b,a,c);
74          c->signbit = MINUS;
75          return;
76      }
77
78      c->lastdigit = max(a->lastdigit,b->lastdigit);
79      borrow = 0;
80
81      for (i=0; i<=(c->lastdigit); i++)
82      {
83          v = (a->digits[i] - borrow - b->digits[i]);
84          if (a->digits[i] > 0)
85              borrow = 0;
86          if (v < 0)
87          {
88              v = v + 10;
89              borrow = 1;
90          }
91          c->digits[i] = (char) v % 10;
92      }
93
94      zero_justify(c);
95  }
96
97  void add_bignum(bignum *a, bignum *b, bignum *c)
98  {
99      int carry;           /* dígito de transporte */
100     int i;                /* contador */

```

```

    initialize_bignum(c);

105   if (a->signbit == b->signbit) c->signbit = a->signbit;
   else
{
    if (a->signbit == MINUS)
    {
      a->signbit = PLUS;
110      subtract_bignum(b,a,c);
      a->signbit = MINUS;
    }
    else
    {
      b->signbit = PLUS;
115      subtract_bignum(a,b,c);
      b->signbit = MINUS;
    }
    return;
}
120

c->lastdigit = max(a->lastdigit,b->lastdigit)+1;
carry = 0;

125 for (i=0; i<=(c->lastdigit); i++)
{
  c->digits[i] = (char) (carry+a->digits[i]+b->digits[i]) % 10;
  carry = (carry + a->digits[i] + b->digits[i]) / 10;
}
130 zero_justify(c);
}

int compare_bignum(bignum *a, bignum *b)
135 {
  int i; /* contador */

  if ((a->signbit == MINUS) && (b->signbit == PLUS)) return(PLUS);
  if ((a->signbit == PLUS) && (b->signbit == MINUS)) return(MINUS);
140
  if (b->lastdigit > a->lastdigit) return (PLUS * a->signbit);
  if (a->lastdigit > b->lastdigit) return (MINUS * a->signbit);

  for (i = a->lastdigit; i>=0; i--)
145
  {
    if (a->digits[i] > b->digits[i]) return(MINUS * a->signbit);
    if (b->digits[i] > a->digits[i]) return(PLUS * a->signbit);
  }

  return(0);
}

void zero_justify(bignum *n)
{
  while ((n->lastdigit > 0) && (n->digits[ n->lastdigit ] == 0))
155  n->lastdigit --;

  if ((n->lastdigit == 0) && (n->digits[0] == 0))
    n->signbit = PLUS; /* hack para evitar -0 */
}

```

```

160    }
165
void digit_shift(bignum *n, int d)           /* multiplica n por  $10^d$ 
   */
{
    int i;                                /* contador */
170
    if ((n->lastdigit == 0) && (n->digits[0] == 0)) return;
175
    for (i=n->lastdigit; i>=0; i--)
        n->digits[i+d] = n->digits[i];
180
    for (i=0; i<d; i++) n->digits[i] = 0;
185
    n->lastdigit = n->lastdigit + d;
}
190
void multiply_bignum(bignum *a, bignum *b, bignum *c)
{
    bignum row;             /* representa a linha "shiftada" */
    bignum tmp;              /* bignum placeholder (apenas para preencher
                               o comprimento) */
    int i,j;                /* contadores */
195
    initialize_bignum(c);
200
    row = *a;
205
    for (i=0; i<=b->lastdigit; i++)
    {
        for (j=1; j<=b->digits[i]; j++)
        {
            add_bignum(c,&row,&tmp);
            *c = tmp;
        }
        digit_shift(&row,1);
    }
210
    c->signbit = a->signbit * b->signbit;
    zero_justify(c);
}
215
void divide_bignum(bignum *a, bignum *b, bignum *c)
{
    bignum row;             /* representa a linha "shiftada" */
    */
    bignum tmp;              /* bignum placeholder (apenas
                               para preencher o comprimento) */
    int asign, bsign;          /* sinais temporários */
    int i,j;                /* contadores */
220
    initialize_bignum(c);
225
    c->signbit = a->signbit * b->signbit;
230
    asign = a->signbit;
    bsign = b->signbit;

```

```

215     a->signbit = PLUS;
216     b->signbit = PLUS;

217     initialize_bignum(&row);
218     initialize_bignum(&tmp);

219     c->lastdigit = a->lastdigit;

220     for (i=a->lastdigit; i>=0; i--)
221     {
222         digit_shift(&row,1);
223         row.digits[0] = a->digits[i];
224         c->digits[i] = 0;
225         while (compare_bignum(&row,b) != PLUS)
226         {
227             c->digits[i]++;
228             subtract_bignum(&row,b,&tmp);
229             row = tmp;
230         }
231     }

235     zero_justify(c);

236     a->signbit = asign;
237     b->signbit = bsign;
238 }

239 int main()
240 {
241     int a,b;
242     bignum n1,n2,n3,zero;

243     while (scanf("%d %d\n",&a,&b) != EOF)
244     {
245         printf("a = %d      b = %d\n",a,b);
246         int_to_bignum(a,&n1);
247         int_to_bignum(b,&n2);

248         add_bignum(&n1,&n2,&n3);
249         printf("addition -- ");
250         print_bignum(&n3);

251         printf("compare_bignum a ? b = %d\n",compare_bignum(&n1, &n2));

252         subtract_bignum(&n1,&n2,&n3);
253         printf("subtraction -- ");
254         print_bignum(&n3);

255         multiply_bignum(&n1,&n2,&n3);
256         printf("multiplication -- ");
257         print_bignum(&n3);

258         int_to_bignum(0,&zero);
259         if (compare_bignum(&zero, &n2) == 0)
260             printf("division -- NaN\n");
261         else
262     {

```

```

        divide_bignum(&n1,&n2,&n3);
        printf( "division -- ");
        print_bignum(&n3);
275    }
    printf( "-----|n");
}

```

5.1.2 Java

Exemplo :

```

class Main
{
    public static void main(String[] args)
    {
5        BigInteger fac = BigInteger.ONE;
        for (int i = 2; i <= 25; i++)
            fac = fac.multiply(BigInteger.valueOf(i));
        System.out.println(fac);
    }
10}

```

Construtores e Métodos :

Constructor Summary

Constructor and Description

<code>BigInteger(byte[] val)</code>	Translates a byte array containing the two's-complement binary representation of a BigInteger into a BigInteger.
<code>BigInteger(int signum, byte[] magnitude)</code>	Translates the sign-magnitude representation of a BigInteger into a BigInteger.
<code>BigInteger(int bitLength, int certainty, Random rnd)</code>	Constructs a randomly generated positive BigInteger that is probably prime, with the specified bitLength.
<code>BigInteger(int numBits, Random rnd)</code>	Constructs a randomly generated BigInteger, uniformly distributed over the range 0 to ($2^{\text{numBits}} - 1$), inclusive.
<code>BigInteger(String val)</code>	Translates the decimal String representation of a BigInteger into a BigInteger.
<code>BigInteger(String val, int radix)</code>	Translates the String representation of a BigInteger in the specified radix into a BigInteger.

Method Summary

Modifier and Type	Method and Description
<code>BigInteger</code>	<code>abs()</code> Returns a BigInteger whose value is the absolute value of this BigInteger.
<code>BigInteger</code>	<code>add(BigInteger val)</code> Returns a BigInteger whose value is <code>(this + val)</code> .
<code>BigInteger</code>	<code>and(BigInteger val)</code> Returns a BigInteger whose value is <code>(this & val)</code> .
<code>BigInteger</code>	<code>andNot(BigInteger val)</code> Returns a BigInteger whose value is <code>(this & ~val)</code> . ●
<code>int</code>	<code>bitCount()</code> Returns the number of bits in the two's complement representation of this BigInteger that differ from its sign bit.
<code>int</code>	<code>bitLength()</code> Returns the number of bits in the minimal two's-complement representation of this BigInteger, <i>excluding</i> a sign bit.
<code>BigInteger</code>	<code>clearBit(int n)</code> Returns a BigInteger whose value is equivalent to this BigInteger with the designated bit cleared.
<code>int</code>	<code>compareTo(BigInteger val)</code> Compares this BigInteger with the specified BigInteger.
<code>BigInteger</code>	<code>divide(BigInteger val)</code> Returns a BigInteger whose value is <code>(this / val)</code> .
<code>BigInteger[]</code>	<code>divideAndRemainder(BigInteger val)</code> Returns an array of two BigIntegers containing <code>(this / val)</code> followed by <code>(this % val)</code> .
<code>double</code>	<code>doubleValue()</code> Converts this BigInteger to a double.
<code>boolean</code>	<code>equals(Object x)</code> Compares this BigInteger with the specified Object for equality.
<code>BigInteger</code>	<code>flipBit(int n)</code> Returns a BigInteger whose value is equivalent to this BigInteger with the designated bit flipped.
<code>float</code>	<code>floatValue()</code> Converts this BigInteger to a float.
<code>BigInteger</code>	<code>gcd(BigInteger val)</code> Returns a BigInteger whose value is the greatest common divisor of <code>abs(this)</code> and <code>abs(val)</code> .
<code>int</code>	<code>getLowestSetBit()</code> Returns the index of the rightmost (lowest-order) one bit in this BigInteger (the number of zero bits to the right of the rightmost one bit).
<code>int</code>	<code>hashCode()</code> Returns the hash code for this BigInteger.
<code>int</code>	<code>intValue()</code> Converts this BigInteger to an int.
<code>boolean</code>	<code>isProbablePrime(int certainty)</code> Returns true if this BigInteger is probably prime, false if it's definitely composite.

long	<code>longValue()</code> Converts this BigInteger to a long.
<code>BigInteger</code>	<code>max(BigInteger val)</code> Returns the maximum of this BigInteger and val.
<code>BigInteger</code>	<code>min(BigInteger val)</code> Returns the minimum of this BigInteger and val.
<code>BigInteger</code>	<code>mod(BigInteger m)</code> Returns a BigInteger whose value is (this mod m).
<code>BigInteger</code>	<code>modInverse(BigInteger m)</code> Returns a BigInteger whose value is ($this^{-1} \bmod m$).
<code>BigInteger</code>	<code>modPow(BigInteger exponent, BigInteger m)</code> Returns a BigInteger whose value is ($this^{\text{exponent}} \bmod m$).
<code>BigInteger</code>	<code>multiply(BigInteger val)</code> Returns a BigInteger whose value is (this * val).
<code>BigInteger</code>	<code>negate()</code> Returns a BigInteger whose value is (-this).
<code>BigInteger</code>	<code>nextProbablePrime()</code> Returns the first integer greater than this BigInteger that is probably prime.
<code>BigInteger</code>	<code>not()</code> Returns a BigInteger whose value is (~this).
<code>BigInteger</code>	<code>or(BigInteger val)</code> Returns a BigInteger whose value is (this val).
<code>BigInteger</code>	<code>pow(int exponent)</code> Returns a BigInteger whose value is ($this^{\text{exponent}}$).
static <code>BigInteger</code>	<code>probablePrime(int bitLength, Random rnd)</code> Returns a BigInteger that is probably prime, with the specified bitLength.
<code>BigInteger</code>	<code>remainder(BigInteger val)</code> Returns a BigInteger whose value is (this % val).
<code>BigInteger</code>	<code>setBit(int n)</code> Returns a BigInteger whose value is equivalent to this BigInteger with the designated bit set.
<code>BigInteger</code>	<code>shiftLeft(int n)</code> Returns a BigInteger whose value is (this << n).
<code>BigInteger</code>	<code>shiftRight(int n)</code> Returns a BigInteger whose value is (this >> n).
int	<code>signum()</code> Returns the signum function of this BigInteger.
<code>BigInteger</code>	<code>subtract(BigInteger val)</code> Returns a BigInteger whose value is (this - val).
boolean	<code>testBit(int n)</code> Returns true if and only if the designated bit is set.
byte[]	<code>toByteArray()</code> Returns a byte array containing the two's-complement representation of this BigInteger.
<code>String</code>	<code>toString()</code> Returns the decimal String representation of this BigInteger.
<code>String</code>	<code>toString(int radix)</code> Returns the String representation of this BigInteger in the given radix.
static <code>BigInteger</code>	<code>valueOf(long val)</code> Returns a BigInteger whose value is equal to that of the specified long.
<code>BigInteger</code>	<code>xor(BigInteger val)</code> Returns a BigInteger whose value is (this ^ val).

5.2 Collections

```
class ch2_02_Collections
{
    public static void main(String[] args)
    {
        5      Vector<Integer> v = new Vector<Integer>();
        v.add(10);
        v.add(5);
        v.add(2);
        v.add(7);
        v.add(1);
        10
        System.out.println("Before sorting :");
        System.out.println(v);
        15
        Collections.sort(v);
        System.out.println("After sorting :");
        System.out.println(v);
        20
        int pos = Collections.binarySearch(v, 7);
        System.out.println("Trying to search for 7 in v, found at index
                           = " + pos);
        pos = Collections.binarySearch(v, 8);
        System.out.println("Trying to search for 8 in v, found at index
                           = " + pos);
        25
        /*
         * binarySearch will returns :
         *
         * index of the search key, if it is contained in the list;
         * otherwise, -(insertion point) - 1).
         * The insertion point is defined as the point at which the key
         * would be inserted into the list:
         * the index of the first element greater than the key,
         * or list.size(), if all elements in the list are less than the
         * specified key.
         * Note that this guarantees that the return value will be >= 0 if
         * and only if the key is found.
         */
        30
    }
}
```

5.3 Declaração de Vetores e Matrizes

```
double vect[6] = { 1.3, 4.5, 2.7, 4.1, 0.0, 100.1 };
double vect[] = { 1.3, 4.5, 2.7, 4.1, 0.0, 100.1 };
int matrix[3][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
char str[10] = { 'J', 'o', 'a', 'o', '|0' };
5   char str[10] = "Joao";
char str_vect[3][10] = { "Joao", "Maria", "Jose" };
char mess[] = "Linguagem C: flexibilidade e poder.";
int matrix[][][2] = { 1, 2, 2, 4, 3, 6, 4, 8, 5, 10 };
```

5.4 Fenwick Tree

5.4.1 C++

```
typedef vector<int> vi;
#define LSOne(S) (S & (-S))

void ft_create(vi &ft, int n) { ft.assign(n + 1, 0); }

5    int ft_rsq(const vi &ft, int b)
{
    int sum = 0; for (; b; b -= LSOne(b)) sum += ft[b];
    return sum;
}

10   int ft_rsq(const vi &ft, int a, int b)
{
    return ft_rsq(ft, b) - (a == 1 ? 0 : ft_rsq(ft, a - 1));
}

15   void ft_adjust(vi &ft, int k, int v)
{
    for (; k < (int)ft.size(); k += LSOne(k)) ft[k] += v;
}

20   int main()
{
    vi ft;
    ft_create(ft, 10);
    ft_adjust(ft, 2, 1);
    ft_adjust(ft, 4, 1);
    ft_adjust(ft, 5, 2);
    ft_adjust(ft, 6, 3);
    ft_adjust(ft, 7, 2);
    ft_adjust(ft, 8, 1);
    ft_adjust(ft, 9, 1);
    printf("%d\n", ft_rsq(ft, 1, 1));
    printf("%d\n", ft_rsq(ft, 1, 2));
    printf("%d\n", ft_rsq(ft, 1, 6));
    printf("%d\n", ft_rsq(ft, 1, 10));
    printf("%d\n", ft_rsq(ft, 3, 6));

    ft_adjust(ft, 5, 2);
    printf("Index: ");
    for (int i = 0; i < (int)ft.size(); i++)
        printf("%d ", i);
    printf("\n");
    printf("FT : ");
    for (int i = 0; i < (int)ft.size(); i++)
        printf("%d ", ft[i]);
    printf("\n");

    return 0;
}
```

5.4.2 Java

```

import java.util.*;

class ch2_10_fenwicktree_ds
{
    private static int LSOne(int S) { return (S & (-S)); }

    private static Vector<Integer> ft_create(int n)
    {
        Vector<Integer> v = new Vector<Integer>();
        for (int i = 0; i <= n; i++) v.add(0);
        return v;
    }

    private static int ft_rsq(Vector<Integer> ft, int b)
    {
        int sum = 0; for (; b > 0; b -= LSOne(b)) sum += ft.get(b);
        return sum;
    }

    private static int ft_rsq(Vector<Integer> ft, int a, int b)
    {
        return ft_rsq(ft, b) - (a == 1 ? 0 : ft_rsq(ft, a - 1));
    }

    private static void ft_adjust(Vector<Integer> ft, int k, int v)
    {
        for (; k < (int)ft.size(); k += LSOne(k)) ft.set(k, ft.get(k) + v);
    }

    public static void main(String[] args)
    {
        Vector<Integer> ft;
        ft = ft_create(10);
        ft_adjust(ft, 2, 1);
        ft_adjust(ft, 4, 1);
        ft_adjust(ft, 5, 2);
        ft_adjust(ft, 6, 3);
        ft_adjust(ft, 7, 2);
        ft_adjust(ft, 8, 1);
        ft_adjust(ft, 9, 1);
        System.out.printf("%d\n", ft_rsq(ft, 1, 1));
        System.out.printf("%d\n", ft_rsq(ft, 1, 2));
        System.out.printf("%d\n", ft_rsq(ft, 1, 6));
        System.out.printf("%d\n", ft_rsq(ft, 1, 10));
        System.out.printf("%d\n", ft_rsq(ft, 3, 6));

        ft_adjust(ft, 5, 2);
        System.out.printf("Index: ");
        for (int i = 0; i < (int)ft.size(); i++)
            System.out.printf("%d ", i);
        System.out.printf("\n");
        System.out.printf("FT : ");
        for (int i = 0; i < (int)ft.size(); i++)
            System.out.printf("%d ", ft.get(i));
        System.out.printf("\n");
    }
}

```

}

5.5 Hashtable

```

***** Construtores *****

Hashtable();
Hashtable(int size);
Hashtable(int size, float fillRatio);
Hashtable(Map m);

***** The following example uses a Hashtable to store the names
      of bank depositors and their current balances *****

import java.util.*;

class HTDemo
{
    public static void main(String args[])
    {
        Hashtable balance = new Hashtable();
        Enumeration names;
        String str;
        double bal;

        balance.put("John Doe", new Double(3434.34));
        balance.put("Tom Smith", new Double(123.22));
        balance.put("Jane Baker", new Double(1378.00));
        balance.put("Todd Hall", new Double(99.22));
        balance.put("Ralph Smith", new Double(-19.08));

        names = balance.keys();
        while(names.hasMoreElements())
        {
            str = (String) names.nextElement();
            System.out.println(str + ": " +
                balance.get(str));
        }
        System.out.println();

        bal = ((Double)balance.get("John Doe")).doubleValue();
        balance.put("John Doe", new Double(bal+1000));
        System.out.println("John Doe's new balance: " +
            balance.get("John Doe"));
    }
}

***** OUTPUT *****/
/*
 * The output from this program is shown here:
 *
 * Ralph Smith: -19.08
 * Tom Smith: 123.22
 * John Doe: 3434.34
 * Todd Hall: 99.22
 * Jane Baker: 1378.0
 * John Doe's new balance: 4434.34
 */

```

```

/*
***** One important point: like the map classes, Hashtable does
not directly support iterators. Thus, the preceding program uses
an enumeration to display the contents of balance. However, you
can obtain set-views of the hash table, which permits the use
of iterators. To do so, you simply use one of the collection-
view methods defined by Map, such as entrySet( ) or keySet( ).
For example, you can obtain a set-view of the keys and iterate
through them. Here is a reworked version of the program that
shows this technique. *****/
55
import java.util.*;
60
class HTDemo2
{
    public static void main(String args[])
    {
        Hashtable balance = new Hashtable();
        String str;
        double bal;
65
        balance.put("John Doe", new Double(3434.34));
        balance.put("Tom Smith", new Double(123.22));
        balance.put("Jane Baker", new Double(1378.00));
        balance.put("Todd Hall", new Double(99.22));
        balance.put("Ralph Smith", new Double(-19.08));
70
        Set set = balance.keySet();
75
        Iterator itr = set.iterator();
        while(itr.hasNext())
        {
            str = (String) itr.next();
            System.out.println(str + ": " +
                balance.get(str));
80
            System.out.println();
85
            bal = ((Double)balance.get("John Doe")).doubleValue();
            balance.put("John Doe", new Double(bal+1000));
            System.out.println("John Doe's new balance: " +
                balance.get("John Doe"));
        }
    }
}

```

5.6 Heap / Fila de Prioridades

5.6.1 C++

```

int main()
{
    int money;
    char name[20];
5    priority_queue< pair<int, string> > pq;
    pair<int, string> result;
}

```

```

10      /*
100 john
10 billy
20 andy
100 steven
70 felix
2000 grace
70 martin
*/
15    for (int i = 0; i < 7; i++)
{
    scanf("%d %s", &money, &name);
    pq.push(make_pair(money, name));
}

20

result = pq.top();
pq.pop();
printf("%s has %d $|n", ((string)result.second).c_str(), result.
       first);
result = pq.top(); pq.pop();
printf("%s has %d $|n", ((string)result.second).c_str(), result.
       first);
result = pq.top(); pq.pop();
printf("%s has %d $|n", ((string)result.second).c_str(), result.
       first);

30   return 0;
}

```

5.6.2 Java

```

import java.util.*;

class pair < X, Y >
{
5    X _first;
    Y _second;

    public pair(X f, Y s) { _first = f; _second = s; }

10   X first() { return _first; }
    Y second() { return _second; }

    void setFirst(X f) { _first = f; }
    void setSecond(Y s) { _second = s; }
15 }

class ch2_06_PriorityQueue
{
20   public static void main(String[] args)
   {
        PriorityQueue < pair < Integer, String > > pq = new
        PriorityQueue < pair < Integer, String > >
        (1,
         new Comparator< pair < Integer, String > >()
         {

```

```

25     public int compare(pair < Integer, String > i, pair <
        Integer, String > j)
    {
        return j.first() - i.first();
    }
30 }

/*
100 john
10 billy
20 andy
100 steven
70 felix
2000 grace
70 martin
*/
Scanner sc = new Scanner(System.in);
for (int i = 0; i < 7; i++)
{
    int money = sc.nextInt();
    String name = sc.nextLine();
    pq.offer(new pair < Integer, String > (money, name));
}

50 pair<Integer, String> result = pq.poll();
System.out.println(result.second() + " has " + result.first() +
    "$");
result = pq.poll();
System.out.println(result.second() + " has " + result.first() +
    "$");
result = pq.poll();
System.out.println(result.second() + " has " + result.first() +
    "$");
}
}

```

5.7 Map and Set

5.7.1 C++

```

int main()
{
    char name[20];
    int value;
5    set<int> used_values;
    map<string, int> mapper;

    /*
john 78
10 billy 69
andy 80
steven 77
felix 82
grace 75
martin 81
15

```

```

*/  

for (int i = 0; i < 7; i++)  

{  

    scanf ("%s %d", &name, &value);  

    mapper[name] = value;  

    used_values.insert(value);
}  

25   for (map<string, int>::iterator it = mapper.begin(); it != mapper  

        .end(); it++)  

        printf ("%s %d\n", ((string)it->first).c_str(), it->second);  

        printf ("steven's score is %d, grace's score is %d\n",
mapper["steven"], mapper["grace"]);  

30   printf ("=====|\n");  

        printf ("%d\n", *used_values.find(77));  

        if (used_values.find(79) == used_values.end())
            printf ("79 not found\n");
  

        return 0;
}

```

5.7.2 Java

```

import java.util.*;  

class ch2_05_TreeMap_TreeSet
{
5   public static void main(String[] args)
{
    TreeSet<Integer> used_values = new TreeSet<Integer>();
    used_values.clear();
    TreeMap<String, Integer> mapper = new TreeMap<String, Integer
        >();
10   mapper.clear();

    /*  

john 78  

billy 69  

andy 80  

steven 77  

felix 82  

grace 75  

martin 81
20   */
    Scanner sc = new Scanner(System.in);
    for (int i = 0; i < 7; i++)
    {
        String name = sc.next();
        int value = sc.nextInt();
        sc.nextLine();
        mapper.put(name, value);
        used_values.add(value);
25   }
}

```

30

```

        System.out.println(mapper.keySet());
        System.out.println(mapper.values());

35      System.out.println("steven's score is " + mapper.get("steven")
+ ", grace's score is " + mapper.get("grace"));
        System.out.println("=====");
        System.out.println(used_values.contains(77));
40      System.out.println(used_values.headSet(77));
        System.out.println(used_values.tailSet(77));
        if (!used_values.contains(79))
            System.out.println("79 not found");
    }
}

```

5.8 memset

```

bool can_reach[210][25];
memset(can_reach, false, sizeof can_reach);

```

5.9 Queue and Stack

5.9.1 C++

```

int main()
{
    stack<char> s;
    queue<char> q;
5     printf("%d\n", s.empty());
     printf("=====|n");
     s.push('a');
     s.push('b');
10    s.push('c');
     printf("%c\n", s.top());
     s.pop();
     printf("%c\n", s.top());
     printf("%d\n", s.empty());
15    printf("=====|n");

     printf("%d\n", q.empty());
     printf("=====|n");
     while (!s.empty())
20    {
        q.push(s.top());
        s.pop();
    }
    q.push('z');
25    printf("%c\n", q.front());
    printf("%c\n", q.back());

     printf("=====|n");
}

```

```

30     while (!q.empty())
31     {
32         printf("%c\n", q.front());
33         q.pop();
34     }
35
36     return 0;
37 }
```

5.9.2 Java

```

import java.util.*;

class ch2_03_Stack_Queue
{
    5   public static void main(String[] args)
    {
        Stack<Character> s = new Stack<Character>();

        Queue<Character> q = new LinkedList<Character>();
10
        System.out.println(s.isEmpty());
        System.out.println("=====");
        s.push('a');
        s.push('b');
        s.push('c');
15
        System.out.println(s.peek());
        s.pop();
        System.out.println(s.peek());
        System.out.println(s.isEmpty());
        System.out.println("=====");

20
        System.out.println(q.isEmpty());
        System.out.println("=====");
        while (!s.isEmpty())
        {
25            q.offer(s.peek());
            s.pop();
        }

30        System.out.println(q.peek());
    }
}
```

5.10 Segment Tree

5.10.1 C++

```

typedef vector<int> vi;

void st_build(vi &st, const vi &A, int vertex, int L, int R)
{
5   if (L == R)
      st[vertex] = L;
    else
```

```

    {
        int nL = 2 * vertex, nR = 2 * vertex + 1;
        st_build(st, A, nL, L , (L + R) / 2);
        st_build(st, A, nR, (L + R) / 2 + 1, R );
        int lContent = st[nL] , rContent = st[nR];
        int lValue = A[lContent], rValue = A[rContent];
        st[vertex] = (lValue <= rValue) ? lContent : rContent;
    }
}

void st_create(vi &st, const vi &A)
{
    int len = (int)(2*pow(2.0, floor((log((double)A.size()) / log(2.0))
+ 1)));
    st.assign(len, 0);
    st_build(st, A, 1, 0, (int)A.size() - 1);
}

int st_rmq(vi &st, const vi &A, int vertex, int L, int R, int i,
           int j)
{
    if (i > R || j < L) return -1;
    if (L >= i && R <= j) return st[vertex];

    int p1 = st_rmq(st, A, 2 * vertex , L , (L+R) /
2, i, j);
    int p2 = st_rmq(st, A, 2 * vertex + 1, (L+R) / 2 + 1, R ,
i, j);

    if (p1 == -1) return p2;
    if (p2 == -1) return p1;
    return (A[p1] <= A[p2]) ? p1 : p2;
}

int st_rmq(vi &st, const vi& A, int i, int j)
{
    return st_rmq(st, A, 1, 0, (int)A.size() - 1, i, j);
}

int st_update_point(vi &st, vi &A, int node, int b, int e, int idx,
                     int new_value)
{
    int i = idx, j = idx;

    if (i > e || j < b)
        return st[node];

    if (b == i && e == j)
    {
        A[i] = new_value;
        return st[node] = b;
    }

    int p1, p2;
    p1 = st_update_point(st, A, 2 * node , b , (b + e
) / 2, idx, new_value);
    p2 = st_update_point(st, A, 2 * node + 1, (b + e) / 2 + 1, e
, idx, new_value);
}

```

```

60     return st[node] = (A[p1] <= A[p2]) ? p1 : p2;
}

int st_update_point(vi &st, vi &A, int idx, int new_value)
{
    return st_update_point(st, A, 1, 0, (int)A.size() - 1, idx,
                           new_value);
}

int main()
{
    int arr[7] = { 8, 7, 3, 9, 5, 1, 10 };
    vi A(arr, arr + 7);
    vi st; st_create(st, A);

    printf("idx      0, 1, 2, 3, 4, 5 , 6\n");
    printf("A is { 8, 7, 3, 9, 5, 1 , 10 }\n");
    printf("RMQ(1, 3) = %d\n", st_rmq(st, A, 1, 3));
    printf("RMQ(4, 6) = %d\n", st_rmq(st, A, 4, 6));
    printf("RMQ(3, 4) = %d\n", st_rmq(st, A, 3, 4));
    printf("RMQ(0, 0) = %d\n", st_rmq(st, A, 0, 0));
    printf("RMQ(0, 1) = %d\n", st_rmq(st, A, 0, 1));
    printf("RMQ(0, 6) = %d\n", st_rmq(st, A, 0, 6));

    printf("idx      0, 1, 2, 3, 4, 5 , 6\n");
    printf("Now, modify A into { 8, 7, 3, 9, 5, 100, 10 }\n");
    st_update_point(st, A, 5, 100);
    printf("These values do not change\n");
    printf("RMQ(1, 3) = %d\n", st_rmq(st, A, 1, 3));
    printf("RMQ(3, 4) = %d\n", st_rmq(st, A, 3, 4));
    printf("RMQ(0, 0) = %d\n", st_rmq(st, A, 0, 0));
    printf("RMQ(0, 1) = %d\n", st_rmq(st, A, 0, 1));
    printf("These values change\n");
    printf("RMQ(0, 6) = %d\n", st_rmq(st, A, 0, 6));
    printf("RMQ(4, 6) = %d\n", st_rmq(st, A, 4, 6));
    printf("RMQ(4, 5) = %d\n", st_rmq(st, A, 4, 5));

    return 0;
}

```

5.10.2 Java

```

import java.util.*;

class ch2_09_segmenttree_ds
{
    private static int[] st;

    public static void st_build(int[] A, int vertex, int L, int R)
    {
        if (L == R)
            st[vertex] = L;
        else
        {
            int nL = 2 * vertex, nR = 2 * vertex + 1;
            st_build(A, nL, L, (L + R) / 2);
            st_build(A, nR, L + 1, (L + R) / 2);
        }
    }
}

```

```

15     st_build(A, nR, (L + R) / 2 + 1, R );
16     int lContent = st[nL] , rContent = st[nR];
17     int lValue = A[lContent], rValue = A[rContent];
18     st[vertex] = (lValue <= rValue) ? lContent : rContent;
19 }
20 }

public static void st_create(int[] A)
{
    int len = (int)(2 * Math.pow(2.0, Math.floor((Math.log((double)
A.length) / Math.log(2.0)) + 1)));
25    st = new int[len];
    for (int i = 0; i < len; i++) st[i] = 0;
    st_build(A, 1, 0, (int)A.length - 1);
}

30 public static int st_rmq(int[] A, int vertex, int L, int R, int i
, int j)
{
    if (i > R || j < L) return -1;
    if (L >= i && R <= j) return st[vertex];

    int p1 = st_rmq(A, 2 * vertex , L , (L + R) /
2, i, j);
    int p2 = st_rmq(A, 2 * vertex + 1, (L + R) / 2 + 1, R
, i, j);

    if (p1 == -1) return p2;
    if (p2 == -1) return p1;
40    return (A[p1] <= A[p2]) ? p1 : p2;
}

50 public static int st_rmq(int[] A, int i, int j)
{
    return st_rmq(A, 1, 0, (int)A.length - 1, i, j);
}

public static int st_update_point(int[] A, int node, int b, int e
, int idx, int new_value)
{
    int i = idx, j = idx;

    if (i > e || j < b)
55    return st[node];

    if (b == i && e == j)
    {
        A[i] = new_value;
        return st[node] = b;
    }

60    int p1, p2;
    p1 = st_update_point(A, 2 * node , b , (b + e) /
2, idx, new_value);
    p2 = st_update_point(A, 2 * node + 1, (b + e) / 2 + 1, e
, idx, new_value);

    return st[node] = (A[p1] <= A[p2]) ? p1 : p2;
}

```

```

    }

    public static int st_update_point(int[] A, int idx, int new_value
        )
    {
        return st_update_point(A, 1, 0, (int)A.length - 1, idx,
            new_value);
    }

    public static void main(String[] args)
    {
        int[] A = new int[] {8, 7, 3, 9, 5, 1, 10};
        st_create(A);

        System.out.printf("           idx      0, 1, 2, 3, 4, 5 , 6|n"
            );
        System.out.printf("           A is { 8, 7, 3, 9, 5, 1 , 10
            }|n");
        System.out.printf("RMQ(1, 3) = %d|n", st_rmq(A, 1, 3));
        System.out.printf("RMQ(4, 6) = %d|n", st_rmq(A, 4, 6));
        System.out.printf("RMQ(3, 4) = %d|n", st_rmq(A, 3, 4));
        System.out.printf("RMQ(0, 0) = %d|n", st_rmq(A, 0, 0));
        System.out.printf("RMQ(0, 1) = %d|n", st_rmq(A, 0, 1));
        System.out.printf("RMQ(0, 6) = %d|n", st_rmq(A, 0, 6));

        System.out.printf("           idx      0, 1, 2, 3, 4, 5 , 6|n"
            );
        System.out.printf("Now, modify A into { 8, 7, 3, 9, 5, 100, 10
            }|n");
        st_update_point(A, 5, 100);
        System.out.printf("These values do not change|n");
        System.out.printf("RMQ(1, 3) = %d|n", st_rmq(A, 1, 3));
        System.out.printf("RMQ(3, 4) = %d|n", st_rmq(A, 3, 4));
        System.out.printf("RMQ(0, 0) = %d|n", st_rmq(A, 0, 0));
        System.out.printf("RMQ(0, 1) = %d|n", st_rmq(A, 0, 1));
        System.out.printf("These values change|n");
        System.out.printf("RMQ(0, 6) = %d|n", st_rmq(A, 0, 6));
        System.out.printf("RMQ(4, 6) = %d|n", st_rmq(A, 4, 6));
        System.out.printf("RMQ(4, 5) = %d|n", st_rmq(A, 4, 5));
    }
}

```

5.11 Set Insertion

```

pair<set<int>::iterator, bool> ret;
ret = att.insert(valObj);
if(ret.second == true)
    ret.first++;
5 printf("%d %d|n", *(ret.first), vec[*((ret.first))]);

```

5.12 Static Array Vs. Vector

5.12.1 C++

```

int main()
{
    int arr[5] = {7,7,7};
    vector<int> v(5, 5);
5
    printf("arr[2] = %d and v[2] = %d\n", arr[2], v[2]);

    for (int i = 0; i < 5; i++)
    {
10        arr[i] = i;
        v[i] = i;
    }

    printf("arr[2] = %d and v[2] = %d\n", arr[2], v[2]);
15

    v.push_back(5);
    printf("v[5] = %d\n", v[5]);

20    return 0;
}

```

5.12.2 Java

```

class ch2_01_staticarray_Vector
{
    public static void main(String[] args)
    {
5        int[] arr = new int[] {7,7,7,0,0};
        Vector<Integer> v = new Vector<Integer>(Collections.nCopies(5,
            5));

        System.out.println("arr[2] = " + arr[2] + " and v[2] = " + v.
            get(2));

10       for (int i = 0; i <= 4; i++)
        {
            arr[i] = i;
            v.set(i, i);
        }

15       System.out.println("arr[2] = " + arr[2] + " and v[2] = " + v.
            get(2));



20       v.add(5);
        System.out.println("v[5] = " + v.get(5));
    }
}

```

5.13 Trie Tree

5.13.1 C++

```

#define foreach(x, v) for (typeof (v).begin() x=(v).begin(); x !=(v).end(); ++x)
#define For(i, a, b) for (int i=(a); i<(b); ++i)
#define D(x) cout << #x " is " << (x) << endl

5 const double EPS = 1e-9;

int cmp(double x, double y = 0, double tol = EPS)
{
    return (x <= y + tol) ? (x + tol < y) ? -1 : 0 : 1;
10}

const int MAXS = 100005;

15 struct Trie
{
    int g[MAXS][26];
    int stateCount;

20 Trie()
{
    clear();
}

25 void clear()
{
    memset(g[0], -1, sizeof g[0]);
    stateCount = 1;
}

30 void add(char * s)
{
    int state = 0;
    for (*s; s++)
    {
        35    int next = (*s - 'a');
        if (g[state][next] == -1)
        {
            g[state][next] = stateCount;
            memset(g[stateCount], -1, sizeof g[stateCount]);
            stateCount++;
        }
        state = g[state][next];
    }
45}
};

Trie prefixTrie, suffixTrie;
int start[26];

50 void suffixDfs(int state, int depth)
{
    for (int e = 0; e < 26; ++e)
    {
        55    if (suffixTrie.g[state][e] == -1) continue;
        if (depth >= 1)

```

```

        {
            start[e]++;
        }
    suffixDfs(suffixTrie.g[state][e], depth + 1);
}

long long prefixDfs(int state, int depth)
{
    long long ans = 0;
    if (depth >= 1) ans += suffixTrie.stateCount - 1;
    for (int e = 0; e < 26; ++e)
    {
        if (prefixTrie.g[state][e] == -1) continue;
        if (depth >= 1)
        {
            ans -= start[e];
        }
        ans += prefixDfs(prefixTrie.g[state][e], depth + 1);
    }
    return ans;
}

int main()
{
    int P, S;
    while (scanf("%d %d ", &P, &S) == 2)
    {
        if (P == 0 and S == 0) break;

        prefixTrie.clear();
        suffixTrie.clear();
        for (int i = 0; i < 26; ++i) start[i] = 0;

        static char buffer[1024];
        for (int i = 0; i < P; ++i)
        {
            gets(buffer);
            prefixTrie.add(buffer);
        }
        for (int i = 0; i < S; ++i)
        {
            gets(buffer);
            int n = strlen(buffer);
            reverse(buffer, buffer + n);
            suffixTrie.add(buffer);
        }
        suffixDfs(0, 0);
        long long ans = prefixDfs(0, 0);

        printf("%lld |n", ans);
    }

    return 0;
}

```

5.13.2 Java

```
/** *****trie.java
*****
The words in the dictionary are stored in a data type trie. In
this type of tree,
each node can have a maximum of 26 children, one for each letter
of the alphabet.
Each node is structured like this.
5    class node
{
    char letter;
    boolean endsWord;
    node down;
    node right;
}
10
Each node stores a char letter, a boolean variable (indicates if
the letter stored
in the node ends a word or not), a pointer to the child of the
node, and a pointer
15 to the right sibling of the node. Each node can have a maximum
of 26 children, one
for each letter of the alphabet. All children are arranged in
alphabetical order
for easy searching. This trie stores each word in the forward
rather than
backward because the searching must be done from the start of the
word.
20
This document is responsible for reading the file whose name is
sent from
wordpop.java. Each word in order is inserted into the trie one
at a time. This
class is also responsible for searching the trie to see if a word
(or phrase) is
in the trie. Finally, this class keeps track of the amount of
each letter stored
in the trie which is saved in amountList[].
25 /**
 *****
import java.io.*;
import java.lang.*;

30 public class trie
{
    public class node
    {
        char letter;
        boolean endsWord;
        node down;
        node right;

        public node(char c)
        {
            letter = c;
            endsWord = false;
        }
    }
}
```

```

        down = null;
        right = null;
    }
    public node(char c, node r)
    {
        letter = c;
        endsWord = false;
        down = null;
        right = r;
    }
}

node root;
node position;
public int[] amountList = new int[26];

public trie()
{
    root = position = null;
}

/*
receives: a String str
returns: a new string able to be stored in the tree
gets rid of all spaces, punctuation, and numbers in str,
converts it to lower case, and stores only words longer
than three
*/
public String fixString(String str)
{
    int index = 0;

    str=str.toLowerCase();

    char[] myChars = str.toCharArray();
    char[] newChars = new char[str.length()];

    for( int x=0 ; x<myChars.length ; x++ )
    {
        if( myChars[x]>='a' && myChars[x]<='z' )
        {
            newChars[index++]=myChars[x];
        }
    }

    return (new String(newChars));
}

/*
receives: a String str
returns: nothing - inserts
insert the String str into the tree starting at the end of the
word.
each letter is inserted , and based on the other elements in the
list
a new child is added to the list , or an existing child is used.
*/

```

```

100    public void insert(String str)
101    {
102        str = fixString(str);
103
104        if(str==null) return;
105
106        int index=0;
107
108        if(root==null)
109        {
110            root = new node(' ');
111        }
112
113        node temp,tempBack,pred;
114        temp = tempBack = pred = root;
115
116        while(index<str.length())
117        {
118            if(temp.down == null)
119            {
120                temp.down = new node(str.charAt(index));
121                temp.down.right = new node(' ');
122                temp.down.right.down = temp;
123                temp = temp.down;
124            }
125            else
126            {
127                pred = temp;
128                temp = temp.down;
129                if(str.charAt(index) < temp.letter)
130                {
131                    temp = new node(str.charAt(index),temp);
132                    pred.down = temp;
133                }
134                else if(str.charAt(index) > temp.letter)
135                {
136                    while(temp.letter != ' ' && str.charAt(index) > temp.letter)
137                    {
138                        tempBack = temp;
139                        temp = temp.right;
140                    }
141                    if(str.charAt(index) != temp.letter)
142                    {
143                        temp = new node(str.charAt(index), temp);
144                        tempBack.right = temp;
145                    }
146                }
147            }
148            if(str.charAt(index)>=97) amountList[str.charAt(index)-97]++;
149            index++;
150        }
151        temp.endsWord = true;
152    }
153
154    /*
155     * receives: a String str (the name of a file)
156     * returns: nothing - builds the tree

```

```

function builds a tree by reading the file named fileName one
string at a time, and calls insert() to add that word to the tree
.

*/
160 public void setupTrie(String fileName)
{
    String word = new String();

    try
    {
        165 FileReader read = new FileReader(fileName);
        BufferedReader in = new BufferedReader(read,50);

        while(in.ready())
        {
            170 word=in.readLine();
            insert(word);
        }
    }
    catch (Exception e)
    {
        175 e.printStackTrace();
        return;
    }
}

180 public void print()
{
    node t=root;
    System.out.println(root.letter + " *");
    print(t.down);
}

185 void print(node t)
{
    190 if(t==null) return;
    node temp=t;
    while(t!=null)
    {
        System.out.print(t.letter + " ");
        195 if(t.endsWord)System.out.print("^\n");
        t=t.right;
    }
    System.out.println("*");
    t=temp;
    while(t!=null)
    {
        200 if(t.down!=null && t.right!=null)
        {
            System.out.println(t.letter + " children");
            print(t.down);
        }
        t=t.right;
    }
}

210 public void resetPosition()
{
}

```

```

        position = root;
    }

215
/*
receives: a String str
returns: true if str is found, otherwise false
follows the tree until the start until the prefix is found.
it doesn't matter if str is a full word or not.
*/
public boolean stringInTrie(String str)
{
    node t = root;

225
    for(int index=0; index<str.length(); index++)
    {
        if(t.down != null)
        {
            t = t.down;
        }
        else
        {
            return false;
        }

        while(str.charAt(index) != t.letter)
        {
            if(t.right==null)
            {
                return false;
            }
            t = t.right;
        };
    }
    position = t;
    return true;
}

250
/*
receives: a String str
returns: true if str is found, otherwise false
calls stringInTrie to see if the string is there, and returns
true
if the last node in the string ends a word
*/
public boolean wordInTrie(String str)
{
    if(stringInTrie(str) && position.endsWord)
    {
        return true;
    }
    else
    {
        return false;
    }
}

```

6 Grafos

6.1 Bellman-Ford

Encontra o menor caminho, inclusive se houver arestas negativas.

```
typedef pair<int, int> ii;
typedef vector<int> vi;
typedef vector<ii> vii;
#define INF 1000000000

5   int main()
{
    int V, E, s, u, v, w;
    vector<vii> AdjList;

10   /*
5 5 0
0 1 1
0 2 10
1 3 2
2 3 -10
3 4 3

3 3 0
0 1 1000
1 2 15
2 1 -42
*/
25   scanf("%d %d %d", &V, &E, &s);

    AdjList.assign(V, vii());
    for (int i = 0; i < E; i++)
    {
30       scanf("%d %d %d", &u, &v, &w);
        AdjList[u].push_back(ii(v, w));
    }

    vi dist(V, INF); dist[s] = 0;
35    for (int i = 0; i < V - 1; i++)
        for (int u = 0; u < V; u++)
            for (int j = 0; j < (int)AdjList[u].size(); j++)
            {
40               ii v = AdjList[u][j];
               dist[v.first] = min(dist[v.first], dist[u] + v.second);
            }

        bool hasNegativeCycle = false;
45        for (int u = 0; u < V; u++)
            for (int j = 0; j < (int)AdjList[u].size(); j++)
            {
50               ii v = AdjList[u][j];
               if (dist[v.first] > dist[u] + v.second)
                  hasNegativeCycle = true;
            }
55        printf("Negative Cycle Exist? %s\n", hasNegativeCycle ? "Yes" : "No");
    }
```

```

55     if (!hasNegativeCycle)
        for (int i = 0; i < V; i++)
            printf("SSSP(%d, %d) = %d\n", s, i, dist[i]);
    }
}

```

6.2 Detecção de Árvores e Ciclos

A Detecção de Árvores é feita usando busca em profundidade \Rightarrow o grafo é uma árvore se, e somente se, não existirem arestas de retorno.

A detecção de ciclo é feita quando a primeira aresta de retorno é identificada.

Tempo: $O(n)$

6.3 Detecção de Ciclo Euleriano

Para verificar se um grafo não direcionado possui um *ciclo Euleriano*: todos os vértices devem ter grau par.

Para verificar se um grafo não direcionado possui um *caminho Euleriano*: somente dois vértices podem ter grau ímpar.

6.4 Dijkstra

Encontra o menor caminho, porém não podem haver arestas negativas.

6.4.1 C

```

#include <cstdlib>
#include <cstdio>
#include <cstring>
#include <string>
5 #include <fstream>
#include <map>
#include <vector>
#include <queue>

10 #define MAXSTRSZ    100000
#define MAXNUMBER   200000
#define INF 999999999
#define EPS          1e-9
#define MAXV         106
15 #define MAXIMO 20600

using namespace std;

20

typedef pair<int,int> ElementType;

```

```

struct HeapStruct;
typedef struct HeapStruct *PriorityQueue;

25
PriorityQueue Initialize(int MaxElements);
void Insert(ElementType X, PriorityQueue H);
ElementType DeleteMin(PriorityQueue H);
ElementType FindMin(PriorityQueue H);
30
int IsEmpty(PriorityQueue H);
int IsFull(PriorityQueue H);
void display(PriorityQueue H);

#define MinPQSize (10)
35
#define MinData (-32767)

struct HeapStruct
{
    int Capacity;
    int Size;
    ElementType *Elements;
};

PriorityQueue Initialize(int MaxElements)
40
{
    PriorityQueue H;

    H = (struct HeapStruct *) malloc(sizeof (struct HeapStruct));

    H->Elements = (pair<int, int> *) malloc((MaxElements + 1) * sizeof
        ( ElementType));
50
    H->Capacity = MaxElements;
    H->Size = 0;
    H->Elements[ 0 ] = make_pair(MinData, 0);

55
    return H;
}

/* END */

60

/* H->Element[ 0 ] is a sentinel */

65
void Insert(ElementType X, PriorityQueue H)
{
    int i;

    for ( i = ++H->Size; (H->Elements[ i>>1 ]).first > X.first; i>>=1)
70
        H->Elements[ i ] = H->Elements[ i>>1 ];
    H->Elements[ i ] = X;
}

/* END */

75

ElementType DeleteMin(PriorityQueue H)
{
    int i, Child;

```

```

80     ElementType MinElement, LastElement;
81
82     MinElement = H->Elements[ 1 ];
83     LastElement = H->Elements[ H->Size-- ];
84
85     for (i = 1; i << 1 <= H->Size; i = Child)
86     {
87         Child = i << 1;
88         if (Child != H->Size && (H->Elements[ Child + 1 ]).first < (H->
89             Elements[ Child ]).first)
90             Child++;
91
92         if (LastElement.first > (H->Elements[ Child ]).first)
93             H->Elements[ i ] = H->Elements[ Child ];
94         else
95             break;
96     }
97     H->Elements[ i ] = LastElement;
98     return MinElement;
99 }
100
101 ElementType FindMin(PriorityQueue H)
102 {
103     if (!IsEmpty(H))
104         return H->Elements[ 1 ];
105     return H->Elements[ 0 ];
106 }
107
108 int IsEmpty(PriorityQueue H)
109 {
110     return H->Size == 0;
111 }
112
113 int IsFull(PriorityQueue H)
114 {
115     return H->Size == H->Capacity;
116 }
117
118 typedef pair<int, int> ii;
119 typedef vector<int> vi;
120 typedef vector<ii> vii;
121
122 int cases, j, V, E, s, u, v, w;
123 int i, d;
124 int tam[10006];
125 PriorityQueue pq;
126 vector<vii> AdjList (10006, vii());
127 int dist[10006];
128 ii front;
129
130 int dijkstra(int s, int de)
131 {
132     for(i = 1; i<=V; i++)
133         dist[i] = INF;
134
135     dist[s] = 0;
136     Insert(make_pair(0, s), pq);

```

```

    while(pq->Size)
    {
        front = DeleteMin(pq);
140      u = front.second;
        if (front.first == dist[u])
            for (j = 0; j != tam[u]; j++)
            {
                if(dist[u] + AdjList[u][j].second < dist[AdjList[u][j].
                    first])
                {
                    dist[AdjList[u][j].first] = dist[u] + AdjList[u][j].
                        second;
                    Insert(make_pair(dist[AdjList[u][j].first], AdjList[u][j].
                        first), pq);
                }
            }
150    }

    return dist[de];
}

155 void fastint(register int *n)
{
    register char c;

    *n = 0;
160    while (!isdigit(c = getc(stdin)));

    do
    {
        (*n) = (*n)*10 + (c - '0');
165    } while (isdigit( c = getc(stdin) ));

}

void faststr(register char *s)
{
    register char c;
    register int i = 0;

    while(c = getc(stdin) , c == ' ' || c == '|n');

    do
    {
        s[i++] = c;
    } while(c = getc(stdin) , c != ' ' && c != '|n');
    s[i] = '|0';
180}

/*
 *
 */
185 int main()
{
    char read[MAXSTRSZ+1];
    char s[MAXSTRSZ+1];
    char d[MAXSTRSZ+1];
190    int i, av;
}

```

```

    pq = Initialize(1000000);

195   fastint(&cases);

    while(cases--)
{
    fastint(&V);

200   map<string, int> getId;

    int curr = 1;

205   for(i = 1; i <= V; i++)
{
    int m, dest, w;

210     scanf("%s", read);
    fastint(&(tam[i]));

    getId[read] = i;

    for(j = 0; j < tam[i]; j++)
{
        fastint(&dest); fastint(&w);

215       if(j < AdjList[i].size())
{
            AdjList[i][j] = make_pair(dest, w);
        }
        else
{
            AdjList[i].push_back(make_pair(dest, w));
        }
    }

220   fastint(&av);

230   for(i = 0; i < av; i++)
{
    scanf("%s%s", s, d);

235   printf("%d | n", dijkstra(getId[s], getId[d]));
}
}

240   return 0;
}

```

6.4.2 C++

```

typedef pair<int, int> ii;
typedef vector<int> vi;
typedef vector<ii> vii;

5 int main()

```

```

{
    int V, E, s, u, v, w;
    vector<vi> AdjList;

10   /*
5 7 2
2 1 2
2 3 7
2 0 6
1 3 3
1 4 6
3 4 5
0 4 1

20  5 5 0
0 1 1
0 2 10
1 3 2
2 3 -10
3 4 3
*/
    scanf( "%d %d %d", &V, &E, &s);

30   AdjList.assign(V, vi());
    for (int i = 0; i < E; i++)
    {
        scanf( "%d %d %d", &u, &v, &w);
        AdjList[u].push_back(ii(v, w));
    }

35   vi dist(V, INF); dist[s] = 0;
    priority_queue< ii, vector<ii>, greater<ii> > pq; pq.push(ii(0, s));
    while (!pq.empty())
    {
40       ii front = pq.top(); pq.pop();
        int d = front.first, u = front.second;
        if (d == dist[u])
            for (int j = 0; j < (int)AdjList[u].size(); j++)
            {
                ii v = AdjList[u][j];
                if (dist[u] + v.second < dist[v.first])
                {
50                   dist[v.first] = dist[u] + v.second;
                    pq.push(ii(dist[v.first], v.first));
                }
            }
    }

45   for (int i = 0; i < V; i++)
        printf( "SSSP(%d, %d) = %d\n", s, i, dist[i]);

55   return 0;
}

```

6.5 Dijkstra Guloso – 1 p/ 1

```
typedef struct ar
{
    int peso;
    int no;
}Aresta;

class mycompare
{
public:
    bool operator() (const Aresta& a, const Aresta& b) const{
        return (a.peso>b.peso);
    }
};

int distancia[10005];

int dijkstra(vector< vector<Aresta> > &grafo, int inicial, int
destino, int tam){
    if(inicial == destino) return 0;
    priority_queue<Aresta, vector<Aresta>, mycompare> myheap;
    Aresta aux;
    aux.peso = INF;
    int n, i;

    for(i=0; i<tam; i++) if(i != inicial){
        aux.no = i;
        myheap.push(aux);
    }

    aux.peso = 0;
    aux.no = inicial;
    myheap.push(aux);

    for(i=0; i<tam; i++) distancia[i] = INF;
    distancia[inicial]=0;

    while (!myheap.empty() && n != destino){
        n = -1;

        if(myheap.top().peso > distancia[myheap.top().no]){
            myheap.pop();
            continue;
        }
        n = myheap.top().no;
        myheap.pop();

        for (i=0; i<grafo[n].size(); i++)
            if (distancia[grafo[n][i].no] > distancia[n] + grafo[n][i].
peso){
                distancia[grafo[n][i].no] = distancia[n] + grafo[n][i].peso
                ;

                aux.no = grafo[n][i].no;
                aux.peso = distancia[n] + grafo[n][i].peso;
                myheap.push(aux);
            }
    }
}
```

```

        }

55     return distancia[destino];
}

int main(){
    int casos, cidades, num;
    int i, j;
    char a1[20], a2[20];
    Aresta aux;

60    scanf( "%d", &casos);

    while(casos--){
        map<string, int> mapeador;
        scanf( "%d", &cidades);
        vector< vector<Aresta> > grafo(cidades);

        for(i = 0; i<cidades; i++){
            scanf( "%s", a1);
            mapeador[string(a1)] = i;
            scanf( "%d", &num);
            for(j=0; j<num; j++){
                scanf( "%d %d", &aux.no, &aux.peso);
                aux.no--;
                75
                grafo[i].push_back(aux);
            }
        }

80    scanf( "%d", &num);

        for(i=0; i<num; i++){
            scanf( "%s %s", a1, a2);
            printf("%d\n", dijkstra(grafo, mapeador[string(a1)], mapeador
                [string(a2)], cidades));
        }
90

        printf("|\n");
    }

95    return 0;
}

```

6.6 Edmonds Karp's

```

typedef vector<int> vi;

#define MAX_V 40
#define INF 1000000000
5
int res[MAX_V][MAX_V], mf, f, s, t;
vi p;

void augment(int v, int minEdge)

```

```

10  {
11      if (v == s) { f = minEdge; return; }
12      else if (p[v] != -1)
13      {
14          augment(p[v], min(minEdge, res[p[v]][v]));
15          res[p[v]][v] -= f; res[v][p[v]] += f;
16      }
17  }

int main()
{
    int V, k, vertex, weight;

    /*
4 0 1
2 2 70 3 30
2 2 25 3 70
3 0 70 3 5 1 25
3 0 30 2 5 1 70

30
4 0 3
2 1 100 3 100
2 2 1 3 100
1 3 100
0

35
5 1 0
0
2 2 100 3 50
3 3 50 4 50 0 50
1 4 100
1 0 125

45
5 1 0
0
2 2 100 3 50
3 3 50 4 50 0 50
1 4 100
1 0 75

50
5 1 0
0
2 2 100 3 50
2 4 5 0 5
1 4 100
1 0 125
*/
    scanf("%d %d %d", &V, &s, &t);

    memset(res, 0, sizeof res);
    for (int i = 0; i < V; i++)
    {
        scanf("%d", &k);
        for (int j = 0; j < k; j++)
        {
            scanf("%d %d", &vertex, &weight);
            res[i][vertex] = weight;
    }
}

```

```

        }
    }

70   mf = 0;
    while (1)
    {
        f = 0;
75   vi dist(MAX_V, INF); dist[s] = 0; queue<int> q; q.push(s);
    p.assign(MAX_V, -1);
    while (!q.empty())
    {
        int u = q.front(); q.pop();
        if (u == t) break;
        for (int v = 0; v < MAX_V; v++)
            if (res[u][v] > 0 && dist[v] == INF)
                dist[v] = dist[u] + 1, q.push(v), p[v] = u;
    }
    augment(t, INF);
    if (f == 0) break;
    mf += f;
}

90 printf("%d\n", mf);

return 0;
}

```

6.7 Euleriano

Caminho Eureliano é aquele que passa por todas as arestas de um Grafo.

```

typedef pair<int, int> ii;
typedef vector<ii> vii;

5  bool EulerTourExist;
int i, T, N, A, B, degree[1010], caseNo = 1;
vector<vii> AdjList;

#define TRAVERSE(c, it) \
    for (vii::iterator it = (c).begin(); it != (c).end(); it++)

10 list<int> cyc;

void EulerTour(list<int>::iterator i, int u)
{
    TRAVERSE (AdjList[u], v)
    if (v->second)
    {
        v->second = 0;
        TRAVERSE (AdjList[v->first], uu)
        if (uu->first == u && uu->second)
20        {
            uu->second = false;
            break;
        }
        EulerTour(cyc.insert(i, u), v->first);
    }
}

```

```

        }

30 int main()
{
    scanf( "%d ", &T);
    while (T--)
    {
45     AdjList.assign(51, vii());
     memset(degree, 0, sizeof degree);
     scanf( "%d ", &N);
     for (i = 1; i <= N; i++)
     {
40         scanf( "%d %d ", &A, &B);
         AdjList[A].push_back(i(B, 1));
         AdjList[B].push_back(i(A, 1));
         degree[A] = (degree[A] + 1) % 2;
         degree[B] = (degree[B] + 1) % 2;
     }

     EulerTourExist = true;
     for (i = 1; i <= 50; i++)
50         if (degree[i] == 1)
             EulerTourExist = false;

     printf( "Case # %d | n", caseNo++);
     if (EulerTourExist)
     {
55         cyc.clear();
         EulerTour(cyc.begin(), A);
         int prev = A;
         for (list<int>::iterator it = cyc.begin(); it != cyc.end();
              it++)
50             {
55                 printf( "%d %d | n", prev, *it);
50                 prev = *it;
55             }
     }
     else
65         printf( "some beads may be lost | n");
     if (T) printf( " | n");
}

70 return 0;
}

```

6.8 Flood Fill

```

/* Wetlands of Florida */

5 #define REP(i, a, b) \
    for (int i = int(a); i <= int(b); i++)

char line[150], grid[150][150];

```

```

int TC, R, C, row, col;

10 int dr[] = {1,1,0,-1,-1,-1, 0, 1};
int dc[] = {0,1,1, 1, 0,-1,-1,-1};
int floodfill(int r, int c, char c1, char c2)
{
    if (r<0 || r>=R || c<0 || c>=C) return 0;
    if (grid[r][c] != c1) return 0;
    grid[r][c] = c2;
    int ans = 1;
    REP (d, 0, 7)
        ans += floodfill(r + dr[d], c + dc[d], c1, c2);
    return ans;
}

int main()
{
25 sscanf(gets(line), "%d", &TC);
gets(line);

while (TC--)
{
30     R = 0;
     while (1)
     {
         gets(grid[R]);
         if (grid[R][0] != 'L' && grid[R][0] != 'W')
             break;
         R++;
     }
     C = (int)strlen(grid[0]);

40     strcpy(line, grid[R]);
     while (1)
     {
         sscanf(line, "%d %d", &row, &col); row--; col--;
         printf("%d\n", floodfill(row, col, 'W', '.'));
         floodfill(row, col, '.', 'W');
         gets(line);
         if (strcmp(line, "") == 0 || feof(stdin))
             break;
     }
50     if (TC)
         printf("|\n");
    }

55     return 0;
}

```

6.9 Floyd Warshall

```

int main()
{
    int V, E, u, v, w, AdjMatrix[200][200];

```

```

5  /*
5   9
0 1 2
0 2 1
0 4 3
10 1 3 4
2 1 1
2 4 1
3 0 1
3 2 3
3 4 5
*/
20
scanf( "%d %d ", &V, &E);
for (int i = 0; i < V; i++)
{
    for (int j = 0; j < V; j++)
        AdjMatrix[i][j] = INF;
    AdjMatrix[i][i] = 0;
}
25
for (int i = 0; i < E; i++)
{
    scanf( "%d %d %d ", &u, &v, &w);
    AdjMatrix[u][v] = w;
}
30
for (int k = 0; k < V; k++)
    for (int i = 0; i < V; i++)
        for (int j = 0; j < V; j++)
            AdjMatrix[i][j] = min(AdjMatrix[i][j], AdjMatrix[i][k] +
                AdjMatrix[k][j]);
35
for (int i = 0; i < V; i++)
    for (int j = 0; j < V; j++)
        printf("APSP(%d, %d) = %d\n", i, j, AdjMatrix[i][j]);
40
return 0;
}

```

6.10 Fluxo Máximo + Corte Mínimo

```

/* Fluxo maximo/ corte minimo - arestas*/
5
#include <iostream>
#include <cstdio>
#include <vector>
#include <map>
#include <queue>
#include <algorithm>
10 #include <string.h>
#include <time.h>
#include <climits>

#define MAX_V 201
#define INF INT_MAX

```

```

15  using namespace std;
16
17  typedef vector<int> vi;
18  int res[MAX_V][MAX_V], mf, f, s, t;
19  bool existe[MAX_V][MAX_V];
20  vi p;
21  int maior;
22
23  bool augment(int v, int minEdge){
24      if(v == 0) f = minEdge;
25
26      else if(p[v] != -1){
27          augment(p[v], min(minEdge, res[p[v]][v]));
28          res[p[v]][v] -= f;
29          res[v][p[v]] += f;
30      }
31
32      return (f != 0);
33  }
34
35
36  vector<bool> visitado;
37  void dfs(int atual){
38      visitado[atual] = true;
39
40      for(int i=0; i<visitado.size(); i++)
41          if(!visitado[i] && res[atual][i] != 0)
42              dfs(i);
43  }
44
45  int maxflow()
46  {
47      mf = 0, f=0;
48      do{
49          mf += f;
50          f = 0;
51          vi dist(MAX_V, INF);
52          dist[s] = 0;
53          queue<int> q;
54          q.push(s);
55          p.assign(MAX_V, -1);
56
57          while(!q.empty()){
58              int u = q.front();
59              q.pop();
60
61              if(u == t) break;
62
63              for(int v = 0; v < MAX_V; v++)
64                  if(res[u][v] > 0 && dist[v] == INF)
65                      dist[v] = dist[u] + 1, q.push(v), p[v] = u;
66          }
67      }while(augment(t, INF));
68
69      return mf;
70  }

```

```

int main(){
    int n, m;
75
    while(scanf("%d %d", &n, &m) && (n+m)){
        maior = 0;

        memset(res, 0, sizeof(res));
        memset(existe, false, sizeof(existe));

        for(int i = 0; i < m; i++)
        {
            int a, b, custo;
85
            scanf("%d %d %d", &a, &b, &custo);
            a--; b--;
            res[a][b] = custo;
            res[b][a] = custo;
            existe[a][b] = true;
            existe[b][a] = true;
        }

        s = 0;
95
        t = 1;

        maxflow();

        visitado.assign(n, false);
        dfs(s);

        for(int i=0; i<n; i++){
            for(int j=0; j<n; j++)
                if(existe[i][j] && !res[i][j] && visitado[i] !=
                   visitado[j])
                    printf("%d %d | n", i+1, j+1);
105
        }

        printf(" | n");
    }
110
    return 0;
}

```

6.11 Fluxo Máximo + Peso nos Vértices

```

/* Algoritmo Fluxo Maximo – Criacao do grafo com peso nos vertices */

#include <iostream>
#include <cstdio>
5 #include <vector>
#include <map>
#include <queue>
#include <algorithm>
#include <string.h>
10 #include <time.h>
#include <climits>

```

```

#define MAX_V 201
#define INF INT_MAX
15
using namespace std;

typedef vector<int> vi;
int res[MAX_V][MAX_V], mf, f;
20
bool existe[MAX_V][MAX_V];
vi p;
int maior;

bool augment(int v, int minEdge){
    if(v == 0) f = minEdge;

    else if(p[v] != -1){
        augment(p[v], min(minEdge, res[p[v]][v]));
        res[p[v]][v] -= f;
        res[v][p[v]] += f;
    }

    return (f != 0);
}
35

vector<bool> visitado;
void dfs(int atual){
    visitado[atual] = true;
40
    for(int i=0; i<visitado.size(); i++)
        if(!visitado[i] && res[atual][i] != 0)
            dfs(i);
}
45
int main(){
    int m, w;

    while(scanf("%d %d", &m, &w) && (m+w))
    {
        maior = 0;

        memset(res, 0, sizeof(res));

        for(int i = 0; i < m-2; i++)
        {
            int a, custo;

            scanf("%d %d", &a, &custo);
            a--;
            res[a][a+m] = res[a+m][a] = custo;

        }
65
        for(int i = 0 ; i < w; i++)
        {
            int a, b, custo;
            scanf("%d %d %d", &a, &b , &custo);
            a--; b--;
            res[a+m][b] = custo;
70
    }
}

```

```

        res[b+m][a] = custo;
    }

    res[0][m] = res[m][0] = INT_MAX;
    res[m-1][2*m -1] = res[2*m -1][m-1] = INT_MAX;

    /*for (int i = 0; i < 2*m; i++)
     for (int j = 0; j < 2*m; j++)
      if (i!=j && res[i][j] != 0)
         printf("%d %d %d\n", i+1, j+1, res[i][j]);
    printf("\n");*/
}

mf = 0, f=0;
do{
    mf += f;
    f = 0;
    vi dist(MAX_V, INF);
    dist[0] = 0;
    queue<int> q;
    q.push(0);
    p.assign(MAX_V, -1);

    while(!q.empty()){
        int u = q.front();
        q.pop();

        if(u == m-1) break;

        for(int v = 0; v < MAX_V; v++)
            if(res[u][v] > 0 && dist[v] == INF)
                dist[v] = dist[u] + 1, q.push(v), p[v] = u;
    }
} while(augment(m-1, INF));

printf("%d\n", mf);
}

return 0;
}

```

6.12 Grafo Bipartido

```

int fila[100000];
int graph[106][106];
int dim[106];
int already[106];
5 int color[106];

int BFS(int source)
{
    int i, inicio = -1, fim = -1, ext;
10
    inicio = fim = 0;
    fila[fim] = source;
    color[source] = 1;

```

```

15 |     while(inicio <= fim)
|     {
|         ext = fila[inicio++];
|         if(already[ext])
|             continue;
|         already[ext] = 1;
|
|         for(i = 0; i < dim[ext]; i++)
|         {
|             if(color[graph[ext][i]] == 0) color[graph[ext][i]] = color[
|                 ext] * (-1);
|             else { if(color[graph[ext][i]] == color[ext]) return 0; }
|             fila[++fim] = graph[ext][i];
|         }
|     }
|
|     return 1;
| }
|
int main()
{
    int i, n, m, v1, v2, cont = 1, prob;
|
    while(scanf("%d%d", &n, &m) == 2)
    {
        for(i = 1; i <= n; i++)
            dim[i] = color[i] = already[i] = 0;
|
        for(i = 0; i < m; i++)
        {
            scanf("%d%d", &v1, &v2);
|
            graph[v1][(dim[v1])++] = v2;
            graph[v2][(dim[v2])++] = v1;
        }
|
        int prob = 0;
        for(i = 1; i <= n; i++)
            if(!already[i])
                if(BFS(i) == 0) { prob = 1; break; }
|
        if(prob) printf("Instancia %d\nnao\n", cont++);
        else printf("Instancia %d\nsim\n", cont++);
    }
|
    return 0;
}

```

6.13 BFS

```

typedef pair<int, int> ii;
typedef vector<ii> vii;
typedef vector<int> vi;
|
5 int V, E, a, b, counter, s;
vector<vii> AdjList;

```

```

map<int , int> mapper, reverseMapper;
vi p;

10 void printPath(int u)
{
    if (u == s) { printf("%d", reverseMapper[u]); return; }
    printPath(p[u]);
    printf(" %d", reverseMapper[u]);
15 }

int main()
{
/*
20 13 16
10 15 15 20 20 25 10 30 30 47 47 50 25 45 45 65
15 35 35 55 20 40 50 55 35 40 55 60 40 60 60 65

13 15
25 10 15 15 20 20 25 10 30 47 50 25 45 45 65
15 35 35 55 20 40 50 55 35 40 55 60 40 60 60 65
*/
}

scanf("%d %d", &V, &E);

30 counter = 0;
AdjList.assign(V, vii());
for (int i = 0; i < E; i++)
{
35    scanf("%d %d", &a, &b);
    if (mapper.find(a) == mapper.end())
    {
        mapper[a] = counter++;
        reverseMapper[mapper[a]] = a;
40    }
    if (mapper.find(b) == mapper.end())
    {
        mapper[b] = counter++;
        reverseMapper[mapper[b]] = b;
    }
45    AdjList[mapper[a]].push_back(ii(mapper[b], 0));
    AdjList[mapper[b]].push_back(ii(mapper[a], 0));
}

50 s = mapper[35];

map<int , int> dist; dist[s] = 0;
queue<int> q; q.push(s);
p.assign(V, -1);
55 int layer = -1;
bool isBipartite = true;

while (!q.empty())
{
60    int u = q.front(); q.pop();
    if (dist[u] != layer) printf(" |nLayer %d : ", dist[u]);
    layer = dist[u];
    printf(", visit %d", reverseMapper[u]);
    for (int j = 0; j < (int)AdjList[u].size(); j++)
}

```

```

65     {
66         ii v = AdjList[u][j];
67         if (!dist.count(v.first))
68         {
69             dist[v.first] = dist[u] + 1;
70             p[v.first] = u;
71             q.push(v.first);
72         }
73         else if ((dist[v.first] % 2) == (dist[u] % 2))
74             isBipartite = false;
75     }
76 }

printf(" | nShortest path: ");
printPath(mapper[30]), printf(" | n");
printf(" is Bipartite? %d | n", isBipartite);

return 0;
}

```

6.14 DFS

```

typedef pair<int, int> ii;
typedef vector<ii> vii;
typedef vector<int> vi;

#define DFS_WHITE -1
#define DFS_BLACK 1

vector<vii> AdjList;

void printThis(char* message)
{
    printf("=====|n");
    printf("%s | n", message);
    printf("=====|n");
}

vi dfs_num;
int numCC;

void dfs(int u)
{
    printf(" %d", u);
    dfs_num[u] = DFS_BLACK;
    for (int j = 0; j < (int)AdjList[u].size(); j++)
    {
        ii v = AdjList[u][j];
        if (dfs_num[v.first] == DFS_WHITE)
            dfs(v.first);
    }
}

void floodfill(int u, int color)
{
    dfs_num[u] = color;
}

```

```

35 |     for (int j = 0; j < (int)AdjList[u].size(); j++)
|     {
|         ii v = AdjList[u][j];
|         if (dfs_num[v.first] == DFS_WHITE)
|             floodfill(v.first, color);
|     }
40 |
| }
|
vi topoSort;

45 void dfs2(int u)
{
    dfs_num[u] = DFS_BLACK;
    for (int j = 0; j < (int)AdjList[u].size(); j++)
    {
50     ii v = AdjList[u][j];
     if (dfs_num[v.first] == DFS_WHITE)
         dfs2(v.first);
    }
    topoSort.push_back(u);
55 }

#define DFS_GRAY 2
vi dfs_parent;

60 void graphCheck(int u)
{
    dfs_num[u] = DFS_GRAY;
    for (int j = 0; j < (int)AdjList[u].size(); j++)
    {
65     ii v = AdjList[u][j];
     if (dfs_num[v.first] == DFS_WHITE)
    {
        dfs_parent[v.first] = u;
        graphCheck(v.first);
    }
70     else if (dfs_num[v.first] == DFS_GRAY)
    {
        if (v.first == dfs_parent[u])
            printf(" Bidirectional (%d, %d) - (%d, %d) | n", u, v.first,
                   v.first, u);
        else
            printf(" Back Edge (%d, %d) (Cycle) | n", u, v.first);
    }
75     else if (dfs_num[v.first] == DFS_BLACK)
            printf(" Forward/Cross Edge (%d, %d) | n", u, v.first);
    }
80 }
    dfs_num[u] = DFS_BLACK;
}

85 vi dfs_low;
vi articulation_vertex;
int dfsNumberCounter, dfsRoot, rootChildren;

void articulationPointAndBridge(int u)
{
90     dfs_low[u] = dfs_num[u] = dfsNumberCounter++;
     for (int j = 0; j < (int)AdjList[u].size(); j++)

```

```

    {
        ii v = AdjList[u][j];
        if (dfs_num[v.first] == DFS_WHITE)
        {
            dfs_parent[v.first] = u;
            if (u == dfsRoot) rootChildren++;

            articulationPointAndBridge(v.first);
        }
        if (dfs_low[v.first] >= dfs_num[u])
            articulation_vertex[u] = true;
        if (dfs_low[v.first] > dfs_num[u])
            printf(" Edge (%d, %d) is a bridge |n", u, v.first);
        dfs_low[u] = min(dfs_low[u], dfs_low[v.first]);
    }
    else if (v.first != dfs_parent[u])
        dfs_low[u] = min(dfs_low[u], dfs_num[v.first]);
}
}

vi S, visited;
int numSCC;

void tarjanSCC(int u)
{
    dfs_low[u] = dfs_num[u] = dfsNumberCounter++;
    S.push_back(u);
    visited[u] = 1;
    for (int j = 0; j < (int)AdjList[u].size(); j++)
    {
        ii v = AdjList[u][j];
        if (dfs_num[v.first] == DFS_WHITE)
            tarjanSCC(v.first);
        if (visited[v.first])
            dfs_low[u] = min(dfs_low[u], dfs_low[v.first]);
    }

    if (dfs_low[u] == dfs_num[u])
    {
        printf("SCC %d:", ++numSCC);
        while (1)
        {
            int v = S.back(); S.pop_back(); visited[v] = 0;
            printf(" %d", v);
            if (u == v) break;
        }
        printf(|n");
    }
}

int main()
{
    int V, total_neighbors, id, weight;
    /*
    9
    1 1 0
    3 0 0 2 0 3 0
    */
}

```

```

150   2 1 0 3 0
      3 1 0 2 0 4 0
      1 3 0
      0
      2 7 0 8 0
155   1 6 0
      1 6 0

      8
      2 1 0 2 0
160   2 2 0 3 0
      2 3 0 5 0
      1 4 0
      0
      0
165   0
      1 6 0

      3
      1 1 0
170   1 2 0
      1 0 0

      6
      1 1 0
175   3 0 0 2 0 4 0
      1 1 0
      1 4 0
      3 1 0 3 0 5 0
      1 4 0
180
      6
      1 1 0
      5 0 0 2 0 3 0 4 0 5 0
      1 1 0
185   1 1 0
      2 1 0 5 0
      2 1 0 4 0

      8
190   1 1 0
      1 3 0
      1 1 0
      2 2 0 4 0
      1 5 0
195   1 7 0
      1 4 0
      1 6 0
      */
200
      scanf( "%d" , &V);

      AdjList.assign(V, vii());
      for (int i = 0; i < V; i++)
      {
205        scanf( "%d" , &total_neighbors);
          for (int j = 0; j < total_neighbors; j++)
          {

```

```

        scanf( "%d %d", &id, &weight);
        AdjList[i].push_back(ii(id, weight));
210    }
}

printThis("Standard DFS Demo (the input graph must be UNDIRECTED)
");
numCC = 0;
215 dfs_num.assign(V, DFS_WHITE);
for (int i = 0; i < V; i++)
    if (dfs_num[i] == DFS_WHITE)
        printf("Component %d:", ++numCC), dfs(i), printf(" |n");
printf("There are %d connected components |n", numCC);

220 printThis("Flood Fill Demo (the input graph must be UNDIRECTED)")
;
numCC = 0;
dfs_num.assign(V, DFS_WHITE);
for (int i = 0; i < V; i++)
225    if (dfs_num[i] == DFS_WHITE)
        floodfill(i, ++numCC);
for (int i = 0; i < V; i++)
    printf("Vertex %d has color %d |n", i, dfs_num[i]);

230 printThis("Topological Sort (the input graph must be DAG)");
topoSort.clear();
dfs_num.assign(V, DFS_WHITE);
for (int i = 0; i < V; i++)
235    if (dfs_num[i] == DFS_WHITE)
        dfs2(i);
reverse(topoSort.begin(), topoSort.end());
for (int i = 0; i < (int)topoSort.size(); i++)
    printf(" %d", topoSort[i]);
printf(" |n");

240 printThis("Graph Edges Property Check");
numCC = 0;
dfs_num.assign(V, DFS_WHITE); dfs_parent.assign(V, -1);
for (int i = 0; i < V; i++)
245    if (dfs_num[i] == DFS_WHITE)
        printf("Component %d: |n", ++numCC), graphCheck(i);

printThis("Articulation Points & Bridges (the input graph must be
UNDIRECTED)");
dfsNumberCounter = 0; dfs_num.assign(V, DFS_WHITE); dfs_low.
assign(V, 0);
250 dfs_parent.assign(V, -1); articulation_vertex.assign(V, 0);
printf("Bridges: |n");
for (int i = 0; i < V; i++)
    if (dfs_num[i] == DFS_WHITE)
    {
        dfsRoot = i; rootChildren = 0;
        articulationPointAndBridge(i);
        articulation_vertex[dfsRoot] = (rootChildren > 1);
    }
255 printf("Articulation Points: |n");
for (int i = 0; i < V; i++)
    if (articulation_vertex[i])

```

```

        printf(" Vertex %d\n", i);

    printThis("Strongly Connected Components (the input graph must be
              DIRECTED)");
265    dfs_num.assign(v, DFS_WHITE); dfs_low.assign(v, 0); visited.
        assign(v, 0);
    dfsNumberCounter = numSCC = 0;
    for (int i = 0; i < V; i++)
        if (dfs_num[i] == DFS_WHITE)
            tarjanSCC(i);

270    return 0;
}

```

6.15 Kruskal

6.15.1 Implementação do Livro (inc. Primm)

```

typedef pair<int, int> ii;
typedef vector<int> vi;
typedef vector<ii> vii;

5   vi pset(1000), setSize(1000); int _numDisjointSets;

void initSet(int N)
{
10    setSize.assign(N, 1); _numDisjointSets = N;
    pset.assign(N, 0); for (int i = 0; i < N; i++) pset[i] = i;
}

int findSet(int i) { return (pset[i] == i) ? i : (pset[i] = findSet(
15    pset[i])); }

bool isSameSet(int i, int j) { return findSet(i) == findSet(j); }

void unionSet(int i, int j)
{
20    if (!isSameSet(i, j))
    {
        _numDisjointSets--;
        setSize[findSet(j)] += setSize[findSet(i)];
        pset[findSet(i)] = findSet(j);
25    }
}

int numDisjointSets() { return _numDisjointSets; }

30 int sizeOfSet(int i) { return setSize[findSet(i)]; }

vector<vii> AdjList;
vi taken;
priority_queue<ii> pq;

35 void process(int vtx)
{
    taken[vtx] = 1;
}

```

```

40   for (int j = 0; j < AdjList[vtx].size(); j++)
41   {
42     ii v = AdjList[vtx][j];
43     if (!taken[v.first]) pq.push(ii(-v.second, -v.first));
44   }
45
46 int main()
47 {
48   int V, E, u, v, w;
49
50   /*
51    5 7
52    0 1 4
53    0 2 4
54    0 3 6
55    0 4 6
56    1 2 2
57    2 3 8
58    3 4 9
59   */
60
61   scanf("%d %d", &V, &E);
62
63   AdjList.assign(V, vii());
64   vector< pair<int, ii> > EdgeList;
65   for (int i = 0; i < E; i++)
66   {
67     scanf("%d %d %d", &u, &v, &w);
68     EdgeList.push_back(make_pair(w, ii(u, v)));
69     AdjList[u].push_back(ii(v, w));
70     AdjList[v].push_back(ii(u, w));
71   }
72   sort(EdgeList.begin(), EdgeList.end());
73
74   int mst_cost = 0; initSet(V);
75   for (int i = 0; i < E; i++)
76   {
77     pair<int, ii> front = EdgeList[i];
78     if (!isSameSet(front.second.first, front.second.second))
79     {
80       mst_cost += front.first;
81       unionSet(front.second.first, front.second.second);
82     }
83   }
84
85   printf("MST cost = %d (Kruskal's) |n", mst_cost);
86
87   taken.assign(V, 0);
88   process(0);
89   mst_cost = 0;
90   while (!pq.empty())
91   {
92     ii front = pq.top(); pq.pop();
93     u = -front.second, w = -front.first;
94     if (!taken[u])
95       mst_cost += w, process(u);
96   }

```

```

    printf( "MST cost = %d (Prim's) |n", mst_cost);

100
    return 0;
}

```

6.15.2 Outra

```

/* Algoritmo de Kruskal para Árvore Geradora Mínima*/

#include <iostream>
#include <vector>
5 #include <set>
#include <algorithm>
#include <cmath>
#include <cstdlib>

10 #define MAX_NODES 100

using namespace std;

15 int pi[MAX_NODES];
int rank[MAX_NODES];

20 typedef struct{
    int origem;
    int destino;
    double peso;
}aresta;

25 bool compara(const aresta& a1, const aresta& a2)
{
    if(a1.peso != a2.peso)
        return a1.peso < a2.peso;

    if(a1.origem != a2.origem)
        return a1.origem < a2.origem;
30
    return a1.destino < a2.destino;
}

35 void makeset(int x)
{
    pi[x] = x;
    rank[x] = 0;
}

40 int procura(int x)
{
    while(x != pi[x])
        x = pi[x];
    return x;
}

45 void uniao(int x, int y)
{

```

```

    int rx = procura(x), ry = procura(y);
50
    if(rx == ry)
        return;

    if(rank[rx] > rank[ry])
55        pi[ry] = rx;
    else
    {
        pi[rx] = ry;
        if(rank[rx] == rank[ry])
60            rank[ry] = rank[ry] + 1;
    }
}

double kruskal(vector<aresta> arestas, int nVertices, vector<pair<
    int, int> & arvore)
65
{
    double custo = 0;

    for(int i = 0; i < nVertices; i++)
        makeset(i);

70    sort(arestas.begin(), arestas.end(), compara);

    for(int i = 0; i < arestas.size(); i++)
    {
        if( procura(arestas[i].origem) != procura(arestas[i].destino) )
45
        {
            custo += arestas[i].peso;
            arvore.push_back(pair<int, int>(arestas[i].origem, arestas[i]
                .destino));
            uniao(arestas[i].origem, arestas[i].destino);
        }
    }

80    return custo;
}
85
int main()
{
    vector<aresta> arestas;
    vector<pair<int, int> > arvore;
90    aresta tmp;

    tmp.origem = 0;
    tmp.destino = 1;
    tmp.peso = 1;
95    arestas.push_back(tmp);

    tmp.origem = 1;
    tmp.destino = 0;
    tmp.peso = 1;
100   arestas.push_back(tmp);

    tmp.origem = 1;
    tmp.destino = 2;
    tmp.peso = 2;

```

```

105     arestas.push_back(tmp);

    tmp.origem = 2;
    tmp.destino = 1;
    tmp.peso = 2;
110    arestas.push_back(tmp);

    tmp.origem = 2;
    tmp.destino = 3;
    tmp.peso = 3;
115    arestas.push_back(tmp);

    tmp.origem = 3;
    tmp.destino = 2;
    tmp.peso = 3;
120    arestas.push_back(tmp);

    tmp.origem = 1;
    tmp.destino = 3;
    tmp.peso = 4;
125    arestas.push_back(tmp);

    tmp.origem = 3;
    tmp.destino = 1;
    tmp.peso = 4;
130    arestas.push_back(tmp);

    tmp.origem = 0;
    tmp.destino = 3;
    tmp.peso = 5;
135    arestas.push_back(tmp);

    tmp.origem = 3;
    tmp.destino = 0;
    tmp.peso = 5;
140    arestas.push_back(tmp);

    double custo = kruskal(arestas, 4, arvore);

145    cout << "Custo: " << custo << endl;
    for(int i = 0; i < arvore.size(); i++)
        cout << arvore[i].first << " " << arvore[i].second << endl;

150
    return 0;
}

```

6.16 Menor Caminho em Árvores

6.16.1 DFS

```

/* Menor caminho em arvores usando DFS */

#include <iostream>
#include <cstdio>
5 #include <vector>

```

```

#include <stack>
#include <stdlib.h>
using namespace std;
vector<int> p2;

10 void dfs(vector< vector<int> > &grafo, int ini, int fim){
    stack<int> pilha;
    vector<bool> map(grafo.size(), false);
    pilha.push(ini);
    map[ini] = true;

    while(!pilha.empty()){
        TESTE:int atual = pilha.top();

        20 if(atual == fim){
            while(!pilha.empty()){
                p2.push_back(pilha.top());
                pilha.pop();
            }
        }

        25 return;
    }

    30 for(int i=0; i<grafo[atual].size(); i++){
        if(map[grafo[atual][i]] == false){
            map[grafo[atual][i]] = true;
            pilha.push(grafo[atual][i]);

            35 goto TESTE;
        }
    }

    pilha.pop();
}

40 }

int main()
{
    int n;

    45 while(scanf("%d", &n) && n)
    {

        vector< vector<int> > grafo(n);

        50 for(int i = 0; i < n-1; i++)
        {
            int a, b;
            scanf("%d %d", &a, &b);
            a--;
            b--;
            grafo[a].push_back(b);
            grafo[b].push_back(a);
        }

        55 int l;
        scanf("%d", &l);

        60 for(int i = 0; i < l; i++)
    }
}

```

```

65      {
66          int a, b;
67
68          scanf( "%d%d", &a, &b);
69          a--; b--;
70
71          dfs(grafo, a, b);
72          if(p2.size() % 2 == 0)
73          {
74              int a = p2[p2.size()/2]+1, b = p2[p2.size()/2-1]+1;
75
76              printf( "The fleas jump forever between %d and %d.\n", min(a
77                  ,b), max(a,b));
78          }
79          else printf( "The fleas meet at %d.\n",p2[p2.size()/2]+1);
80
81          p2.clear();
82      }
83
84      return 0;
85  }

```

6.16.2 Menor Ancestral Comum

```

/* Menor caminho em arvores (algoritmo específico) */

5 #include <iostream>
# include <cstring>
# include <cstdlib>
# include <cstdio>

using namespace std;

10 typedef struct ar{
    unsigned long long peso;
    int pai;
}Aresta;

15 int main(){
    int n, m, c1, c2, atual, mapAtual;
    Aresta formigueiros[100001];
    int mapeador[100001];

20
    while( scanf( "%d", &n) && n){
        for( int i=1; i<n; i++)
            scanf( "%d %lld", &formigueiros[i].pai, &formigueiros[i].peso)
            ;
    }

25    scanf( "%d", &m);
    memset(mapeador, 0, sizeof(int)*n);
    mapAtual = 1;

30    for( int i=0; i<m; i++){
        if(i != 0) printf( " ");
        scanf( "%d %d", &c1, &c2);
    }
}

```

```

    unsigned long long somatorio = 0;

    atual = c1;
35   while(atual != 0){
        if(mapeador[atual] != mapAtual){
            mapeador[atual] = mapAtual;
            somatorio += formigueiros[atual].peso;
        }
        else somatorio -= formigueiros[atual].peso;

        atual = formigueiros[atual].pai;
    }

45   atual = c2;
    while(atual != 0){
        if(mapeador[atual] != mapAtual){
            mapeador[atual] = mapAtual;
            somatorio += formigueiros[atual].peso;
50        }
        else somatorio -= formigueiros[atual].peso;

        atual = formigueiros[atual].pai;
    }

55   printf("%lld ", somatorio);
    mapAtual++;
}

60   printf("|\n");
}

return 0;
}

```

6.17 MCBM (Max Cardinality Bipartite Matching)

```

typedef pair<int, int> ii;
typedef vector<int> vi;
typedef vector<ii> vii;

5  vector<vii> AdjList;
vi owner, visited;

int Aug(int left)
{
10  if(visited[left])
      return 0;

    visited[left] = true;
    for(int j = 0; j < (int)AdjList[left].size(); j++)
15  {
        int right = AdjList[left][j].first;
        if (owner[right] == -1 || Aug(owner[right]))
        {
            owner[right] = left;
            return 1;
20

```

```

        }
    }

    return 0;
25}

bool isprime(int v)
{
    int primes[10] = {2,3,5,7,11,13,17,19,23,29};
    for(int i = 0; i < 10; i++)
        if(primes[i] == v)
            return true;
    return false;
}
35

int main()
{
    int i, j;

    /*
     * Graph in Figure 4.34 can be built on the fly
     * we know there are 6 vertices in this bipartite graph, left
     * side are numbered 0,1,2, right side 3,4,5
     * int V = 6, num_vertices_on_left = 3, set1[3] = {1,7,11}, set2
     * [3] = {4,10,12};

     * Graph in Figure 4.35 can be built on the fly
     * we know there are 5 vertices in this bipartite graph, left
     * side are numbered 0,1, right side 3,4,5
     * int V = 5, num_vertices_on_left = 2, set1[2] = {1,7}, set2[3]
     * = {4,10,12};

     * build the bipartite graph, only directed edge from left to
     * right is needed
50
     * AdjList.assign(V, vii());
     * for (i = 0; i < num_vertices_on_left; i++)
     * for (j = 0; j < 3; j++)
     * if (isprime(set1[i] + set2[j]))
     * AdjList[i].push_back(ii(3 + j, 0));
     */
55

    int V = 4, num_vertices_on_left = 2;

60    AdjList.assign(V, vii());

    AdjList[0].push_back(ii(2, 1));
    AdjList[0].push_back(ii(3, 1));
    AdjList[1].push_back(ii(2, 1));

65

    int cardinality = 0;
    owner.assign(V, -1);
    for (int left = 0; left < num_vertices_on_left; left++)
    {
        visited.assign(num_vertices_on_left, 0);
        cardinality += Aug(left);
    }

```

```

75   printf( "Found %d matchings\n", cardinality);

    return 0;
}

```

6.18 Ordenação Topológica

Vide DFS.

6.19 Representações

```

typedef pair<int, int> ii;
typedef vector<ii> vii;

int main()
{
    int V, E, total_neighbors, id, weight, a, b;
    int AdjMat[100][100];
    vector<vii> AdjList;
    priority_queue< pair<int, ii> > EdgeList;
10

/*
6
    0  10  0  0  100  0
    10 0  7  0  8  0
    0  7  0  9  0  0
    0  0  9  0  20  5
    100 8  0  20  0  0
    0  0  0  5  0  0
20
6
    2 2 10 5 100
    3 1 10 3 7 5 8
    2 2 7 4 9
    3 3 9 5 20 6 5
25
    3 1 100 2 8 4 20
    1 4 5
    7
    1 2 10
    1 5 100
30
    2 3 7
    2 5 8
    3 4 9
    4 5 20
    4 6 5
35
*/
36

scanf( "%d", &V);
for (int i = 0; i < V; i++)
    for (int j = 0; j < V; j++)
        scanf( "%d", &AdjMat[i][j]);

printf( "Neighbors of vertex 0:\n");
40
for (int j = 0; j < V; j++)
    if (AdjMat[0][j])

```

```

45     printf("Edge 0-%d (weight = %d) | n", j, AdjMat[0][j]);
50
54     scanf("%d", &V);
      AdjList.assign(V, vii());
      for (int i = 0; i < V; i++)
      {
        scanf("%d", &total_neighbors);
        for (int j = 0; j < total_neighbors; j++)
        {
          scanf("%d %d", &id, &weight);
          AdjList[i].push_back(ii(id - 1, weight));
        }
      }
55
60   printf("Neighbors of vertex 0: | n");
61   for (vii::iterator j = AdjList[0].begin(); j != AdjList[0].end();
         j++)
62     printf("Edge 0-%d (weight = %d) | n", j->first, j->second);

63   scanf("%d", &E);
64   for (int i = 0; i < E; i++)
65   {
     scanf("%d %d %d", &a, &b, &weight);
     EdgeList.push(make_pair(-weight, ii(a, b)));
   }
66
67   for (int i = 0; i < E; i++)
68   {
     pair<int, ii> edge = EdgeList.top(); EdgeList.pop();
     printf("weight: %d (%d-%d) | n", -edge.first, edge.second.first,
           edge.second.second);
   }
69
70   return 0;
71 }
```

6.20 Algoritmo de Tarjan

```

#define MAX 4000

5   int n, t=0;
    int cont, numcomp;
    int id[MAX], pre[MAX], low[MAX];
    int stack[MAX], top;
    int graph[MAX][MAX];
    int dim[MAX];
10  int present[MAX];

    int mat[MAX][MAX];

    void empilha(v)
15  {
    stack[top++] = v;
}
```

```

10   int desempilha()
20   {
21     return stack[--top];
22   }

23   void directedFW(int v)
24   {
25     int i, aux, min = low[v] = pre[v] = cont++;
26
27     empilha(v);
28
29     for(i=0;i<dim[v];i++)
30     {
31       aux = graph[v][i];
32       if(pre[aux] == -1)
33         directedFW(aux);
34       if(low[aux] < min)
35         min = low[aux];
36     }
37
38     if(min < low[v])
39     {
40       low[v] = min;
41       return;
42     }
43
44     do
45     {
46       id[i=desempilha()] = numcomp;
47       low[i] = n;
48     } while(i!=v);
49
50     numcomp++;
51   }

52   int main()
53   {
54     int i, j, m, v1, v2, test = 1;
55
56     while(1)
57     {
58       scanf( "%d%d ", &n, &m);
59
60       if(!n && !m)
61         break;
62
63       for(i=0; i<n; i++)
64       {
65         pre[i] = low[i] = id[i] = -1;
66       }
67
68       memset(dim, 0, n*sizeof(int));
69
70       numcomp = cont = 0;
71
72       for(i = 0; i<m; i++)
73       {
74         scanf( "%d%d ", &v1, &v2);
75       }
76     }
77   }

```

```

        graph[v1-1][dim[v1-1]++] = v2-1;
    }

80   for(i = 0; i<n; i++)
{
    if(pre[i] == -1)
    {
        directedFW(i);
    }
}

    printf( "Teste %d | n%s | n| n", test++, numcomp==1? "S": "N");
}

90 return 0;
}

```

6.21 Union Find

```

typedef vector<int> vi;

vi pset(1000), setSize(1000); int _numDisjointSets;
5
void initSet(int N)
{
    setSize.assign(N, 1);
    _numDisjointSets = N;
10   pset.assign(N, 0);
    for (int i = 0; i < N; i++)
        pset[i] = i;
}

15 int findSet(int i)
{
    return (pset[i] == i) ? i : (pset[i] = findSet(pset[i]));
}

20 bool isSameSet(int i, int j) { return findSet(i) == findSet(j); }

25 void unionSet(int i, int j)
{
    if (!isSameSet(i, j))
    {
        _numDisjointSets--;
        setSize[findSet(j)] += setSize[findSet(i)];
        pset[findSet(i)] = findSet(j);
    }
30 }

int numDisjointSets() { return _numDisjointSets; }

int sizeOfSet(int i) { return setSize[findSet(i)]; }

35 int main()
{

```

```

    printf("Assume that there are 5 disjoint sets initially |n");
    initSet(5);
40   unionSet('A' - 'A', 'B' - 'A');
    unionSet('A' - 'A', 'C' - 'A');
    unionSet('D' - 'A', 'B' - 'A');
    printf("findSet(A) = %d\n", findSet('A' - 'A'));
    printf("findSet(B) = %d\n", findSet('B' - 'A'));
45   printf("findSet(C) = %d\n", findSet('C' - 'A'));
    printf("findSet(D) = %d\n", findSet('D' - 'A'));
    printf("findSet(E) = %d\n", findSet('E' - 'A'));
    printf("isSameSet(A, E) = %d\n", isSameSet('A' - 'A', 'E' - 'A'))
        ;
    printf("isSameSet(A, B) = %d\n", isSameSet('A' - 'A', 'B' - 'A'))
        ;
50
    return 0;
}

```

6.22 Union Find

```

typedef vector<int> vi;

vi pset(1000), setSize(1000); int _numDisjointSets;
5
void initSet(int N)
{
    setSize.assign(N, 1);
    _numDisjointSets = N;
10   pset.assign(N, 0);
    for (int i = 0; i < N; i++)
        pset[i] = i;
}

15 int findSet(int i)
{
    return (pset[i] == i) ? i : (pset[i] = findSet(pset[i]));
}

20 bool isSameSet(int i, int j) { return findSet(i) == findSet(j); }

25 void unionSet(int i, int j)
{
    if (!isSameSet(i, j))
    {
        _numDisjointSets--;
        setSize[findSet(j)] += setSize[findSet(i)];
        pset[findSet(i)] = findSet(j);
    }
30 }

int numDisjointSets() { return _numDisjointSets; }

int sizeOfSet(int i) { return setSize[findSet(i)]; }

35 int main()

```

```

{
    printf("Assume that there are 5 disjoint sets initially |n");
    initSet(5);
40    unionSet('A' - 'A', 'B' - 'A');
    unionSet('A' - 'A', 'C' - 'A');
    unionSet('D' - 'A', 'B' - 'A');
    printf("findSet(A) = %d |n", findSet('A' - 'A'));
    printf("findSet(B) = %d |n", findSet('B' - 'A'));
45    printf("findSet(C) = %d |n", findSet('C' - 'A'));
    printf("findSet(D) = %d |n", findSet('D' - 'A'));
    printf("findSet(E) = %d |n", findSet('E' - 'A'));
    printf("isSameSet(A, E) = %d |n", isSameSet('A' - 'A', 'E' - 'A'))
        ;
    printf("isSameSet(A, B) = %d |n", isSameSet('A' - 'A', 'B' - 'A'))
        ;
50
    return 0;
}

```

6.23 Union Find

```

typedef vector<int> vi;

vi pset(1000), setSize(1000); int _numDisjointSets;
5
void initSet(int N)
{
    setSize.assign(N, 1);
    _numDisjointSets = N;
10    pset.assign(N, 0);
    for (int i = 0; i < N; i++)
        pset[i] = i;
}

15 int findSet(int i)
{
    return (pset[i] == i) ? i : (pset[i] = findSet(pset[i]));
}

20 bool isSameSet(int i, int j) { return findSet(i) == findSet(j); }

void unionSet(int i, int j)
{
25    if (!isSameSet(i, j))
    {
        _numDisjointSets--;
        setSize[findSet(j)] += setSize[findSet(i)];
        pset[findSet(i)] = findSet(j);
    }
30}
35 int numDisjointSets() { return _numDisjointSets; }

int sizeOfSet(int i) { return setSize[findSet(i)]; }

```

```

int main()
{
    printf("Assume that there are 5 disjoint sets initially |n|");
    initSet(5);
    unionSet('A' - 'A', 'B' - 'A');
    unionSet('A' - 'A', 'C' - 'A');
    unionSet('D' - 'A', 'B' - 'A');
    printf("findSet(A) = %d |n|", findSet('A' - 'A'));
    printf("findSet(B) = %d |n|", findSet('B' - 'A'));
    printf("findSet(C) = %d |n|", findSet('C' - 'A'));
    printf("findSet(D) = %d |n|", findSet('D' - 'A'));
    printf("findSet(E) = %d |n|", findSet('E' - 'A'));
    printf("isSameSet(A, E) = %d |n|", isSameSet('A' - 'A', 'E' - 'A'));
    ;
    printf("isSameSet(A, B) = %d |n|", isSameSet('A' - 'A', 'B' - 'A'));
    ;
50
    return 0;
}

```

6.24 Cordal

```

#include <stdio.h>
#include <vector>
#include <list>
#include <queue>
5
#define INF 1000000
#define itrList list< list<int>*>::iterator

using namespace std;

10
struct Edge{
    int u, v;

    Edge(int u=0, int v=0):u(u),v(v){}
};

15
    int other(int a=0){ return (u==a)?v:u; }
};

20
struct Graph{
    vector< vector<int> > Adj;
    vector<Edge> E;
    vector<vector<bool> > A;
    vector<bool> vis;
    int n, m, pos;
    bool istree;
};

25
Graph(int n=0, int m=0, bool ist=false):Adj(n),E(m),n(n),m(m),
    istree(ist){
    if(!ist){
        A.resize(n);
        for(int i=0; i<n; i++)
            A[i].assign(n, false);
    }
}

```

```

35   inline
void insert(int u, int v){
    E.push_back(Edge(u,v));
    Adj[u].push_back(E.size()-1);

    Adj[v].push_back(E.size()-1); //Se for nao direcionado

    if (!istree) A[u][v] = A[v][u] = true;
}

40
45   vector<int> Lex_BFS();
bool isChordal();

50   void dfs_post(int u, int seq[]){
    int i, v, p;
    p=Adj[u].size();
    vis[u]=1;

    for (i=0;i<p;i++){
        v = E[Adj[u][i]].other(u);
        if (!vis[v])
            dfs_post(v,seq);
    }
    seq[pos]=u;
    pos++;
}
60
};

65   bool Graph::isChordal(){
    vector<int> pi;
    int u, v, i, p;
    vector<vector<int>> RN(n+1);

70   int seq[n+1], parent[n+1];
    bool X[n+1], Y[n+1];
    pi=Lex_BFS();

75   for (u=0; u<n; u++){
        p = Adj[u].size();
        parent[u] = n;
        for (i=0; i<p; i++){
            v = E[Adj[u][i]].other(u);

            if (pi[v]>pi[u]){
                RN[u].push_back(v);
                if (pi[v]<pi[parent[u]]) parent[u]=v;
            }
        }
    }

85   Graph T(n+1,n-1,true);
    for (u=0; u<n; u++)
        T.insert(parent[u],u);

    pos=0;
90   vis.assign(n+1, false);
    dfs_post(n-1,seq);
}

```

```

95
    for(u=0; u<n; u++){
        for(i=0; i<n; i++) X[i] = Y[i] = false;
        p = RN[u].size();
        for(i=0; i<p; i++) X[RN[u][i]] = true;

        X[parent[u]] = false;
        v = parent[u];
        p = RN[v].size();

        for(i=0; i<p; i++) Y[RN[v][i]] = true;

        for(i=0; i<n; i++) if(X[i] && !Y[i]) return false;
    }

    return true;
}

110
vector<int> Graph::Lex_BFS(){
    vector<int> pi(n+1);
    list<int>::iterator it;
    vector<itrList> pos;
    int u, x, i;
    list<list<int>*> L;

    list<int> *S, *Y;
    itrList X;

120
    S=new list<int>();
    for(u=0; u<n; u++) S->push_back(u);

    L.push_back(S);
    pi[n] = n;
    i = n-1;

    while(!L.empty()){
130
        S = L.front();
        x = S->front();
        S->pop_front();

        if(S->empty()){
            L.pop_front();
            delete S;
        }

        pi[x] = i--;
140
        for(X = L.begin(); X != L.end();){
            Y=new list<int>();
            S=*X;

            for(it=S->begin(); it!=S->end(); ) {
                if(A[x][*it]){
                    Y->push_back(*it);
                    it=S->erase(it);
                }
            }
        }
    }
}

```

```

150     else it++;
    }

    if(Y->size() > 0) L.insert(X,Y);
    else delete Y;
155
    if(S->size()==0) X=L.erase(X);
    else X++;
}
}

160 return pi;
}

int main()
{
    int n, m;
    int u, v;

    while(scanf("%d %d", &n, &m) == 2 && (n+m))
    {
        Graph G(n,m);

        while(m--)
        {
            scanf("%d %d",&u,&v);

            G.insert(u-1,v-1);
        }

        if(G.isChordal()) printf("1");
        else printf("0");
    }
    printf("\n");
    return 0;
}

```

7 Grid

7.1 Order

```

/* order.c

Demonstrate traversal orders on a grid.

5   by: Steven Skiena
      begun: July 14, 2002
*/
/*
10  Copyright 2003 by Steven S. Skiena; all rights reserved.

Permission is granted for use in non-commercial applications
provided this copyright notice remains intact and unchanged.

```

15 | This program appears in my book:

"Programming Challenges: The Programming Contest Training Manual"
 by Steven Skiena and Miguel Revilla , Springer-Verlag , New York
 2003.

20 | See our website www.programming-challenges.com for additional
 information .

This book can be ordered from Amazon.com at

<http://www.amazon.com/exec/obidos/ASIN/0387001638/thealgorithmrepo/>

```

25 */
*/



#include "geometry.h"

30 row_major(int n, int m)
{
    int i,j; /* counters */

35     for (i=1; i<=n; i++)
        for (j=1; j<=m; j++)
            process(i,j);
}

40 column_major(int n, int m)
{
    int i,j; /* counters */

45     for (j=1; j<=m; j++)
        for (i=1; i<=n; i++)
            process(i,j);
}

50 snake_order(int n, int m)
{
    int i,j; /* counters */

55     for (i=1; i<=n; i++)
        for (j=1; j<=m; j++)
            process(i, j + (m+1-2*j) * ((i+1) % 2));
}

60 diagonal_order(int n, int m)
{
    int d,j; /* diagonal and point counters */
    int pcount; /* points on diagonal */
    int height; /* row of lowest point */

65     for (d=1; d<=(m+n-1); d++) {
        height = 1 + max(0,d-m);
        pcount = min(d, (n-height+1));
        for (j=0; j<pcount; j++)
            process(min(m,d)-j, height+j);
70     }
}
```

```

    }

process(int i, int j)
{
    printf("(%d,%d) | n",i,j);
}

main()
{
    printf("row_major | n");
    row_major(5,5);

    printf(" | ncolumn_major | n");
    column_major(3,3);

printf(" | nsnake_order | n");
    snake_order(5,5);

printf(" | ndiagonal_order | n");
    diagonal_order(3,4);

    printf(" | ndiagonal_order | n");
    diagonal_order(4,3);

}

```

7.2 Plates

```

/* plates.c

Compute the number of circles in two different disk packings.
Assuming we have an $w \times l$ box, how many unit disks
can we pack in there assuming we have w disks on the bottom?

by: Steven Skiena
begun: April 4, 2002
*/
10
/*
Copyright 2003 by Steven S. Skiena; all rights reserved.

Permission is granted for use in non-commerical applications
provided this copyright notice remains intact and unchanged.

This program appears in my book:

"Programming Challenges: The Programming Contest Training Manual"
by Steven Skiena and Miguel Revilla, Springer-Verlag, New York
2003.

See our website www.programming-challenges.com for additional
information.

This book can be ordered from Amazon.com at

```

```

http://www.amazon.com/exec/obidos/ASIN/0387001638/thealgorithmrepo/
*/
30
#include <stdio.h>
#include <math.h>

35 /* how many triangular-lattice layers of radius r balls fit in
   height h?
*/
40 int dense_layers(double w, double h, double r)
{
    double gap;      /* distance between layers */
45    if ((2*r) > h) return(0);

    gap = 2.0 * r * (sqrt(3)/2.0);
    return( 1 + floor((h-2.0*r)/gap) );
}

50 int plates_per_row(int row, double w, double r)
{
    int plates_per_full_row;          /* number of plates in full
   /even row */

    plates_per_full_row = floor(w/(2*r));

    if ((row % 2) == 0) return(plates_per_full_row);
55    if (((w/(2*r))-plates_per_full_row) >= 0.5)      /* odd row
   full, too */
        return(plates_per_full_row);
    else
        return(plates_per_full_row - 1);
60 }

/* How many radius r plates fit in a hexagonal-lattice packed w*h
box?
*/
65 int dense_plates(double w, double l, double r)
{
    int layers;      /* number of layers of balls */
70    layers = dense_layers(w,l,r);

    return (ceil(layers/2.0) * plates_per_row(0,w,r) +
            floor(layers/2.0) * plates_per_row(1,w,r));
}

75 int grid_plates(double w, double h, double r)
{
    int layers;      /* number of layers of balls */
    layers = floor(h/(2*r));
}

```

```

80         return (layers * plates_per_row(0,w,r));
}

/* Hexagonal coordinates start with the center of disk (0,0) at
85   geometric point (0,0). The hexagonal coordinate $(xh,yh)$
   refers to the center of the disk on the horizontal row $xh$ and
   positive-slope diagonal $yh$. The geometric coordinate of such a
   point is a function of the radius of the disk $r$.
*/
90
hex_to_geo(int xh, int yh, double r, double *xg, double *yg)
{
    *yg = (2.0 * r) * xh * (sqrt(3)/2.0);
    *xg = (2.0 * r) * xh * (1.0/2.0) + (2.0 * r) * yh;
}
95

geo_to_hex(double xg, double yg, double r, double *xh, double *yh)
{
    *xh = (2.0/sqrt(3)) * yg / (2.0 * r);
100   *yh = (xg - (2.0 * r) * (*xh) * (1.0/2.0)) / (2.0 * r);
}

/* Under the hexagonal coordinate system, the set of hexagons
   defined by coordinates $(hx,hy)$, where $0 \leq hx \leq xmax$ and $0 \leq hx \leq
   ymax$ forms a diamond-shaped patch, not a conventional axis-oriented
105  rectangle. To solve this problem, we define array coordinates so that $(ax,ay)$ refers to the position in an axis-oriented rectangle with (0,0) as the lower righthand point in the matrix.
*/
110
array_to_hex(int xa, int ya, int *xh, int *yh)
{
    *xh = xa;
    *yh = ya - xa + ceil(xa/2.0);
}
115

hex_to_array(int xh, int yh, int *xa, int *ya)
{
    *xa = xh;
120   *ya = yh + xh - ceil(xh/2.0);
}

int plates_on_top(int xh, int yh, double w, double l, double r)
125 {
    int number_on_top = 0; /* total plates on top */
    int layers; /* number of rows in grid */
    int rowlength; /* number of plates in row */
    int row; /* counter */
    int xla,yla,xra,yra; /* array coordinates */

    layers = dense_layers(w,l,r);

    for (row=xh+1; row<layers; row++) {
        rowlength = plates_per_row(row,w,r) - 1;
135

```

```

    hex_to_array(row,yh-(row-xh),&xla,&yla);
    if (yla < 0) yla = 0; /* left boundary */

140   hex_to_array(row,yh,&xra,&yra);
    if (yra > rowlength) yra = rowlength; /* right boundary */

/* printf("row=%d yla=%d yra=%d | n",row,yla,yra); */

145   number_on_top += yra - yla + 1;
}

return(number_on_top);
}

150 main()
{
    double w; /* box width */
    double l; /* box length */
    double r; /* plate radius */

155   int i,j; /* counters */
    int xh,yh,xa,ya;
    double xhf,yhf,xg,yg;
    int xmax,ymax;

160   printf("input box width, box length, and plate radius: | n");
    scanf("%lf %lf %lf",&w,&l,&r);
    printf("box width=%lf, box length=%lf, and plate radius=%lf: | n",w
        ,l,r);

165   printf("dense packing = %d | n", dense_plates(w,l,r));
    printf("grid packing = %d | n", grid_plates(w,l,r));

    /* print all the possible hexes in the box */

170   xmax = floor(w / (2*r));
    ymax = dense_layers(w,l,r);

    /*
     * for (i=0; i<=xmax; i++)
     *   for (j=0; j<=ymax; j++) {
     *     printf("array(%d,%d) ",i,j);
     *     array_to_hex(i,j,&xh,&yh);
     *     printf("to hex(%d,%d) ",xh,yh);
     *     hex_to_geo(xh,yh,r,&xg,&yg);
     *     printf("to geo(%4.2f,%4.2f) ",xg,yg);
     *     geo_to_hex(xg,yg,r,&xhf,&yhf);
     *     printf("to hex(%4.2f,%4.2f) ",xhf,yhf);
     *     hex_to_array(xh,yh,&xa,&ya);
     *     printf("to array(%d,%d) | n",xa,ya);
     *   }
     */
190   for (i=0; i<xmax; i++)
    printf("(0,%d) has %d on top. | n",i,plates_on_top(0,i,w,l,r));
}

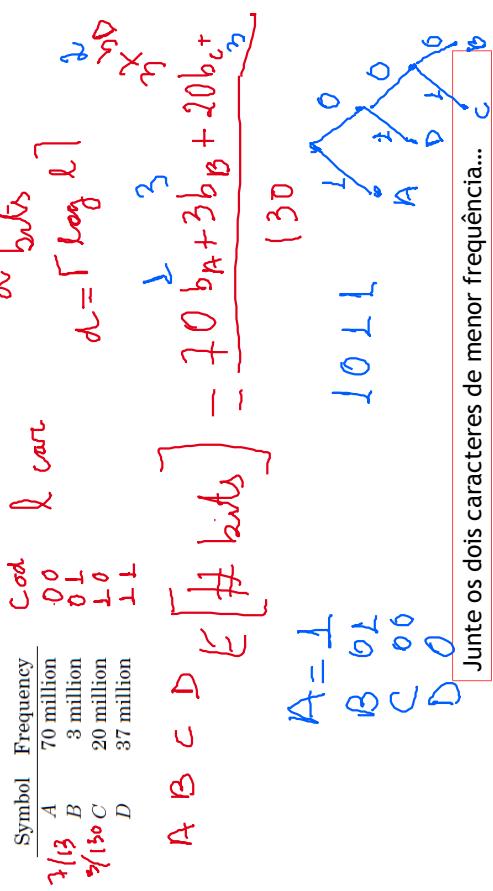
```

}]

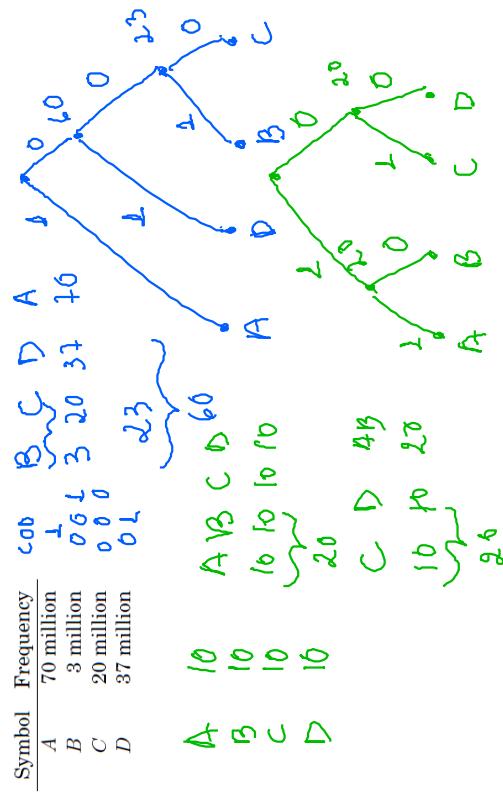
8 Gulosos

8.1 Algoritmo de Huffman + Cláusulas de Horn

Código de Huffman



Código de Huffman



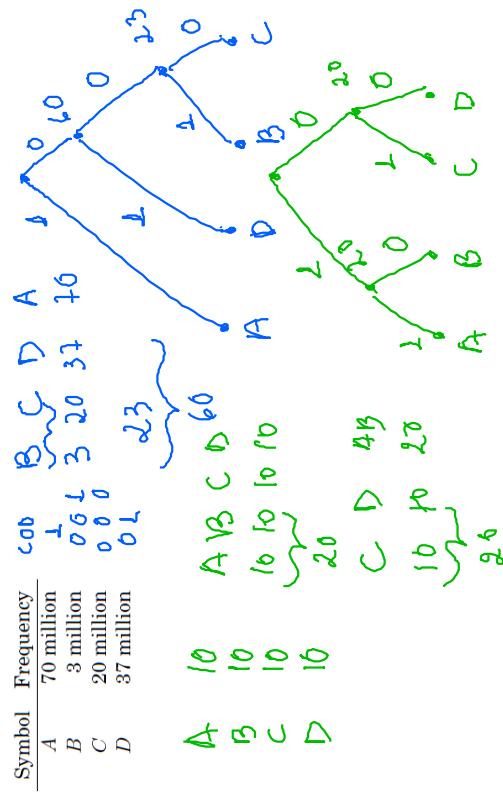
4

BCCC41 / 2011-2

decom

departamento
de comunicação
e computação

Código de Huffman



BCCC41 / 2011-2

decom

departamento
de comunicação
e computação

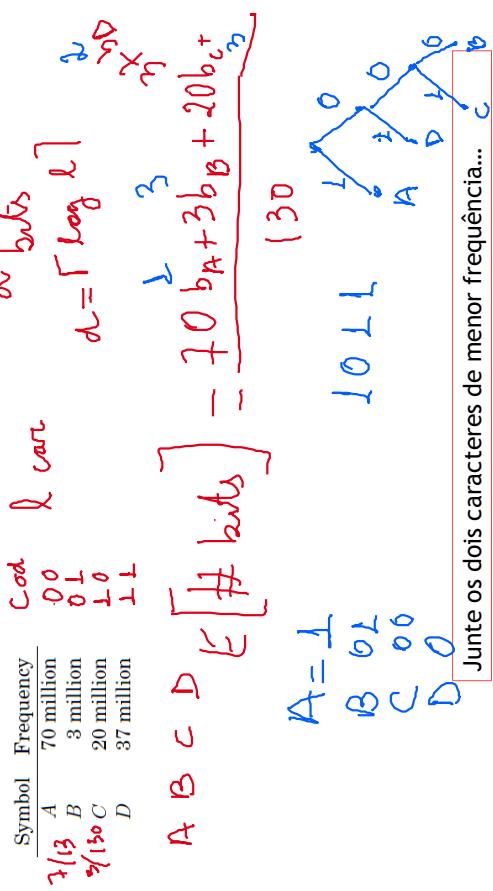
3

BCCC41 / 2011-2

decom

departamento
de comunicação
e computação

Código de Huffman



BCCC41 / 2011-2

decom

departamento
de comunicação
e computação

4

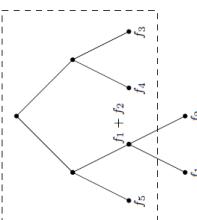
BCCC41 / 2011-2

decom

departamento
de comunicação
e computação

Código de Huffman

Symbol	Frequency
A	70 million
B	3 million
C	20 million
D	37 million



Código de Huffman

- 1952, David Huffman
- Compressão (MP3, por exemplo)
- Propriedades
 - Número variável de bits
 - Não-ambiguidade (árvore binária completa)
 - Diversidade e compressão

BCCC41 / 2011-2

decom

departamento
de comunicação
e computação

5

BCCC41 / 2011-2

decom

departamento
de comunicação
e computação

6

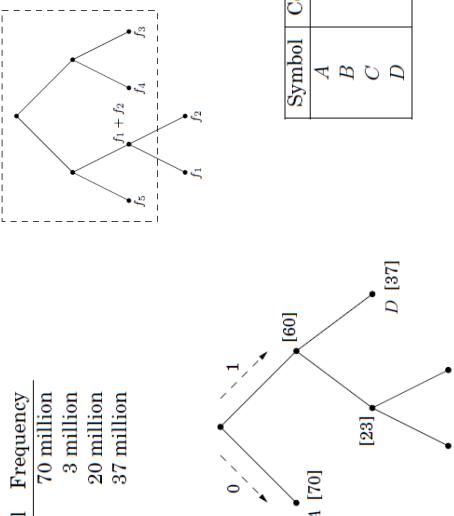
BCCC41 / 2011-2

decom

departamento
de comunicação
e computação

Código de Huffman

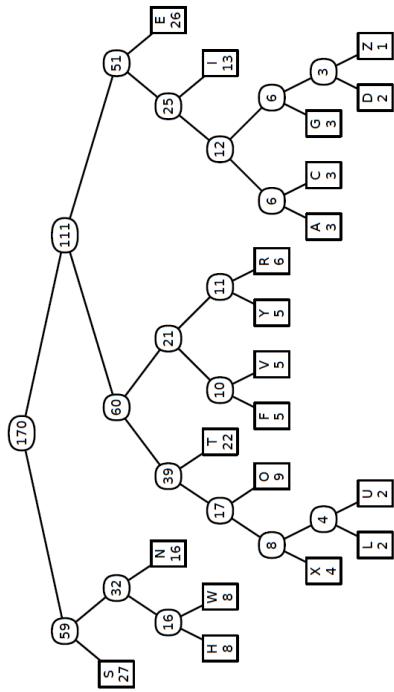
Symbol	Codeword
A	0
B	100
C	101
D	11



Algoritmo Guloso: Huffman Ótimo

procedure Huffman(f)
 Input : An array $f[1 \dots n]$ of frequencies
 Output : An encoding tree with n leaves
 let H be a priority queue of integers, ordered by f
 for $i = 1$ to n : insert(H, i)
 for $k = n+1$ to $2n-1$:
 | $i = \text{deletemin}(H)$, $j = \text{deletemin}(H)$
 | create a node numbered k with children i, j
 $[k] = f[i] + f[j]$
 insert(H, k)

$$\sum_{i=1}^n \log(i) = O(n)$$



This sentence contains three a's, three c's, two d's, twenty-six e's, five f's, three g's, eight h's, thirteen i's, two l's, sixteen n's, nine o's, six r's, twenty-seven s's, twenty-two t's, two u's, five v's, eight w's, four x's, five y's, and only one z.

A	C	D	E	F	G	H	I	L	N	O	R	S	T	U	V	W	X	Y	Z
3	3	2	26	5	3	8	13	2	16	9	6	27	22	2	5	8	4	5	1
A	C	E	F	G	H	I	L	N	O	R	S	T	U	V	W	X	Y	Z	
3	3	26	5	3	8	13	2	16	9	6	27	22	2	5	8	4	5	3	

Huffman: Codifica/Decodifica

$m \leftarrow 1$
 for $i \leftarrow 1$ to k
HUFFMANENCODEONE($A[1..k]$)
HUFFMANENCODEONE(x):
 if $x < 2n - 1$
 HUFFMANENCODEONE($P[x]$)
 if $x = L[P[x]]$
 $B[m] \leftarrow 0$
 else
 $B[m] \leftarrow 1$
 $m \leftarrow m + 1$

```

HUFFMANDECODE(B[1..m]):
    k ← 1
    v ← 2n - 1
    for i ← 1 to m
        if B[i] = 0
            v ← L[v]
        else
            v ← R[v]
    if L[v] = 0
        A[k] ← v
        k ← k + 1
    v ← 2n - 1

```

Cláusulas de Horn

$$\begin{array}{c}
 \text{Definition: } \overline{x} \vee y \equiv x \Rightarrow y \\
 \text{antizwischenste } \overline{x} \Rightarrow y \equiv \neg x \vee y \\
 \text{komplementärer } \overline{x} \Rightarrow y \equiv \neg x \vee \neg y \\
 \text{Zusammenfassung: } \overline{x} \Rightarrow y \equiv \neg x \vee \neg y \\
 \text{Beweis: } \overline{x} \Rightarrow y \equiv \neg x \vee \neg y \\
 \text{Voraussetzung: } x \Rightarrow y \equiv \neg x \vee y \\
 \text{Ziel: } \overline{x} \vee y \equiv x \Rightarrow y
 \end{array}$$

Cláusulas de Horn

$(w \wedge y \wedge z) \Rightarrow x, (x \wedge z) \Rightarrow w, x \Rightarrow y, \neg x \Rightarrow w, (x \wedge y) \Rightarrow w, (\bar{w} \vee \bar{x} \vee \bar{y}), (\bar{z})$

$$\begin{array}{ll} w \wedge y \wedge z \Rightarrow x & 1 \quad (\bar{w} \vee \bar{y} \vee \bar{z} \vee x) \wedge \\ \neg x \Rightarrow w & 2 \quad (\bar{x} \vee \bar{z} \vee w) \wedge \\ \neg x \Rightarrow y & 3 \quad (\bar{x} \vee y) \wedge \\ \neg x \Rightarrow \neg y & 4 \quad (\neg x) \wedge \\ \neg x \Rightarrow w & 5 \quad (\bar{x} \vee \bar{y} \vee w) \wedge \\ x \not\models \perp & 6 \quad (\bar{w} \vee \bar{x} \vee \bar{y}) \wedge \\ y \not\models \perp & 7 \\ z \not\models \perp & \end{array}$$

Cláusulas de Horn

- 1951, Alfred Horn
 - É possível deduzir um fato a partir de um conjunto de implicações?
 - Fatos: variáveis lógicas
 - Implicações : com antecedente de conjunto de fatos (no máximo um negado) e um consequente unitário
- Conjunção de cláusulas de disjunção com no máximo uma variável não-negada
 - Subconjunto de problemas de satisfabilidade

Cláusulas de Horn

$(w \wedge y \wedge z) \Rightarrow x, (x \wedge z) \Rightarrow w, x \Rightarrow y, \neg x \Rightarrow w, (x \wedge y) \Rightarrow w, (\bar{w} \vee \bar{x} \vee \bar{y}), (\bar{z})$

$$\begin{array}{ll} w \wedge y \wedge z \Rightarrow x & 1 \quad (\bar{w} \vee \bar{y} \vee \bar{z} \vee x) \wedge \\ \neg x \Rightarrow w & 2 \quad (\bar{x} \vee \bar{z} \vee w) \wedge \\ \neg x \Rightarrow y & 3 \quad (\bar{x} \vee y) \wedge \\ \neg x \Rightarrow \neg y & 4 \quad (\neg x) \wedge \\ \neg x \Rightarrow w & 5 \quad (\bar{x} \vee \bar{y} \vee w) \wedge \\ x \not\models \perp & 6 \quad (\bar{w} \vee \bar{x} \vee \bar{y}) \wedge \\ y \not\models \perp & 7 \\ z \not\models \perp & \end{array}$$

```
Input: a Horn formula
Output: a satisfying assignment, if one exists
set all variables to false
```

```
while there is an implication that is not satisfied:
    set the right-hand variable of the implication to true
```

```
if all pure negative clauses are satisfied: return the assignment
else: return 'formula is not satisfiable'
```

Horn - Algoritmo Guloso Exato

Implementação em tempo linear: Norvig and Russel,
Lógica Proposicional Forward Chaining

9 Math

9.1 Área da Esfera

A área de uma esfera é equivalente a $4 \times \pi \times r^2$.

9.2 Área do Triângulo, dados 3 pontos

```
struct point { double x, y;
    point(double _x, double _y) { x = _x, y = _y; }
    point(){}
5     bool operator < (point other) {
        if (fabs(x - other.x) > EPS)
            return x < other.x;
        return y < other.y;
    }
10    point operator+(const point k){ return point(x + k.x, y + k.y); }

    point operator-(const point k){ return point(x - k.x, y - k.y); }

15    point operator*(const double k){ return point(k * x, k * y); }
};

double dist(point p1, point p2) {

20    return hypot(p1.x - p2.x, p1.y - p2.y); }

struct line { double a, b, c; };

void pointsToLine(point p1, point p2, line *l) {
25    if (p1.x == p2.x) {
        l->a = 1.0;    l->b = 0.0;    l->c = -p1.x;
    } else {
        l->a = -(double)(p1.y - p2.y) / (p1.x - p2.x);
        l->b = 1.0;
30        l->c = -(double)(l->a * p1.x) - (l->b * p1.y);
    } }

bool areParallel(line l1, line l2) {
35    return (fabs(l1.a-l2.a) < EPS) && (fabs(l1.b-l2.b) < EPS); }

bool areSame(line l1, line l2) {
    return areParallel(l1 ,l2) && (fabs(l1.c - l2.c) < EPS); }

40 bool areIntersect(line l1, line l2, point *p) {
    if (areSame(l1, l2)) return false;
    if (areParallel(l1, l2)) return false;
    p->x = (double)(l2.b * l1.c - l1.b * l2.c) /
        (l2.a * l1.b - l1.a * l2.b);
    if (fabs(l1.b) > EPS)
45        p->y = - (l1.a * p->x + l1.c) / l1.b;
    else
        p->y = - (l2.a * p->x + l2.c) / l2.b;
    return true; }
```

```

50 int main()
{
    int n;

    scanf( "%d", &n);

55    for( int i = 0; i < n; i++)
    {
        double Ax, Ay, Bx, By, Cx, Cy;
        scanf( "%lf %lf %lf %lf %lf %lf", &Ax, &Ay, &Bx, &By, &Cx, &Cy);

60        point A(Ax, Ay), B(Bx, By), C(Cx, Cy), D, E, F, P, Q, R;

        D = B + ( (C - B) * (1./3) );
        E = C + ( (A - C) * (1./3) );
        F = A + ( (B - A) * (1./3) );

        line AD, BE, CF;

        pointsToLine(A, D, &AD);
        pointsToLine(B, E, &BE);
        pointsToLine(C, F, &CF);

        areIntersect(AD, BE, &P);
        areIntersect(BE, CF, &Q);
        areIntersect(AD, CF, &R);

        double area, s, pq, pr, qr;

80        pq = dist(P, Q);
        pr = dist(P, R);
        qr = dist(Q, R);

        s = (pq + pr + qr) / 2;
        area = sqrt(s * (s - pq) * (s - pr) * (s - qr));

85        printf( "%d\n", (int)round(area));
    }
    return 0;
}

```

9.3 Algoritmo de Briot-Ruffini

Algoritmo de Briot-Ruffini é um método de resolução de frações polinomiais. Criado por Paolo Ruffini. Esse algoritmo consiste em efetuar a divisão fazendo cálculos apenas com coeficientes e só serve para divisões de um polinômio por um binômio.

As divisões de polinômios por binômios, como por exemplo: $(x - 2)$, $(x + 3/2)$ e $(x + 5)$, surgem em problemas de matemática mais frequentemente do que quaisquer outras divisões de polinômios e desempenham papel importante na pesquisa de zeros de funções e na resolução de equações.

O quociente e o resto da divisão de um polinômio $P(x)$ por um binômio do tipo

$(x-a)$ podem ser obtidos através de um dispositivo prático, conhecido como divisão sintética ou algoritmo de Briot-Ruffini.

Divisão de um polinômio por $x - a$:

Sejam: $P(x) = 2x^3 + 3x^2 - 4$ e $D(x) = x + 1$.

Queremos dividir $P(x)$ por $D(x)$ usando a regra de Ruffini. Primeiro observamos que $D(x)$ não é um binômio da forma $x - a$, mas da forma $x + a$. Então reescrevemos $D(x)$ deste modo:

$$D(x) = x + 1 = x - (-1) \quad (3)$$

Agora aplicamos o algoritmo:

- Transcrevemos os coeficientes e a . Note que, como $P(x)$ não contém um

coeficiente para x , então escrevemos 0:
$$\begin{array}{r} 2 & 3 & 0 & -4 \\ \hline -1 & & & \end{array}$$

- Passe o primeiro coeficiente para baixo:
$$\begin{array}{r} 2 & 3 & 0 & -4 \\ \hline -1 & & & \\ & 2 & & \end{array}$$

- Multiplique-o por a :
$$\begin{array}{r} 2 & 3 & 0 & -4 \\ -1 & & -2 & \\ \hline 2 & & & \end{array}$$

- Some os valores da coluna:
$$\begin{array}{r} 2 & 3 & 0 & -4 \\ -1 & & -2 & \\ \hline 2 & 1 & & \end{array}$$

- Repita os passos 3 e 4 até a última coluna:
$$\begin{array}{r} 2 & 3 & 0 & -4 \\ -1 & & -2 & -1 & 1 \\ \hline 2 & 1 & -1 & -3 \end{array}$$

2, 1 e -1 são os coeficientes, -3 é o resto.

$$Q(x) = 2x^2 + x - 1 ; \quad e \quad r = -3 \quad (4)$$

$$P(x) = (2x^3 + 3x^2 - 4) = D(x)Q(x) + r = (x + 1)(2x^2 + x - 1) - 3 \quad (5)$$

9.4 Algoritmo de Euclides

```

int interactiveEuclides(int a, int b)
{
    int divisor, dividendo, c;
    dividendo = a;
    divisor = b;
}

```

```

10     while(dividendo % divisor != 0)
11     {
12         c = dividendo % divisor;
13         dividendo = divisor;
14         divisor = c;
15     }
16
17     return divisor;
18 }
19
20 int recursiveEuclides(int a, int b)
21 {
22     if(b == 0)
23         return a;
24
25     return recursiveEuclides(b, a%b);
26 }
```

9.5 Ângulo entre duas Retas

Considere duas retas distintas e concorrentes do plano, r e s , ambas oblíquas aos eixos coordenados e não perpendiculares entre si. As duas formam um ângulo entre si, que denominaremos de α . Esse ângulo α é tal que (vide figura 9.5):

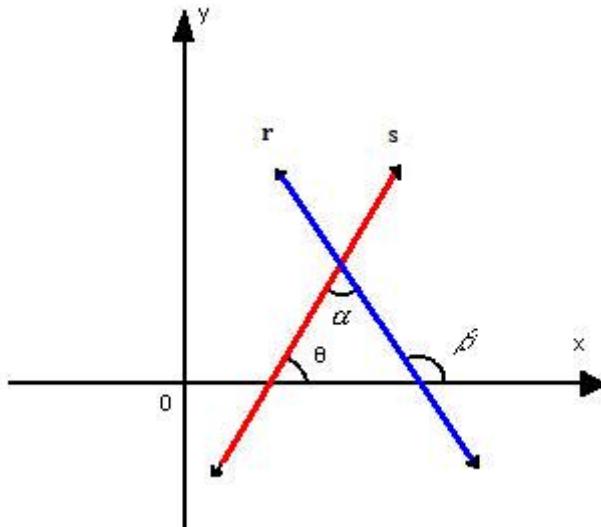


Figura 1: Ângulo entre duas retas

$$\operatorname{tg} \alpha = \frac{m_s - m_r}{1 + m_s \times m_r} \quad (6)$$

Onde m_s e m_r são os coeficientes angulares das retas s e r , respectivamente. Se ocorrer de uma das retas ser vertical e a outra oblíqua, o ângulo α formado entre elas é tal que

$$\operatorname{tg} \alpha = |1/m_r| \quad (7)$$

9.6 Divisores de um número Natural

O número 200 decomposto possui dois fatores primos. Um com expoente 3 (2^3) e outro com expoente 2 (5^2). A multiplicação desses expoentes adicionados em uma unidade cada um deles, irá nos fornecer a informação procurada: $(3+1) \times (2+1) == 12$

9.7 Exponenciação e Logaritmo

9.7.1 Regras da função exponencial

$$a^x (\forall x \in R, \forall a > 0)$$

- $a^{x+y} = a^x a^y$
- $(ab)^x = a^x b^x$
- $a^{x-y} = \frac{a^x}{a^y}$
- $(\frac{a}{b})^x = \frac{a^x}{b^x}$
- $(a^x)^y = a^{xy}$
- $a^{-x} = \frac{1}{a^x}$
- $a^0 = 1$

A função inversa da exponencial é a função logarítmica,
 $y = a^x \Leftrightarrow x = \log_a(y)$, para $a \neq 1$.

9.7.2 Propriedades da função logarítmica

$$\log_a(x) (\forall x \in R^+, \forall a \in R^+ \setminus \{1\})$$

- $\log_a(xy) = \log_a(x) + \log_a(y)$
- $\log_a(\frac{x}{y}) = \log_a(x) - \log_a(y)$
- $\log_a(x^c) = c \times \log_a(x), \forall x \in R$
- Fórmula da mudança de base:
 - $\log_b x = \log_a x \times \log_b a$
 - $\log_b x = \frac{\log_a x}{\log_a b}$
 - $\log_b x = \frac{\log x}{\log b}$
 - $\log_b x = \frac{\ln x}{\ln b}$

9.8 Geometry

9.8.1 Estruturas de Pontos, Linhas e Círculos

```
typedef double dat;

***** Ponto (2D) *****/
struct p2d
5 {
    dat x, y;
    ***** Operators *****/
    p2d operator+(p2d p) { return p2d(x + p.x, y + p.y); }
    p2d operator-(p2d p) { return p2d(x - p.x, y - p.y); }
10   p2d operator*(dat k) { return p2d(k * x, k * y); }
    p2d operator/(dat k){ return p2d(x / k, y / k); }
    dat operator*(p2d p){ return (x * p.y - y * p.x); }
    dat operator^(p2d p){ return (x * p.x + y * p.y); }
    bool operator==(p2d p){ return (x == p.x && y == p.y); }
15   bool operator<(p2d p) const { return (x < p.x || (x == p.x && y <
        p.y)); }
    bool operator>(p2d p) const { return (x > p.x || (x == p.x && y >
        p.y)); }
    ***** Funcoes *****/
    bool point_on_line(p2d p0, p2d p1){ return ((p1 - p0) * (*this -
        - p0)) == 0; }
20   dat sqr(dat x){ return (x * x); }
    double dist(p2d p){ return hypot(x - p.x, y - p.y); }
    double dist2(p2d p){ return sqr(x - p.x) + sqr(y - p.y); }
    double mod(){ return sqrt(x * x + y * y); }
    double mod2(){ return (x * x + y * y); }
    double ang(p2d &p){ return acos((*this ^ p)/((*this).mod() * p.
        mod())); }
25   double point_line_distance(p2d &p0, p2d &p1)
    {
        p2d v1 = *this - p0, v2 = p1 - p0;
        double u = (v1 ^ v2)/v2.mod2();
        p2d p = p0 + v2 * u;
        return (*this).dist(p);
    }
    double point_line_segment_distance(p2d &p0, p2d &p1)
30   {
        p2d v1 = *this - p0, v2 = p1 - p0;
        double u = (v1 ^ v2)/v2.mod2();
        if(u < 0) return (*this).dist(p0);
        if(u > 1) return (*this).dist(p1);
        return (*this).dist(p0 + v2 * u);
    }
40   p2d(dat _x = 0, dat _y = 0): x(_x), y(_y) {};
};

***** Linha (2D) *****/
struct l2d
45 {
    p2d s, v;
    ***** Construtores *****/
    l2d(){}
    l2d(p2d &p0, p2d &p1)
50   {
```

```

        s = p2d(p0.x, p0.y);
        v = p1 - p0;
    }
l2d(p2d p, double m) { s = p, v = p2d(cos(m), sin(m)); }
l2d(double a, double b, double c)
{
    if(fabs(a) < EPS) v = p2d(1, -a/b), s = p2d(0, c/b);
    else v = p2d(-b/a, 1), s = p2d(c/a, 0);
}
/****** Funcoes *****/
int line_line_intersection(l2d &line2, p2d &pi)
{
    p2d p1 = p2d(s.x, s.y), p2 = s + v;
    p2d p3 = p2d(line2.s.x, line2.s.y), p4 = line2.s + line2.v;
    dat den = (p1.x - p2.x) * (p3.y - p4.y) - (p3.x - p4.x) * (p1.y
        - p2.y);
    if(fabs(den) < EPS) return -1;
    dat x = ((p1 * p2) * (p3.x - p4.x) - (p3 * p4) * (p1.x - p2.x))
        ;
    dat y = ((p1 * p2) * (p3.y - p4.y) - (p3 * p4) * (p1.y - p2.y))
        ;
    pi = p2d(x / den, y / den);
    return 0;
}
bool same(l2d l)
{
    return fabs(v * l.v) < EPS && fabs((l.s - s) * v) < EPS;
}
bool point_on_line(p2d p){ return (v * (p - s)) == 0; }
double slope(){ return v.y/(double)v.x; }
};

/****** CÁculo (2D) *****/
struct c2d
{
    p2d center;
    double r;
    c2d(){}
    c2d(p2d p0, p2d p1, double _r)
    {
        p2d mid = (p1 + p0)/(double)2, v = (p1 - p0);
        double q = p0.dist(p1), d = sqrt(r * r - q * q / (double) 4);
        v = v / v.mod();
        swap(v.x, v.y);
        v.x = -v.x;
        center = mid + v * d;
        r = _r;
    };
    c2d(p2d p0, p2d p1, p2d p2){}
};

/****** Funcoes *****/
int l2d_l2d_intersection(l2d l1, l2d l2, p2d &pi)
{
    p2d p1 = p2d(l1.s.x, l1.s.y), p2 = l1.s + l1.v;
    p2d p3 = p2d(l2.s.x, l2.s.y), p4 = l2.s + l2.v;

```

```

105    dat den = (p1.x - p2.x) * (p3.y - p4.y) - (p3.x - p4.x) * (p1.y -
     p2.y);
    if(fabs(den) < EPS) return 0;
    dat x = ((p1 * p2) * (p3.x - p4.x) - (p3 * p4) * (p1.x - p2.x));
    dat y = ((p1 * p2) * (p3.y - p4.y) - (p3 * p4) * (p1.y - p2.y));
    pi = p2d(x, y)/den;
    return 1;
}

p2d p[128];
int x[128], y[128];
115
dat sqr(dat x)
{
    return (x * x);
}
120
double DEG_to_RAD(double d) { return d * PI / 180.0; }

double RAD_to_DEG(double r) { return r * 180.0 / PI; }

125 int inCircle(point_i p, point_i c, int r) {
    int dx = p.x - c.x, dy = p.y - c.y;
    int Euc = dx * dx + dy * dy, rSq = r * r;
    return Euc < rSq ? 0 : Euc == rSq ? 1 : 2; }

130 bool circle2PtsRad(point p1, point p2, double r, point *c) {
    double d2 = (p1.x - p2.x) * (p1.x - p2.x) +
               (p1.y - p2.y) * (p1.y - p2.y);
    double det = r * r / d2 - 0.25;
    if (det < 0.0) return false;
    135 double h = sqrt(det);
    c->x = (p1.x + p2.x) * 0.5 + (p1.y - p2.y) * h;
    c->y = (p1.y + p2.y) * 0.5 + (p2.x - p1.x) * h;
    return true; }

```

9.8.2 Bola na Caixa

```

int main()
{
    long long diam, val;
5     scanf("%lld", &diam);

    bool problem = false;

    scanf("%lld", &val);
10    if(val < diam)
        problem = true;

    scanf("%lld", &val);
15    if(val < diam)
        problem = true;

    scanf("%lld", &val);
    if(val < diam)
        problem = true;

```

```

20   if(problem)
      printf( "N|n");
    else
      printf( "S|n");
25
  return 0;
}

```

9.8.3 Caixa na Bola

```

int main()
{
    double larg, alt, prof, diag, rMin, r;

5   scanf( "%lf%lf%lf%lf", &larg, &alt, &prof, &r);

    diag = sqrt(larg*larg + prof*prof);

    rMin = sqrt((diag/2)*(diag/2) + (alt/2)*(alt/2));
10
    if(r >= rMin)
        printf( "S|n");
    else
        printf( "N|n");
15
    return 0;
}

```

9.8.4 Calcular $x^y \% n$

```

class Main /* LA 4104 - MODEX */
{
    public static void main(String[] args)
    {
5       Scanner sc = new Scanner(System.in);
        int c = sc.nextInt();
        while (c-- > 0)
        {
            BigInteger x = BigInteger.valueOf(sc.nextInt());
            BigInteger y = BigInteger.valueOf(sc.nextInt());
            BigInteger n = BigInteger.valueOf(sc.nextInt());
            System.out.println(x.modPow(y, n));
        }
    }
15
}

```

9.8.5 Circles Sample – Competitive Programming

```

int main()
{
    point_i pt(2, 2);
    int r = 7;
5     point_i inside(8, 2);

```

```

    printf("%d\n", inCircle(inside, pt, r));
    point_i border(9, 2);
    printf("%d\n", inCircle(border, pt, r));
    point_i outside(10, 2);
    printf("%d\n", inCircle(outside, pt, r));

    double d = 2 * r;
    printf("Diameter = %.2lf\n", d);
    double c = PI * d;
    printf("Circumference / Perimeter = %.2lf\n", c);
    double A = PI * r * r;
    printf("Area of circle = %.2lf\n", A);

    printf("Length of arc (central angle = 30 degrees) = %.2lf\n",
           30.0 / 360.0 * c);
    printf("Length of chord (central angle = 30 degrees) = %.2lf\n",
           sqrt((2 * r * r) * (1 - cos(DEG_to_RAD(30.0)))));
    printf("Area of sector (central angle = 30 degrees) = %.2lf\n",
           30.0 / 360.0 * A);

    point p1(0.0, 1.0);
    point p2(0.0, -1.0);
    point ans(0.0, 0.0);
    circle2PtsRad(p1, p2, 2.0, &ans);
    printf("One of the center is (%.2lf, %.2lf)\n", ans.x, ans.y);
    circle2PtsRad(p2, p1, 2.0, &ans);
    printf("The other center is (%.2lf, %.2lf)\n", ans.x, ans.y);

    return 0;
}

```

9.8.6 Cycle-Finding - Pseudo-Random Numbers

```

/* Pseudo-Random Numbers */

typedef pair<int, int> ii;

int caseNo = 1, Z, I, M, L;

int f(int x) { return (Z * x + I) % M; }

ii floydCycleFinding(int x0)
{
    int tortoise = f(x0), hare = f(f(x0));
    while (tortoise != hare) { tortoise = f(tortoise); hare = f(f(
        hare)); }
    int mu = 0; hare = x0;
    while (tortoise != hare) { tortoise = f(tortoise); hare = f(hare)
        ; mu++; }
    int lambda = 1; hare = f(tortoise);
    while (tortoise != hare) { hare = f(hare); lambda++; }
    return ii(mu, lambda);
}

int main()
{
    while (scanf("%d %d %d %d", &Z, &I, &M, &L), (Z || I || M || L))

```

```

25     {
        ii result = floydCycleFinding(L);
        printf( "Case %d: %d\n", caseNo++, result.second);
    }
    return 0;
}

```

9.8.7 Distância entre dois pontos (Esfera / 3D)

Minha Solução:

```

double calc_angle_3d(double x0, double y0, double x1, double y1)
{
    double dx = fabs(x0 - x1) * PI / 180.;
    while(comp(dx, 2. * PI) == 1)
    {
        dx -= PI * 2.;
    }

    if(comp(dx, PI) == 1)
        dx = 2. * PI - dx;

    y0 *= PI / 180.;
    y1 *= PI / 180.;

    return acos(cos(y0)*cos(y1)*cos(dx) + sin(y0)*sin(y1));
}

int main()
{
    int n;
    p2d A, B;
    double r = 6371009.;

    cin >> n;
    for(int i = 0; i < n; i++)
    {
        cin >> A.y >> A.x >> B.y >> B.x;

        double d1, d2, ang = calc_angle_3d(A.x, A.y, B.x, B.y);
30
        d1 = ang * r;
        d2 = r * pow(2. - 2*cos(ang), 0.5);

        printf("%lf\n", round(fabs(d1-d2)));
    }

    return 0;
}

```

Competitive Programming:

```

/* Tunnelling the Earth */

#define PI acos(-1.0)

```

```

5 #define EARTH_RAD (6371009)

double gcDistance(double pLat, double pLong,
                  double qLat, double qLong, double radius)
{
10    pLat *= PI / 180; pLong *= PI / 180;
    qLat *= PI / 180; qLong *= PI / 180;
    return radius * acos(cos(pLat)*cos(pLong)*cos(qLat)*cos(qLong) +
                          cos(pLat)*sin(pLong)*cos(qLat)*sin(qLong) +
                          sin(pLat)*sin(qLat));
15}

double EuclidianDistance(double pLat, double pLong,
                         double qLat, double qLong, double radius)
{
20    double phi1 = (90 - pLat) * PI / 180;
    double theta1 = (360 - pLong) * PI / 180;
    double x1 = radius * sin(phi1) * cos(theta1);
    double y1 = radius * sin(phi1) * sin(theta1);
    double z1 = radius * cos(phi1);

25    double phi2 = (90 - qLat) * PI / 180;
    double theta2 = (360 - qLong) * PI / 180;
    double x2 = radius * sin(phi2) * cos(theta2);
    double y2 = radius * sin(phi2) * sin(theta2);
    double z2 = radius * cos(phi2);

30    double dx = x1 - x2, dy = y1 - y2, dz = z1 - z2;
    return sqrt(dx * dx + dy * dy + dz * dz);
}

35 int main()
{
40    int TC;
    double lat1, lon1, lat2, lon2;
    scanf("%d", &TC);
    while (TC--)
    {
45        scanf("%lf %lf %lf %lf", &lat1, &lon1, &lat2, &lon2);
        printf("%.0lf\n", gcDistance(lat1, lon1, lat2, lon2, EARTH_RAD) -
               EuclidianDistance(lat1, lon1, lat2, lon2, EARTH_RAD));
    }

50    return 0;
}

```

9.8.8 Raios do Círculo inscrito e circunscrito em um Triângulo

```

int main()
{
    double l1, l2, l3, sp, triangleArea, rMin, rMax;

5     while (scanf("%lf%lf%lf", &l1, &l2, &l3) == 3)

```

```

    {
        sp = (l1 + l2 + l3) / 2;
        triangleArea = sqrt(sp * (sp - l1) * (sp - l2) * (sp - l3));
10       rMin = triangleArea / sp;
        rMax = (l1 * l2 * l3) / (4 * triangleArea);

        printf( "% .04lf % .04lf % .04lf\n", PI * rMax * rMax -
            triangleArea, triangleArea - PI * rMin * rMin, PI * rMin *
            rMin);

15    }

    return 0;
}

```

9.8.9 Triangle Area

```

A_triangle = sqrt(p*(p-a)*(p-b)*(p-c)), p = (a+b+c) / 2.;
A_triangle = (b * c * sen(Á)) / 2., Á = angulo oposto a a;

```

9.8.10 Intersecting Circles

```

struct point
{
    double x, y;
    point(double _x, double _y)
5    {
        x = _x, y = _y;
    }
};

10   double dist(point p1, point p2)
{
    return hypot(p1.x - p2.x, p1.y - p2.y);
}

15   int main()
{
    double x1, y1, r1, x2, y2, r2;

    while (cin >> x1 >> y1 >> r1 >> x2 >> y2 >> r2)
20    {
        cout << fixed << setprecision(3);

        point p1(x1, y1), p2(x2, y2);
        double d = dist(p1, p2);

25        if (d - (r1 + r2) > EPS)
            cout << "NO INTERSECTION" << endl;

        else if (d - fabs(r1 - r2) < -EPS)
            cout << "NO INTERSECTION" << endl;

30        else if (d == 0 && r1 == r2)
        {

```

```

35         if(r1 == 0)
            cout<<setprecision(3)<<"(" <<x1<< ", " <<y1<< ")" <<endl;
        else
            cout<<"THE CIRCLES ARE THE SAME"<<endl;
    }

40 else
{
    double dis2, ha, raizx, raizy;
    double xtemp, ytemp;

45     dis2 = ((r1 * r1) - (r2 * r2) + (d * d))/(2.0 * d);
    xtemp = x1 + ((x2 - x1) * dis2/d);
    ytemp = y1 + ((y2 - y1) * dis2/d);

    ha = sqrt(fabs((r1 * r1) - (dis2 * dis2)));
50
    raizx = (-(y2 - y1)) * (ha/d);
    raizy = (x2 - x1) * (ha/d);

    /* Os 2 pontos */
55    double x1res, y1res, x2res, y2res;

    x1res = xtemp + raizx;
    y1res = ytemp + raizy;
    x2res = xtemp - raizx;
    y2res = ytemp - raizy;

60
    if(fabs(x1res) < EPS)
        x1res = 0;
    if(fabs(y1res) < EPS)
        y1res = 0;
    if(fabs(x2res) < EPS)
        x2res = 0;
    if(fabs(y2res) < EPS)
        y2res = 0;

65
70    if(fabs(ha) < EPS)
        cout<<setprecision(3)<<"(" <<x1res<< ", " <<y1res<< ")" "
            <<endl;
    else
{
75        if(x1res < x2res || (x1res == x2res && y1res <=
            y2res))
            cout<<setprecision(3)<<"(" <<x1res<< ", " <<y1res<<
                ")" "(" <<x2res<< ", " <<y2res<< ")" <<endl;
        else
            cout<<setprecision(3)<<"(" <<x2res<< ", " <<y2res<<
                ")" "(" <<x1res<< ", " <<y1res<< ")" <<endl;
    }
80
}
return 0;
}

```

9.8.11 Intersecting Lines

```

int main()
{
    int t; scanf( "%d", &t);
    p2d p, pi, q, r, s;
    puts( "INTERSECTING LINES OUTPUT");
    FOR(cc, t)
    {
        scanf( "%lf %lf %lf %lf %lf %lf %lf", &p.x, &p.y, &q.x, &q.y
              , &r.x, &r.y, &s.x, &s.y);
        l2d l1 = l2d(p, q), l2 = l2d(r, s);
        int k = l2d_l2d_intersection(l1, l2, pi);
        if(k) printf( "POINT %.2lf %.2lf\n", pi.x, pi.y);
        else
        {
            if(l1.same(l2)) puts( "LINE");
            else puts( "NONE");
        }
    }
    puts( "END OF OUTPUT");
    return 0;
}

```

9.8.12 Intersecting Segments

```

bool seg_intersect(point p, point q, point r, point s)
{
    point A = q - p;
    point B = s - r;
    point C = r - p;
    point D = s - q;

    int a = cmp(A % C) + 2 * cmp(A % D);
    int b = cmp(B % C) + 2 * cmp(B % D);

    if (a == 3 || a == -3 || b == 3 || b == -3) return false;
    if (a || b || p == r || p == s || q == r || q == s) return
        true;

    int t = (p < r) + (p < s) + (q < r) + (q < s);
    return t != 0 && t != 4;
}

```

9.8.13 Teorema de Pitágoras Completo / Lei dos Cossenos

$$c^2 = a^2 + b^2 - 2 * a * b * \cos\theta$$

9.8.14 Polígonos

```

#define EPS 1e-9
#define PI acos(-1.0)

double DEG_to_RAD(double d) { return d * PI / 180.0; }

5 double RAD_to_DEG(double r) { return r * 180.0 / PI; }

```

```

struct point
{
    double x, y;
    point(double _x, double _y) { x = _x, y = _y; }

double dist(point p1, point p2)
{
    return hypot(p1.x - p2.x, p1.y - p2.y);
}

double perimeter(vector<point> P)
{
    double result = 0.0;
    for (int i = 0; i < (int)P.size(); i++)
        result += dist(P[i], P[(i + 1) % P.size()]);
    return result;
}

double area(vector<point> P)
{
    double result = 0.0, x1, y1, x2, y2;
    for (int i = 0; i < (int)P.size(); i++)
    {
        x1 = P[i].x; x2 = P[(i + 1) % P.size()].x;
        y1 = P[i].y; y2 = P[(i + 1) % P.size()].y;
        result += (x1 * y2 - x2 * y1);
    }
    return fabs(result) / 2.0;
}

double cross(point p, point q, point r)
{
    return (r.x - q.x) * (p.y - q.y) - (r.y - q.y) * (p.x - q.x);
}

bool collinear(point p, point q, point r)
{
    return fabs(cross(p, q, r)) < EPS;
}

bool ccw(point p, point q, point r)
{
    return cross(p, q, r) > 0;
}

bool isConvex(vector<point> P)
{
    int sz = (int)P.size();
    if (sz < 3)
        return false;
    bool isLeft = ccw(P[0], P[1], P[2]);
    for (int i = 1; i < sz; i++)
        if (ccw(P[i], P[(i + 1) % sz], P[(i + 2) % sz]) != isLeft)
            return false;
    return true;
}

```

```

65   double angle(point a, point b, point c)
{
    double ux = b.x - a.x, uy = b.y - a.y;
    double vx = c.x - a.x, vy = c.y - a.y;
    return acos((ux*vx + uy*vy) /
                 sqrt((ux*ux + uy*uy) * (vx*vx + vy*vy)));
}

70   bool inPolygon(point p, vector<point> P)
{
    if ((int)P.size() == 0) return false;
    double sum = 0;
    for (int i = 0; i < (int)P.size() - 1; i++)
    {
        if (cross(p, P[i], P[i + 1]) < 0)
            sum -= angle(p, P[i], P[i + 1]);
        else sum += angle(p, P[i], P[i + 1]);
    }
    return (fabs(sum - 2*PI) < EPS || fabs(sum + 2*PI) < EPS);
}

75   point lineIntersectSeg(point p, point q, point A, point B)
{
    double a = B.y - A.y;
    double b = A.x - B.x;
    double c = B.x * A.y - A.x * B.y;
    double u = fabs(a * p.x + b * p.y + c);
    double v = fabs(a * q.x + b * q.y + c);
    return point((p.x * v + q.x * u) / (u + v),
                 (p.y * v + q.y * u) / (u + v));
}

80   vector<point> cutPolygon(point a, point b, vector<point> Q)
{
    vector<point> P;
    for (int i = 0; i < (int)Q.size(); i++)
    {
        double left1 = cross(a, b, Q[i]), left2 = 0.0;
        if (i != (int)Q.size() - 1) left2 = cross(a, b, Q[i + 1]);
        if (left1 > -EPS) P.push_back(Q[i]);
        if (left1 * left2 < -EPS)
            P.push_back(lineIntersectSeg(Q[i], Q[i + 1], a, b));
    }
    if (P.empty()) return P;
    if (fabs(P.back().x - P.front().x) > EPS ||
        fabs(P.back().y - P.front().y) > EPS)
        P.push_back(P.front());
    return P;
}

85   point pivot(0, 0);
90   bool angle_cmp(point a, point b)
{
    if (collinear(pivot, a, b))
        return dist(pivot, a) < dist(pivot, b);
    double d1x = a.x - pivot.x, d1y = a.y - pivot.y;
    double d2x = b.x - pivot.x, d2y = b.y - pivot.y;

```

```

        return (atan2(d1y, d1x) - atan2(d2y, d2x)) < 0;
    }

125   vector<point> CH(vector<point> P)
    {
        int i, N = (int)P.size();
        if (N <= 3) return P;

130       int P0 = 0;
        for (i = 1; i < N; i++)
            if (P[i].y < P[P0].y ||
                (P[i].y == P[P0].y && P[i].x > P[P0].x))
                P0 = i;
        point temp = P[0]; P[0] = P[P0]; P[P0] = temp;

135       pivot = P[0];
        sort(++P.begin(), P.end(), angle_cmp);

140       point prev(0, 0), now(0, 0);
        stack<point> S; S.push(P[N - 1]); S.push(P[0]);
        i = 1;
        while (i < N)
        {
            now = S.top();
            S.pop(); prev = S.top(); S.push(now);
            if (ccw(prev, now, P[i])) S.push(P[i++]);
            else S.pop();
        }

150       vector<point> ConvexHull;
        while (!S.empty()) { ConvexHull.push_back(S.top()); S.pop(); }
        return ConvexHull;
    }

155   int main()
    {
        vector<point> P;
        P.push_back(point(1, 1));
        P.push_back(point(3, 3));
        P.push_back(point(9, 1));
        P.push_back(point(12, 4));
        P.push_back(point(9, 7));
        P.push_back(point(1, 7));
        P.push_back(P[0]);

160       printf("Perimeter of polygon = %.2lf\n", perimeter(P));
        printf("Area of polygon = %.2lf\n", area(P));
        printf("Is convex = %d\n", isConvex(P));

165       point P6(3, 2);
        printf("Point P6 is inside this polygon = %d\n", inPolygon(P6, P));
        );
        point P7(3, 4);
        printf("Point P7 is inside this polygon = %d\n", inPolygon(P7, P));
        );
    }

```

```

180     P = cutPolygon(P[2], P[4], P);
printf("Perimeter of polygon = %.2lf\n", perimeter(P));
printf("Area of polygon = %.2lf\n", area(P));

P = CH(P);
185 printf("Perimeter of polygon = %.2lf\n", perimeter(P));
printf("Area of polygon = %.2lf\n", area(P));
printf("Is convex = %d\n", isConvex(P));
printf("Point P7 is inside this polygon = %d\n", inPolygon(P7, P)
    );
}

190 return 0;
}

```

9.8.15 PI

double PI == acos(-1.);

9.8.16 Polygon in a circle

```

int main()
{
    double r, maxd;
    for(int n, maxd2; scanf("%d", &n) && n; )
5    {
        maxd2 = 0;
        for(int i = 0; i < n; ++i)
        {
            scanf("%d %d", &x[i], &y[i]);
10       p[i] = p2d(x[i], y[i]);
        }
        scanf("%lf", &r);
        maxd = sqrt(maxd2);
        bool ok = 0;
15       for(int i = 0; !ok && i < n; ++i)
        {
            for(int j = 0; !ok && j < n; ++j)
            {
                if(i == j) continue;
20                c2d circle = c2d(p[i], p[j], r);
                ok = 1;
                for(int k = 0; ok && k < n; ++k)
                ok &= circle.center.dist(p[k]) <= r;
            }
        }
25       if(ok) puts("The polygon can be packed in the circle.");
        else puts("There is no way of packing that polygon.");
    }
    return 0;
30 }

```

9.8.17 Triangles

```

#define INF 1e9
#define EPS 1e-9

```

```

#define PI acos(-1.0)

5   double DEG_to_RAD(double d) { return d * PI / 180.0; }

    double RAD_to_DEG(double r) { return r * 180.0 / PI; }

10  struct point_i
{
    int x, y;
    point_i(int _x, int _y) { x = _x, y = _y; }
};

15  struct point
{
    double x, y;
    point(double _x, double _y) { x = _x, y = _y; }
};

20  double dist(point p1, point p2)
{
    return hypot(p1.x - p2.x, p1.y - p2.y);
}

25  double perimeter(double ab, double bc, double ca)
{
    return ab + bc + ca;
}

30  double perimeter(point a, point b, point c)
{
    return dist(a, b) + dist(b, c) + dist(c, a);
}

35  double area(double ab, double bc, double ca)
{
    double s = 0.5 * perimeter(ab, bc, ca);
    return sqrt(s) * sqrt(s - ab) * sqrt(s - bc) * sqrt(s - ca);
}

40

45  double area(point a, point b, point c)
{
    return area(dist(a, b), dist(b, c), dist(c, a));
}

double rInCircle(double ab, double bc, double ca)
{
    return area(ab, bc, ca) / (0.5 * perimeter(ab, bc, ca));
}

50

55  double rInCircle(point a, point b, point c)
{
    return rInCircle(dist(a, b), dist(b, c), dist(c, a));
}

60  double rCircumCircle(double ab, double bc, double ca)
{
    return ab * bc * ca / (4.0 * area(ab, bc, ca));
}

```

```

double rCircumCircle(point a, point b, point c)
{
    return rCircumCircle(dist(a, b), dist(b, c), dist(c, a));
}

bool canFormTriangle(double a, double b, double c)
{
    return (a + b > c) && (a + c > b) && (b + c > a);
}

int circle(point p1, point p2, point p3, point *ctr, double *r)
{
    double a, b, c, d, e, f, g;
    a = p2.x - p1.x;
    b = p2.y - p1.y;
    c = p3.x - p1.x;
    d = p3.y - p1.y;
    e = a * (p1.x + p2.x) + b * (p1.y + p2.y);
    f = c * (p1.x + p3.x) + d * (p1.y + p3.y);
    g = 2.0 * (a * (p3.y - p2.y) - b * (p3.x - p2.x));
    if (fabs(g) < EPS)
        return 0;

    ctr->x = (d*e - b*f) / g;
    ctr->y = (a*f - c*e) / g;
    *r = sqrt((p1.x-ctr->x) * (p1.x-ctr->x) + (p1.y-ctr->y) * (p1.y-
        ctr->y));
    return 1;
}

int inCircle2(point p, point q, point r, point s)
{
    return 0;
}

int main()
{
    double base = 4.0, h = 3.0;
    double A = 0.5 * base * h;
    printf("Area = %.2lf\n", A);

    point a(0.0, 0.0);
    point b(4.0, 0.0);
    point c(4.0, 3.0);

    double p = perimeter(a, b, c);
    double s = 0.5 * p;
    A = area(a, b, c);
    printf("Area = %.2lf\n", A);

    printf("R1 (radius of inner circle) = %.2lf\n", rInCircle(a, b, c));
    printf("R2 (radius of outer circle) = %.2lf\n", rCircumCircle(a,
        b, c));

    double ab = dist(a, b);
    double bc = dist(b, c);
}

```

```

double ca = dist(c, a);
double alpha = RAD_to_DEGacos((ca * ca + ab * ab - bc * bc) /
(2.0 * ca * ab));
printf("alpha = %.2lf\n", alpha);
double beta = RAD_to_DEGacos((ab * ab + bc * bc - ca * ca) /
(2.0 * ab * bc));
printf("beta = %.2lf\n", beta);
double gamma = RAD_to_DEGacos((bc * bc + ca * ca - ab * ab) /
(2.0 * bc * ca));
printf("gamma = %.2lf\n", gamma);

printf("%.2lf == %.2lf == %.2lf\n", bc / sin(DEG_to_RAD(alpha)),
ca / sin(DEG_to_RAD(beta)), ab / sin(DEG_to_RAD(gamma)));

125 printf("%.2lf^2 == %.2lf^2 + %.2lf^2\n", ca, ab, bc);

printf("(%d, %d, %d) => can form triangle? %d\n", 3, 4, 5,
canFormTriangle(3, 4, 5));
printf("(%d, %d, %d) => can form triangle? %d\n", 3, 4, 7,
canFormTriangle(3, 4, 7));
130 printf("(%d, %d, %d) => can form triangle? %d\n", 3, 4, 8,
canFormTriangle(3, 4, 8));

return 0;
}

```

9.8.18 Triângulo Acutângulo, Retângulo ou Obtusângulo

```

a = maior lado
b e c = lados menores

Retangulo <-> a^2 = b^2 + c^2;
5 Acutangulo <-> a^2 < b^2 + c^2;
Obtusangulo <-> a^2 > b^2 + c^2;

Advindo da Lei dos Cossenos:
a^2 = b^2 + c^2 - 2*(cos A)*b*c;

```

9.9 Infix to Post Suffix Equation

```

/*
* File: main.cpp
* Author: ubuntulap
*
5 * Created on February 28, 2008, 6:16 PM
*/

```

```

#include <stdlib.h>
#include <stdio.h>
10 #include <string.h>
#include <math.h>
#include <limits.h>
#include <assert.h>
#include <cctype>
15 #include <iomanip>

```

```

#include <iostream>
#include <string>
#include <algorithm>
#include <utility>
20 #include <iterator>
#include <numeric>
#include <sstream>
#include <fstream>
#include <bitset>
25 #include <map>
#include <vector>
#include <set>
#include <stack>
#include <queue>
30 #include <deque>
#include <list>

#define SWAP(a,b) ( (a)=(b), (b)=(a), (a)=(b) )
#define MIN(a,b) ( (a)<(b) ? (a) : (b) )
35 #define MAX(a,b) ( (a)>(b) ? (a) : (b) )
#define ABS(a) ( (a)<(0) ? (-1)*(a) : (a) )
#define ROUND(a) ( (a)>=(0) ? ((int)((double)(a) + 0.5)) : ((int)((
    double)(a) - 0.5)) )

#define PI acos(-1.0)
40
#define FI(a) fastint(&a)
#define PL printf(" | n")

#define MAXSTRSZ 100000
45 #define MAXNUMBER 200000
#define INFINITO 999999999
#define EPS 1e-9

using namespace std;

50 typedef long long ll;
typedef unsigned long long ull;
typedef pair<int,int> ii;
typedef vector< pair<int,int> > vii;
55 typedef vector<int> vi;

void fastint(register int *n)
{
    register char c;
    register int neg = 0;
60    *n = 0;

    while(c = getc(stdin), c < '0' || c > '9')
        neg |= c == '-';
65
    do
    {
        (*n) = (*n)*10 + (c - '0');
70    } while(c = getc(stdin), c >= '0' && c <= '9');

    (*n) = neg?(*n)*(-1):(*n);
}

```

```

stack <char> s;
75 vector <char> output;

bool checkOperator (char p)
{
    if( p == '+' || p == '-' || p == '*' || p == '/')
80    return true;

    return false;
}

85 int hasPrecedence (char p, char q)
{
    if ( (p == '*' || p == '/') && (q == '+' || q == '-'))
        return 1;

    if ( (p == '+' || p == '-') && (q == '*' || q == '/'))
        return -1;

    return 0;
}
95

void setOperator (char op)
{
    queue <char> temp;

100    if ( !s.empty () && s.top () != '(' && hasPrecedence(op, s.top ())
        ) <= 0 )
    {
        output.push_back (s.top ());
        s.pop ();
    }

105    while ( !s.empty () && s.top () != '(' && hasPrecedence(op, s.top ()
        ) <= 0 )
    {
        temp.push (s.top ());
        s.pop ();
    }

110    s.push (op);

    while ( !temp.empty ())
115    {
        output.push_back (temp.front ());
        temp.pop ();
    }
}

120 int main ()
{

```

int testCase;
125 scanf ("%d", &testCase);
 bool blank = false;
 getchar (); getchar ();

```

130     while ( testCase-- )
131     {
132         char ch [5];
133         string input;
134         output.clear ();
135
136         while ( gets (ch) && strlen (ch) )
137         {
138             if ( ch [0] == '(' )
139             {
140                 if ( input.length() && isdigit (input [input.length () - 1]))
141                     input += 'x';
142                 else if ( isdigit (ch [0]) )
143                 {
144                     if ( input.length() && input [input.length () - 1] == ')')
145                         input += 'x';
146                 }
147
148                 input += ch [0];
149             }
150
151             for ( size_t i = 0; i < input.length (); i++ )
152             {
153                 if ( isdigit (input [i]) )
154                     output.push_back (input [i]);
155                 else if ( checkOperator (input [i]) )
156                 {
157                     setOperator (input [i]);
158
159                     /* if ( !s.empty() && checkOperator (s.top ()) )
160                     {
161                         if ( hasPrecedence (s.top (), input [i]))
162                         {
163                             output.push_back (s.top ());
164                             s.pop ();
165                         }
166                     }
167
168                     s.push (input [i]); */
169             }
170             else if ( input [i] == '(' )
171                 s.push ('(');
172             else if ( input [i] == ')' )
173             {
174                 while ( s.top () != '(' )
175                 {
176                     output.push_back (s.top ());
177                     s.pop ();
178                 }
179             }
180         }
181
182         while ( !s.empty () )
183         {
184             output.push_back (s.top ());

```

```

        s.pop ();
    }

    if ( blank )
        PL;

    blank = true;

    for ( size_t i = 0; i < output.size (); i++ )
    {
        if ( output [i] != 'x' )
            printf ("%c", output [i]);
    }
    PL;
}

return 0;
}

```

9.10 Mudança de Base

```

import java.util.Scanner;
import java.math.BigInteger;

class Main /* UVa 10551 - Basic Remains */
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        while (true)
        {
            int b = sc.nextInt();
            if (b == 0) break;
            String p_str = sc.next();
            BigInteger p = new BigInteger(p_str, b);
            String m_str = sc.next();
            BigInteger m = new BigInteger(m_str, b);
            System.out.println((p.mod(m)).toString(b));
        }
    }
}

```

9.11 Multiplicação de Matrizes

```

programa multiplica_matrizes;
matriz mat1, mat2, mat3;
inteiro linha, coluna, i, acumula;
"leia mat1";
"leia mat2";
"verifique se mat1 é compativel com mat2";
para linha de 1 até "numero de linhas de mat1" faça
    para coluna de 1 até "numero de colunas de mat2" faça
        acumula=0;
        para i de 1 até "numero de colunas de mat1" faça
            acumula=acumula+mat1[linha][i]*mat2[i][coluna];

```

```

    fimpara;
    mat3[linha][coluna]=acumula;
    fimpara;
15   fimpara;
    imprima mat3;
fim programa;

```

9.12 Número de Euler

$$e = \sum_{n=0}^{\infty} \frac{1}{n!} \quad (8)$$

9.13 Números Primos – map impl.

```

typedef map<int, int> prime_map;
void squeeze(prime_map& M, int& n, int p)
{
    for (; n % p == 0; n /= p) M[p]++;
5 }
void factor(int n, prime_map& M)
{
    if (n < 0) { factor(-n, M); return; }
    if (n < 2) return;
10
    squeeze(M, n, 2);
    squeeze(M, n, 3);

    int maxP = sqrt(n) + 2;
    for (int p = 5; p < maxP; p += 6)
15 {
        squeeze(M, n, p);
        squeeze(M, n, p+2);
    }
20   if (n > 1) M[n]++;
}

```

9.14 Progressões

9.14.1 Aritméticas

n -ésimo termo == $a_n = a_1 + (n - 1) \times r$

Soma dos termos do intervalo $[a_p, a_q] == \frac{(q-p+1) \times (a_p+a_q)}{2}$

Soma dos termos de uma P. A. == $\frac{n \times (a_1+a_n)}{2}$

Progressão aritmética de segunda ordem == sequência de números em que as diferenças entre os termos consecutivos segue uma progressão aritmética; exemplo: 1, 3, 7, 13, 21, 31, 43, 57, 73

9.14.2 Geométricas

Soma dos termos de uma P. G. $\sum a_1 q^{n-1} = \frac{a_1(1-q^n)}{1-q}$

Soma dos infinitos termos de uma P. G. ($|q| < 1$) $= \frac{a_1}{1-q}$

Produto dos termos de uma P. G. $= (a_1 \times a_n)^{\frac{n}{2}}$

$(\sum == \text{somatório});$
 $(\prod == \text{produtório});$

9.15 Resto sem Divisão

```

void adjust(char* str)
{
    int i;

    for(i = 0; str[i]; i++)
    {
        str[i] = str[i] - '0';
    }
}

int main()
{
    int curr, div, i;
    char* read = (char*)malloc(2*(MAXSTRSZ+1)*sizeof(char));
    char* num = &(read[MAXSTRSZ+1]);

    while(1)
    {
        fgets(num, MAXSTRSZ, stdin);
        if(num[strlen(num)-1] == '\n')
            num[strlen(num)-1] = '0';

        if(num[0] == '0' && num[1] == '1' | '0')
            break;
    }

    sprintf(read, "%s", num);

    adjust(num);

    curr = 0;

    div = 0;
    while(1)
    {
        div = div*10 + num[curr++];
        while(div < 11)
        {
            if(curr == strlen(read))
                break;

            div = div*10+num[curr++];
        }
        div = div%11;
    }
}

```

```

45     if(curr == strlen(read))
        break;
    }

    if(div == 0)
        printf("%s is a multiple of 11.\n", read);
    else
        printf("%s is not a multiple of 11.\n", read);
}

return 0;
}

```

9.16 Roman Numbers

Many persons are familiar with the Roman numerals for relatively small numbers. The symbols “i”, “v”, “x”, “l” and “c” represent the decimal values 1, 5, 10, 50 and 100 respectively. To represent other values, these symbols, and multiples where necessary, are concatenated, with the smaller-valued symbols written further to the right. For example, the number 3 is represented as “iii”, and the value 73 is represented as “lxxiii”. The exceptions to this rule occur for numbers having units values of 4 or 9, and for tens values of 40 or 90. For these cases, the Roman numeral representations are “iv” (4), “ix” (9), “xl” (40), and “xc” (90). So the Roman numeral representations for 24, 39, 44, 49, and 94 are “xxiv”, “xxxix”, “xliv”, “xlix”, and “xciv”, respectively.

9.17 Teste de Primalidade

9.17.1 Crivo de Eratóstenes com Fatoração

Minha Implementação

```

#define SIZE 78498

long long p[SIZE];
vector<bool> aval(1000001, false);
5 map<long long, set<long long>> fatoracao;

bool isPrime(long long key)
{
    if(key == 0 || key == 1)
        return false;
    if(key == 2 || key == 3)
        return true;

    long long i, lim = round(ceil(sqrt(key)));
10
    for(i = 2; i <= lim; i++)
    {
        if(key % i == 0)
            return false;
20    }
}

```

```

        return true;
    }

25 long long modulo(long long a, long long b, long long c)
{
    long long x=1, y=a;
    while(b > 0)
    {
        if(b%2 == 1)
        {
            x=(x*y)%c;
        }

35     y = (y*y)%c;
     b >>= 1;
    }

        return x%c;
40 }

bool fermat(long long key)
{
    if(key == 0 || key == 1)
        return false;

    if(key == 2 || key == 3)
        return true;

50 long long sort = rand()%(key-1) + 1;

    if(modulo(sort, key - 1, key) != 1)
        return false;

    return true;
55 }

long long nextFactor(long long key, long long* first)
{
    for(long long i = *first; i<SIZE; i++)
    {
        if(key%p[i] == 0)
        {
            *first = i;
            65 return p[i];
        }
    }

        return -1;
70 }

void una(set<long long> &fat, long long ind)
{
    for(set<long long>::iterator it = fatoracao[ind].begin(); it!=
        fatoracao[ind].end(); it++)
        fat.insert(*it);
75 }

```

```

void fatore(long long key, set<long long> &fatores)
{
    80   key = key<0?key*(-1):key;

    long long val = key;

    long long first = 0;
    85   while(val != 1)
    {
        if(fatoracao.find(val)!=fatoracao.end())
        {
            una(fatores, val);
            break;
        }

        long long d = nextFactor(val, &first);

        fatores.insert(d);

        val /= d;
    }
}

100 void crivoEratostenes(long long limE, long long limD, long long*
    add)
{
    long long i, j;

    105  for(i = limE; i<=limD; i++)
    {
        if(!aval[i])
        {
            p[(*add)++] = i;
            110   for(j = i; j<=1000000; j+=i)
                aval[j] = true;
        }
    }
}

115 int main()
{
    long long key, i, j, add = 0;

    120   crivoEratostenes(2, 1000000, &add);

    while(1)
    {
        scanf("%lld ", &key);

        125   if(!key)
            break;

        set<long long> fatores;
        fatore(key, fatores);
        fatoracao[key] = fatores;

        cout << key << " : " << fatores.size() << endl;
    }
}

```

```

135     return 0;
}

```

Programming Challenges

```

typedef long long ll;
typedef vector<int> vi;
typedef map<int, int> mii;

5   ll _sieve_size;
bitset<10000010> bs;
vi primes;

10  void sieve(ll upperbound)
{
    _sieve_size = upperbound + 1;
    bs.set();
    bs[0] = bs[1] = 0;
    for (ll i = 2; i <= _sieve_size; i++) if (bs[i])
15    {
        for (ll j = i * i; j <= _sieve_size; j += i) bs[j] = 0;
        primes.push_back((int)i);
    }
20}

25  bool isPrime(ll N)
{
    if (N <= _sieve_size) return bs[N];
    for (int i = 0; i < (int)primes.size(); i++)
        if (N % primes[i] == 0) return false;
    return true;
}

30  vi primeFactors(ll N)
{
    vi factors;
    ll PF_idx = 0, PF = primes[PF_idx];
    while (N != 1 && (PF * PF <= N))
35    {
        while (N % PF == 0) { N /= PF; factors.push_back(PF); }
        PF = primes[++PF_idx];
    }
    if (N != 1) factors.push_back(N);
    return factors;
}

45  ll numPF(ll N)
{
    ll PF_idx = 0, PF = primes[PF_idx], ans = 0;
    while (N != 1 && (PF * PF <= N))
50    {
        while (N % PF == 0) { N /= PF; ans++; }
    }
}

```

```

        PF = primes[PF_idx];
    }
    if (N != 1) ans++;
    return ans;
55 }

ll numDiffPF(ll N)
{
    ll PF_idx = 0, PF = primes[PF_idx], ans = 0;
    while (N != 1 && (PF * PF <= N))
    {
        if (N % PF == 0) ans++;
        while (N % PF == 0) N /= PF;
        PF = primes[PF_idx];
65 }
        if (N != 1) ans++;
        return ans;
    }
}

70 ll sumPF(ll N)
{
    ll PF_idx = 0, PF = primes[PF_idx], ans = 0;
    while (N != 1 && (PF * PF <= N))
    {
        while (N % PF == 0) { N /= PF; ans += PF; }
        PF = primes[PF_idx];
    }
    if (N != 1) ans += N;
    return ans;
80 }

ll numDiv(ll N)
{
    ll PF_idx = 0, PF = primes[PF_idx], ans = 1;
    while (N != 1 && (PF * PF <= N))
    {
        ll power = 0;
        while (N % PF == 0) { N /= PF; power++; }
        ans *= (power + 1);
50     PF = primes[PF_idx];
    }
    if (N != 1) ans *= 2;
    return ans;
}
95 }

ll sumDiv(ll N)
{
    ll PF_idx = 0, PF = primes[PF_idx], ans = 1;
    while (N != 1 && (PF * PF <= N)) {
100     ll power = 0;
        while (N % PF == 0) { N /= PF; power++; }
        ans *= ((ll)pow((double)PF, power + 1.0) - 1) / (PF - 1);
        PF = primes[PF_idx];
    }
    if (N != 1) ans *= ((ll)pow((double)N, 2.0) - 1) / (N - 1);
105    return ans;
}

```

```

111 ll EulerPhi(ll N)
110 {
112     ll PF_idx = 0, PF = primes[PF_idx], ans = N;
113     while (N != 1 && (PF * PF <= N)) {
114         if (N % PF == 0) ans -= ans / PF;
115         while (N % PF == 0) N /= PF;
116         PF = primes[++PF_idx];
117     }
118     if (N != 1) ans -= ans / N;
119     return ans;
120 }
121
122 int main()
123 {
124     sieve(10000000);
125     printf("%d\n", isPrime(2147483647));
126     printf("%d\n", isPrime(136117223861LL));
127
128     vi res = primeFactors(2147483647);
129     for (vi::iterator i = res.begin(); i != res.end(); i++) printf(">%d\n", *i);
130
131     res = primeFactors(136117223861LL);
132     for (vi::iterator i = res.begin(); i != res.end(); i++) printf("#%d\n", *i);
133
134     res = primeFactors(142391208960LL);
135     for (vi::iterator i = res.begin(); i != res.end(); i++) printf("!%d\n", *i);
136
137
138     printf("numPF(%d) = %lld\n", 50, numPF(50));
139     printf("numDiffPF(%d) = %lld\n", 50, numDiffPF(50));
140     printf("sumPF(%d) = %lld\n", 50, sumPF(50));
141     printf("numDiv(%d) = %lld\n", 50, numDiv(50));
142     printf("sumDiv(%d) = %lld\n", 50, sumDiv(50));
143     printf("EulerPhi(%d) = %lld\n", 50, EulerPhi(50));
144
145     return 0;
146 }

```

9.17.2 Fermat

```

long long modulo(long long a, long long b, long long c)
{
    long long x=1, y=a;
    while(b > 0)
    {
        if(b%2 == 1)
        {
            x=(x*y)%c;
        }
        5
        y = (y*y)%c;
        b >= 1;
10
}

```

```

        }

15     return x%c;
}

long long potencia(long long base, long long exp)
{
    return (long long)(pow(double(base), double(exp)));
}

bool fermat(long long key)
{
25    if(key == 0 || key == 1)
        return false;

    if(key == 2 || key == 3)
        return true;
30
map<long long, bool> already;

long long sort;
for(long long i = 0; i<100; i++)
{
35    do
    {
        sort = rand()% (key - 1) + 1;
    } while(already.find(sort)!=already.end());
40
    already[sort] = true;

    if(modulo(sort, key - 1, key) != 1)
        return false;
45
    if(already.size() == key - 1)
        return true;
}

50    return true;
}

int main(int argc, char** argv)
{
55    long long i, j, k;

    vector<long long> primos;
    primos.push_back(2);

60    long long key;
    cin >> key;

    if(key < 0) key*=-1;

65    if(fermat(key))
        cout << "sim" << endl;
    else
        cout << "nao" << endl;

70    return 0;
}

```

}

9.17.3 \sqrt{n}

```
bool isPrime(long long &n)
{
    if (n == 0 || n == 1)
        return false;
    if (n==2 || n==3)
        return true;

    long long lim = round(ceil(sqrt(n)));
    for (long long i = 2; i<=lim; i++)
    {
        if (n%i == 0)
            return false;
    }
    return true;
}
```

10 Programação Dinâmica

1. Problemas alvo para Programação Dinâmica (PD / DP):
 - sub-estrutura ótima (princípio da otimalidade) → Solução ótima do problema inclui soluções ótimas dos subproblemas;
 - sub-problemas são sobrepostos → número “pequeno” de subproblemas distintos;
2. Linearização de Grafos Direcionados Acíclicos (DAGs);
3. Evitar recálculos dos subproblemas em comum;
 - menor para maior (bottom-up) → Iterativa;
 - tabelas ou memoização → Recursiva;
4. Etapas:
 - (a) defina estrutura de recorrência e o DAG associado
 - (b) resolva recursivamente com memoização;
 - (c) resolva iterativamente por tabela, na ordenação parcial do DAG implícito;
 - (d) construa a solução ótima a partir da tabela.

10.1 0-1 Knapsack

```
#define MAX_N 1010
#define MAX_W 40

int N, MW, V[MAX_N], W[MAX_N], memo[MAX_N][MAX_W];
5
int value(int id, int w)
{
    if (id == N || w == 0) return 0;
    if (memo[id][w] != -1) return memo[id][w];
    if (W[id] > w)           return memo[id][w] = value(id + 1, w);
    return memo[id][w] = max(value(id + 1, w), V[id] + value(id + 1,
        w - W[id]));
}

int main()
15
{
    int i, T, G, ans;

    scanf("%d", &T);
    while (T--)
    {
        memset(memo, -1, sizeof memo);

        scanf("%d", &N);
        for (i = 0; i < N; i++)
            scanf("%d %d", &V[i], &W[i]);

        ans = 0;
        scanf("%d", &G);
        while (G--)
        {
            scanf("%d", &MW);
            ans += value(0, MW);
        }

        printf("%d\n", ans);
    }

    return 0;
}
40
/*
45 #define MAX_N 1010
#define MAX_W 40

int main()
{
    int i, w, T, N, G, MW, V[MAX_N], W[MAX_N], C[MAX_N][MAX_W], ans;
50
    scanf("%d", &T);
    while (T--)
    {
        scanf("%d", &N);
        for (i = 1; i <= N; i++)
```

```

    scanf("%d %d", &V[i], &W[i]) ;

    ans = 0;
    scanf("%d", &G);
    while (G--)
    {
        scanf("%d", &MW);

        for (i = 0; i <= N; i++) C[i][0] = 0;
        for (w = 0; w <= MW; w++) C[0][w] = 0;

        for (i = 1; i <= N; i++)
        for (w = 1; w <= MW; w++)
        {
            if (Wi[i] > w) C[i][w] = C[i - 1][w];
            else C[i][w] = max(C[i - 1][w], V[i] + C[i - 1][w
                - W[i]]);
        }

        ans += C[N][MW];
    }

    printf("%d\n", ans);
}

return 0;
}

*/

```

10.2 Bitmasks

```

#define MAX_BITMASK 65536

char line[1000], name[1000];
int i, j, N, x[20], y[20], caseNo = 1;
5 double dist[20][20], memo[65536];

double matching(int bitmask)
{
    if (memo[bitmask] > -0.1)
10     return memo[bitmask];

    if (bitmask == (int)(pow(2.0, 2.0 * N) - 1))
        return memo[bitmask] = 0;

    15    double matching_value = 32767 * 32767;
    for (int p1 = 0; p1 < 2 * N; p1++)
        if (!(bitmask & (1 << p1)))
    {
        for (int p2 = p1 + 1; p2 < 2 * N; p2++)
20            if (!(bitmask & (1 << p2)))
                matching_value = min(matching_value,
                    dist[p1][p2] + matching(bitmask | (1 << p1) | (1 << p2)));
            break;
    }
}

```

```

25     return memo[bitmask] = matching_value;
}

int main()
{
    while (sscanf(gets(line), "%d", &N), N)
    {
        for (i = 0; i < 2 * N; i++)
            sscanf(gets(line), "%s %d %d", &name, &x[i], &y[i]);

        for (i = 0; i < 2 * N; i++)
            for (j = 0; j < 2 * N; j++)
                dist[i][j] = sqrt((double)(x[i] - x[j]) * (x[i] - x[j]) +
                    (y[i] - y[j]) * (y[i] - y[j]));
    }

    memset(memo, -1, sizeof memo);
    printf("Case %d: %.2lf\n", caseNo++, matching(0));
}

return 0;
}

```

10.3 Cutting Sticks (3.4.3)

```

int arr[55], memo[55][55];

int cut(int left, int right)
{
    if (left + 1 == right)
        return 0;
    if (memo[left][right] != -1)
        return memo[left][right];

    int ans = 2000000000;
    for (int i = left + 1; i < right; i++)
        ans = min(ans, cut(left, i) + cut(i, right) + (arr[right] - arr
            [left]));
    return memo[left][right] = ans;
}

int main()
{
    int l, n;

    while (scanf("%d", &l), l)
    {
        arr[0] = 0;
        scanf("%d", &n);
        for (int i = 1; i <= n; i++)
            scanf("%d", &arr[i]);
        arr[n + 1] = l;

        memset(memo, -1, sizeof memo);
        printf("The minimum cutting is %d.\n", cut(0, n + 1));
    }
}

```

```

    return 0;
}

```

10.4 Fishmonger (3.4.3)

```

typedef pair<int, int> ii;
#define MAX_N 55
#define MAX_T 1005
#define INF 2000000000

5   int i, j, n, t, toll[MAX_N][MAX_N], travelTime[MAX_N][MAX_N];
ii memo[MAX_N][MAX_T];

10  ii go(int cur, int t_left)
{
    if (t_left < 0)
        return ii(INF, INF);
    if (cur == n - 1)
        return ii(0, 0);
15  if (memo[cur][t_left] != ii(-1, -1))
    return memo[cur][t_left];
    ii ans = ii(INF, INF);
    for (int nxt = 0; nxt < n; nxt++)
        if (cur != nxt)
    {
        ii nextCity = go(nxt, t_left - travelTime[cur][nxt]);
        if (nextCity.first + toll[cur][nxt] < ans.first)
        {
            ans.first = nextCity.first + toll[cur][nxt];
            ans.second = nextCity.second + travelTime[cur][nxt];
        }
    }
    return memo[cur][t_left] = ans;
}
30

int main()
{
/*
4 7
35
0 5 2 3
5 0 2 3
3 1 0 2
3 3 2 0

40
0 2 2 7
2 0 1 2
2 2 0 5
7 2 5 0

45
0 0

50
6 6
*/
}

50 while (scanf("%d %d", &n, &t), (n || t))

```

```

    {
        for (i = 0; i < n; i++)
            for (j = 0; j < n; j++)
                scanf("%d", &travelTime[i][j]);
    55     for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            scanf("%d", &toll[i][j]);

        for (i = 0; i <= n; i++)
            for (j = 0; j <= t; j++)
                memo[i][j] = ii(-1, -1);
        ii res = go(0, t);
        printf("%d %d\n", res.first, res.second);
    }
65     return 0;
}

```

10.5 Maximum Sum

```

#define REP(i, a, b) for (int i = int(a); i <= int(b); i++)

int main()
{
    5     int i, j, k, l, n, arr[101][101], maxSubRect, subRect;

    scanf("%d", &n);
    REP (i, 0, n - 1)
        REP (j, 0, n - 1)
    10    {
        scanf("%d", &arr[i][j]);
        if (i > 0)
            arr[i][j] += arr[i - 1][j];
        if (j > 0)
            arr[i][j] += arr[i][j - 1];
        15    if (i > 0 && j > 0)
            arr[i][j] -= arr[i - 1][j - 1];
    }

    20    maxSubRect = -127*100*100;
    REP (i, 0, n - 1) REP (j, 0, n - 1)
        REP (k, i, n - 1) REP (l, j, n - 1)
    25    {
        subRect = arr[k][l];
        if (i > 0) subRect -= arr[i - 1][l];
        if (j > 0) subRect -= arr[k][j - 1];
        if (i > 0 && j > 0) subRect += arr[i - 1][j - 1];
        maxSubRect = max(maxSubRect, subRect);
    }

    30    printf("%d\n", maxSubRect);

    return 0;
}
35 */

```

```

#define REP(i, a, b) for (int i = int(a); i <= int(b); i++)
40 int main()
{
    int n, unCounted = 1, currentMax = 0, possibleMax, matrix
    [101][101];
    scanf("%d", &n);
    REP (i, 0, n - 1)
        REP (j, 0, n - 1)
    {
        scanf("%d", &matrix[i][j]);
        if (j > 0) matrix[i][j] += matrix[i][j - 1];
    }
    REP (left, 0, n - 1)
        REP (right, left, n - 1)
    {
        possibleMax = 0;
        REP (row, 0, n - 1)
        {
            if (left > 0)
                possibleMax += matrix[row][right] - matrix[row][left - 1];
            else
                possibleMax += matrix[row][right];
        }
        if (possibleMax < 0) possibleMax = 0;
        if (unCounted || possibleMax > currentMax)
        {
            currentMax = possibleMax;
            unCounted = 0;
        }
    }
    printf("%d\n", currentMax);
    return 0;
}
*/

```

10.6 Problema da Mochila

```

typedef struct
{
    int p, b;
} TCel;
5
int dp[1066][100];
TCel celulas[100];
10
int max(int a, int b)
{

```

```

        return a > b?a:b;
    }

int calculaF0(int peso, int indIni, int* n)
15 {
    int res1, res2;

    if(indIni == *n)
        return 0;

20    if(dp[peso][indIni] != -1)
        return dp[peso][indIni];

    if(celulas[indIni].p > peso)
25    {
        dp[peso][indIni] = calculaF0(peso, indIni+1, n);
        return dp[peso][indIni];
    }

30    dp[peso][indIni] = max(calculaF0(peso, indIni+1, n), celulas[
        indIni].b + calculaF0(peso-celulas[indIni].p, indIni+1, n));

    return dp[peso][indIni];
}

35 int main()
{
    int c, n, i, j, v1, v2;
    int cont = 1;

    while(1)
40    {
        scanf( "%d%d" , &c , &n);

        if(!c && !n)
            break;

        for(i = 0; i<=c; i++)
45        {
            for(j = 0; j<=n; j++)
            {
                dp[i][j] = -1;
            }
        }

50        for(i = 0; i<n; i++)
        {
            scanf( "%d%d" , &(celulas[i].p) , &(celulas[i].b));
        }

        printf( "Teste %d | n%d | n | n" , cont++, calculaF0(c, 0, &n));
60    }

    return 0;
}

```

10.7 Subset Sum

```
#define MAX_SUM 10000
int n;
int vet[TAM];
bool m[MAX_SUM];

5    bool subSetSum(int M, int c)
{
    for (int i = 0; i <= M; i++) m[i] = false;
    m[0] = true;

10   for (int i = 0; i < n; i++)
    {
        for (int j = M; j >= vet[i]; j--)
        {
            m[j] |= m[j - vet[i]];
        }
    }

15   return m[c];
20 }
```

11 Meta-Heurística: Roll and Improve

```
bool bestImprovement2(int* sol, int n, job* jobs, double** setup,
                      map<double, bool> &improved)
{
    int i, j, iMelhor = -1, jMelhor;
    double foMin;
5    det_tempos_conclusao_2(n, jobs, sol, setup, &foMin);

    map<double, bool>::iterator it;
    it = improved.find(foMin);

    if (it != improved.end())
10    {
        return false;
    }

    improved[foMin] = true;

    for (i = 1; i <= n - 1; i++)
15    {
        for (j = i + 1; j <= n; j++)
        {
            int aux = sol[i];
            sol[i] = sol[j];
            sol[j] = aux;

20            double fo;
            det_tempos_conclusao_2(n, jobs, sol, setup, &fo);

            if (fo < foMin)
25            {
```

```

30     foMin = fo;
31
32     iMelhor = i;
33     jMelhor = j;
34 }
35
36     aux = sol[i];
37     sol[i] = sol[j];
38     sol[j] = aux;
39 }
40
41 if(iMelhor == -1)
42     return false;
43
44 int aux = sol[iMelhor];
45 sol[iMelhor] = sol[jMelhor];
46 sol[jMelhor] = aux;
47 return true;
48 }
49
50 void defineProbabilidades(vector<double> &prob, int n, int posIni,
51                           double den, int typeProb)
52 {
53     prob[posIni] = 1000000.;
54
55     int i;
56
57     if(typeProb == 1)
58     {
59         for(i = posIni-1; i>=1; i--)
60         {
61             prob[i] = prob[i+1]/den;
62         }
63         for(i = posIni+1; i<=n; i++)
64         {
65             prob[i] = prob[i-1]/den;
66         }
67     }
68     else
69     {
70         for(i = posIni-1; i>=1; i--)
71             prob[i] = prob[i+1]*den;
72         for(i = posIni+1; i<=n; i++)
73             prob[i] = prob[i-1]*den;
74     }
75 }
76
77 void generateNew(int* sol, vector<vector<double>> &prob, int n)
78 {
79     int cs, currElem, i, j, add, last;
80     double acc, target, sort, valSort, possSort;
81     vector<int> candidates;
82     int* solAux = (int*)malloc((n+1)*sizeof(int));
83     for(i = 1; i<=n; i++)
84         solAux[i] = sol[i];
85     for(i = 1; i<=n; i++)

```

```

        candidates.push_back(i);

90      add = 0;
while(add<n)
{
    cs = rand()%candidates.size();
    currElem = candidates[ cs ];
    candidates.erase(candidates.begin() + cs);
95
    acc = 0.;

    target = 0.;

100   for(i = 1; i<=n; i++)
        target+=prob[currElem][i];

    last = -1;

105   sort = (((double)(rand()%100000001))/100000000.)*target;

    vector<int> positions;
    for(i = 1; i<=n; i++)
        positions.push_back(i);
110   while(1)
{
    valSort = rand()%positions.size();
    posSort = positions[valSort];
    positions.erase(positions.begin() + valSort);

115   acc+=prob[currElem][posSort];

    last = posSort;

    if(last == -1)
        continue;

    if(acc>=sort)
        break;
120 }
125

    for(i = 1; i<=n; i++)
        prob[i][last] = 0.;

130   solAux[last] = sol[currElem];
    add++;
}
135

    for(i = 1; i<=n; i++)
        sol[i] = solAux[i];
    free(solAux);
}

void newHeuristic2(int* solution, int n, job* jobs, double** setup)
{
    double ALPHA = 2.0; // Valor minimo do denominador;
    int BETA = 10; // Número de convergência;
    int OMEGA = 25; // Num Iterações por denominador;
    double KAPPA = 1.0; // Incremento do denominador;
}

```

```

145 //      double THETA = 200.0; // Percentual/100 minimo a ser
     alcançado;
double GAMMA;

//srand(time(NULL));
srand(1000000);

150 int i;
double valIni, min;
int exec, nBad, nCons, ult = -1;
double denominador;

155 myStr = (char*)malloc((1000000)*sizeof(char));

160 int* best = (int*)malloc((n+1)*sizeof(int));
for(i = 1; i<=n; i++)
    best[i] = solution[i];
vector< vector<double> > roletaBest;

165 vector<double> x;
for(i = 0; i<=n; i++)
    x.push_back(0.);
for(i = 0; i<=n; i++)
    roletaBest.push_back(x);

170 det_tempos_conclusao_2(n, jobs, solution, setup, &min);
valIni = min;

175 exec = 1;
denominador = 1.;
nBad = nCons = 0;
bool completed = false;
map<double, bool> bestImproved;
map<string, bool> foCalculated;

180 while(exec<=2)
{
    do
    {
        for(i = 1; i<=n; i++)
            defineProbabilidades(roletaBest[i], n, i, denominador, exec
                );
185

        generateNew(solution, roletaBest, n);
        double resultInter;
        det_tempos_conclusao_2(n, jobs, solution, setup, &resultInter
            );
190

        while(bestImprovement2(solution, n, jobs, setup, bestImproved
            ));

        double result;
        det_tempos_conclusao_2(n, jobs, solution, setup, &result);
195

        printf("Denominador == %.2lf |n", denominador);
        printf("Melhor Soluçao: %.0lf |n", min);
        printf("Soluçao Otimizada 1: %.0lf |n", resultInter);
    }
}

```

```

200     printf( "Solução Optimal 2: %.0lf\n", result);
201     printf( "\n");
202
203     if(result < min)
204     {
205         min = result;
206
207         for(i = 1; i<=n; i++)
208             best[i] = solution[i];
209
210         nBad = 0;
211         denominador = completed?ALPHA:1.;
212         exec = 1;
213         nCons = 0;
214
215         continue;
216     }
217     else
218         nBad++;
219
220     if(nBad%OMEGA==0) // || (resultInter > THETA*min/100)
221     {
222         if(denominador == 1.)
223             completed = true;
224         denominador+=KAPPA;
225     }
226
227     if(resultInter==ult)
228     {
229         nCons++;
230     }
231     else
232     {
233         ult = resultInter;
234         nCons = 0;
235     }
236
237     if(exec == 1)
238     {
239         if(nCons>BETA)
240             break;
241     }
242     else
243     {
244         if(denominador>GAMMA)
245             break;
246     }
247 } while(1);
248
249 if(exec == 1)
250     GAMMA = denominador;
251
252 denominador = 1.0;
253
254     exec++;
255 }
256
257 for(i = 1; i<=n; i++)

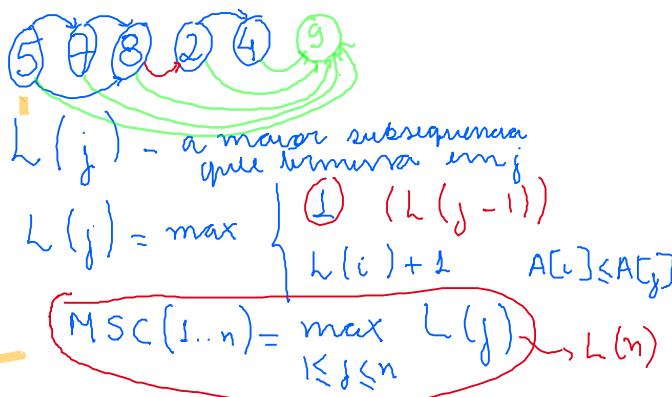
```

```
260     solution[i] = best[i];
        free(best);
        free(myStr);
    }
```

12 Strings

12.1 Distância de Edição

Erro da Aula Anterior...



Etapa 1: Definição da Recorrência

Com a equação proposta na aula passada, pode dar valor errado!!!!!!!!!!!!!!

$L(j) = 1 + \max\{L(i) : (i, j) \in E\}$ o } $(i, j) \in E \Rightarrow A[i] \leq A[j]$
 $L[1..n] \leftarrow 1$
 $\max = 1$
 $\text{for } i = 2 \dots n$
 $\text{for } j = i \dots 1$
 if $(A[j] \leq A[i])$: $\max = j$
 if $(L[j] + 1 > L[i])$: $L[i] = L[j] + 1$
 if $(L[i] > \max)$: $\max = L[i]$

Etapa 3: Algoritmo Iterativo (Tabela)

```
for j = 1, 2, ..., n:  
    L(j) = 1 + max{L(i) : (i, j) ∈ E}  
return max_j L(j)
```

Etapa 4: Construindo solução

```
for j = 1, 2, ..., n:  
    L(j) = 1 + max{L(i) : (i, j) ∈ E}  
return max_j L(j)
```

- Como calcular complexidade?
 - Número de subproblemas
 - Complexidade de composição

5 2 8 6 3 6 9 7

Distância de Edição

- Transformar uma sequência em outra ao menor custo.
 - Casamento, substituição, inserção, remoção.

n SITUADO m ESTUDO-
 m SITUADO ESTUDO
 n -S I T U A D O
 m E S - T U - D O

-SITUADO
 ES-TU-DO

Edição: Subproblemas

- Problema: Alinhar duas sequências de caracteres

$$E(m, n) = \min \left\{ \begin{array}{l} c(m, n) + E(m-1, n-1) \\ \perp + E(m-1, n) \\ \perp + E(m, n-1) \end{array} \right\}$$

$$E(0, 0) = 0$$

Etapa 1: Equação de Recorrência

- Problema: Alinhar duas sequências de caracteres

$$x[1 \dots m] \quad y[1 \dots n] \quad E(m, n)$$

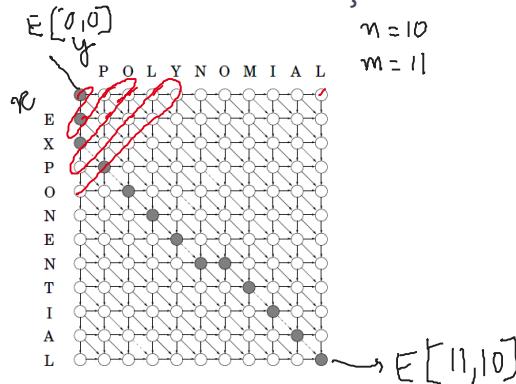
- Subproblema: alinhamento de prefixos

- Composição: inserir, remover, casar

$$\begin{array}{llll} E(i, j) & x[i] & - & y[j] \\ \text{x}[1 \dots i] & - & \text{or} & \text{y}[j] \\ \text{y}[1 \dots j] & - & \text{or} & \text{y}[j] \end{array}$$

$$E(i, j) = \min\{1 + E(i-1, j), 1 + E(i, j-1), \text{diff}(i, j) + E(i-1, j-1)\}$$

Distância de Edição - DAG



Etapa 2: Algoritmo Recursivo

$$E(i, j) = \min\{1 + E(i - 1, j), 1 + E(i, j - 1), \text{diff}(i, j) + E(i - 1, j - 1)\}$$

$E(m, n)$ - tabela $m \times n$ com linhas 0 e colunas preenchidas

```

function EdR( $i, j$ )
     $\rightarrow$  if ( $E(i, j) \neq -1$ ) : return  $E(i, j)$ ;
    ms = 1 + EdR( $i, j - 1$ )
    del = 1 + EdR( $i - 1, j$ )
    sub = diff( $x[i], y[j]$ ) + EdR( $i - 1, j - 1$ )
     $\rightarrow$   $E[i, j] = \min(ms, del, sub)$ 
    return  $E[i, j]$ 
```

Etapa 3: Algoritmo Iterativo

$$E(i, j) = \min\{1 + E(i - 1, j), 1 + E(i, j - 1), \text{diff}(i, j) + E(i - 1, j - 1)\}$$

0	1	2	
1	2	2	
2			10
3			11

Etapa 3: Complexidade

```

for i = 0, 1, 2, ..., m:
    E(i, 0) = i
    for j = 1, 2, ..., n:
        E(0, j) = j
        for i = 1, 2, ..., m:
            for j = 1, 2, ..., n:
                E(i, j) = min{E(i - 1, j) + 1, E(i, j - 1) + 1, E(i - 1, j - 1) + diff(i, j)}
return E(m, n)
```

$$\Theta(\text{diff}) = 1$$

- $\Theta(mn)$ de tempo e espaço

12.2 Highest-scoring Alignment

12.2.1 C++

```
int main()
{
    char A[20] = "ACAATCC", B[20] = "AGCATGC";
    int n = (int)strlen(A), m = (int)strlen(B);
    int i, j, table[20][20];

    memset(table, 0, sizeof table);
    for (i = 1; i <= n; i++)
        table[i][0] = i * -1;
    for (j = 1; j <= m; j++)
        table[0][j] = j * -1;

    for (i = 1; i <= n; i++)
        for (j = 1; j <= m; j++)
    {
        table[i][j] = table[i - 1][j - 1] + (A[i - 1] == B[j - 1] ? 2
            : -1);
        table[i][j] = max(table[i][j], table[i - 1][j] - 1);
        table[i][j] = max(table[i][j], table[i][j - 1] - 1);
    }

    printf("DP table:|n");
    for (i = 0; i <= n; i++)
    {
        for (j = 0; j <= m; j++)
            printf("%3d", table[i][j]);
        printf("|n");
    }
    printf("Maximum Alignment Score: %d|n", table[n][m]);

    return 0;
}
```

12.2.2 Java

```
class ch6_02_str_align
{
    public static void main(String[] args)
    {
        char[] A = "ACAATCC".toCharArray(), B = "AGCATGC".toCharArray()
        ;
        int[][] table = new int[20][20];
        int i, j, n = A.length, m = B.length;

        for (i = 1; i <= n; i++)
            table[i][0] = i * -1;
        for (j = 1; j <= m; j++)
            table[0][j] = j * -1;

        for (i = 1; i <= n; i++)
            for (j = 1; j <= m; j++)
    {
```

```

        table[i][j] = table[i - 1][j - 1] + (A[i - 1] == B[j - 1] ? 
            2 : -1);
        table[i][j] = Math.max(table[i][j], table[i - 1][j] - 1);
        table[i][j] = Math.max(table[i][j], table[i][j - 1] - 1);
    }

    System.out.printf("DP table: |n");
    for (i = 0; i <= n; i++)
    {
        for (j = 0; j <= m; j++)
            System.out.printf("%3d", table[i][j]);
        System.out.printf(" |n");
    }
    System.out.printf("Maximum Alignment Score: %d |n", table[n][m]);
}
}

```

12.3 KMP

12.3.1 Implementação do Livro

```

#define MAX_N 100010

char T[MAX_N], P[MAX_N];
int b[MAX_N], n, m;

void naiveMatching()
{
    for (int i = 0; i < n; i++)
    {
        bool found = true;
        for (int j = 0; j < m && found; j++)
            if (P[j] != T[i + j])
                found = false;
        if (found)
            printf("P is found at index %d in T|n", i);
    }
}

void kmpPreprocess()
{
    int i = 0, j = -1; b[0] = -1;
    while (i < m)
    {
        while (j >= 0 && P[i] != P[j]) j = b[j];
        i++; j++;
        b[i] = j;
    }
}

void kmpSearch()
{
    int i = 0, j = 0;
    while (i < n)
    {
        while (j >= 0 && T[i] != P[j]) j = b[j];

```

```

        i++;
        if (j == m)
        {
            printf("P is found at index %d in T\n", i - j);
            j = b[j];
        }
    }

45 int main()
{
    n = (int)strlen(gets(T));
    m = (int)strlen(gets(P));

    printf("Naive\n");
    naiveMatching();

    printf("KMP\n");
    kmpPreprocess();
    kmpSearch();

    return 0;
}

```

12.3.2 Outra

```

#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <string>
5 #include <vector>
#include <set>

using namespace std;

10 #define Iter set<string>::iterator

set<string> padroes;
int maxi;

15 void kmp(string S, string K){
    vector<int> T(K.size() + 1, -1);
    register int i;

    if(K.size() == 0) return;
    if(K.size() < maxi) return;

20    for(i = 1; i <= K.size(); i++){
        int pos = T[i - 1];
        while(pos != -1 && K[pos] != K[i - 1]) pos = T[pos];
        T[i] = pos + 1;
    }

25    int sp = 0;
    int kp = 0;
    while(sp < S.size()){

30

```

```

    while(kp != -1 && (kp == K.size() || K[kp] != S[sp])) kp =
        T[kp];
    kp++;
    sp++;
    if(kp == K.size()){
        padroes.insert(K);
        if(K.size() > maxi){
            maxi = K.size();
        }
        return;
    }
}

int main(){
    string string1, string2;
    bool first = true;
    register int i, j;

    while(getline(cin, string1) && getline(cin, string2)){
        getchar();
        padroes.clear();
        maxi = 0;

        if(!first) printf(" | n ");

        if(string1.size() > string2.size()){
            for(i=0; i<string2.size(); i++)
                for(j=string2.size(); j>=i; j--)
                    kmp(string1, string2.substr(i, j-i));
        }
        else{
            for(i=0; i<string1.size(); i++)
                for(j=string1.size(); j>=i; j--)
                    kmp(string2, string1.substr(i, j-i));
        }

        if(padroes.size() == 0) printf("No common sequence. | n ");
        else for(Iter it = padroes.begin(); it != padroes.end(); ++it)
            if(it->size() == maxi)
                printf("%s | n ", it->c_str());
    }

    first = false;
}

return 0;
}

```

12.4 Longest Common Substring

12.4.1 C++

```

typedef pair<int, int> ii;

#define MAX_N 100010

char T[MAX_N];

```

```

int n;
int RA[MAX_N], tempRA[MAX_N];
int SA[MAX_N], tempSA[MAX_N];
int c[MAX_N];
10
char P[MAX_N];
int m;

int Phi[MAX_N];
15 int PLCP[MAX_N];
int LCP[MAX_N];

bool cmp(int a, int b) { return strcmp(T + a, T + b) < 0; }

20 void constructSA_slow()
{
    for (int i = 0; i < n; i++) SA[i] = i;
    sort(SA, SA + n, cmp);
}
25

void countingSort(int k)
{
    int i, sum, maxi = max(300, n);
    memset(c, 0, sizeof c);
    for (i = 0; i < n; i++)
        c[i + k < n ? RA[i + k] : 0]++;
    for (i = sum = 0; i < maxi; i++)
    {
        int t = c[i]; c[i] = sum; sum += t;
    }
    for (i = 0; i < n; i++)
        tempSA[c[SA[i] + k < n ? RA[SA[i] + k] : 0]++] = SA[i];
    for (i = 0; i < n; i++)
        SA[i] = tempSA[i];
}
40

void constructSA()
{
    int i, k, r;
    for (i = 0; i < n; i++) RA[i] = T[i] - ' ';
    for (i = 0; i < n; i++) SA[i] = i;
    for (k = 1; k < n; k <<= 1)
    {
        countingSort(k);
        countingSort(0);
        tempRA[SA[0]] = r = 0;
        for (i = 1; i < n; i++)
            tempRA[SA[i]] =
                (RA[SA[i]] == RA[SA[i-1]] && RA[SA[i]+k] == RA[SA[i-1]+k])
                    ? r : ++r;
        for (i = 0; i < n; i++)
            RA[i] = tempRA[i];
    }
}
50

void computeLCP_slow()
{
    LCP[0] = 0;
}

```

```

    for (int i = 1; i < n; i++)
    {
        int L = 0;
        while (T[SA[i] + L] == T[SA[i-1] + L]) L++;
        LCP[i] = L;
    }
}

void computeLCP()
{
    int i, L;
    Phi[SA[0]] = -1;
    for (i = 1; i < n; i++)
        Phi[SA[i]] = SA[i-1];
    for (i = L = 0; i < n; i++)
    {
        if (Phi[i] == -1) { PLCP[i] = 0; continue; }
        while (T[i + L] == T[Phi[i] + L]) L++;
        PLCP[i] = L;
        L = max(L-1, 0);
    }
    for (i = 1; i < n; i++)
        LCP[i] = PLCP[SA[i]];
}

ii stringMatching()
{
    int lo = 0, hi = n-1, mid = lo;
    while (lo < hi)
    {
        mid = (lo + hi) / 2;
        int res = strncmp(T + SA[mid], P, m);
        if (res >= 0) hi = mid;
        else           lo = mid + 1;
    }
    if (strncmp(T + SA[lo], P, m) != 0) return ii(-1, -1);
    ii ans; ans.first = lo;
    lo = 0; hi = n - 1; mid = lo;
    while (lo < hi)
    {
        mid = (lo + hi) / 2;
        int res = strncmp(T + SA[mid], P, m);
        if (res > 0) hi = mid;
        else           lo = mid + 1;
    }
    if (strncmp(T + SA[hi], P, m) != 0) hi--;
    ans.second = hi;
    return ans;
}

void LRS()
{
    int i, maxLCP = 0;
    char ans[MAX_N];
    strcpy(ans, "");
    for (i = 1; i < n; i++)

```

```

    if (LCP[i] > maxLCP)
    {
        maxLCP = LCP[i];
        strncpy(ans, T + SA[i], maxLCP);
        ans[maxLCP] = 0;
    }

    printf(" | nThe LRS is '%s' with length = %d | n", ans, maxLCP);
}

130 int owner(int idx) { return (idx < n-m-1) ? 1 : 2; }

void LCS()
{
    135 int i, j, maxLCP = 0, idx = 0;
    char ans[MAX_N];
    strcpy(ans, "");

    printf(" | nRemember, T = '%s' | nNow, enter another string P: | n", T)
        ;
    140 m = (int)strlen(gets(P));
    strcat(T, P);
    strcat(T, "/");
    n = (int)strlen(T);
    constructSA();
    computeLCP();
    printf(" | nThe LCP information of 'T.P' = '%s': | n", T);
    printf(" i | tSA[i] | tLCP[i] | tOwner | tSuffix | n");
    for (i = 0; i < n; i++)
        printf(" %2d | %2d | %2d | %2d | %s | n", i, SA[i], LCP[i], owner(SA[i]),
               T + SA[i]);

    150 for (i = 1, maxLCP = -1; i < n; i++)
        if (LCP[i] > maxLCP && owner(SA[i]) != owner(SA[i-1]))
    {
        maxLCP = LCP[i];
        idx = i;
    }

    strncpy(ans, T + SA[idx], maxLCP);
    ans[maxLCP] = 0;
    printf(" | nThe LCS is '%s' with length = %d | n", ans, maxLCP);
}

160 int main()
{
    165 printf("Enter a string T below, we will compute its Suffix Array
              : | n");
    n = (int)strlen(gets(T));

    constructSA_slow();
    printf(" | nThe Suffix Array of string T = '%s' is shown below (O(n
              ^2 log n) version): | n", T);
    printf(" i | tSA[i] | tSuffix | n");
    for (int i = 0; i < n; i++) printf(" %2d | %2d | %s | n", i, SA[i], T
        + SA[i]);
    T[n++] = '.',';
}

```

```

175    constructSA();
printf("|\nThe Suffix Array of string T = '%s' is shown below (O(n
    log n) version):|\n", T);
printf("i | tSA[i] | tSuffix |\n");
for (int i = 0; i < n; i++) printf("%2d | %2d | %s |\n", i, SA[i], T
    + SA[i]);
180
computeLCP();
LRS();

printf("|\nNow, enter a string P below, we will try to find P in T
    :|\n");
printf("Enter an empty string to stop this string search demo!|\n"
    );
185 while (m = (int)strlen(gets(P)), m)
{
    if pos = stringMatching();
    if (pos.first != -1 && pos.second != -1)
    {
        printf("%s is found SA [%d .. %d] of %s |\n", P, pos.first, pos
            .second, T);
        printf("They are :|\n");
        for (int i = pos.first; i <= pos.second; i++)
            printf(" %s |\n", T + SA[i]);
    }
    else printf("%s is not found in %s |\n", P, T);
195
}
196 LCS();
197
return 0;
200 }
```

12.4.2 Java

```

class ch6_03_sa
{
    Scanner scan;
    char T[];
    int n;

    int[] RA, tempRA;
    Integer[] SA, tempSA;
    int[] c;

    10   char P[];
    int m;

    int[] Phi;
    int[] PLCP;
    int[] LCP;

    void constructSA_slow()
    {
        for (int i = 0; i < n; i++) SA[i] = i;

        Arrays.sort(SA, 0, n, new Comparator<Integer>())
20 }
```

```

    {
        public int compare(Integer a, Integer b)
        {
            for (int i=0; a+i < T.length && b+i < T.length; i++)
            {
                if (T[a+i] != T[b+i]) return T[a+i] - T[b+i];
            }
            return b - a;
        }
    });

35 void countingSort(int k)
{
    int i, sum, maxi = Math.max(300, n);
    for (i = 0; i < 100010; i++) c[i] = 0;
    for (i = 0; i < n; i++)
        c[i + k < n ? RA[i + k] : 0]++;
    for (i = sum = 0; i < maxi; i++)
    {
        int t = c[i]; c[i] = sum; sum += t;
    }
    for (i = 0; i < n; i++)
        tempSA[c[SA[i] + k < n ? RA[SA[i] + k] : 0]] = SA[i];
    for (i = 0; i < n; i++)
        SA[i] = tempSA[i];
}

50 void constructSA()
{
    int i, k, r;
    for (i = 0; i < n; i++) RA[i] = T[i] - ' ';
    for (i = 0; i < n; i++) SA[i] = i;
    for (k = 1; k < n; k <= 1)
    {
        countingSort(k);
        countingSort(0);
        tempRA[SA[0]] = r = 0;
        for (i = 1; i < n; i++)
            tempRA[SA[i]] =
                (RA[SA[i]] == RA[SA[i-1]] && RA[SA[i]+k] == RA[SA[i-1]+k]
                ]) ? r : ++r;
        for (i = 0; i < n; i++)
            RA[i] = tempRA[i];
    }
}

70 void computeLCP_slow()
{
    LCP[0] = 0;
    for (int i = 1; i < n; i++)
    {
        int L = 0;
        while (T[SA[i] + L] == T[SA[i-1] + L]) L++;
        LCP[i] = L;
    }
}

```

```

80 void computeLCP()
{
    int i, L;
    Phi[SA[0]] = -1;
    for (i = 1; i < n; i++)
        Phi[SA[i]] = SA[i-1];
    for (i = L = 0; i < n; i++)
    {
        if (Phi[i] == -1) { PLCP[i] = 0; continue; }
        while (i + L < T.length && Phi[i] + L < T.length && T[i + L]
               == T[Phi[i] + L]) L++;
        PLCP[i] = L;
        L = Math.max(L-1, 0);
    }
    for (i = 1; i < n; i++)
        LCP[i] = PLCP[SA[i]];
}

int strncmp(char[] a, int i, char[] b, int j, int n)
{
    for (int k=0; i+k < a.length && j+k < b.length; k++)
    {
        if (a[i+k] != b[j+k]) return a[i+k] - b[j+k];
    }
    return 0;
}

int[] stringMatching()
{
    int lo = 0, hi = n-1, mid = lo;
    while (lo < hi)
    {
        mid = (lo + hi) / 2;
        int res = strncmp(T, SA[mid], P, 0, m);
        if (res >= 0) hi = mid;
        else           lo = mid + 1;
    }
    if (strncmp(T, SA[lo], P, 0, m) != 0) return new int[]{-1, -1};
    int[] ans = new int[]{lo, 0};

    lo = 0; hi = n - 1; mid = lo;
    while (lo < hi)
    {
        mid = (lo + hi) / 2;
        int res = strncmp(T, SA[mid], P, 0, m);
        if (res > 0) hi = mid;
        else           lo = mid + 1;
    }
    if (strncmp(T, SA[hi], P, 0, m) != 0) hi--;
    ans[1] = hi;
    return ans;
}

void LRS()
{
    int i, idx = 0, maxLCP = 0;

    for (i = 1; i < n; i++)

```

```

    if (LCP[i] > maxLCP)
    {
        maxLCP = LCP[i];
        idx = i;
    }

    System.out.printf("The LRS is '%s' with length = %d\n",
        new String(T).substring(SA[idx], maxLCP), maxLCP);
}

int owner(int idx) { return (idx < n-m-1) ? 1 : 2; }

void LCS()
{
    int i, j, maxLCP = 0, idx = 0;

    System.out.printf("\nRemember, T = '%s'\nNow, enter another
        string P:\n", new String(T));
    P = scan.nextLine().toCharArray();
    m = P.length;
    T = (new String(T) + new String(P) + "/").toCharArray();
    n = T.length;
    constructSA_slow();
    computeLCP();
    System.out.printf("\nThe LCP information of 'T.P' = '%s':\n",
        new String(T));
    System.out.printf("i | tSA[i] | tLCP[i] | tOwner | tSuffix |\n");
    for (i = 0; i < n; i++)
        System.out.printf("%2d | %2d | %2d | %2d | %s |\n", i, SA[i], LCP[i],
            owner(SA[i]),
            new String(T, SA[i], T.length - SA[i]));
}

for (i = 1, maxLCP = -1; i < n; i++)
    if (LCP[i] > maxLCP && owner(SA[i]) != owner(SA[i-1]))
    {
        maxLCP = LCP[i];
        idx = i;
    }

    System.out.printf("The LCS is '%s' with length = %d\n",
        new String(T).substring(SA[idx], SA[idx] + maxLCP),
        maxLCP);
}

void run()
{
    int MAX_N = 100010;
    c = new int[MAX_N];
    RA = new int[MAX_N];
    tempRA = new int[MAX_N];
    SA = new Integer[MAX_N];
    tempSA = new Integer[MAX_N];
    Phi = new int[MAX_N];
    PLCP = new int[MAX_N];
    LCP = new int[MAX_N];

    scan = new Scanner(System.in);
}

```

```

        System.out.printf("Enter a string T below, we will compute its
                           Suffix Array:|n");
        T = scan.nextLine().toCharArray();
        n = T.length;

195       constructSA_slow();
        System.out.printf("|nThe Suffix Array of string T = '%s' is
                           shown below ( $O(n^2 \log n)$  version):|n",
                           new String(T));
        System.out.printf("i | tSA[i] | tSuffix |n");
        for (int i = 0; i < n; i++)
200           System.out.printf("%2d | t%2d | t%s |n", i, SA[i], new String(T, SA
                           [i], T.length - SA[i]));

        T = (new String(T) + ".").toCharArray();
        constructSA();
        System.out.printf("|nThe Suffix Array of string T = '%s' is
                           shown below ( $O(n \log n)$  version):|n",
                           new String(T));
        System.out.printf("i | tSA[i] | tSuffix |n");
        for (int i = 0; i < n; i++)
205           System.out.printf("%2d | t%2d | t%s |n", i, SA[i], new String(T, SA
                           [i], T.length - SA[i]));

        computeLCP();
210       LRS();

        System.out.printf("|nNow, enter a string P below, we will try
                           to find P in T:|n");
        System.out.printf("Enter an empty string to stop this string
                           search demo!|n");
        while (true)
215       {
            P = scan.nextLine().toCharArray();
            if ((m = P.length) == 0) break;
            int[] pos = stringMatching();
            if (pos[0] != -1 && pos[1] != -1)
220           {
                System.out.printf("%s is found SA [%d .. %d] of %s |n",
                                  new String(P), pos[0], pos[1], new String(T));
                System.out.printf("They are:|n");
                for (int i = pos[0]; i <= pos[1]; i++)
225                   System.out.printf(" %s |n", new String(T, SA[i], T.length
                           - SA[i]));
                }
            else
230               System.out.printf("%s is not found in %s |n", new String(P),
                                 new String(T));
        }

        LCS();
    }

    public static void main(String[] args)
235    {
        new ch6_03_sa().run();
    }
}

```

12.5 Longest Increasing Subsequence

12.5.1 C++

```
/* What Goes Up */

#define MAX_N 200000
5
int main()
{
    int i, lnis, n, X[MAX_N], A[MAX_N], caseNo = 1;
    bool first = true;
10
    while (1)
    {
        for (n = 0; scanf("%d", &X[n]), X[n] != -1; n++)
        if (X[0] == -1) break;
15
        for (A[0] = X[0], i = lnis = 1; i < n; i++)
        {
            int *l = upper_bound(A, A + lnis, X[i], greater<int>());
            lnis = max(lnis, (int)(l - A) + 1);
            *l = X[i];
        }
20
        if (!first) printf(" | n");
        first = false;
        printf(" Test #%d : | n", caseNo++);
        printf(" maximum possible interceptions : %d | n", lnis);
    }

    return 0;
30 }
```

12.5.2 Java

```
class Main /* What Goes Up */
{
5
    private static int upperbound(int[] A, int len, int key)
    {
        int i;
        for (i = 0; i < len; i++)
        if (A[i] < key)
10         break;
        return i;
    }

    public static void main(String[] args)
15    {
        final int MAX_N = 200000;
        int i, lnis = 0, n, caseNo = 1;
        int[] X = new int[MAX_N], A = new int[MAX_N];
        Boolean first = true;
20    }
```

```

Scanner sc = new Scanner(System.in);
while (true)
{
    n = 0;
    while (true)
    {
        X[n] = sc.nextInt();
        if (X[n] == -1) break;
        n++;
    }

    if (X[0] == -1) break;

    for (A[0] = X[0], i = lnis = 1; i < n; i++)
    {
        int l = upperbound(A, lnis, X[i]);
        lnis = Math.max(lnis, l + 1);
        A[l] = X[i];
    }

    if (!first) System.out.printf("\n");
    first = false;
    System.out.printf("Test #%-d:\n", caseNo++);
    System.out.printf("maximum possible interceptions: %d\n",
                      lnis);
}
}
}

```

13 Tutoriais

13.1 scanf

```

int N; // using global variables in contests can be a
       // good strategy
char x[110]; // make it a habit to set array size slightly
              // larger than needed

int main()
{
    freopen("ch1_02_scanf_input.txt", "r", stdin);

    scanf("%d\n", &N);
    while (N--) // we simply loop from N, N-1, N-2, ...
    {
        scanf("0.%[0-9]...%n", &x); // if you are surprised with
                                       // this line,
                                       // check scanf details in www.cppreference.
                                       // com
        printf("the digits are 0.%s\n", x);
    }
    return 0;
}

```

13.2 Java Scanner

```
/*
 * Para finalizar uma leitura , pode-se utilizar :
 * scan.hasNext();
 */
5   * scan.hasNextLine();
 *
 * Ambos retornam um valor booleano .
 */

10 //Scan tokens from the file named "data.txt"
Scanner inFile = new Scanner("data.txt");
double d; char c; String s;
int tokenType;
tokenType = inFile.getToken();

15 switch (tokenType)
{
    case Scanner.DOUBLE: System.out.println("Found a Double = "+
        inFile.lexAsDouble()); break;
    case Scanner.CHARACTER: System.out.println("Found a Character = "+
        + inFile.lexAsCharacter()); break;
    case Scanner.WORD: System.out.println("Found a Word = "+ inFile.
        lexAsWord()); break;
    case Scanner.EOF: System.out.println("Found end of file . ");
}
}
```

Parte II

Exercícios Resolvidos

14 Ad-hoc

14.1 Ano Bissexto

- São bissextos todos os anos múltiplos de 400: 1600, 2000...
- São bissextos todos os múltiplos de 4 e não múltiplos de 100: 1996, 2004...
- Não são bissextos todos os demais.

14.2 Braile

14.3 Calculadora

```
int main()
{
    long long n, i;
    char c1, c2;
5     double totMult = 1, totDiv = 1, res = 1;
```

```

scanf( "%lld | n", &n);
for( i = 0; i<n; i++)
{
    scanf( "%c %c | n", &c1, &c2);
    if( c2 == '*')
        totMult*=c1- '0';
    else
        totDiv*=c1- '0';
    if( totMult > 100000000000 || totDiv > 100000000000)
    {
        res*=totMult/totDiv;
        totMult = totDiv = 1;
    }
}
printf( "%.0lf | n", res*totMult/totDiv);

return 0;
}

```

14.4 Dobradura

```

long long val[20];

int potencia(int b, int e)
{
    int i, res = 1;

    for( i = 0; i<e; i++)
    {
        res*=b;
    }

    return res;
}

int main()
{
    int i, cont, n;
    long long res;

    /* val[0] = 2;
    val[1] = 3;
    val[2] = 5;
    val[3] = 9;
    val[4] = 17;
    val[5] = 33;
    val[6] = 65;
    val[7] = 129;
    val[8] = 257;
    val[9] = 513;
    val[10] = 1025;
    val[11] = 2049;
    val[12] = 4097;
    val[13] = 8193;
    val[14] = 16385;
    val[15] = 32769;
}

```

```

  val[16] = 65537; /*

40   val[0] = 4;
    val[1] = 9;
    val[2] = 25;
    val[3] = 81;
    val[4] = 289;
    val[5] = 1089;
    val[6] = 4225;
45   val[7] = 16641;
    val[8] = 66049;
    val[9] = 263169;
    val[10] = 1050625;
    val[11] = 4198401;
50   val[12] = 16785409;
    val[13] = 67125249;
    val[14] = 268468225;
    val[15] = 1073807361;
    val[16] = 4295098369;

55   cont = 1;

  while(1)
{
60   scanf( "%d", &n);
    if(n == -1)
      break;

    printf( "Teste %d\n%d\n", cont++, val[n]);
65 }

  return 0;
}

```

14.5 Jogo da Vida

14.5.1 Especificação

Você provavelmente já ouviu falar de futebol, um jogo para 22 jogadores. Também provavelmente já ouviu falar de xadrez, que é um jogo para dois jogadores, e de paciência, um jogo para um único jogador. O jogo da vida é um jogo peculiar porque ele é um jogo para... zero jogadores.

O jogo da vida é composto por um tabuleiro 2D infinito, que possui infinitas colunas numeradas (...,-2,-1,0,1,2,...) e infinitas linhas também numeradas (...,-2,-1,0,1,2,...). Em cada posição do tabuleiro, há uma célula, que pode estar em dois estados: viva ou morta. O jogo começa em um dado estado inicial, que é completamente especificado por uma lista das células que estão vivas. A partir daí, há várias etapas. Em cada etapa, uma célula vai “nascer” (passar de morta para viva), morrer ou manter o estado atual, de acordo com os seus vizinhos. Os vizinhos de uma célula são as 8 células adjacentes, incluindo as diagonais. As regras que definem o que acontece com cada célula são:

- Se uma célula viva possui menos de dois vizinhos vivos, ela morre de solidão.

- Se uma célula viva possui mais de três vizinhos vivos, ela morre por superpopulação.
- Uma célula viva que possua dois ou três vizinhos vivos continua viva.
- Uma célula morta com exatamente três vizinhos vivos se torna uma célula viva.

Em cada etapa, todas as células atualizam seus estados de acordo com essas regras, simultaneamente.

Dada uma configuração inicial do jogo da vida, calcule e imprima a lista de células vivas após K etapas.

Entrada

Há vários casos de teste.

Cada caso de teste começa com um inteiro N , que representa o número de células vivas na configuração inicial ($0 < N \leq 1000$). Em seguida, há N linhas, cada uma das quais descrevendo uma célula viva. Essas linhas possuem dois inteiros L e C que indicam a linha e a coluna, respectivamente, de uma célula viva ($-1048576 < L, C < 1048576$). Você pode supor que a mesma célula não aparece duas vezes nessa lista. Por fim, a última linha do caso de teste possui um único inteiro K , que representa o número de etapas ($0 < K \leq 100$).

A entrada termina com $N = 0$, que não deve ser processado.

Saída

Para cada caso de teste, imprima uma linha contendo um inteiro C , o número de células vivas após K etapas. Em seguida, imprima C linhas, contendo as coordenadas L e C das células vivas. As células devem ser impressas ordenadas pelo número da linha. Células que possuem o mesmo número de linha devem ser ordenadas pelo número da coluna.

14.5.2 Solução

```

int contaVizinhos(map< pair<int,int> , bool > &status, pair<int,int> aval)
{
    int nNeigh = 0;
    map< pair<int,int> , bool >::iterator it2;
5     it2 = status.find(make_pair(aval.first - 1,aval.second - 1));
    if(it2!=status.end())
        if(it2->second)
            nNeigh++;
10    it2 = status.find(make_pair(aval.first - 1,aval.second));
    if(it2!=status.end())

```

```

    if(it2->second)
        nNeigh++;
15
    it2 = status.find(make_pair(aval.first - 1,aval.second + 1));
    if(it2!=status.end())
        if(it2->second)
            nNeigh++;
20
    it2 = status.find(make_pair(aval.first,aval.second - 1));
    if(it2!=status.end())
        if(it2->second)
            nNeigh++;
25
    it2 = status.find(make_pair(aval.first,aval.second + 1));
    if(it2!=status.end())
        if(it2->second)
            nNeigh++;
30
    it2 = status.find(make_pair(aval.first + 1,aval.second - 1));
    if(it2!=status.end())
        if(it2->second)
            nNeigh++;
35
    it2 = status.find(make_pair(aval.first + 1,aval.second));
    if(it2!=status.end())
        if(it2->second)
            nNeigh++;
40
    it2 = status.find(make_pair(aval.first + 1,aval.second + 1));
    if(it2!=status.end())
        if(it2->second)
            nNeigh++;
45
    return nNeigh;
}

void executaRodada(map< pair<int ,int > , bool > &status)
{
    map< pair<int ,int > , bool >::iterator it, it2;
    vector< pair<int ,int > > kill;
    vector< pair<int ,int > > born;

55
    for(it = status.begin(); it!=status.end(); it++)
    {
        int nNeigh = 0;

        it2 = status.find(make_pair(it->first.first - 1,it->first.
            second - 1));
60
        if(it2!=status.end())
        {
            if(it2->second)
            {
                nNeigh++;
            }
65
        }
        else
        {

```

```

    int res = contaVizinhos(status, make_pair(it->first.first -
      1,it->first.second - 1));
    if(res == 3)
      born.push_back(make_pair(it->first.first - 1,it->first.
        second - 1));
  }

  it2 = status.find(make_pair(it->first.first - 1,it->first.
    second));
  if(it2!=status.end())
  {
    if(it2->second)
    {
      nNeigh++;
    }
  }
  else
  {
    int res = contaVizinhos(status, make_pair(it->first.first -
      1,it->first.second));
    if(res == 3)
      born.push_back(make_pair(it->first.first - 1,it->first.
        second));
  }

  it2 = status.find(make_pair(it->first.first - 1,it->first.
    second + 1));
  if(it2!=status.end())
  {
    if(it2->second)
    {
      nNeigh++;
    }
  }
  else
  {
    int res = contaVizinhos(status, make_pair(it->first.first -
      1,it->first.second + 1));
    if(res == 3)
      born.push_back(make_pair(it->first.first - 1,it->first.
        second + 1));
  }

  it2 = status.find(make_pair(it->first.first,it->first.second -
    1));
  if(it2!=status.end())
  {
    if(it2->second)
    {
      nNeigh++;
    }
  }
  else
  {
    int res = contaVizinhos(status, make_pair(it->first.first,it-
      ->first.second - 1));
    if(res == 3)

```

```

        born.push_back(make_pair(it->first.first,it->first.second -
            1));
    }

    it2 = status.find(make_pair(it->first.first,it->first.second +
        1));
120   if(it2!=status.end())
    {
        if(it2->second)
        {
            nNeigh++;
        }
125   }
    else
    {
        int res = contaVizinhos(status, make_pair(it->first.first,it-
            ->first.second + 1));
130   if(res == 3)
        born.push_back(make_pair(it->first.first,it->first.second +
            1));
    }

    it2 = status.find(make_pair(it->first.first + 1,it->first.
        second - 1));
135   if(it2!=status.end())
    {
        if(it2->second)
        {
            nNeigh++;
        }
140   }
    else
    {
        int res = contaVizinhos(status, make_pair(it->first.first +
            1,it->first.second - 1));
145   if(res == 3)
        born.push_back(make_pair(it->first.first + 1,it->first.
            second - 1));
    }

    it2 = status.find(make_pair(it->first.first + 1,it->first.
        second));
150   if(it2!=status.end())
    {
        if(it2->second)
        {
            nNeigh++;
        }
155   }
    else
    {
        int res = contaVizinhos(status, make_pair(it->first.first +
            1,it->first.second));
160   if(res == 3)
        born.push_back(make_pair(it->first.first + 1,it->first.
            second));
    }

```

```

    it2 = status.find(make_pair(it->first.first + 1,it->first.
165      second + 1));
    if(it2!=status.end())
    {
        if(it2->second)
        {
            nNeigh++;
        }
170    }
    else
    {
        int res = contaVizinhos(status, make_pair(it->first.first +
175          1,it->first.second + 1));
        if(res == 3)
            born.push_back(make_pair(it->first.first + 1,it->first.
              second + 1));
    }

    if(it->second)
180    {
        if(nNeigh < 2 || nNeigh > 3)
            kill.push_back(it->first);
    }
    else
185    {
        if(nNeigh == 3)
            born.push_back(it->first);
    }
}
190
int i;
for(i = 0; i<kill.size(); i++)
{
    status[kill[i]] = false;
}
195
for(i = 0; i<born.size(); i++)
{
    status[born[i]] = true;
}
200}

int main(int argc, char** argv)
{
    long long i, j, k, nCases;
205    char* read = (char*)malloc((MAXSTRSZ+1)*sizeof(char));
    char c;

    while(1)
    {
        int nCells;

        scanf("%d", &nCells);
        if(!nCells)
            break;
210
        map< pair<int,int> , bool > status;
        for(i = 0; i<nCells; i++)
        {

```

```

220     int x, y;
221
222     scanf( "%d%d", &x, &y);
223
224     status[make_pair(x,y)] = true;
225 }
226
227 int nRounds;
228 scanf( "%d", &nRounds);
229
230 for(i = 0; i<nRounds; i++)
231 {
232     executaRodada(status);
233 }
234
235 vector< pair<int,int> > imprimir;
236
237 for(map< pair<int,int> , bool >::iterator it = status.begin();
238      it!=status.end(); it++)
239 {
240     if(it->second)
241     {
242         imprimir.push_back(it->first);
243     }
244 }
245 sort(imprimir.begin(), imprimir.end());
246 cout << imprimir.size() << endl;
247 for(i = 0; i<imprimir.size(); i++)
248 {
249     cout << imprimir[i].first << " " << imprimir[i].second <<
250         endl;
251 }
252
253 return 0;
254 }
```

14.6 Ones

```

int main(int argc, char** argv)
{
    unsigned int i, j, k; int n;
    char* read = (char*)malloc((MAXSTRSZ+1)*sizeof(char));
5
    map<int,int> answer;
    map<int,int>::iterator it;

    answer[1] = 3;
10   answer[11] = 4;
    answer[111] = 3;
    answer[1111] = 4;

    while((cin >> n)!=NULL)
15    {
        it = answer.find(n);
```

```

10      if(it == answer.end())
11      {
12          int n0nes = 1;
13
14          while(1)
15          {
16              int div = 1;
17              for(i = 0; i<n0nes; i++)
18              {
19                  div = (div % n)*10 + 1;
20
21                  if(div % n == 0)
22                  {
23                      answer[n] = n0nes+1;
24                      break;
25                  }
26                  n0nes++;
27              }
28          }
29
30          printf("%d\n", answer[n]);
31      }
32
33      return 0;
34
35

```

14.7 Posição em permutações / anagramas

```

double fatorial(double n)
{
    double acc = 1;
    for(double i = n; i>=1; i--)
{
    acc*=i;
}

return acc;
}

double nAnagrams(char* str, int indIni)
{
    map<char, int> occurs;
    map<char, int>::iterator it;

    for(int i = indIni; str[i]; i++)
{
    it = occurs.find(str[i]);
    if(it == occurs.end())
    {
        occurs[str[i]] = 1;
    }
    else
    {
        (occurs[str[i]])++;
    }
}

```

```

30     double tot = fatorial(strlen(str)-indIni);
31     for(it = occurs.begin(); it!=occurs.end(); it++)
32     {
33         tot/=fatorial(it->second);
34     }
35
36     return tot;
37 }
38
39 double findPerm(char* str)
40 {
41     if(strlen(str) == 1)
42         return 1;
43
44     set<char> aval;
45     set<char>::iterator it;
46
47     double pos = 0;
48     for(int i = 1; str[i]; i++)
49     {
50         if(str[i] < str[0] && !binary_search(aval.begin(), aval.end(),
51             str[i]))
52         {
53             aval.insert(str[i]);
54
54             char c = str[0];
55             str[0] = str[i];
56             str[i] = c;
57
58             pos+=nAnagrams(str, 1);
59
60             str[i] = str[0];
61             str[0] = c;
62         }
63     }
64
65     return pos + findPerm(str+1);
66 }
67
68 int main()
69 {
70     int i, j, k;
71     char* read = (char*)malloc((MAXSTRSZ+1)*sizeof(char));
72     char* readIni = read;
73
74     while(1)
75     {
76         cin.getline(read, MAXSTRSZ);
77         if(read[0] == '#' && read[1] == '\0')
78             break;
79
80         printf("%lf\n", findPerm(read));
81
82         read = readIni;
83     }
84
85     return 0;

```

```
}
```

14.8 Roman Counting

```
int encontraMaior(int n)
{
    if(n >= 100)
        return 100;
    if(n >= 50)
        return 50;
    if(n >= 10)
        return 10;
    if(n >= 5)
        return 5;
    return 1;
}

void generate(int num, map<int, vector<int> > &specials, int* contI
, int* contV, int* contX, int* contL, int* contC)
{
    if(!num)
        return;

    int factor;
    int i;
    for(i = 1; i*i<=10)
        if(i*i > num)
            break;
    factor = i;

    int n = num/factor*factor;

    if(specials.find(n)!=specials.end())
    {
        (*contI)+=specials[n][0];
        (*contV)+=specials[n][1];
        (*contX)+=specials[n][2];
        (*contL)+=specials[n][3];
        (*contC)+=specials[n][4];

        generate(num-n, specials, contI, contV, contX, contL, contC);
    }
    else
    {
        int ret = encontraMaior(n);

        switch(ret)
        {
            case 100: (*contC)++; break;
            case 50: (*contL)++; break;
            case 10: (*contX)++; break;
            case 5: (*contV)++; break;
            case 1: (*contI)++; break;
        }

        generate(num-ret, specials, contI, contV, contX, contL, contC);
    }
}
```

```

        }

55 int main()
{
    long long i, j, k;
    char* read = (char*)malloc((MAXSTRSZ+1)*sizeof(char));
    int num;
    int contI, contV, contX, contL, contC;
    map< int , vector<int> > specials;

    vector<int> aux (5,0);
    aux[0] = 1;
    aux[1] = 1;
65    specials[4] = aux;

    aux[0] = aux[1] = 0;
    aux[0] = 1;
    aux[2] = 1;
    specials[9] = aux;

    aux[0] = aux[2] = 0;
    aux[2] = 1;
70    aux[3] = 1;
    specials[40] = aux;

    aux[2] = aux[3] = 0;
    aux[2] = 1;
    aux[4] = 1;
80    specials[90] = aux;

    while(true)
    {
85        scanf( "%d", &num);

        if(!num)
            break;

        contI = contV = contX = contL = contC = 0;

        for(i = 1; i<=num; i++)
            generate(i, specials, &contI, &contV, &contX, &contL, &contC);
            ;

95        printf( "%d: %d i, %d v, %d x, %d l, %d c |n", num, contI, contV,
            contX, contL, contC);
    }

    return 0;
}

```

14.9 Sudoku

```

/*
 * Verifica se a matriz é solução para um problema de Sudoku
 */

```

```

5 #include <iostream>
6 #include <cstdio>
7 #include <cstring>

8 using namespace std;

10 int matriz[9][9];

11 bool sudoku(int lin, int col)
12 {
13     bool verifica[9];

14     for(int i = 0; i < 9; i++)
15         verifica[i] = false;

16     if(lin + 3 > 9 || col + 3 > 9)
17         return true;

18     for(int i = lin; i < lin + 3; i++)
19     {
20         for(int j = col; j < col + 3; j++)
21         {
22             if(!verifica[matriz[i][j] - 1])
23                 verifica[matriz[i][j] - 1] = true;
24             else
25                 return false;
26         }
27     }

28     return (sudoku(lin + 3, col) && sudoku(lin, col + 3));
29 }

30 int main()
31 {
32     int n;
33     scanf("%d", &n);

34     for(int i = 0; i < n; i++)
35     {
36         bool resposta = true;

37         for(int j = 0; j < 9; j++)
38         {
39             bool verifica[9];
40             for(int l = 0; l < 9; l++)
41                 verifica[l] = false;

42             for(int k = 0; k < 9; k++)
43             {
44                 scanf("%d", &matriz[j][k]);
45                 if(!verifica[matriz[j][k] - 1])
46                     verifica[matriz[j][k] - 1] = true;
47                 else resposta = false;
48             }
49         }
50     }
51 }
```

```

    for( int j = 0; j < 9; j++)
    {
        bool verifica[9];
        for( int l = 0; l < 9; l++)
            verifica[l] = false;

        for( int k = 0; k < 9; k++)
        {
            if(!verifica[matriz[k][j] - 1])
                verifica[matriz[k][j] - 1] = true;
            else resposta = false;
        }
    }
    printf( "Instancia %d\n", i + 1);

    if(!resposta)
    {
        printf( "NAO\n");
        continue;
    }

    resposta = sudoku(0, 0);

    if(resposta)
        printf( "SIM\n");
    else printf( "NAO\n");
}
return 0;
}

```

14.10 VETOR PARA REVERSÃO - DECODIFICAÇÃO

```

int trans[100];
int undo[100];

void nextStr(char* read, char* aux, int lim)
{
    int i;

    aux[lim] = '|0';

    for(i = 0; i<lim; i++)
    {
        aux[i] = read[trans[i]-1];
    }
}

void nextStr2(char* read, char* aux, int lim)
{
    int i;

    aux[lim] = '|0';

    for(i = 0; i<lim; i++)
    {
        aux[i] = read[undo[i]-1];
    }
}

```

```

25     }
}

int findFrequency(char* strSource, char* aux, char* aux2, int n)
{
    int i, freq = 0;

    for(i = 0; i < n; i++)
        aux[i] = (char)(i+1);

    while(1)
    {
        HERE:

        freq++;

        if(freq%2)
        {
            nextStr(aux, aux2, n);

            for(i = 0; i < n-1; i++)
                if(aux2[i] > aux2[i+1])
                    goto HERE;
        }
        else
        {
            nextStr(aux2, aux, n);

            for(i = 0; i < n-1; i++)
                if(aux[i] > aux[i+1])
                    goto HERE;
        }

        return freq;
    }
    return 0;
}

void undoTrans(int n)
{
    int i;

    for(i = 0; i < n; i++)
    {
        undo[trans[i]-1] = i+1;
    }
}

int main(int argc, char** argv)
{
    unsigned int i, j, k, n, m, freq, done; char c;
    char* read = (char*)malloc(3*(MAXSTRSZ+1)*sizeof(char));
    char* aux = &(read[MAXSTRSZ+1]);
    char* aux2 = &(aux[MAXSTRSZ+1]);

    while(1)
    {

```

```

        cin >> n >> m;

85      if(n == 0 && m == 0)
            break;

90      for(i = 0; i<n; i++)
            cin >> trans[i];

95      cin.getline(read, MAXSTRSZ);
      cin.getline(read, MAXSTRSZ);

100     if(2 + 2 != 4)
    {
        freq = findFrequence(read, aux, aux2, n);

        printf("freq == %d | n", freq);

105     done = m%freq;

        int tot = freq - done;
        for(i = 0; i<tot; i++)
    {
        if(!(i%2))
            nextStr(read, aux, n);
        else
            nextStr(aux, read, n);
    }

110     if(i%2)
            printf("%s | n", aux);
        else
            printf("%s | n", read);
    }
else
{
    undoTrans(n);

120     for(i = 0; i<m; i++)
    {
        if(!(i%2))
            nextStr2(read, aux, n);
        else
            nextStr2(aux, read, n);
    }

125     if(i%2)
            printf("%s | n", aux);
        else
            printf("%s | n", read);
    }
}

130     return 0;
}

```

14.11 Xadrez

```

/*Maximo de rainhas , cavalos , reis e torres em um tabuleiro*/

#include <stdio.h>
#include <algorithm>
5 #include <cmath>

using namespace std;

int rei(int x, int y)
10 {
    int cont = 0;

    for(int i = 0; i < x; i+= 2)
        for(int j = 0; j < y; j+= 2)
            cont++;

    return cont;
}

20 int cavalo(int x, int y)
{
    int cont = 0;
    int linhaPar = 0, linhaImpar = 0;

    for(int i = 1; i < y; i+=2)
        linhaImpar++;

    for(int i = 0; i < y; i+=2)
        linhaPar++;

30

    for(int i = 0; i < x; i++)
        cont += i%2?linhaImpar:linhaPar;
}
35
    return cont;
}

int main()
{
40    int t;
    scanf( "%d ", &t);

    for(int i = 0; i < t; i++)
    {
45        int m, n;
        char peca;

        scanf( " %c %d %d ", &peca, &m, &n);

        switch(peca)
50        {
            case 'r':
                printf( "%d | n", min(m, n));
                break;
            case 'Q':
                printf( "%d | n", min(m, n));
                break;
55        }
    }
}

```

```

    case 'K':
        printf("%d\n", rei(m, n));
        break;
    case 'k':
        printf("%d\n", cavalo(m, n));
    }
}
return 0;
}

```

15 Busca Exaustiva

15.1 8-Rainhas

```

vector< vector<int> > posicoes;
vector<int> pos(8, 0);

void backtracking(int posicao){
    register int i, j, k, l;
    int sobAtaque = 0;
    for(i=0; i<8 && !sobAtaque; i++){
        for(j=0; j<8 && !sobAtaque; j++) if(i != j){
            if(pos[i] == pos[j]) sobAtaque = 1;

10           if(sobAtaque == 0){
                if(j > i && pos[j] > pos[i]){
                    if(pos[i]-i == pos[j]-j) sobAtaque = 1;
                }
15           else if(j < i && pos[j] < pos[i]){
                    if(pos[i]+i == pos[j]+j) sobAtaque = 1;
                }
                else if(j > i && pos[j] < pos[i]){
                    if(pos[i]+i == pos[j]+j) sobAtaque = 1;
                }
20           else if(j < i && pos[j] > pos[i]){
                    if(pos[i]-i == pos[j]-j) sobAtaque = 1;
                }
            }
        }
25       }

if(sobAtaque == 0) posicoes.push_back(pos);

30   if(posicao >= 8) return;
    while(pos[posicao] < 8){
        backtracking(posicao+1);
        pos[posicao]++;
    }
35   pos[posicao] = 0;
}

int main(){
    int caso = 1;
40
}

```

```

register int i, j;
backtracking(0);
int minimo, atual;

45 while (scanf ("%d %d %d %d %d %d %d", &pos[0], &pos[1], &pos[2],
    &pos[3], &pos[4], &pos[5], &pos[6], &pos[7]) != EOF){
    for (i=0; i<8; i++) pos[i]--;

    minimo = INF;
    for (i=0; i<posicoes.size(); i++){
50        atual = 0;
        for (j=0; j<8; j++) if (pos[j] != posicoes[i][j])
            atual++;

        minimo = minimo < atual ? minimo : atual;
    }

    printf ("Case %d: %d\n", caso++, minimo);
}

60 return 0;
}

```

15.2 Crypt Kicker

```

typedef vector<string>::iterator It;

void imprimir(string a, map<char, char> mapeador){
    for (int i=0; i<a.length(); i++){
        if (a[i] >= 'a' && a[i] <= 'z') cout << mapeador[a[i]];
        else cout << a[i];
    }
}

10 void asteriscos(string a){
    for (int i=0; i<a.length(); i++){
        if (a[i] >= 'a' && a[i] <= 'z') cout << "*";
        else cout << a[i];
    }
}

15 bool compare(string a, string b){
    return (a.length() > b.length());
}

20 bool busca(vector<string> crypt, vector<string> dicionario, map<
    char, char> &mapeador, map<char, int> &usada, map<char, int> &
    usadaD, int n){
    if (n == crypt.size()) return true;

    bool perfeito;
25    for (int j=0; j<dicionario.size(); j++){
        if (dicionario[j].length() < crypt[n].length()) break;
        if (dicionario[j].length() != crypt[n].length()) continue;

```

```

30     perfeito = true;

    for(int k=0; k<dicionario[j].length(); k++){
        if(usada[crypt[n][k]] == -1 && usadaD[dicionario[j][k]] == 0)
        {
            usada[crypt[n][k]] = n;
            mapeador[crypt[n][k]] = dicionario[j][k];
            usadaD[dicionario[j][k]] = 1;
        }

        else if(mapeador[crypt[n][k]] != dicionario[j][k]){
            for(int l=0; l<k; l++) if(usada[crypt[n][l]] == n){
                usada[crypt[n][l]] = -1;
                usadaD[dicionario[j][l]] = 0;
            }
        }

        perfeito = false;
        break;
    }

50    if(perfeito){
        if(busca(crypt, dicionario, mapeador, usada, usadaD, n+1))
            return true;
        else
            for(int l=0; l<crypt[n].length(); l++) if(usada[crypt[n][l]] == n){
                usada[crypt[n][l]] = -1;
                usadaD[mapeador[crypt[n][l]]] = 0;
            }
    }

60    return false;
}

vector<string> tokenizer(char linha[]){
    char *token;
    vector<string> retorno;

    token = strtok(linha, " |n|r");
    while(token != NULL){
        retorno.push_back(string(token));
        token = strtok(NULL, " |n|r");
    }

    return retorno;
}

75 int main(){
    int n;
    string entrada;
    char backup[100];
    char linha[100];
    vector<string> dicionario;
    map<char, int> usada;
    map<char, int> usadaD;
}

```

```

85    cin >> n;

  for(int i=0; i<n; i++){
    cin >> entrada;
    dicionario.push_back(entrada);
  }

  sort(dicionario.begin(), dicionario.end(), compare);

95  fgets(linha, 5, stdin);

  while(fgets(linha, 100, stdin)){
    strcpy(backup, linha);

    map<char, char> mapeador;
    vector<string> crypt = tokenizer(linha);

    for(int i= 'a'; i<= 'z'; i++){
      usada[(char)i] = -1;
      usadaD[(char)i] = 0;
    }

105   sort(crypt.begin(), crypt.end(), compare);

    if(busca(crypt, dicionario, mapeador, usada, usadaD, 0))
      imprimir(string(backup), mapeador);
    else asteriscos(string(backup));
  }

  return 0;
}

```

16 Dividir para Conquistar

16.1 Roteadores

```

double minDist(int val, vector<double> &vec)
{
  double dist = 100000000000;
  for(int i = 0; i<vec.size(); i++)
  {
    double res = abs(vec[i] - val);
    if(res < dist)
      dist = res;
  }

10   return dist;
}

double maxDist(int ind, vector<int> &vec, int nDisp, double val)
15 {
  int i;

  vector<double> routers;
  double dist = -1;

```

```

20   if(nDisp)
21   {
22     routers.push_back(vec[ind] + val);
23     nDisp--;
24   }
25   for(i = 0; i<vec.size(); i++)
26   {
27     double res = minDist(vec[i], routers);
28
29     if(res > val)
30     {
31       if(nDisp)
32       {
33         routers.push_back(vec[i] + val);
34         dist = 0;
35         nDisp--;
36       }
37       else
38       {
39         dist = res;
40       }
41     }
42   }
43
44   return dist;
45 }

double buscaBinaria(vector<int> &vec, double limInf, double limSup,
        int nDisp)
{
  if(round(limInf*10) == round(limSup*10))
    return (double)(round(limInf*10)) / 10.;

  double mid = (limInf + limSup)/2;

  bool ok = false;
  for(int i = 0; i<vec.size(); i++)
  {
    if(maxDist(i, vec, nDisp, mid) <= mid)
    {
      ok = true;
      break;
    }
  }

  if(ok)
    return buscaBinaria(vec, limInf, mid, nDisp);

  return buscaBinaria(vec, mid, limSup, nDisp);
}

int main()
{
  long long i, j, k, nCases;
  char* read = (char*)malloc((MAXSTRSZ+1)*sizeof(char));
  char c;

  cin >> nCases;
}

```

```

for(long long cases = 0; cases < nCases; cases++)
{
    int nAccess, nHouses;
    vector<int> houses;

    cin >> nAccess >> nHouses;

    for(i = 0; i < nHouses; i++)
    {
        cin >> j;
        houses.push_back(j);
    }

    sort(houses.begin(), houses.end());
}

double limInf = 0, limSup = -1;

for(i = 0; i < nHouses - 1; i++)
{
    for(j = i + 1; j < nHouses; j++)
    {
        if(houses[j] - houses[i] > limSup)
            limSup = houses[j] - houses[i];
    }
}

double elem = buscaBinaria(houses, limInf, limSup, nAccess);

printf("%.1lf\n", elem);
}

return 0;
}

```

17 Estruturas de Dados

17.1 BigInteger – Krakovia

```

public class Main
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);

        int n, m;
        BigInteger tot, div;
        String help;
        int cont = 1;

        while(true)
        {
            n = sc.nextInt();
            m = sc.nextInt();

```

```

    if (n == 0 && m == 0)
        break;

20     tot = BigInteger.ZERO;

    for (int i = 0; i<n; i++)
    {
        tot = tot.add(sc.nextBigInteger());
    }

25     System.out.print("Bill #" + (cont++) + " costs " + tot + ":");
        System.out.println("each friend should pay " + tot.divide(BigInteger.valueOf(m)));
        System.out.println(" " + "| n | n");
    }
}
30 }
```

17.2 Pilha – Apaga

17.2.1 Especificação

Juliano é fã do programa de auditório Apagando e Ganhando, um programa no qual os participantes são selecionados através de um sorteio e recebem prêmios em dinheiro por participarem.

No programa, o apresentador escreve um número de N dígitos em uma lousa. O participante então deve apagar exatamente D dígitos do número que está na lousa; o número formado pelos dígitos que restaram é então o prêmio do participante.

Juliano finalmente foi selecionado para participar do programa, e pediu que você escrevesse um programa que, dados o número que o apresentador escreveu na lousa, e quantos dígitos Juliano tem que apagar, determina o valor do maior prêmio que Juliano pode ganhar.

Entrada

A entrada contém vários casos de teste. A primeira linha de cada caso de teste contém dois inteiros N e D ($1 \leq D < N \leq 10^5$), indicando a quantidade de dígitos do número que o apresentador escreveu na lousa e quantos dígitos devem ser apagados. A linha seguinte contém o número escrito pelo apresentador, que não contém zeros à esquerda.

O final da entrada é indicado por uma linha que contém apenas dois zeros, separados por um espaço em branco.

Saída

Para cada caso de teste da entrada seu programa deve imprimir uma única linha na saída, contendo o maior prêmio que Juliano pode ganhar.

Exemplo

4 2 3759 → 79
 6 3 123123 → 323
 7 4 1000000 → 100

17.2.2 Solução

```

char pilha[100010];

int main(int argc, char** argv)
{
  int n, m, i, j;
  char* read = (char*)malloc((MAXSTRSZ+1)*sizeof(char));

  while(1)
  {
    scanf( "%d%d ", &n, &m);

    if(!n && !m)
      break;

    scanf( "%s", read);

    if(n == m)
      printf( "0|n");
    else
    {
      int totElim = 0;
      int topo = -1;

      int curr;
      for(curr = 0; curr < n; curr++)
      {
        if(topo == -1)
        {
          topo++;
          pilha[topo] = read[curr];

          continue;
        }

        while(pilha[topo] < read[curr])
        {
          topo--;
          totElim++;
          if(totElim == m)
            break;
          if(topo == -1)
            break;
        }
        topo++;
        pilha[topo] = read[curr];

        if(totElim == m)
          break;
      }
      topo-=m-totElim;
    }
  }
}

```

```

        for(i = 0; i<=topo; i++)
            printf("%c", pilha[i]);
    for(curr++; curr < n; curr++)
        printf("%c", read[curr]);
    printf(" | n");
}
}

55
60
return 0;
}

```

17.3 Pilha – Estação

```

int main(int argc, char** argv)
{
    int i, j, k;

5     int read, nElem;

    bool change = true;
    int nChanges = 0;
    while(1)
10    {
        if(change)
        {
            cin >> nElem;

15        if(nChanges)
        {
            cout << endl;
        }

20        if(!nElem)
            break;

            change = false;
            nChanges++;
        }

25        cin >> read;
        if(!read)
        {
30            change = true;
            continue;
        }
        vector<int> target;

35        target.push_back(read);

        for(i = 1; i<nElem; i++)
        {
            cin >> read;
40            target.push_back(read);
        }
    }
}

```

```

queue<int> A;
for(i = 1; i<=nElem; i++)
    A.push(i);

stack<int> station;

bool issue = false;
for(i = 0; i<target.size(); i++)
{
    if(A.size())
    {
        if(A.front() <= target[i])
        {
            while(A.front() != target[i])
            {
                station.push(A.front());
                A.pop();
            }
            station.push(A.front());
            A.pop();
        }
    }
}

if(station.top() != target[i])
{
    issue = true;
    break;
}
station.pop();
}

if(issue)
    cout << "No" << endl;
else
    cout << "Yes" << endl;
}

return 0;
}

```

18 Grafos

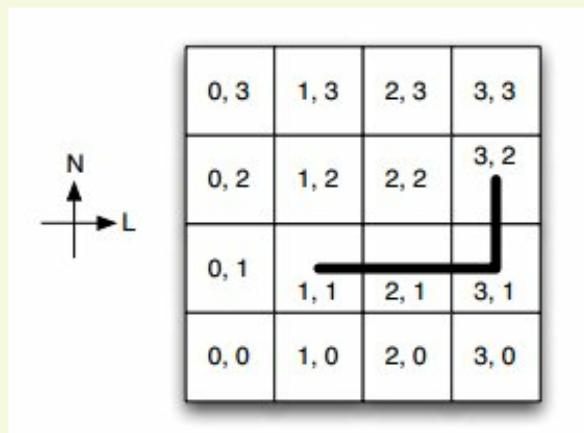
18.1 BFS with Levels – Busca ao Tesouro

18.1.1 Especificação

Capitão Tornado é um pirata muito cruel que faz qualquer coisa por dinheiro. Há alguns dias, o capitão soube da existência de um tesouro numa ilha deserta, e agora tenta determinar sua posição.

A ilha pode ser vista como um quadriculado $N \times N$ de terra cuja posição $(0, 0)$ está a sudoeste, a posição $(N-1, 0)$ está a sudeste, a posição $(0, N-1)$ está a noroeste e a posição $(N-1, N-1)$ está a nordeste. Em alguma posição desse quadriculado está o tesouro.

Uma curiosidade importante é a perna de pau que o capitão possui. Ela impede que o capitão se locomova em direções que não a horizontal ou a vertical: para ir da posição $(1, 1)$ para a posição $(3, 2)$, por exemplo, o capitão é obrigado a gastar três passos. É claro que o capitão sempre escolhe, dentro de suas limitações, um caminho com o menor número de passos possível. Chamamos esse modo de andar de passos de capitão. Um exemplo de caminho por passos de capitão entre $(1, 1)$ e $(3, 2)$ é ilustrado na figura a seguir.



Como em toda boa caça ao tesouro, o capitão não conhece a posição onde o tesouro se encontra: ele possui um mapa que corresponde à geografia da ilha. Em algumas posições desse mapa, existem pistas escritas. Cada pista consiste em um número D , que indica a menor distância em passos de capitão entre a posição em que a pista se encontra e a do tesouro.

	X		
3			4

	2		
X?	1	X?	
	2		
	3		

Observe que, dependendo da disposição das pistas, a posição do tesouro pode estar determinada de maneira única ou não. Na figura acima e à esquerda, as duas pistas são suficientes para se saber, com certeza, onde está o tesouro; na figura à direita, as quatro pistas dadas ainda possibilitam que tanto a posição $(0, 2)$ quanto a $(2, 2)$ guardem o tesouro. Nesse último caso, não se pode determinar, com certeza, qual é a localização do tesouro.

Dadas as pistas que o capitão possui, sua tarefa é determinar se as pistas fornecem a localização exata do tesouro e, caso positivo, qual ela é.

Entrada

A primeira linha contém dois inteiros positivos N e K , onde N é a dimensão do quadriculado e K é o número de pistas no mapa que o capitão possui.

Cada uma das próximas K linhas contêm três inteiros X , Y e D , informando que existe uma pista na posição (X, Y) contendo o número D . Essa pista indica que o tesouro encontra-se a D passos de capitão da posição da pista.

É garantido que, com essas pistas, existe ao menos uma localização possível para o tesouro. Além disso, o mapa não contém duas pistas na mesma posição.

Saída

Se as pistas forem suficientes para determinar com certeza a localização do tesouro, seu programa deve imprimir uma única linha com dois inteiros, X e Y , indicando que o tesouro encontra-se na posição (X, Y) .

Caso contrário, seu programa deve imprimir uma única linha com dois inteiros iguais a -1 , como nos exemplos de saída a seguir.

Restrições

- $2 \leq N \leq 100$
- $1 \leq K \leq 100$

Exemplos

Entrada

```
4 2
0 0 3
3 0 4
```

Saída

```
1 2
```

Entrada

```
4 4
1 0 3
1 1 2
1 2 1
1 3 2
```

Saída

```
-1 -1
```

Entrada

```
3 3
0 0 2
1 1 2
2 0 4
```

Saída

```
0 2
```

18.1.2 Solução

```
vector<bool> already;
vector< vector<int> > graph;

void BFS(int n, int source, int levInt, set<int> &p)
5
{
    int i, ext, lvl;

    queue<int> fila;
    queue<int> level;

10
    fila.push(source);
    level.push(0);

    already.clear();
15
    for(i = 0; i<n; i++)
        already.push_back(false);

    while(fila.size())
    {
20
        ext = fila.front(); fila.pop();
        lvl = level.front(); level.pop();

        if(lvl == levInt)
            p.insert(ext);
        if(lvl > levInt)
            break;

        if(already[ext])
        {
30
            continue;
        }

        already[ext] = true;

        for(i = 0; i<graph[ext].size(); i++)
        {
            if(!(already[graph[ext][i]]))
            {
40
                fila.push(graph[ext][i]);
                level.push(lvl+1);
            }
        }
    }
45
    int codifica(int a, int b, int c)
    {
        return a*c + b;
    }
50
    void decodifica(int v, int n, int* x, int* y)
    {
        *x = v/n;
        *y = v%n;
55
    }
}
```

```

void geraVizinhos(int x, int y, int n)
{
    int ref = codifica(x,y,n);
    vector<int> w;
    graph.push_back(w);

    if(x-1 >= 0)
    {
        graph[ref].push_back(codifica(x-1, y, n));
    }
    if(x+1 < n)
    {
        graph[ref].push_back(codifica(x+1, y, n));
    }
    if(y-1 >= 0)
    {
        graph[ref].push_back(codifica(x, y-1, n));
    }
    if(y+1 < n)
    {
        graph[ref].push_back(codifica(x, y+1, n));
    }
}

bool atAll(int val, vector< set<int> > &points)
{
    int i, j;

    for(vector< set<int> >::iterator it = points.begin(); it!=points.end(); it++)
    {
        bool isNot = true;

        for(set<int>::iterator iter = (*it).begin(); iter != (*it).end();
            () ; iter++)
        {
            if(*iter == val)
            {
                isNot = false;
                break;
            }
        }

        if(isNot)
            return false;
    }
}

return true;
}

int main()
{
    int dim, n, tot, sol, x, y;
    int i, j;

    scanf("%d%d", &dim, &n);
    int lim = dim*dim;
}

```

```

115     for(i = 0; i<dim; i++)
    {
        for(j = 0; j<dim; j++)
        {
            geraVizinhos(i, j, dim);
        }
    }

120    vector< set<int> > points;
    int v1, v2, level;
    for(i = 0; i<n; i++)
    {
        scanf( "%d%d%d", &v1, &v2, &level);

        set<int> x;
        BFS(lim, codifica(v1,v2,dim), level, x);
        points.push_back(x);
    }

130    tot = 0; sol = -1;
    for(set<int>::iterator it = points[0].begin(); it!=points[0].end()
        (); it++)
    {
        if(atAll(*it, points))
        {
            tot++;
            sol = *it;
            if(tot == 2)
                break;
        }
    }

140    if(tot == 1)
    {
        decodifica(sol, dim, &x, &y);
        printf( "%d %d\n", x, y);
    }
    else
        printf( "-1 -1\n");

        return 0;
}

```

18.2 BFS with Path

```

void BFS(map< int ,vector<int> > &graph, int n, int source, int dest
        , int* nPath)
{
    int i;

5     for(i = 0; i<n; i++)
    {
        already[i] = false;
    }

10    *nPath = 0;

```

```

queue<int> fila;

15   bool found = false;
    pred[source] = pred[dest] = -1;
    fila.push(source);
    already[source] = true;

20   while(!fila.empty())
{
    int ext = fila.front(); fila.pop();

    it2 = graph.find(ext);

25   for(it3 = (it2->second).begin(); it3 != (it2->second).end()
        ; it3++)
    {
        if(!already[*it3])
        {
            fila.push(*it3);
            already[*it3] = true;
            pred[*it3] = ext;

            if(*it3 == dest)
            {
                found = true;
                break;
            }
        }
    }
    if(found)
        break;
}

35   int next = dest;
40   if(pred[next] == -1)
{
    *nPath = 0;
}
else
{
    while(next != -1)
    {
        path[*nPath] = next;
        (*nPath)++;
        next = pred[next];
    }
}
50
}

```

18.3 Contour Paiting

18.3.1 Especificação

782 Contour Painting

A contour of points is represented on a two dimensional (2D) grid as illustrated in figure 1. The points of the contour are specified by the same character which can be any printable character, different than ‘*’, ‘#’, ‘_’ and space. In figure 1 this character is ‘X’. All the other points of the grid are represented by spaces. The contour is connected, i.e. any two points on the contour can be reached from one another by traveling vertically, horizontally and diagonally. Moreover, it is considered that a contour can close a single non empty zone of grid points.

```
012345678901234567890123456789
-----
0|      XXXXXXXXXXXX
1|      XXXX       XX
2|      X           X
3|      X           X
4|      X       X   XXXXXXXX
5|      XXXXXXXX           XX
6|      X       X   XXXXXXXX
7|      X           X
8|      XXXX       XX
9|      XXXXXXXXXXXX
0|
```

Figure 1: A contour on a 2D grid

The character ‘#’ represents the colour used to paint the contour as illustrated in figure 3. The paint is added on one side of the contour in such a way that each contour point of the painted side has at least one ‘#’ neighbour horizontally or vertically as shown in figure 2:

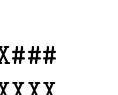
		
flat zone	concave corner	convex corner

Figure 2: Cases of point painting

A contour can be painted either from inside or from outside. The painting side is specified by the presence of the character ‘*’ inside or outside the contour as shown in figure 3. Notice that the star is removed from the grid once the painting is done.

<pre> XXXXXXXXXXXX XXXX XX X * X X X XXXXXX XXXXXXX XX X X XXXXXX X X XXXX XX XXXXXXXXXX </pre>	<pre> XXXXXXXXXXXX XXXX#####XX X### # ##X X#####X# #XXXXXX XXXXXXX# #####XX X#####X# #XXXXXX X### # ##X XXXX#####XX XXXXXXXXXX </pre>	interior painting
<pre> * XXXXXXXXXX XXXX XX X X X X XXXXXX XXXXXXX XX X X XXXXXX X X XXXX XX XXXXXXXXXX </pre>	<pre> ##### #XXXXXXX### #XXXXX XX# #X X##### #X X XXXXXX## #XXXXXXX XX# #X X XXXXXX## #X X##### #XXXXX XX# #XXXXXXX### ##### </pre>	exterior painting
before painting	after painting	

Figure 3: Painting a closed contour

Your problem is to write a program which: reads from a text file a number n and n grids, each grid containing a single contour and a single star, paints each grid according to the position of the star and outputs the painted grids to a text file. Each contour is placed on its grid in such a way that it is fully surrounded by free grid points (spaces).

Input

The first line of the input text file contains the number of grids to be painted. The next lines of the file contain the grids. The lines which represent a grid are terminated by a separation line full of underscores ('_'). There are at most 30 lines and at most 80 characters in a line for each grid. The lines can be of different length.

Output

The standard output file contains the grids with the painted contours and with the stars removed. Each grid is output in the same format it has been read from the input file, including the separation line. It follows an example of the input and the output of the program for a single simple contour.

Sample Input

1

XXXXXXX
X * X
XXXXXXX

Sample Output

XXXXXXX
X####X
XXXXXXX

18.3.2 Solução

```
bool already[MAXVERT];
map< int, vector<int> >::iterator it1;
vector<int>::iterator it2;

5 void decodifica(int code, int* x, int l, int* y, int c)
{
    *x = code/c;
    *y = code%c;
}

10 int BFS(map< int, vector<int> > &graph, int n, int source, char** mat,
         int l, int c)
{
    int i, x, y;

15    for(i = 0; i < n; i++)
    {
        already[i] = false;
    }

20    queue<int> fila;

        fila.push(source);
        already[source] = true;

25    while(!fila.empty())
    {
        int ext = fila.front(); fila.pop();

        for(it2 = graph[ext].begin(); it2 != graph[ext].end(); it2++)
30        {
            if(!already[*it2])
            {
                decodifica(*it2, &x, l, &y, c);
                if(mat[x][y] == 'X')
                {
                    decodifica(ext, &x, l, &y, c);
                    mat[x][y] = '#';
                }
                else
                {
                    fila.push(*it2);
                    already[*it2] = true;
                }
            }
45        }
    }

46    int codifica(int x, int l, int y, int c)
    {
        return x*c + y;
    }

50    bool onlyUnders(char* str)
```

```

55  {
56      int i;
57      if(!str[i])
58          return false;
59      for(i = 0; str[i]; i++)
60      {
61          if(str[i]!='_')
62              return false;
63      }
64      return true;
65  }

66 void removeQuebra(char* str)
67 {
68     int i;
69     for(i = 0; str[i]!='\n'; i++); str[i] = '\0';
70 }

71 void removeEndBlanks(char* str)
72 {
73     int i = strlen(str)-1;
74     while(str[i]==' ')
75     {
76         str[i] = '\0';
77         i--;
78         if(i<0)
79             break;
80     }
81 }

82 int main(int argc, char** argv)
83 {
84     int i, j, k;

85     char** mat = (char**)malloc(100*sizeof(char*));
86     mat[0] = (char*)malloc(100*102*sizeof(char));
87     for(i = 1; i<100; i++)
88         mat[i] = &(mat[0][i*100]);
89     char* read = &(mat[0][i*100]);
90     char* und = &(mat[0][i*100]);

91     int line, col;
92     int nCases, iC;

93     int startAt;

94     fgets(read, 99, stdin);
95     nCases = atoi(read);

96     for(iC = 0; iC<nCases; iC++)
97     {
98         line = 0;

99         map< int ,vector<int> > graph;

100        col = 0;
101        while(1)
102        {

```

```

        fgets(mat[line], 99, stdin); removeQuebra(mat[line]);

115    if(onlyUnders(mat[line]))
    {
        break;
    }

120    int taml = strlen(mat[line]);
    if(taml > col)
        col = taml;

        line++;
125    }
    col++;

    for(i = 0; i<line; i++)
    {
        for(j = strlen(mat[i]); j<col; j++)
        {
            mat[i][j] = ' ';
        }
        mat[i][j] = '|0';
    }

135    for(i = 0; i<line; i++)
    {
        for(j = 0; j<col; j++)
        {
            if(mat[i][j] == 'X')
                continue;

            if(mat[i][j] == '*')
            {
                startAt = codifica(i, line, j, col);
                mat[i][j] = ' ';
            }
        }

145        int ref = codifica(i, line, j, col);

        if(i-1>=0)
            graph[ref].push_back(codifica(i-1, line, j, col));
        if(i+1<line)
            graph[ref].push_back(codifica(i+1, line, j, col));
        if(j-1>=0)
            graph[ref].push_back(codifica(i, line, j-1, col));
        if(j+1<col)
            graph[ref].push_back(codifica(i, line, j+1, col));
    }

155    }
}

160    BFS(graph, line*col, startAt, mat, line, col);

    for(i = 0; i<line; i++)
    {
        removeEndBlanks(mat[i]);
        printf("%s |n", mat[i]);
    }
165    printf("%s |n", mat[line]);

```

```

        }

    return 0;
}

```

18.4 Dominó – BFS em grafo direcionado

```

bool already[MAXVERT];
bool explored[MAXVERT];
int father[MAXVERT];

5   void decodifica(int code, int* x, int l, int* y, int c)
{
    *x = code/c;
    *y = code%c;
}

10  int BFS(map<int,vector<int> > &graph, int n, int source)
{
    int i, x, y;

15    for(i = 0; i<=n; i++)
    {
        if(!explored[i])
            already[i] = false;
        else
            already[i] = true;
    }

    queue<int> fila;

25    fila.push(source);
    already[source] = true;
    explored[source] = true;

    while(!fila.empty())
    {
        int ext = fila.front(); fila.pop();

        for(vector<int>::iterator it2 = graph[ext].begin(); it2 != graph[ext].end(); it2++)
        {
35            if(!already[*it2])
            {
                fila.push(*it2);
                already[*it2] = explored[*it2] = true;
            }
        }
    }

45    int findFather(int i)
    {
        int next = i;

        while(1)

```

```

50     {
51         if(father[next] == next)
52             return next;
53
54         next = father[next];
55     }
56
57     int main(int argc, char** argv)
58     {
59         int i, j, k;
60
61         int iCases, nCases;
62         int nTiles;
63         int iLines, nLines;
64
65         scanf( "%d", &nCases);
66
67         for(iCases = 0; iCases < nCases; iCases++)
68         {
69             map< int ,vector<int> > graph;
70
71             scanf( "%d%d", &nTiles, &nLines);
72
73             for(i = 0; i<=nTiles; i++)
74                 father[i] = i;
75
76             for(iLines = 0; iLines < nLines; iLines++)
77             {
78                 int x, y;
79
80                 scanf( "%d%d", &x, &y);
81
82                 graph[x].push_back(y);
83                 father[y] = father[x];
84             }
85
86             for(i = 0; i<=nTiles; i++)
87                 explored[i] = false;
88
89             int cont = 0;
90             for(i = 1; i<=nTiles; i++)
91             {
92                 if(!explored[i])
93                 {
94                     cont++;
95                     BFS(graph, nTiles, findFather(i));
96                 }
97             }
98
99             printf( "%d\n", cont);
100        }
101
102        return 0;
103    }

```

18.5 Escalonamento Ótimo – Ordenação Topológica

18.5.1 Especificação

O SBC (*System for Batch Computing*) é um sistema operacional voltado para a execução sequencial de tarefas. O operador do sistema cria tarefas e o sistema operacional é responsável por agendar a execução destas tarefas.

Cada tarefa pode depender da conclusão de algumas tarefas para poder começar. Se uma tarefa A depende de uma tarefa B , a tarefa B deve terminar antes que a tarefa A inicie sua execução.

Além disto, cada tarefa possui uma prioridade. É sempre mais vantajoso para o sistema começar executando uma tarefa de mais alta prioridade, depois continuar executando uma tarefa de mais alta prioridade dentre as que sobraram e assim por diante.

Neste problema, será dado um inteiro N , que irá representar o número de tarefas no sistema. As tarefas serão numeradas de 0 até $N-1$. Tarefas com índice menor possuem prioridade maior, de forma que a tarefa 0 é a tarefa de mais alta prioridade, a tarefa 1 é a tarefa com a segunda maior prioridade e assim por diante, até a tarefa $N-1$, que é a tarefa com a menor prioridade. Além disso, serão dadas M relações de dependência entre as tarefas.

Seu objetivo será decidir se é possível executar as tarefas em alguma ordem. Caso seja possível, você deverá produzir uma ordem de execução ótima para as tarefas, isto é, desempate as ordens possíveis pela prioridade da primeira tarefa. Se o empate ainda persistir, desempate pela prioridade da segunda tarefa, e assim por diante.

Entrada

A primeira linha da entrada contém inteiros N e M . As próximas M linhas descrevem, cada uma, uma dependência entre as tarefas da entrada. Cada uma dessas linhas irá conter dois inteiros A e B que indicam que a tarefa B depende da tarefa A , isto é, que a tarefa A deve terminar antes que a tarefa B inicie.

Saída

Se não for possível ordenar as tarefas de forma que as dependências sejam satisfeitas, imprima uma única linha contendo o caractere "*". Caso contrário, imprima N linhas contendo cada uma um número inteiro. O inteiro na i -ésima linha deve ser o índice da i -ésima tarefa a ser executada na ordem ótima de execução das tarefas.

Restrições

- $0 \leq N \leq 50000$.
- $0 \leq M \leq 200000$.
- $0 \leq A, B < N$.

Exemplos

Entrada

3 1
2 0

Saída

1
2
0

Entrada

2 2
0 1
1 0

Saída

*

18.5.2 Solução

```
#define MAX 51000
#define MMAX 201000

5   struct Node
{
    int v;
    Node * nxt;
};

10  Node pilha[MMAX];
int pi;

Node * aloca(int v, Node * nxt)
{
15    pilha[pi].v=v;
    pilha[pi].nxt=nxt;
    return &pilha[pi++];
}

20  Node * g[MAX];
int gr[MAX];
int resp[MAX];
int nresp;
set<int> pq;

25  int main()
{
    int n,m;
    int i;
    int a,b;

30    scanf( "%d %d", &n ,&m);

    pi=0;
    nresp=0;

40    for(i=0;i<n;++i)
    {
        gr[i]=0;
        g[i]=NULL;
    }

45    for(i=0;i<m;++i)
    {
        scanf( "%d %d", &a ,&b);
        ++gr[b];
        g[a]=aloca(b,g[a]);
    }

50    for(i=0;i<n;++i)
        if(gr[i]==0)
            pq.insert(i);

55    while(!pq.empty())
    {
        int now = *pq.begin();
```

```

    pq.erase(pq.begin());
    resp[nresp++]=now;

60   for(Node * ar=g[now]; ar!=NULL; ar=ar->nxt)
      if((--gr[ar->v])==0)
          pq.insert(ar->v);
    }

65   if(nresp<n)
      printf("*|n");
    else
      for(i=0;i<n;++i)
        printf("%d|n",resp[i]);
70
    return 0;
}

```

18.6 Floyd-Warshall

```

void fw(int n)
{
    int i, j, k;

5   for(k = 1; k<=n; k++)
    {
        for(i = 1; i<=n; i++)
        {
            for(j = 1; j<=n; j++)
            {
                dist[i][j] = menor(dist[i][j], dist[i][k] + dist[k][j]);
            }
        }
    }

15
int main()
{
    int nV, nA, orig, dest, i, j, iMin, min, max, cont = 1;
20
    while(1)
    {
        scanf("%d", &nV);

        if(!nV)
            break;

        nA = nV-1;

30        for(i = 1; i<=nV; i++)
            for(j = 1; j<=nV; j++)
                dist[i][j] = INFINITO;

        for(i = 0; i<nA; i++)
    {
        scanf("%d%d", &orig, &dest);
        dist[orig][dest] = dist[dest][orig] = 1;
    }
}

```

```

        }
        for(i = 1; i<=nV; i++)
            dist[i][i] = 0;

        fw(nV);

        min = 10000000;
45
        for(i = 1; i<=nV; i++)
        {
            max = -1;

            for(j = 1; j<=nV; j++)
                if(dist[i][j] > max)
                    max = dist[i][j];

                if(max < min)
                {
                    min = max;
                    iMin = i;
                }
            }

60
        printf( "Teste %d | n%d | n| n", cont++, iMin);
    }

    return 0;
65
}

```

18.7 Geração de Valores com BFS

```

/*
 * Número mínimo de moedas
 * Resolução utilizando BFS
 */
5
#include <iostream>
#include <vector>
#include <cstdio>
#include <queue>
10 #include <climits>

using namespace std;

int numMoedas[50001];
15 int moedas[101];

int minMoedas(int valor, int n)
{
    numMoedas[0] = 0;
20
    for (int i = 1; i <= valor; i++)
        numMoedas[i] = INT_MAX;

    queue<int> q;
25     q.push(0);

```

```

    while (!q.empty())
    {
        int atual = q.front();
        q.pop();

        if (atual == valor)
            break;

        int novoNumeroMoedas = numMoedas[atual] + 1;

        for (int i = 0; i < n; i++)
        {
            int novoValor = atual + moedas[i];
            if (novoValor <= valor && novoNumeroMoedas < numMoedas[novoValor])
            {
                numMoedas[novoValor] = novoNumeroMoedas;
                q.push(novoValor);
            }
        }
    }

    return numMoedas[valor];
}

int main()
{
    int m;
    scanf("%d", &m);

    while (m)
    {
        int n;
        scanf("%d", &n);

        for (int i = 0; i < n; i++)
            scanf("%d", &moedas[i]);

        int resposta = minMoedas(m, n);

        if (resposta == INT_MAX)
            printf("Impossible\n");
        else
            printf("%d\n", resposta);

        scanf("%d", &m);
    }

    return 0;
}

```

18.8 Knight Moves

```

bool graph[64][64];
int dist[64][64];

```

```

bool already[64];
queue<int> fila;
queue<int> level;

int code(int x, int y)
{
    return x*8 + y;
}

void decode(int val, int* x, int* y)
{
    *x = val/8;
    *y = val%8;
}

void removeBlanks(char* str, char* newStr)
{
    int i, k=0;
    for(i = 0; str[i]; i++)
    {
        if(str[i]!=' ' && str[i]!='\n')
            newStr[k++] = str[i];
    }
    newStr[k] = '\0';
}

void calculaDistancias(int i)
{
    int aval, lvl, j;

    for(j = 0; j<64; j++)
        already[j] = false;
    fila.push(i);
    level.push(0);

    already[i] = true;
}

while(!fila.empty())
{
    aval = fila.front(); fila.pop();
    lvl = level.front() + 1; level.pop();

    for(j = 0; j<64; j++)
    {
        if(graph[aval][j])
        {
            if(!already[j])
            {
                dist[i][j] = lvl;
                dist[j][i] = lvl;
                fila.push(j);
                level.push(lvl);
                already[j] = true;
            }
        }
    }
}

```

```

} int main()
{
    int i, j, k;
    for(i = 0; i<64; i++)
        for(j = 0; j<64; j++)
    {
        dist[i][j] = -1;
        graph[i][j] = false;
    }
    int x, y;

    for(i = 0; i<8; i++)
    {
        for(j = 0; j<8; j++)
        {
            int first = code(i,j);
            x = i-1;
            if(x>=0)
            {
                y = j-2;
                if(y>=0)
                    graph[first][code(x,y)] = true;
                y = j+2;
                if(y<8)
                    graph[first][code(x,y)] = true;
            }
            x = i-2;
            if(x>=0)
            {
                y = j-1;
                if(y>=0)
                    graph[first][code(x,y)] = true;
                y = j+1;
                if(y<8)
                    graph[first][code(x,y)] = true;
            }
            x = i+1;
            if(x<8)
            {
                y = j-2;
                if(y>=0)
                    graph[first][code(x,y)] = true;
                y = j+2;
                if(y<8)
                    graph[first][code(x,y)] = true;
            }
            x = i+2;
            if(x<8)
            {
                y = j-1;
                if(y>=0)
                    graph[first][code(x,y)] = true;
                y = j+1;
                if(y<8)
                    graph[first][code(x,y)] = true;
            }
        }
    }
}

```

```

120           if (y>=0)
121             graph[first][code(x,y)] = true;
122
123             y = j+1;
124             if (y<8)
125               graph[first][code(x,y)] = true;
126           }
127     }
128
129   for (i = 0; i<64; i++)
130     dist[i][i] = 0;
131
132   char* read = (char*)malloc(10*sizeof(char));
133   char* aux = (char*)malloc(10*sizeof(char));
134
135   int x1, y1, x2, y2;
136   int source, destiny;
137
138   string v1, v2;
139   while (cin >> v1 && cin >> v2)
140   {
141     removeBlanks(aux, read);
142
143     x1 = v1[0] - 'a';
144     y1 = v1[1] - '1';
145     x2 = v2[0] - 'a';
146     y2 = v2[1] - '1';
147
148     source = code(x1,y1);
149     destiny = code(x2,y2);
150
151     if (dist[source][destiny] == -1)
152       calculaDistancias(source);
153
154     printf("To get from %c%c to %c%c takes %d knight moves.\n",
155           v1[0], v1[1], v2[0], v2[1], dist[source][destiny]);
156   }
157
158   return 0;
159 }
```

18.9 Ordenação Topológica

```

/* Ordenacao topologica */

#include <iostream>
#include <vector>
5  #include <map>
#include <stdio.h>

using namespace std;

10 vector<int> ord;

/* Grafo usando lista de adjacencias */

```

```

bool ordTop(vector< vector<int > &graph){
    int colocados = 0;
    bool modificado;
    vector<int> mapeador(graph.size());

    for(int i=0; i<graph.size(); i++)
        mapeador[i] = graph[i].size();
    /* Mapeador guarda quantos filhos tem cada no, que inicialmente
     * é o tamanho do vector dele */
    while(colocados != graph.size()){
        modificado = true;
        for(int i=0; i<graph.size(); i++){
            if(mapeador[i] == 0){
                ord.push_back(i);
                mapeador[i] = -1;
                colocados++;
                modificado = false;
                for(int j=0; j<graph.size(); j++)
                    for(int k=0; k<graph[j].size(); k++)
                        if(graph[j][k] == i){
                            mapeador[j]--;
                            break;
                        }
                }
            }
        if(modificado) return false;
    }

    return true;
}

int main()
{
    int n, cont = 1;

    while(scanf( "%d ", &n)==1 && n)
    {
        map<string, int> Mpessoas;
        vector<string> pessoas;
        vector<vector<int> > grafo(n);

        for(int i = 0; i < n; i++)
        {
            string nome;
            cin>>nome;
            pessoas.push_back(nome);
            Mpessoas.insert(pair<string, int(nome, i));
        }

        for(int i = 0; i < n; i++)
        {
            string tmp;
            int u, m;

            cin>>tmp;
        }
    }
}

```

```

70     scanf( "%d ", &m);
    u = Mpessoas[tmp];

    for(int j = 0; j < m; j++)
    {
        string tmp2;
        int v;

        cin>>tmp2;

        v = Mpessoas[tmp2];

        grafo[v].push_back(u);
    }
}

85 ord.clear();

bool verifica = ordTop(grafo);
printf("Teste %d\n", cont++);

90 if(verifica)
{
    for(int i = ord.size() - 1; i >= 0; i--)
    {
        if(i < ord.size() - 1)
            printf(" ");
        cout<<pessoas[ord[i]];
    }
    cout<<endl;
}

100 }

105 else
    printf("impossivel\n\n");
}
return 0;
}

```

18.10 Ordenação Topológica – Min e Max

18.10.1 Gabriel's

```

/* Ordenação Topológica (max e min) – Agendamento de Tarefas*/

5 #include <iostream>
#include <cstdio>
#include <vector>
#include <algorithm>

10 using namespace std;

typedef struct a{
    int mini;
    int minIndice;
    int maxi;
    int mapa;
    int mapa2;
}

```

```

    int duracao;
}Tarefa;

vector<Tarefa> ord;
20 int graph[1001][1001];

void ordTopMin(int n){
    int colocados = 0;
    register int i, j;
25
    while(colocados != n)
        for(i=0; i<n; i++)
            if(ord[i].mapa == 0){
                ord[i].mapa--;
                colocados++;
30

                for(j=0; j<n; j++)
                    if(i != j)
                        if(graph[j][i] == 1){
                            ord[j].mapa--;
                            ord[j].mini = max(ord[j].mini, ord[i].mini+ord[i].
                                duracao);
                        }
                }
}
35

40 int ordTopMax(int n){
    int maior = -1;
    int definidos = 0;
    register int i, j;

    for(i=0; i<n; i++)
        maior = max(maior, ord[i].mini + ord[i].duracao);

    for(i=0; i<n; i++)
        if(ord[i].mapa2 == 0)
            ord[i].maxi = maior-ord[i].duracao;
50
    while(definidos != n)
        for(i=0; i<n; i++)
            if(ord[i].mapa2 == 0){
                ord[i].mapa2--;
                definidos++;

                for(j=0; j<n; j++)
                    if(i != j)
                        if(graph[i][j] == 1){
                            ord[j].mapa2--;
                            ord[j].maxi = min(ord[j].maxi, ord[i].maxi-ord[j].
                                duracao);
                        }
                }
}
55

65 return maior;
}

int main(){
    int n, in, in2;
    register int i, j;
70 Tarefa aux;

```

```

aux.mini = 0;
aux.maxi = 99999;
aux.mapa2 = 0;

75   while(scanf( "%d", &n) && n){
        ord.clear();

        for(i=0; i<n; i++){
            ord.push_back(aux);
            for(j=0; j<n; j++) graph[i][j] = 0;
        }

        for(i=0; i<n; i++){
            scanf( "%d", &in);
            scanf( "%d%d", &ord[in].duracao, &ord[in].mapa);

            for(j=0; j<ord[in].mapa; j++){
                scanf( "%d", &in2);
                graph[in][in2] = 1;
                ord[in2].mapa2++;
            }
        }

        ordTopMin(n);
        printf( "Prazo: %d dias | n", ordTopMax(n));
        for(i=0; i<n; i++)
            printf( "Tarefa # %d: min=%d, max=%d | n", i, ord[i].mini, ord[i].maxi);
        printf( "---| n");
    }

    return 0;
}

```

18.11 Pontos de Articulação

```

/* Ponto de articulação - Manut */

#include <iostream>
#include <cstdio>
5  #include <cstdlib>
#include <set>
#include <algorithm>

using namespace std;

10 #define MAX 410
#define DEBUG (cout << "Here" << endl)
#define Iter set<int>::iterator

15 int grafo[MAX][MAX];
set<int> art;
int visitado[MAX];
int n, tam;

20 int dfs(int u){

```

```

    int filhos = 0;
    visitado[u] = tam++;
    int menor = visitado[u];

25   for(int i = 0; i < n; i++) if(grafo[u][i] == 1){
        if(visitado[i]==0){
            filhos++;
            int m = dfs(i);
            menor = min(menor,m);
        }
        if(visitado[u]<=m && (u!=0 || filhos>=2)) art.insert(u+1);
    }

35   else menor = min(menor, visitado[i]);
}

        return menor;
}

40 int main(){
    int m, caso=1;
    int in1, in2;

    while(scanf("%d %d", &n, &m) && n+m != 0){
        for(int i=0; i<n; i++)
            for(int j=0; j<n; j++)
                grafo[i][j] = 0;

        for(int i=0; i<m; i++){
            scanf("%d%d", &in1, &in2);
            in1--;
            in2--;
            grafo[in1][in2] = 1;
            grafo[in2][in1] = 1;
        }
    }

        tam = 1;
        for(int i=0; i<n; i++)
            visitado[i] = 0;
60

        art.clear();
        dfs(0);

        printf("Teste %d\n", caso++);
        bool inicio = true;
        for(Iter it = art.begin(); it != art.end(); ++it){
            if(!inicio) printf(" ");
            inicio = false;
            printf("%d", (*it));
        }
        if(art.size() == 0) printf("nenhum");
        printf("\n\n");
    }

75    return 0;
}

```

18.12 Pontos de Articulação em Grafos Desconexos

```
/* Ponto de articulação - Grafo desconexo */

#include <iostream>
#include <cstdio>
5 #include <cstdlib>
#include <set>
#include <vector>
#include <map>
#include <string>
10 #include <algorithm>

using namespace std;

#define MAX 120
15 #define DEBUG (cout << "Here" << endl)
#define Iter set<string>::iterator

set<string> art;
int visitado[MAX];
20 int n, tam, inicial;

int dfs(vector< vector<int> > &grafo, int u, vector<string> &ruas){
    int filhos = 0;
    visitado[u] = tam++;
    25 int menor = visitado[u];

    for(int i = 0; i<grafo[u].size(); i++){
        if(visitado[grafo[u][i]]==0){
            filhos++;
            30 int m = dfs(grafo, grafo[u][i], ruas);
            menor = min(menor,m);

            if(visitado[u]<= m && (u!=inicial || filhos>=2)) art.insert(
                ruas[u]);
        }
    }

    35 else menor = min(menor, visitado[grafo[u][i]]);
}
    return menor;
}

40 int main(){
    int m, caso=1;
    string no1, no2;

    while(scanf("%d", &n) && n){
        vector<string> ruas(n);
        map<string, int> ruasInt;
        vector <vector<int> > grafo(n);

        45 for(int i=0; i<n; i++){
            cin >> ruas[i];
            ruasInt[ruas[i]] = i;
            visitado[i] = 0;
        }
    }

    50
```

```

        scanf( "%d", &m);

        if(caso != 1) printf( " | n");

60      for(int i=0; i<m; i++){
        cin >> no1 >> no2;
        grafo[ruasInt[no1]].push_back(ruasInt[no2]);
        grafo[ruasInt[no2]].push_back(ruasInt[no1]);
    }

65      art.clear();
    for(int i=0; i<n; i++){
        tam = 1;
        if(visitado[i] == 0){
            inicial = i;
            dfs(grafo, i, ruas);
        }
    }

75      printf( " City map # %d : %ld camera(s) found | n", caso++, art.size
        ());
    }

        for(Iter it = art.begin(); it != art.end(); ++it){
            printf( "%s | n", it->c_str());
        }
    }

80      return 0;
}

```

19 Gulosos

19.1 Packs 6x6

```

int dimensao[7];

typedef struct
{
5    int capacity;
    bool available[7];
    int n2, n3;
} TCont;

10   bool noProblem(int key, TCont &cont)
{
    if(cont.capacity < dimensao[key])
        return false;

    if(!cont.available[key])
        return false;

    if(key == 2)
    {
20        if(cont.n3 == 1)
        {

```

```

        if(cont.n2 >= 5)
        {
            return false;
        }
    }
    if(cont.n3 == 2)
    {
        if(cont.n2 >= 3)
        {
            return false;
        }
    }
    if(cont.n3 == 3)
    {
        if(cont.n2 >= 1)
        {
            return false;
        }
    }
}
if(key == 3)
{
    if(cont.n3 == 0)
    {
        if(cont.n2 > 5)
        {
            return false;
        }
    }
    if(cont.n3 == 1)
    {
        if(cont.n2 > 3)
        {
            return false;
        }
    }
    if(cont.n3 == 2)
    {
        if(cont.n2 > 1)
        {
            return false;
        }
    }
}
return true;
}

void addition(int key, TCont* cont)
{
    cont->capacity -= dimensao[key];

    switch(key)
    {
        case 2: cont->available[6] = cont->available[5] = false; break;
        case 3: cont->available[6] = cont->available[5] = cont->available
                  [4] = false; break;
    }
}

```

```

    case 4: cont->available[6] = cont->available[5] = cont->available
        [4] = cont->available[3] = false; break;
    case 5: cont->available[6] = cont->available[5] = cont->available
        [4] = cont->available[3] = cont->available[2] = false; break;
    case 6: cont->available[6] = cont->available[5] = cont->available
        [4] = cont->available[3] = cont->available[2] = cont->
        available[1] = false; break;
}

if(key == 2)
    cont->n2++;
else if(key == 3)
    cont->n3++;
}

void iniciaContainer(TCont* cont)
{
    int i;
    cont->capacity = 36;
    for(i = 0; i<7; i++)
        cont->available[i] = true;
    cont->n2 = cont->n3 = 0;
}

int main(int argc, char** argv)
{
    int i, j, k, k1, k2, k3, k4, k5, k6;
    char* read = (char*)malloc((MAXSTRSZ+1)*sizeof(char));

    dimensao[1] = 1;
    dimensao[2] = 4;
    dimensao[3] = 9;
    dimensao[4] = 16;
    dimensao[5] = 25;
    dimensao[6] = 36;

    while(1)
    {
        vector<int> add;
        vector<TCont> containers;

        cin >> k1 >> k2 >> k3 >> k4 >> k5 >> k6;

        if(k1 == 0 && k2 == 0 && k3 == 0 && k4 == 0 && k5 == 0 &&
           k6 == 0)
            break;

        for(i = 0; i<k6; i++)
            add.push_back(6);
        for(i = 0; i<k5; i++)
            add.push_back(5);
        for(i = 0; i<k4; i++)
            add.push_back(4);
        for(i = 0; i<k3; i++)
            add.push_back(3);
        for(i = 0; i<k2; i++)
            add.push_back(2);
        for(i = 0; i<k1; i++)

```

```

        add.push_back(1);

    for(i = 0; i<add.size(); i++)
    {
        bool foundOne = false;

        for(j = 0; j<containers.size(); j++)
        {
            if(noProblem(add[i], containers[j]))
            {
                foundOne = true;
                addition(add[i], &(containers[j]));
                break;
            }
        }

        if(!foundOne)
        {
            TCont x;
            inicialContainer(&x);

            addition(add[i], &x);
            containers.push_back(x);
        }
    }

    int tam = containers.size();
    printf("%d\n", tam);
}

return 0;
}

```

20 Matemática

20.1 Geometry

20.1.1 Coordenadas Geográficas

```

/* Tunneling de Earth - Coordenadas Geograficas , distancia */

#include <stdio.h>
#include <stdlib.h>
5 #include <math.h>

#define PI 3.14159265358
#define RAI0 6371009
#define toRad(x) x*=(PI/180)

10 typedef struct p{
    double lat, lon;
}TPonto;

15 double angulo(TPonto a, TPonto b){
    toRad(a.lon);
    toRad(b.lon);
}

```

```

    toRad(a.lat);
    toRad(b.lat);
20   double beta = fabs(a.lon-b.lon);

    beta = sin(a.lat) * sin(b.lat) + cos(a.lat) * cos(b.lat) * cos(
        beta);
    return acos(beta);
}

25

int main(){
    TPonto a, b;
    int n;
30   double dist;
    double degree, disArc;

    scanf( "%d ", &n);

    while(n--){
        scanf( "%lf %lf %lf %lf ", &a.lat, &a.lon, &b.lat, &b.lon);

        degree = angulo(a, b);
        dist = sqrt(2 - 2*cos(degree))*RAIO;
        disArc = degree*RAIO;

        printf( "%0.1f | n", disArc-dist);
    }

45   return 0;
}

```

20.1.2 Distância de Manhattan

```

int n;

int absol(int a)
{
5   if(a < 0)
      a = -a;
   return a;
}

10 int manhattan(int ax, int ay, int bx, int by)
{
   return absol(ax-bx) + absol(ay-by);
}

15 int main()
{
   int k;
   vector<pair<int, int> > posicoes;

20   scanf( "%d %d ", &n, &k);

   int px, py, pd;
   scanf( "%d %d %d ", &px, &py, &pd);

```

```

25 |     for(int i = 0; i < n; i++)
|     {
|         for(int j = 0; j < n; j++)
|         {
|             if(manhattan(px, py, i, j) == pd)
30 |                 posicoes.push_back(pair<int, int(i, j));
|         }
|     }
|
35 |     for(int i = 0; i < k - 1; i++)
|     {
|         int x, y, d;
|         scanf("%d %d %d", &x, &y, &d);
|
40 |         for(int j = 0; j < posicoes.size(); j++)
|             if(manhattan(posicoes[j].first, posicoes[j].second, x, y) != d
|                 )
|             {
|                 posicoes.erase(posicoes.begin() + j);
|                 j--;
|             }
45 |
|         if(posicoes.size() == 1)
|             printf("%d %d\n", posicoes[0].first, posicoes[0].second);
|         else printf("-1 -1\n");
50 |
|     return 0;
}

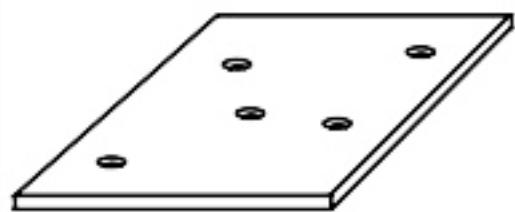
```

20.1.3 Furos

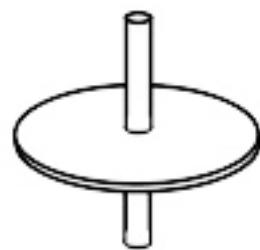
Especificação

Diâmetro = 5mm;

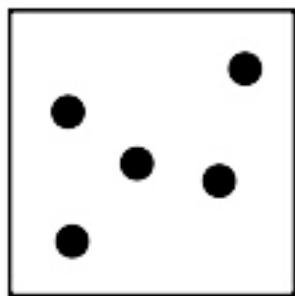
Você deve escrever um programa para determinar o diâmetro mínimo que a peça deve ter de tal forma que, com seu eixo encaixado em um dos furos da placa, a parte circular cubra completamente todos os outros furos da placa.



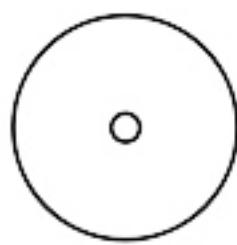
placa



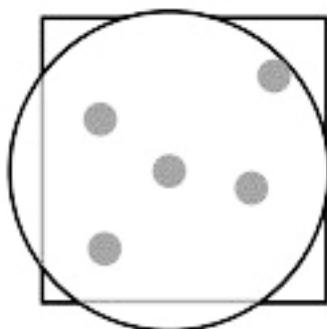
peça



vista frontal
da placa



vista frontal
da peça



vista frontal
da peça encaixada
na placa

Solução

```
#define MAXFUROS 1060

typedef struct
{
    int x, y;
    int dist[MAXFUROS];
} TFuro;

TFuro furos[MAXFUROS];

double calculaDitancia(int x1, int y1, int x2, int y2)
{
    return sqrt((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2));
}

int main()
{
    int n, min, max, nF, i, j, cont = 1;

    while (scanf("%d", &n), n)
    {
        nF = 0;

        for (i = 0; i < n; i++)
        {
            scanf("%d%d", &(furos[nF].x), &(furos[nF].y));
            nF++;
        }

        for (i = 0; i < n; i++)
            furos[i].dist[i] = 0;
        for (i = 0; i < n - 1; i++)
        {
            for (j = i + 1; j < n; j++)
            {
                furos[i].dist[j] = round(ceil(2 * calculaDitancia(furos[i].x,
                    furos[i].y, furos[j].x, furos[j].y)));
                furos[j].dist[i] = furos[i].dist[j];
            }
        }
    }

    min = INFINITO;
    for (i = 0; i < n; i++)
    {
        max = -1;

        for (j = 0; j < n; j++)
        {
            if (furos[i].dist[j] > max)
                max = furos[i].dist[j];
        }

        if (max < min)
            min = max;
    }
}
```

```

55     printf( "Teste %d | n%d | n| n", cont++, min+5);
}
56
57     return 0;
60 }

```

20.1.4 Interseção de Linhas

```

/* Intersecao de linhas */

5 #include <algorithm>
# include <cstdio>
# include <cmath>
# include <vector>
# include <iostream>
# include <iomanip>

10 using namespace std;

# define INF 1e9
# define EPS 1e-9
# define PI acos(-1.0)

15 struct point
{
    double x, y;
    point(double _x, double _y)
20    {
        x = _x, y = _y;
    }
    bool operator < (point other)
25    {
        if (fabs(x - other.x) > EPS)
            return x < other.x;
        return y < other.y;
    }
30};

35 struct line
{
    double a, b, c;
};

void pointsToLine(point p1, point p2, line *l)
{
    if (p1.x == p2.x)
40    {
        l->a = 1.0;
        l->b = 0.0;
        l->c = -p1.x;
    }
    else
45    {
        l->a = -(double)(p1.y - p2.y) / (p1.x - p2.x);
        l->b = 1.0;
        l->c = -(double)(l->a * p1.x) - (l->b * p1.y);
    }
}

```

```

        }

50    }

55    bool areParallel(line l1, line l2)
{
    return (fabs(l1.a-l2.a) < EPS) && (fabs(l1.b-l2.b) < EPS);
}

60    bool areSame(line l1, line l2)
{
    return areParallel(l1 ,l2) && (fabs(l1.c - l2.c) < EPS);
}

65    bool areIntersect(line l1, line l2, point *p)
{
    if (areSame(l1, l2)) return false;
    if (areParallel(l1, l2)) return false;
    p->x = (double)(l2.b * l1.c - l1.b * l2.c) /
        (l2.a * l1.b - l1.a * l2.b);
    if (fabs(l1.b) > EPS)
        p->y = - (l1.a * p->x + l1.c) / l1.b;
    else
        p->y = - (l2.a * p->x + l2.c) / l2.b;
    return true;
}

75    int main()
{
    int n;

80    cin>>n;
    scanf( "%d", &n);
    printf( "INTERSECTING LINES OUTPUT|n");

85    for(int i = 0; i < n; i++)
    {
        double x1,y1,x2,y2,x3,y3,x4,y4;

90        scanf( "%lf %lf %lf %lf", &x1, &y1, &x2, &y2);
        scanf( "%lf %lf %lf %lf", &x3, &y3, &x4, &y4);

95        point p1(x1, y1), p2(x2, y2), p3(x3, y3), p4(x4, y4);

        line l1, l2;

100       pointsToLine(p1, p2, &l1);
        pointsToLine(p3, p4, &l2);

        if(areSame(l1, l2))
            printf( "LINE");
        else if(areParallel(l1, l2))
            printf( "NONE");
        else
        {
            point p(0, 0);

105       if(areIntersect(l1, l2, &p))

```

```

    {
        if(p.x == 0 && p.y == 0)
            printf("POINT 0.00 0.00\n");
        else
            printf("POINT %.2lf %.2lf\n", p.x, p.y);
    }
}

printf("END OF OUTPUT\n");

return 0;
}

```

20.1.5 Interseção total de quadrados

```

int main()
{
    int n;
    int INTERINFX, INTERINFY, INTERSUPX, INTERSUPY;
    int xInf, yInf, xSup, ySup;

    int i, test = 0;

    while(1)
    {
        scanf("%d", &n);

        if(!n)
            break;

        scanf("%d%d%d%d", &INTERINFX, &INTERSUPY, &INTERSUPX, &
              INTERINFY);

        for(i = 1; i<n; i++)
        {
            scanf("%d%d%d%d", &xInf, &ySup, &xSup, &yInf);

            INTERINFX = xInf > INTERINFX?xInf:INTERINFX;
            INTERINFY = yInf > INTERINFY?yInf:INTERINFY;
            INTERSUPX = xSup < INTERSUPX?xSup:INTERSUPX;
            INTERSUPY = ySup < INTERSUPY?ySup:INTERSUPY;
        }

        if(INTERSUPX >= INTERINFX && INTERSUPY >= INTERINFY)
            printf("Teste %d\n%d %d %d %d\n", ++test, INTERINFX,
                  INTERSUPY, INTERSUPX, INTERINFY);
        else
            printf("Teste %d\nnenhum\n", ++test);
    }

    return 0;
}

```

20.1.6 Ponto → Círculo → Ponto

Especificação

Problem M

Rope Crisis in Ropeland !

Input: standard input
Output: standard output
Time Limit: 2 seconds

This is a story of Ropeland where rope pulling is a very popular game (like cricket in Bangladesh). Perhaps you know the game rope pulling: two groups of players hold two ends of a rope. When a certain signal is given they start pulling ropes. The group that can snatch the rope from the other group is declared winner. Today is a very happy day in Ropeland as they have got rope status (something like Bangladesh's test status). So the people of Ropeland are on the street and they are willing to be engaged in rope pulling. But the shops in the city fail to supply enough rope and so now a rope crisis has begun. The King of the country declares a new rule that two groups will not be allowed to buy more ropes than what they require.

The problem is that rope-pulling takes place in a large hall room that has a large round pillar in the middle with certain radius. So if two groups are on the opposite side of the pillar their pulled rope is never in a straight line. Given the position of the two groups you are to find out the minimum length of rope required by them to start rope-pulling. You can assume that a point represents the position of each group.

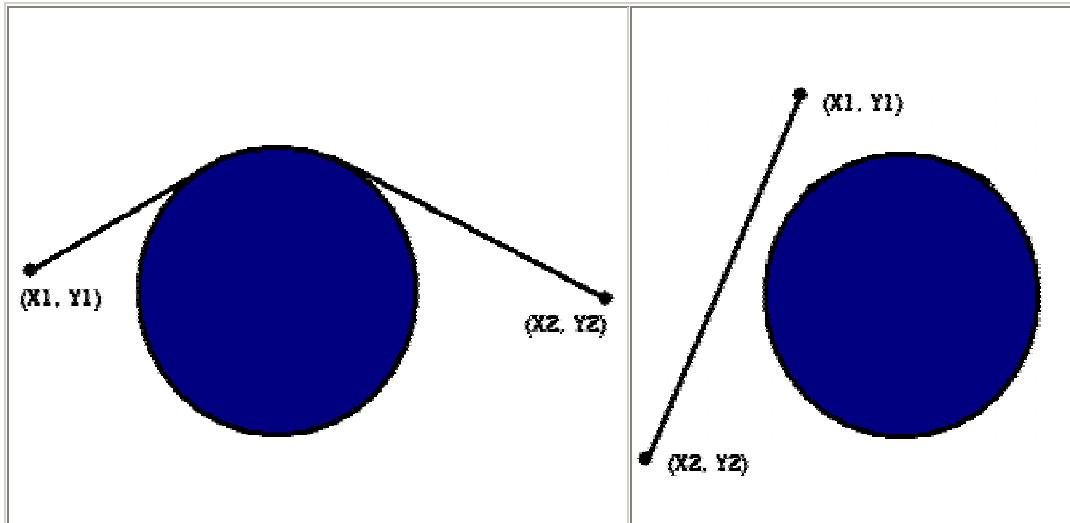


Fig 1: Situation when two groups have the round pillar between them. The pulled rope is never a straight line

Fig 2: Situation when the two groups don't have the round pillar between them

Input

The first line of the input file contains an integer **N**, which tells how many sets of input are there. Next there are **N** lines of input.

Each line contains five numbers **X1**, **Y1**, **X2**, **Y2** and **R** (> 0) where **(X1, Y1)** and **(X2, Y2)** are the coordinates of the two groups and **R** is the radius of the pillar. The coordinate of the center of the pillar is always the origin. You can also assume that none of the coordinates will be inside the circle. All input numbers except **N** are floating point numbers and none of their absolute value is greater than **10000**.

Output

For each set of input output a floating-point number in a new line rounded to the third digit after the decimal point and this number denotes the minimum length of the rope required.

Sample Input

```
2
1 1 -1 -1 1
1 1 -1 1 1
```

Sample Output

```
3.571
2.000
```

Rezaul Alam Chowdhury (Idea generator and judge solution writer) & **Shahriar Manzoor** (Problem statement writer)

“It may seem weird to many, but there are more triangles than there are points in this universe.”

Solução

```
#define EPS 1e-9
#define DBGR
#define PI 3.141592653589793115997963468544

5  typedef double pdata;

struct p2d
{
    pdata x, y;
10
    p2d operator+(p2d p)
    {
        return p2d(x + p.x, y + p.y);
    }
15
    p2d operator-(p2d p)
    {
        return p2d(x - p.x, y - p.y);
    }
20
    p2d operator*(pdata k)
    {
        return p2d(k * x, k * y);
    }
25
    p2d operator/(pdata k)
    {
        return p2d(x / k, y / k);
    }
30
    pdata operator*(p2d p)
    {
        return (x * p.y - y * p.x);
    }
35
    pdata operator^(p2d p)
    {
        return (x * p.x + y * p.y);
    }
40
    bool operator==(p2d p)
    {
        return (x == p.x && y == p.y);
    }
45
    bool operator<(p2d p) const
    {
        return (x < p.x || (x == p.x && y < p.y));
    }
50
    bool operator>(p2d p) const
    {
        return (x > p.x || (x == p.x && y > p.y));
    }
55
```

```

  bool point_on_line(p2d p0, p2d p1)
  {
    return ((p1 - p0) * (*this - p0)) == 0;
  }

60  pdata sqr(pdata x)
{
  return (x * x);
}

65  double dist(p2d p)
{
  return hypot(x - p.x, y - p.y);
}

70  double dist2(p2d p)
{
  return sqr(x - p.x) + sqr(y - p.y);
}

75  double mod()
{
  return sqrt(x * x + y * y);
}

80  double mod2()
{
  return (x * x + y * y);
}

85  double ang(p2d p)
{
  return acos((*this ^ p)/((*this).mod() * p.mod()));
}

90  double point_line_distance(p2d p0, p2d p1)
{
  p2d v1 = *this - p0, v2 = p1 - p0;
  double u = (v1 ^ v2)/v2.mod2();
  p2d p = p0 + v2 * u;

  return (*this).dist(p);
}

100 double point_line_segment_distance(p2d p0, p2d p1)
{
  p2d v1 = *this - p0, v2 = p1 - p0;
  double u = (v1 ^ v2)/v2.mod2();

  if(u < 0)
    return (*this).dist(p0);

  if(u > 1)
    return (*this).dist(p1);

110 return (*this).dist(p0 + v2 * u);
}

```

```

115     p2d(pdata _x = 0, pdata _y = 0): x(_x), y(_y) {};
};

120 void calc_tangent(double xp, double yp, double r, p2d *p1, p2d *p2)
{
    double a, b, c, r2, delta, xp2, yp2;
    double x1, x2, y1, y2;

    r2 = r * r;
    xp2 = xp * xp;
    yp2 = yp * yp;
125
    a = xp2 + yp2;

    if(fabs(xp) > EPS)
    {
        b = - 2 * r2 * yp;
        c = r2 *(r2 - xp2);
        delta = sqrt(b * b - 4 * a * c);
        y1 = (-b + delta)/(2 * a);
        x1 = (r2 - yp * y1)/xp;
135
        y2 = (-b - delta)/(2 * a);
        x2 = (r2 - yp * y2)/xp;
    }
    else
    {
        b = - 2 * r2 * xp;
        c = r2 *(r2 - yp2);
        delta = sqrt(b * b - 4 * a * c);
        x1 = (-b + delta)/(2 * a);
        y1 = (r2 - xp * x1)/yp;
145
        x2 = (-b - delta)/(2 * a);
        y2 = (r2 - xp * x2)/yp;
    }

    *p1 = p2d(x1, y1);
    *p2 = p2d(x2, y2);
}

double calc_arc(p2d p1, p2d p2)
{
    return p1.ang(p2);
}

int main()
{
    int i, nCases;
    double r, x0, x1, y0, y1, dist, arc, temp;
    p2d t1, t2, t3, t4;

    scanf("%d", &nCases);
165
    for(i = 0; i < nCases; i++)
    {
        scanf("%lf %lf %lf %lf %lf", &x0, &y0, &x1, &y1, &r);

        p2d orig = p2d(0, 0), p0 = p2d(x0, y0), p1 = p2d(x1, y1);

```

```

    dist = orig.point_line_segment_distance(p0, p1);

    if(dist < r)
175    {
        calc_tangent(x0, y0, r, &t1, &t2);
        calc_tangent(x1, y1, r, &t3, &t4);

        arc = 2e15;
180        arc = min(arc, calc_arc(t1, t3));
        arc = min(arc, calc_arc(t1, t4));
        arc = min(arc, calc_arc(t2, t3));
        arc = min(arc, calc_arc(t2, t4));

        printf("%.3lf\n", p0.dist(t1) + p1.dist(t3) + arc * r);
    }
    else
190    {
        printf("%.3lf\n", p0.dist(p1));
    }
}

return 0;
}

```

20.1.7 TV da Vovó

```

#define MAX 1006

int** matrix;

5 /*
 *
 */
int main()
{
10    int n, m, d1, d2, i, j, k;
    int* aux;

    matrix = (int**)malloc(MAX*sizeof(int *));
    for(i = 0; i<MAX; i++)
15    {
        matrix[i] = (int*)malloc(MAX*sizeof(int));
    }

    int test = 1;
20    while(1)
    {
        scanf("%d%d", &n, &m);
        if(!n && !m)
            break;
25    for(i = 0; i<n; i++)
    {
        for(j = 0; j<m; j++)
        {
30            scanf("%d", &(matrix[i][j]));
        }
    }
}

```

```

        }

35     int x = 0, y = 0;
      while(1)
    {
        scanf("%d%d", &d1, &d2);
        if(!d1 && !d2)
            break;
40
        x+=d1;
        y+=d2;
    }

45     printf("Teste %d\n", test++);
      for(i = 0; i<n; i++)
    {
        for(j = 0; j<m; j++)
        {
50         int iPr, jPr;

            if(y < 0)
            {
                iPr = i + y;
                while(iPr < 0)
                    iPr += n;
            }
            else
            {
60             iPr = (i + y)%n;
            }

            if(x < 0)
            {
                jPr = (j - x)%m;
            }
            else
            {
70             jPr = j - x;
                while(jPr < 0)
                    jPr += m;
            }

            if(j)
                printf(" %d", matrix[iPr][jPr]);
            else
                printf("%d", matrix[iPr][jPr]);
        }
        printf(" |n");
    }

80     printf(" |n");
}

85     return 0;
}

```

20.2 MDC (BigInteger)

```
import java.util.Scanner;
import java.math.BigInteger;

class Main /* UVa 10814 - Simplifying Fractions */
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        int N = sc.nextInt();
        while (N-- > 0)
        {
            BigInteger p = sc.nextBigInteger();
            String ch = sc.next();
            BigInteger q = sc.nextBigInteger();
            BigInteger gcd_pq = p.gcd(q);
            System.out.println(p.divide(gcd_pq) + " / " + q.divide(gcd_pq));
        }
    }
}
```

20.3 Modular Fibonacci

20.3.1 Especificação

Write a program which calculates $M_n = F_n \bmod 2^m$ for given pair of n and m . $0 \leq n \leq 2147483647$ and $0 \leq m < 20$. Note that $a \bmod b$ gives the remainder when a is divided by b .

Input and Output

Input consists of several lines specifying a pair of n and m . Output should be corresponding M_n , one per line.

Samples

```
11 7 → 89
11 6 → 25
```

20.3.2 Solução

```
typedef long long ll;
ll MOD;

#define MAX_N 2
5 struct Matrix { ll mat[MAX_N][MAX_N]; };

Matrix matMul(Matrix a, Matrix b)
{
    Matrix ans; int i, j, k;
    for (i = 0; i < MAX_N; i++)
10
```

```

    for (j = 0; j < MAX_N; j++)
        for (ans.mat[i][j] = k = 0; k < MAX_N; k++)
    {
        ans.mat[i][j] += (a.mat[i][k] % MOD) * (b.mat[k][j] % MOD);
        ans.mat[i][j] %= MOD;
    }
    return ans;
}

20 Matrix matPow(Matrix base, int p)
{
    Matrix ans; int i, j;
    for (i = 0; i < MAX_N; i++)
        for (j = 0; j < MAX_N; j++)
            ans.mat[i][j] = (i == j);
    while (p)
    {
        if (p & 1)
            ans = matMul(ans, base);
        base = matMul(base, base);
        p >>= 1;
    }
    return ans;
}

35 int normalExp(int base, int p)
{
    int ans = 1;
    for (int i = 0; i < p; i++) ans *= base;
    return ans;
}

40 int fastExp(int base, int p)
{
    if (p == 0) return 1;
    else if (p == 1) return base;
    else
    {
        int res = fastExp(base, p / 2);
        res *= res;
        if (p % 2 == 1) res *= base;
        return res;
    }
}

55 int main()
{
    int i, n, m;
    while (scanf("%d %d", &n, &m) == 2)
    {
        Matrix ans;
        ans.mat[0][0] = 1; ans.mat[0][1] = 1;
        ans.mat[1][0] = 1; ans.mat[1][1] = 0;
        for (MOD = 1, i = 0; i < m; i++)
            MOD *= 2;
        ans = matPow(ans, n);
    }
}

```

```

70     printf( "%lld\n", ans.mat[0][1]);
    }

    return 0;
}

```

21 Programação Dinâmica

21.1 Altitude

```

bool already[200];
int altitude[200];
int dp[200];

5 int calculaMax(int s, int n, map<int, vector<int>> &graph)
{
    if(!graph[s].size())
        return 0;

10   if(dp[s] != -1)
        return dp[s];

    int i, val;

15   int valMax = -1;
    for(i = 0; i < graph[s].size(); i++)
    {
        val = calculaMax(graph[s][i], n, graph);
        if(val > valMax)
            valMax = val;
    }

20   dp[s] = valMax + 1;

25   return dp[s];
}

int main()
{
30   int v, a, source, maxLevel, v1, v2;
    int i, j, test = 0;

    while(1)
    {
35     cin >> v >> a >> source;

        if(!v && !a && !source)
            break;

40     map< int, vector<int> > graph;

        vector<int> x;
        for(i = 1; i <= v; i++)
        {
45         graph[i] = x;
    }
}

```

```

        dp[i] = -1;

        cin >> altitude[i];
    }

50    for(i = 0; i<a; i++)
    {
        cin >> v1 >> v2;

55        if(altitude[v2] < altitude[v1])
            graph[v1].push_back(v2);
    }

        printf( "Teste %d | n%d | n| n", ++test, calculaMax(source, v, graph)
            );
60    }

        return 0;
}

```

21.2 Bottom-Up – Exemplo

```

/* Programação dinamica – Bottom Up */

#include <iostream>
#include <algorithm>
5 #include <cmath>
#include <vector>

using namespace std;

10 int main(){
    int n;
    vector<unsigned long long> formas(10001, 0);
    vector<int> coins;

15    for(int i=1; i<=21; i++)
        coins.push_back(pow(i, 3));

    formas[0]=1;

20    for(int i=0; i<coins.size(); i++)
        for(int j = coins[i]; j<10001; j++)
            formas[j] += formas[j-coins[i]];

    while(cin >> n) cout << formas[n] << endl;
25    return 0;
}

```

21.3 Coin Change

```

/* Coin Change */

```

```

100   int N = 5, V, coinValue[5] = {1, 5, 10, 25, 50}, memo[6][7500];
105
110   int ways(int type, int value)
115   {
120     if (value == 0)           return 1;
125     if (value < 0 || type == N) return 0;
130     if (memo[type][value] != -1) return memo[type][value];
135     return memo[type][value] = ways(type + 1, value) + ways(type,
140       value - coinValue[type]);
145
150   int main()
155   {
160     memset(memo, -1, sizeof memo);
165     while (scanf("%d", &V) != EOF)
170       printf("%d | %d\n", V, ways(0, V));
175
180   return 0;
185 }
```

21.4 Flight Simulator

21.4.1 Especificação

Problem B

Flight Planner

Input: standard input

Output: standard output

Time Limit: 1 second

Memory Limit: 32 MB

Calculating the minimal cost for a flight involves calculating an optimal flight-altitude depending on wind-strengths changing with different altitudes. It's not enough just to ask for the route with optimal wind-strength, because due to the mass of a plane you need a certain amount of fuel to rise. Moreover due to safety regulations it's forbidden to fly above a certain altitude and you can't fly under zero-level.

In order to simplify the problem for now, we assume that for each 100 miles of flight you have only three possibilities: to climb one mile, to hold your altitude or to sink one mile.

Climb flight requires 60 units of fuel, holding your altitude requires 30 units and sinking requires 20 units.

In the case of headwind you need more fuel while you can save fuel flying with tailwind. Windstrength w will satisfy the condition $-10 \leq w \leq 10$, where negative windstrength is meant to be headwind and positive windstrength is tailwind.

For one unit of tailwind you can save one unit of fuel each 100 miles; each unit of headwind will cost an extra unit of fuel.

For example to climb under conditions of windstrength $w = -5$, you need 65 units of fuel for this 100 miles.

Given the windstrengths on different altitudes for a way from here to X, calculate the minimal amount of fuel you need to fly to X.

Input

The first line of the input file contains the number N of test cases in the file. The first line of each test case contains a single integer X, the distance to fly, with $1 \leq X \leq 100000$ miles and X is a multiple of 100. Notice that it's not allowed to fly higher than 9 miles over zero and that you have to decide whether to climb, hold your altitude or to sink only for every 100 miles. For every mile of allowed altitude (starting at altitude 9 down to altitude 0) there follow $X/100$ windstrengths, starting with the windstrength at your current position up to the windstrength at position $X-100$ in steps of 100 miles. Test cases are separated by one or more blank lines.

Output

For each test case output the minimal amount of fuel used flying from your current position (at altitude 0) to X (also at altitude 0), followed by a blank line.

Sample Input

3

400
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 9 9 1
1 -9 -9 1

1000
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 9
7 7 7 7 7 7 7 7 7 7
-5 -5 -5 -5 -5 -5 -5 -5 -5
-7 -3 -7 -7 -7 -7 -7 -7 -7
-9 -9 -9 -9 -9 -9 -9 -9 -9

Sample Output

120

354

Frank Hutter

21.4.2 Solução

```
int matrix[12][100010];

int min(int a, int b)
{
    if(a < b)
        return a;
    return b;
}

int dynamicProgramming(int i, int j, int l, int c, map< pair<int,
    int>, int> &memoize)
{
    if(j == 0)
    {
        if(i == l-1)
            return 0;
        return 10000000;
    }

    if(i == l-1 && j < c)
    {
        return 10000000;
    }

    map< pair<int,int>, int>::iterator it;
    it = memoize.find(make_pair(i,j));

    if(it == memoize.end())
    {
        int res1, res2, res3;

        if(i-1 < 0 || j-1 < 0)
            res1 = 10000000;
        else
            res1 = dynamicProgramming(i-1,j-1,l,c, memoize) + 20 - matrix
                [i-1][j-1];

        if(j-1 < 0)
            res2 = 10000000;
        else
            res2 = dynamicProgramming(i,j-1,l,c, memoize) + 30 - matrix[i
                ][j-1];

        if(j-1 < 0 || i+1 == l)
            res3 = 10000000;
        else
            res3 = dynamicProgramming(i+1,j-1,l,c, memoize) + 60 - matrix
                [i+1][j-1];

        memoize[make_pair(i,j)] = min(min(res1,res2),res3);
    }

    return memoize[make_pair(i,j)];
}

int main(int argc, char** argv)
```

```

{
    long long i, j, k;
    char* read = (char*)malloc((MAXSTRSZ+1)*sizeof(char));

    int nCases;
    cin >> nCases;

60   for(i = 0; i<nCases; i++)
    {
        int lines, col, aux, n;
        map< pair<int,int> , int > memoize;

65       cin >> aux;
        lines = 10;
        col = aux/100;

        for(j = 0; j<10; j++)
        {
            for(k = 0; k<col; k++)
            {
                cin >> matrix[j][k];
            }
        }

        printf("%d\n\n", dinamicProgramming(lines-1, col, lines, col,
                                           memoize));
    }

80   return 0;
}

```

21.5 Geração de Valores com Recursos Limitados

21.5.1 Especificação

166 Making Change

Given an amount of money and unlimited (almost) numbers of coins we know that an amount of money may be made up in a variety of ways. A more interesting problem arises when goods are bought and need to be paid for, with the possibility that change may need to be given. Given the finite resources of most wallets nowadays, we are constrained in the number of ways in which we can make up an amount to pay for our purchases—assuming that we can make up the amount in the first place, but that is another story.

The problem we will be concerned with will be to minimise the number of coins that change hands at such a transaction, given that the shopkeeper has an adequate supply of all coins. (The set of New Zealand coins comprises 5c, 10c, 20c, 50c, \$1 and \$2.) Thus if we need to pay 55c, and we do not hold a 50c coin, we could pay this as $2*20c + 10c + 5c$ to make a total of 4 coins. If we tender \$1 we will receive 45c in change which also involves 4 coins, but if we tender \$1.05 (\$1 + 5c), we get 50c change and the total number of coins that changes hands is only 3.

Write a program that will read in the resources available to you and the amount of the purchase and will determine the minimum number of coins that change hands.

Input

Input will consist of a series of lines, each line defining a different situation. Each line will consist of 6 integers representing the numbers of coins available to you in the order given above, followed by a real number representing the value of the transaction, which will always be less than \$5.00. The file will be terminated by six zeroes (0 0 0 0 0 0). The total value of the coins will always be sufficient to make up the amount and the amount will always be achievable, that is it will always be a multiple of 5c.

Output

Output will consist of a series of lines, one for each situation defined in the input. Each line will consist of the minimum number of coins that change hands right justified in a field 3 characters wide.

Sample input

```
2 4 2 2 1 0  0.95
2 4 2 0 1 0  0.55
0 0 0 0 0 0
```

Sample output

```
2
3
```

21.5.2 Solução

```
vector<int> valores;
char help[100];

int trunque(double x)
{
    sprintf(help, "%.0lf", x*100);
    return atoi(help);
}

void generateAll(vector<int> &resources, vector<pair< int , vector<
    int > > &vec)
{
    long long i, j, k;

    vector<int> x;

    pair< int , vector<int> > add;
    add.first = 0;
    add.second = x;

    vec.push_back(add);

    add.first = INF;

    for(i = 1; i<=500; i++)
    {
        vec.push_back(add);
    }

    for(i = 0; i<6; i++)
        for(j = 0; j<resources[i]; j++)
            for(k = 500; k >= 0; k-=5)
                if(vec[k].first < INF)
                    if(k + valores[i] <= 500)
                        if(vec[k].first + 1 < vec[k + valores[i]].first)
                            {
                                vec[k + valores[i]].first = vec[k].first + 1;
                                vec[k + valores[i]].second = vec[k].second;
                                vec[k + valores[i]].second.push_back(valores[i]);
                            }
    }

    int generateOne(int val)
    {
        int i;
        vector<int> uses (6,0);

        int v = val;
        for(i = 5; i>=0; i--)
        {
            uses[i] = v/valores[i];
            v = v%valores[i];
        }

        int tot = 0;
        for(i = 0; i<6; i++)
    }
```

```

        tot+=uses[i];

    return tot;
}

60 int main(int argc, char** argv)
{
    long long i, j, k, nCases;
    char* read = (char*)malloc((MAXSTRSZ+1)*sizeof(char));
    char c;

    valores.push_back(5);
    valores.push_back(10);
    valores.push_back(20);
70    valores.push_back(50);
    valores.push_back(100);
    valores.push_back(200);
    vector<int> resources (6, 0);

    map<int, int> memoize;

    for(long long cases = 0; 1/* cases < nCases */; cases++)
    {
        cin.getline(read, MAXSTRSZ);

80        int r;
        sscanf(read, "%d%d%d%d%d%d", &(resources[0]), &(resources[1])
            , &(resources[2]), &(resources[3]), &(resources[4]), &(
            resources[5]), &r);

        if(!resources[0] && !resources[1] && !resources[2] && !
            resources[3] && !resources[4] && !resources[5])
            break;

        read+=r;
        double v;
        sscanf(read, "%lf", &v);
        int valIni = trunque(v);
        read-=r;

        vector<pair< int, vector<int> > > comprador;
        generateAll(resources, comprador);

95        long long min = 1000000000;

        for(i = valIni; i<=500; i++)
        {
            if(comprador[i].first == INF)
                continue;

            int quant = generateOne(i - valIni);

            if(comprador[i].first + quant < min)
            {
                min = comprador[i].first + quant;
            }
        }
    }

110
}

```

```

    printf( "%3lld\n", min);
}

115 return 0;
}

```

21.6 Geração de Valores com Recursos Ilimitados

21.6.1 Especificação

Escrever um programa que, dados o preço de uma mercadoria e os valores das moedas disponíveis, calcule o menor número possível de moedas necessário para comprar o produto sem voltar troco, ou seja, o menor número de moedas tal que o total seja exatamente o preço da mercadoria.

21.6.2 Solução

```

int main()
{
    int i, j, k;
    while(1)
    {
        int val, n;

        cin >> val;

        10 if(!val)
            break;

        vector<int> moedas;
        vector<int> dp (val + 101, INFINITO);
        15 int lim = val + 100;
        dp[0] = 0;

        cin >> n;

        20 for(i = 0; i<n; i++)
        {
            cin >> j;
            moedas.push_back(j);
        }

        25 for(j = 0; j<n; j++)
            for(i = 0; i<=val; i++)
                if(dp[i] < INFINITO)
                    if(i+moedas[j] <= lim)
                        dp[i+moedas[j]] = min(dp[i] + 1, dp[i+moedas[j]]);

                if(dp[val]!=INFINITO)
                    cout << dp[val] << endl;
                else
                    cout << "Impossivel" << endl;
            }

        35 return 0;
}

```

```
}
```

21.7 Maximum Sum 2D (Matrix)

```
/* Maior soma em uma matriz*/  
  
#include <iostream>  
#include <string.h>  
#include <stdio.h>  
  
5    int main()  
{  
    int n;  
10   int matriz[105][105];  
    int tabela[105][105];  
  
    while (scanf("%d", &n) != EOF)  
    {  
15     int vMax = -99999;  
        int somaParcial, somaTotal;  
  
        for (int i = 0; i < n; i++)  
            for (int j = 0; j < n; j++)  
                scanf("%d", &matriz[i][j]);  
  
20       memset(tabela, 0, sizeof(tabela));  
  
        for (int i = 0; i < n; i++)  
25        {  
            for (int j = 0; j < n; j++)  
            {  
                somaParcial = 0;  
  
30                for (int k = j; k < n; k++)  
                {  
                    somaParcial += matriz[i][k];  
                    somaTotal = somaParcial + tabela[j][k];  
  
35                    if (somaTotal > vMax)  
                        vMax = somaTotal;  
  
                    if (somaTotal < 0)  
                        somaTotal = 0;  
  
40                    tabela[j][k] = somaTotal;  
                }  
            }  
        }  
45        printf("%d\n", vMax);  
    }  
  
    return 0;  
}
```

21.8 Maximum Sum with Range

```
int saldos[10006];
int dp1[10006];
int dp2[10006];

5 int main()
{
    int i, j, k, n, v1, v2, cont, max, iMax, res, diffSol, iSol, fSol
        ;

    cont = 1;
10
    while(1)
    {
        scanf("%d", &n);

        if(!n)
            break;

        for(i = 0; i<n; i++)
        {
            scanf("%d%d", &v1, &v2);

            saldos[i] = v1-v2;
        }

25
        dp1[0] = 0;
        dp2[0] = saldos[0];

        diffSol = 0, fSol = dp2[0], iSol = 0;

30
        for(i = 1; i<n; i++)
        {

            if(dp2[i-1] >= 0)
            {
                dp1[i] = dp1[i-1];
                dp2[i] = dp2[i-1] + saldos[i];
            }
            else
            {
40
                if(saldos[i] > dp2[i-1])
                {
                    dp1[i] = i;
                    dp2[i] = saldos[i];
                }
                else
                {
45
                    dp1[i] = dp1[i-1];
                    dp2[i] = dp2[i-1] + saldos[i];
                }
            }
        }

50
            if(dp2[i] > fSol)
            {
                fSol = dp2[i];
                diffSol = i-dp1[i];
            }
        }
    }
}
```

```

        iSol = i;
    }
    else if(dp2[i] == fSol)
    {
        if(i - dp1[i] > diffSol)
        {
            diffSol = i-dp1[i];
            iSol = i;
        }
    }
}

if(dp2[iSol] > 0)
    printf("Teste %d | n%d %d | n| n", cont++, dp1[iSol] + 1, iSol +
1);
else
    printf("Teste %d | nnenhum | n| n", cont++);
}

return 0;
}

```

21.9 *nWays* - Ilha do Cubo

```

vector<long long> valores;

int main(int argc, char** argv)
{
    long long i, j, k, nCases;
    char* read = (char*)malloc((MAXSTRSZ+1)*sizeof(char));
    char c;

    valores.push_back(1);
    valores.push_back(8);
    valores.push_back(27);
    valores.push_back(64);
    valores.push_back(125);
    valores.push_back(216);
    valores.push_back(343);
    valores.push_back(512);
    valores.push_back(729);
    valores.push_back(1000);
    valores.push_back(1331);
    valores.push_back(1728);
    valores.push_back(2197);
    valores.push_back(2744);
    valores.push_back(3375);
    valores.push_back(4096);
    valores.push_back(4913);
    valores.push_back(5832);
    valores.push_back(6859);
    valores.push_back(8000);
    valores.push_back(9261);

    vector<long long> DP (10666, 0);
    DP[0] = 1;
}

```

```

35   for(i = 0; i<valores.size(); i++)
{
    for(j = 1; j<=10600; j++)
    {
        if(j - valores[i] < 0)
            continue;
        DP[j] += DP[j - valores[i]];
    }
}

long long search;
while((cin >> search) != NULL)
{
    cout << DP[search] << endl;
}

return 0;
}

```

21.10 Série de Cantor / Count or Cantor

21.10.1 Especificação

264 Count on Cantor

One of the famous proofs of modern mathematics is Georg Cantor's demonstration that the set of rational numbers is enumerable. The proof works by using an explicit enumeration of rational numbers as shown in the diagram below.

1/1	1/2	1/3	1/4	1/5	...
2/1	2/2	2/3	2/4		
3/1	3/2	3/3			
4/1	4/2				
5/1					

In the above diagram, the first term is 1/1, the second term is 1/2, the third term is 2/1, the fourth term is 3/1, the fifth term is 2/2, and so on.

Input and Output

You are to write a program that will read a list of numbers in the range from 1 to 10^7 and will print for each number the corresponding term in Cantor's enumeration as given below. No blank line should appear after the last number.

The input list contains a single number per line and will be terminated by end-of-file. No more than 30 numbers will appear in the input file.

Sample input

```
3
14
7
```

Sample output

```
TERM 3 IS 2/1
TERM 14 IS 2/4
TERM 7 IS 1/4
```

21.10.2 Solução

```
void geraLinha(int i, vector< pair<int,int> > &elements)
{
    int j;

    if(i%2 == 0)
    {
        for(j = 0; j<i; j++)
        {
            elements.push_back(make_pair(1+j, i-j));
        }
    }
    else
    {
        vector< pair<int,int> > aux;
        for(j = 0; j<i; j++)
        {
            aux.push_back(make_pair(1+j, i-j));
        }

        for(j = aux.size()-1; j>=0; j--)
            elements.push_back(aux[j]);
    }
}

int main(int argc, char** argv)
{
    long long i, j, k;
    char* read = (char*)malloc((MAXSTRSZ+1)*sizeof(char));

    vector< pair<int,int> > elements;
    elements.push_back(make_pair(0,0));

    int key;
    int last = 0;

    while(scanf("%d", &key) == 1)
    {
        while(key >= elements.size())
        {
            last++;
            geraLinha(last, elements);
        }

        printf("TERM %d IS %d/%d\n", key, elements[key].first, elements[key].second);
    }

    return 0;
}
```

21.11 Soma de dois == k

```

/* How do you add? */

5   /*
int N, K, memo[110][110];

10  int ways(int N, int K)
11  {
12      if (K == 1)
13          return 1;
14      else if (memo[N][K] != -1)
15          return memo[N][K];
16
17      int total_ways = 0;
18      for (int split = 0; split <= N; split++)
19          total_ways = (total_ways + ways(N - split, K - 1)) % 1000000;
20      return memo[N][K] = total_ways;
21  }

22  int main()
23  {
24      memset(memo, -1, sizeof memo);
25      while (scanf("%d %d", &N, &K), (N || K))
26          printf("%d\n", ways(N, K));
27      return 0;
28  }
29 */

30 int main()
31 {
32     int i, j, split, dp[110][110], N, K;
33
34     memset(dp, 0, sizeof dp);
35
36     for (i = 0; i <= 100; i++)
37         dp[i][1] = 1;
38
39     for (j = 1; j < 100; j++)
40         for (i = 0; i <= 100; i++)
41             for (split = 0; split <= 100 - i; split++)
42             {
43                 dp[i + split][j + 1] += dp[i][j];
44                 dp[i + split][j + 1] %= 1000000;
45             }
46
47     while (scanf("%d %d", &N, &K), (N || K))
48         printf("%d\n", dp[N][K]);
49
50     return 0;
51 }
```

21.12 Stacking Boxes

21.12.1 Especificação

103 Stacking Boxes

Background

Some concepts in Mathematics and Computer Science are simple in one or two dimensions but become more complex when extended to arbitrary dimensions. Consider solving differential equations in several dimensions and analyzing the topology of an n -dimensional hypercube. The former is much more complicated than its one dimensional relative while the latter bears a remarkable resemblance to its “lower-class” cousin.

The Problem

Consider an n -dimensional “box” given by its dimensions. In two dimensions the box (2,3) might represent a box with length 2 units and width 3 units. In three dimensions the box (4,8,9) can represent a box $4 \times 8 \times 9$ (length, width, and height). In 6 dimensions it is, perhaps, unclear what the box (4,5,6,7,8,9) represents; but we can analyze properties of the box such as the sum of its dimensions.

In this problem you will analyze a property of a group of n -dimensional boxes. You are to determine the longest *nesting string* of boxes, that is a sequence of boxes b_1, b_2, \dots, b_k such that each box b_i nests in box b_{i+1} ($1 \leq i < k$).

A box $D = (d_1, d_2, \dots, d_n)$ nests in a box $E = (e_1, e_2, \dots, e_n)$ if there is some rearrangement of the d_i such that when rearranged each dimension is less than the corresponding dimension in box E. This loosely corresponds to turning box D to see if it will fit in box E. However, since any rearrangement suffices, box D can be contorted, not just turned (see examples below).

For example, the box $D = (2,6)$ nests in the box $E = (7,3)$ since D can be rearranged as (6,2) so that each dimension is less than the corresponding dimension in E. The box $D = (9,5,7,3)$ does NOT nest in the box $E = (2,10,6,8)$ since no rearrangement of D results in a box that satisfies the nesting property, but $F = (9,5,7,1)$ does nest in box E since F can be rearranged as (1,9,5,7) which nests in E.

Formally, we define nesting as follows: box $D = (d_1, d_2, \dots, d_n)$ nests in box $E = (e_1, e_2, \dots, e_n)$ if there is a permutation π of $1 \dots n$ such that $(d_{\pi(1)}, d_{\pi(2)}, \dots, d_{\pi(n)})$ “fits” in (e_1, e_2, \dots, e_n) i.e., if $d_{\pi(i)} < e_i$ for all $1 \leq i \leq n$.

The Input

The input consists of a series of box sequences. Each box sequence begins with a line consisting of the number of boxes k in the sequence followed by the dimensionality of the boxes, n (on the same line.)

This line is followed by k lines, one line per box with the n measurements of each box on one line separated by one or more spaces. The i^{th} line in the sequence ($1 \leq i \leq k$) gives the measurements for the i^{th} box.

There may be several box sequences in the input file. Your program should process all of them and determine, for each sequence, which of the k boxes determine the longest nesting string and the length of that nesting string (the number of boxes in the string).

In this problem the maximum dimensionality is 10 and the minimum dimensionality is 1. The maximum number of boxes in a sequence is 30.

The Output

For each box sequence in the input file, output the length of the longest nesting string on one line followed on the next line by a list of the boxes that comprise this string in order. The “smallest” or “innermost” box of the nesting string should be listed first, the next box (if there is one) should be listed second, etc.

The boxes should be numbered according to the order in which they appeared in the input file (first box is box 1, etc.).

If there is more than one longest nesting string then any one of them can be output.

Sample Input

```
5 2
3 7
8 10
5 2
9 11
21 18
8 6
5 2 20 1 30 10
23 15 7 9 11 3
40 50 34 24 14 4
9 10 11 12 13 14
31 4 18 8 27 17
44 32 13 19 41 19
1 2 3 4 5 6
80 37 47 18 21 9
```

Sample Output

```
5
3 1 2 4 5
4
7 2 5 6
```

21.12.2 Solução

```
bool satisfaz(vector<int> &a, vector<int> &b)
{
    int i;

    for(i = 0; i<a.size(); i++)
    {
        if(b[i] <= a[i])
            return false;
    }
    return true;
}

int main(int argc, char** argv)
{
    int i, j, k;

    while(1)
    {
        int nBoxes, dimension, val;

        if(scanf("%d%d", &nBoxes, &dimension) != 2)
            break;

        vector<vector<int>> valores;
        map<vector<int>, int> indice;

        vector<int> z;
        for(i = 0; i<nBoxes; i++)
        {
            valores.push_back(z);
        }

        for(i = 0; i<nBoxes; i++)
        {
            for(j = 0; j<dimension; j++)
            {
                scanf("%d", &val);
                valores[i].push_back(val);
            }
        }

        for(i = 0; i<nBoxes; i++)
        {
            sort(valores[i].begin(), valores[i].end());
            indice[valores[i]] = i;
        }

        sort(valores.begin(), valores.end());

        vector<int> tam (valores.size(), 0);
        vector<int> keep (valores.size(), -1);

        int largest = 0, indLarg = -1;
        for(i = valores.size()-1; i>=0; i--)
    }
```

```

{
    int max = 0, indMax = -1;

60    for(j = i+1; j<valores.size(); j++)
    {
        if(satisfaz(valores[i], valores[j]))
        {
            if(tam[j] >= max)
            {
                max = tam[j];
                indMax = j;
            }
        }
    }

    tam[i] = max + 1;

    if(indMax != -1)
    {
        keep[i] = indMax;
    }

    if(tam[i] >= largest)
    {
        largest = tam[i];
        indLarg = i;
    }
}

85 printf("%d\n", largest);

int next = indLarg;
printf("%d", indice[valores[next]] + 1);
next = keep[next];
while(next != -1)
{
    printf(" %d", indice[valores[next]] + 1);
    next = keep[next];
}
printf("\n");
}

return 0;
}

```

21.13 Supermarket

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

5 typedef struct a{
    int id;
    double preco;
}TPair;

```

```

10 #define INF 99999999
#define MIN(a, b) (a<b?a:b)

15 TPair produtos[100010];
int lista[101];
int n, m;

20 double memoize[101];
double PD(){
    int i, j;
    double backup, backup2;

25    for(i=0; i<m; i++) memoize[i] = INF;

    for(i=0; i<n; i++)
        for(backup = 0, j=0; j<m; j++){
            backup2 = memoize[j];
            if(lista[j] == produtos[i].id)
                memoize[j] = MIN(memoize[j], backup + produtos[i].preco);
            backup = backup2;
        }

30    printf("%.2lf\n", memoize[m-1]);
}

35 int main(){
    int i;
    double resposta;
    int ehImp;

40    while(scanf("%d %d", &m, &n) && (n+m)){
        ehImp = 0;
        for(i=0; i<m; i++) scanf("%d", &lista[i]);
        for(i=0; i<n; i++){
            scanf("%d %lf", &produtos[i].id, &produtos[i].preco);
            if(lista[ehImp] == produtos[i].id) ehImp++;
        }

45        if(ehImp == m) PD();
        else printf("Impossible\n");
    }

50    return 0;
}

```

21.14 Top-Down – Exemplos

Exemplo #1:

```

/* Maior subsequencia estritamente crescente */

#include <iostream>
#include <vector>
5
using namespace std;

typedef struct teste{

```

```

10     int tam;
11     int no;
12 }TCelula;

13 int main(){
14     int in;
15     TCelula aux;
16     vector<int> seq;
17     vector<int> maior;

18     aux.tam = 1;
19     aux.no = -1;

20     while(cin >> in) seq.push_back(in);

21     maior.push_back(seq[0]);

22     vector<TCelula> sequencia(seq.size());
23     fill(sequencia.begin(), sequencia.end(), aux);

24     for(int i = seq.size()-1; i >= 0; i--)
25         for(int j = i+1; j < seq.size(); j++)
26             if(seq[j] > seq[i]){
27                 sequencia[i].tam = max(sequencia[i].tam, 1+sequencia[j].tam);
28             }
29             if(sequencia[i].tam == 1+sequencia[j].tam) sequencia[i].no = j;

30             if(sequencia[i].tam > aux.tam){
31                 aux.no = i;
32                 aux.tam = sequencia[i].tam;
33                 break;
34             }
35         }

36     maior.clear();
37     int k = aux.no;
38     while(k != -1){
39         maior.push_back(seq[k]);
40         k = sequencia[k].no;
41     }

42     cout << maior.size() << endl << "—" << endl;
43     for(int i=0; i<maior.size(); i++)
44         cout << maior[i] << endl;

45     return 0;
46 }
```

Exemplo #2:

```

/* Programacao Dinamica - Top Down*/
5 #include <vector>
# include <cstdlib>
# include <cstdio>
```

```

#include <cstring>
#define INF 99999

using namespace std;

int C;
int tabela[201][21];
vector<vector<int>> produtos;

int optimal(int g, int m)
{
    if(m < 0)
        return -INF;
    else if(g == C && m >= 0)
        return 0;
    else if(tabela[m][g] != -1)
        return tabela[m][g];
    else
    {
        int v = -INF;

        for(int j = 0; j < produtos[g].size(); j++)
            v = max(produtos[g][j] + optimal(g + 1, m - produtos[g][j]), v);

        tabela[m][g] = v;
    }
}

int main()
{
    int n;

    scanf("%d", &n);

    for(int i = 0; i < n; i++)
    {
        int M;

        scanf("%d %d", &M, &C);

        for(int j = 0; j <= M; j++)
            for(int k = 0; k <= C; k++)
                tabela[j][k] = -1;

        produtos.clear();

        for(int j = 0; j < C; j++)
        {
            int k;
            scanf("%d", &k);
            vector<int> tmp;

            for(int l = 1; l <= k; l++)
            {
                int a;
                scanf("%d", &a);
                tmp.push_back(a);
            }

            produtos.push_back(tmp);
        }
    }
}

```

```

        }
        produtos.push_back(tmp);
    }

    if(optimal(0,M) > 0)
        printf("%d\n", optimal(0,M));
    else
        printf("no solution\n");
}

return 0;
}

```

Exemplo #3:

```

/* Programação Dinamica - Top down - Pedido de Desculpas */

#include <stdio.h>
#include <string.h>

#define max(a, b) a>b?a:b

typedef struct c{
    int tamanho;
    int desculpa;
}TFrase;

int memoize[1010][60];
TFrase vetor[1010];
int num;

int recursivo(int size, int a){
    int b=0, c=0;
    if(a == num) return 0;

    if(memoize[size][a] != -1) return memoize[size][a];

    b = recursivo(size, a+1);
    if(size - vetor[a].tamanho >= 0) c = vetor[a].desculpa +
        recursivo(size-vetor[a].tamanho, a+1);
    memoize[size][a] = max(b, c);

    return memoize[size][a];
}

int main(){
    int tamanhoTotal, caso=1, i;

    while(scanf("%d %d", &tamanhoTotal, &num) && (tamanhoTotal+num)){
        for(i=0; i<num; i++)
            scanf("%d %d", &vetor[i].tamanho, &vetor[i].desculpa);

        for(i=0; i<=tamanhoTotal; i++)
            memset(memoize[i], -1, sizeof(int)*num);

        printf("Teste %d\n%d\n", caso++, recursivo(tamanhoTotal, 0));
    }
}

```

```

    }
    return 0;
}

```

21.15 Wedding Shopping

21.15.1 Especificação

One of our best friends is getting married and we all are nervous because he is the first of us who is doing something similar. In fact, we have never assisted to a wedding, so we have no clothes or accessories, and to solve the problem we are going to a famous department store of our city to buy all we need: a shirt, a belt, some shoes, a tie, etcetera.

We are offered different models for each class of garment (for example, three shirts, two belts, four shoes, ..). We have to buy one model of each class of garment, and just one.

As our budget is limited, we cannot spend more money than it, but we want to spend the maximum possible. It's possible that we cannot buy one model of each class of garment due to the short amount of money we have.

The Input

The first line of the input contains an integer, N , indicating the number of test cases. For each test case, some lines appear, the first one contains two integers, M and C , separated by blanks ($1 \leq M \leq 200$, and $1 \leq C \leq 20$), where M is the available amount of money and C is the number of garments you have to buy. Following this line, there are C lines, each one with some integers separated by blanks; in each of these lines the first integer, K ($1 \leq K \leq 20$), indicates the number of different models for each garment and it is followed by K integers indicating the price of each model of that garment.

The Output

For each test case, the output should consist of one integer indicating the maximum amount of money necessary to buy one element of each garment without exceeding the initial amount of money. If there is no solution, you must print "no solution".

21.15.2 Solução

```

int main(int argc, char** argv)
{
    long long i, j, k, nCases;
    char* read = (char*)malloc((MAXSTRSZ+1)*sizeof(char));
    char c;

```

```

    cin >> nCases;

10   for(long long cases = 0; cases < nCases; cases++)
{
    int money, nObj;
    cin >> money >> nObj;
    vector< vector<int> > objetos;

15   for(i = 0; i < nObj; i++)
{
    int n;
    cin >> n;

20   vector<int> x;
    for(j = 0; j < n; j++)
{
    cin >> k;
    x.push_back(k);
}
25

    objetos.push_back(x);
}

30   vector< vector<int> > dp;
vector<int> x(nObj+1, -1);
x[0] = 0;

35   for(i = 0; i <= money; i++)
{
    dp.push_back(x);
}

40   for(i = 0; i <= money; i++)
{
    for(j = 1; j <= nObj; j++)
{
    for(k = 0; k < objetos[j-1].size(); k++)
{
45        if(i - objetos[j-1][k] < 0)
            continue;
        if(dp[i - objetos[j-1][k]][j-1] == -1)
            continue;

50        int res = dp[i - objetos[j-1][k]][j-1] + objetos[j-1][k];

        if(res <= i && res > dp[i][j])
{
            dp[i][j] = res;
}
55        }
    }
}
55

60   if(dp[money][nObj] < 0)
    cout << "no solution" << endl;
else
    cout << dp[money][nObj] << endl;

```

```

    }
65   return 0;
}

```

Parte III

UVa & Spoj

22 2-SAT

22.1 Spoj-BR - CARDAPIO (Algoritmo de Tarjan)

```

#define NN 4060

vector< int > adj[NN];
int idx[NN], nnnn, low[NN], SCC[NN], s[NN], scc_count, top;
5  bool ins[NN], NO;
void push(int a){
    s[top++] = a;
    ins[a] = 1;
}
10 int pop(){
    ins[s[--top]] = 0;
    return s[top];
}

15 set<int> elements;

void tarjan(int k, bool* prob){
    idx[k] = low[k] = nnnn = nnnn+1;
    push(k);
20    for(typeof(adj[k].begin())it = adj[k].begin(); it != adj[k].end
        (); it++){
        int nadj = (*it);
        if(idx[nadj] == -1){
            tarjan(nadj, prob);
            low[k] = MIN(low[k], low[nadj]);
25        } else if(ins[nadj]) {
            low[k] = MIN(low[k], idx[nadj]);
        }
    }
    if(low[k] == idx[k]){
30        scc_count++;
        int x = -1;
        elements.clear();
        while(x != k)
        {
35            SCC[x = pop()] = scc_count;
            int opp = x%2 ? x-1 : x+1;
            if(elements.find(opp) != elements.end())
                *prob = true;
40        }
    }
}

```

```

        elements.insert(x);
    }
}

/*
 */
50 int main()
{
    std::cout.sync_with_stdio(false);

    char* fst = new char[MAXSTRSZ+1];
    char* snd = new char[MAXSTRSZ+1];
    string av, neg1, neg2;
    int i, curr, n, test = 1;

    //printf("top == %d\n", top);

60    while(cin >> n)
    {
        map<string,int> getId;
        for(i = 0; i<NN; i++)
            adj[i].clear(), idx[i] = -1;

        curr = 0;

        for(i = 0; i<n; i++)
        {
            cin >> fst >> snd;

            if(getId.find(fst) == getId.end())
                getId[fst] = curr++;

75            if(fst[0] == '!')
            {
                fst++;
                if(getId.find(fst) == getId.end())
                    getId[fst] = curr++;
                neg1 = fst;
                fst--;
            }
            else
85            {
                av = string("!!") + string(fst);
                if(getId.find(av) == getId.end())
                    getId[av] = curr++;
                neg1 = av;
            }

            if(getId.find(snd) == getId.end())
                getId[snd] = curr++;

95            if(snd[0] == '!')
            {
                snd++;
                if(getId.find(snd) == getId.end())

```

```

100         getId[snd] = curr++;
101         neg2 = snd;
102         snd--;
103     }
104     else
105     {
106         av = string(" ! ") + string(snd);
107         if(getId.find(av) == getId.end())
108             getId[av] = curr++;
109         neg2 = av;
110     }
111
112     adj[getId[neg1]].push_back(getId[snd]);
113     adj[getId[neg2]].push_back(getId[fst]);
114 }
115 /* for(typeof(getId.begin()) it = getId.begin(); it != getId.end()
116    () ; it++)
117 {
118     cout << it->first << " --> " << it->second << endl;
119 } */
120
121 scc_count = nnnn = top = 0;
122
123 bool issue = false;
124 for(i = 0; i < curr; i++)
125 {
126     if(idx[i] == -1)
127     {
128         bool prob = false;
129         tarjan(i, &prob);
130         if(prob)
131         {
132             issue = true;
133             break;
134         }
135     }
136
137     printf("Instancia %d\n%s\n", test++, issue? "nao": "sim");
138 }
139
140 return 0;
}

```

23 Ad-hoc

23.1 UVa - (120) Stacks of Flapjacks

```

/* Stacks of Flapjacks */
int val[50];
int aux[50];
5
void flip(int posMax)

```

```

{
    int i;

10   for(i = 0; i <= posMax; i++)
    {
        aux[i] = val[i];
    }

15   for(i = posMax; i >= 0; i--)
    {
        val[i] = aux[posMax-i];
    }
}

20 void findMin(int posMax, int len)
{
    int i;
    int maior = val[0], indexMaior = 0;

25   for(i = 1; i <= posMax; i++)
    {
        if(val[i] > maior)
        {
30            maior = val[i];
            indexMaior = i;
        }
    }

35   if(indexMaior == posMax)
        return;

        if(indexMaior == 0)
    {
40        flip(posMax);
        printf("%d ", len-posMax);
        return;
    }

45   flip(indexMaior);
   flip(posMax);
   printf("%d %d ", len-indexMaior, len-posMax);
}

50 /*
 */
int main(int argc, char** argv)
{
    char* read = (char*)malloc(5001 * sizeof(char));
    char* aux = (char*)malloc(5001 * sizeof(char));

    int i, j, k;

60   while(cin.getline(read, 5000) != NULL)
    {
        if(strlen(read) == 0)
            continue;

```

```

65    strcpy(aux, read);

70    char * tok;
    tok = strtok(aux, " ");

75    int nElem = 0;
    while(tok != NULL)
    {
        val[nElem++] = atoi(tok);

75      tok = strtok(NULL, " ");
    }

80    // for(i = 0; i < nElem; i++)
    // {
    //     printf("%d ", val[i]);
    // }
    printf("|n");

85    printf("%s |n", read);

90    int valMax = nElem - 1;
    while(valMax >= 0)
    {
        findMin(valMax--, nElem);
    }
    printf("0|n");
}

95    return EXIT_SUCCESS;
}

```

23.2 UVa - (200) Rare Order

/ A rare book collector recently discovered a book written in an unfamiliar language that used the same characters as the English language. The book contained a short index, but the ordering of the items in the index was different from what one would expect if the characters were ordered the same way as in the English alphabet. The collector tried to use the index to determine the ordering of characters (i.e., the collating sequence) of the strange alphabet, then gave up with frustration at the tedium of the task.*

*You are to write a program to complete the collector's work. In particular, your program will take a set of strings that has been sorted according to a particular collating sequence and determine what that sequence is. */*

```

5 void possOrd(vector<string> words, map<char, set<char> > &fathers)
{
    long long i, j, k;

10   if(words.size() <= 1)
      return;

```

```

/* printf("words:| n");
for(i = 0; i<words.size(); i++)
    cout << '|t' << words[i] << endl; */
15
for(i = 0; i<words.size()-1; i++)
{
    for(j = i+1; j<words.size(); j++)
    {
        fathers[words[j][0]].insert(words[i][0]);
    }
}

char lastStart = words[0][0];
25
vector<string> aval;
for(i = 0; i<words.size(); i++)
{
    if(words[i].size() > 1)
    {
//        printf("palavra avaliada == "); cout << words[i];
//        printf(" , lastStart == %c |n", lastStart);
        if(words[i][0] == lastStart)
        {
//            cout << "entrei1 |n";
            words[i].erase(words[i].begin());
            aval.push_back(words[i]);
        }
        else
        {
//            cout << "entrei2 |n";
            lastStart = words[i][0];
            possOrd(aval, fathers);
            aval.clear();
45
            words[i].erase(words[i].begin());
            aval.push_back(words[i]);
        }
    }
}
possOrd(aval, fathers);
}

void remove(set<char> &conj, char c)
55
{
    set<char>::iterator it = conj.find(c);
    if(it != conj.end())
        conj.erase(it);
}

/*
 *
 */
int main(int argc, char** argv)
65
{
    long long i, j, k;
    char* read = (char*)malloc((MAXSTRSZ+1)*sizeof(char));
    // cin.getline(read, MAXSTRSZ);
}

```

```

70   map<char , set<char> > fathers;
    set<char> charact;

    vector<string> words;
75
80   while(1)
{
    cin.getline(read, MAXSTRSZ);

    if(read[0] == '#' && read[1] == '|0')
        break;

    words.push_back(string(read));
}
85
90   for(i = 0; i<words.size(); i++)
{
    for(j = 0; words[i][j]; j++)
    {
        charact.insert(words[i][j]);
    }
}

set<char> x;
95   for(set<char>::iterator it = charact.begin(); it!=charact.end();
      it++)
    fathers[*it] = x;

poss0rd(words, fathers);
100   for(map<char , set<char> ::iterator it = fathers.begin(); it!=
      fathers.end(); it++)
{
    remove(it->second, it->first);
}

/* printf("map:|n");
105   for (map<char , set<char> ::iterator it = fathers.begin(); it!=
      fathers.end(); it++)
{
    cout << it->first << ":";
    for (set<char>::iterator it2 = it->second.begin(); it2!=it->
      second.end(); it2++)
    {
        cout << " " << *it2;
    }
    cout << endl;
} */

110
115   set<char> already;
    for(i = 0; i<fathers.size(); i++)
{
    for(map<char , set<char> ::iterator it = fathers.begin(); it!=
      fathers.end(); it++)
{
    if(already.find(it->first) == already.end())
    {

```

```

    if(!it->second.size())
125    {
        printf( "%c", it->first);
        for(map<char, set<char> >::iterator it2 = fathers.begin()
            ; it2!=fathers.end(); it2++)
            remove(it2->second, it->first);
        already.insert(it->first);
        break;
130    }
}
printf( " | n");
135 return 0;
}

```

23.3 UVa - (11455) Behold my quadrangle

```

/*
 */
int main(int argc, char** argv)
{
    long long i, j, k;
    char* read = (char*)malloc((MAXSTRSZ+1)*sizeof(char));

    long long nCases;
10
    scanf( "%lld", &nCases);

    for(long long cases = 0; cases < nCases; cases++)
    {
        long long l1, l2, l3, l4;

        scanf( "%lld%lld%lld%lld ", &l1, &l2, &l3, &l4);

        if(l1 == l2 && l1 == l3 && l1 == l4)
20
        {
            printf( "square | n");
        }
        else if((l1 == l2 && l3 == l4) || (l1 == l3 && l2 == l4) || (l1
            == l4 && l2 == l3))
        {
            printf( "rectangle | n");
        }
        else if(l1 < l2 + l3 + l4 && l2 < l1 + l3 + l4 && l3 < l1 + l2
30
            + l4 && l4 < l1 + l2 + l3)
        {
            printf( "quadrangle | n");
        }
        else
        {
            printf( "banana | n");
        }
35
    }
}

```

```

    return 0;
}

```

23.4 Spoj-BR - APAGA

```

char pilha[100010];

/*
 *
 */
5 int main(int argc, char** argv)
{
    int n, m, i, j;
    char* read = (char*)malloc((MAXSTRSZ+1)*sizeof(char));
10
    while(1)
    {
        scanf("%d%d", &n, &m);

15
        if(!n && !m)
            break;

        scanf("%s", read);

20
        if(n == m)
            printf("0\n");
        else
        {
            int totElim = 0;
            int topo = -1;

25
            /*      int ref = 0;
                    while(restantes > 0)
                {
30
                    if(ref == strlen(read)-1)
                    {
                        for(j = ref-restantes+1; j<=ref; j++)
                            read[j] = 'x';
                        restantes = 0;
35
                    }
                    else
                    {
                        int elim = 0;
                        for(i = ref+1, j = 0; j<restantes && i<n; i++, j++)
40
                        {
                            if(read[i] > read[ref])
                            {
                                for(j = ref; j<i; j++)
                                {
                                    read[j] = 'x';
                                }
45
                                restantes -= i-ref;
                                elim = 1;
                                break;
                            }
50
                        }
                    }
                }
            }
        }
    }
}

```

```

        }
        if (!elim)
            ref++;
    }
}

for (i = 0; i < n; i++)
    if (read[i] != 'x')
        printf("%c", read[i]);

printf("\n"); /*

int curr;
for (curr = 0; curr < n; curr++)
{
    if (topo == -1)
    {
        topo++;
        pilha[topo] = read[curr];
    }

    continue;
}

while (pilha[topo] < read[curr])
{
    topo--;
    totElim++;
    if (totElim == m)
        break;
    if (topo == -1)
        break;
}
topo++;
pilha[topo] = read[curr];

if (totElim == m)
    break;
}

topo -= m - totElim;

for (i = 0; i <= topo; i++)
    printf("%c", pilha[i]);
for (curr++; curr < n; curr++)
    printf("%c", read[curr]);
printf("\n");
}
}

return 0;
}

```

23.5 Spoj-BR - CUBOS

```

int cubos[1060][7];
int aux[7]; int add[7];

```

```

int k;

5 void rotateZ(int* x)
{
    aux[1] = x[1];
    aux[6] = x[6];

10   x[1] = x[5];
// x[3] = aux[1];
// x[5] = aux[6];
    x[6] = x[3];

15   x[3] = aux[1];
    x[5] = aux[6];
}

20 void rotateY(int* x)
{
    aux[2] = x[2];
    aux[4] = x[4];

25   x[2] = x[5];
// x[3] = aux[2];
    x[4] = x[3];
// x[5] = aux[4];

30   x[3] = aux[2];
    x[5] = aux[4];
}

35 void rotateX(int* x)
{
    aux[6] = x[6];
    aux[1] = x[1];

40   x[1] = x[2];
// x[2] = aux[6];
// x[4] = aux[1];
    x[6] = x[4];

45   x[2] = aux[6];
    x[4] = aux[1];
}

50 int equals(int* A, int* B)
{
    if(A[1] == B[1] && A[2] == B[2] && A[3] == B[3] && A[4] == B[4]
       && A[5] == B[5] && A[6] == B[6])
        return 1;

    return 0;
}

55 int jaFoi()
{
    int i, j, m;

    for(m = 0; m < k; m++)
}

```

```

60  {
61    rotateZ(add);
62    if>equals(add, cubos[m]))
63      return 1;
64    rotateZ(add);
65    if>equals(add, cubos[m]))
66      return 1;
67    rotateZ(add);
68    if>equals(add, cubos[m]))
69      return 1;
70    rotateZ(add);
71    if>equals(add, cubos[m]))
72      return 1;

75    rotateY(add);
76    rotateZ(add);
77    if>equals(add, cubos[m]))
78      return 1;
79    rotateZ(add);
80    if>equals(add, cubos[m]))
81      return 1;
82    rotateZ(add);
83    if>equals(add, cubos[m]))
84      return 1;
85    rotateZ(add);
86    if>equals(add, cubos[m]))
87      return 1;

90    rotateY(add);
91    rotateZ(add);
92    if>equals(add, cubos[m]))
93      return 1;
94    rotateZ(add);
95    if>equals(add, cubos[m]))
96      return 1;
97    rotateZ(add);
98    if>equals(add, cubos[m]))
99      return 1;

100   rotateY(add);
101   rotateZ(add);
102   if>equals(add, cubos[m]))
103     return 1;
104   rotateZ(add);
105   if>equals(add, cubos[m]))
106     return 1;
107   rotateZ(add);
108   if>equals(add, cubos[m]))
109     return 1;
110   rotateZ(add);
111   if>equals(add, cubos[m]))
112     return 1;
113   rotateZ(add);
114   if>equals(add, cubos[m]))
115     return 1;

116   rotateY(add);

```

```

    rotateX(add);
    rotateZ(add);
120    if>equals(add, cubos[m]))
        return 1;
    rotateZ(add);
    if>equals(add, cubos[m]))
        return 1;
125    rotateZ(add);
    if>equals(add, cubos[m]))
        return 1;
    rotateZ(add);
    if>equals(add, cubos[m]))
        return 1;
130

rotateX(add);
rotateX(add);
rotateZ(add);
135    if>equals(add, cubos[m]))
        return 1;
    rotateZ(add);
    if>equals(add, cubos[m]))
        return 1;
140    rotateZ(add);
    if>equals(add, cubos[m]))
        return 1;
    rotateZ(add);
    if>equals(add, cubos[m]))
        return 1;
145
}

return 0;
}
150
/*
 *
 */
int main()
{
    int n, i, j;

    while(fastint(&n), n)
    {
        k = 0;

        for(i = 0; i < n; i++)
        {
            fastint(&(add[1])), fastint(&(add[2])), fastint(&(add[3])),
                fastint(&(add[4])), fastint(&(add[5])), fastint(&(add[6]))
            ;
        }
165        if(!jaFoi())
        {
            cubos[k][1] = add[1];
            cubos[k][2] = add[2];
            cubos[k][3] = add[3];
            cubos[k][4] = add[4];
            cubos[k][5] = add[5];
            cubos[k][6] = add[6];
170

```

```

175           k++ ;
        }

        printf( "%d\n", k) ;
    }

180    return 0;
}

```

23.6 Spoj-BR - LASERR

```

/*
 *
 */
5 int main()
{
    unsigned short int n, m, v1, v2;
    int j, k, acc;

    while(twoint(&n, &m), n || m)
10    {
        acc = k = 0;

        fastint(&v1);

        for(j = 2; j <= m; j++)
        {
            fastint(&v2);

            acc = v2 > v1 ? acc + v2 - v1 : acc;
20            v1 = v2;
        }

        printf( "%d\n", acc + n - v1);
25    }

    return 0;
}

```

23.7 Spoj-BR - LOOPMUSI

```

/*
 * Imprime o nÃ³mero de ‘‘picos’’ em uma faixa de frequencia
 */
5 int main()
{
    int n, i, ant, seg, av, pic, sent, av2;

    while(scanf("%d", &n), n)
8    {
        if(n == 1)
10        {
            scanf("%d", &av);

```

```

        printf( "1|n");
        continue;
    }

    sent = pic = 0;
    scanf( "%d", &ant); av = ant;
    for(i = 1; i<=n+1; i++)
    {
        if(i == n)
            seg = av;
        else if(i == n+1)
            seg = av2;
        else
            scanf( "%d", &seg);

        if(i == 1)
            av2 = seg;

//        printf("ant == %d , seg == %d|n", ant, seg);

        if(seg > ant)
        {
            if(sent < 0)
            {
//                printf("det %d (%d)|n", i+1, seg);
                pic++;
                sent = 1;
            }
            else if(sent == 0)
            {
                sent = 1;
            }
        }
        else if(seg < ant)
        {
            if(sent > 0)
            {
//                printf("det %d (%d)|n", i+1, seg);
                pic++;
                sent = -1;
            }
            else if(sent == 0)
            {
                sent = -1;
            }
        }

        ant = seg;
    }

    if(!pic)
        pic++;

    printf("%d|n", pic);
}

return 0;
}

```

23.8 Spoj-BR - PARIDADE

```
int binary[106];

void toBinary(long long key, long long* firstDigit)
{
    long long i = 0, div;

    div = key;

    while(div > 0)
    {
        binary[i++] = div%2;
        div >>= 1;
    }
    *firstDigit = i-1;
}

/*
 */
int main()
{
    long long key, fd, tot, i;

    while(1)
    {
        scanf("%lld", &key);

        if(!key)
            break;

        toBinary(key, &fd);

        tot = 0;

        printf("The parity of ");
        for(i = fd; i>=0; i--)
        {
            printf("%d", binary[i]);
            tot+=binary[i];
        }
        printf(" is %lld (mod 2). |n", tot);
    }

    return 0;
}
```

24 AGM

24.1 Spoj-BR - CIPO

```
int arestaN1[2000006];
int arestaN2[2000006];
int weight[2000006];
```

```

5   int father[1006];
  int rank[1006];

10  void makeset(int* x)
{
    father[*x] = *x;
    rank[*x] = 0;
}

15  int fFind(int* x)
{
    while(*x != father[*x])
    {
        *x = father[*x];
    }

20    return *x;
}

25  void join(int* x, int* y)
{
    int rx = fFind(x), ry = fFind(y);

    if(rx == ry)
        return;

30    if(rank[rx] > rank[ry])
        father[ry] = rx;
    else
    {
        father[rx] = ry;
        if(rank[rx] == rank[ry])
            rank[ry]++;
    }
}

40  void kruskal(int nArestas, int* fo, int nV)
{
    int i, j, totAdd;

45    *fo = 0;
    for(i = 1; i <= nV; i++)
    {
        makeset(&i);
    }

50    for(i = 0; i < nArestas; i++)
    {
        if(weight[i] == 1235)
        {
            if(fFind(&(arestaN1[i])) != fFind(&(arestaN2[i])))
            {
                (*fo) += weight[i];
                join(&(arestaN1[i]), &(arestaN2[i]));
            }
        }
    }
}

```

```

for(i = 0; i<nAreastas; i++)
{
    if(weight[i] == 8977)
    {
        if(fFind(&(arestaN1[i])) != fFind(&(arestaN2[i])))
        {
            (*fo)+=weight[i];
            join(&(arestaN1[i]), &(arestaN2[i]));
        }
    }
}

for(i = 0; i<nAreastas; i++)
{
    if(weight[i] == 10923)
    {
        if(fFind(&(arestaN1[i])) != fFind(&(arestaN2[i])))
        {
            (*fo)+=weight[i];
            join(&(arestaN1[i]), &(arestaN2[i]));
        }
    }
}

/*
 *
 */
int main()
{
    int nV, nA, fo, i, cont = 1;

    while(scanf("%d", &nV) == 1)
    {
        scanf("%d", &nA);

        for(i = 0; i<nA; i++)
        {
            scanf("%d%d%d", &(arestaN1[i]), &(arestaN2[i]), &(weight[i]));
            ;
        }

        kruskal(nA, &fo, nV);
    }

    if(cont != 1)
        printf("|\n");

    printf("Instancia %d |n", cont++);
    printf("%d |n", fo);
}

return 0;
}

```

24.2 Spoj-BR - INTERLMG

```

typedef pair<int, double> id;
typedef vector<id> vid;

vi pset(1600), setSize(1600); int _numDisjointSets;
5
void initSet(int N)
{
    setSize.assign(N, 1); _numDisjointSets = N;
    pset.assign(N, 0); for (int i = 0; i < N; i++) pset[i] = i;
10
}

int findSet(int i) { return (pset[i] == i) ? i : (pset[i] = findSet(pset[i])); }

bool isSameSet(int i, int j) { return findSet(i) == findSet(j); }
15
void unionSet(int i, int j)
{
    if (!isSameSet(i, j))
    {
20        _numDisjointSets--;
        setSize[findSet(j)] += setSize[findSet(i)];
        pset[findSet(i)] = findSet(j);
    }
}
25

int numDisjointSets() { return _numDisjointSets; }

int sizeOfSet(int i) { return setSize[findSet(i)]; }

30 vector<vid> AdjList;
vi taken;                                     // global boolean flag
                                                to avoid cycle
priority_queue<id> pq;                     // priority queue to help choose
                                                shorter edges

void process(int vtx)
35
{
    taken[vtx] = 1;
    for (int j = 0; j < AdjList[vtx].size(); j++)
    {
        ii v = AdjList[vtx][j];
40        if (!taken[v.first]) pq.push(ii(-v.second, -v.first));
    }
} // sort by (inc) weight then by (inc) id by using -ve sign to
   reverse order

double dist(double x1, double y1, double x2, double y2)
45
{
    return sqrt((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2));
}

/*
 *
 */
50 int main()
{

```

```

55   int nCases, n, i, j, u, v;
      double w;

56   while(cin >> n, n)
57   {
58     vector< pair<double, double> > coord (n, make_pair(0.,0.));
59
60     for(i = 0; i<n; i++)
61     {
62       cin >> coord[i].first >> coord[i].second;
63     }
64
65     double mst_cost = 0.;

66     if(n > 1)
67     {
68       AdjList.assign(n, vid());
69       vector< pair<double, ii> > EdgeList; // format: weight, two
90         vertices of the edge
91
92       for(i = 0; i<n-1; i++)
93       {
94         for(j = i+1; j<n; j++)
95         {
96           w = dist(coord[i].first, coord[i].second, coord[j].first,
97                     coord[j].second);
98
99           u = i, v = j;
100
101          EdgeList.push_back(make_pair(w, ii(u, v))); // but
102            store it as: (w, a, b)
103          AdjList[u].push_back(ii(v, w));
104        }
105      }
106
107      sort(EdgeList.begin(), EdgeList.end()); // sort by edge
108        weight in O(E log E)
109
110      initSet(n); // all V are disjoint sets initially
111      for (int i = 0; i < EdgeList.size(); i++) // for each edge, O(E)
112      {
113        pair<double, ii> front = EdgeList[i];
114        if (!isSameSet(front.second.first, front.second.second))
115          // if no cycle
116        {
117          // printf("front.first == %lf\n", front.first);
118          if(front.first > mst_cost)
119            mst_cost = front.first; // add the
120              weight of e to MST
121            unionSet(front.second.first, front.second.second);
122              // link endpoints
123            }
124          }
125        }
126
127        printf("%.4lf\n", mst_cost);
128    }

```

```
105 }  
    return 0;  
}
```

25 BFS

25.1 Spoj-BR - FORRO

```
/* Para cada conjunto de teste voce deve imprimir a rota para Chico  
ir da cidade dele para a cidade da festa de Forro, de modo que  
ele freie somente uma vez e atravesse o menor numero de cidades  
possivel. Se ha mais de uma rota, imprima aquela que aparece  
primeiro na entrada (veja o ultimo exemplo de entrada). Se nÃ£o  
ha rota possÃvel, imprima No valid route. Imprima uma linha em  
branco entre cada saÃda. Veja os exemplos a seguir para o  
formato exato de entrada-saÃda. */  
  
typedef struct  
{  
    int id, w;  
} TEdge;  
  
bool comp(const TEdge &a, const TEdge &b)  
{  
    if(a.id == b.id)  
    {  
        return a.w < b.w;  
    }  
  
    return a.id < b.id;  
}  
  
vector< vector<TEdge> > graph;  
vector<TEdge> emp;  
vector<int> ext;  
vector<string> getStr;  
  
bool BFS(int source, int dest, int n)  
{  
    int i;  
    queue< vector<int> > fila;  
    queue<int> speed;  
  
    vector<int> x;  
    x.push_back(source);  
  
    fila.push(x);  
    speed.push(-1);  
  
    while(!fila.empty())  
    {  
        ext = fila.front(); fila.pop();  
        int mins = speed.front(); speed.pop();  
  
        for(i = 0; i < graph[ext.size() - 1].size(); i++)  
        {
```

```

    TEdge aux = graph[ext[ext.size()-1]][i];

    bool tenho = false;
    for(int j = ext.size()-1; j >= 0; j--)
    {
        if(ext[j] == aux.id)
        {
            tenho = true;
            break;
        }
    }

    if(tenho)
        continue;

    if(aux.w >= mins)
    {
        if(aux.id == dest)
        {
            ext.push_back(aux.id);
            return true;
        }
    }

    ext.push_back(aux.id);
    fila.push(ext);
    ext.pop_back();
    speed.push(aux.w);

    //if(aux.id == dest)
    //{
    //    path = enf;
    //    return true;
    /* else if(enf.size() == menorTam)
    {
        bool act = true;

        for(int j = 0; j < menorTam; j++)
        {
            if(enf[j] < path[j])
            {
                act = false;
                break;
            }
            else if(enf[j] > path[j])
            {
                break;
            }
        }

        if(!act)
        {
            path = enf;
        }
    } */
    //}
}
}

```

```

100     }
101     return false;
102 }

103 /*
104 *
105 */
106 int main()
107 {
108 //  std::cout.sync_with_stdio(false);
109 //  srand(time(NULL));

110     int n, curr, val, id1, id2, i;
111     TEdge add;
112     char* name1 = (char*)malloc(2*(MAXSTRSZ+1)*sizeof(char));
113     char* name2 = &(name1[MAXSTRSZ+1]);
114     bool prin = false;

115     while (scanf("%d", &n) == 1)
116     {
117         map<string, int> getId;
118         getStr.clear();
119         graph.assign(506, vector<TEdge>());
120         curr = 0;

121         for (i = 0; i < n; i++)
122         {
123             //      cin >> name1 >> name2 >> val;
124             scanf("%s%s", name1, name2);
125             fastint(&(add.w));

126             if (getId.find(name1) == getId.end())
127             {
128                 id1 = curr;
129                 getId[name1] = curr++;
130                 getStr.push_back(name1);
131             }
132             else
133             {
134                 id1 = getId[name1];
135             }

136             if (getId.find(name2) == getId.end())
137             {
138                 id2 = curr;
139                 getId[name2] = curr++;
140                 getStr.push_back(name2);
141             }
142             else
143             {
144                 id2 = getId[name2];
145             }

146             /* while (graph.size() <= id1)
147             {
148                 graph.push_back(emp);
149             }

```

```

160
    while (graph.size () <= id2)
    {
        graph.push_back (emp);
    } */

// cout << "tam == " << graph.size () << endl;

165
add.id = id2;
graph[id1].push_back (add);
add.id = id1;
graph[id2].push_back (add);
}

170
// exit (0);

for (i = 0; i < curr; i++)
{
    sort (graph[i].begin (), graph[i].end (), comp);
}

/* for (i = 0; i < curr; i++)
{
    cout << getStr[i] << " --->";
    for (int j = 0; j < graph[i].size (); j++)
    {
        cout << " " << getStr[graph[i][j].id];
    }
    cout << endl;
} */

scanf ("%s%s", name1, name2);

190
if (prin)
    printf ("\n");
else
    prin = true;

195
if (BFS (getId[name1], getId[name2], curr))
{
    for (i = 0; getStr[ext[0]][i]; i++)
    {
        putc (getStr[ext[0]][i], stdout);
    }

// cout << getStr[ext[0]];
    for (i = 1; i < ext.size (); i++)
    {
        putc (' ', stdout);
        for (int j = 0; getStr[ext[i]][j]; j++)
        {
            putchar (getStr[ext[i]][j]);
        }
    }
    printf ("\n");
}
else
{
    printf ("No valid route.\n");
}

```

```

        }
    }

    return 0;
220 }
```

25.2 Spoj-BR - NUMERDOS

```

int erdos[2006];

typedef struct
{
    string first, second;
    int erd;
}TAdd;

bool comp(TAdd a, TAdd b)
{
    if(a.second == b.second)
        return a.first < b.first;
    return a.second < b.second;
}

void BFS(map< int ,vector<int> > &graph, int n)
{
    int i;
    int ext, lvl;

    queue<int> fila;
    queue<int> level;

    fila.push(0);
    level.push(0);

    bool* already = (bool*)malloc(n*sizeof(bool));
    for(i = 0; i<n; i++)
        already[i] = false;
30
    already[0] = true;

    while(!fila.empty())
    {
        ext = fila.front(); fila.pop();
        lvl = level.front() + 1; level.pop();

        for(vector<int>::iterator iter = (graph[ext]).begin(); iter
            != (graph[ext]).end(); iter++)
        {
35
            if(!already[*iter])
            {
                erdos[*iter] = lvl;
                fila.push(*iter);
                level.push(lvl);
                already[*iter] = true;
40
            }
        }
45
    }
}
```

```

        }

50    }

void ajeitaStr(char* orig)
{
    int i;

55    for(i = 0; orig[i]; i++)
    {
        if(orig[i] == ',', )
        {
            orig[i] = '/';
            orig[i+1] = '/';
        }
        orig[i-1] = '|0';
    }
}

60    char help[100000];
65    char help2[100000];

70    /*
80     */
int main()
{
    int nPapers, iPapers, i, j, k, currId, cont = 1;
75    char* read = (char*)malloc((MAXSTRSZ+1)*sizeof(char));

80    map< string, int >::iterator it1;
     map< int, string >::iterator it2;
     map< int, vector<int> >::iterator it3;

85    while(1)
    {
        cin.getline(read, MAXSTRSZ);

90        nPapers = atoi(read);

95        if(!nPapers)
            break;

100       currId = 0;
       map< string, int > getID;
       map< int, string > getName;
       map< int, vector<int> > graph;

105      sprintf(read, "P. Erdos");
         getID[string(read)] = currId;
         getName[currId] = string(read);
         currId++;
}
//      printf("nPapers == %d\n", nPapers);

110      for(iPapers = 0; iPapers < nPapers; iPapers++)
{
    vector<int> participantes;
}

```

```

    cin.getline(read, MAXSTRSZ);
    ajeitaStr(read);

    //      printf("read == %s | n", read);

110   char * tok = strtok(read, "/");

        while(tok != NULL)
    {
115       it1 = getID.find(tok);
116       if(it1 == getID.end())
117       {
118           getID[tok] = currId;
119           getName[currId] = tok;
120           participantes.push_back(currId);
121           currId++;
122       }
123       else
124       {
125           participantes.push_back(it1->second);
126       }

127       tok = strtok(NULL, "/");
128   }

129   for(i = 0; i<participantes.size() -1; i++)
130   {
131       for(j = i+1; j<participantes.size(); j++)
132       {
133           (graph[participantes[i]]).push_back(
134               participantes[j]);
135           (graph[participantes[j]]).push_back(
136               participantes[i]);
137       }
138   }

139   for(i = 1; i<=currId; i++)
140       erdos[i] = INFINITO;

141   /*
142     printf("grafo :| n");
143     for(map<int , vector<int> >::iterator it16 = graph.begin();
144         it16!=graph.end(); it16++)
145     {
146         printf("%d ->, it16->first);
147         for(i = 0; i<(it16->second).size(); i++)
148         {
149             cout << " " << (it16->second)[i];
150         }
151         cout << endl;
152     }
153     cout << endl; */

154     BFS(graph, currId);

155     vector<TAdd> elements;

156     TAdd x;

```

```

    for(i = 1; i<currId; i++)
    {
        sprintf(read, "%s", getName[i].c_str());

165     char * tok = strtok(read, " ");
        x.first = string(tok);
        tok = strtok(NULL, " ");
        x.second = string(tok);
        x.erd = erdos[i];

170     elements.push_back(x);
    }

    sort(elements.begin(), elements.end(), comp);

175    printf("Teste %d\n", cont++);
    for(i = 0; i<elements.size(); i++)
    {
        if(elements[i].erd == INFINITO)
        {
            cout << elements[i].first << " " << elements[i].second <<
                ": infinito " << endl;
        }
        else
        {
185            cout << elements[i].first << " " << elements[i].second <<
                ": " << elements[i].erd << endl;
        }
    }
    printf("\n");
}

190 return 0;
}

```

25.3 Spoj-BR - TESOUR11

```

vector<bool> already;
vector< vector<int> > graph;

void BFS(int n, int source, int levInt, set<int> &p)
5 {
    int i, ext, lvl;

    queue<int> fila;
    queue<int> level;

10    fila.push(source);
    level.push(0);

    already.clear();
    for(i = 0; i<n; i++)
15        already.push_back(false);

    while(fila.size())
    {

```

```

20     ext = fila.front(); fila.pop();
      lvl = level.front(); level.pop();

//      printf("ext == %d\n", ext);

25     if(lvl == levInt)
        p.insert(ext);
     if(lvl > levInt)
        break;

30     if(already[ext])
    {
        continue;
    }

35     already[ext] = true;

40     for(i = 0; i<graph[ext].size(); i++)
    {
        if(!(already[graph[ext][i]]))
        {
            fila.push(graph[ext][i]);
            level.push(lvl+1);
        }
    }
45   }

int codifica(int a, int b, int c)
{
    return a*c + b;
}

void decodifica(int v, int n, int* x, int* y)
{
    *x = v/n;
    *y = v%n;
}

void geraVizinhos(int x, int y, int n)
{
    int ref = codifica(x,y,n);
    vector<int> w;
    graph.push_back(w);

65 //  printf("ref == %d\n", ref);

    if(x-1 >= 0)
    {
        graph[ref].push_back(codifica(x-1, y, n));
    }
    if(x+1 < n)
    {
        graph[ref].push_back(codifica(x+1, y, n));
    }
70    if(y-1 >= 0)
    {
        graph[ref].push_back(codifica(x, y-1, n));
    }
75}

```

```

    }
    if(y+1 < n)
    {
        graph[ref].push_back(codifica(x, y+1, n));
    }
}

85 bool atAll(int val, vector< set<int> > &points)
{
    int i, j;

    for(vector< set<int> >::iterator it = points.begin(); it!=points.
        end(); it++)
    {
        bool isNot = true;

        for(set<int>::iterator iter = (*it).begin(); iter != (*it).end
            (); iter++)
        {
            if(*iter == val)
            {
                isNot = false;
                break;
            }
        }

        if(isNot)
            return false;
    }
105    return true;
}

/*
 *
 */
110 int main()
{
    int dim, n, tot, sol, x, y;
    int i, j;

    scanf("%d%d", &dim, &n);
    int lim = dim*dim;

120    for(i = 0; i<dim; i++)
    {
        for(j = 0; j<dim; j++)
        {
            geraVizinhos(i, j, dim);
        }
    }

    /* for (i = 0; i<dim*dim; i++)
    {
        printf("vizinhos %d:", i);
        for(j = 0; j<graph[i].size(); j++)
            printf(" %d", graph[i][j]);
        printf("\n");
    }

```

```

135     } */
140
145     vector< set<int> > points;
150     int v1, v2, level;
155     for(i = 0; i <n; i++)
160     {
165         scanf( "%d%d%d", &v1, &v2, &level);
170
175         set<int> x;
180         BFS(lim, codifica(v1,v2,dim), level, x);
        points.push_back(x);
    }

// printf("here !| n");

/* for(vector< set<int> >::iterator it = points.begin(); it !=
   points.end(); it++)
{
    printf("pontos:");
    for(set<int>::iterator iter = (*it).begin(); iter != (*it).end
        (); iter++)
    {
        printf(" %d", *iter);
    }
    printf(" | n");
}
 */

tot = 0; sol = -1;
for(set<int>::iterator it = points[0].begin(); it!=points[0].end
    (); it++)
{
    if(atAll(*it, points))
    {
        tot++;
        sol = *it;
        if(tot == 2)
            break;
    }
}

if(tot == 1)
{
    decodifica(sol, dim, &x, &y);
    printf( "%d %d| n", x, y);
}
else
    printf( "-1 -1| n");

return 0;
}

```

26 Busca Exaustiva

26.1 Spoj-BR - EUROVIMG

```

/*
 */
int main()
{
    char* v1 = (char*)malloc((MAXSTRSZ+1)*sizeof(char));
    char* v2 = (char*)malloc((MAXSTRSZ+1)*sizeof(char));
    int cases, i, j;
    vector<int> solution (16);

    scanf( "%d", &cases);

    for(int k = 1; k<=cases; k++)
    {
        int n;
        scanf( "%d", &n);

        solution.resize(n);

        map< ii, bool > restriction;
        map<string, int> id;

        int curr = 0;
        for(i = 0; i<n; i++)
        {
            int m;
            scanf( "%s%d", v1, &m);

            if(id.find(v1) == id.end())
                id[v1] = curr++;

            for(j = 0; j<m; j++)
            {
                scanf( "%s", v2);

                if(id.find(v2) == id.end())
                    id[v2] = curr++;

                restriction[ii(id[v1],id[v2])] = true;
                restriction[ii(id[v2],id[v1])] = true;
            }
        }

        for(i = 0; i<n; i++)
            solution[i] = i;

        int acc = 0;
        do
        {
            bool ok = true;
            for(i = 1; i<n; i++)
            {
                if(restriction[ii(solution[i-1],solution[i])])
                {
                    ok = false;
                    break;
                }
            }
        }
    }
}

```

```

        }

60     if(ok)
        acc++;
    } while(next_permutation(solution.begin(), solution.end()));

    printf("Teste #%d: %d\n", k, acc);
65 }

    return 0;
}

```

26.2 Spoj-BR - JUNINA (Clique Máximo)

```

int apt[26][26];
int clique[26];
int tamClique = 0;

5 int addOK(int ind)
{
    int i;

    for(i = 0; i<tamClique; i++)
10    /* printf("i == %d , clique[i] == %d\n", i, clique[i]); */
        printf("apt0 == %d , aptF == %d\n", apt[clique[i]][i], apt[i][
            clique[i]]); */
        if(!(apt[clique[i]][ind] && apt[ind][clique[i]]))
            return 0;
15    }

    return 1;
}

20 void montaClique(int ind, int* maxClique, int n)
{
    int i;

//    printf("cliqSize == "); cout << clique.size() << endl;
25 //    printf("maxClig == %d\n", *maxClique);

    if(tamClique > *maxClique)
    {
        *maxClique = tamClique;
30    }

    for(i = ind; i<n; i++)
    {
        if(addOK(i))
35        {
            clique[tamClique++] = i;
            montaClique(i+1, maxClique, n);
            tamClique--;
        }
40    }
}

```

```

/*
 *
 */
45 int main()
{
    int n, i, j, v1, cont = 1, maxVal;

50    while(scanf("%d", &n), n)
    {
        for(i = 0; i<n; i++)
        {
            for(j = 0; j<n; j++)
            {
                apt[i][j] = 1;
            }
            apt[i][i] = 0;

55        while(scanf("%d", &v1), v1)
        {
            apt[i][v1-1] = 0;
        }
    }

65    /* for (i = 0; i<n; i++)
    {
        printf("%d -->, i);
        for (j = 0; j<neighbors[i].size(); j++)
        {
            printf(" %d", neighbors[i][j]);
        }
        printf("\n");
    }
    printf("\n"); */

70    maxVal = 0;

    montaClique(0, &maxVal, n);

80    printf("Teste %d | n%d | n | n", cont++, maxVal);
}

85    return 0;
}

```

27 Dijkstra

27.1 Spoj - SHPATH

```

typedef pair<int,int> ElementType;
struct HeapStruct;
typedef struct HeapStruct *PriorityQueue;

5 PriorityQueue Initialize(int MaxElements);
void Insert(ElementType X, PriorityQueue H);

```

```

ElementType DeleteMin(PriorityQueue H);
ElementType FindMin(PriorityQueue H);
int IsEmpty(PriorityQueue H);
10 int IsFull(PriorityQueue H);
void display(PriorityQueue H);

#define MinPQSize (10)
#define MinData (-32767)

15 struct HeapStruct
{
    int Capacity;
    int Size;
20 ElementType *Elements;
};

PriorityQueue Initialize(int MaxElements)
{
25    PriorityQueue H;

    H = (struct HeapStruct *) malloc(sizeof ( struct HeapStruct));

    H->Elements = (pair<int,int> *) malloc((MaxElements + 1) * sizeof
30        ( ElementType));
    H->Capacity = MaxElements;
    H->Size = 0;
    H->Elements[ 0 ] = make_pair(MinData, 0);

35    return H;
}

/* END */

40

/* H->Element[ 0 ] is a sentinel */

void Insert(ElementType X, PriorityQueue H)
45 {
    int i;

    for (i = ++H->Size; (H->Elements[ i>>1 ]).first > X.first; i>>=1)
        H->Elements[ i ] = H->Elements[ i>>1 ];
50    H->Elements[ i ] = X;
}

/* END */

55 ElementType DeleteMin(PriorityQueue H)
{
    int i, Child;
    ElementType MinElement, LastElement;
60    MinElement = H->Elements[ 1 ];
    LastElement = H->Elements[ H->Size-- ];
}

```

```

65   for (i = 1; i << 1 <= H->Size; i = Child)
{
    Child = i << 1;
    if (Child != H->Size && (H->Elements[ Child + 1 ]).first < (H->
        Elements[ Child ]).first)
        Child++;

70   if (LastElement.first > (H->Elements[ Child ]).first)
    H->Elements[ i ] = H->Elements[ Child ];
    else
        break;
}
H->Elements[ i ] = LastElement;
return MinElement;
}

80 ElementType FindMin(PriorityQueue H)
{
    if (!IsEmpty(H))
        return H->Elements[ 1 ];
    return H->Elements[ 0 ];
}
85 int IsEmpty(PriorityQueue H)
{
    return H->Size == 0;
}
90 int IsFull(PriorityQueue H)
{
    return H->Size == H->Capacity;
}
95 typedef pair<int, int> ii;
typedef vector<int> vi;
typedef vector<ii> vii;

100 int cases, j, V, E, s, u, v, w;
int i, d;
int tam[10006];
//priority_queue< ii , vector<ii>, greater<ii> > pq;
PriorityQueue pq;
105 vector<vii> AdjList (10006, vii());
int dist[10006];
ii front;

110 int dijkstra(int s, int de)
{
    for(i = 1; i<=V; i++)
        dist[i] = INF;

    dist[s] = 0;
115    Insert(make_pair(0, s), pq);
    while(pq->Size)
    {
//      printf("size == %d\n", pq->Size);
        front = DeleteMin(pq);
120
}
}

```

```

//      printf("size == %d\n", pq->Size);
125     u = front.second;
     if (front.first == dist[u])
        for (j = 0; j != tam[u]; j++)
    {
        if (dist[u] + AdjList[u][j].second < dist[AdjList[u][j].
           first])
        {
            dist[AdjList[u][j].first] = dist[u] + AdjList[u][j].
               second;
//            printf("mandei inserir\n");
130            Insert(make_pair(dist[AdjList[u][j].first], AdjList[u][j].
               first), pq);
//            printf("inseriu\n");
        }
    }
135
     return dist[de];
}

void fastint(register int *n)
140 {
    register char c;

    *n = 0;
    while (!isdigit(c = getc(stdin)));
145
    do
    {
        (*n) = (*n)*10 + (c - '0');
    } while (isdigit( c = getc(stdin) ));
150 }

void faststr(register char *s)
{
    register char c;
    register int i = 0;

    while (c = getc(stdin) , c == ' ' || c == '\n');

    do
160    {
//        printf("c == %c\n", c);
        s[i++] = c;
    } while (c = getc(stdin) , c != ' ' && c != '\n');
    s[i] = '\0';
165 }

/*
 */
170 int main()
{
    char read[MAXSTRSZ+1];
    char s[MAXSTRSZ+1];
    char d[MAXSTRSZ+1];
175

```

```

int i, av;

pq = Initialize(1000000);

180 fastint(&cases);

while(cases--)
{
    fastint(&V);

185 map<string, int> getId;

    int curr = 1;

190 for(i = 1; i <= V; i++)
{
    int m, dest, w;

    scanf("%s", read);
    fastint(&(tam[i]));

195    getId[read] = i;

    for(j = 0; j < tam[i]; j++)
    {
        fastint(&dest); fastint(&w);

        if(j < AdjList[i].size())
        {
            AdjList[i][j] = make_pair(dest, w);
        }
        else
        {
            AdjList[i].push_back(make_pair(dest, w));
        }
    }
}

200 fastint(&av);

205 for(i = 0; i < av; i++)
{
    scanf("%s%s", s, d);

210 //      printf("chamei\n");
    printf("%d\n", dijkstra(getId[s], getId[d]));
}

215 return 0;
}

```

28 Fluxo Máximo

28.1 Spoj-BR - CAVALOS

```

#define MAX_V 210
#define MAX_F 600
#define INF 10000000000

5   int res[MAX_V][MAX_V], mf, f, s, t;
    int dist[MAX_V];
    int p[MAX_V];

    int q[MAX_F];

10  void augment(int v, int minEdge)
{
    if (v == s) { f = minEdge; return; }
    else if (p[v] != -1)
    {
        augment(p[v], minEdge < res[p[v]][v] ? minEdge : res[p[v]][v]); // recursive
        res[p[v]][v] -= f; res[v][p[v]] += f; // update
    }
}

20  /*
 * */
int main()
{
    int V, k, vertex, weight;
    int n, m, a, i, j, v1, v2, u, v, curr, cont = 1;

    while (scanf("%d%d%d", &n, &m, &a) == 3)
    {
        memset(res, 0, sizeof(res));

        int last = n + m + 1;

35        for (i = 1; i <= n; i++)
        {
            fastint(&(res[i][last]));
        }

        for (i = n+1; i < last; i++)
        {
            res[0][i] = 1;
        }

45        for (i = 0; i < a; i++)
        {
            fastint(&v1); fastint(&v2);
            res[n+v2][v1] = 1;
        }

50        s = 0, t = last;

        mf = 0;
        while (1)
        {
            curr = 0;

```

```

f = 0;
for(i = 0; i <= last; i++)
{
    dist[i] = INF;
    p[i] = -1;
}

dist[s] = 0; q[curr++] = s;

while(curr)
{
    u = q[--curr];
    if(u == t)
        break;
    for(v = 0; v <= last; v++)
        if(res[u][v] > 0 && dist[v] == INF)
            dist[v] = dist[u] + 1, q[curr++] = v, p[v] = u;
}
augment(t, INF);

if(!f)
    break;
mf += f;
}

printf("Instancia %d |n%d |n|n", cont++, mf);
}

return 0;
}

```

29 Game Theory

29.1 Spoj-BR - CHOC09

```

/* Carlos e Paula acabaram de ganhar um saco com bolinhas de
chocolate. Como sabem que vao comer tudo muito rapido inventaram
uma brincadeira:

Eles vao comer de forma alternada, um depois o outro, sendo que
sempre a Paula comeca.
A cada vez, so se pode comer de 1 a M bolinhas, sendo o M decidido
pela mae de Paula, de forma que nao engasguem com o chocolate.
Se um comeu K bolinhas em sua vez, o proximo nao pode comer o mesmo
tanto, tendo que comer um nÃºmero de bolinhas distinto.
Quem nao puder mais jogar de maneira valida perde. */

static int count[1024 * 1024];
static int forbid[1024 * 1024];
int solve (int n, int m)
{
    int i, j;

```

```

15   for (i = 0; i <= n; i++)
      count[i] = forbid[i] = 0;

20   for (i = 0; i <= n; i++) {
      if (count[i] == 0) {
          for (j = 1; j <= m; j++) {
              count[i + j]++;
              forbid[i + j] = j;
          }
      } else if (count[i] == 1) {
          count[i + forbid[i]]++;
          forbid[i + forbid[i]] = forbid[i];
      }
  }
  return count[n];
}

30 int main (int argc, char **argv)
{
    int n, m;
    while (1) {
        n = m = 0;
        scanf ("%d %d", &n, &m);
        if (n == 0) break;
        printf ("%s\n", solve (n, m) ? "Paula" : "Carlos");
    }
    return 0;
}

```

29.2 Spoj-BR - CHOCPJ09

```

/* Carlos e Paula acabaram de ganhar um saco com bolinhas de
chocolate. Como sabem que vão comer tudo muito rápido
inventaram uma brincadeira:

Eles vão comer de forma alternada, um depois o outro, sendo que
sempre a Paula começa.
Quem comer a ultima bolinha ganha a brincadeira.
5 A cada vez, só se pode comer de 1 a M bolinhas, sendo o M decidido
pela mão de Paula, de forma que não engasguem com o chocolate.
*/

```

```

int main()
{
    int n, m;
10    fastint(&n), fastint(&m);

    if(n%(m+1))
        puts("Paula");
    else
15        puts("Carlos");

    return 0;
}

```

29.3 Spoj-BR - MARIENBA

```
/*
O jogo comeca com 6 fileiras de palitos. A primeira fileira contem
1 palito, a segunda contem 3, a terceira 5, a quarta 7, a quinta
9 e a sexta 11. Segue abaixo um desenho com o esquema do jogo
inicial.

Participam do jogo duas pessoas, que alternam seus movimentos. Em
cada jogada, uma pessoa deve tirar uma quantidade diferente de
zero de palitos do tabuleiro. Todos os palitos retirados em uma
jogada devem pertencer a mesma fileira. Assim, se uma fileira
contem k palitos e um jogador decide retirar palitos dessa
fileira em sua jogada atual, ele tem k opcoes distintas de
jogadas (podera remover entre 1 e k palitos).

5
Se apos uma jogada o tabuleiro ficar completamente vazio (i.e., sem
palitos em qualquer uma das 6 fileiras), o jogador que realizou
a ultima jogada (o jogador que removeu os ultimos palitos) perde o jogo.

Dada a descriçao de uma configuracão do tabuleiro apos algumas
jogadas, determinar se o jogador que fara a proxima jogada pode
vencer o jogo, assumindo que o adversario e inteligente e
portanto sempre escolhe a melhor jogada possivel.

10 A entrada comeca com um numero inteiro n na primeira linha,
indicando numero de instancias do problema que seu programa deve
resolver. As proximas n linhas contêm a descrição das
instancias. Cada uma dessas linhas contem uma seqüencia de 6
numeros inteiros. O i-esimo numero da seqüencia indica quantos
palitos ainda restam na i-esima fileira de palitos do jogo.
Todos os numeros da seqüencia são validos (ou seja, o i-esimo
inteiro contem um valor entre 0 e o numero de palitos com o
qual a i-esima fileira comeca o jogo).

Para cada instancia, voca deve imprimir um identificador
Instancia k, onde k e o numero da instancia atual. Na linha
seguinte, seu programa deve imprimir sim se o jogador pode
vencer a partida, e nao caso contrario. Imprima uma linha em
branco apos cada instancia.

*/
15 */

int main()
{
    int cases, curr, val, orExc, allLess, n0nes, i;

    fastint(&cases);

    for(curr = 1; curr <= cases; curr++)
    {
        orExc = 0;
        allLess = 1; n0nes = 0;
        for(i = 0; i < 6; i++)
        {
```

```

30     fastint(&val);

        orExc ^= val;

        if(val > 1)
            allLess = 0;
        else if(val)
            n0nes++;
    }

40 if(allLess)
{
    if(n0nes%2)
        printf("Instancia %d\nnao\n", curr);
    else
        printf("Instancia %d\nsim\n", curr);
}
else
{
    if(!orExc)
        printf("Instancia %d\nnao\n", curr);
    else
        printf("Instancia %d\nsim\n", curr);
}
}

55 return 0;
}

```

30 Grafo

30.1 Spoj-BR - TEOBALDO

```

/*
Teobaldo trabalha para o governo brasileiro. No seu trabalho, ele
costuma viajar muito. Quando Teobaldo viaja de uma cidade S para
uma cidade E ele pode gastar ate D dias nesta viagem.

Como Teobaldo nÃ£o gosta de trabalhar, ele sempre gasta o numero
maximo de dias nas suas viagens. Em cada dia da viagem, Teobaldo
dorme em uma cidade diferente da cidade do dia anterior, pois
ele acha chato ficar na mesma cidade dois dias consecutivos.

5 Neste problema, vocÃº deve ajudar Teobaldo a planejar suas viagens.

O arquivo de entrada contem varios conjuntos de teste. A
descriÃ§Ã£o de cada conjunto e dada a seguir:

10 Cada conjunto comeÃ§a com dois inteiros C (0 < C <= 100), o numero
de cidades, e L (0 <= L <= 500), o numero de estradas entre as
cidades. Seguem L linhas, onde cada linha possui dois numeros: A
e B (1 <= A,B <= C), indicando que ha uma estrada entre estas
duas cidades. Voce pode assumir que A e B sao numeros diferentes.
Depois dessas L linhas, ha tres inteiros: S, E e D, onde S e a
cidade onde a viagem deve comecar, E e a cidade onde a viagem

```

deve terminar, e D ($0 \leq D \leq 200$) e o numero maximo de dias para Teobaldo ir de S para E .

A entrada e terminada por um conjunto onde $C = L = 0$. Este conjunto n \tilde{A} o deve ser processado. Ha uma linha em branco entre dois conjuntos de entrada.

Para cada conjunto de entrada produza uma linha de sa \tilde{A} da indicando se Teobaldo pode viajar do jeito que ele deseja. Veja os exemplos a seguir para o formato exato de entrada/sa \tilde{A} da.

```
15    */
16
17    /* Gera uma pertinencia com todos os poss $\tilde{A}$ veis lugares em que ele
18       pode estar dada uma ‘‘rodada’’, r */
19
20    #define MAX_V 106
21
22    int adj[MAX_V][MAX_V];
23    int current[MAX_V];
24    int newCurrent[MAX_V];
25
26    /*
27     *
28     */
29    int main()
30    {
31        int n, m, v1, v2, i, j, k, s, d, t;
32
33        while(scanf("%d%d", &n, &m), n || m)
34        {
35            for(i = 0; i<=n; i++)
36                for(j = 0; j<=n; j++)
37                    adj[i][j] = 0;
38
39            for(i = 0; i<m; i++)
40            {
41                fastint(&v1), fastint(&v2);
42
43                adj[v1][v2] = adj[v2][v1] = 1;
44            }
45
46            for(i = 1; i<=n; i++)
47                current[i] = adj[i][i] = 0;
48
49            fastint(&s), fastint(&d), fastint(&t);
50
51            current[s] = 1;
52
53            for(k = 0; k<t; k++)
54            {
55                for(i = 1; i<=n; i++)
56                    newCurrent[i] = 0;
57
58                for(i = 1; i<=n; i++)
59                {
60                    if(current[i])
61                    {
62                        for(j = 1; j<=n; j++)
63                            if(adj[i][j])
```

```

        {
            if(adj[i][j])
            {
                newCurrent[j] = 1;
            }
        }
    }

65   for(i = 1; i<=n; i++)
    current[i] = newCurrent[i];
}

70   puts(current[d]?"Yes, Teobaldo can travel.":"No, Teobaldo can
not travel.");
}

75   return 0;
}

```

31 Grafo de Intervalos

31.1 Spoj-BR - ATLETAMG

```

vector<int> base (11);
int adj[11][11];
int el[11];
bool nadj[11][11];
5  bool line[11];
bool col[11];

bool isIntervalGraph(int &n, int &rem)
{
10  int i, j, k;
    bool prob;

    vector<int> elem (base.begin(), base.begin() + n + 1);

15  memset(nadj, 0, sizeof(nadj));

    for(i = 1; i<=n; i++)
    {
        if(i == rem) continue;
        for(j = 0; j<el[i]; j++)
        {
20            nadj[i][adj[i][j]] = nadj[adj[i][j]][i] = 1;
        }
    }
25

    do
    {
        prob = false;

30        for(i = 1; i<=n-2; i++)
        {

```

```

    for(j = i+1; j<=n-1; j++)
    {
        for(k = j+1; k<=n; k++)
        {
            if(nadj [elem[i]][elem[k]] && !nadj [elem[j]][elem[k]])
            {
                prob = true;
                break;
            }
            if(prob)
                break;
        }
        if(prob)
            break;
    }

    if(!prob)
    {
        /* printf("== Ordenacao:");
        for(i = 1; i<=n; i++)
            printf("%d", elem[i]);
        printf("\n");
        printf("Adjacencia Final:\n");
        for(i = 1; i<=n; i++)
        {
            printf("| t ");
            for(j = 1; j<=n; j++)
                printf("%s", nadj[i][j]?"1":"0");
            printf("\n");
        }
        printf("== Comparacoes:\n");
        for(i = 1; i<=n-2; i++)
        {
            for(j = i+1; j<=n-1; j++)
            {
                for(k = j+1; k<=n; k++)
                {
                    printf("| t%d,%d %d,%d\n", elem[i], elem[k], elem[j], k);
                }
            }
        }
    } */

        return true;
    }
} while(next_permutation(elem.begin()+1, elem.end()));

return false;
}

/*
*/
85 int main()
{
    std::cout.sync_with_stdio(false);
}

```

```

90      int i, j;
95      for(i = 0; i<=10; i++)
          base[i] = i;

100     int n, m, v;
105
110     while(fastint(&n), n)
    {
        for(i = 1; i<=n; i++)
        {
            fastint(&(el[i]));
            for(j = 0; j<el[i]; j++)
            {
                fastint(&(adj[i][j]));
            }
        }

        i = -1;
        if(isIntervalGraph(n, i))
        {
            puts("nenhum mentiroso");
        }
        else
        {
115            for(i = 1; i<=n; i++)
            {
                if(isIntervalGraph(n, i))
                {
                    printf("%d\n", i);
                    break;
                }
            }
        }
    }

125    return 0;
}

```

32 Grid

32.1 UVa - (121) Pipe Filters

```

/* oooooooo      o o o o o o
   oooooooo  Vs.  o o o o o o
   oooooooo      o o o o o o

5   grid  Vs.  screw */

double a, b, den;
int c1, c2, d, grid, skew, skew1, skew2;

10 int getskew(double &a, double &b)
{

```

```

    c1 = floor(a);
    d = 1 + floor((b-1)/den);
    c2 = a-c1 >= 0.5 ? c1 : c1-1;
15
    return d%2 ? (d/2)*(c1+c2)+c1 : (d/2)*(c1+c2);
}

/*
 */
20
int main()
{
    den = sqrt(3) / 2;
25
    while(scanf("%lf %lf",&a,&b) == 2)
    {
        grid = floor(a) * floor(b);

        skew1 = getskew(a, b);
        skew2 = getskew(b, a);
        skew = MAX(skew1, skew2);

        if(skew > grid)
            printf("%d skew\n",skew);
        else
            printf("%d grid\n",grid);
    }

    return 0;
}

```

33 Java BigInteger

33.1 Spoj - FCTRL2

```

/* Calculo de fatoriais que extrapolariam o ll */

public class Main
{                                         // standard Java class name in UVa
    OJ
    public static void main(String[] args)
    {
        Hashtable values = new Hashtable();
        BigInteger fac = BigInteger.ONE;           // :)
10
        for (int i = 1; i <= 100; i++)
        {
            fac = fac.multiply(BigInteger.valueOf(i)); // wow :)
            values.put(i, fac);
        }

        Scanner sc = new Scanner(System.in);
15
        int n, key;

```

```

        n = sc.nextInt();
20
    while(n-- > 0)
    {
        key = sc.nextInt();

25        System.out.println(values.get(key));
    }
}
}

```

33.2 Spoj - JULKA

```

public class Main
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);

        for(int i = 0; i<10; i++)
        {
            BigInteger v1, v2, v3, v4;
10
            v1 = sc.nextBigInteger();
            v2 = sc.nextBigInteger();

            v3 = v1.subtract(v2);
            v3 = v3.divide(new BigInteger("2"));

            v4 = v2.add(v3);

            System.out.print(v4 + " | " + v3 + " | ");
20
        }
    }
}

```

34 Java GregorianCalendar

34.1 UVa - (11947) Cancer of Scorpio

```

public class Main
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);

        // get the supported ids for GMT-08:00 (Pacific Standard Time)
5       String[] ids = TimeZone.getAvailableIDs(-8 * 60 * 60 * 1000);

        // create a Pacific Standard Time time zone
10      SimpleTimeZone pdt = new SimpleTimeZone(-8 * 60 * 60 * 1000,
            ids[0]);
}

```

```

// set up rules for daylight savings time
pdt.setStartRule(Calendar.APRIL, 1, Calendar.SUNDAY, 2 * 60 *
60 * 1000);
pdt.setEndRule(Calendar.OCTOBER, -1, Calendar.SUNDAY, 2 * 60 *
60 * 1000);

15   Calendar calendar = new GregorianCalendar(pdt);

20   int cases = sc.nextInt();

25   for(int k = 1; k<=cases; k++)
{
    int all = sc.nextInt(), dd, mm, yyyy, d, m, y;

30     yyyy = all%10000;
     dd = (all/10000)%100;
     mm = (all/1000000)%100;

35     // System.out.printf("yyyy == %

40     Date trialTime = new Date((yyyy - 1900), mm-1, dd+1);
     calendar.setTime(trialTime);

45     calendar.add(Calendar.DATE, 40*7);

50     d = calendar.get(Calendar.DAY_OF_MONTH); m = calendar.get(
         Calendar.MONTH); y = calendar.get(Calendar.YEAR);

55     String sign = "";

60     if(m == 0)
    {
        if(d >= 21)
        {
            sign = "aquarius";
        }
        else
        {
            sign = "capricorn";
        }
    }
65     else if(m == 1)
    {
        if(d >= 20)
        {
            sign = "pisces";
        }
        else
        {
            sign = "aquarius";
        }
    }
    else if(m == 2)
    {
        if(d >= 21)
        {
            sign = "aries";
        }
    }
}

```

```

    else
    {
        sign = "pisces";
    }
}
else if(m == 3)
{
    if(d >= 21)
    {
        sign = "taurus";
    }
    else
    {
        sign = "aries";
    }
}
else if(m == 4)
{
    if(d >= 22)
    {
        sign = "gemini";
    }
    else
    {
        sign = "taurus";
    }
}
else if(m == 5)
{
    if(d >= 22)
    {
        sign = "cancer";
    }
    else
    {
        sign = "gemini";
    }
}
else if(m == 6)
{
    if(d >= 23)
    {
        sign = "leo";
    }
    else
    {
        sign = "cancer";
    }
}
else if(m == 7)
{
    if(d >= 22)
    {
        sign = "virgo";
    }
    else
    {
        sign = "leo";
    }
}

```

```

        }
    }
    else if(m == 8)
    {
        if(d >= 24)
        {
            sign = "libra";
        }
        else
        {
            sign = "virgo";
        }
    }
    else if(m == 9)
    {
        if(d >= 24)
        {
            sign = "scorpio";
        }
        else
        {
            sign = "libra";
        }
    }
    else if(m == 10)
    {
        if(d >= 23)
        {
            sign = "sagittarius";
        }
        else
        {
            sign = "scorpio";
        }
    }
    else if(m == 11)
    {
        if(d >= 23)
        {
            sign = "capricorn";
        }
        else
        {
            sign = "sagittarius";
        }
    }

    System.out.printf("%d %02d/%02d/%04d", k, m+1, d, y);
    System.out.println(' ' + sign);
}

// create a GregorianCalendar with the Pacific Daylight time
// zone
// and the current date and time

```

```

//      calendar.setTime(trialTime);

185     // print out a bunch of interesting things
/* System.out.println("ERA: " + calendar.get(Calendar.ERA));
System.out.println("YEAR: " + calendar.get(Calendar.YEAR));
System.out.println("MONTH: " + calendar.get(Calendar.MONTH));
System.out.println("WEEK_OF_YEAR: " + calendar.get(Calendar.
    WEEK_OF_YEAR));
190    System.out.println("WEEK_OF_MONTH: " + calendar.get(Calendar.
    WEEK_OF_MONTH));
System.out.println("DATE: " + calendar.get(Calendar.DATE));
System.out.println("DAY_OF_MONTH: " + calendar.get(Calendar.
    DAY_OF_MONTH));
System.out.println("DAY_OF_YEAR: " + calendar.get(Calendar.
    DAY_OF_YEAR));
System.out.println("DAY_OF_WEEK: " + calendar.get(Calendar.
    DAY_OF_WEEK)); */
195    }
}

```

35 Joseph

35.1 UVa - (151) Power Crisis

```

vi source;

bool thirteenIsLast(int n, int m)
{
5    vi elim;
    elim.assign(source.begin(), source.begin() + n);

    int curr = 0;

10   while(elim.size() > 1)
    {
//        printf("elim == %d == %d | n", curr, elim[curr]);

        if(elim[curr] == 12)
            return false;

15        elim.erase(elim.begin() + curr);

        curr = (-1 + curr + m) % elim.size();
    }

20   return true;
}

25 /*
 */
int main()
{
30   int i;

```

```

35   for(i = 0; i<=106; i++)
{  

    source.push_back(i);
}  

int n;  

while(FI(n), n)
40 {  

    int m;  

    for(m = 0; 1; m++)  

    {  

        if(thirteenIsLast(n, m))  

45         break;  

    }  

    printf("%d | n", m);  

}  

return 0;
50 }

```

35.2 UVa - (305) Joseph

```

vi source;  

bool allRight(int n, int m)
{  

5   vi elim;  

    elim.assign(source.begin(), source.begin() + n);  

    int lim = n >> 1;  

    int last = lim - 1;  

10   int totBad = 0;  

    int curr = m-1;  

// printf("ATT == %d | n", curr);  

15   while(1)
{  

    curr %= elim.size();  

20 //     printf("curr == %d , elm == %d | n", curr, elim[curr]);  

    if(elim[curr] <= last)
        return false;  

25   elim.erase(elim.begin() + curr);
    totBad++;  

    if(totBad == lim)
        return true;  

30   curr += m-1;
}

```

```

    return true;
35 }

/*
 *
 */
40 int main()
{
    /*int i;

    for (i = 0; i <=106; i++)
    {
        source.push_back(i);
    }

    int n;

    for (n = 1; n<=16; n++)
    {
        int m;

        for (m = 1; 1; m++)
        {
            // printf("m == %d\n", m);
            if (allRight(n << 1, m))
                break;
        }
        printf("\tans[%d] = %d\n", n, m);
    }*/
}

map<int ,int > ans;

65 ans[1] = 2;
ans[2] = 7;
ans[3] = 5;
ans[4] = 30;
70 ans[5] = 169;
ans[6] = 441;
ans[7] = 1872;
ans[8] = 7632;
ans[9] = 1740;
75 ans[10] = 93313;
ans[11] = 459901;
ans[12] = 1358657;
ans[13] = 2504881;
ans[14] = 13482720;

80 int n;

while(FI(n), n)
{
    printf("%d\n", ans[n]);
}

return 0;
}

```

35.3 UVa - (10015) Joseph Cousin

```
vi source;
vi primes;

int getLast(int n)
5 {
    vi elim;
    elim.assign(source.begin(), source.begin() + n);

    int p = 0;
    int curr = 0;

    // printf("ATT == %d | n", curr);

    while(elim.size() > 1)
15 {
        curr = (curr + primes[p++] - 1) % elim.size();

        // printf("curr == %d , elm == %d | n", curr, elim[curr]);

        elim.erase(elim.begin() + curr);
20    }

    return elim[0];
}
25

bool isPrime(int &val)
{
    int lim = ceil(sqrt(val));

    for(int i = 3; i<=lim; i++)
30        if(val % i == 0)
            return false;

    return true;
}
35

/*
 *
 */
40 int main()
{
    /*int i;

    for(i = 0; i<=106; i++)
45    {
        source.push_back(i);
    }

    int n;
50    for(n = 1; n<=16; n++)
    {
        int m;

        for(m = 1; 1; m++)
55        {
```

```

//      printf("m == %d | n ", m);
if(allRight(n << 1, m))
    break;
}
printf("|tans[%d] = %d | n ", n, m);
}/*
int i;
for(i = 0; i<=3600; i++)
{
    source.push_back(i);
}
primes.push_back(2);
primes.push_back(3);
primes.push_back(5);
primes.push_back(7);
primes.push_back(11);
primes.push_back(13);

for(i = 17; primes.size()<3501; i+=2)
{
    if(isPrime(i))
        primes.push_back(i);
}

int n;
while(FI(n), n)
{
    printf("%d | n ", getLast(n) + 1);
}
return 0;
}

```

36 Math

36.1 Spoj-BR - COMCAMEL

```

/*
3
1+2*3*4+5
5
4*18+14+7*10
3+11+4*1*13*12*8+3*3+8

The maximum and minimum are 81 and 30.
The maximum and minimum are 1560 and 156.
10 The maximum and minimum are 339768 and 5023.

*/
char* signals;

```

```

15 | void getSignals(char* str)
| {
|     long long i, k = 0;
|     for(i = 0; str[i]; i++)
|     {
|         if(str[i] == '+' || str[i] == '*')
|             signals[k++] = str[i];
|     }
| }
25 |
long long mult(char* str)
{
    getSignals(str);
    long long aux, lim = strlen(str), i;
    stack<long long> values;

    char * tok = strtok(str, "+* ");
    long long curr = 0;

    values.push(atoi(tok));

    tok = strtok(NULL, "+* ");
    while(tok!=NULL)
    {
        if(signals[curr] == '+')
        {
            aux = values.top(); values.pop();
            aux += atoi(tok);
            values.push(aux);
        }
        else
        {
            values.push(atoi(tok));
        }
45
        curr++;
        tok = strtok(NULL, "+* ");
    }

    long long res = 1;

    while(values.size())
    {
        aux = values.top(); values.pop();
60 //        prlong longf("aux == %d\n", aux);

        res*=aux;
    }

    return res;
}

long long add(char* str)
{
    getSignals(str);
    long long aux, lim = strlen(str), i;
    stack<long long> values;

```

```

    char * tok = strtok(str, "/* ");
75   long long curr = 0;

    values.push(atoi(tok));

    tok = strtok(NULL, "/* ");
80   while(tok!=NULL)
{
    if(signals[curr] == '*')
    {
        aux = values.top(); values.pop();
        aux *= atoi(tok);
        values.push(aux);
    }
    else
    {
        values.push(atoi(tok));
    }

    curr++;
    tok = strtok(NULL, "/* ");
95 }

long long res = 0;

while(values.size())
100 {
    aux = values.top(); values.pop();
//    prlong longf("aux == %d\n", aux);

    res+=aux;
105 }

return res;
}

/*
 */
int main()
{
110   char* read = (char*)malloc(3*(MAXSTRSZ+1)*sizeof(char));
signals = &(read[MAXSTRSZ+1]);
char* read2 = &(signals[MAXSTRSZ+1]);
long long n;

120   cin.getline(read, MAXSTRSZ);

n = atoi(read);

while(n--)
125 {
    cin.getline(read, MAXSTRSZ);
    sprintf(read2, "%s", read);

    printf("The maximum and minimum are %lld and %lld.\n", mult(
        read), add(read2));
}

```

```

130    }
    return 0;
}

```

36.2 Spoj-BR - FATORIAL

```

/* Joaozinho e um garoto esperto da sexta serie. Ele gosta muito de
matematica, e descobriu que sua professora e muito preguiçosa.
Nas provas da materia a professora pede que as crianças circulem a resposta com um quadrado colorido, e que façam o
primeiro digito diferente de zero (da direita para esquerda) do
numero especialmente grande com caneta. Joaozinho desconfiou que
a professora olhava apenas para aquele digito para corrigir a
questao.

A turma aprendeu a calcular o fatorial de um numero, e isso sera
cobrado na proxima prova. Joaozinho esta convencido de que nao
precisa escrever de fato o numero correto, desde que o primeiro
digito (olhando da direita para esquerda) seja o correto. Sua
tarefa neste problema e ajudar Joaozinho a calcular para um
numero inteiro n da entrada, o primeiro digito (da direita para
esquerda) de  $n!$  que seja diferente de zero. */

5  char fatorial[1000006];

10 int myPow(int a, int b)
{
    if(b == 0)
        return 1;

    long long res = myPow(a, b>>1);
    res = b%2?res*res*a:res*res;

15    return res%10;
}

20 /*
 */
int main()
{
    int n, x = 1, t = 0, f = 0, i, k, cont = 1;

25    fatorial[0] = 1;

    for(i = 1; i <= 1000001; i++)
    {
        k = i;

30        while (k % 2 == 0) { k >>= 1; t++; }
        while (k % 5 == 0) { k /= 5; f++; }

        if(t > f)
        {
            t -= f;
        }
    }
}

```

```

        f = 0;
    }
    else
    {
        f -= t;
        t = 0;
    }

    x = (x*k)%10;
    fatorial[i] = (x * myPow(2, t) * myPow(5, f))%10;
}

while(scanf("%d", &n) == 1)
{
    printf("Instancia %d | n%d | n | n", cont++, fatorial[n]);
}

return 0;
}

```

36.3 Spoj-BR - MACACO

```

/*
 * INTERSEÇÃO DE QUADRADOS
 */
int main()
{
    int n;
    int INTERINFX, INTERINFY, INTERSUPX, INTERSUPY;
    int xInf, yInf, xSup, ySup;

    int i, test = 0;

    while(1)
    {
        scanf("%d", &n);

        if(!n)
            break;

        scanf("%d%d%d%d", &INTERINFX, &INTERSUPY, &INTERSUPX, &
              INTERINFY);

//        printf("valores: %d %d %d %d | n", INTERINFX, INTERINFY,
//               INTERSUPX, INTERSUPY);
        for(i = 1; i<n; i++)
        {
            scanf("%d%d%d%d", &xInf, &ySup, &xSup, &yInf);

            INTERINFX = xInf > INTERINFX?xInf:INTERINFX;
            INTERINFY = yInf > INTERINFY?yInf:INTERINFY;
            INTERSUPX = xSup < INTERSUPX?xSup:INTERSUPX;
            INTERSUPY = ySup < INTERSUPY?ySup:INTERSUPY;

//            printf("valores: %d %d %d %d | n", INTERINFX, INTERINFY,
//                   INTERSUPX, INTERSUPY);
        }
    }
}

```

```

        }

35     if(INTERSUPX >= INTERINFX && INTERSUPY >= INTERINFY)
    {
        printf( "Teste %d | n%d %d %d %d | n | n", ++test, INTERINFX,
            INTERSUPY, INTERSUPX, INTERINFY);
    }
    else
    {
40        printf( "Teste %d | nnenhum | n | n", ++test);
    }
}

45     return 0;
}

```

37 Median Generator + LIS

37.1 Spoj-BR - ACOES2MG

```

int max_heap[200004];
int min_heap[200004];

void swap(int *a, int *b)
{
    (*a) ^= (*b);
    (*b) ^= (*a);
    (*a) ^= (*b);
}

10 void insert_max(int* n, int val)
{
    (*n)++;

15    int key = (*n);

    max_heap[key] = val;

    while((key>>1) > 0 && max_heap[key>>1] < val)
20    {
        swap(&(max_heap[key]), &(max_heap[key>>1]));
        key >>= 1;
    }
}

25 void delete_max(int* n)
{
    int key = 1, max;

    max_heap[key] = max_heap[*n];
    max_heap[*n] = 0;
    (*n)--;

30    int l = key << 1;

    if(max_heap[l] > max_heap[l+1])

```

```

        max = l;
    else
        max = l+1;
    while((key<<1) <= (*n) && max_heap[key] < max_heap[max])
    {
        swap(&(max_heap[key]), &(max_heap[max]));
        key = max;
        l = key << 1;
        if(max_heap[l] > max_heap[l+1])
            max = l;
        else
            max = l+1;
    }
}
/* MAX HEAP IMPLEMENTATION ENDS */

/* MIN HEAP IMPLEMENTATION STARTS */

void insert_min(int* n, int val)
{
    (*n)++;
    int key = (*n);
    min_heap[key] = val;

    while( (key>>1) > 0 && min_heap[key >> 1] > min_heap[key])
    {
        swap(&(min_heap[key]), &(min_heap[key>>1]));
        key >>= 1;
    }
}

int findmin(int key)
{
    int l = key << 1;
    if(min_heap[1] == -1)
        return (l+1);
    if(min_heap[l+1] == -1)
        return l;
    if(min_heap[l] < min_heap[l+1])
        return l;
    else
        return l+1;
}

void delete_min(int* n)
{
    int key = 1, min;
    min_heap[key] = min_heap[(*n)];
    min = findmin(key);

    while((key<<1) <= (*n) && min_heap[key] > min_heap[min])
    {
        swap(&(min_heap[key]), &(min_heap[min]));
        key = min;
        min = findmin(key);
    }

    min_heap[(*n)] = -1;
}

```

```

95      (*n)--;
}
/* MIN HEAP IMPLEMENTATION ENDS */

100     void fastint(register int *n)
{
    register char c;

105     *n = 0;
    while(c = getc(stdin), c < '0' || c > '9');

    do
    {
        (*n) = (*n)*10 + (c - '0');
    } while(c = getc(stdin), c >= '0' && c <= '9');
}

110     int n, m;
int nums[200106], b[200106], tot;

115     int lis()
{
    int tail, i, p, u, v;

    tail = -1;

120     if(tot <= 1)
        return tot;

    b[++tail] = 0;

125     for(i = 1; i < tot; i++)
    {
        if(nums[b[tail]] < nums[i])
        {
            b[++tail] = i;
            continue;
        }

        for(u = 0, v = tail; u < v;)
        {
            int c = (u + v) / 2;
            if (nums[b[c]] < nums[i]) u=c+1; else v=c;
        }

        if(nums[i] < nums[b[u]])
        {
            b[u] = i;
        }
    }
}

145     return tail+1;
}

/*
 */
150     int main()

```

```

{
    int i , j ;
    int *read , *medians ;

155
    read = (int*)malloc(200106*sizeof(int));
    medians = (int*)malloc(200106*sizeof(int));

    while(fastint(&n) , n)
    {
        memset(min_heap,-1,sizeof(min_heap));
        memset(max_heap,0,sizeof(max_heap));

        map<int,int> getInd;

165
        for(i = 0; i < n; i++)
            fastint(&(read[i]));

        int m , v;
        fastint(&m);

        for(i = 0; i < m; i++)
        {
            fastint(&v);
            getInd[v] = i;
        }

        int max_heap_size = 0 , min_heap_size = 0 , median = read[0];

180
        int k = 0;
        for(i = 1 ; i < n ; i++)
        {
//            printf("%d\n", median);
            medians[k++] = median;
            if(read[i] < median )
            {
                insert_max(&(max_heap_size), read[i]);
            }
            else
            {
                insert_min(&(min_heap_size), read[i]);
            }

            if( max_heap_size > min_heap_size )
            {
                insert_min(&(min_heap_size), median);
                median = max_heap[1];
                delete_max(&(max_heap_size));
            }
200
            else if( min_heap_size > max_heap_size + 1 )
            {
                insert_max(&(max_heap_size), median);
                median = min_heap[1];
                delete_min(&min_heap_size);
            }
        }
    }

    medians[k] = median;
}

```

```

210     tot = 0;
211     for(i = 0; i < n; i++)
212     {
213         map<int,int>::iterator it = getInd.find(medians[i]);
214
215         if(it != getInd.end())
216         {
217             nums[tot++] = it->second;
218         }
219     }
220
221     /* printf("nums ==");
222      for(i = 0; i < tot; i++)
223          printf("%d ", nums[i]);
224      printf("\n"); */
225
226     printf("%d\n", lis());
227
228     return 0;
229 }
```

38 Ordenação Topológica

38.1 Spoj-BR - ESCALO11

```

/*
Seu objetivo sera decidir se e possivel executar as tarefas em
alguma ordem. Caso seja possivel, voce devera produzir uma ordem
de execuÃ§Ã£o otima para as tarefas, isto e, desempate as
ordens possiveis pela prioridade da primeira tarefa. Se o empate
ainda persistir, desempate pela prioridade da segunda tarefa, e
assim por diante.
```

A primeira linha da entrada contem inteiros N e M . As proximas M linhas descrevem, cada uma, uma dependencia entre as tarefas da entrada. Cada uma dessas linhas ira conter dois inteiros A e B que indicam que a tarefa B depende da tarefa A , isto e, que a tarefa A deve terminar antes que a tarefa B inicie.

5 Se nÃ£o for possivel ordenar as tarefas de forma que as dependencias sejam satisfeitas, imprima uma unica linha contendo o caracter "*". Caso contrario, imprima N linhas contendo cada uma um numero inteiro. O inteiro na i -esima linha deve ser o indice da i -esima tarefa a ser executada na ordem otima de execuÃ§Ã£o das tarefas.

*/

```

10    typedef pair<int, int> ii;
11    typedef vector<ii> vii;
12    typedef vector<int> vi;

#define MAX 56000
#define MMAX 206000
```

15

```

struct Node
{
    int v;
    Node * nxt;
};

Node pilha[MMAX];
int pi;

Node * aloca(int v, Node * nxt)
{
    pilha[pi].v=v;
    pilha[pi].nxt=nxt;
    return &pilha[pi++];
}

Node * g[MAX];
int gr[MAX];
int resp[MAX];
int nresp;
set<int> pq;

int main()
{
    int n,m;
    int i;
    int a,b;

    fastint(&n);
    fastint(&m);

    pi=0;
    nresp=0;

    for (i=0;i<n;++i)
    {
        gr[i]=0;
        g[i]=NULL;
    }

    for (i=0;i<m;++i)
    {
        fastint(&a); fastint(&b);
        ++gr[b];
        g[a]=aloca(b,g[a]);
    }

    for (i=0;i<n;++i)
        if(gr[i]==0)
            pq.insert(i);

    while (!pq.empty())
    {
        int now = *pq.begin();
        pq.erase(pq.begin());
        resp[nresp++]=now;

        for (Node * ar=g[now]; ar!=NULL; ar=ar->nxt)

```

```

75     if((--gr[ar->v])==0)
76         pq.insert(ar->v);
77
78     if(nresp<n)
79         printf(" * | %n");
80     else
81         for(i=0;i<n;++i)
82             printf("%d | %n",resp[i]);
83
84     return 0;
85 }

```

38.2 Spoj-BR - TAREFMG

```

typedef struct
{
    int id, duracao, min, max;
    vector<int> pais;
    vector<int> reverse;
}TJob;

vector<TJob> jobs;

bool comp(TJob a, TJob b)
{
    return a.id < b.id;
}

/*
 *
 */
int main()
{
    int n, m;
    int i, j, mmin, endd;
    bool all, thereis;

    while(fastint(&n), n)
    {
        jobs.resize(n);
        for(i = 0; i<n; i++)
        {
            fastint(&jobs[i].id), fastint(&jobs[i].duracao), fastint(&m);

30       jobs[i].min = -1;
            jobs[i].max = INFINITO;
            jobs[i].pais.resize(m);
            jobs[i].reverse.clear();

35       for(j = 0; j<m; j++)
        {
            fastint(&jobs[i].pais[j]);
        }
    }
}

```

```

        sort(jobs.begin(), jobs.end(), comp);

    45   for(i = 0; i<n; i++)
    {
        for(j = 0; j<jobs[i].pais.size(); j++)
        {
            jobs[jobs[i].pais[j]].reverse.push_back(i);
        }
    }

    thereis = true;
    endd = -1;
    while(thereis)
    {
        thereis = false;
        for(i = 0; i<n; i++)
        {
            if(jobs[i].min != -1)
                continue;

            thereis = true;

            if(!jobs[i].pais.size())
            {
                jobs[i].min = 0;
                if(jobs[i].min + jobs[i].duracao > endd)
                    endd = jobs[i].min + jobs[i].duracao;
            }
        70      else
        {
            mmin = -1;
            all = true;
            for(j = 0; j<jobs[i].pais.size(); j++)
            {
                if(jobs[jobs[i].pais[j]].min == -1)
                {
                    all = false;
                    break;
                }
            80            else
            {
                if(jobs[jobs[i].pais[j]].min + jobs[jobs[i].pais[j]].duracao > mmin)
                    mmin = jobs[jobs[i].pais[j]].min + jobs[jobs[i].pais[j]].duracao;
            }
        }

        if(all)
        {
            jobs[i].min = mmin;
            if(jobs[i].min + jobs[i].duracao > endd)
                endd = jobs[i].min + jobs[i].duracao;
        }
    }
}

```

```

    thereis = true;
100   while(thereis)
    {
        thereis = false;
        for(i = 0; i<n; i++)
        {
            if(jobs[i].max != INFINITO)
105             continue;

            thereis = true;

            if(!jobs[i].reverse.size())
110            {
                jobs[i].max = jobs[i].min + endd - jobs[i].min - jobs[i].
                               duracao;
            }
            else
            {
115                mmin = INFINITO*10;
                all = true;
                for(j = 0; j<jobs[i].reverse.size(); j++)
                {
                    if(jobs[jobs[i].reverse[j]].max == INFINITO)
120                    {
                        all = false;
                        break;
                    }
                    else
                    {
125                        if(jobs[jobs[i].reverse[j]].max < mmin)
                            mmin = jobs[jobs[i].reverse[j]].max;
                    }
                }
130                if(all)
                {
                    jobs[i].max = mmin - jobs[i].duracao;
                }
135            }
        }
    }

    printf("Prazo : %d dias\n", endd);
140   for(i = 0; i<n; i++)
    {
        printf("Tarefa #%-d: min=%d, max=%d\n", i, jobs[i].min, jobs[i].
                           .max);
    }
    printf("----\n");
145   }

    return 0;
}

```

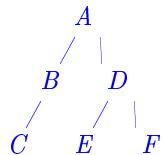
39 Percorso em Árvores

39.1 Spoj-BR - PREEMPOS

```
/* Um problema comum em estruturas de dados é determinar o
   percorrido de uma árvore binária. Existem três maneiras
   clássicas de fazer isso:

   Pre-ordem: Você deve visitar primeiro a raiz, depois a sub-árvore
   esquerda e por último a sub-árvore direita.
   Em-ordem: Você deve visitar primeiro a sub-árvore esquerda, depois
   a raiz e por último a sub-árvore direita.
   Pos-ordem: Você deve visitar primeiro a sub-árvore esquerda,
   depois a sub-árvore direita e por último a raiz.

   O resultado do percurso em pre, em e pos-ordem é,
   respectivamente: ABCDEF, CBAEDF e CBEFDA. Neste problema,
   você deve computar o percurso em pos-ordem de uma árvore
   binária dados os seus percursos em-ordem e pre-ordem.
```



```
O resultado do percurso em pre, em e pos-ordem é, respectivamente:
ABCDEF, CBAEDF e CBEFDA.
```

```
/*
char pre[60], em[60], position[60];
int ind;

void imprimePos(int l, int r)
{
    if(l > r)
        return;

    char ch = pre[ind++];
    imprimePos(l, position[ch] - 1);
    imprimePos(position[ch] + 1, r);

    putc(ch, stdout);
}

/*
 *
*/
int main()
{
    int cases, n, i;

    fastint(&cases);
    while(cases--)
    {
        scanf("%d%s%s", &n, pre, em);
```

```

45     for(i = 0; em[i]; i++)
{
    position[em[i]] = i;
}

50     ind = 0;
    imprimePos(0, n-1);
    putc(' |n', stdout);
}

55     return 0;
}

```

40 Pontos de Articulação

40.1 Spoj-BR - MANUT

```

#define MAX 406

int graph[MAX][MAX]; /* grafo */
int visited[MAX]; /* visitados */
5 int depth[MAX]; /* profundidade na arvore de busca de cada
vertice */
int is_art[MAX]; /* vertice eh um ponto de articulacao?? */
int none; /* "nenhum" deve ser impresso */

int n;

10 int dfs(int v, int level)
{
    int i, low, k, is = 0;

    visited[v] = 1;
    depth[v] = level;
    low = level;

    /* low representa a menor profundidade na arvore de busca
    alcançada pelo vertice ou seus filhos por arestas de retorno
    na arvore */
20
    for(i = 0; i < n; i++)
        if(graph[v][i])
    {
        if(!visited[i])
        {
            k = dfs(i, level+1);
            if(k >= level)
                is++;
            if(low > k)
                low = k;
        }
        else
30            if(low > depth[i])
                low = depth[i];
    }
}

```

```

35     }

    is_art[v] = (level && is) || (!level && (is>1));
    /* vertice e articulacao se for raiz da busca e tiver mais do que
       um filho na arvore de busca ou se nao for a raiz e tiver um
       filho cuja menor profundidade alcanizada por arestas de retorno
       seja maior ou igual que a profundidade do vertice atual */

40    none = none && (!is_art[v]); /* encontrou um ponto de articulacao
        ou nao */

    return low;
}

45 /*
 */
int main ( void )
{
    int m, i, a, b, teste = 1, first;

    while (scanf( "%d%d" , &n, &m), n || m)
    {
        memset(graph,0,sizeof(graph));
        memset(visited,0,sizeof(visited));
        memset(is_art,0,sizeof(is_art));

        for (i=0;i<m;i++)
        {
            scanf( "%d%d" ,&a,&b);
            a--; b--;
            graph[a][b] = graph[b][a] = 1;
        }

65        none = 1;
        dfs(0,0);

        printf( "Teste %d | n" , teste++ );
        if (none)
            printf( "nenhum | n | n" );
        else
        {
            first = 1;
            for (i = 0; i < n; i++)
                if (is_art[i])
                {
                    if (first)
                    {
80                        printf( "%d" , i+1 );
                        first = 0;
                    }
                    else
                        printf( " %d" , i+1 );
                }
85            printf( " | n | n" );
        }
    }
}

```

```

90 }  
    return 0;
}

```

41 Programação Dinâmica

41.1 Spoj-BR - DOCE

```

/* Pequeno Charlie e um bom garoto viciado em doces. Ele ate assina
   a Revista Todos Doces (All Candies Magazine) e foi selecionado
   para participar na Competicao Internacional de Coleta de Doces (
   International Candy Picking Contest).

```

Nessa competicao um numero aleatorio de caixas contendo doces sao dispostas em M linhas com N colunas cada (entao, existe um total de $M \times N$ caixas). Cada caixa tem um numero indicando quantos doces ela contem.

O competidor pode pegar uma caixa (qualquer uma) e pegar todos os doces dentro dela. Mas existe uma sacada (sempre existe uma sacada): quando uma caixa e escolhida, todas as caixas das linhas logo acima e logo abaixo sao esvaziadas, assim como as caixas à direita e à esquerda da caixa escolhida. O competidor continua pegando uma caixa ate que nao hajam mais doces. */

```

int main()
{
    int n, m, back, val, i, j;
    int vLinha1, vLinha2, vCol1, vCol2;

    while((FI(n), FI(m)), m)
    {
        vLinha1 = vLinha2 = 0;
        for(i = 0; i < n; i++)
        {
            vCol1 = vCol2 = 0;
            for(j = 0; j < m; j++)
            {
                FI(val);

                back = vCol1;
                vCol1 = MAX(vCol1, vCol2 + val);
                vCol2 = back;
            }

            back = vLinha1;
            vLinha1 = MAX(vLinha1, vCol1 + vLinha2);
            vLinha2 = back;
        }

        printf("%d | n", vLinha1);
    }

    return 0;
}

```

41.2 Spoj-BR - MINHOCAS

```

/*
Um pesquisador da OBM inventou e construiu uma maquina
colhedeira de minhocas, e quer testa-la na fazenda. A maquina
tem a largura de uma celula, e em uma passada pelo terreno de
uma celula colhe todas as minhocas dessa celula, separando-as,
limpando-as e empacotando-as. Ou seja, a maquina eliminara uma
das etapas mais intensivas de mao-de-obra no processo de
produção de minhocas. A maquina, porém, ainda está em
desenvolvimento, e tem duas restrições:

-> nôo se desloca em acente, de forma que deve movimentar-se ou no
mesmo nível, ou em declive.

-> a maquina não consegue efetuar a colheita de minhocas da
especie minhocus enroscus. Todas as minhocas dessa especie
colhidas são inutilizadas e, pior, como a maquina enguiça, um
número igual de minhocas de outras especies (ja colhidas ou que
venham a ser colhidas) e tambem inutilizado.

Escreva um programa que, fornecido o mapa do campo de minhocas,
descrevendo a produtividade estimada em cada celula, calcule o
maior número esperado de minhocas que podem ser colhidas e
aproveitadas pela maquina durante o teste, conforme descrito
acima. */

int value[TABLE], up[TABLE], right[TABLE], left[TABLE];
10
/*
 *
 */
int main()
15 {
    int i, j, n, m;
    int cont = 1;

    while((fastint(&n), fastint(&m)), n || m)
20 {
        for(i = 0; i<m; i++)
            fastint(&(value[i]));

        up[0] = value[0];
25
        for(i = 1; i<m; i++)
        {
            up[i] = up[i-1] + value[i];
        }
30
        for(j = 1; j<n; j++)
        {
            for(i = 0; i<m; i++)
                fastint(&(value[i]));
35
            right[0] = up[0] + value[0];

            for(i = 1; i<m; i++)
                right[i] = right[i-1] > up[i] ? (value[i] + right[i-1]) : (
                    value[i] + up[i]);
        }
    }
}

```

```

40     left[m-1] = up[m-1] + value[m-1];
41
42     for(i = m-2; i>=0; i--)
43         left[i] = left[i+1] > up[i] ? (value[i] + left[i+1]) : (
44             value[i] + up[i]);
45
46     for(i = 0; i<m; i++)
47         up[i] = right[i] > left[i] ? right[i] : left[i];
48
49     printf( "Teste %d | n%d | n", cont++, up[m-1]);
50 }
51
52 return 0;
53 }
```

41.3 Spoj-BR - ROBUSMG

```

double dp[1500];
int cost[1500];
double apt[1500];
vector< vector<int> > solution;
5
int calculaValor(vector<int> &sol, int n)
{
    int acc = 0, i;

10   for(i = 0; i<n; i++)
    {
        acc += sol[i] * cost[i];
    }

15   return acc;
}

double calculaF0(vector<int> &sol, int n)
{
20   double acc = 1;

    for(int i = 0; i<n; i++)
    {
        acc = acc * (1. - pow(apt[i], sol[i]));
25    }

    return acc;
}

30 /**
 */
int main()
{
35   int i, j, n, m;
    vector<int> x (2000, 0);
```

```

    vector<int> uni (2000, 1);
    solution.assign(2000, x);
40
    while(scanf("%d%d", &n, &m), n || m)
    {
//        solution.assign(2000, x);

45
        for(i = 0; i<=m; i++)
            dp[i] = -1;

        for(i = 0; i<n; i++)
        {
            scanf("%d%lf", &(cost[i]), &(apt[i]));
        }

50
        if(calculaValor(uni, n) <= m)
        {
            int temp = calculaValor(uni, n);
            dp[temp] = calculaF0(uni, n);
            solution[temp] = uni;

            for(i = 0; i<=m; i++)
            {
                if(dp[i] > -0.5)
                {
                    for(j = 0; j<n; j++)
                    {
                        solution[i][j]++;
                        double result = dp[i] / (1. - pow(apt[j], solution[i][j]-1)) * (1. - pow(apt[j], solution[i][j]));
                        int ref = i + cost[j];
                        if(result > dp[ref])
                        {
70
                            dp[ref] = result;
                            solution[ref] = solution[i];
                        }
                        solution[i][j]--;
                    }
                }
            }
        }
    }

75
/* printf("solucoes:\n");
for(i = 0; i<=m; i++)
{
    printf("%d -->, i);
    for(j =0 ; j<n; j++)
        printf(" %d", solution[i][j]);
    printf("\n");
} */

80
double fo = dp[temp];
for(i = temp+1; i<=m; i++)
    fo = dp[i] > fo?dp[i]:fo;

85
/* printf("solucao:");
for(i = 0; i<n; i++)
    printf(" %d", solution[iB][i]);
printf("\n"); */

```

```

95     printf( "%.3lf\n", fo);
    }
else
{
100    printf( "0.000\n");
}
}

105   return 0;
}

```

42 Segment Tree

42.1 Spoj - LITE (with Lazy Propagation)

```

#define LEFT(x)          (x << 1)
#define RIGHT(x)         (1 + (x << 1))

5   struct interval
{
    int l, m, r, s;
    bool u;
} d[282144];

10  void init_interval(int idx, int left, int right)
{
    interval *i = d + idx;

    i->m = ((i->l = left) + (i->r = right)) >> 1;
    i->s = i->u = 0;

15  if(left >= right)
      return;

    init_interval(LEFT(idx), left, i->m);
    init_interval(RIGHT(idx), i->m + 1, right);
}

20  void refresh(int idx)
{
    if(!d[idx].u)
        return;

    interval *i = d + idx;

25  i->u = 0;
    i->s = i->r - i->l + 1 - i->s;

    if(i->l >= i->r)
        return;

30  d[LEFT(idx)].u = !d[LEFT(idx)].u;
    d[RIGHT(idx)].u = !d[RIGHT(idx)].u;
}

```

```

40  int query(int idx, int b, int e)
{
    interval *i = d + idx;

45  if(b > i->r || e < i->l)
    return 0;

    refresh(idx);

50  if(b <= i->l && e >= i->r)
    return i->s;

    return query(LEFT(idx), b, e) + query(RIGHT(idx), b, e);
}

55  void update(int idx, int b, int e)
{
    interval *i = d + idx;

60  refresh(idx);

    if(b > i->r || e < i->l)
        return;

65  if(b <= i->l && e >= i->r)
    {
        i->u = !i->u;

        refresh(idx);
    }
    else
    {
        update(LEFT(idx), b, e), update(RIGHT(idx), b, e);
        i->s = d[LEFT(idx)].s + d[RIGHT(idx)].s;
    }
}

70  void fastint(register int *n)
{
    register char c;

80  *n = 0;
    while(c = getc(stdin), c < '0' || c > '9');

85  do
    {
        (*n) = (*n)*10 + (c - '0');
    } while(c = getc(stdin), c >= '0' && c <= '9');
}

90  int main()
{
    int a, b, e, n, m;

95  fastint(&n);
    fastint(&m);
}

```

```

    init_interval(1, 1, n);

100   for(int i = 0; i < m; ++i)
{
    fastint(&a); fastint(&b); fastint(&e);

    if(!a)
        update(1, b, e);
    else
        printf("%d\n", query(1, b, e));
}
}

```

42.2 Spoj-BR - BANFARAO

```

int elem[100060];
int dp1[100060];
int dp2[100060];

5 void getMin(int limE, int limD, int &startSol, int &diffSol, int &
fSol)
{
// printf("limE == %d , limD == %d\n", limE, limD);
dp1[limE] = limE;
dp2[limE] = elem[limE];
10 diffSol = 0, fSol = dp2[limE], startSol = limE;

    for(int i = limE+1; i<=limD; i++)
{
    if(dp2[i-1] >= 0)
    {
        dp1[i] = dp1[i-1];
        dp2[i] = dp2[i-1] + elem[i];
    }
    else
20    {
        if(elem[i] > dp2[i-1])
        {
            dp1[i] = i;
            dp2[i] = elem[i];
        }
        else
25        {
            dp1[i] = dp1[i-1];
            dp2[i] = dp2[i-1] + elem[i];
        }
    }
30}
}

    if(dp2[i] > fSol)
{
    fSol = dp2[i];

//     printf("i == %d, ini == %d\n", i, dp1[i]);
    diffSol = i - dp1[i];
    startSol = dp1[i];
40}
}

```

```

    else if(dp2[i] == fSol)
    {
        if(i - dp1[i] > diffSol)
        {
            diffSol = i - dp1[i];
            startSol = dp1[i];
        }
    }
}

struct interval
{
    int l, m, r;
    int leftSum, rightSum, totalSum, bestSum;
    int nLeftSum, nRightSum, nTotalSum, nBestSum;
    int u;
} d[3*100060];

void init_interval(int idx, int left, int right)
{
    interval *i = d + idx;

    i->m = ((i->l = left) + (i->r = right)) >> 1;

    if(left > right)
    {
        i->leftSum = i->rightSum = i->totalSum = i->bestSum = -INFINITO
        ;
    }

    // printf("%d %d -> %d,%d\n", i->l, i->r, i->vMin, i->vMax);
    return;
}

if(left == right)
{
    i->leftSum = i->rightSum = i->totalSum = i->bestSum = elem[left]
    ];
    i->nLeftSum = i->nRightSum = i->nTotalSum = i->nBestSum = 1;

    // printf("(%d to %d): l == %d , r == %d , tot == %d , best == %
    d\n", i->l, i->r, i->leftSum, i->rightSum, i->totalSum, i->
    bestSum);

    // printf("%d %d -> %d,%d\n", i->l, i->r, i->vMin, i->vMax);
    return;
}

init_interval(LEFT(idx), left, i->m);
init_interval(RIGHT(idx), i->m + 1, right);

int trash1, trash2;

if(d[LEFT(idx)].leftSum > d[LEFT(idx)].totalSum + d[RIGHT(idx)].
    leftSum)
{
    i->leftSum = d[LEFT(idx)].leftSum;
    i->nLeftSum = d[LEFT(idx)].nLeftSum;
}

```

```

    }
95  else if(d[LEFT(idx)].leftSum < d[LEFT(idx)].totalSum + d[RIGHT(
    idx)].leftSum)
{
    i->leftSum = d[LEFT(idx)].totalSum + d[RIGHT(idx)].leftSum;
    i->nLeftSum = d[LEFT(idx)].nTotalSum + d[RIGHT(idx)].nLeftSum;
}
100 else
{
    i->leftSum = d[LEFT(idx)].leftSum;
    i->nLeftSum = MAX(d[LEFT(idx)].nLeftSum, d[LEFT(idx)].nTotalSum
        + d[RIGHT(idx)].nLeftSum);
}

105 if(d[RIGHT(idx)].rightSum > d[RIGHT(idx)].totalSum + d[LEFT(
    idx)].rightSum)
{
    i->rightSum = d[RIGHT(idx)].rightSum;
    i->nRightSum = d[RIGHT(idx)].nRightSum;
}
110 else if(d[RIGHT(idx)].rightSum < d[RIGHT(idx)].totalSum + d[LEFT(
    idx)].rightSum)
{
    i->rightSum = d[RIGHT(idx)].totalSum + d[LEFT(idx)].rightSum;
    i->nRightSum = d[RIGHT(idx)].nTotalSum + d[LEFT(idx)].nRightSum
        ;
}
115 else
{
    i->rightSum = d[RIGHT(idx)].rightSum;
    i->nRightSum = MAX(d[RIGHT(idx)].nRightSum, d[RIGHT(idx)].nTotalSum
        + d[LEFT(idx)].nRightSum);
}

120 i->totalSum = d[LEFT(idx)].totalSum + d[RIGHT(idx)].totalSum;
i->nTotalSum = d[LEFT(idx)].nTotalSum + d[RIGHT(idx)].nTotalSum;

125 // getMin(i->l, i->r, trash1, trash2, i->bestSum);

// i->nBestSum = trash2 + 1;

i->bestSum = MAX(d[LEFT(idx)].rightSum + d[RIGHT(idx)].leftSum,
    MAX(d[LEFT(idx)].bestSum, d[RIGHT(idx)].bestSum));
i->nBestSum = -INFINITO;

130 if(d[LEFT(idx)].bestSum == i->bestSum)
{
    i->nBestSum = MAX(i->nBestSum, d[LEFT(idx)].nBestSum);
}
135 if(d[RIGHT(idx)].bestSum == i->bestSum)
{
    i->nBestSum = MAX(i->nBestSum, d[RIGHT(idx)].nBestSum);
}
140 if(d[LEFT(idx)].rightSum + d[RIGHT(idx)].leftSum == i->bestSum)
{
    i->nBestSum = MAX(i->nBestSum, d[LEFT(idx)].nRightSum + d[RIGHT(
        idx)].nLeftSum);
}

```

```

145 // printf("(d to %d): l == %d , r == %d , tot == %d , best == %d|\n",
146 // n", i->l, i->r, i->leftSum, i->rightSum, i->totalSum, i->bestSum
147 );
148
149 // i->vMin = min(d[LEFT(i dx)].vMin, d[RIGHT(i dx)].vMin);
150 // i->vMax = max(d[LEFT(i dx)].vMax, d[RIGHT(i dx)].vMax);
151
152 // printf("%d %d -> %d,%d|n", i->l, i->r, i->vMin, i->vMax);
153 }
154
155 int foMax, nFoMax, actual, nActual;
156
157 void query(int idx, int b, int e)
158 {
159     interval *i = d + idx;
160
161     if(b > i->r || e < i->l)
162     {
163         return;
164     }
165
166     if(b <= i->l && e >= i->r)
167     {
168         // printf("/|| avalie i|n");
169
170         if(i->bestSum > foMax)
171         {
172             foMax = i->bestSum;
173             nFoMax = i->nBestSum;
174         }
175         else if(i->bestSum == foMax)
176         {
177             nFoMax = MAX(nFoMax, i->nBestSum);
178         }
179
180         int pref = actual + i->leftSum;
181         int nPref = nActual + i->nLeftSum;
182
183         if(pref > foMax)
184         {
185             foMax = pref;
186             nFoMax = nPref;
187         }
188         else if(pref == foMax)
189         {
190             nFoMax = MAX(nFoMax, nPref);
191         }
192
193         int acc = actual + i->totalSum;
194         int nAcc = nActual + i->nTotalSum;
195
196         if(acc > foMax)
197         {
198             foMax = acc;
199             nFoMax = nAcc;
200         }
201         else if(acc == foMax)
202         {
203             foMax = acc;
204             nFoMax = nAcc;
205         }
206     }
207 }

```

```

200     {
201         nFoMax = MAX(nFoMax, nAcc);
202     }
203
204     if(acc > i->rightSum)
205     {
206         actual = acc;
207         nActual = nAcc;
208     }
209     else if(acc < i->rightSum)
210     {
211         actual = i->rightSum;
212         nActual = i->nRightSum;
213     }
214     else
215     {
216         actual = acc;
217         nActual = MAX(nAcc, i->nRightSum);
218     }
219
220     if(actual < 0)
221     {
222         actual = 0;
223         nActual = 0;
224     }
225
226     // foMin = min(foMin, i->vMin);
227     // foMax = max(foMax, i->vMax);
228
229     return;
230 }
231
232     query(LEFT(idx), b, e);
233     query(RIGHT(idx), b, e);
234 }
235
236 void encontraFO(int b, int e)
237 {
238     foMax = -INFINITO;
239     nFoMax = actual = nActual = 0;
240
241     query(1, b, e);
242 }
243
244 /*
245 *
246 */
247
248 int main()
249 {
250     int cases;
251
252     FI(cases);
253
254     while(cases--)
255     {
256         int n, q;
257
258         FI(n);

```

```

260     for( int i = 1; i<=n; i++)
261         FI( elem[i] );
262
263     init_interval(1, 1, n);
264
265 //    PL;
266
267     int v1, v2;
268
269     FI(q);
270
271     while(q--)
272     {
273         FI(v1), FI(v2);
274
275         encontraFO(v1, v2);
276
277         printf("%d %d\n", foMax, nFoMax);
278     }
279
280 //    init_interval(1, 1, p);
281 //    encontraFO(i, i+j-1);
282
283     return 0;
284 }
```

42.3 Spoj-BR - HOMEM (with Lazy Propagation)

```

#define LEFT(x)          (x << 1)
#define RIGHT(x)         (1 + (x << 1))

5   struct interval
{   int l, m, r, s[3];
   int u;
} d[282144];

10  void init_interval(int idx, int left, int right)
{   interval *i = d + idx;

   i->m = ((i->l = left) + (i->r = right)) >> 1;
15   i->s[1] = i->s[2] = i->u = 0;
   i->s[0] = right-left+1;

   if(left >= right)
20     return;

   init_interval(LEFT(idx), left, i->m);
   init_interval(RIGHT(idx), i->m + 1, right);
}

25  void refresh(int idx)
{
```

```

    interval *i = d + idx;

    if(i->u == 0)
        return;

    if(i->u == 1)
    {
        int aux0 = i->s[0], aux1 = i->s[1], aux2 = i->s[2];
        i->s[0] = aux2;
        i->s[1] = aux0;
        i->s[2] = aux1;
    }
    else
    {
        int aux0 = i->s[0], aux1 = i->s[1], aux2 = i->s[2];

        i->s[0] = aux1;
        i->s[1] = aux2;
        i->s[2] = aux0;
    }

    int add = i->u;
    i->u = 0;

    if(i->l >= i->r)
        return;

    d[LEFT(idx)].u = (d[LEFT(idx)].u + add)%3;
    d[RIGHT(idx)].u = (d[RIGHT(idx)].u + add)%3;
}

int result[2];
void query(int idx, int b, int e)
{
    interval *i = d + idx;

    if(b > i->r || e < i->l)
    {
        return;
    }

    refresh(idx);

    if(b <= i->l && e >= i->r)
    {
        result[0]+=i->s[0];
        result[1]+=i->s[1];
        return;
    }

    query(LEFT(idx), b, e);
    query(RIGHT(idx), b, e);
}

void update(int idx, int b, int e)
{

```

```

85     interval *i = d + idx;
90         refresh(idx);
95         if(b > i->r || e < i->l)
100            return;
105        if(b <= i->l && e >= i->r)
110        {
115            (i->u) = ((i->u)+1)%3;
120        refresh(idx);
125    }
130    else
135    {
140        update(LEFT(idx), b, e); update(RIGHT(idx), b, e);
145        i->s[0] = d[LEFT(idx)].s[0] + d[RIGHT(idx)].s[0];
150        i->s[1] = d[LEFT(idx)].s[1] + d[RIGHT(idx)].s[1];
155        i->s[2] = d[LEFT(idx)].s[2] + d[RIGHT(idx)].s[2];
160    }
165 }

void fastint(register int *n)
{
    register char c;
170
    *n = 0;
    while(c = getc(stdin), c < '0' || c > '9');

    do
175
    {
        (*n) = (*n)*10 + (c - '0');
    } while(c = getc(stdin), c >= '0' && c <= '9');
180}

int main()
{
    int n, m, v1, v2;
    char c;

    while(scanf("%d %d\n", &n, &m) == 2)
185
    {
        init_interval(1, 1, n);

        while(m--)
190
        {
            while(c = getc(stdin), c != 'M' && c != 'C');
            fastint(&v1); fastint(&v2);

            if(c == 'M')
195
            {
                update(1, v1, v2);
            }
            else
200
            {
                result[0] = result[1] = 0;
                query(1, v1, v2);
205
            }
        }
    }
}

```

```

        printf( "%d %d %d\n", result[0], result[1], v2 - v1 + 1 -
            result[0] - result[1]);
    }
}

145   printf( "\n");
}

150   return 0;
}

```

43 Simulation

43.1 UVa - (10901) Ferry Loading III

/ Before bridges were common, ferries were used to transport cars across rivers. River ferries, unlike their larger cousins, run on a guide line and are powered by the river's current. Cars drive onto the ferry from one end, the ferry crosses the river, and the cars exit from the other end of the ferry.*

There is a ferry across the river that can take n cars across the river in t minutes and return in t minutes. A car may arrive at either river bank to be transported by the ferry to the opposite bank. The ferry travels continuously back and forth between the banks so long it is carrying a car or there is at least one car waiting at either bank. Whenever the ferry arrives at one of the banks, it unloads its cargo and loads up to n cars that are waiting to cross. If there are more than n, those that have been waiting the longest are loaded. If there are no cars waiting on either bank, the ferry waits until one arrives, loads it (if it arrives on the same bank of the ferry), and crosses the river.

*At what time does each car reach the other side of the river? */*

```

/*
 *
 */
int main()
{
    int n, t, m, v1, v2, i, j, k;
    char* read = (char*)malloc((MAXSTRSZ+1)*sizeof(char));
    vector< int > cars;
    vi leaving;
    vi carsLeft, carsRight;
    vi addLeft, addRight;
    int time, lado, total, curr0, curr1;

    int cases;

    FI(cases);

    for(k = 1; k<=cases; k++)
    {
        cars.clear(), carsLeft.clear(), carsRight.clear();
        time = lado = total = 0;

```

```

FI(n), FI(t), FI(m);

30   for(i = 0 ; i<m; i++)
{
    scanf( "%d%s ", &v1, read);

    v2 = strcmp(read, "left") ? 1 : 0;

35   cars.push_back(ii(v1,v2));

    if(v2)
        carsRight.push_back(i);
    else
40       carsLeft.push_back(i);
}

leaving.resize(m);

45 curr0 = curr1 = 0;

if(k != 1)
    PL;

50 addLeft.clear(), addRight.clear();
while(curr0 < carsLeft.size() || curr1 < carsRight.size() ||
      addLeft.size() || addRight.size())
{
    if(!lado)
    {
55      for(i = 0; i<addRight.size(); i++)
      {
        leaving[addRight[i]] = time;
      }

60      total = 0; addRight.clear();

      if((curr0 >= carsLeft.size() && curr1 < carsRight.size())
          || (curr0 < carsLeft.size() && time < cars[carsLeft[
            curr0]].first && curr1 < carsRight.size() && cars[
            carsRight[curr1]].first < cars[carsLeft[curr0]].first))
    {
65        time = MAX(time, cars[carsRight[curr1]].first) + t;
        lado ^= 1;
    }
    else
    {
        if(curr0 < carsLeft.size())
70        {
            time = MAX(time, cars[carsLeft[curr0]].first);

            addLeft.clear();
            while(curr0 < carsLeft.size() && cars[carsLeft[curr0]].first
                  <= time && total < n)
            {
                addLeft.push_back(carsLeft[curr0]);
                total++;
                curr0++;
            }
        }
    }
}

```

```

        }
    }

    ladoo ^= 1;
    time += t;
}
}
else
{
    for(i = 0; i<addLeft.size(); i++)
    {
        leaving[addLeft[i]] = time;
    }

    total = 0; addLeft.clear();

    if((curr1 >= carsRight.size() && curr0 < carsLeft.size())
       || (curr1 < carsRight.size() && time < cars[carsRight[
           curr1]].first && curr0 < carsLeft.size() && cars[
           carsLeft[curr0]].first < cars[carsRight[curr1]].first))
    {
        time = MAX(time, cars[carsLeft[curr0]].first) + t;
        ladoo ^= 1;
    }
    else
    {
        if(curr1 < carsRight.size())
        {
            time = MAX(time, cars[carsRight[curr1]].first);

            addRight.clear();
            while(curr1 < carsRight.size() && cars[carsRight[curr1
                ]].first <= time && total < n)
            {
                addRight.push_back(carsRight[curr1]);
                total++;
                curr1++;
            }
        }

        ladoo ^= 1;
        time += t;
    }
}
}

for(i = 0; i<m; i++)
    printf("%d | %n", leaving[i]);
}

return 0;
}

```

44 String

44.1 UVa - (11221) Magic square palindromes

```
/* Palindromos 2D */

int isSquare(int v)
{
    int x = (int)(round(sqrt(v)) + 0.1);

    return x * x == v;
}

int isMagic(char* str, int dim, int n)
{
    int third, zero, fourth, fin, i;

    third = 0, zero = 1;
    fourth = n-1, fin = n-2;
    for(i = 0; i<n; i++)
    {
        if(str[i] != str[n-1-i] || str[i] != str[third] || str[i] !=
           str[fourth])
            return 0;

        third += dim; if(third >= n) third = zero++;
        fourth -= dim; if(fourth < 0) fourth = fin--;
    }

    return 1;
}

/*
 */
int main()
{
    int i, tam, n, k, add, dim;

    char* read = (char*)malloc(2*(MAXSTRSZ+1)*sizeof(char));
    char* str = &(read[MAXSTRSZ+1]);

    fgets(read, MAXSTRSZ, stdin);

    sscanf(read, "%d", &n);

    for(k = 1; k<=n; k++)
    {
        fgets(read, MAXSTRSZ, stdin);
        tam = strlen(read);
        if(read[tam-1] == '\n')
            read[--tam] = '\0';

        add = 0;
        for(i = 0; read[i]; i++)
        {
            if(read[i] >= 'a' && read[i] <= 'z')
```

```

        str[add++] = read[i];
    }
    str[add] = '|0';

    printf("Case #%-d:\n", k);
    if(!isSquare(add))
    {
        printf("No magic :(\n");
    }
    else
    {
        dim = (int)(round(sqrt(add)) + 0.1);
    }
    if(isMagic(str, dim, add))
    {
        printf("%d\n", dim);
    }
    else
    {
        printf("No magic :(\n");
    }
}

return 0;
}

```

44.2 Spoj-BR - DICC11

```

const int MAXSS = 130;
vi x (MAXSS, -1);

struct Trie
{
    vector< vi > g;
    int stateCount;

    Trie()
    {
        clear();
    }

    void clear()
    {
        g.clear();
        g.push_back(x);
        stateCount = 1;
    }

    void add(char* s)
    {
        int state = 0; // root
        int next;

```

```

        next = s[i];

30      if(g[state][next] == -1)
{
    g[state][next] = stateCount;
    g.push_back(x);
    stateCount++;
}

        state = g[state][next];
}
}

40 void reverseAdd(char* s)
{
    int state = 0; // root
    int next;

45    for(int i = strlen(s)-1; i>=0; i--)
{
    next = s[i];

    if(g[state][next] == -1)
{
        g[state][next] = stateCount;
        g.push_back(x);
        stateCount++;
}

    state = g[state][next];
}
}

60 };

Trie prefixTrie, suffixTrie;
int startWith[MAXSS];

65 void suffixDFS(int state, int depth)
{
    for(int i = 0; i<MAXSS; i++)
{
    if(suffixTrie.g[state][i] != -1)
{
        if(depth >= 1)
{
            startWith[i]++;
        }

        suffixDFS(suffixTrie.g[state][i], depth + 1);
    }
}
}

75 long long prefixDFS(int state, int depth)
{
    long long acc = 0;

85    if(depth >= 1)

```

```

    acc += suffixTrie.stateCount - 1;

    for(int i = 0; i<MAXSS; i++)
    {
        if(prefixTrie.g[state][i] != -1)
        {
            if(depth >= 1)
                acc -= startWith[i];

            acc += prefixDFS(prefixTrie.g[state][i], depth + 1);
        }
    }

    return acc;
}

int main()
{
    char* read = (char*)malloc((MAXSTRSZ+1)*sizeof(char));
    int i;

    int n, m;

    while(scanf("%d%d", &n, &m), n || m)
    {
        prefixTrie.clear();
        suffixTrie.clear();
        for(i = 0; i<MAXSS; i++)
            startWith[i] = 0;

        for(i = 0; i<n; i++)
        {
            scanf("%s", read);
            prefixTrie.add(read);
        }

        for(i = 0; i<m; i++)
        {
            scanf("%s", read);
            suffixTrie.reverseAdd(read);
        }

        suffixDFS(0, 0);
        printf("%lld |n", prefixDFS(0, 0));
    }

    return 0;
}

```

44.3 Spoj-BR - PAL

/* Se uma cadeia não é palindrome, ela pode ser dividida em cadeias menores que são palindromes. Por exemplo, a cadeia 'aaxyx' pode ser dividida de quatro maneiras distintas, todas elas contendo apenas cadeias palíndromes: {'aa', 'xyx'}, {'aa', 'x', 'y', 'x'}, {'a', 'a', 'xyx'} e {'a', 'a', 'x', 'y', 'x'}.

*Escreva um programa que determine qual o menor número de partes em que uma cadeia deve ser dividida de forma que todas as partes sejam palíndromes. */*

```
5 #define MAX_V 5060

10 char pal[MAX_V][MAX_V];
int nPal[MAX_V];

15 /* 
 */
int main()
{
16     int trash, i, j, k, menor, cont = 1;
17     char* str = (char*)malloc((MAXSTRSZ+1)*sizeof(char));

20     while(fastint(&trash), trash)
21     {
22         scanf("%s", str);

23         memset(nPal, 0, trash*sizeof(int));
24         for(i = 0; i<trash; i++)
25         {
26             memset(pal[i], 0, trash*sizeof(char));
27             pal[i][i] = 1;
28         }

29         for(i = 0; str[i+1]; i++)
30         {
31             pal[i][i+1] = str[i] == str[i+1];
32         }

33         for(k = 2; str[k-1]; k++)
34         {
35             for(i = 0; i<trash-k; i++)
36             {
37                 j = i + k;
38                 if(pal[i+1][j-1] && str[i] == str[j])
39                     pal[i][j] = 1;
40             }
41         }
42     }

43     for(i = 0; str[i]; i++)
44     {
45         if(pal[0][i])
46         {
47             nPal[i] = 1;
48         }
49         else
50         {
51             menor = INFINITO;

52             for(j = 0; j<i; j++)
53             {
54                 if(pal[j+1][i] && nPal[j] < menor)
55                     menor = nPal[j];
56             }
57         }
58     }
59 }
```

```

        nPal[i] = menor + 1;
    }
}

printf( "Teste %d | n%d | n| n", cont++, nPal[trash-1]);
}

return 0;
}

```

44.4 Spoj-BR - PLAGIO

```

#define MAX_PATTERN_SIZE 10010
int F[MAX_PATTERN_SIZE];
int n, m;

5 void build_failure_function( char* pattern )
{
    int i;
    F[0] = -1;
    for(i = 0; i < m; i++)
    {
        F[i+1] = F[i] + 1;
        while( F[i+1] > 0 && pattern[i] != pattern[ F[i+1]-1 ] )
            F[i+1] = F[ F[i+1]-1 ] + 1;
    }
}

int KMP( char* text, char* pattern )
{
    build_failure_function( pattern );
    int i, j = 0;

    for(i = 0; i < n; i++)
    {
        while( 1 )
        {
            if ( text[i] == pattern[j] )
            {
                if ( ++j == m )
30                    return 1;
            }
            break;
        }

35        if ( j == 0 )
            break;
        j = F[j];
    }
}

40 return 0;
}

```

```

    char ord(char* str)
45 {
    if (!strcmp(str, "Cb")) return 12;
    if (!strcmp(str, "C'")) return 1;
    if (!strcmp(str, "C#")) return 2;
    if (!strcmp(str, "Db")) return 2;
50
    if (!strcmp(str, "D")) return 3;
    if (!strcmp(str, "D#")) return 4;
    if (!strcmp(str, "Eb")) return 4;
    if (!strcmp(str, "E")) return 5;
    if (!strcmp(str, "E#")) return 6;
55
    if (!strcmp(str, "Fb")) return 5;
    if (!strcmp(str, "F")) return 6;
    if (!strcmp(str, "F#")) return 7;
    if (!strcmp(str, "Gb")) return 7;
    if (!strcmp(str, "G")) return 8;
60
    if (!strcmp(str, "G#")) return 9;
    if (!strcmp(str, "Ab")) return 9;
    if (!strcmp(str, "A")) return 10;
    if (!strcmp(str, "A#")) return 11;
    if (!strcmp(str, "Bb")) return 11;
65
    if (!strcmp(str, "B")) return 12;
    return 1;
}

/*
 *
 */
int main()
{
    char* base = (char*)malloc(4*(MAXSTRSZ+1)*sizeof(char));
70
    char* busca = &(base[MAXSTRSZ+1]);
    char* prim = &(busca[MAXSTRSZ+1]);
    char* seg = &(prim[MAXSTRSZ+1]);

    int value, i, j;
80
    while (scanf("%d%d", &n, &m), n || m)
    {
//        printf("n == %d , m == %d | n ", n, m);
//        sleep(1);
85
        scanf("%s", prim);
        for (i = 1; i < n; i++)
        {
            scanf("%s", seg);
            value = ord(seg) - ord(prim);
            value = value < 0 ? value + 12 : value;
            base[i-1] = value;
            strcpy(prim, seg);
        }

        scanf("%s", prim);
90
        for (i = 1; i < m; i++)
        {
            scanf("%s", seg);
            value = ord(seg) - ord(prim);
            value = value < 0 ? value + 12 : value;
            busca[i-1] = value;
        }
    }
}

```

```

        strcpy(prim, seg);
    }

105    n--, m--;
    /* printf("base :");
    for(i = 0; i <n; i++)
        printf("%d", base[i]);
    printf("\n");
    printf("busca :");
    for(i = 0; i <m; i++)
        printf("%d", busca[i]);
    printf("\n\n"); */

115    /* int lim = n - m;
    int found = 0;
    for(i = 0; i <=lim ; i++)
    {
        int allEq = 1;
        for(j = 0; j <m; j++)
        {
            if(base[i+j] != busca[j])
            {
                allEq = 0;
                break;
            }
        }
        if(allEq)
        {
            found = 1;
            break;
        }
    } */
    // puts(found ? "S": "N");

    puts(KMP(base, busca) ? "S": "N");
}
return 0;
}

```

45 Union Set

45.1 Spoj-BR - FUSOES1

```

int father[MAXN];
int rank[MAXN];

5 int fFind(int x)
{
    while(x != father[x])
    {
        x = father[x];
    }
}

```

```

    }

10   return x;
}

void join(int* x, int* y)
{
    int rx = fFind(*x), ry = fFind(*y);

    if(rx == ry)
        return;

20   if(rank[rx] > rank[ry])
        father[ry] = rx;
    else
    {
        father[rx] = ry;
        if(rank[rx] == rank[ry])
            rank[ry]++;
    }
}

30 /*
 *
 */
int main()
{
    int n, k, res, v1, v2, i, j;
    char c;

    scanf( "%d%d\n", &n, &k);

40   for(i = 1; i<=n; i++)
    {
        father[i] = i;
        rank[i] = 0;
    }

45   for(i = 0; i<k; i++)
    {
//      printf("fathers:\n");
//      for(j = 1; j<=n; j++)
//          printf(" %d -> %d\n", j, father[j]);
    }

50   scanf( "%c%d%d\n", &c, &v1, &v2);

    // printf("c == %c , v1 == %d , v2 == %d\n", c, v1, v2);

55   if(c == 'C')
    {
        printf( "%s ", fFind(v1) == fFind(v2)? "S\n": "N\n");
    }
    else
    {
        join(&v1, &v2);
    }
}

```

```

    return 0;
}

```

Parte IV

Most Recent 2.0

45.2 all differents

```

/*
 * File : main.cpp
 * Author: ubuntulap
 *
5  * Created on February 28, 2008, 6:16 PM
 */

10 #include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <limits.h>
#include <assert.h>
#include <cctype>
15 #include <iomanip>
#include <iostream>
#include <string>
#include <algorithm>
#include <utility>
20 #include <iterator>
#include <numeric>
#include <sstream>
#include <fstream>
#include <bitset>
25 #include <map>
#include <vector>
#include <set>
#include <stack>
#include <queue>
30 #include <deque>
#include <list>

#define SWAP(a,b) ( (a)=(b), (b)=(a), (a)=(b) )
#define MIN(a,b) ( (a)<(b) ? (a) : (b) )
35 #define MAX(a,b) ( (a)>(b) ? (a) : (b) )
#define ABS(a) ( (a)<(0) ? (-1)*(a) : (a) )
#define ROUND(a) ( (a)>=(0) ? ((int)((double)(a) + 0.5)) : ((int)((
    double)(a) - 0.5)) )
#define READ cin.getline(read, MAXSTRSZ)
#define para(x) for(typeof(x.begin()) it = x.begin(); it!=x.end(); 
    it++)
40 #define PI acos(-1.0)

#define FI(a) fastint(&a)
#define PL printf("|\n")

```

```

45
#define MAXSTRSZ      100000
#define MAXNUMBER     200000
#define INFINITO      999999999
#define EPS           1e-9

50
using namespace std;

typedef long long ll;
typedef unsigned long long ull;
55
typedef pair<int,int> ii;
typedef vector< pair<int,int> > vii;
typedef vector<int> vi;

void fastint(register int *n)
60
{
    register char c;
    register int neg = 0;
    *n = 0;

65
    while(c = getc(stdin), c < '0' || c > '9')
        neg |= c == '_';

    do
    {
50
        (*n) = (*n)*10 + (c - '0');
    } while(c = getc(stdin), c >= '0' && c <= '9');

        (*n) = neg?(*n)*(-1):(*n);
    }

75
int quant[6000];
bool itis[6000];

/*
 *
 */
int main()
{
    char* str = (char*)malloc((MAXSTRSZ+1)*sizeof(char));
    int total = 0;
    int i, j, l, r;

    for(i = 1; i<=5100; i++)
    {
        itis[i] = false;
        sprintf(str, "%d", i);

        set<char> elem;

        for(j = 0; str[j]; j++)
        {
            elem.insert(str[j]);
        }

        if(elem.size() == strlen(str))
        {
            total++;
        }
    }
}

```

```

        itis[i] = true;
    }

105    quant[i] = total;
}

110    while(scanf("%d%d", &l, &r) == 2)
{
    int ans = quant[r] - quant[l];
    if(itis[l])
        ans++;
    printf("%d\n", ans);
}

115    return 0;
}

```

45.3 Almost

```

/* Quase menor caminho */

#include <iostream>
#include <vector>
5 #include <queue>
#include <stdio.h>
#include <stdlib.h>

using namespace std;

10 #define INF 99999
#define TNo pair<int, int>

15 class mycompare{
public:
    bool operator() (const TNo& a, const TNo& b) const{
        return (a.second>b.second);
    }
};

20 int menor;
vector< vector<bool> > pode;

int dijkstra(const vector< vector<TNo> > &grafo, int inicial, int
final){
25 priority_queue<TNo, vector<TNo>, mycompare> myheap;
for(int i=0; i<grafo.size(); i++) if(i != inicial)
    myheap.push(make_pair(i, INF));

myheap.push(make_pair(inicial, 0));
30 vector<int> distancia(grafo.size(), INF);
distancia[inicial]=0;

while (!myheap.empty()){
    int n = myheap.top().first;
    myheap.pop();

```

```

        for (int i=0; i<grafo[n].size(); i++) {
            if(pode[n][grafo[n][i].first])
                if (distancia[grafo[n][i].first] > distancia[n] + grafo[n][i].
                    second){
                    distancia[grafo[n][i].first] = distancia[n] + grafo[n][i].
                        second;
                    myheap.push(make_pair(grafo[n][i].first, distancia[n] +
                        grafo[n][i].second));
                }
            }
        }

        return distancia[final];
    }

50 vector<bool> visitado;
vector<int> pilha;
void backtracking(const vector< vector<TNo> > &grafo, int peso, int
    atual, int destino){
    if(peso == menor && atual == destino){
        for(int i=0; i<pilha.size()-1; i++)
            pode[pilha[i]][pilha[i+1]] = false;

        pode[pilha.back()][destino] = false;
        return;
    }
    if(peso > menor) return;

    pilha.push_back(atual);
    visitado[atual] = true;

65    for(int i=0; i<grafo[atual].size(); i++) if(!visitado[grafo[atual]
        ][i].first])
        backtracking(grafo, peso+grafo[atual][i].second, grafo[atual][i].
            first, destino);

    pilha.pop_back();
    visitado[atual] = false;
}

70    return;
}

int main(){
    int m, n;
    int org;
    TNo dest;
    int origem, destino;

75    while(scanf("%d %d", &n, &m) && (n+m)){
        scanf("%d %d", &origem, &destino);

        visitado.clear();
        pode.clear();
        visitado.resize(n);
        pode.resize(n);

85        vector< vector<TNo> > grafo(n);

```

```

90     for( int i=0; i<n; i++){
91         visitado[i] = false;
92         for( int j=0; j<n; j++)
93             podes[i].push_back(true); // [j] = true;
94     }
95
96
97     for( int i=0; i<m; i++){
98         scanf( "%d %d %d", &org, &dest.first, &dest.second);
99
100        grafo[org].push_back(dest);
101    }
102
103    menor = dijkstra(grafo, origem, destino);
104    if(menor != INF){
105        backtracking(grafo, 0, origem, destino);
106        pilha.clear();
107        menor = dijkstra(grafo, origem, destino);
108    }
109
110    printf( "%d\n", menor==INF?-1:menor);
111
112}
113
114 return 0;
115 }
```

45.4 Tree DP

```

/* Arvores coloridas (Seletiva da UFMG) - PD em Árvore */

#include <stdio.h>
#include <stdlib.h>
5 #include <vector>
#include <iostream>
#include <string>

using namespace std;

10 int contador;
long long memoize[2][262150];
long long dfs(vector< vector<int> > &grafo, int pos, bool azul_ant,
    int anterior){
    long long a, b;

15    if(azul_ant){
        if(memoize[1][pos] != -1) return memoize[1][pos];

        memoize[1][pos] = 1;
        for( int i=0; i<grafo[pos].size(); i++) if(grafo[pos][i] !=
20            anterior)
            memoize[1][pos] = (memoize[1][pos]*dfs(grafo, grafo[pos][i],
                false, pos))%1000000007;

        return memoize[1][pos];
    }
}
```

```

        }

25    if(memoize[0][pos] != -1) return memoize[0][pos];

    a = 1;
    for(int i=0; i<grafo[pos].size(); i++) if(grafo[pos][i] !=
        anterior)
30    a = (a*dfs(grafo, grafo[pos][i], false, pos))%1000000007;

    b = 1;
    for(int i=0; i<grafo[pos].size(); i++) if(grafo[pos][i] !=
        anterior)
        b = (b*dfs(grafo, grafo[pos][i], true, pos))%1000000007;

35    memoize[0][pos] = (a+b)%1000000007;
    return memoize[0][pos];
}

40 int main(){
    int n, a, b;

    while(scanf("%d", &n) && n){
        vector<vector<int>> grafo(n);

45    for(int i=0; i<n-1; i++){
        scanf("%d %d", &a, &b);
        grafo[b].push_back(a);
        grafo[a].push_back(b);
    }

50    for(int i=0; i<n; i++){
        memoize[0][i] = -1;
        memoize[1][i] = -1;
    }
    long long conta = dfs(grafo, 0, false, -1);

55    printf("%lld | n", conta%1000000007);
}
    return 0;
}

```

45.5 Campus – Articulation Points

```

#include <stdio.h>
#include <iostream>
#include <vector>
#include <set>

5 #define MAX 90010

using namespace std;

10 vector<vector<int>> grafo;
set<int> art;
int visitado[MAX];
int N, tam;

```

```

15   int ini;
16
17   int dfs(int u)
18   {
19       int filhos = 0;
20       visitado[u] = tam++;
21       int menor = visitado[u];
22
23       for(int i = 0; i < (int)grafo[u].size(); i++)
24       {
25           if(visitado[grafo[u][i]] == 0)
26           {
27               filhos++;
28               int m = dfs(grafo[u][i]);
29               menor = min(menor, m);
30
31               if(visitado[u] <= m && (u != ini || filhos >= 2))
32                   art.insert(u);
33           }
34           else menor = min(menor, visitado[grafo[u][i]]);
35       }
36
37       return menor;
38   }
39
40   int main()
41   {
42       int n, m;
43
44       while(scanf("%d %d", &n, &m) != EOF)
45       {
46           grafo.clear();
47           grafo.resize(n*m);
48
49           char entrada[n][m];
50           getchar();
51
52           N = n*m;
53           int cont = 0;
54           for(int i = 0; i < n; i++)
55           {
56               for(int j = 0; j < m; j++)
57               {
58                   scanf("%c", &entrada[i][j]);
59                   if(entrada[i][j] == '#')
60                   {
61                       cont++;
62                       ini = i * m + j;
63                   }
64               }
65               getchar();
66           }
67
68           if(cont <= 2)
69           {
70               printf("-1\n");
71               continue;
72           }
73       }
74   }

```

```

    for(int i = 0; i < n; i++)
    {
        for(int j = 0; j < m; j++)
        {
            if(entrada[i][j] == '#')
            {
                if(i > 0 && entrada[i-1][j] == '#')
                    grafo[i*m+j].push_back((i-1)*m + j);

                if(i < n-1 && entrada[i+1][j] == '#')
                    grafo[i*m+j].push_back((i+1)*m + j);
            }

            if(j > 0 && entrada[i][j-1] == '#')
                grafo[i*m+j].push_back(i * m + j - 1);

            if(j < m-1 && entrada[i][j+1] == '#')
                grafo[i*m+j].push_back(i * m + j + 1);
        }
    }

    for(int i = 0; i < N; i++)
        visitado[i] = 0;

    tam = 1;
    dfs(ini);

    //printf("tam: %d\n", (int)art.size());
    if((int)art.size())
        printf("1\n");
    else printf("2\n");

    art.clear();
}

return 0;
}

```

45.6 Cads – MiniMax

```

#include <stdio.h>
#include <vector>

using namespace std;

5      typedef struct a{
        bool already;
        long i, f;
    }TMem;

10     TMem memoize[10010][10010];
        vector<int> v;

long maior(int ini, int fim, bool alberto){
15     if(ini == fim) return 0;

```

```

1      if(1){
2          memoize[ini][fim].i = maior(ini+1, fim, !alberto);
3          memoize[ini][fim].f = maior(ini, fim-1, !alberto);
4          memoize[ini][fim].already = true;
5      }
6
7      if(alberto){
8          if((v[ini]+memoize[ini][fim].i) > (v[fim]+memoize[ini][fim].f))
9              return (v[ini]+memoize[ini][fim].i);
10         else return (v[fim]+memoize[ini][fim].f);
11     }
12     else{
13         if(memoize[ini][fim].i < memoize[ini][fim].f) return memoize[
14             ini][fim].i;
15         else return memoize[ini][fim].f;
16     }
17 }
18
19 int main(){
20     int n;
21
22     while(scanf( "%d", &n) != EOF){
23         v.resize(n);
24         for(int i=0; i<n; i++)
25             scanf( "%d", &v[i]);
26
27         for(int i=0; i<n; i++)
28             for(int j=i+1; j<n; j++)
29                 memoize[i][j].already = false;
30
31         printf( "%ld |n", maior(0, n-1, true));
32     }
33
34     return 0;
35 }
```

45.7 Clock Angle

```

/*
 * File:    main.cpp
 * Author:  ubuntulap
 *
5  * Created on February 28, 2008, 6:16 PM
 */

10 #include <stdlib.h>
11 #include <stdio.h>
12 #include <string.h>
13 #include <math.h>
14 #include <limits.h>
15 #include <assert.h>
16 #include <cctype>
17 #include <iomanip>
18 #include <iostream>
19 #include <string>
```

```

#include <algorithm>
#include <utility>
20 #include <iterator>
#include <numeric>
#include <iostream>
#include <fstream>
#include <bitset>
25 #include <map>
#include <vector>
#include <set>
#include <stack>
#include <queue>
30 #include <deque>
#include <list>

#define SWAP(a,b) ( (a)=(b), (b)=(a), (a)=(b) )
#define MIN(a,b) ( (a)<(b) ? (a) : (b) )
35 #define MAX(a,b) ( (a)>(b) ? (a) : (b) )
#define ABS(a) ( (a)<(0) ? (-1)*(a) : (a) )
#define ROUND(a) ( (a)>=(0) ? ((int)((double)(a) + 0.5)) : ((int)((
    double)(a) - 0.5)) )
#define READ cin.getline(read, MAXSTRSZ)
#define para(x) for(typeof(x.begin()) it = x.begin(); it!=x.end(); 
    it++)
40 #define PI acos(-1.0)

#define FI(a) fastint(&a)
#define PL printf(" | %n")
45
#define MAXSTRSZ      100000
#define MAXNUMBER     200000
#define INFINITO      999999999
#define EPS            1e-9

50 using namespace std;

typedef long long ll;
typedef unsigned long long ull;
55 typedef pair<int,int> ii;
typedef vector< pair<int,int> > vii;
typedef vector<int> vi;

void fastint(register int *n)
60 {
    register char c;
    register int neg = 0;
    *n = 0;

    while(c = getc(stdin), c < '0' || c > '9')
        neg |= c == '-';
65
    do
    {
        (*n) = (*n)*10 + (c - '0');
    } while(c = getc(stdin), c >= '0' && c <= '9');

    (*n) = neg?(*n)*(-1):(*n);

```

```

    }

75
/*
 *
 */
int main()
{
    int n;

    bool is[181];
    for(int i=0; i<=180; i++)
        if(i%6 == 0) is[i] = true;
        else is[i] = false;

    while(scanf("%d", &n) != EOF) printf("%c | %n", is[n]? 'Y': 'N');

90    return 0;
}

```

45.8 Couples – TLE – Gabriel

```

#include <stdio.h>
#include <iostream>
#include <set>
#include <vector>
5 #include <queue>
#include <map>

using namespace std;

10 int n;
#define NO fila.front()

vector<bool> mapa;
15 void bfs(vector< vector<int> > &grafo, vector< vector<int> > &mem,
           int ini)
{
    queue<int> fila;

    mapa[ini] = true;
20    fila.push(ini);
    int prof = -1;
    vector< pair<int, int> > toAtt;
    toAtt.push_back(make_pair(ini, 0));

    while(!fila.empty()){

25        int tam = fila.size();

        prof++;
        for(int j=0; j<tam; j++){

30            for(int k=0; k<toAtt.size(); k++)
                if(toAtt[k].first == ini)
                    mem[ toAtt[k].first ][NO] = mem[ NO ][ toAtt[k].first ] =
                        prof;

```

```

    else if(toAtt[k].second != prof)
        mem[ toAtt[k].first ][NO] = mem[ NO ][ toAtt[k].first ] =
            (prof-mem[ini][ toAtt[k].first ]);
    else
        mem[ toAtt[k].first ][NO] = mem[ NO ][ toAtt[k].first ] =
            prof*2;

40   for(int i=0; i<grafo[NO].size(); i++) if( mapa[grafo[NO][i] ] ==
        == false){
        fila.push(grafo[NO][i]);
        mapa[ grafo[NO][i] ] = true;
        toAtt.push_back( make_pair(grafo[NO][i], prof) );
    }
45   fila.pop();
}
}
}

50 int main()
{
    int n;
    while(scanf( "%d ", &n) != EOF)
    {
        vector< vector<int> > grafo(n);
        vector< vector<int> > mem(n, vector<int>(n, -1));
        mapa.clear();
        mapa.assign(n, false);

60    for(int i = 0; i < n; i++)
    {
        int v1;
        scanf( "%d ", &v1);
65        grafo[i].push_back(v1-1);
        grafo[v1-1].push_back(i);
    }

67    int q;
70    scanf( "%d ", &q);

        for(int i=0; i<n; i++)
            if(mapa[i] == false){
                bfs(grafo, mem, i);
            }

75        for(int i = 0; i < q; i++)
    {
        int a, b;

80        scanf( "%d %d ", &a, &b);
        printf("%d | n",mem[a-1][b-1]);
    }
85    }
    return 0;
}

```

45.9 Couples – TLE – Leandro

```
/*
 * File: main.cpp
 * Author: ubuntulap
 *
5 * Created on February 28, 2008, 6:16 PM
 */

10 #include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <limits.h>
#include <assert.h>
#include <cctype>
15 #include <iomanip>
#include <iostream>
#include <string>
#include <algorithm>
#include <utility>
20 #include <iterator>
#include <numeric>
#include <sstream>
#include <fstream>
#include <bitset>
25 #include <map>
#include <vector>
#include <set>
#include <stack>
#include <queue>
30 #include <deque>
#include <list>

#define SWAP(a,b) ( (a)=(b), (b)=(a), (a)=(b) )
#define MIN(a,b) ( (a)<(b) ? (a) : (b) )
35 #define MAX(a,b) ( (a)>(b) ? (a) : (b) )
#define ABS(a) ( (a)<(0) ? (-1)*(a) : (a) )
#define ROUND(a) ( (a)>=(0) ? ((int)((double)(a) + 0.5)) : ((int)((
    double)(a) - 0.5)) )
#define READ cin.getline(read, MAXSTRSZ)
#define para(x) for(typeof(x.begin()) it = x.begin(); it!=x.end(); it++)
40
#define PI acos(-1.0)

#define FI(a) fastint(&a)
#define PL printf(" | n")
45
#define MAXSTRSZ 100000
#define MAXNUMBER 200000
#define INFINITO 999999999
#define EPS 1e-9
50
using namespace std;

typedef long long ll;
typedef unsigned long long ull;
```

```

55 | typedef pair<int,int> ii;
56 | typedef vector< pair<int,int> > vii;
57 | typedef vector<int> vi;

58 | void fastint(register int *n)
59 |
60 | {
61 |     register char c;
62 |     register int neg = 0;
63 |     *n = 0;
64 |

65 |     while(c = getc(stdin), c < '0' || c > '9')
66 |         neg |= c == '_';
67 |

68 |     do
69 |     {
70 |         (*n) = (*n)*10 + (c - '0');
71 |     } while(c = getc(stdin), c >= '0' && c <= '9');

72 |     (*n) = neg?(*n)*(-1):(*n);
73 | }
74 |

75 | int mapa[100060];
76 | int rank[100060];

77 | int next[100060];
78 | int t1[100060];
79 | int t2[100060];
80 | bool chega1[100060];
81 | bool chega2[100060];

82 | int numComp;
83 | int id[100060];

84 | int find(int a)
85 |
86 | {
87 |     while(a != mapa[a]) a = mapa[a];
88 |

89 |     return a;
90 | }

91 | void uniao(int x, int y)
92 |
93 | {
94 |     int xRoot = find(x);
95 |     int yRoot = find(y);
96 |     if (rank[xRoot] > rank[yRoot])
97 |         mapa[yRoot] = xRoot;
98 |     else if (rank[xRoot] < rank[yRoot])
99 |         mapa[xRoot] = yRoot;
100 |     else if (xRoot != yRoot)
101 |         mapa[yRoot] = xRoot, rank[xRoot] = rank[xRoot] + 1;
102 | }

103 | /*
104 |  *
105 | */
106 | int main(int argc, char** argv)
107 |
108 | {
109 |     int i, j;

```

```

int n, m;

115 int cont = 0;
while(scanf("%d", &n) == 1)
{
    for(i = 1; i<=n; i++)
    {
        scanf("%d", &next[i]);
        mapa[i] = i;
        rank[i] = 0;
    }

125    for(i = 1; i<=n; i++)
    {
        uniao(i, next[i]);
    }

130    numComp = 0;

        for(i=1; i<=n; i++)
    {
        id[i] = -1;
    }

135    scanf("%d", &m);

        for(i = 0; i<m; i++)
    {
        int from, to;

        scanf("%d%d", &from, &to);

140        if(from == to)
    {
        printf("0\n");
        continue;
    }

145        if(find(from) != find(to))
    {
        printf("-1\n");
        continue;
    }

150    }

155    for(int j = 1; j<=n; j++)
        chega1[j] = chega2[j] = false;

        int inst = 0;

160        chega1[from] = chega2[to] = true;
        bool change1 = true, change2 = true;
        int curr1 = from, curr2 = to;
        t1[curr1] = t2[curr2] = 0;

165        int ans = 10000000;
        while(1)
    {

```

```

170 //      printf("cidades : %d e %d\nproximas %d e %d\n", curr1 ,
175   curr2 , next[curr1] , next[curr2]);
176
177 /* printf("inst%d:\n", inst);
178 para(cities1)
179   cout << " " << *it;
180 cout << endl;
181 para(cities2)
182   cout << " " << *it;
183 cout << endl << endl; */
184
185 /* para(cities2)
186 {
187   if(cities1.find(*it) != cities1.end())
188   {
189     met = true;
190     ans = MAX(t1[*it] + t2[*it], MIN(t1[*it] + inst, inst +
191       t2[*it]));
192     break;
193   }
194   if(met)
195     break; */
196
197 if(change1 == false && change2 == false)
198   break;
199
200 if(inst > ans)
201   break;
202
203 if(change1)
204 {
205   if(chega1[next[curr1]])
206   {
207     change1 = false;
208   }
209   else
210   {
211     chega1[next[curr1]] = true;
212     t1[next[curr1]] = inst + 1;
213
214     if(chega2[next[curr1]])
215     {
216       ans = MIN(ans, t1[next[curr1]] + t2[next[curr1]]);
217       //break;
218     }
219   }
220 }
221
222 if(change2)
223 {
224   if(chega2[next[curr2]])
225   {
226     change2 = false;
227   }
228   else
229   {
230     chega2[next[curr2]] = true;
231   }
232 }

```

```

        t2[next[curr2]] = inst + 1;

        if(chega1[next[curr2]])
        {
            ans = MIN(ans, t1[next[curr2]] + t2[next[curr2]]);
            //break;
        }
    }

235    curr1 = next[curr1];
    curr2 = next[curr2];
    inst++;
}

240    printf("%d\n", ans != 10000000 ? ans : -1);
}
}

245    return 0;
}

```

45.10 Prefix Tree

```

/*
 * File: main.cpp
 * Author: ubuntulap
 *
5 * Created on February 28, 2008, 6:16 PM
 */

#include <stdlib.h>
#include <stdio.h>
10 #include <string.h>
#include <math.h>
#include <limits.h>
#include <assert.h>
#include <cctype>
15 #include <iomanip>
#include <iostream>
#include <string>
#include <algorithm>
#include <utility>
20 #include <iterator>
#include <numeric>
#include <sstream>
#include <fstream>
#include <bitset>
25 #include <map>
#include <vector>
#include <set>
#include <stack>
#include <queue>
30 #include <deque>
#include <list>

```

```

#define SWAP(a,b) ( (a)=(b), (b)=(a), (a)=(b) )
#define MIN(a,b) ( (a)<(b) ? (a) : (b) )
#define MAX(a,b) ( (a)>(b) ? (a) : (b) )
#define ABS(a) ( (a)<(0) ? (-1)*(a) : (a) )
#define ROUND(a) ( (a)>=(0) ? ((int)((double)(a) + 0.5)) : ((int)((
    double)(a) - 0.5)) )
#define READ cin.getline(read, MAXSTRSZ)
#define para(x) for(typeof(x.begin()) it = x.begin(); it!=x.end(); 
    it++)
#define PI acos(-1.0)

#define FI(a) fastint(&a)
#define PL printf(" | %n")
#define MAXSTRSZ      100000
#define MAXNUMBER     200000
#define INFINITO      999999999
#define EPS           1e-9
#define ATUAL          ((*p).filhos)

using namespace std;

typedef long long ll;
typedef unsigned long long ull;
typedef pair<int,int> ii;
typedef vector< pair<int,int> > vii;
typedef vector<int> vi;

struct no{
    bool endWord;
    map<char, struct no> filhos;
};

void fastint(register int *n)
{
    register char c;
    register int neg = 0;
    *n = 0;
    while(c = getc(stdin), c < '0' || c > '9')
        neg |= c == '_';

    do
    {
        (*n) = (*n)*10 + (c - '0');
    } while(c = getc(stdin), c >= '0' && c <= '9');

    (*n) = neg?(*n)*(-1):(*n);
}

/*
 */
double contar(struct no &arvore, string &palavra){
    struct no *p = &(arvore.filhos[palavra[0]]);
    double contador = 1;

```

```

90   for(int i=1; i<palavra.size(); i++){
91     if(ATUAL.size() != 1) contador += 1.0;
92     else if ((*p).endWord) contador += 1.0;
93     p = &ATUAL[palavra[i]];
94   }
95
96   //cout << "Palavra: " << palavra << " and " << "contador" <<
97   //contador << endl;
98   return contador;
99 }
100
101 void insere(struct no &arvore, string &palavra){
102   struct no *p = &arvore;
103   struct no aux;
104   aux.endWord = false;
105
106   for(int i=0; i<palavra.size(); i++){
107     if(ATUAL.find(palavra[i]) == ATUAL.end()) ATUAL.insert(
108       make_pair(palavra[i], aux));
109
110     p = &ATUAL[palavra[i]];
111     if(i == palavra.size()-1) (*p).endWord = true;
112   }
113 }
114
115 int main()
116 {
117   int n;
118   double contador;
119   struct no *p;
120   struct no arvore;
121
122   while(scanf("%d", &n) != EOF){
123     getchar();
124     arvore.filhos.clear();
125     vector<string> v(n);
126
127     for(int i=0; i<n; i++){
128       cin >> v[i];
129
130       insere(arvore, v[i]);
131     }
132     contador = 0.0;
133     for(int i=0; i<n; i++)
134       contador += contar(arvore, v[i]);
135
136     printf("%.2lf\n", contador/n);
137   }
138
139   return 0;
140 }
```

45.11 Interval Product – Gabriel – 1

```

#include <stdio.h>
#include <stdlib.h>
```

```

#include <vector>

5  using namespace std;

int num[100005], zeros[100005];

int main(){
    int n, m, entrada;
    char l;
    int v, v2;

    num[0] = zeros[0] = 0;
15   while(scanf("%d %d", &n, &m) != EOF){
        for(int i=1; i<=n; i++){
            scanf("%d", &entrada);
            zeros[i] = zeros[i-1];
            num[i] = num[i-1];

20        if(entrada == 0) zeros[i]++;
        else if(entrada < 0) num[i-1]++;
    }

25    for(int i=0; i<m; i++){
        getchar();
        scanf("%c %d %d", &l, &v, &v2);

        if(l == 'C'){
            if(v2 == 0){
                if(zeros[v] == zeros[v-1] && num[i] > num[i-1]){
                    for(int j=v; j<=n; j++){
                        zeros[j]++;
                        num[j]--;
                    }
                }
            }
            else if(zeros[v] == zeros[v-1]){
                for(int j=v; j<=n; j++)
                    zeros[j]++;
            }
        }
        else if(v2 > 0 && num[v] > num[v-1]){
            for(int j=v; v<=n; j++)
                num[j]--;
        }
        else if(v2 < 0 && num[v] == num[v-1]){
            for(int j=v; v<=n; j++)
                num[j]++;
        }
    }

55    else if(l == 'P'){
        if(zeros[v2]-zeros[v] > 0) printf("0");
        else if((num[v2]-num[v])%2 == 1) printf("-");
        else if((num[v2]-num[v])%2 == 0) printf("+");
    }
}
}

```

```

    return 0;
}

```

45.12 Interval Product – Gabriel – 2

```

/* Intervalos (Melhor soluÃ§Ã£o seria com Segtree), implementaÃ§Ã£o
   ingÃ¡nua */

#include <stdio.h>
#include <stdlib.h>
5 #include <vector>
#include <iostream>

using namespace std;

10 #define FI(a) fastint(&a)

void fastint(register int *n)
{
    register char c;
    register int neg = 0;
    *n = 0;

    while(c = getc(stdin), c < '0' || c > '9')
        neg |= c == '-';
20
    do
    {
        (*n) = (*n)*10 + (c - '0');
        } while(c = getc(stdin), c >= '0' && c <= '9');

    25 (*n) = neg?(*n)*(-1):(*n);
}

void fastchar(register char *n)
30
{
    while(*n = getc(stdin), *n != 'C' && *n != 'P');
}

int num[100005], zeros[100005];
35
int main(){
    int n, m, entrada;
    char l;
    int v, v2;

    40 num[0] = zeros[0] = 0;
    while(scanf("%d %d", &n, &m) != EOF){
        for(int i=1; i<=n; i++){
            FI(entrada);
            zeros[i] = zeros[i-1];
            num[i] = num[i-1];

            if(entrada == 0) zeros[i]++;
            else if(entrada < 0) num[i]++;
45        }
    }
50
}

```

```

for( int i=0; i<m; i++) {
    fastchar(&l), FI(v), FI(v2);

    if(l == 'C'){
        if(v2 == 0){
            /* De - para 0 */
            if(num[v] > num[v-1]){
                for( int j=v; j<=n; j++){
                    zeros[j]++;
                    num[j]--;
                }
            }
        }

        /* De + para 0 */
        else if(zeros[v] == zeros[v-1]){
            for( int j=v; j<=n; j++)
                zeros[j]++;
        }
    }
    else if(v2 > 0){
        /* De - para + */
        if(num[v] > num[v-1]){
            for( int j=v; j<=n; j++)
                num[j]--;
        }

        /* De 0 para + */
        else if(zeros[v] > zeros[v-1]){
            for( int j=v; j<=n; j++)
                zeros[j]--;
        }
    }

    else if(v2 < 0){
        /* De + para - */
        if(zeros[v] == zeros[v-1] && num[v] == num[v-1]){
            for( int j=v; j<=n; j++)
                num[j]++;
        }

        /* De 0 para - */
        else if(zeros[v] > zeros[v-1]){
            for( int j=v; j<=n; j++){
                num[j]++;
                zeros[j]--;
            }
        }
    }
}

else{
    if(zeros[v2]-zeros[v-1] > 0) printf("0");
    else if((num[v2]-num[v-1])%2 == 1) printf("-");
    else printf("+");
}
}

```

```

110     printf( "#\n");
    }
    return 0;
}

```

45.13 Interval Product – Fenwick Tree / Binary Indexed Tree

```

//Interval Product - UVA 12532

#include <iostream>
#include <stdio.h>
5 #include <string.h>

using namespace std;

const int MAXN = 100001;
10 int zeros[MAXN], negativos[MAXN] , vetor[MAXN];

int read(int* tree, int idx)
{
    int sum = 0;
15
    while(idx > 0)
    {
        printf("idx == %d\n", idx);
        sum += tree[idx];
        idx -= (idx & -idx);
20    }
    printf("\n");

    return sum;
25}

void update(int *tree, int idx, int val)
{
    while(idx < MAXN)
    {
30        tree[idx] += val;
        idx += (idx & -idx);
    }
35}
int main()
{
    int n, k;
40
    vetor[0] = 1;

    while(scanf("%d %d", &n, &k) == 2)
    {
45        memset(negativos, 0, sizeof(negativos));
        memset(zeros, 0, sizeof(zeros));

        for(int i = 1; i <= n; i++)
        {
            scanf("%d", &vetor[i]);
        }
    }
}

```

```

50     if(vetor[i] == 0)
      update(zeros, i, 1);

    else if(vetor[i] < 0)
      update(negativos, i, 1);
}

getchar();

60 for(int i = 0; i < k; i++)
{
    char command;
    int s, t;

65   scanf("%c %d %d", &command, &s, &t);
   getchar();

    if(command == 'P')
    {
50       int tmp1, tmp2;

       tmp1 = read(zeros, t) - read(zeros, s - 1);
       tmp2 = read(negativos, t) - read(negativos, s - 1);

55       if(tmp1 > 0)
          printf("0");

       else if(tmp2 % 2 == 0)
          printf("+");
80       else printf("-");
    }

    else
75    {
85       if(t < 0)
        {
          if(vetor[s] > 0)
            update(negativos, s, +1);

90         else if(vetor[s] == 0)
            {
              update(negativos, s, +1);
              update(zeros, s, -1);
            }
        }
95    }

    else if(t == 0)
100   {
100      if(vetor[s] < 0)
        {
          update(negativos, s, -1);
          update(zeros, s, +1);
        }
105     else if(vetor[s] > 0)
          update(zeros, s, +1);
    }
}

```

```

        }

110    else
    {
        if(vetor[s] == 0)
            update(zeros, s, -1);

115    else if(vetor[s] < 0)
            update(negativos, s, -1);
    }

        vetor[s] = t;
120    }
}

printf( "#\n");
}
125
return 0;
}

```

45.14 Interval Product – Segmentation Tree

```

/*
 * File: main.cpp
 * Author: ubuntulap
 *
5  * Created on February 28, 2008, 6:16 PM
 */

#include <stdlib.h>
#include <stdio.h>
10 #include <string.h>
#include <math.h>
#include <limits.h>
#include <assert.h>
#include <cctype>
15 #include <iomanip>
#include <iostream>
#include <string>
#include <algorithm>
#include <utility>
20 #include <iterator>
#include <numeric>
#include <sstream>
#include <fstream>
#include <bitset>
25 #include <map>
#include <vector>
#include <set>
#include <stack>
#include <queue>
30 #include <deque>
#include <list>

#define SWAP(a,b) ( (a)=(b), (b)=(a), (a)=(b) )

```

```

#define MIN(a,b) ( (a)<(b) ? (a) : (b) )
#define MAX(a,b) ( (a)>(b) ? (a) : (b) )
#define ABS(a) ( (a)<(0) ? (-1)*(a) : (a) )
#define ROUND(a) ( (a)>=(0) ? ((int)((double)(a) + 0.5)) : ((int)((
    double)(a) - 0.5)) )
#define READ cin.getline(read, MAXSTRSZ)
#define para(x) for(typeof(x.begin()) it = x.begin(); it!=x.end(); 
    it++)
#define PI acos(-1.0)

#define FI(a) fastint(&a)
#define PL printf(" | n")

#define MAXSTRSZ      100000
#define MAXNUMBER     200000
#define INFINITO      999999999
#define EPS           1e-9

using namespace std;

typedef long long ll;
typedef unsigned long long ull;
typedef pair<int,int> ii;
typedef vector< pair<int,int> > vii;
typedef vector<int> vi;

#define LEFT(x)          (x << 1)
#define RIGHT(x)         (1 + (x << 1))

typedef struct
{
    int l, m, r, quant;
    int u;
}interval;

interval zero[1000600];
interval neg[1000600];
int elem[100600];

void init_interval(int idx, int left, int right, int& arv)
{
    interval *i;
    if(arv == 0)
    {
        i = zero + idx;
    }
    else
    {
        i = neg + idx;
    }
    i->u = 0;

    if(left == right)
    {
        if(arv == 0)
        {
            if(elem[left] == 0)

```

```

90     i->quant = 1;
  else
    i->quant = 0;
}
else
{
  if(elem[left] == -1)
    i->quant = 1;
  else
    i->quant = 0;
}
100
i->m = ((i->l = left) + (i->r = right)) >> 1;

105 if(left >= right)
  return;

init_interval(LEFT(idx), left, i->m, arv);
init_interval(RIGHT(idx), i->m + 1, right, arv);

110 if(arv == 0)
  i->quant = zero[LEFT(idx)].quant + zero[RIGHT(idx)].quant;
else
  i->quant = neg[LEFT(idx)].quant + neg[RIGHT(idx)].quant;
}

115

void printTree(int idx, int left, int right, int& arv)
{
  interval *i;
  if(arv == 0)
  {
    i = zero + idx;
  }
  else
  {
    i = neg + idx;
  }

  if(left >= right)
    return;

  printTree(LEFT(idx), left, i->m, arv);
  printTree(RIGHT(idx), i->m + 1, right, arv);

130

  if(arv == 0)
    printf("Zeros %d %d == %d\n", i->l, i->r, i->quant);
  else
    printf("Neg %d %d == %d\n", i->l, i->r, i->quant);
}

135

void refresh(int idx, int arv)
{
  interval *i;

  if(arv == 0)
    i = zero + idx;
  else

```

```

        i = neg + idx;

150    if(i->u == 0)
        return;

        i->quant = i->quant + i->u;

155    int add = i->u;

        i->u = 0;

        if(i->l >= i->r)
        return;

160    if(arv == 0)
    {
        zero[LEFT(idx)].u = zero[LEFT(idx)].u + add;
        zero[RIGHT(idx)].u = zero[RIGHT(idx)].u + add;
    }
    else
    {
        neg[LEFT(idx)].u = neg[LEFT(idx)].u + add;
        neg[RIGHT(idx)].u = neg[RIGHT(idx)].u + add;
    }
}

int nZeros, nNegs;

175 void query(int idx, int b, int e, int arv)
{
    interval *i;

180    if(arv == 0)
        i = zero + idx;
    else
        i = neg + idx;

185    if(b > i->r || e < i->l)
    {
        return;
    }

190    refresh(idx, arv);

        if(b <= i->l && e >= i->r)
    {
        if(arv == 0)
            nZeros += i->quant;
        else
            nNegs += i->quant;
        return;
    }

200    query(LEFT(idx), b, e, arv);
    query(RIGHT(idx), b, e, arv);
}

205 void update(int idx, int b, int e, int arv, int val)

```

```

{
    interval *i;

    if(arv == 0)
        i = zero + idx;
    else
        i = neg + idx;

    refresh(idx, argv);
215
    if(b > i->r || e < i->l)
        return;

    if(b <= i->l && e >= i->r)
220
    {
        (i->u) = (i->u) + val;

        refresh(idx, argv);
    }
225
    else
    {
        update(LEFT(idx), b, e, argv, val); update(RIGHT(idx), b, e, argv,
            , val);
        if(argv == 0)
            i->quant = zero[LEFT(idx)].quant + zero[RIGHT(idx)].quant;
230
        else
            i->quant = neg[LEFT(idx)].quant + neg[RIGHT(idx)].quant;
    }
}

235 void adjustval(int& val)
{
    if(val > 0)
        val = 1;
    else if (val < 0)
        val = -1;
}

int main()
{
245
    /* int i, j;
    int n, m, v1, v2;
    char c;

    while (scanf("%d %d\n", &n, &m) == 2)
    {
        for (i = 0; i < n; i++)
        {
            if (i != n-1)
                scanf("%d", &v1);
            else
                scanf("%d\n", &v1);

            if (v1 == 0)
            {
                isZero[i] = true;
            }
            else if(v1 < 0)
260
}
}

```

```

265      {
266          isNeg[i] = true;
267          isZero[i] = false;
268      }
269      else
270      {
271          isNeg[i] = false;
272          isZero[i] = false;
273      }
274  }
275
276  for(i = 0; i < m; i++)
277  {
278      scanf("%c %d %d\n", &c, &v1, &v2);
279      v1--;
280
281      if(c == 'C')
282      {
283          if(v2 == 0)
284          {
285              isZero[v1] = true;
286          }
287          else if(v2 < 0)
288          {
289              isNeg[v1] = true;
290              isZero[v1] = false;
291          }
292          else
293          {
294              isNeg[v1] = false;
295              isZero[v1] = false;
296          }
297      }
298      else
299      {
300          bool pos = true;
301          bool z = false;
302
303          for(j = v1; j < v2; j++)
304          {
305              if(isZero[j])
306              {
307                  z = true;
308                  break;
309              }
310
311              if(isNeg[j])
312                  pos = !pos;
313
314              if(z)
315                  printf("0");
316              else
317                  if(pos)
318                      printf("+");
319                  else
320                      printf("-");
321          }
322      }
323  }

```

```

        }
        PL;
    }

325   return 0; */

int i, j;
int n, m, v1, v2;
char c;

330 while (scanf ("%d %d\n", &n, &m) == 2)
{
    for (i = 1; i<=n; i++)
    {
335        if (i != n)
            scanf ("%d", &elem[i]);
        else
            scanf ("%d\n", &elem[i]);

        adjustval(elem[i]);
    }

    int arv = 0;
    init_interval(1, 1, n, arv);
345    arv = 1;
    init_interval(1, 1, n, arv);

    for (i = 0; i<m; i++)
    {
350        scanf ("%c %d %d\n", &c, &v1, &v2);

        if (c == 'C')
        {
            adjustval(v2);

355        if (elem[v1] != v2)
            {
                if (elem[v1] == 0 && v2 == -1)
                {
360                    update(1, v1, v1, 0, -1);
                    update(1, v1, v1, 1, 1);
                }
                else if (elem[v1] == 0 && v2 == 1)
                {
365                    update(1, v1, v1, 0, -1);
                }
                else if (elem[v1] == 1 && v2 == 0)
                {
                    update(1, v1, v1, 0, 1);
                }
370                else if (elem[v1] == 1 && v2 == -1)
                {
                    update(1, v1, v1, 1, 1);
                }
                else if (elem[v1] == -1 && v2 == 0)
                {
375                    update(1, v1, v1, 0, 1);
                    update(1, v1, v1, 1, -1);
                }
            }
        }
    }
}

```

```

380         }
381     else if(elem[v1] == -1 && v2 == 1)
382     {
383         update(1, v1, v1, 1, -1);
384     }
385     elem[v1] = v2;
386 }
387 else if(c == 'P')
388 {
389     nZeros = 0;
390
391     query(1, v1, v2, 0);
392
393     if(nZeros > 0)
394         printf("0");
395     else
396     {
397         nNegs = 0;
398         query(1, v1, v2, 1);
399
400         if(nNegs % 2)
401         {
402             printf("-");
403         }
404         else
405         {
406             printf("+");
407         }
408     }
409 }
410
411     printf(" | n");
412 }
413
414 return 0;
415 }
```

45.15 Kids – Gabriel

```

#include <stdio.h>
#include <iostream>
#include <set>
#include <vector>
5 #include <map>
#include <stack>

using namespace std;

10 map<int, set<int> > grafo;
map<int, int> pert;
int k, w, sizeCiclo, ini;

bool dfs(int ini, int prof, int anterior){
```

```

15    set<int>::iterator it2, it;

16    pair<int, int> v;
17    v.second = -1;
18    v.first = ini;
19    stack< pair<int, int> > pilha;
20    pilha.push(v);
21    pert[v.first] = -1;

22    while(!pilha.empty()){
23        v = pilha.top();
24        it2 = grafo[v.first].end();
25        it = grafo[v.first].begin();

26        while(it != it2){
27            if(pert.find(*it) == pert.end()){
28                v.second = v.first;
29                v.first = *it;
30                pilha.push(v);
31                pert[v.first] = v.second;
32
33                it = grafo[v.first].begin();
34                it2 = grafo[v.first].end();
35            }
36            else if(pert[*it] != v.first && *it != v.second){
37                sizeCiclo = pilha.size();
38
39                return true;
40            }
41            else ++it;
42        }
43
44        pilha.pop();
45    }
46
47    return false;
48 }

49
50 }

51
52 int main()
53 {
54     bool ok;
55     int a, b;
56     while(scanf("%d %d", &k, &w) && (k+w))
57     {
58         ok = true;
59         grafo.clear();
60
61         for(int i = 0; i < w; i++){
62             scanf("%d %d", &a, &b);
63             a--; b--;
64
65             if(!ok) continue;
66
67             grafo[a].insert(b);
68             grafo[b].insert(a);
69
70             if((int)grafo[a].size() > 2 || (int)grafo[b].size() > 2)
71                 ok = false;
72     }
73 }

```

```

        }

75     sizeCiclo = -1;
    if(!ok) printf( "N|n" );

    else{
        pert.clear();
80     for(map<int, set<int> >::iterator it = grafo.begin(); it != grafo.end(); ++it){
        if(pert.find(it->first) == pert.end()){
            sizeCiclo = -1;
            ini = it->first;
            if(dfs(it->first, 0, -1)) break;
85     }
    }

    if(sizeCiclo != -1) printf( "%c|n", sizeCiclo==k ? 'Y' : 'N' );
    else printf( "Y|n" );
90 }
}

return 0;
}

```

45.16 Kids – Leandro

```

/*
 * File: main.cpp
 * Author: ubuntulap
 *
5 * Created on February 28, 2008, 6:16 PM
 */

#include <stdlib.h>
#include <stdio.h>
10 #include <string.h>
#include <math.h>
#include <limits.h>
#include <assert.h>
#include <cctype>
#include <iomanip>
#include <iostream>
#include <string>
#include <algorithm>
#include <utility>
20 #include <iterator>
#include <numeric>
#include <sstream>
#include <fstream>
#include <bitset>
#include <map>
#include <vector>
25 #include <set>
#include <stack>
#include <queue>
#include <deque>
30

```

```

#include <list>

#define SWAP(a,b) ( (a)=(b), (b)=(a), (a)=(b) )
#define MIN(a,b) ( (a)<(b) ? (a) : (b) )
#define MAX(a,b) ( (a)>(b) ? (a) : (b) )
#define ABS(a) ( (a)<(0) ? (-1)*(a) : (a) )
#define ROUND(a) ( (a)>=(0) ? ((int)((double)(a) + 0.5)) : ((int)((
    double)(a) - 0.5)) )
#define READ cin.getline(read, MAXSTRSZ)
#define para(x) for(typeof(x.begin()) it = x.begin(); it!=x.end(); 
    it++)
#define PI acos(-1.0)

#define FI(a) fastint(&a)
#define PL printf(" | %n")
#define MAXSTRSZ      100000
#define MAXNUMBER     200000
#define INFINITO      999999999
#define EPS           1e-9

using namespace std;

typedef long long ll;
typedef unsigned long long ull;
typedef pair<int,int> ii;
typedef vector< pair<int,int> > vii;
typedef vector<int> vi;

void fastint(register int *n)
{
    register char c;
    *n = 0;

    while(c = getc(stdin), c < '0' || c > '9');

    do
    {
        (*n) = (*n)*10 + (c - '0');
    } while(c = getc(stdin), c >= '0' && c <= '9');
}

typedef struct
{
    int first, second, third;
} iii;

iii aux, ext;

iii newiii(int v1, int v2, int v3)
{
    aux.first = v1, aux.second = v2, aux.third = v3;
    return aux;
}

map<int, ii> anteriorprofundidade;
map<int, set<int> > graph;

```

```

int findCycles(int prim)
{
    int i;
    stack<iii> pilha;
    //printf("prim == %d\n", prim);

    pilha.push(newiii(prim, -1, 0));
    while(pilha.size())
    {
        ext = pilha.top(); pilha.pop();
        map<int, ii>::iterator it = anteriorprofundidade.find(ext.first);
        if(it != anteriorprofundidade.end())
        {
            if(anteriorprofundidade[ext.second].first != ext.first)
            {
                return ext.third - anteriorprofundidade[ext.first].second;
            }

            continue;
        }

        anteriorprofundidade[ext.first] = ii(ext.second, ext.third);
        typeof(graph[ext.first].begin()) itcomp = graph[ext.first].end();
        for(typeof(graph[ext.first].begin()) it = graph[ext.first].begin();
            it != itcomp; it++)
        {
            pilha.push(newiii(*it, ext.first, ext.third+1));
        }
    }

    return -1;
}

/*
 */
int main(int argc, char** argv)
{
    // checa se todos tem grau menor que 3 -> checa se haja ciclo (se sim, verifique se o tamanho é n)
    // map<int, set<int>> graph;
    int i, j, v1, v2;
    int n, m;

    while((FI(n), FI(m)), n || m)
    {
        anteriorprofundidade.clear(), graph.clear();
        bool could = true;

        int id = 0;

```

```

    for(i = 0; i<m; i++)
    {
        FI(v1), FI(v2);

145     if(!could)
        continue;

        graph[v1].insert(v2);
        graph[v2].insert(v1);

150     if(graph[v1].size() > 2 || graph[v2].size() > 2)
        could = false;
    }

155     if(could)
    {
        int c = -1;
        typeof(graph.begin()) itcomp = graph.end();
        for(typeof(graph.begin()) it = graph.begin(); it != itcomp;
            it++)
        {
            if(anteriorprofundidade.find(it->first) ==
                anteriorprofundidade.end())
            {
                c = findCycles(it->first);
                if(c != -1)
                    break;
            }
        }

160         if(c == -1 || c == n)
        {
            printf("Y\n");
        }
        else
        {
            printf("N\n");
        }
    }
    else
    {
165        printf("N\n");
    }

170         return 0;
}
175
180
185
}

```

45.17 Show sequence

```

/* Recursão , gerador de sequências */
#include <stdio.h>
#include <map>
5 #include <vector>
#include <iostream>

```

```

using namespace std;

10 int n;

vector<long long> processa(string &lei){
    vector<long long> gerada(n);
    int i=1;
    long long prim;

    if(lei[1] == '_') i = 2;
    for(; (lei[i] >= '0' && lei[i] <= '9'); i++);
    char s = lei[i];
    20 lei[i] = 0;
    lei[0] = ',';
    sscanf(lei.c_str(), "%lld", &prim);
    lei[0] = '/';
    lei[i] = s;

    if(s != '/'){

        string sublei = "";
        for(int j=i+1; j<lei.size()-1; j++)
            sublei.push_back(lei[j]);
        30
        vector<long long> ultima = processa(sublei);

        if(s == '+'){
            gerada[0] = prim;
            for(int k=1; k<n; k++)
                35 gerada[k] = gerada[k-1]+ultima[k-1];
        }

        else if(s == '_'){
            gerada[0] = prim;
            for(int k=1; k<n; k++)
                40 gerada[k] = gerada[k-1]-ultima[k-1];
        }

        else if(s == '*'){
            gerada[0] = prim*ultima[0];
            for(int k=1; k<n; k++)
                45 gerada[k] = gerada[k-1]*ultima[k];
        }
    }
    50 else
        for(int i=0; i<n; i++)
            gerada[i] = prim;

    return gerada;
55 }

int main(){
    string lei;

    60 while(cin >> lei && lei != "0"){
        scanf("%d", &n);

        vector<long long> seq = processa(lei);
}

```

```

65     bool first = true;
    for(int i=0; i<seq.size(); i++){
        if(!first) printf(" ");
        printf("%lld ", seq[i]);
        first = false;
    }

    printf("|\n");
75 }
}

```

45.18 Proportional [0,100]

```

/*
 * File: main.cpp
 * Author: ubuntulap
 *
5 * Created on February 28, 2008, 6:16 PM
 */

#include <stdlib.h>
#include <stdio.h>
10 #include <string.h>
#include <math.h>
#include <limits.h>
#include <assert.h>
#include <cctype>
15 #include <iomanip>
#include <iostream>
#include <string>
#include <algorithm>
#include <utility>
20 #include <iterator>
#include <numeric>
#include <sstream>
#include <fstream>
#include <bitset>
25 #include <map>
#include <vector>
#include <set>
#include <stack>
#include <queue>
30 #include <deque>
#include <list>

#define SWAP(a,b) ( (a)=(b), (b)=(a), (a)=(b) )
#define MIN(a,b) ( (a)<(b) ? (a) : (b) )
35 #define MAX(a,b) ( (a)>(b) ? (a) : (b) )
#define ABS(a) ( (a)<(0) ? (-1)*(a) : (a) )
#define ROUND(a) ( (a)>=(0) ? ((int)((double)(a) + 0.5)) : ((int)((
    double)(a) - 0.5)) )
#define READ cin.getline(read, MAXSTRSZ)
#define para(x) for(typeof(x.begin()) it = x.begin(); it!=x.end();
    it++)

```

```

40 #define PI acos(-1.0)

#define FI(a) fastint(&a)
#define PL printf(" | %n")
45 #define MAXSTRSZ      100000
#define MAXNUMBER     200000
#define INFINITO    999999999
#define EPS          1e-9

50 using namespace std;

typedef long long ll;
typedef unsigned long long ull;
55 typedef pair<int,int> ii;
typedef vector< pair<int,int> > vii;
typedef vector<int> vi;

void fastint(register int *n)
60 {
    register char c;
    register int neg = 0;
    *n = 0;

65    while(c = getc(stdin), c < '0' || c > '9')
        neg |= c == '_';

    do
    {
        (*n) = (*n)*10 + (c - '0');
    } while(c = getc(stdin), c >= '0' && c <= '9');

        (*n) = neg?(*n)*(-1):(*n);
    }

75 /*
 */
int main()
80 {
    int diff, p1, p2, dez1, cen1, dez2, cen2;
    char* str = (char*)malloc((MAXSTRSZ+1)*sizeof(char));
    int i, j;

    while(1)
    {
        scanf("%d", &diff);
        if(diff == -1)
            break;
85        scanf("%d%d", &p1, &p2);

        bool found = false;
        int alg;
90        for(i = 0; i<=2000; i++)
        {
            j = i + diff;

```

```

100      dez1 = (j/10)%10;
      cen1 = (j/100)%10;

      dez2 = (i/10)%10;
      cen2 = (i/100)%10;

105      if(dez1 == cen2 && dez2 == cen1 && dez2 == p2 * cen2 / p1)
    {
        printf( "%d\n", dez1);
        break;
    }
110  }

      return 0;
}

```

45.19 Tree Simulation (Space Elevator)

```

/*
 * File: main.cpp
 * Author: ubuntulap
 *
5  * Created on February 28, 2008, 6:16 PM
 */

10 #include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <limits.h>
#include <assert.h>
#include <cctype>
15 #include <iomanip>
#include <iostream>
#include <string>
#include <algorithm>
#include <utility>
20 #include <iterator>
#include <numeric>
#include <sstream>
#include <fstream>
#include <bitset>
25 #include <map>
#include <vector>
#include <set>
#include <stack>
#include <queue>
30 #include <deque>
#include <list>

#define SWAP(a,b) ( (a)=(b), (b)=(a), (a)=(b) )
#define MIN(a,b) ( (a)<(b) ? (a) : (b) )
35 #define MAX(a,b) ( (a)>(b) ? (a) : (b) )
#define ABS(a) ( (a)<(0) ? (-1)*(a) : (a) )

```

```

#define ROUND(a) ((a)>=(0) ? ((int)((double)(a)+0.5)) : ((int)((double)(a)-0.5)))
#define READ cin.getline(read, MAXSTRSZ)
#define para(x) for(typeof(x.begin()) it = x.begin(); it!=x.end(); it++)
40 #define PI acos(-1.0)

#define FI(a) fastint(&a)
#define PL printf("%n")
45 #define MAXSTRSZ 100000
#define MAXNUMBER 200000
#define INFINITO 999999999
#define EPS 1e-9
50 using namespace std;

typedef long long ll;
typedef unsigned long long ull;
55 typedef pair<int,int> ii;
typedef vector<pair<int,int>> vii;
typedef vector<int> vi;

void fastint(register int *n)
60 {
    register char c;
    register int neg = 0;
    *n = 0;

65    while(c = getc(stdin), c < '0' || c > '9')
        neg |= c == '-';
        do
    {
        (*n) = (*n)*10 + (c - '0');
    } while(c = getc(stdin), c >= '0' && c <= '9');

    (*n) = neg?(*n)*(-1):(*n);
}
75 int correctAns(int val, char* str)
{
    int valid = 0;
    int i, j;
    for(i = 1; i<=val; i++)
    {
        bool has = false;
        sprintf(str, "%d", i);

85        for(j = 0; str[j]; j++)
        {
            if(str[j] == '4' || (str[j] == '1' && str[j+1] == '3'))
                has = true;
        }

90        if(!has)
            valid++;
}

```

```

        }

95     return valid;
}

vector<string> nines (20);

100 #define cstr pair<char, string>

map< cstr , ull > mem;

ull calcula(char last, string orig)
{
    if(orig.size() == 0)
        return 1;

    cstr myPair = cstr(last, orig);

110     typeof(mem.begin()) it = mem.find(myPair);
    if(it != mem.end())
        return it->second;

    int i;
    int limit = orig[0] - '0';
    ull result1 = 0, result2 = 0, result3 = 0;

    int times = 0, toPass = -1;

120    for(i = 0; i<=limit; i++)
    {
        if(i == 4)
            continue;
        if(last == 1 && i == 3)
            continue;

        if(i == limit)
        {
            result1 += calcula(limit, orig.substr(1));
        }
        else
        {
            if(i == 1)
                result2 += calcula(i, nines[orig.size()-1]);
            else
                times++;
        }
    }

130    if(times)
        result3 += times * calcula(-1, nines[orig.size()-1]);

    mem[myPair] = result1 + result2 + result3;
}

140    return mem[myPair];
}

ull getRes(string orig)
{

```

```

        return calcula(-1, orig) - 1;
    }

char* str;
155
ull encontraValor(ull val)
{
    ull le = val, ld = ULLONG_MAX, mid, result, res1, res2;

160
    while(le <= ld)
    {
        ull res1 = le >> 1;
        ull res2 = ld >> 1;
        mid = res1 + res2;

165
        if(le & 1 && ld & 1)
            mid+=1LL;

        sprintf(str, "%llu ", mid);
        result = getRes(str);
        if(result == val)
        {
            ull lastRes = result;

175
            if(val > 1)
            {
                do
                {
                    sprintf(str, "%llu ", mid);
                    result = getRes(str);

180
                    if(result != lastRes)
                        break;

                    lastRes = result;
                    mid--;
                } while(1);
                mid++;
            }
        }

185
        return mid;
    }
    else if(result < val)
        le = mid + 1;
195
    else
        ld = mid;
}
}

200 /*
 * */
int main(int argc, char** argv)
{
    int i, j;
    str = (char*)malloc((MAXSTRSZ+1) * sizeof(char));
    string add = "";

```

```

210     for(i = 0; i<20; i++)
211     {
212         nines[i] = add;
213         add = add + '9';
214     }
215
216     /* for (ull aux = 0; aux <= 1000000LL; aux++)
217     {
218         if (correctAns(aux, str) != encontraValor(aux))
219         {
220             printf("we have a problem\n");
221             exit(0);
222         }
223     }
224
225     printf("%llu\n", getRes(argv[1]));
226     printf("%llu\n", encontraValor(atoi(argv[1]))); */
227
228     ull key;
229     char aux[120];
230     while(scanf("%llu", &key) == 1)
231     {
232         sprintf(aux, "%llu", key);
233         // printf("%llu\n", getRes(aux));
234         printf("%llu\n", encontraValor(key));
235     }
236
237     return 0;
238 }
```

45.20 Space Elevator – Internet Solution

```

//12486
//Space Elevator
//Misc; Binary Search
#include <iostream>
5 #include <cmath>
#include <string>
#include <cstring>
#include <iomanip>
#include <vector>
10 #include <algorithm>
#include <cstdio>
#define ull unsigned long long int
using namespace std;

15 ull T[20][10];

bool has(ull n, ull k, ull p) {
    while(n) {
        if (n%p==k) return true;
20        n/=10;
    }
    return false;
}
```

```

25 | bool has(ull n) {
|     return has(n, 13, 100) || has(n, 4, 10);
| }
|
30 |     ull right(ull n) {
|         int log10 = 0;
|         ull right = 0;
|
|         if (!has(n))
|             right++;
|
35 |         while(n) {
|             ull hi = n/10;
|             ull lo = n%10;
|
|             if (!has(hi)) {
|                 for(ull i=0; i<lo; i++) {
|                     if (i!=4 && (hi%10 != 1 || i!=3))
|                         right += T[log10][i];
|                 }
|             }
|
|             log10++;
|             n/=10;
|         }
|
50 |         return right-1;
|     }
|
55 |     ull answer(ull n) {
|         ull begin=0, end=-1;
|
|         while(begin+1 < end) {
|             ull mid = begin + (end - begin)/2;
|             ull v = right(mid);
|
60 |             if (v>=n)
|                 end = mid;
|             else
|                 begin = mid;
|         }
|
|         if (right(begin) == n)
|             return begin;
|         else
|             return end;
|     }
|
70 | }
|
75 | int main() {
|     for(ull i=0; i<10; i++) {
|         if (i==4) continue;
|         T[0][i] = 1;
|     }
|
|     for(ull i=1; i<20; i++) {
|         for(ull j=0; j<10; j++) {
|             if (j==4) continue;
|             for(ull k=0; k<10; k++) {
|
|                 if ((i+j+k)%10 == 4) continue;
|                 T[i][j][k] = 1;
|             }
|         }
|     }
| }
```

```
85         if (k==3 and j==1) continue;
     T[i][j] += T[i-1][k];
    }
}

90 ull n;
while(cin >> n) {
    cout << answer(n) << endl;
}
}
```

Parte V

Anexos

UFMG – Universidade Federal de Minas Gerais

1.	TEMPLATE	1
2.	NOTAS	1
3.	ÁLGEBRA	2
3.1	Simplex	2
3.2	Sistemas lineares	2
3.3	Eliminação Gaussiana	3
4.	GEOMETRIA	3
4.1	Intersecção de polígonos	4
4.2	Classificação de reta em relação a um círculo	4
4.3	Par de pontos mais próximos - Line scan sweeping	4
4.4	União de retângulos - Line scan sweeping e árvore de intervalos	4
4.5	Convex hull	5
4.6	Interseccão de retas e segmentos	5
4.7	Classificação de ponto em relação a um polígono	6
4.8	Centróide	6
4.9	Círculo definido por 3 pontos	6
4.10	Círculo mínimo que engloba uma lista de pontos	6
4.11	Distância na esfera	6
4.12	Geometria 3D	6
5.	CLASSES DE NÚMEROS	7
5.1	Inteiros de Precisão Arbitrária	7
5.2	Frações	7
6.	PROGRAMAÇÃO DINÂMICA	8
6.1	LIS 1D	8
6.2	LIS 2D	8
6.3	LCS	8
6.4	Maximum Sum 2D	9
6.5	Mochila 0-1 com conflito e mochila inteira	9
6.6	TSP	9
7.	JOGOS COMBINATÓRIAS	9
7.1	Posições vencedoras/perdeedoras (detecção de ciclos)	9
7.2	Nim	9
7.3	Grundy Numbers	9
8.	GRAFOS	10
8.1	Caminho Mínimo	10
8.2	AGM	10
8.3	Diâmetro de uma árvore	10
8.4	Componentes fortemente conectados (Aplicações: 2-SAT)	10
8.5	Pontes, pontos de articulação e componentes biconectados	11
8.6	Ordenação Topológica	11
8.7	Círcuito Euleriano	11
8.8	Fluxo máximo (Vertex cut, Cobertura mínima por caminhos em DAG)	11
8.9	Fluxo máximo de custo mínimo	12
8.10	Matching máximo em um grafo bipartido valorado - Hungariano	13
8.11	Grafo de restrições de diferenças	14
8.12	Cobertura mínima de vértices em árvores	14
8.13	Permanente de uma matriz (cobertura por ciclos e matching perfeito)	14
8.14	Stable marriage problem	14
8.15	Matching em grafo bipartido	15
8.16	Corte mínimo global	15
8.17	Matching em grafos quaisquer - Edmonds	15
9.	TEORIA DOS NÚMEROS	16
9.1	MDC e MMC	16
9.2	Eudílides estendido: $ax + by = \text{gcd}(a, b)$	16
9.3	Equações diofantinas lineares: $ax + by = c$	16
9.4	Inverso multiplicativo: $ax \equiv 1 \pmod{m}$	16
9.5	Menor solução não-negativa de $ax \equiv b \pmod{m}$	16
9.6	$C(n, k) \pmod{m}$	16
9.7	Exponencial modular: $b^e \pmod{m}$	16
9.8	Baby-step Giant-step: menor solução para e em $b^e \equiv n \pmod{m}$	16
9.9	Legendre symbol: existe x tal que $x^2 \equiv a \pmod{m}$	16
9.10	Fatoração em número primos	16
9.11	MDC($x!$, y)	16
9.12	Cálculo dos divisores de um inteiro n	17

UFMG – Universidade Federal de Minas Gerais

```

9.13    Cálculo do número de divisores dos inteiros de 1 a  $n$  ..... 17
9.14    Crivo de Eratóstenes ..... 17
9.15    Triângulo de Pascal ..... 17
9.16    Teste de primalidade com Miller-Rabin e Pollard-Rho ..... 17
9.17    Totient ..... 17
9.18     $a * b \pmod{m}$  ..... 18
9.19    Quantidade de números  $\leq n$  múltiplos de algum elemento do vetor  $v$  ..... 18
9.20    MATRÔIDES ..... 4
9.21    Intersecção de dois matróides (um gráfico e um de Partição) ..... 18
9.22    ESTRUTURAS DE DADOS ..... 19
9.23    LCA ..... 19
9.24    Árvore de segmentos - RMQ ..... 19
9.25    Árvore de intervalos ..... 21
9.26    Fenwick Tree (BIT) ..... 21
9.27    NOTA ..... 21
9.28    Bitwise operations ..... 21
9.29    Contagem de pontos que não são dominados por outros pontos em 3D ..... 22
9.30    Soma de medianas de intervalos de tamanho fixo de um vetor ..... 22
9.31    Algoritmo probabilístico ..... 22
9.32    Distância mínima (cavalos) em um tabuleiro de xadrez ..... 22
9.33    Josephus's problem ..... 22
9.34    Índice de uma permutação / permutação de um índice ..... 23
9.35    STRINGS ..... 23
9.36    Minimum Lexicographic Rotation ..... 23
9.37    Suffix array ..... 23
9.38    String matching - KMP ..... 25
9.39    String matching - Aho-Corasick ..... 25
9.40    **** TEMPLATE **** ..... 25
9.41    // Equipe: UFMG SUDO. Competidores: Felipe Menezes Machado, Leonardo Conegundes
9.42    // Martinez e Thiago Sonego Goulart. Coach: Itamar Sakae Viana Hata.
9.43    // algoritm, iostream, cmath, cstdio, cstlib, cstring, ctine, deque, stack,
9.44    // functional, iostream, list, map, numeric, queue, set, sstream, utility,
9.45    // iomanip, string, vector, tri/tuple, tri/unordered_map, tri/unordered_set
9.46    using namespace std; using namespace tr1;
9.47    #define for(i, n) for (int i = 0; i < (n); ++i)
9.48    #define for(i, a, b) for (int i = (a); i <= (b); ++i)
9.49    #define tr(T, i) for (typeof(T.begin()) i = T.begin(); i != T.end(); ++i)
9.50    #define sz size()
9.51    #define all(x) (x).begin(), (x).end()
9.52    #define _sort(x) sort(all(x))
9.53    #define pb push_back
9.54    #define TRACE(x...) x
9.55    #define PRINT(x...) TRACE(printh(x))
9.56    #define WATCH(x) TRACE(cout << "#x" = " " << x << "\n")
9.57    const double EPS = 1e-9; const int INF = 0x3F3F3F;
9.58    int cmpd(double x, double y = 0, double tol = EPS) {
9.59        return (x <= y + tol) ? (x + tol < y) ? -1 : 0 : 1;
9.60    }
9.61    //***** NOTAS ***** /////
9.62    -> PI: 3.14159265358979323846
9.63    Number or primes: n/in(n) < f(n) < 1.26*n/ln(n)
9.64    -> Modular multiplicative inverse:  $A^x \equiv 1 \pmod{M}$ 
9.65    If  $A$  and  $M$  are coprime, then  $x = \phi(m) - 1$ , where:
9.66     $\phi(n) = (p_1-1)*p_1^{e_1-1} * \dots * (p_n-1)*p_n^{e_n-1}$ 
9.67    -> Polygon number:  $P(s, n) = ((s-2)*n*n - (s-4)*n) / 2$ 
9.68    Não confundir com binomial coefficients!
9.69    -> Catalan Number:
9.70    1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900,

```

```

    } // minizacao

    class Mixture {
public:
    double mix(vector<int> mixture, vector<string> availableMixtures) {
        string s; int k; int m = mixture.size(); int n = availableMixtures.size();
        vector<vector<double>> A(m+n, vector<double>(n));
        vector<double> c(m), b(m+n);
        for(i,m) b[m+i] = -(b[i] = mixture[i]); // vetor b
        for(i,n) { // matriz A e vetor b
            s = availableMixtures[j];
            istringstream is(s);
            for(i,m){ is >> k; A[m+i][j] = -(A[i][j] = k); }
            is >> k; c[j] = -k;
        }
        simplex(S(A, b, c)); double asw = 0;
        vector<double> sol = S.solve();
        if (sol.empty()) asw = 1;
        else for(i,n) asw += c[i] * sol[i];
        return -asw;
    }

    / Linear Systems: You should fill the tableau T. Let
    / A be an m x n matrix, so the tableau will be (with b[m] and c[n]): 
    T[m+1][n+1] = [ [-f / c[n]] ]
                  [ b[m] / A[m][n] ]
    }

    vector<double> solve() {
        m = A.size(); n = A[0].size(); int k = m+n+1;
        N = vector<bool>(k, true); vector<double> c_copy = c;
        for(i,m){ N[i+m] = false; kt.resize(m); }

        A[i].resize(k); A[i][n+i] = 1; A[i][k-1] = -1;
        kt[i] = n+i; N[kt[i]] = false;
    }

    int l = min_element(all(b)) - b.begin();
    if (cmpD(b[l]) < 0) {
        c = vector<double>(k, 0);
        c[k-1] = -1; pivot(1, k-1); sol = go(k);
        if (cmpD(sol[k-1])>0) return vector<double>(); // infeasible
        for(i,m) if (kt[i] == k-1) {
            for(j,k-1) if (N[j] && cmpD(A[j][i]) != 0) {
                pivot(k, i, j); break;
            }
        }
        c = c_copy; c.resize(k, 0); c[j] -= c[kt[i]] * A[i][j];
    }
    sol = go(k-1);
    if (!sol.empty()) sol.resize(n);
    return sol;
}

    / minimizacao
}

```

```

 $\text{// Invert a matrix } A[m][m] - \text{store the matrix in the tableau and the identity in}$ 
 $\text{// } T[1..m][m+1..2m], \text{make } n = 2*m \text{ and call solve\_linear\_system(). Get the inv.}$ 
 $\text{// matrix at } T[1..m][m+1..2m]. \text{ This code doesn't suppose } m == n. \text{ After the}$ 
 $\text{// execution, if possible, } m \text{ is the rank of the matrix. } m \text{ is the number of lines}$ 
 $\text{// and } n \text{ the number of columns ( variables ).}$ 
int n, m; double x[MAXN+1], T[MAXN+1][MAXN+1]; // tableau
void pivot( int l, int j ) { (k!=j) T[l][k] /= T[l][j]; T[l][j] = 1;
    for(i,1,m) if ( i != l ) { (k!=j) T[i][k] -= T[l][k] * T[i][j]; T[i][j] = 0;
        for(k,n+1) if ( k!=j ) T[i][k] -= T[l][k] * T[i][j]; T[i][j] = 0;
    }
}
bool solve_linear_system() {
    for(j,1,n) T[0][j] = j;
    for(i,1,min(m,n)) {
        int p = i;
        for(k,i+1,m) if ( cmpD( fabs(T[k][i]), fabs(T[p][i]) ) > 0 ) p = k;
        if ( p!=i ) for(j,j+1,m) swap(T[i][j], T[p][j]);
        if ( cmpD(T[i][i])==0 ) { return false;
            else if ( cmpD(T[i][i])!=0 ) pivot(i,i);
            else {
                for(i,j,n+1) swap(T[i][j], T[m][j]);
                --i; --m;
            }
        }
        forr(i,(min(m,n)+1,m) if ( cmpD(T[i][0]) != 0 ) return false;
        if ( m>n ) m = n;
        return true;
    }
}
void get_solution()
{
    forr(i,1,n) x[i] = 0;
    forr(j,1,m) x[(int)T[0][j]] = T[j][0];
}
// eliminacao gaussiana
void gaussian_elimination(double a[MAX][MAX], double b[MAX], int n) {
    for (i, 1, k, l, maxi; double f, aux;
         (i=0; i < n; i++) {
        maxi = i;
        for (l=i; l < n; l++) {
            if (fabs(a[l][i]) > fabs(a[maxi][i])) maxi = l;
        }
        for (l=0; l < n; l++) swap(a[i][l], a[maxi][l]);
        aux = b[i]; b[i] = b[maxi]; b[maxi] = aux;
        for (k=i+1; k < n; k++) {
            f = a[k][i] / a[i][i];
            for (j=i; j < n; j++) a[k][j] -= a[i][j] * f;
            b[k] -= b[i] * f;
        }
    }
    for (i=n-1; i >= 0; i--) {
        b[i] = b[i] / a[i][i]; a[i][i] = 1.0;
        for (j=i-1; j >= 0; j--) { b[j] -= a[j][i] * b[i]; a[j][i] = 0.0; }
    }
}
// ***** GEOMETRIA *****
struct point {
    double x, y;
    point(double x = 0, double y = 0): x(x), y(y) {}
}

```

```

point operator +(point q) { return point(x + q.x, y + q.y); }
point operator -(point q) { return point(x - q.x, y - q.y); }
point operator *(double t) { return point(x * t, y * t); }
point operator /(double t) { return point(x / t, y / t); }
double operator %(point q) { return x * q.x + y * q.y; }
int cmp(point q) const { return x * q.y - y * q.x; }
if (int t = ::cmp(x, q.x)) return t; return ::cmp(y, q.y);
}
bool operator ==(point q) const { return cmp(q) == 0; }
bool operator !=(point q) const { return cmp(q) != 0; }
bool operator <(point q) const { return cmp(q) < 0; }
friend ostream& operator <<(ostream& o, point p) {
    return o << "(" << p.x << "," << p.y << ")";
}
static point pivot;
point point::pivot;
double abs(point p) { return hypot(p.x, p.y); }
double arg(point p) { return atan2(p.y, p.x); }
typedef vector<point> polygon;
int ccw(point p, point q, point r) { return cmp((p - r) % (q - r)); }
double angle(point p, point q, point r) { return atan2(q.y - p.y, q.x - p.x); }
point u = p - q, v = r - q; return atan2(u.y, u.x);
}
// Normaliza o vetor para tamanho unitario. Retorna -1 se o vector e' 0
int normalize(point& p) {
    double r = abs(p);
    if( cmp(r) != 0 ) return -1;
    p.x /= r; p.y /= r;
    return 0;
}
// Decide se q esta sobre o segmento fechado pr
bool between(point p, point q, point r) { return (p - q) * (r - q) * (r - q) <= 0; }
// Decide se os segmentos fechados pq e rs tem pontos em comun.
// s e' o ponto de pq que esta mais proximo de r
bool seg_intersect(point p, point q, point r, point s) {
    point A = q - p, B = s - r, C = r - p, D = s - q;
    int a = cmp(A % C) + 2 * cmp(A % D), b = cmp(B % C) + 2 * cmp(B % D);
    if (a == 3 && a == -3 || b == 3 && b == -3) return false;
    if (a || b || p == r || p == s || q == r || q == s) return true;
    int t = (p < r) ? (p < s) + (q < r) + (q < s);
    int t = (p < r) + (p < s) + (q < r) + (q < s);
    return t != 4;
}
// Calcula a distancia do ponto r ao segmento pq.
double seg_distance(point p, point q, point r) {
    point A = r - q, B = r - p, C = q - p;
    double a = A * A, b = B * B, c = C * C;
    if (cmp(b, a + c) >= 0) return sqrt(a);
    else if (cmp(a, b + c) >= 0) return sqrt(b);
    else return fabs(A % B) / sqrt(c);
}
// classifica o ponto p em relacao ao poligono T. Retorna 0, -1 ou 1 dependendo
// se p esta no exterior, na fronteira ou no interior de T, respectivamente.
int in_poly(point p, polygon T) {
    double a = 0; int N = T.size();
    for (int i = 0; i < N; i++) {
        if (between(T[i], p, T[(i+1) % N])) return -1;
        a += angle(T[i], p, T[(i+1) % N]);
    }
    return cmp(a) != 0;
}

```

```

 $\text{// Encontra o ponto de intersecao das retas } p\text{ e }r\text{s.}$ 
 $\text{point line\_intersect(point } p\text{, point } q\text{, point } r\text{, point } s\text{) \{} \text{// reacao a coordenada } x \text{ do ponto atual}$ 
 $\text{    point a = q - p\text{, }b = s - r\text{, }c = point(p \% q\text{, }r \% s)\text{;}$ 
 $\text{    return point(point(a.x, b.x) \% c, point(a.y, b.y) \% c)) / (a \% b)\text{;}$ 
 $\text{\}}$ 
 $\text{// Determina o poligono intersecao dos dois poligonos convexos } P \text{ e } Q\text{.}$ 
 $\text{int m = Q.size()\text{, }n = P.size()\text{;}$ 
 $\text{int a = 0\text{, }b = 0\text{, }aa = 0\text{, }ba = 0\text{, }inflag = 0\text{;}$ 
 $\text{polygon R\text{;}}$ 
 $\text{while ((aa < n} \text{|| } ba < m) \&\& aa < 2*n \&\& ba < 2*m) \{$ 
 $\text{    point p1 = p1 + p[(a+1) \% n]\text{, }q1 = Q[b]\text{, }q2 = Q[(b+1) \% m]\text{;}$ 
 $\text{    point A = p2 - p1\text{, }B = q2 - q1\text{; }$ 
 $\text{    int cross = cmp(A \% B).ha = ccw(p2, q2, p1)\text{, }hb = ccw(q2, p2, q1)\text{;}$ 
 $\text{    if (cross == 0 \&\& ccw(p1, q1, p2) == 0 \&\& cmp(A * B) < 0) \{$ 
 $\text{        if (between(p1, q1, p2)) R.push_back(q1)\text{;}$ 
 $\text{        if (between(p1, q2, p2)) R.push_back(q2)\text{;}$ 
 $\text{        if (between(q1, p1, q2)) R.push_back(p1)\text{;}$ 
 $\text{        if (between(q1, p2, q2)) R.push_back(p2)\text{;}$ 
 $\text{        if (R.size() < 2) return polygon()\text{;}$ 
 $\text{        inflag = 1\text{; break}\text{;}}$ 
 $\text{    } \text{else if (cross != 0 \&\& seg_intersect(pl, p2, q1, q2)) \{$ 
 $\text{        if (inflag == 0) aa = ba = 0\text{;}$ 
 $\text{        R.push_back(line_intersect(pl, p2, q1, q2))\;}$ 
 $\text{        inflag = (hb > 0) ? 1 : -1\text{;}}$ 
 $\text{    } \text{else if (cross == 0 \&\& hb < 0 \&\& ha < 0) return R\text{;}$ 
 $\text{    } \text{if (t ? (inflag == 1) : (cross >= 0) ? (ha <= 0) : (hb > 0) ) \{$ 
 $\text{        ba++; b++; b \%= m\text{;}}$ 
 $\text{    } \text{else \{$ 
 $\text{        if (inflag == 1) R.push_back(q2)\;}$ 
 $\text{        aa++; a++; a \%= n\text{;}}$ 
 $\text{    } \text{if (inflag == 0) \{$ 
 $\text{        if (in_poly(P[0], Q)) return P\text{;}$ 
 $\text{        if (in_poly(Q[0], P)) return Q\text{;}}$ 
 $\text{    } \text{R.erase(unique(all(R)', R.end'))\;}$ 
 $\text{    if (R.size() > 1 \&\& R.front() == R.back()) R.pop_back()\;}$ 
 $\text{    return R\text{;}}$ 
 $\text{\}}$ 
 $\text{// Classifica a reta } pq \text{ em relacao ao circulo } C \text{ - NAO FOI TESTADA}$ 
 $\text{// Retorna } 0 \text{ se } pq \text{ intersecta } C \text{ em } 0 \text{ pontos, } 1 \text{ se } pq \text{ intersecta } C \text{ em } 1$ 
 $\text{// ponto, } 2 \text{ se } pq \text{ intersecta } C \text{ em } 2 \text{ pontos, } r \text{ eh } s \text{ eh onde ha intersecao}$ 
 $\text{int line_circle(circle } C\text{, point } p\text{, point } q\text{, point\& } r\text{, point\& } s\text{) \{} \text{// reacao a coordenada } x \text{ do ponto mais proximos}$ 
 $\text{    point m, double r0 = seg_distance(p, q, C.first, m)\text{;}$ 
 $\text{    if ( cmp(r0, C.second) > 0) return 0\text{;}$ 
 $\text{    else if ( cmp(r0, C.second) == 0 ) \{r = s = m; return 1\text{;}}$ 
 $\text{    else \{$ 
 $\text{        double dd = sqrt(C.second*C.second+r0*r0)\text{; point } v = q-p\text{;}$ 
 $\text{        normalize(v)\text{; } r = m-v*dd\text{; } s = m+v*dd\text{;}}$ 
 $\text{        return 2\text{;}}$ 
 $\text{\}}$ 
 $\text{// line scan sweeping}$ 
 $\text{\#define px second}$ 
 $\text{\#define py first}$ 
 $\text{typedef pair<double, double> pairdd;$ 
 $\text{int n, t; // numero de pontos e numero de casos de testes}$ 
 $\text{pairdd pts [10000]; // conjunto de ponto}$ 

```

```

 $\text{set< pairdd > box: // armazena todos os pontos a uma distancia maxima em}$ 
 $\text{    // reacao a coordenada } x \text{ do ponto atual}$ 
 $\text{    double best; // distancia entre os dois pontos mais proximos}$ 
 $\text{    pairdd pnt_a, pnt_b; // os dois pontos mais proximos}$ 
 $\text{    int compx( pairdd a, pairdd b ) \{return cmp( a.px, b.px ) < 0\; \}}$ 
 $\text{    // armazena em pnt_a e pnt_b o par de pontos mais proximos}$ 
 $\text{    void closest_points() \{sort( pts, pts+n, compx )\;}$ 
 $\text{    best = INF\;}$ 
 $\text{    box.insert( pts[0] )\;}$ 
 $\text{    int left = 0\;}$ 
 $\text{    for(i,1,n-1) \{$ 
 $\text{        // remove pontos a uma distancia maior que best na coordenada } x \text{ do } pts[i]$ 
 $\text{        while ( left < i \&\& cmpD( pts[i].px - pts[left].px, best ) > 0 )$ 
 $\text{            box.erase( pts[i].left++ )\;}$ 
 $\text{        // compara todos os pontos do conjunto box com o } pts[i]$ 
 $\text{        for ( typeof( box.begin() ) it = box.lower_bound(make_pair( pts[i].py - best, pts[i].px - best )) ;$ 
 $\text{            it != box.end() \&\& cmpD( pts[i].py + best, it->py ) >= 0; ++it ) \{$ 
 $\text{            double dx = pts[i].px - it->px, dy = pts[i].py - it->py\;}$ 
 $\text{            double dist = sqrt( dx * dx + dy * dy )\;}$ 
 $\text{            if ( cmpD(dist, best) < 0 ) \{best = dist; pnt_a = pts[i]; pnt_b = *it;\}}$ 
 $\text{        } \text{box.insert( pts[i] )\;}$ 
 $\text{    } \text{int main () \{$ 
 $\text{        scanf( "%d", \&t );$ 
 $\text{        while ( t-- ) \{$ 
 $\text{            scanf( "%d", \&n );$ 
 $\text{            box.clear();$ 
 $\text{            for(i,1,n) scanf( "%lf%lf", \&pts[i].px, \&pts[i].py );$ 
 $\text{            closest_points();$ 
 $\text{            printf( "%3lf%.3lf\n", (pnt_a.px + pnt_b.px)/2, (pnt_a.py + pnt_b.py)/2 );$ 
 $\text{            if ( t ) puts( "\n" );$ 
 $\text{        } \text{box.insert( pts[i] )\;}$ 
 $\text{        return 0; \}}$ 
 $\text{    } \text{// union of rectangles}$ 
 $\text{    // O(n^2), em que n eh o numero de retangulos}$ 
 $\text{    struct event \{$ 
 $\text{        int ind; // Index of rectangle in rects$ 
 $\text{        bool type; // Type of event: 0 = Lower-left; 1 = Upper-right$ 
 $\text{        event() \{} \text{event( int ind, int type ) : ind( ind ), type( type ) \{\};$ 
 $\text{    } \text{struct point \{ int x, y; \};}$ 
 $\text{    } \text{int n, e; // n = number of rectangles; e = number of edges}$ 
 $\text{    } \text{point rects[10010][2]; // Each rectangle consists of 2 points:$ 
 $\text{    } \text{int delta_x, delta_y; // distance between current sweep line and previous$ 
 $\text{    } \text{event events[20010], events_h[20010]; // Events of vertical/horiz. Sweep line}$ 
 $\text{    } \text{bool compare_x( event a, event b ) \{rects[a.ind][a.type].x < rects[b.ind][b.type].x; \}}$ 
 $\text{    } \text{bool compare_y( event a, event b ) \{rects[a.ind][a.type].y < rects[b.ind][b.type].y; \}}$ 
 $\text{    } \text{return rects[a.ind][a.type].y < rects[b.ind][b.type].y \};$ 
 $\text{    } \text{int in_set[10010]; // Boolean array in place of balanced binary tree (set)}$ 
 $\text{    } \text{long long area; // The output: Area of the union}$ 
 $\text{    } \text{int main() \{$ 
 $\text{        while ( scanf( "%d", \&n ) == 1 ) \{ // x -> v; y -> h$ 
 $\text{            e = area = 0\;}$ 
 $\text{            memset( in_set, 0, sizeof( in_set ) )\;}$ 

```

```

for(i=1,n) {
    scanf ("%ld %ld", &rects[i][0].x, &rects[i][0].y); // Lower-left coordinate
    scanf ("%ld %ld", &rects[i][1].x, &rects[i][1].y); // Upper-right coordinate
    events_v[e] = event( i, 0 );
    events_h[e] = event( i, 1 );
    events_h[e+1] = event( i, 1 );
}

sort( events_v, events_v + e, compare_x );
sort( events_h, events_h + e, compare_y ); // sort set of horizontal edges
in_set[events_v[0].ind] = 1;
for(i,1,e-1) {
    event c = events_v[i]; // Vertical sweep line
    int cnt = 0; // how many rectangles are currently overlapping?
    delta_x = rects[c.ind][c.type].x - rects[events_v[i-1].ind][events_v[i-1].type].x;
    int begin_y = 0;
    //if ( delta_x == 0 ) continue;
    if ( events_h[j].type == 0 ) { // Horizontal sweep line
        for(j,e) if ( in_set[events_h[j].ind] == 1 ) {
            if ( events_h[j].type == 0 ) { // Block starts
                if ( cnt == 0 ) begin_y = rects[events_h[j].ind][0].y; ++cnt;
            }
            else {
                --cnt;
                if ( cnt == 0 ) { // Block ends
                    delta_y = (rects[events_h[j].ind][1].y - begin_y);
                    area += delta_x * delta_y;
                }
            }
        }
        in_set[c.ind] = ( c.type == 0 );
    }
    printf( "%lld\n", area );
}
return 0;
}

// O(n log K) com arvore de intervalos' onde n eh o numero de retangulos
// e K eh o tamanho do maior intervalo (tamanho do eixo Y valido)
int main() {
    scanf( "%d", &n ) == 1 ) { // x -> v; y -> h
    e = area = 0;
    int max_y = 0, min_y = INF;
    for(i,n) {
        scanf( "%d %d", &rects[i][0].x, &rects[i][0].y );
        min_y = min(min_y, rects[i][0].y); max_y = max(max_y, rects[i][1].y);
        events_v[e++] = event( i, 0 );
        events_v[e++ ] = event( i, 1 );
    }
    sort( events_v, events_v + e, compare_x );
    create( min_y, max_y );
    update( rects[events_v[0].ind][0].y, rects[events_v[0].ind][1].y, 1 );
    for(i,1,e-1) {
        event c = events_v[i];
        delta_x = rects[c.ind][c.type].x - rects[events_v[i-1].ind][events_v[i-1].type].x;
        delta_y = count_occurrences( min_y, max_y );
        area += delta_x * delta_y;
        if ( c.type == 0 ) update( rects[c.ind][0].y, rects[c.ind][1].y, -1 );
    }
    printf( "%lld\n", area );
}
return 0;
}

```

```

typedef struct {
    int x, y;
} Point;
typedef Point Polygon[MAX];
double dist_pr(double x, double y, double a, double b, int inf) {
    if (inf) return fabs(b-x);
    if (equals(a, 0.0)) return fabs(b-y);
    double tgaux, baux;
    if (a->x == xmin && b->x == xmin) return b->y - a->y;
    if (a->x == xmin && b->x == xmax) return 1;
    if (b->x == xmin) return -1;
    if ((a->y == ymax && b->y == ymax) ||
        (a->y == ymin && b->y == ymin) ||
        prod_vet(a->x-xmin, a->y-ymax, b->x-xmin, b->y-ymax) == 0) return a->x - b->x;
    return -prod_vet(a->x-xmin, a->y-ymax, b->x-xmin, b->y-ymax);
}

void sort(Polygon p, int n) {
    int i, j, min = 0;
    Point aux;
    for (i=1; i < n; i++) {
        if (p[i].x < p[min].x || (p[i].x == p[min].x && p[i].y > p[min].y))
            min = i;
    }
    if (min != 0) aux = p[0], p[0] = p[min], p[min] = aux;
    xmin = p[0].x, ymax = p[0].y;
    qsort(&p[1],n-1,SIZEOF(Point), (void*)cmp_sort);
    for (i=n-1; i > 0 && prod_vet(p[i].x-p[0].x,p[i].y-p[0].y,p[i-1].x-p[0].x,
        p[i-1].y-p[0].y) == 0; i--);
    for (j=0; j < (n-i)/2; j++) aux = p[i+j], p[i+j] = p[n-j-1], p[n-j-1] = aux;
}

int convex_hull(Polygon p, int n, Polygon hull) {
    int qtdePontos = 2, i;
    double xl, yl, x2, y2;
    sort(p,n);
    hull[0] = p[0], hull[1] = p[1];
    for (i=2; i <= n; i++) {
        do {
            x1 = p[i%n].x - hull[qtdePontos-1].x;
            y1 = p[i%n].y - hull[qtdePontos-1].y;
            x2 = hull[qtdePontos-2].x - hull[qtdePontos-1].x;
            y2 = hull[qtdePontos-2].y - hull[qtdePontos-1].y;
            if (prod_vet(x1,y1,x2,y2) <= 0 && i != n) qtdePontos--;
        } while (qtdePontos > 1);
        if (i == n) hull[qtdePontos++] = p[i];
    }
    return qtdePontos;
}

// Interseccao de retas e segmentos.
int intersect(double x0, double y0, double xl, double yl, double x2, double y2,
    double al, bl, a2, b2, double x3, double y3, double x, double y) {
    if (equals(x0,x1) && equals(x2,x3)) return 0; // tratar retas verticais?
    if (!equals(x2,x3)) a1 = (yl-y0)/(x1-x0), bl = y0 - a1*x0;
    if (!equals(x2,x3)) a2 = (y3-y2)/(x3-x2), b2 = y2 - a2*x2;
    if (equals(x0,x1)) *x = x0, *y = a2*x0 + b2;
    else if (equals(x2,x3)) *x = x2, *y = a1*x2 + bl;
}

```

```

else {
    if (equals(al,a2)) return 0; // tratar retas colineares?
    *x = (b2 - b1) / (al - a2), *y = al*(*x) + bl;
}
if (!equals(dist_PP(*x,*y,x0,y0)+dist_PP(*x,*y,x1,y1)))
    return 0; // se (p0,p1) eh segmento
if (equals(dist_PP(*x,*y,x2,y2)+dist_PP(*x,*y,x3,y3),dist_PP(x2,y2,x3,y3)))
    return 0; // se (p2,p3) eh segmento
return 1;
}

// Interseccao de segmentos (v2). pv = prod_vet
int segment_intersect(int xl, int yl, int x2, int y2, int x3, int y3, int x4, int y4) {
    ((pv(xl-x2,yl-y2,x3-x2,y3-y2) < 0 && pv(xl-x2,yl-y2,x4-x2,y4-y2) > 0) ||
     (pv(x4-x3,y4-y3,x1-x3,y1-y3) < 0 && pv(x4-x3,y4-y3,x2-x3,y2-y3) > 0) ||
     (pv(x4-x3,y4-y3,x1-x3,y1-y3) > 0 && pv(x4-x3,y4-y3,x2-x3,y2-y3) < 0));
}

// Ponto dentro de um poligono anti-horario qualquer.
int point_in_pol(int x, int y, Polygon p, int n) {
    int i, cuts = 0;
    double xaux;
    for (i=1; i <= n; i++) {
        if ((p[i-1].x == x && p[i-1].y == y) ||
            (prod_vet(p[i-1].x-x,p[i-1].y-y,p[i-1].x-x,p[i-1].y-y) == 0 &&
             equals(dist_PP(x,y,p[i-1].x,p[i-1].y)+dist_PP(x,y,p[i-1].x,p[i-1].y)),
            dist_PP(p[i-1].x,p[i-1].y,p[i-1].x,p[i-1].y))) {
                return 1;
            }
        if (p[i-1].y == y && p[i-1].y == p[i-1].y) { cuts++; continue; }
        else xaux = (y - p[i-1].y + (p[i-1].y-p[i-1].y)/(double)(p[i-1].x-p[i-1].x)*
                    p[i-1].x) / ((p[i-1].y-p[i-1].y)/(double)(p[i-1].x-p[i-1].x));
        if (xaux+EPS > x && ((y > p[i-1].y) || (p[(n+i-2)%n].y > y && p[(i+1)%n].y < y))) cuts++;
        continue;
    }
    if (p[i%n].y == y && p[i%n].x > x && ((p[i-1].y < y && p[(i+1)%n].y > y) ||
        (p[i-1].y > y && p[(i+1)%n].y < y))) { cuts++; continue; }
    else xaux = (y - p[i-1].y + (p[i-1].y-p[i-1].y)/(double)(p[i-1].x-p[i-1].x)*
                    p[i-1].x) / ((p[i-1].y-p[i-1].y)/(double)(p[i-1].x-p[i-1].x));
    if (xaux+EPS > x && ((y > p[i-1].y) || (y < p[i-1].y && y > p[i-1].y))) cuts++;
    return cuts & 1;
}

void centroid(Polygon p, int n, Point_d *r) {
    int i, x = 0, y = 0; double s = area(p,n);
    for (i=1; i <= n; i++) {
        x += (p[i-1].x + p[i].x)*prod_vet(p[i-1].x, p[i-1].y, p[i-1].y, p[i-1].y);
        y += (p[i-1].y + p[i].y)*prod_vet(p[i-1].x, p[i-1].y, p[i-1].y, p[i-1].y);
    }
    r->x = x/(6.0*s); r->y = y/(6.0*s);
}

// Circulo definido por 3 pontos.
double x, y, r;
void find_circle(Point *p1, Point *p2, Point *p3) {
    void find_x1_y1_x2_y2_x3_y3(temp);
    double x1, y1, x2, y2, x3, y3, temp;
    x1 = p1->x, y1 = p1->y;
    x2 = p2->x, y2 = p2->y;
    x3 = p3->x, y3 = p3->y;
    if (equals(x1,x2)*x1*x1+y1*y1*x2*x2-y3*y3-(x1-x3)*(x1*x1+y1*y1-x2*x2-y2*y2)) /
        (2*((Y2-Y1)*(x1-x3)-(Y3-Y1)*(x1-x2)));
    x = (x1*x1+y1*y1-x2*x2-Y2*x2+2*y*(Y2-Y1)) / (2*(x1-x2));
    r = sqrt((x-x1)*(x-x1)+(y-y1)*(y-y1));
}

```

```

// Menor circulo que engloba um conjunto de pontos.
// -> soh funcoes para pontos que facam parte do convex hull do conjunto!!!!!
void min_circle(Polygon p, int n) {
    int i = 0, j = 1, k, posmin;
    double PI = acos(-1.0), min, a;
    while (1) {
        min = PI;
        for (k=0; k < n; k++) {
            if (k != i && k != j) {
                a = acos(((p[i].x-p[k].x)*(p[j].x-p[k].x)+(p[i].y-p[k].y)*(p[j].y-p[k].y)) /
                    (sqrt((p[i].x-p[k].x)*(p[i].x-p[k].x)+(p[i].y-p[k].y)*(p[i].y-p[k].y)))*
                    sqrt((p[j].x-p[k].x)*(p[j].x-p[k].x)+(p[j].y-p[k].y)*(p[j].y-p[k].y))));
                if (a < min) min = a, posmin = k;
            }
        }
        if (min+EPS > PI/2) {
            x = (p[i].x + p[j].x) / 2.0; y = (p[i].y + p[j].y) / 2.0;
            r = sqrt((p[i].x-p[j].x)*(p[i].x-p[j].x) +
                    (p[i].y-p[j].y)*(p[i].y-p[j].y)) / 2;
            return;
        }
        if (acos(((p[posmin].x-p[i].x)*(p[j].x-p[i].x)+(p[posmin].y-p[i].y)*
                    (p[j].y-p[i].y)) / (sqrt((p[posmin].x-p[i].x)*(p[posmin].y-p[i].y))*sqrt((p[j].x-p[i].x)*
                    (p[posmin].y-p[i].y)*(p[posmin].y-p[i].y))))-EPS > PI/2)
            i = posmin;
        else if (acos(((p[i].x-p[j].x)*(p[posmin].x-p[j].x)+(p[i].y-p[j].y)*
                    (p[j].y-p[i].y)) / (sqrt((p[i].x-p[j].x)*(p[i].y-p[j].y))*sqrt((p[posmin].x-p[i].x)*(p[i].y-p[j].y))))-EPS > PI/2)
            j = posmin;
        else { find_circle(&p[i], &p[j], &p[posmin]); return; }
    }
}

// distancia esferica
double spherical_dist(double p_lat, double p_long, double q_lat, double q_long){
    return acos(sin(p_lat)*sin(q_lat)+cos(p_lat)*cos(q_lat)*cos(p_long-q_long))*6378
}

int cmpd(double a, double b) {
    struct Point {
        double x, y, z;
        Point(double a=0.0, double b=0.0, double c=0.0) { x=a, y=b, z=c; }
        Point operator+(const Point &p) const { return Point(x+p.x, y+p.y, z+p.z); }
        Point operator-(const Point &p) const { return Point(x-p.x, y-p.y, z-p.z); }
        Point operator*(double c) const { return Point(x*c, y*c, z*c); }
        double operator/() const { return sqrt(x*x+y*y+z*z); }
    };
    double dot(Point A, Point B) { return A.x*B.x + A.y*B.y + A.z*B.z; }
    Point cross(Point A, Point B) { return Point(A.y*B.z-B.y, A.z*B.x-A.x*B.z, A.x*B.y-A.y*B.x); }
    Point project(Point W, Point V) { return V * dot(W,V) / dot(V,V); }
    // do the segments A-B and C-D intersect? (assumes coplanar)
    bool seg_intersect(Point A, Point B, Point C, Point D) {
        return cmpd(dot(cross(A-B,C-B), cross(A-B,D-B))) <= 0 &&
            cmpd(dot(cross(C-D,A-D), cross(C-D,B-D))) <= 0;
    }
    double dist_point_seg(Point P, Point A, Point B) {
        Point PP = A + project(P-A,B-A);
        if ((cmpd(!A-PP)!PP-B,!((A-B)) == 0) return !(P-PP)); // distance point-line!
        return min(!(P-A), !(P-B));
    }
}

```

```

    // segment-segment distance (lines too!)
    double dist_seg_seg(Point A, Point B, Point C, Point D) {
        Point E = project(A-D, cross(B-A, D-C)); // distance between lines!
        if (seg_intersect(A, B, C+E, D+E)) return !E;
        return min( min( dist_point_seg(A,C,D), dist_point_seg(B,C,D) ), min( dist_point_seg(C,A,B), dist_point_seg(D,A,B) ) );
    }

    // point-triangle distance. dps = dist_point_seg
    double dist_point_tri(Point P, Point A, Point B, Point C) {
        Point N = cross(A-C, B-C);
        Point PP = P + project(C-P, N);
        Point V1 = cross(PP-A, B-A);
        Point V2 = cross(PP-B, C-B);
        Point V3 = cross(PP-C, A-C);
        if (cmpD(dot(V1,V2)) >= 0 && cmpD(dot(V1,V3)) >= 0 && cmpD(dot(V2,V3)) >= 0)
            return !(PP-P); // distance Point-plane!
        return min(dps(P,A,B),min(dps(P,A,C),dps(P,B,C)));
    }

    double dist_tet_tet(Point T1[4], Point T2[4]) {
        double ans = INF;
        for (int i=0; i < 4; i++) // arestas -> arestas
            for (int j=i+1; j < 4; j++)
                for (int ii=0; ii < 4; ii++)
                    for (int jj=ii+1; jj < 4; jj++)
                        ans = min( ans, dist_point_tri(T1[i],T1[j],T2[ii],T2[jj]) );
        // pontos -> planos
        for (int i=0; i < 4; i++)
            for (int j=i+1; j < 4; j++)
                for (int k=j+1; k < 4; k++)
                    for (int x=0; x < 4; x++)
                        ans = min( ans, dist_point_seg(T1[x],T2[i],T2[j],T2[k]) );
        return ans;
    }

    double volume(Point T[4]) {return fabs(dot(T[3],cross(T[1]-T[0],T[2]-T[0]))/6.);}
}

// ***** CLASSES DE NÚMEROS *****
const int DIG = 4;
const int BASE = 10000; // BASE**3 < 2**51
const int TAM = 2048;

struct bigint {
    int VTAM[], n;
    bigint(int x = 0): n(1) { memset(v, 0, sizeof(v)); v[n++] = x; fix(); }
    bigint(char *s): n(1) { memset(v, 0, sizeof(v)); }
    int sign = 1;
    char *t = strdup(s), *p = t + strlen(t);
    *p = 0; p = max(t, p - DIG);
    sscanf(p, "%6d", &v[n]);
    v[n+] *= sign;
}
free(t); fix();
bigint& fix(int m = 0) {
    n = max(m, n);
    int sign = 0;
    for (int i = 1; e = 0; i <= n || e && (n = i); i++)
        if (v[i] += e; e = v[i] / BASE; v[i] %= BASE;
            if (v[i]) sign = (v[i] > 0) ? 1 : -1;
    for (int i = 1; i < n; i++)
}

```

```

    if (v[i] * sign < 0) { v[i] += sign * BASE; v[i+1] -= sign; }
    while (n && !v[n]) n--;
    return *this;
}

int cmp(const bigint& x = 0) const {
    int i = max(n, x.n); t = 0;
    while (1) if ('('t = ::cmp(v[i], x.v[i])) || i-- == 0) return t;
}
bool operator <(const bigint& x) const { return cmp(x) < 0; }
bool operator ==(const bigint& x) const { return cmp(x) == 0; }
bool operator !=(const bigint& x) const { return cmp(x) != 0; }

operator String() const {
    ostringstream stream s; s << v[n];
    for (int i = n-1; i > 0; i--) { s.width(DIG); s << abs(v[i]); }
    return s.str();
}

friend ostream& operator << (ostream& o, const bigint& x){ return o << (string)x; }

bigint& operator +=(const bigint& x) {
    for (int i = 1; i <= x.n; i++) v[i] += x.v[i];
    return fix(x.n);
}

bigint operator +(const bigint& x) { return bigint(*this) += x; }

bigint& operator -=(const bigint& x) {
    for (int i = 1; i <= x.n; i++) v[i] -= x.v[i];
    return fix(x.n);
}

bigint operator -(const bigint& x) { return bigint(*this) -= x; }

void operator_-(const bigint r = 0, bigint x, int m, int b) {
    for (int i = 1, e = 0; (i <= x.n || e) && (n = i + b); i++) {
        v[i+b] += x.v[i] * m + e; e = v[i+b] / BASE; v[i+b] %= BASE;
    }
}

bigint operator *(const bigint& x) const {
    bigint r;
    for (int i = 1; i <= n; i++) r.ams(x, v[i], i-1);
    return r;
}

bigint& operator *=(const bigint& x) { return *this = *this * x; }

// cmp(x / y) == cmp(x) * cmp(y); cmp(x % y) == cmp(x);
bigint divisor(const bigint& x) {
    if (x == 0) return 0;
    bigint q; q.n = max(n - x.n + 1, 0);
    int d = x.v[x.n] * BASE + x.v[x.n-1];
    for (int i = q.n, i > 0; i--) {
        int j = x.n + i - 1;
        q.v[i] = int((v[j] * double(BASE) + v[j-1]) / d);
        if (i == 1 || j == 1) break;
        v[j-1] += BASE * v[j]; v[j] = 0;
    }
    fix(x.n); return q;
}

fix(x.n); return q;
}

bigint& operator /=(const bigint& x) { return *this = div(x); }
bigint& operator %=(const bigint& x) { div(x); return *this; }
bigint operator /=(const bigint& x) { return bigint(*this).div(x); }
bigint operator %(const bigint& x) { return bigint(*this) %= x; }

bigint pow(int x) {
    if (x < 0) return (*this == 1 || *this == -1) ? pow(-x) : 0;
    bigint r = 1;
    for (int i = 0; i < x; i++) r *= *this;
    return r;
}

bigint root(int x) {

```

```

if ( cmp( ) == 0 || cmp( ) < 0 && x % 2 == 0 ) return 0;
if (*this == 1 || x == 1) return *this;
if ( cmp( ) < 0 ) return -(~*this).root(x);
bigint a = 1, d = *this;
while (d != 1) {
    bigint b = a + (d / 2);
    if (cmp(b.pow(x)) >= 0) { d += 1; a = b; }
}
return a;
}

friend bigint gcd(bigint x, bigint y){return (y.compare() != 0 ? gcd(y,x%y) : x);}

// fracoes
typedef long long number;
number gcd(number a, number b) { return b ? gcd(b,a%b) : a; }

struct Frac {
    number n, d; bool ok;
    Frac(number a, number b, bool v = true) {
        ok = v && b;
        if (ok) { n = a/gcd(a,b), d = b/gcd(a,b); if (d < 0) n *= -1, d *= -1; }
        Frac operator+(const Frac &f) const {return Frac(n*f.d+f.n*d,d*f.d,ok&&f.ok);}
        Frac operator-(const Frac &f) const {return Frac(n*f.d-f.n*d,d*f.d,ok&&f.ok);}
        Frac operator* (const Frac &f) const {return Frac(n*f.n,d*f.d,ok && f.ok);}
        Frac operator/ (const Frac &f) const {return Frac(n*f.d,d*f.n,ok && f.ok);}
        void print() const { if (!ok) puts("INVALID"); else printf("%lld%lld\n",n,d); }
    };
};

Frac eval(char s[], int from, int to) {
    for (int k=0; k < 3; k++) for (int i=to,d=0; i >= from; i--) {
        if (s[i] == '+') d++; else if (s[i] == '-') d--;
        if (d > 0) continue; assert(d == 0);
        if (k == 0 && s[i] == '+') return eval(s,from,i-1) + eval(s,i+1,to);
        if (k == 0 && s[i] == '-') return eval(s,from,i-1) - eval(s,i+1,to);
        if (k == 1 && s[i] == '*') return eval(s,from,i-1) * eval(s,i+1,to);
        if (k == 2 && s[i] == '/') return eval(s,from,i-1) / eval(s,i+1,to);
    }
    if (s[from] == '(' && s[to] == ')') return eval(s,from+1,to-1);
    number n = 0;
    while (from <= to) {
        assert(s[from] >= '0' && s[from] <= '9');
        n = n*10 + s[from++]- '0';
    }
    return Frac(n,1);
}

// **** DP **** // LIS O(n log n)
// Longest Increasing Subsequence - LIS O(n log n)
const vector<int> lis(const vector<int> &v, vector<int> &aw ) {
    vector<int> pd(v.size(),0), pd.index(v.size()), pred(v.size());
    int maxi = 0, x, j, ind;
    for(i,v.size()) {
        x = v[i];
        j = lower_bound(pd.begin(), pd.begin() + maxi, x) - pd.begin();
        pd[j] = x; pd_index[j] = i; }
    if(j == maxi) { maxi++; ind = i; }
    else { pd[ind] = j ? pd_index[j-1] : -1;
    int pos = maxi-1, k = v.size();
    aw.resize(maxi);
    aw.resize(maxi+1); k = v.size();
    while (pos >= 0) {
        asw[pos--] = k;
        ind = pred[ind];
        k = v[ind];
    }
}

```

```

// LIS 2D - O(n log n log m), em que m é o tamanho da resposta
struct point {
    int x, y;
    bool operator< (const point& o) const { if(x!=o.x) return x<o.x; return y<o.y; }
    bool is_dominated( const point & o ) { return o.x <= x && o.y <= y; }
};

const int MAXN = 100010;
int list2d() {
    int tam = 1, cnt_pnts, cnt_levels = 0; point p, p2;
    set<point> S; set<point> level[MAXN]; set<point> aux[MAXN];
    int n;
    point points[MAXN]; set<point> point >; iterator it;
    level[0].insert(points[0]);
    for(i,1,n-1) {
        p = points[i]; // proximo ponto
        // determina nivel do prox. ponto: o proximo ponto sera colocado no maior
        // nivel i tal que exista algum ponto no nivel i-1 que tenha coordenadas x
        // e Y menores ou iguais as coordenadas do novo ponto: busca binaria
        int lo = 0, hi = tam, lev = 0;
        while ( lo < hi ) {
            int m = lo+(hi-lo)/2;
            // p+ saber se no nivel m existe algum pto que domina o ponto p, testar
            // a dominancia do ponto imediatamente anterior ao primeiro ponto >= p
            it = level[m].lower_bound( p );
            if ( it != level[m].begin() ) --it;
            if ( p.is_dominated( *it ) ) { lo = m+1; lev = m+1; } else hi = m;
        }
        if ( lev == tam ) { // cria-se novo nivel
            level[cnt_levels].clear(); level[cnt_levels++].insert(p); tam++;
        }
        else { // remove pts r de level[lev] tais que p.x <= r.x e p.y <= r.y.
            // Assim, em qualquer instante, os pontos de level[lev] terao
            // coordenadas x crescentes e coordenadas y decrescentes
            cnt.pnts = 0;
            for (it=level[lev].lower_bound(p); it!=level[lev].end(); ++it) {
                if ( p.y <= it->y ) aux[cnt_pnts++] = it;
                else break; // ja que a coordenada y esta diminuindo..
            }
            ford(j,cnt_pnts-1,0) level[lev].erase( aux[j] );
            level[lev].insert(p);
        }
    }
    return cnt_levels; // o numero de niveis eh igual ao tamanho da LIS
}

// Longest Common Subsequence: LCS O(nm) tempo e espaco
typedef pair<int, int> ii;
string lcs( vector<string> s1, vector<string> s2 ) {
    int M = s1.size(), N = s2.size();
    vector< vector<int> > m(M+1, vector<int>(N+1, 0));
    vector< vector<int> > P(M+1, vector<ii>(N+1, ii()));
    ford(i,M-1,0) ford(j,N-1,0) {
        if(s1[i] == s2[j]) {m[i][j] = 1 + m[i+1][j+1]; p[i][j] = make_pair(i, j);
        else {
            if (m[i][j+1] > m[i+1][j]) {m[i][j] = m[i][j+1]; p[i][j] = make_pair(i, j+1);
            else { m[i][j] = s2[j]; m[i+1][j] = m[i+1][j]; p[i][j] = make_pair(i+1, j); }
        }
        string asw = "";
        int len = m[0][0], i = 0, j = 0, b1, b2;
        while ( len ) {
            if (p[i][j]==make_pair(1,1)) {asw+=s1[i]; len--; if (len != 0) asw+=" ";}
            b1 = p[i][j].first; b2 = p[i][j].second;
            i += b1; j += b2;
        }
    }
}

```

```

    return awi;
}

// Maximum Sum 2D: O(n^3)
int maxSum2D( const vector< vector< int> >& M ) {
    int N = M.sz;
    // s[i][j][k] = soma dos nros. da k-esima coluna entre as linhas [i,j] > (N, vector< int>(N, 0)));
    vector<vector<vector<int>> s(N, vector< vector< int> >(N, vector< int>(N, 0)));
    fori(k,N) fori(i,N) forr(j,N-1) s[i][j][k] = M[i][k];
    if ( i == j ) s[i][j][k] = M[i][k] + M[j][k];
    else s[i][j][k] = s[i][j-1][k] + M[j][k];
    int MAX = -INF; fori(i,N) fori(j,N) MAX = max( MAX, maxSum1D( s[i][j] ) );
    return MAX;
}

// Knapsack - O(nW) - conflito: estabelecer ordem de precedencia entre itens.
// pred[i] = ind. do ult. item que pode ser colocado na mochila antes do item i
vector<int> knapsack( const vector<int>& w, const vector<int>& u, int W
    /* const vector<int>& pred */ ) {
    int N = w.sz, aux;
    else { // caso 2: item i cabe na mochila de capacidade j
        aux = M[i-1][j-w[i-1]] + u[i-1];
        // aux = M[i-1][j-w[i-1]] + u[i-1];
        // => se quiser maximizar a quantidade de itens na mochila
        if( aux > M[i-1][j] ) { M[i][j] = aux; s[i][j] = 1; }
        else M[i][j] = M[i-1][j];
    }

    // MIN[W] - utilidade maxima da mochila
    vector<int> sol; // subconjunto de itens que devem ser colocados na mochila
    fordi(i,N,1) {
        if( s[i][j] == 1 ) {
            sol.pb(w[i-1]); j -= w[i-1];
            // i = pred[i-1]+1; // adicionar se for mochila com conflito
        }
    }
    reverse(all(sol));
    return sol;
}

// intearia (com repeticao) - O(nW)
int knapsack( const vector<int>& w, const vector<int>& u, int W ) {
    // M[i] - maxima utilidade de uma mochila de capacidade maxima i
    int N = w.sz, aux;
    vector<int> M( W+1, 0 );
    forr(i,1,W) {
        aux = -INF;
        if( w[j] <= i && (u[j] + M[i-w[j]]) > aux) aux=u[j] + M[i-w[j]];
        M[i] = max( aux, M[i-1] );
    }
    return M[W];
}

// TSP: dado um subconjunto S dos vertices com 0 \in S, seja C(S,j) o menor
// percurso começando em 0 que visita todos os vertices de S e termina em j.
// Se |S| = 2, entao C(S,j) = d(0)[j], \forall all j = 2, ..., n. Se |S| > 2, entao
// C(S,j) = min{ k \neq j } ( C(S\{j\},k) + d[k][j] ) = custo minimo de
// visitar os vertices do estado i começando em 0 e terminando em j - O(2^n n^2)
int tsp( const vector< vector< int> >& dp, const vector< int> >& g ) {
    int n = g.sz, x, current_cost;
    vector< vector< int> > dp( 1 << n, vector< int> ( n, INF ) );
    fori(j,n) dp[0][j] = 0, dp[1]<j>j[j] = g[0][j];
    fori(j,n) dp[0][j] = 0, dp[1]<j>j[j] = g[0][j];
}

```

```

forr(i,1,(1<<n)-1) fori(j,n) if( !( i & (1<<j) ) ) {
    fori(k,n) if( i & (1<<k) ) dp[x][j] = min( dp[x][j], dp[i][k] + g[k][j] );
}
return dp[(1<n)-1][0];

// ***** JOGOS *****
// Dadas N moedas, a cada jogada um jogador pode retirar moves[i] moedas.
// Ganha quem retirar a ultima moeda. (1) Todas as posicoes terminais sao
// perdedoras. (2) Se eh possivel mover a partir da posicao i para uma posicao
// perdedora, entao a posicao i eh vencedora. (3) Se eh possivel mover a partir
// da posicao i apenas para posicoes perdedoras, entao a posicao i eh perdedora.
// conta o numero de posicoes perdedoras do jogador 1 entre 0 e N
int MAX_COINS = 22; // no. maximo de moedas retiradas em uma jogada
int number_of_loosing_positions( int N, vector< int > moves ) {
    // ultimas MAX_COINS posicoes em relacao ao jogador 1
    int mask = ( 1 << MAX_COINS ) - 1; // 1 -> vencedora e 0 -> perdedora
    int res = -1; // numero de posicoes perdedoras do jogador 1 de 0 a N
    vector< int > r; // r[i]=numero de posicoes perdedoras do jogador 1 de 0 ate i
    forr(i,0,N) {
        // assume inicialmente que a posicao i eh perdedora para o jogador 1
        mask = ( ultimo_bit_de mask igual a 0 )
        mask <= 1; ++res;
        tr(moves, i) { // decide se posicao i eh vencedora para o jogador 1
            if( !( mask & ( 1 << *it) ) ) { // eh possivel ir p/ uma posicao perd.?
                ++mask; --res; break; // a posicao i eh vencedora para o jogador 1
            }
        }
    }
    mask &= ( 1 << MAX_COINS)-1; // considera apenas as ult. MAX_COINS posicoes
    if( last.find( mask ) != last.end() ) { // detecta repeticoes
        int cycle_size = i - last[mask]; // tamanho do ciclo
        int num_cycles = ( N - i ) / cycle_size; // ciclos de i ate N
        int losing_positions_in_cycle = res - r[ last[mask] ];
        res += num_cycles * losing_positions_in_cycle;
        i += num_cycles * cycle_size;
    }
    last[mask] = i; // armazena mascara atual
    r.pb(res);
}
return res;

// Jogo de Nim: N pilhas com moedas. A cada jogada um jogador retira uma
// quantidade positiva de moedas de uma das pilhas. Ganha quem retirar moedas
// pela ultima vez. Seja n_1, n_2, ..., n_N o numero de moedas nas N pilhas.
// Esta eh uma posicao perdedora se e somente se n_1 xor ... xor n_N = 0
int is_winning_nim( vector< int > piles ) {
    int res = 0; fori(i,piles.sz) res ^= piles[i]; return res != 0;
}
return res;

// Grundy numbers - cada posicao na matriz representa o menor inteiro nao
// negativo que nao pode ser alcançado a partir daquela posicao
// OBS: Inicializar matrix com -1
const int K = 1, ROWS = 8, COLUMNS = 8;
int matrix[K][ROWS][COLUMNS], NUM_LEGAL_MOVES = 4;
int dx[4] = {-2,-2,-1,+1}, dy[4] = {+1,-1,-2,-2};
#define valid(x,y) (x) >= 0 && (x) < COLUMNS && (y) >= 0 && (y) < ROWS
int Grundy_number( int ind, int x, int y ) {
    if ( matrix[ind][y][x] != -1 ) return matrix[ind][y][x];
    int nx, ny, aux;
    vector< int > alcancavel( NUM_LEGAL_MOVES + 1, 0 );
    fori(i,NUM_LEGAL_MOVES) {
        nx = x + dx[i]; ny = y + dy[i];
        if( valid( nx, ny ) ) alcancavel[Grundy_number( ind, nx, ny )] = 1;
    }
}

```

```

        }
        for(i,1,NUM_LEGAL_MOVES+1) if ( !alcanceavel[i] ) return matrix[ind][y][x] = i;
    return matrix[ind][y][x] = NUM_LEGAL_MOVES + 1;
}

// Dijkstra - O(m log n) - lista de adjacencia. arco: (cost, w)
// Dijkstra pair<int> dijkstra( const vector<vector<int>>& dist ) {
typedef pair<int, int> pair;
void dijkstra( const vector<vector<int>>& g, int v, vector<int>& dist ) {
    int d, cost, w; set<int> Q;
    dist[v] = 0; Q.insert( {v, 0} );
    while( !Q.empty() ) {
        int top = *Q.begin();
        Q.erase( Q.begin() );
        v = top.second; d = top.first;
        for(i,g[v].sz) {
            w = g[v][i].second; cost = g[v][i].first;
            if ( dist[w] != INF ) Q.erase( Q.find( {w, dist[w]} ) );
            dist[w] = dist[v] + cost; Q.insert( {w, dist[w]} );
        }
    }
}

// Bellman Ford - O(mn) - Atualiza dist, retorna se ha ciclo de custo negativo
struct edge { int s, t, w; }; // origem, destino e custo
bool bellman_ford( const vector<edge>& edges, int N, int v, vector<int>& dist ) {
    dist.assign( N, INF ); dist[v] = 0;
    for(i,N) {
        stop = true;
        for(j,m) if( dist[edges[j].s] + edges[j].w < dist[edges[j].t] ) {
            dist[edges[j].t] = dist[edges[j].s] + edges[j].w;
            stop = false;
        }
        if ( stop ) break;
    }
}

// detecta ciclos negativos
for(i,m) if( dist[edges[i].s] + edges[i].w < dist[edges[i].t] ) return true;
return false;
}

// Prim - O(n^2)
double mst_prim( const vector<vector<double>>& g ) {
    int N = g.sz, v = 0; double weight, distance, result = 0;
    vector<double> dist(N, INF); vector<bool> intree(N, false);
    while( !intree[v] ) {
        intree[v] = true;
        for(i,N) if(g[v][i] != INF && !intree[i]) dist[i] = min(dist[i],g[v][i]);
        v = 0; distance = INF;
        for(i,1,N-1) if ( !intree[i] && dist[i] < distance ) {
            distance = dist[i]; v = i;
        }
        if ( distance != INF ) result += distance;
    }
    return result;
}

// Kruskal - O(m log m)
struct edge {
    int s, t; double w; // origem, destino, custo
    bool operator<( const edge& e ) const { return cmp(w, e.w) < 0; }
};

double mst_kruskal( int N, vector<edge> &edges ) {
    int u, v, k; double result = 0; edge e;
    vector<int> pa(N), comp_sz(N, 1);
    ...
}

```

```

for(i,1,N) pa[i] = i;
sort(all(edges));
for ( int i = 0; i < edges.sz && k < N - 1; ++i ) {
    e = edges[i];
    for ( u = e.s; u != pa[u]; u = pa[u] );
    if ( u == e.t, v != pa[v], v = pa[v] );
    if ( comp_sz[u] < comp_sz[v] ) { pa[u] = v; comp_sz[v] += comp_sz[u]; }
    else { pa[v] = u; comp_sz[u] += comp_sz[v]; }
    result += e.w;
}
return result;
}

// Diametro de uma arvore - O(n). o caminho [farthest1, ..., farthest2] forma o
// diametro. central_node eh um vertice central da arvore
const int MAXN = 101;
int dist[MAXN], parent[MAXN], visited[MAXN], tree[MAXN][MAXN], cnt_edges[MAXN];
int n, farthest1, farthest2;
void dfs( int v, int p, int &farthest ) {
    visited[v] = 1; parent[v] = p;
    if ( p != -1 ) dist[v] = dist[p] + 1;
    if ( dist[v] > dist[farthest] ) farthest = v;
    for(i,cnt_edges[v]) if (!visited[tree[v][i]]) dfs(tree[v][i], v, farthest);
}

int calc_diameter() {
    int half_diameter, central_node;
    farthest1 = farthest2 = 0;
    memset(parent,-1,n*sizeof(int)); memset(visited,0,n*sizeof(int));
    dist[0] = 0; dfs(0,-1,farthest1);
    memset(parent,-1,n*sizeof(int)); memset(visited,0,n*sizeof(int));
    dist[farthest1] = 0; dfs(farthest1,-1,farthest2);
    half_diameter = dist[farthest2] / 2; central_node = parent[central_node];
    return central_node;
}

// Componentes fortemente conectados - Tarjan - O(n+m)
// scc = nro. de componentes fortemente conectados do grafo direcionado g.
// components[i] = id do componente ao qual pertence o vertice i
const int MAXN = 3001;
vector<int> g[MAXN], S;
int indices[MAXN], lowlinks[MAXN], component[MAXN], indice, scc;
bool in_stack[MAXN];
void do_tarjan( int v ) {
    indices[v] = indice; lowlinks[v] = indice;
    for(i,g[v].sz) {
        for(i,g[v][i].sz) {
            int w = g[v][i];
            if ( indices[w] == -1 ) {
                do_tarjan(w); lowlinks[v] = min( lowlinks[v], lowlinks[w] );
            }
            else if ( in_stack[w] ) lowlinks[v]=min(lowlinks[v], indices[w]);
        }
    }
    if ( lowlinks[v] == indices[v] ) {
        int w = S[S.sz-1]; S.pop_back();
        while ( w != v ) {
            in_stack[w] = false; component[w] = scc; w = S[S.sz-1]; S.pop_back();
        }
        component[v] = scc++; in_stack[v] = false;
    }
}
void scc_tarjan( int n ) {
    for(i,n) { indices[i] = lowlinks[i] = component[i] = -1; in_stack[i] = 0; }
    for(i,n) if( component[i] == -1 ) do_tarjan(i);
}

```

```

 $\text{// 2-SAT - Cria-se um grafo de implicacao da seguinte forma: Vertices: um para cada variavel e sua negacao. Arestas: p/ uma disjuncao da forma } (x_0 \vee \sim x_3), cria-se duas arestas: (\sim x_0 \rightarrow \sim x_3) e (x_3 \rightarrow x_0). OBS: indexar vertices a partir de 1. Todos os vertices de um mesmo componente fortemente conectado ou sao todos verdadeiros ou todos falsos.$ 
int N;
#define INDEX(i) i > 0 ? i : (2 * N + 1 + i)
while ( cin >> N >> M ) {
    vector< vector<int>> g(2*N+1);
    for(i,1,2*N+1) component[i] = -1;
    while ( M-- ) {
        cin >> u >> v;
        g[INDEX(-u)].pb(INDEX(v));
        g[INDEX(-v)].pb(INDEX(u));
    }
}

scc_tarjan( g, component, cnt_vertex );
if ( component[ INDEX(i) ] == component[ INDEX(-i) ] ) { sat = 0; break; }

cout << sat << endl;
}

Pontes, pontos de articulacao e componentes biconectados
ord_cnt = contador do numero de vertices visitados na arvore de DFS
root_children_cnt = contador do numero de filhos da raiz
ord[v] = ordem em que o vertice v foi visitado na DFS-tree em pre-orden
low[v] = menor numero em preordem do vertice da DFS-tree que pode ser
alcançado por uma sequencia de tree edges e uma back edge
parent[v] = antecessor do vertice v na arvore de DFS
bridges = lista com as pontes, art_pts = booleanos para pontos de articulacao
biconnected_component[i] = arestas do componente biconectado i
edges = pilha com as arestas exploradas (util para descobrir CB)
int ord_cnt, root_children_cnt;
vector< int > ord, low, parent, art_pts;
vector< pair< int, int > > bridges;
stack< pair< int, int > > edges;
void dfs_bridges_art_pts( const vector< vector<int> &g, int v ) {
    ord[v] = ord_cnt++;
    low[v] = ord[v];
    for(i,g[v].sz) {
        edges.push( make_pair(g[v][i], null_edge) );
    }
    int w = g[v][i];
    if ( parent[v] != w ) {
        if ( ord[w] == -1 ) { // (v, w) e uma tree edge
            edges.push( make_pair(v, w) );
            parent[w] = v;
        }
        dfs_bridges_art_pts(g, w);
    }
    if ( parent[v] == v && parent[w] == v ) {
        art_pts_bridges_art_pts(g, w);
        vector< pair< int, int > > bc_edges;
        while ( (e = edges.top()) != null_edge ) {
            if ( (e.first != e.second) ) bc_edges.pb(e);
            edges.pop();
        }
        biconnected_component.pb(bc_edges);
        root_children_cnt++;
        if ( root_children_cnt == 2 ) art_pts[v] = 1;
    }
    else if ( low[w] >= ord[v] ) {
        art_pts[v] = 1;
        vector< pair< int, int > > bc_edges;
        while ( (e = edges.top()) != null_edge ) {
            if ( (e.first != e.second) ) bc_edges.pb(e);
            edges.pop();
        }
        biconnected_component.pb(bc_edges);
    }
}

```

```

edges.pop();
} biconnected_component.pb(bc_edges);
} low[v] = min( low[v], low[w] );
if ( low[w] == ord[w] ) bridges.push_back( make_pair(v, w) );
}

g[INDEX(-u)].pb(INDEX(v));
g[INDEX(-v)].pb(INDEX(u));
}

scc Tarjan( g, component, cnt_vertex );
if ( component[ INDEX(i) ] == component[ INDEX(-i) ] ) { sat = 0; break; }

cout << sat << endl;
}

Pontes, pontos de articulacao e componentes biconectados
ord_cnt = contador do numero de vertices visitados na arvore de DFS
root_children_cnt = contador do numero de filhos da raiz
ord[v] = ordem em que o vertice v foi visitado na DFS-tree em pre-orden
low[v] = menor numero em preordem do vertice da DFS-tree que pode ser
parent[v] = antecessor do vertice v na arvore de DFS
bridges = lista com as pontes, art_pts = booleanos para pontos de articulacao
biconnected_component[i] = arestas do componente biconectado i
edges = pilha com as arestas exploradas (util para descobrir CB)
int ord_cnt, root_children_cnt;
vector< int > ord, low, parent, art_pts;
vector< pair< int, int > > bridges;
stack< pair< int, int > > edges;
void dfs_bridges_art_pts( const vector< vector<int> &g, int v ) {
    ord[v] = ord_cnt++;
    low[v] = ord[v];
    for(i,g[v].sz) {
        edges.push( make_pair(g[v][i], null_edge) );
    }
    int w = g[v][i];
    if ( parent[v] != w ) {
        if ( ord[w] == -1 ) { // (v, w) e uma tree edge
            edges.push( make_pair(v, w) );
            parent[w] = v;
        }
        dfs_bridges_art_pts(g, w);
    }
    if ( parent[v] == v && parent[w] == v ) {
        art_pts_bridges_art_pts(g, w);
        vector< pair< int, int > > bc_edges;
        while ( (e = edges.top()) != null_edge ) {
            if ( (e.first != e.second) ) bc_edges.pb(e);
            edges.pop();
        }
        biconnected_component.pb(bc_edges);
        root_children_cnt++;
        if ( root_children_cnt == 2 ) art_pts[v] = 1;
    }
    else if ( low[w] >= ord[v] ) {
        art_pts[v] = 1;
        vector< pair< int, int > > bc_edges;
        while ( (e = edges.top()) != null_edge ) {
            if ( (e.first != e.second) ) bc_edges.pb(e);
            edges.pop();
        }
        biconnected_component.pb(bc_edges);
    }
}

```

```

void all_bridges_art_points( const vector< vector<int> >& g ) {
    int N = g.size();
    ord.assign(N,-1); art_pts.assign(N,0); low.resize(N); parent.resize(N);
    bridges.clear(); biconnected_component.clear();
    ord_cnt = 0;
    for(i,g.size) if ( ord[i] == -1 ) {
        while ( !edges.empty() ) edges.pop();
        parent[i] = i; root_children_cnt = 0;
        dfs_bridges_art_pts(g, i);
    }
}

// Ordenacao topologica de um DAG - O(m+n)
vector<int> topsort( const vector< vector<int> >& g, vector<int> >& sorted; queue<int> zeroin;
for(i,1,N) if ( indegree[i] == 0 ) zeroin.push(i);
while ( !zeroin.empty() ) {
    v = zeroin.front(); zeroin.pop(); sorted.pb(v);
    for(i,1,N) if ( g[v][i] != INF ) {
        indegree[i]--;
        if ( indegree[i] == 0 ) zeroin.push(i);
    }
}
return sorted;
}

// Circuito Euleriano - O(m+n)
void dfs2( vector< vector<int> >& g, int v, vector<int> >& dfs_path, vector<int> >& indegree ) {
    for(i,g.size) if(g[v][i] != INF) { g[v][i] = INF; dfs2(g, i, dfs_path); }
    dfs_path.pb(v);
}

void print_eulerian_circuit( const vector< vector<int> >& g ) {
    vector< vector< int > > g2 = g; vector< int > dfs_path;
    dfs2(g2, 0, dfs_path);
    for(i,dfs_path.size-1) cout << dfs_path[i] << ' ' << dfs_path[i+1] << endl;
}

// Maxflow - Ford-Fulkerson - Retorna o valor do fluxo maximo
// prev[v] - Pai do vertice v no caminho encontrado do source ao sink
// res[] - copia da capacidade (eh alterada em cada chamada)
// g[i] - lista de adjacencias do vertice i (adicionar aresta nos dois sentidos)
// OBS: memset( cap, 0, sizeof( cap ) ); for(i,1,n) g[i].clear();
const int TAM = 110;
int cap[TAM][TAM], res[TAM][TAM], prev[TAM];
vector<int> g[TAM];
int maxflow( int n, int source, int sink ) {
    int ans = 0, i;
    memset( cap, 0, sizeof( cap ) );
    memcpy( res, cap, sizeof( res ) );
    while( true ) {
        memset( prev, -1, sizeof( prev ) );
        prev[source] = -2;
        queue<int> q; q.push( source );
        while( !q.empty() && prev[sink] == -1 ) {

```

```

        int v = g[u][i];
        if( prev[v] == -1 && res[u][v] > 0 ) { prev[v] = u; q.push(v); }

    } if( prev[sink] == -1 ) break;
    int bot = INF, len;
    for(i = sink; prev[i] >= 0; i = prev[i]) bot = min(bot, res[prev[i]][i]);
    res[prev[i]][i] -= bot; res[i][prev[i]] += bot;
}
ans += bot;

} return ans;

// Vertex Cut ( num. minimo de vertices que se removidos desconectam o grafo )
menset( cap, 0, sizeof( cap ) ); for(i,2*n) g[i].clear();
// cria aresta do vertice original i para a sua copia e da copia para o original
for(i, n) { cap[2*i][2*i+1] = 1; g[2*i].pb(2*i+1); g[2*i+1].pb(2*i); }

for(i,m) {
    cap[2*a+1][2*b] = INF; cap[2*b+1][2*a] = INF;
    g[2*a].pb(2*b+1); g[2*b+1].pb(2*a); g[2*a+1].pb(2*b);
}

int mincut, asw = INF, s = 0;
for(t,1,n-1) { mincut = maxflow(2*n, 2*s+1, 2*t); asw = min(asw, mincut); }
if( ( asw >= INF ) asw = n; // se o corte minimo for INF
// Cobertura minima por caminhos: dado um DAG, determinar o menor numero de
// caminhos disjuntos em vertices que cobrem todos os vertices do digrafo
// Ideia: criar graf bipartido e resolver o matching maximo nesse grafo. Para
// cada arco (a,b) da entrada, criar arco com capacidade 1 no grafo bipartido.
// min_cover_by_direted_paths()
// cria grafo
int asw = maxflow(n, source, sink);

return n - asw;
}

// MaxFlow - Improved Shortest Augmenting Path Algorithm, O(n^2 m)
#define MAX 203
int n, m, source, sink; // numero de vertices, arestas, origem e destino
int G[MAX][MAX], F[MAX][MAX], g[MAX][MAX]; // capacidade, fluxo e grafo
int pi[MAX]; // predecessores
int CurrentNode[MAX]; // aresta atual para cada vertice
int queue[MAX]; // fila para a reverse BFS
int d[MAX]; // distancia
int numsbs[MAX]; // numsbs[k] eh o numero de vertices i com d[i]==k
int rev_BFS() {
    int i, j, head(0), tail(0);
    numsbs[n]--; d[sink] = 0; numsbs[0]++;
    queue[++tail] = sink;
    while( head != tail ) {
        i = queue[++head];
        for(j,n) {
            if( d[j] < n || g[j][i] == 0 ) continue;
            queue[++tail] = j; numsbs[n]--; d[j] = d[i] + 1; numsbs[d[j]]++;
        }
    }
    return 0;
}

int Augment() {
    int i, j, tmp, width(INF);
    for( i = sink, j = pi[i]; i != source; i = j, j = pi[j] ) {
        tmp = g[j][i]; if(tmp < width) width = tmp;
}

```

```

        }
    }

    int Retreat( int &i ) {
        int tmp, j, mind(n-1);
        for( j = sink; j == pi[i]; i != source; i = j, j = pi[j] ) {
            G[j][i] -= width; F[j][i] += width;
            G[i][j] += width; F[i][j] -= width;
        }
        return width;
    }

    int maxflow() {
        int flow(0), i = source, j;
        rev.BFS();
        for( k,n ) CurrentNode[k] = 0;
        while( d[source] < n ) {
            for( j = CurrentNode[i]; j < n; ++j ) if(G[i][j] > 0 && d[i]==d[j]+1) break;
            if( j < n ) {
                CurrentNode[i] = j; pi[j] = i; i = j;
                if( i == sink ) { flow += Augment(); i = source; }
            }
            else {
                CurrentNode[i] = 0;
                if( Retreat(i) == 0 ) break;
            }
        }
        return flow;
    }

    // Min-cost MaxFlow - Edmonds-Karp relabelling + Dijkstra
    // Retorna: o fluxo, o custo na variavel fcost
    // fnet armazena o fluxo da rede. fnet[u][v] e fnet[v][u] podem ser
    // positivos. neste caso, pegar a diferenca.
    // NAO ESQUECER DO CLR(cap, 0); CLR(cost, INF); criar grafo a partir desses dois
    #define NV 105
    #define Pot(u,v) (d[u] + pi[u] - pi[v])
    #define CLR(a,x) memset(a,x, sizeof( a ))
    int cap[NV][NV], cost[NV][NV]; // usados para criar a rede
    int adj[NV][NV], fnet[NV][NV], deg[NV], par[NV], d[NV], pi[NV];
    int minflow( int n, int s, int t, int &fcost );
    CLR( deg, 0 ); CLR( fnet, 0 ); CLR( pi, 0 );
    for( i,n ) for( j,n ) if( cap[i][j] || cap[j][i] ) adj[i][deg[i]++] = j;
    int flow = fcost = 0;
    while( dijkstra( n, s, t ) ) {
        int bot = INF;
        for( int v = t, u = par[v]; v != s; u = par[v = u] )
            bot = min( bot, fnet[v][u] ? fnet[v][u] : ( cap[u][v]-fnet[u][v] ) );
        for( int v = t, u = par[v]; v != s; u = par[v = u] )
            if( fnet[v][u] ) { fnet[v][u] -= bot; fcost -= bot * cost[v][u]; }
        else { fnet[u][v] += bot; fcost += bot * cost[u][v]; }
        flow += bot;
    }
    return flow;
}

// para grafos DENSO
bool dijkstra( int n, int s, int t ) {
    for( i,n ) d[i] = INF, par[i] = -1;
    d[s] = 0; par[s] = -n - 1;
    while( 1 ) {
        int u = -1, bestD = INF;
        for( i,n ) if( par[i] < 0 && d[i] < bestD ) bestD = d[u = i];
        if( bestD == INF ) break;

```

```

par[u] = -par[u] - 1;
for(i, deg[u]) {
    int v = adj[u][i];
    if( par[v] >= 0 ) continue;
    if( fnet[v][u] && d[v] > Pot(u,v) - cost[v][u] ) {
        d[v] = Pot( u, v ) - cost[v][u]; par[v] = -u - 1;
    }
}
for(i,n) if( pil[i] < INF ) pil[i] += d[i];
return par[t] >= 0;
}

// para grafos ESPARSSOS
#define BUBL { \
    t = q[i]; q[i] = q[j]; q[j] = t; \
    t = inq[q[i]]; inq[q[i]] = inq[q[j]]; inq[q[j]] = t; \
}

int q[NV], inq[NV], qs;
bool dijkstra( int n, int s, int t ) {
    CLR( d, 0x3F ); CLR( par, -1 ); CLR( inq, -1 );
    d[s] = qs = 0; inq[q[qs++]] = s = 0; par[s] = n;
    while( qs ) {
        int u = q[0]; inq[u] = -1;
        q[0] = q[-qs];
        if( qs ) inq[q[0]] = 0;
        for( int i = 0, j = 2*i + 1, t; j < qs; i = j, j = 2*i + 1 ) {
            if( j + 1 < qs && d[q[j+1]] < d[q[j]] ) j++;
            if( d[q[j]] >= d[q[i]] ) break;
        }
        BUBL;
        for( int k = 0, v = adj[u][k]; k < deg[u]; v = adj[u][+k] ) {
            if( fnet[v][u] && d[v] > Pot(u,v) - cost[v][u] )
                d[v] = Pot(u,v) - cost[v][par[v] = u];
            if( fnet[u][v] < cap[u][v] && d[v] > Pot(u,v) + cost[u][v] )
                d[v] = Pot(u,v) + cost[par[v] = u][v];
            if( par[v] == u ) {
                if( inq[v] < 0 ) { inq[q[qs]] = v = qs; qs++; }
                for( int i = inq[v], j = (i - 1)/2, t; i = j, j = (i - 1)/2 ) BUBL;
            }
        }
    }
    for(i,n) if( pil[i] < INF ) pil[i] += d[i];
}

// Encontrar K caminhos disjuntos em um DAG que passem pelo maior no. de vert.
// Ideia: criar um source de capacidade K e duplicar cada vertice original.
//           cin >> n >> m;
//           CLR(cap, 0); CLR(cost, INF); CLR(adj, 0);
for(i, m) {
    cin >> a >> b; a--; b--;
    cap[2*a][2*a+1] = 1; cost[2*a][2*a+1] = 0;
    cap[2*b][2*b+1] = 1; cost[2*b][2*b+1] = 0;
    cap[2*a+1][2*b] = 1; cost[2*a+1][2*b] = -1;
}
int pre_src = 2*n, source = 2*n+1, sink = 2*n+2, asw = 0, maxflow;
cap[pre_src][source] = K; cost[pre_src][source] = 0;
for(i, n) {
    cap[i][source][2*i] = 1; cost[source][2*i] = -1;
    cap[2*i+1][sink] = 1; cost[2*i+1][sink] = 0;
}

```

```

maxflow = minflow( 2*n+3, pre_src, sink, asw );
cout << -asw << endl;
// Algoritmo Hungariano - O(n^3): matching maximo em um grafo bipartido valorado
// Na funcao main(): forr(i,1,N) forr(j,1,N) scanf( "%d", &w[i][j] );
// hungarian_algorithm(); int asw = 0; forr(i,1,N) asw += lx[i] + ly[i];
// p/ matching min, inicializar matriz w com custos neg. e imprimir -asw
const int MAXN = 501;
int N; // numero de vertices em cada participo (|X| = |Y| = N)
int lx[MAXN], ly[MAXN], w[MAXN][MAXN]; // lx[i] + ly[j] >= w[i][j]
int mark_T[MAXN], vizinhos_X[MAXN][MAXN], num_vizinhos_X[MAXN];
int saturadox[MAXN], saturadox[MAXN], paix[MAXN], match[MAXN];
vector<int> S, T;
int not_saturated() {
    for(i,1,N) if( saturadox[i] == 0 ) return i;
    return 0;
}

int choose( int *x_pai ) {
    for(i,S.sz) forr(j,num_vizinhos_X[S[i]]) {
        if( !mark_T[vizinhos_X[S[i]][j]] ) {
            *x_pai = S[i];
            return vizinhos_X[S[i]][j];
        }
    }
    return -1;
}

int clean_and_create() {
    CLR( num_vizinhos_X, 0 ); CLR( mark_T, 0 ); CLR( saturadox, 0 );
    CLR( saturodoxy, 0 ); CLR( match, -1 ); CLR( paix, -1 ); CLR( paixy, -1 );
    forr(i,1,N) forr(j,1,N) if( lx[i] + ly[j] == w[i][j] )
        vizinhos_X[i][num_vizinhos_X[i]+] = j;
}

void start() {
    forr(i,1,N) {
        int k = -1;
        forr(j,1,N) if( w[i][j] > k ) k = w[i][j];
        ly[i] = 0; lx[i] = k; // maior valor da linha i
    }
    clean_and_create();
}

void compute() {
    int best = INF;
    forr(i,S.sz) forr(j,1,N) if( !mark_T[j] ) {
        if( lx[S[i]]+ly[j]-w[S[i]][j] < best ) best = lx[S[i]] + ly[j] - w[S[i]][j];
    }
    forr(i,S.sz) lx[S[i]] -= best;
    forr(i,T.sz) ly[T[i]] += best;
    clean_and_create();
}

void hungarian_algorithm() {
    S.clear(); T.clear(); S.push_back(u);
    CLR( mark_T, 0 );
    while(1) {
        // pega vertice em Y (viz. de alguem que esteja em S) q n pertence a T
        y = choose( &x_pai );
        if( y == -1 ) { compute(); break; } // nao encontrou matching perfeito
        else {
            if( saturadox[y] ) {
                S.push_back( match[y] ); T.push_back(Y);
                mark_T[y] = 1; paix[y] = x_pai; paix[y] = match[y] = y;
            }
            else { // caminho de aumento

```

```

// Difference Constraints - Problema: Resolver um sistema linear com
// desigualdades da forma:  $x_i - x_j \leq b_k$ ,  $1 \leq i, j \leq n$ . Solucao: Criar um
// grafo direcionado apropriado (grafo de restricoes) e encontrar o menor
// caminho de um vertice artificial  $v_{\{n+1\}}$  p/ todos os demais vertices.
// 1. Construcao do digrafo  $G$ :
//    1.1.  $x_i - x_j \leq b_k \rightarrow$  arco  $(v_j, v_i)$  com custo  $b_k$ ,  $1 \leq i, j \leq n$ 
//    1.2. introduzir vertice  $v_{\{n+1\}}$  com arco  $(v_{\{n+1\}}, v_{\{i\}})$ ,  $i = 0, 1 \leq i \leq n$ 
// 2. Achar caminho minimo a partir do vertice  $v_{\{n+1\}}$  para todos os outros
//    vertices, por exemplo, com algoritmo de bellman-ford
// 3. Se  $G$  nao tem ciclo com custo negativo, entao  $x_{\{i\}} = dist[i]$ 
//    Se  $G$  tem ciclo com custo negativo, entao o sistema eh inviavel
// Resolvo tambem problemas com restricoes das formas:
// (1)  $x_a + x_{\{a+1\}} + \dots + x_{\{b\}} \leq A_1$ , (2)  $x_a + x_{\{a+1\}} + \dots + x_{\{b\}} \geq A_2$ 
// (3)  $-x_a - x_{\{a+1\}} - \dots - x_{\{b\}} \geq A_3$ , (4)  $-x_a - x_{\{a+1\}} - \dots - x_{\{b\}} \leq A_4$ 
// Seja  $S(k) = x_0 + \dots + x_k$ ;  $x_0$  eh uma variavel artificial tal que  $x_0 = 0$ 
// Com isso,  $x_a + x_{\{a+1\}} + \dots + x_{\{b\}} = S(a-1) - S(b)$ 
//  $-x_a - x_{\{a+1\}} - \dots - x_{\{b\}} = S(a-1) - S(b)$ 
// Portanto, eh possivel colocar (1), (2), (3) e (4) da forma  $s_i - s_j \leq b_k$ .
// Se existe solucao, entao  $S(i) = dist[i] - dist[i-1]$ 
vector<int> dist(n+2, INF); vector<edge> edges; edge e;
vector<int> cin(>> a >> b >> c >> d,
    if (c == "lt") {e.s = a-1; e.t = a+b; e.w = -(d+1);}
    else {e.s = a+b; e.t = a-1; e.w = -(d+1);}
edges.pb(e);
edges.pb(e);

for(i,0,n) { e.s = n+1; e.t = i; e.w = 0; edges.pb(e); }
bool negative_cycle = bellman_ford(edges, n+2, n+1, dist);
if (negative_cycle) printf("-1");
else for(i,1,n) printf("%d", dist[i]-dist[i-1]);
// Min Vertex Cover em arvores - Encontrar o tamanho do menor subconjunto de
// vertices que cobre todos os outros vertices de uma arvore.
const int MAXN = 1510;
int visited[MAXN], p[MAXN], num_child[MAXN], leaves[MAXN], marks[MAXN], f, n;
int min_vertex_cover_in_trees() {
    CLR(visited, 0);
    int k;
    for(i,1,f) if( p[leaves[i]] != -1 && !visited[ p[leaves[i]] ] ) {
        visited[ p[leaves[i]] ] = 1;
        if( p[ p[leaves[i]] ] != -1 ) num_child[ p[ p[leaves[i]] ] ]--;
        if( p[ p[leaves[i]] ] != -1 && !visited[ p[ p[leaves[i]] ] ] &&
            num_child[ p[ p[leaves[i]] ] ] == 0 && !marks[ p[ p[leaves[i]] ] ] ) {
            marks[ p[ p[leaves[i]] ] ] = 1;
            leaves[ f++ ] = p[ p[leaves[i]] ];
        }
    }
    int asw = 0;
    for(i,1,n) if( visited[i] ) asw++;
    return asw;
}

int main() {
    // Entrada: n - numero de vertices
}

```

```

// id:(q) v_1 ... v_q - id do vertice, numero de vizinhos e lista de vizinhos
while( scanf( "%d", &n ) == 1 ) {
    int k, child, tam;
    CLR( p, -1 ); CLR( num_child, 0 ); CLR( marks, 0 );
    f = 0;
    for(i,n) {
        scanf( "%d:(%d)", &k, &tam );
        if( tam == 0 ) { leaves[f++] = k; marks[k] = 1; }
        num_child[k] = tam;
        for(j,tam) { scanf( "%d", &child ); p[child] = k; }
    }
    int asw = min_vertex_cover_in_trees();
    printf( "%dn", asw );
}
return 0;
}

// Permanente da matriz quadrada n x n: perm(M) = (-1)^n * sum_{S \subset S' \subset {1,...,n}} (-1)^{|S|} \prod_{i \in I} a_{ij}
// Aplicações: (1) cycle covers em digrafos valorados - permanente igual a soma dos pesos de todos os pesos cobertos por ciclo do digrafo. (2) perfect matching em grafos bipartidos valorados - permanente igual a soma dos pesos de todos os matchings perfeitos do grafo. Em grafos não valorados, o permanente é igual ao número de cycle covers ou de matchings perfeitos.
// Complexidade: O(2^n * n) - usando código gray.
int graycode[1 < MAXN]; // Gera o código gray
oid gen_gray_code( int nbits ) {
    int toggled = 0; graycode[0] = 0;
    for( int n = 1; n < (1 << nbits); ++n ) {
        toggled ^= n & ~(n-1);
        graycode[n] = toggled;
    }
}

const int MAXN = 20;
// line_sum[i]: soma dos elementos da i-esima linha da matrix
int N, matrix[MAXN][MAXN], line_sum[MAXN];
long long calc_permanent() {
    int set, last_set, ub = 1 << N, tam, diff, index;
    long product, sum, permanent = 0;
    for(i,MAXN) line_sum[i] = 0;
    for(j,1,ub-1) {
        last_set = graycode[j-1]; set = graycode[j]; tam = __builtin_popcount(set);
        diff = last_set ^ last_set; // diferença entre set e last_set
        index = __builtin_ctz( diff ); // indice do bit alterado
        product = 1;
        for(i,N) {
            sum = line_sum[i];
            if( ( set & ( 1 << index ) ) sum += matrix[i][index];
            else sum -= matrix[i][index];
            line_sum[i] = sum; product *= sum;
        }
        permanent += ( tam % 2 ) ? -product : product;
    }
    return ( N % 2 ) ? -permanent : permanent;
}

// Stable marriage problem
int men[MAX][MAX], women[MAX];
int order[MAX][MAX];
oid mry(int men[], int women[], int n, vector<int> &sans, vector<int> &sans_rev) {
    vector<int> proposals(n,0);
    stack<int> singles;
    int i, j, k;
    for( i=0; i < n; i++ ) for( j=0; j < n; j++ ) order[i][women[i][j]] = j;
    ans = vector<int>(n,-1), ans_rev = vector<int>(n,-1);
    for( i=0; i < n; i++ ) singles.push(i);
}

```

```

while (!singles.empty()) {
    singles.top(); j=men[i][proposals[i]++]; k=ans_rev[j];
    if (k == -1) ans[i] = j; ans_rev[j] = i;
    else if (order[j][i] < order[j][k]) {
        ans[k] = -1; ans[i] = j; ans_rev[j] = i; singles.push(k);
    } else singles.push(i);
}
// Matching em grafo bipartido -> Inicializar nl e nr.
int nr, nl, mate[MAX];
char visited[MAX];
int dfs(int i){
    int j; if (visited[i]) return 0; visited[i] = 1;
    for (j=0; j < nr; j++) {
        if (g[i][j] && (mate[j] == -1 || dfs(mate[j]))) {mate[j] = i; return 1;}
    }
    return 0;
}
int match(){
    int i, ans = 0;
    memset(mate,-1,sizeof(mate));
    for (i=0; i < nl; i++) memset(visited,0,sizeof(visited)), ans += dfs(i);
    return ans;
}
// Min-cut entre quaisquer vertices
int stoer_wagner(Graph g, int n) {
    int order[MAX], used[MAX], merged[MAX], i, j, k, p, cut, ans = INF;
    Graph aux;
    memset(merged,0,sizeof(merged));
    for (p=1; p < n; p++) {
        memset(used,0,sizeof(used));
        memcpy(aux,g,sizeof(Graph));
        for (k=1; k < n; k++) {
            for (i=-1; j=1; j < n; j++) {
                if (!used[j] && !merged[j] && (i == -1 || aux[0][j] > aux[0][i])) {
                    i = j;
                }
            }
            if (i == -1) break;
        }
        used[order[k-2]] = i;
        for (j=1; j < n; j++) {
            aux[0][j] += aux[i][j]; aux[j][0] += aux[i][j];
            aux[i][j] = aux[j][i] = 0;
        }
    }
    for (i=cut=0; i < n; i++) {
        if (order[i-2] != i) {
            g[order[i-2]][i] += g[order[k-1]][i];
        }
        cut += g[order[k-1]][i];
        g[order[k-1]][i] = g[i][order[k-1]] = 0;
    }
    merged[order[k-1]] = 1;
    if (cut < ans) ans = cut;
}
return ans;
}
// Matching em grafos quaisquer - Edmonds
// IMPORTANTE: na main inicializar n e graph (1-based)
const int MAXN = 256;
int n, start, finish, new_base, match[MAXN], father[MAXN], base[MAXN];
int graph[MAXN][MAXN], in_queue[MAXN], in_path[MAXN], in_blossom[MAXN];
queue<int> q;

```

```

int find_common_ancestor(int u, int v) {
    memset(in_path, 0, sizeof(in_path));
    while (1) {u=base[u]; in_path[u]=1; if (u==start) break; u=father[match[u]]; }
    if (base[v] != u) {v=base[v]; if (in_path[v]) break; v = father[match[v]]; }
    return v;
}
void reset_trace(int u) {
    while (base[u] != new_base) {
        int v = match[u]; in_blossom[base[u]] = 1; in_blossom[base[v]] = 1;
        u = father[v];
        if (base[u] != new_base) father[u] = v;
    }
}
void contract(int u, int v) {
    new_base = find_common_ancestor(u, v);
    memset(in_blossom, 0, sizeof(in_blossom));
    reset_trace(u); reset_trace(v);
    if (base[u] != new_base) father[u] = v;
    if (base[v] != new_base) father[v] = u;
    for (i,1,n) if (in_blossom[base[i]]) {
        base[i] = new_base; if (!in_queue[i]) q.push(i);
    }
}
void find_augmenting_path() {
    memset(in_queue, 0, sizeof(in_queue));
    memset(father, 0, sizeof(father));
    for (i,1,n) base[i] = i;
    finish = 0;
    while (!q.empty()) q.pop(); q.push(start); in_queue[start] = 1;
    while ( !q.empty() ) {
        int u = q.front(); q.pop();
        for (v,1,n) if ((graph[u][v]) && (base[u] != base[v]) && (match[u] != v)) {
            if ((v==start) | | (match[v]>0) && (father[match[v]]>0)) contract(u,v);
            else if (father[v] == 0) {
                father[v] = u;
                if (match[v] > 0) q.push(match[v]);
                if (match[v] > 0) match[v] = u;
            }
            else { finish = v; return; }
        }
    }
}
void augment_path() {
    int u = finish, v, w;
    while (u > 0){v=father[u]; w = match[v]; match[v] = u; match[u] = v; u = w;}
}
int edmonds(){
    memset(match,0,sizeof(match));
    for (i,1,n) if (match[i] == 0) {
        start = i; find_augmenting_path();
        if (finish > 0) augment_path();
    }
    int r = 0;
    for (i,1,n) if (match[i] > 0) r++;
    return r / 2;
}
void print() { for (i,1,n) if ( i < match[i] ) printf(" %d %d\n", i, match[i]); }
// ***** ***** ***** ***** TEORIA DOS NÚMEROS *****
// mdc(x,Y)
int gcd(int x, int Y) { return Y ? gcd(Y, x % Y) : abs(x); }
// mmc(x,Y) - mmc(x,Y) = (x / mdc(x,Y)) * Y
long long lcm(int x, int Y) {
    if (x && Y) return abs(x) / gcd(x, Y) * (long long) abs(Y);
    else return (long long) abs(x | Y);
}

```

```

}
// Determina x e y tais que ax + by == gcd(a, b)
typedef Pair<int, int> bezout;
beout find_bezout(int a, int b) {
    if (b == 0) return bezout(1, 0);
    bezout u = find_bezout(b, a % b);
    return bezout(u.second, u.first - (a/b) * u.second);
}

// Determina x e y tais que ax + by == c. (x, y) eh uma solucao particular para
// o problema. As demais solucoes sao da forma: (x + bk, y - ak), para todo
// inteiro k, positivo ou negativo. Retorna (INF, INF) se nao tiver solucao.
int d = gcd(a, b);
if (c % d) return bezout(INF, INF);
int new_c = c / d;
bezout asw = find_bezout(a / d, b / d);
asw.first *= new_c; asw.second *= new_c;
return asw;

// Determina x tal que a * x congruente 1 (mod m) pelo teorema de Fermat
long long inv_mult( long long n, long long mod ) {
    return mod_exp( n, mod-2, mod );
}

// Acha a menor solucao nao-negativa de a * x = b (mod m)
int mod(int x, int m) { return (x % m) + ( (x < 0) ? m : 0 ); }

int solve_mod(int a, int b, int m) {
    if (m < 0) return solve_mod(a, b, -m);
    if (a < 0 || a >= m || b < 0 || b >= m)
        return solve_mod(mod(a, m), mod(b, m), m);
    bezout t = find_bezout(a, m);
    int d = t.first * a + t.second * m;
    if (b % d) return -1;
    else return mod(t.first * (b / d), m / gcd(a, m));
}

// while ( . . . ) x += m / gcd(a, m); // outras solucoes da equacao

// Calcula o valor de x em a * x = b mod c. Nao eh necessario que c seja primo.
// Testar a condicao a condicao para saber se a equacao tem solucao:
// if ( (a*x) % c + c ) % c != ( b % c + c ) % c
// outras solucoes: while ( . . . ) x += c / gcd(a, c);
long long solve_mod(long long a, long long b, long long c) {
    return a ? (solve_mod(c % a, (a - b % a) % a, a) * c + b) / a : 0;
}

// C(n,k) modulo m. Necessario que gcd(fat[n-k], m) = gcd(fat[k], m) = 1
typedef long long int lli;
int factorial[MAXN+1]; // pre-calcular factorial[i] mod m
int calc_combination( int n, int k, int m ) {
    int a = factorial[n];
    int b = inv_mult(factorial[n-k], m), c = inv_mult(factorial[k], m);
    return ( ( (lli)a * (lli)b ) % m ) * (lli)c ) % m;
}

// C(n,k) modulo m
long long binom( int n, int k, int m ) {
    int mdc, num[k], den[k]; long long resp = 1;
    for(i, k) { num[i] = n - i; den[i] = i + 1; }
    mdc = gcd( den[i], num[j] );
    den[i] /= mdc; num[j] /= mdc;
    for(i, k) { resp *= num[i]; resp %= m; }
    return resp;
}

// b^e % m
long long mod_exp( int b, int e, int m ) {

```

```

long long res = 1;
while ( e > 0 ) {
    if ( e & 1 ) res = (res * b) % m; e >>= 1; b = ( (long long) b * b ) % m;
}
return res;

}

// Calcula o menor valor de e na expressao b^e = n % p
// Baby-step Giant-step algorithm. Retorna -1 se eh impossivel.
long long discrete_logarithm( long long b, long long n, long long p ) {
    if ( n == 1 ) return 0;
    map < long long, int > table;
    long long m = sqrt(p) + 1, pot = 1, pot2 = 1;
    for(j,m) {
        if ( pot == n ) return j;
        table[ (n * inv_mult( pot, p ) ) % p] = j;
        pot = ( pot * b ) % p;
    }
    for(i,m) {
        if ( table.find( pot2 ) != table.end() ) return i * m + table[pot2];
        pot2 = ( pot * pot2 ) % p;
    }
    return -1;
}

// Calcula o Legendre symbol (a/p) - p eh um primo impar
// (a/p) = 0 se a = 0 (mod p)
// (a/p) = +1 se a \neq 0 (mod p) e existe x, tal que x^2 = a (mod p)
// (a/p) = -1 se a \neq 0 (mod p) e nao existe x, tal que x^2 = a (mod p)
int legendre_symbol( long long a, long long p ) {
    long long signal = 1, asw;
    if ( a < 0 ) { signal = ( p % 4 == 1 ) ? 1 : -1; a = -a; }
    a %= p; asw = mod_exp(a,(p-1)/2,p);
    if ( asw == p-1 ) asw = -1;
    return signal * asw;
}

// Retorna a factoracao em numero primos de abs(n)
typedef map<int, int> prime_map;
void divide(prime_map & M, int & n, int p) { while(n%p==0) { M[p]++;
    n /= p; } }

prime_map factoring(int n) {
    prime_map M;
    if (n < 0) return factoring(-n);
    if (n < 2) return M;
    divide(M, n, 2);
    divide(M, n, 3);
    int maxP = (int) sqrt(n) + 2;
    for ( int p = 5; p < maxP; p += 6 ) { divide(M, n, p); divide(M, n, p+2); }
    if (n > 1) M[n]++;
    return M;
}

// Decide se o inteiro n eh primo
bool is_prime(int n) {
    if (n < 0) return is_prime(-n);
    if (n < 5 || n % 2 == 0 || n % 3 == 0) return (n == 2 || n == 3);
    int maxP = sqrt(n) + 2;
    for (int p = 5; p < maxP; p += 6)
        if (n % p == 0 || n % (p+2) == 0) return false;
    return true;
}

// mdc(x!,y)
int fact_gcd( int x, int y) {
    if ( y <= x ) return y;
    int asw = 1;
    prime_map M = factoring(y);
    for ( prime_map::iterator it = M.begin(); it != M.end(); ++it ) {
        int p = it->first, n = x;

```

UFMG – Universidade Federal de Minas Gerais

```

int d1 = it->second; // d1 = qtas vezes p aparece como fator de y
int d2 = 0; // d2 = qtas vezes p aparece como fator de x!
while( n > 0 ) d2 += (n /= p);
for ( int i = 0; i < min(d1, d2); i++ ) asw *= p;
}
return asw;

// Retorna os divisores do inteiro n (divisores pode conter numeros repetidos)
vector<int> divisores( int n ) {
    int maxP = (int) sqrt(n) + 2;
    vector<int> divisores;
    for(i=1,maxP) if ( n % i == 0 ) { divisores.pb( i ); divisores.pb( n/i ); }
    return divisores;
}

// Retorna o nro. de divisores de cada inteiro no intervalo [l..abs(maxv)]
vector<int> number_of_divisors( int maxn ) {
    for(i=1,maxn) for ( int k = i; k <= maxn; k += i ) v[k]++;
}
return v;

// Crivo otimizado
const int MAXN = 10000001, SIZE = MAXN/16+1;
const int MAX_PRIMES = 685000; // 1.26 * MAXN / log(MAXN);
char mark[SIZE]; // ( mark[n>>4&(1<<(n>>1)&7) ) == 0 se 2*n+1 eh primo
char is_prime[MAXN];
int primes[MAX_PRIMES], cnt_primes;
void sieve( int N ) {
    int i, j;
    for ( j = 3; i*i <= N; i += 2 ) if ( ( mark[i>>4] & ( 1<<((i>>1)&7) ) ) == 0 ) {
        cnt_primes = 0;
        for ( i = 3; i <= N; i += 2 ) if ( ( mark[i>>4] & ( 1<<((j>>1)&7) ) ) == 0 )
            primes[cnt_primes++] = 2;
        primes[cnt_primes++] = i;
        for(i,i,TAM){ c[i][0] = c[i][i] = 1;
            for(j,1,i-1){ c[i][j] = c[i-1][j-1] + c[i-1][j];
        }
    }
}

// Triangulo de Pascal
const int TAM = 50;
long long C[TAM][TAM];
void calc_pascal(){
    memset(C, 0, sizeof( C ) );
    for(i,1,TAM){ c[i][0] = c[i][i] = 1;
        for(j,1,i-1){ c[i][j] = C[i-1][j-1] + C[i-1][j];
    }
}

// Teste de primalidade com Miller-Rabin Pollard-Rho
typedef long long int Int;
#define MAX ( (Int) 1 << 63 ) -1
#define gcd 10007
Int p[10] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29};
Int Gcd(Int x, Int y) { return y ? Gcd(y, x % y) : x; }
inline Int produc_mod(Int a, Int b, Int mod) {
    Int sum = 0;
    while(b) { if(b & 1) sum = (sum + a) % mod; a = (a + a) % mod; b /= 2; }
    return sum;
}

inline Int power_mod(Int a, Int b, Int mod) {
    Int sum = 1;
    while(b) {
        if(b & 1) sum = produc_mod(sum, a, mod);
        a = produc_mod(a, a, mod); b /= 2;
    }
}

```

UFMG – Universidade Federal de Minas Gerais

```

    return sum;
}
bool rabin_miller(Int n) {
    int i, j, k = 0, Int u, m, buf;
    if(n < 2 || !(n & 1)) return false;
    m = n-1;
    while(! (m & 1)) { k++; m /= 2; }
    for(i = 0; i < 9; i++) {
        if(p[i] >= n) return true;
        u = power_mod(p[i], m, n);
        if(u == 1) continue;
        for(j = 0; j < k; j++)
            buf = produc_mod(u, u, n);
        if(buf == 1 && u != 1 && u != n-1) return false;
        u = buf;
    }
    if(u-1) return false;
}
return true;

int pollard_rho(Int n) {
    while(1) {
        int i = 1;
        int x = rand() % (n-1) + 1, y = x, k = 2, d;
        do {
            i++;
            d = Gcd(n + y - x, n);
            if(d > 1 && d < n) return d;
            if(i == k) y = x, k *= 2;
            x = ( produc_mod(x, x, n) + n - gcd ) % n;
        } while(y != x);
    }
    int prime[10000];
    int top,
    void prime_divisor(Int key) { prime[top++] = key;
        if( rabin_miller(key) ) prime[top++] = key;
        else {
            Int buf = pollard_rho(key);
            if( rabin_miller(buf) ) prime[top++] = buf;
            else prime_divisor(buf);
            if( rabin_miller(key / buf) ) prime[top++] = key / buf;
            else prime_divisor(key / buf);
        }
    }

    // Determina se n pode ser escrito como a soma de quadrados perfeitos
    int main() {
        int t, i;
        Int n;
        scanf("%d", &t);
        while(t--) {
            scanf("%ld", &n);
            if( n < 0 ) printf("NO\n");
            else if ( n == 0 || n == 1 ) printf("YES\n");
            else {
                top = 0; prime_divisor(n);
                for(i = 0; i < top; ++i) if(prime[i] != 2 && (prime[i]-1)%4) break;
                if(i < top) printf("NO\n"); else printf("YES\n");
            }
        }
    }

    // phi
    int phi[MAX+1];
    void totient() {

```

```

int i, j;
for (i=1; i <= MAX; i++) phi[i] = i;
for (i=2; i <= MAX; i+=2) phi[i] >>= 1;
for (j=3; j <= MAX; j+=2) {
    if (phi[j] == j) { phi[i] = phi[i] / j * (j - 1); }
}
// (a * b) % MOD sem overflow. ATENCAO: mod < 2^62
uint64 mult_mod(uint64 a, uint64 b, uint64 MOD) {
    uint64 y = (float64)a*(float64)b/MOD + (float64)1/2;
    y *= MOD; uint64 x = a * b; uint64 r = x - y;
    ((long long)r < 0) r += MOD, y--;
    return r;
}
// qtd. de nros. <= n que sao multiplos de algum elemento de v. V1: generica.
// long long count_vl(vector <long long> &v, int index, long long n, long long LCM) {
    long long req, ans = 0;
    for (int i=0; i < index; ++i) {
        req = lcm(LCM,v[i]); if (req <= n) ans += n / req - cnt(v,i,n,req);
    }
    return ans;
}
// V2: mais rapida, so funciona quando gcd(v[i],v[j]) == 1, para qualquer i != j
int count_v2(int v[], int index, int n) {
    int ans = 0, i;
    for (i=0; i < index && v[i] <= n; i++) ans += n / v[i] - count(v,i,n/v[i]);
    return ans;
}
// ***** MATROÍDES *****
const int N = 1001; // numero de elementos do conjunto base (arestas)
struct UnionFind { // estrutura union find para matroid grafico (florestas)
    int p[N], comp_sz[N];
    void clear( int m ) {
        memset( p, -1, m * sizeof( int ) );
        memset( comp_sz, 0, m * sizeof( int ) );
    }
    int find( int x ) {
        while( p[x] != -1 ) x = p[x];
        return x;
    }
    void union_set( int x, int y ) {
        int u = find(x), v = find(y);
        if( u != v ) {
            if( comp_sz[u] < comp_sz[v] ) { p[u] = v; comp_sz[v] += comp_sz[u]; }
            else { p[v] = u; comp_sz[u] += comp_sz[v]; }
        }
    }
    UnionFind() {
        // M1 = (E,I1): matroide grafico (arvores geradoras)
        // M2 = (E,I2): matroide de particao - cores (cor i pode aparecer no maximo
        // Algoritmo: Intersecao de dois matroides para encontrar o subconjunto
        // independente maximal de I1 \cap I2
        int n, m, K; // vertices, arestas, cores
        int x[N], y[N]; // i-esima aresta = (x[i],y[i])
        int maxcor[N]; // maxcor[i] - numero de vezes que a cor i pode ainda ser
        vector<int> cor[N]; // cor[i] - cor da i-esima aresta
        bool base[N]; // base[i] - se a i-esima aresta pertence ao conjunto
    }
}

```

```

bool X1[N], X2[N]; // X1 = {x \in S \setminus I : \{I + x\} \in I2}, X2 = {x \in S \setminus I : \{I + x\} \in I1}
int prev[N]; // para bfs que encontra caminho minimo em g de X1 para X2
bool visited[N];
// no caso de achar a intersecao de matroides valorados, trocar bfs por
// caminho de custo minimo, e no caso de empate, escolher aquele que
// visita o menor numero de vertices
int bfs() {
    for(i,m) if( X1[i] && X2[i] ) return i;
    memset( prev, -1, m * sizeof( int ) );
    memset( visited, false, m * sizeof( bool ) );
    queue<int> Q;
    for(i,m) if( X1[i] ) { Q.push(i); visited[i] = true; }
    while( !Q.empty() ) {
        int v = Q.front(); Q.pop();
        for(j,g[v].sz) {
            int u = g[v][j];
            if( visited[u] ) continue;
            prev[u] = v;
            if( X2[u] ) return u;
            Q.push(u); visited[u] = true;
        }
    }
    return -1;
}
int matroid_intersection() {
    int res;
    for( res = 0; true; res++ ) {
        for(i,m) g[i].clear(); // monta grafo auxiliar
        for(i,1,m) if( base[i] ) { // for(i,m), i -> Y e j -> x
            ufs.clear(n); // arcos do matroide I1
            for(j,m) if( base[j] && i != j ) ufs.union_set( x[j], y[j] );
            for(j,m) if( base[j] && ufs.find(x[j]) != ufs.find(y[j]) ) g[i].pb(j);
            for(j,m) if( !base[j] ) { // arcos do matroide I2
                maxcor[cor[i]]++;
                if( maxcor[cor[i]] >= 1 ) g[j].pb(i);
                maxcor[cor[i]]--;
            }
        }
        // inicializa estrutura union find (vertices de arestas da base devem
        // pertencer ao mesmo componente)
        ufs.clear(n);
        for(i,m) if( base[i] ) ufs.union_set( x[i], y[i] );
        for(i,m) { // inicializa X1 e X2
            X1[i] = X2[i] = false;
            if( !base[i] ) {
                // se a aresta i ainda pode ser utilizada (nao forma ciclo com
                // as arestas da base), entra em X1
                if( ufs.find(x[i]) != ufs.find(y[i]) ) X1[i] = true;
                // se a cor[i] ainda pode ser utilizada, entra em X2
                if( maxcor[cor[i]] >= 1 ) X2[i] = true;
            }
        }
        // menor caminho no grafo g de X1 para X2
        int v = bfs(); // v eh o vertice de X2 que foi alcancado primeiro
        if( v == -1 ) break; // encontrou subc. independente maximal de I1 \cap I2
        while (true) { // atualiza elementos da base
            maxcor[cor[v]] += (base[v] ? 1 : -1); base[v] = base[v];
            if( X1[v] ) break; v = prev[v];
        }
    }
    return res;
}

```

UFMG – Universidade Federal de Minas Gerais

```

}
int main() {
    int cas = 1;
    while ( scanf( "%d%d%d", &n, &m, &K ) == 3 ) {
        for(i,K) maxcor[i] = 1; // cada empresa so pode ser respons. por 1 aresta
        for(i,m) {
            scanf( "%d%d%d", &x[i], &y[i], &cor[i] );
            x[i]--; y[i]--;
        }
        int res = matroid_intersection();
        printf( "Instancia%d\n",cas++ );
        if( res == n-1 ? "sim" : "nao" );
    }
    return 0;
}

// ***** ESTRUTURAS DE DADOS *****
// LCA: lowest common ancestor
const int MAXN = 1010; // numero maximo de vertices
int N; // numero de vertices
int parent[MAXN], level[MAXN];
vector< vector< int >> graph; vector< int > in_degree, visited;
void determine_level( int v, int lev ) { // O(N)
    visited[v] = true;
    for(i,graph[v].sz) if ( !visited[graph[v][i]] )
        determine_level( graph[v][i], lev+1 );
}

// implementacao < O(N log N), O(log N) >
const int LOGMAXN = 12;
const int P[MAXN][LOGMAXN]; // P[i][j] - (2^j)-esimo ancestral de i
void pre_process() { // O(N log N)
    for(i,N) for(j,(1<N)) P[i][j] = -1;
    for(i,N) P[i][0] = parent[i];
    for( ; int j = 1; 1 <N; ++j ) for(i,N) if ( P[i][j-1] != -1 )
        int tmp, log;
        for( log = 1; 1 <i) >= level[q] ) p = P[p][i];
        if ( p == q ) return p;
        for(i,log,0) if ( P[p][i] != -1 && P[p][i] != P[q][i] )
            p = P[p][i], q = P[q][i];
    return parent[p];
}

int main() {
    // ... computa in_degree, graph e parent
    int root;
    for(i,N) if( in_degree[i] == 0 ) { root = i; break; }
    determine_level( root, 0 );
    visited.assign( N, 0 );
    pre_process();
}

// implementacao < O(N), O(sqrt(N)) >
int P[MAXN]; // P[i] - pai do v. i q esta situado no ult. niv. da secao anterior
void dfs( int v, int nr ) { // O(N)
    visited[v] = 1;
    if ( level[v] < nr ) P[v] = 1;
    else {
        if( !( level[v] % nr ) ) P[v] = parent[v];
        else P[v] = P[parent[v]];
    }
}

```

UFMG – Universidade Federal de Minas Gerais

```

for(i,graph[v].sz) if ( !visited[graph[v][i]] ) dfs( graph[v][i], nr );
}

int query( int x, int y ) { // O(sqrt(N))
    while ( P[x] != P[y] ) {
        if ( level[x] > level[y] ) x = P[x];
        else y = P[y];
    }
    while ( x != y ) {
        if ( level[x] > level[y] ) x = parent[x];
        else y = parent[y];
    }
    return x;
}

int main() {
    // ... computa in_degree, graph e parent
    int root, height = -1;
    for(i,N) if( in_degree[i] == 0 ) { root = i; break; }
    determine_level( root, 0 );
    for(i,N) height = max( height, level[i] );
    visited.assign( N, 0 );
    dfs( root, height );
}

// RMQ: Range Minimum Query - Arvore de segmentos - <O(N),O(log N)>
const int MAXN = 10010; // nro. maximo de elementos da sequencia original
const int MAXIND = 3000; // 2 * 2^{logN + 1} + 1
int N; // numero de elementos do vetor A
int M[MAXIND]; // M[i]-ind. em A do menor valor do intervalo
int A[MAXN]; // A[i] - i-esimo elemento da sequencia original
void pair( int intervals[MAXIND]; // intervalo associado a cada no i
void initialize( int node, int b, int e ) { // O(N) - node = 1, b = 0, e = N-1
    if ( intervals[node] = make_pair( b, e );
        if ( b == e ) M[node] = b;
    else {
        int m = ( b + e ) / 2;
        initialize( 2 * node, b, m );
        initialize( 2 * node + 1, m + 1, e );
        if ( A[M[2 * node]] <= A[M[2 * node + 1]] ) M[node] = M[2 * node];
        else M[node] = M[2 * node + 1];
    }
}

int query( int node, int b, int e, int i, int j ) {
    if ( i > e || j < b ) return -1;
    if ( b >= i && e <= j ) return M[node];
    int m = ( b + e ) / 2;
    int p1 = query( 2 * node, b, m, i, j );
    int p2 = query( 2 * node + 1, m + 1, e, i, j );
    if ( p1 == -1 ) return p2; if ( p2 == -1 ) return p1;
    if ( A[p1] < A[p2] ) return p1;
    return p2;
}

// O(log N) - node = 1, b = 0, e = N-1
// OBS: alterar valor de A[k] para dessa funcao
void update( int node, int b, int e, int k ) {
    if ( k > e || k < b ) return; // faz a funcao ser O(log N)
    if ( b == e ) M[node] = k;
    else {
        int m = ( b + e ) / 2;
        update( 2 * node, b, m, k );
        update( 2 * node + 1, m + 1, e, k );
        if ( A[M[2 * node]] <= A[M[2 * node + 1]] ) M[node] = M[2 * node];
        else M[node] = M[2 * node + 1];
    }
}

```

```

int main() {
    int Q, a, b;
    scanf( "%d %d", &Q );
    initialize( -1, sizeof( M ) );
    for(i,N) scanf( "%d", &A[i] );
    for(i,Q) { i.mr = 0; i.ml = 0; }
    scanf( "%d %d", &a, &b );
    a--; b--;
    printf( "%d\n", A[query( 1, 0, N-1, a, b )] );
}
return 0;
}

// Aplicacoes de RMQ:
// 1 - Determinar a maior area dada pelo produto do tamanho do segmento entre i e j e o menor valor contido nesse segmento. Solucao: divisao e conquista
long long greater_area( int i, int j )
{
    if ( i > j ) return -1;
    if ( i == j ) return A[i];
    long index = query( 1, 0, N-1, i, j );
    long long total_area = greater_area( i, index - 1 );
    long long right_area = greater_area( index + 1, j );
    return max( total_area, max( left_area, right_area ) );
}

// 2 - Determinar o valor mais frequente de um intervalo ordenado.
// Solucao: transformar o problema em RMQ (maximum).
int v[MAXN], // entrada do problema
int inicio[MAXN], // inicio[i] - indice em que inicia o valor v[i]
int fim[MAXN], // fim[i] - indice em que termina o valor v[i]
int preprocess_most_frequent_value() {
    // v em um problema de RMQ no vetor A
    // Ex: v = [-1 -1 3 3 3 4 5 10 10]
    // A = [ 2 2 3 3 3 2 2 1 2 2 ] - frequencia de cada valor
    int freq = 0, last = INF, first_index = 0;
    if ( v[i] != last ) {
        for(j,first_index,i-1) { A[j]=-freq; inicio[j]=first_index; fim[j]=i-1; }
        freq = 1; first_index = i;
    }
    else freq++;
    last = v[i];
}
for(j,first_index,N-1) { A[j]=-freq; inicio[j]=first_index; fim[j]=N-1; }

memset(M,-1, sizeof( M ) );
initialize( 1, 0, N-1 );
}

// retorna o indice do valor mais frequente. trata o primeiro e o ultimo valores
// do intervalo como casos especiais
int query_most_frequent_value( int b, int e ) {
    int freq1, freq2, freq3 = -INF, index, MAX;
    freq1 = min( fim[b], e ) - b + 1; freq2 = e - max( inicio[e], b ) + 1;
    if ( b + freq1 < e - freq2 ) { index = query( 1, 0, N-1, b + freq1, e - freq2 ); freq3 = -A[index]; }
    else freq3 = -MAX;
    MAX = max( freq1, max( freq2, freq3 ) );
    return ( MAX == freq1 ) ? ( b ) : ( MAX == freq2 ? e : index );
}

// 3 - determinar o segmento de soma maxima.
struct interval { int m_n, mr, ml, m_n, m_center, full, n; };
interval M[MAXIND]; // unica alteracao nos dados globais

```

```

interval join( const interval &i1, const interval &i2 ) {
    int max_sum = i1.mr + i2.ml, max_n = i1.mr_n + i2.ml_n;
    interval i;
    if( i1.mr + i2.full >= i2.mr ) {
        i.mr = i1.mr + i2.full; i.mr_n = i1.mr_n + i2.n;
        if( i.mr > max_sum ) { max_sum = i.mr; max_n = i.mr_n; }
        else if ( i.mr == max_sum && i.mr_n > max_n ) max_n = i.mr_n;
    }
    else { i.mr = i2.mr; i.mr_n = i2.mr_n; }
    if( i2.ml + i1.full >= i1.ml ) {
        i.ml = i2.ml + i1.full; i.ml_n = i2.ml_n + i1.n;
        if( i.ml > max_sum ) { max_sum = i.ml; max_n = i.ml_n; }
    }
    else if( i.ml == max_sum && i.ml_n > max_n ) max_n = i.ml_n;
    return i;
}

void initialize( int node, int b, int e ) {
    if ( b == e ) {
        M[node].mr_n = M[node].ml_n = M[node].m_n = M[node].m_center = A[b];
        M[node].full = M[node].mr = M[node].ml = M[node].m_center = 1;
    }
    else {
        initialize( 2 * node, b, (b + e) / 2 );
        initialize( 2 * node + 1, (b + e) / 2 + 1, e );
        M[node] = join( M[2*node], M[2*node + 1] );
    }
}

interval query( int node, int b, int e, int i, int j ) {
    if ( i > e || j < b ) { pl.n = -1; return pl; }
    if ( b >= i && e <= j ) return M[node];
    pl = query( 2 * node, b, (b + e) / 2, i, j );
    interval p2 = query( 2 * node + 1, (b + e) / 2 + 1, e, i, j );
    if ( pl.n == -1 ) return p2; if ( p2.n == -1 ) return pl;
    return join( pl, p2 );
}

// 4 - construir uma treap (Binary Search Heap). Cada no tem uma prioridade
// (usada para montar o heap) e um label (usado para montar a arvore binaria).
// Solucao: Divisao e conquista com arvore de segmentos, em que cada consulta
// retorna o proximo no da treap
typedef pair<string, int> treap_node;
treap_node A[MAXN]; // A[i] - i-esimo elemento da sequencia original
void build_treap( int i, int j ) {
    if ( i > j ) return;
    if ( i == j ) { printf( "(%s%d)", A[i].first.c_str(), -A[i].second ); return; }
    int index = query( 1, 0, N-1, i, j );
    print( "(", build_treap( 1, index - 1 );
    print( "%s%d", A[index].first.c_str(), -A[index].second );
    build_treap( index + 1, j ); printf( ")" );
}

int main() {
    string s, label; int a;
    while ( cin >> N && N ) {
        memset( M, -1, sizeof( M ) );
        for(i,N) {
            cin >> s;
            cin >> a;
            cout << s << endl;
        }
        int separator = s.find('/');
    }
}

```

```

label = s.substr( 0, separator );
sscanf( s.substr( separator + 1 ), "%d", &a );
A[i] = make_pair( label, -a );
}

sort( A, A + N ); // ordena de acordo com os labels
initialize( 1, 0, N-1 );
build_treap( 0, N-1 );
printf( "\n" );
}

return 0;
}

// 5 - determinar a soma dos dois elementos de maior valor em [i, j].
int max_sum( int node, int b, int e, int i, int j ) {
    int index, left, right;
    index = query( node, b, e, i, j );
    left = query( node, b, e, i, index - 1 );
    right = query( node, b, e, index + 1, j );
    return -A[index] + max( -A[left], -A[right] );
}

// 6 - determinar qual o no mais a esquerda na arvore tem valor
// maior que K, e atualizar os dados apropriadamente.
void update( int node, int l, int r, int K ) {
    if ( l == r ) { A[l] -= K; M[node] = l; return; }
    if ( A[M[2 * node]] >= K ) update( 2 * node, l, m, K );
    else if ( A[M[2 * node + 1]] >= K ) update( 2 * node + 1, m + 1, r, K );
    else M[node] = M[2 * node + 1];
}

// Árvore de intervalos
#define MAX 30010
struct Node { int l, r, x, count, cl, cr; };

Node tree[4 * MAX];
void create(int l, int r, int i = 1) {
    tree[i].l = l; tree[i].r = r; tree[i].x = (l + r) / 2;
    tree[i].count = tree[i].cl = tree[i].cr = 0;
    if (r-1 == l) return;
    if (l < tree[i].x) create(l,tree[i].x,r,2*i+1);
    if (r > tree[i].x) create(tree[i].x,r,2*i+1);
}

int update(int l, int r, int w = 1, int i = 1) {
    if (l <= tree[i].l && r >= tree[i].r) tree[i].count += w;
    else if (tree[i].r-tree[i].l > 1) {
        if (l < tree[i].x) tree[i].cl = update(l,r,w,i*2);
        if (r > tree[i].x) tree[i].cr = update(l,r,w,i*2+1);
    }
    return tree[i].count > 0 ? tree[i].r-tree[i].l:tree[i].cl+tree[i].cr;
}

int query(int x, int i = 1) {
    int ans = tree[i].count;
    if (tree[i].r-tree[i].l == 1) return ans;
    if (x < tree[i].x) ans += query(x,2*i);
    if (x >= tree[i].x) ans += query(x,2*i+1);
    return ans;
}

int count_occurrences(int l, int r, int i = 1) {
    int ans = 0;
    if (tree[i].count > 0) return r - l;
    if (tree[i].r - tree[i].l == 1) return 0;
    if (l == tree[i].l && r >= tree[i].x) ans += tree[i].cl;
    else if (l < tree[i].x && tree[i].cl)
        ans += count_occurrences(l,min(r,tree[i].x),2*i);
    if (r == tree[i].x && l < tree[i].x) ans += tree[i].cr;
}

// Fenwick Tree (BIT) 1D
int tree[MAX];
void create(int t[], int n) { memset(t, 0, n*sizeof(int)); }
int query( int tree[], int from, int to ) {
    int sum;
    if (from != 0) return query(tree, 0, to) - query(tree, 0, from-1);
    for (sum=0; to >= 0; to = (to & (to + 1)) - 1) sum += tree[to];
    return sum;
}

// Fenwick Tree (BIT) 2D
int tree[MAX1][MAX2];
void create(int nl, int n2) {
    for (int i=0; i < nl; i++) memset(tree[i], 0, n2*sizeof(int));
}
int query2( int x, int y ) {
    int sum; for (sum=0; y >= 0; y=(y&(y+1))-1) sum += tree[x][y];
    return sum;
}

int query( int xl, int y1, int x2, int y2 ) {
    int sum;
    if (xl == 0) return query(0,y1,x2,y2) - query(0,y1,x1-1,y2);
    for (sum=0; x2 >= 0; x2=(x2&(x2+1))-1) {
        sum += query2(x2,y2);
    }
    if (y1) sum -= query2(x2,y1-1);
}

return sum;
}

void update( int x, int y, int inc, int nl, int n2) {
    int y2; for ( ; x < nl|x|=x+1) tree[x][y2]+=inc;
}

// Bitwise operations
// MISC *****
A | B; // set union
A & B; // set intersection
A & ~B; // set subtraction
((1 << N) - 1) & ~A // set negation
A |= 1 << bit; // set bit
A &= ~ (1 << bit); // clear bit
(A & 1 << bit) != 0; // test whether element bit is in A
(A > 0) && (A & (A - 1)) == 0 // whether A has exactly 1 element
A = A & (A - 1) // remove the smallest element from A
A = A & ~ (A - 1) // remove all but the small. elem. from A
A & (A << 1) // check if there are adjacent bits
_builtin_ctz(A) // count trailing zeros
_builtin_ctz(A) // count leading zeros, do not use when A=0
_builtin_popcount(A) // number of 1 bits
for( int x = 0; x < (1 << n); ++x ) // all subsets of {0,...,n-1}
for( int x = s; x != 0; x = (x - 1) & s ) // all non-empty subsets of s
for( int x = 0; x = x - s & s; ) // all subsets of s in increa. order
// all subsets of {0,...,n-1} in gray code order
set = 0;
for(j=1,(1<n)-1) {
    last_set = set; set ^= j & ~ (j-1);
    diff = set ^ last_set; index = __builtin_ctz( diff );
}
// iterate through all k-element subsets of {0,...,n-1} in increase order

```

```

int x = (1 << k) - 1; // do not use when k=0
while ( !(x & 1 << n) ) {
    int lo = x & ~(x - 1), lz = (x + lo) & ~x;
    x |= lz; x &= ~(lz - 1); x |= (lz / lo / 2) - 1;
}

// Dados n pontos distintos 3D, p1<p2 se e somente se: p1.x<=p2.x AND p1.y < p2.y
// AND p1.z<p2.z. Um ponto p1 eh excelente se nao existe j tal que pj < pi.
// Problema: determinar quantos pontos sao excellentes. Solucao: ordenar pontos
// pela coordenada z, e varrer os pontos nesta ordem decidindo quais sao
// excellentes. Quando o ponto pi eh avaliado, todos os pontos que podem ser
// melhores do que ele ja terao sido avaliados, uma vez que estes possuem
// coordenada z menor que pi.z. A ideia eh manter um conjunto S de pontos
// ordenados (em relacao a x) que sao excellentes e nao sao dominados por nenhum
// outro ponto excelente. Complexidade: O(n log n).

struct point { int x, Y, z; }

bool operator<( const Point & a, const Point & b ) { return a.z < b.z; }

bool comp_z( const point & a, const point & b ) { return a.z < b.z; }

point v[100010]; set< point >::iterator S; // ordenado em relacao a x

vector< point > V; // auxiliar

if ( j == S.begin() ) return true; --j;

return v[i].y < j->y;
}

int main() {
    int T, N;
    scanf( "%d", &T );
    while ( T-- )
    {
        scanf( "%d", &N );
        for(i, 0; i < N; i++) scanf( "%d %d %d", &x[i].x, &x[i].y, &x[i].z );
        sort( v, v+N, comp_z );
        int asw = 1; // o lo. ponto eh excelente
        S.clear(); S.insert( v[0] );
        for(i, 1, N-1) if ( is_excellent( i ) ) {
            asw++; // ponto i eh excelente
            // remove pontos excellentes que sao dominados por v[i]
            // dominado: ponto p2 eh dominado pelo ponto p1 se p1.x < p2.x
            // e p1.y < p2.y; ainda assim p2 eh um ponto excelente
            V.clear();
            for(set<point>::iterator it = S.lower_bound(v[i]); it!=S.end(); ++it){
                if ( v[i].y < it->y ) V.push_back(*it);
                else break; // ja que a coordenada y esta diminuindo
            }
            for(j, V.size(), 0; j > i; j--) S.erase( V[j] );
            S.insert( v[i] );
            printf( "%dn", asw );
        }
    }
}

// Dados N valores inteiros em um vetor v, calcula a soma das medianas de cada
// intervalo de tamanho K. Complexidade: O(n log n)
const int LOGMAXN = 17, MAXN = 65536; // valores de v no intervalo [0, MAXN]
int tree[LOGMAXN][MAXN];
void insert(int x) { for(i,1,LOGMAXN) { tree[i][x]++; x/=2; } }
int kth_element(int k) { for(i,1,LOGMAXN) { tree[i][x]--; x/=2; } }
while (b--) { a*=2; if (tree[b][a]<k) k -= tree[b][a++]; }
return a;

long long sum_of_medians( const vector< int > & v, int N, int K ) {
}

```

```

memset( tree, 0, sizeof(tree) );
for(i,N) {
    insert( v[i] );
    if (i>=K) erase( v[i-K] );
    if (i>=K-1) result += kth_element( (K+1)/2 );
}
return result;

// Algoritmo probabilistico: determinar se existe um nro q aparece + que
// n/2 vezes em um intervalo de tamanho n. Prob. de acertar: 1-(1/2)^ITER
#define MAX 300000
#define ITER 18
int n, Q, C, lo, hi, m, iter, x, cnt, a[MAX], pair<int, int> b[MAX];
while ( scanf( "%d%d", &n, &C ) == 2 ) {
    for(i,0; i < n; i++) { scanf( "%d", &a[i] ); b[i] = make_pair( a[i], i ); }
    sort( b, b+n );
    for(i,0; i < n; i++) {
        scanf( "%d", &Q );
        for(i,qq,Q) {
            m = hi-1-lo+1;
            for(litter,ITER) {
                x = a[lo+rand()%m];
                cnt = upper_bound( b, b+m, make_pair(x, hi) ) -
                    lower_bound( b, b+m, make_pair(x, lo) );
                if ( cnt*2 > m ) break;
                if( iter == ITER ) printf( "no\n" );
                else printf( "yes %dn", x );
            }
        }
    }
    return 0;
}

// Distancia minima entre dois cavalos em um tabuleiro de xadrez - O(1)
long long dist( long long xl, long long yl, long long x2, long long y2 ) {
    long long dx = abs(x2-xl), dy = abs(y2-yl), lb = (dx+1)/2;
    if ( abs(dx) == 1 && dy == 0 || abs(dy) == 1 && dx == 0 ) return 3;
    if ( abs(dx) == 2 && abs(dy) == 2 ) return 4;
    lb = max( lb, (dy+1)/2 ); lb = max( lb, (ax+dy+2)/3 );
    while ( (lb%2) != (dx+dy)%2 ) lb++;
    return lb;
}

// Notacao convencional p/ notacao polonesa reversa. Operandos: a, b, ... z.
// Operadores em ordem de precedencia: +-*^/, expressoes com parentizacao.
void infix( reverse polish( const string & s ) {
    int n = s.size(), pilha[MAXN], topo = 0; string res, operators = "+-*^/";
    for(i,0; i < n; i++) {
        if ( isalpha( s[i] ) ) printf( "%c", s[i] );
        else if ( s[i] == '(' ) pilha[topo++] = '(';
        else if ( s[i] == ')' ) {
            while ( topo != -1 ) {
                if ( pilha[topo-1] == '(' ) { topo--; break; }
                print( "%c", pilha[--topo] );
            }
        }
        else if ( operators.find(s[i]) != string::npos ) {
            while ( topo && operators.find(pilha[topo-1])!=string::npos &&
                    operators.find(s[i]) <= operators.find(pilha[topo-1]) )
                print( "%c", pilha[--topo] );
            pilha[topo++] = s[i];
        }
    }
    while ( topo != 0 ) print( "%c", pilha[--topo] );
    puts( "" );
}

```

```

    // josephus
    long long Josephus( long long n, long long d ) {
        long long K = 1; K = ( d * K + d - 2 ) / ( d - 1 );
        return d * n + 1 - K;
    }

    // returns the index (0-based!) of s among all the possible anagrams
    int freq[256];
    long long count;
    long long find_index(char *s, int n) {
        if (n < 2) { freq[*s-'a']++; count = 1; return 0; }
        ans = find_index(s+1,n-1);
        count = (count * n) / ++freq[*s-'a'];
        for (i=0; i < *s-'a'; i++) ans += freq[i] * count / n;
        return ans;
    }

    // returns the index k-th (1-based!) anagram of s
    string getAnagram(string s, int k) {
        int n = s.length();
        vector<int> freq(26,0), index(26);
        long long acc = 0, count;
        if (n) return "";
        sort(s.begin(),s.end());
        for (int i=0; i < n; i++) { freq[s[i]-'a']++; index[s[i]-'a'] = i; }
        count = fat(n);
        for (int i=0; i < 26; i++) count /= fat(freq[i]);
        if (freq[i] && acc+count*freq[i]/n >= k) {
            s[index[i]] = s[0];
            return (char)('a'+i) + getAnagram(s.substr(1),k-acc);
        }
        acc += count * freq[i] / n;
    }
    return "";
}

// Minimum Lexicographic Rotation - O(n lg n)
int min_index( const string& s ) {
    int n = s.size(); vector<int> v(n);
    string ss, s1, s2; ss = s + s;
    for(i, v.sz) v[i] = i;
    while ( v.sz != 1 ) {
        vector<int> vv;
        for ( int i = 0; i < v.sz; i += 2 ) {
            if ( i < v.sz - 1 ) {
                s1 = ss.substr( v[i], v[i+1] - v[i] );
                s2 = ss.substr( v[i+1], v[i+1] - v[i] );
                if ( s1 <= s2 ) vv.pb(v[i]);
                else vv.pb(v[i+1]);
            }
            else vv.pb(v[i]);
        }
        v = vv;
    }
    return v[0];
}

// Suffix array
// ASCII: 33 a 47 -> !#$%&(*+-./
const int MAXN = 200010; // numero maximo de chars na string
int N, v[MAXN]; // tamanho da string atual e representacao em inteiros da string
int SA[MAXN], SAR[MAXN]; // suffix array e suffix array "reverso"
int LCP[MAXN]; // longest common prefix

```

```

int CUR_ALF; // tamanho do alfabeto da string atual
const char FIRST_CHAR = ','; // primeiro char do alfabeto (tabela ASCII)
const char LAST_CHAR = 'z'; // ultimo char do alfabeto (tabela ASCII)
const int ALF = LAST_CHAR - FIRST_CHAR + 1; // tamanho do alfabeto
vector<int> > occ[ALF]; // occ[i] - posicoes em que o char i aparece
#define GetI() ( SA12[t] < n0 ? SA12[t] * 3 + 1 : (SA12[t] - n0) * 3 + 2 )
inline bool leq( int al, int a2, int bl, int b2 ) {
    return( al < bl || al == bl && a2 <= b2 );
}
inline bool leq( int al, int a2, int a3, int bl, int b2, int b3 ) {
    return( al < bl || al == bl && leq( a2, a3, b2, b3 ) );
}
static void radix_pass( int* a, int* b, int* r, int n, int K ) {
    int* c = new int[n+1];
    for(i,0,K) c[i] = 0;
    for(i,n) c[r[a[i]]]++;
    int sum = 0;
    for(i,0,K) { int t = c[i]; c[i] = sum; sum += t; }
    for(i,n) b[c[r[a[i]]++]] = a[i];
    delete [] c;
}

void suffix_array( int* s, int* SA, int n, int K ) {
    int i, j, n0 = (n+2)/3, n1 = (n+1)/3, n2 = n/3, n02 = n0+n2;
    int* s12 = new int[n02+3]; int* SA12 = new int[n02+3];
    int* s0 = new int[n0]; int* SA0 = new int[n0];
    int name = 0, c0 = -1, c1 = -1, c2 = -1;

    for(i = 0, j = 0; i < n+(n0-n1); i++) if ( i % 3 != 0 ) s12[j++] = i;
    s12[n02] = s12[n02+1] = s12[n02+2] = SA12[n02] = SA12[n02+1] = SA12[n02+2]=0;
    radix_pass( s12, SA12, s+2, n02, K ); radix_pass( SA12, s12, s+1, n02, K );
    radix_pass( s12, SA12, s, n02, K );

    for(i, n02) {
        if (s[SA12[i]] != c0 || s[SA12[i]+1] != c1 || s[SA12[i]+2] != c2) {
            name++; c0 = s[SA12[i]]; c1 = s[SA12[i]+1]; c2 = s[SA12[i]+2];
        }
        else s12[SA12[i]/3 + n0] = name;
    }

    if ( name < n02 ) {
        suffix_array( s12, SA12, n02, name );
        for(i, n02) s12[SA12[i]] = i + 1;
    }
    else for(i, n02) SA12[s12[i]-1] = i;

    j = 0;
    for(i, n02) if ( SA12[i] < n0 ) s0[j++] = 3 * SA12[i];
    radix_pass(s0, SA0, s, n0, K);

    for ( int p = 0, t = n0-n1, k = 0; k < n; k++ ) {
        int i = GetI();
        int j = SA0[p];
        if ( SA12[t] < n0 )
            leq( s[i], s12[SA12[t] + n0], s[j], s12[j/3] );
        else {
            leq( s[i], s12[SA12[t] + n0], s[j], s12[j/3] );
            leq( s[i], s12[SA12[t] + n0], s[j+1], s12[j/3+n0] );
        }
        if ( p == n0 ) for ( k++; t < n02; t++, k++ ) SA[k] = GetI();
        if ( t == n02 ) for ( k++; p < n0; p++, k++ ) SA[k] = SA0[p];
    }
    else {
        SA[k] = j; p++;
        if ( p == n0 ) for ( k++; t < n02; t++, k++ ) SA[k] = GetI();
    }
    delete [] s12; delete [] SA12; delete [] SA0; delete [] s0;
}

```

```

    // Transforma alfabeto qualquer em alfabeto inteiro
    // Ex: "aebeg" para "1 3 2 3 4". Complexidade: O(N + ALF)
    void initialize( const string & s ) {
        N = s.sz;
        memset( v, 0, sizeof( v ) );
        int cnt = 1;
        for(i,N) occ[i].clear();
        for(i,N) if (iocc[i].empty()) {for(j,occ[i].sz) v[occ[i][j]]=cnt; ++cnt;}
        CUR_ALF = cnt-1;
    }
    // Constrói o vetor LCP. Complexidade: O(N).
    // LCP[i] = longest common prefix dos sufixos i e i+1
    void lcp( const string & s ) {
        for(i,N) SAR[SA[i]] = i;
        int h = 0, r;
        for(p,N) if (SAR[p]+1 < N) {
            r = SA[SAR[p]+1];
            while( r+h < N && p+h < N && s[r+h] == s[p+h] ) h++;
            LCP[SAR[p]] = h;
            if( h > 0 ) h--;
        }
        // apenas se for necessário calcular lcp em O(log n) - usando segtree
        memset( M, -1, sizeof( M ) );
        initialize( 1, 0, N-2 );
    }
    // Retorna o LCP dos sufixos i e j. Complexidade: O(n)
    // Propriedade: lcp(i, j) = min{ $i \leq k < j$ } LCP[k]
    int lcp( linear( int i, int j ) ) {
        int MIN = INF;
        for(k,i,j-1) if ( LCP[k] < MIN ) MIN = LCP[k];
        return MIN;
    }
    // Segtree para queries de qualquer intervalo em lcp em O(log n)
    // Overhead da segtree pode não compensar
    const int MAXIND = 525000;
    int M[MAXIND];
    void initialize( int node, int b, int e ) { ... } // LCP no lugar de A
    int query( int node, int b, int e, int i, int j ) { ... } // LCP no lugar de A
    // Retorna o LCP dos sufixos i e j. Complexidade O(log n)
    // Necesário criar a segtree para fazer RMQ
    int lcp_log( int i, int j ) { return LCP[query( 1, 0, N-2, i, j-1 )]; }
    // Aplicações do suffix array:
    // 1 - Dada uma string s de tamanho N, retorna a maior substring de s que
    // aparece pelo menos M vezes. Em caso de empate retorna a menor lexicográfica.
    // Solução: construir o suffix array de s e iterar com o contador i da posição
    string longest_substring( const string & s, int M ) {
        if ( M == 1 ) return s; // apenas com arvore de segmentos
        N = s.sz;
        string suffix_array( N ); suffix_array( v, SA, N, CUR_ALF ); lcp( s );
        initialize( s ); suffix_array( v, SA, N, CUR_ALF );
        lcp( s );
        // o intervalo [start,end] deve conter pelo menos um sufixo
        // de cada uma das strings de vs
        for(i,start,N-1) {
            if ( start != 0 ) {
                occur[init[SA[start-1]]]--;
                if ( occur[init[SA[start-1]]] == 0 ) sum--;
            }
            while ( sum != num_words && end < N ) {
                if ( occur[init[SA[end]]] == 0 ) occur[init[SA[end]]] = 1; sum++;
                else occur[init[SA[end]]]++;
            }
            if ( sum == num_words && ( aux = lcp( start, end - 1 ) ) > asw ) asw = aux;
        }
        return asw;
    }
}

```

```

    initialize( s ); suffix_array( v, SA, N, CUR_ALF ); lcp( s );
    int MAX = 0, pos = 0, cnt = 0;
    for(i,N-1) {
        if ( LCP[i] > MAX ) { MAX = LCP[i]; pos = i; cnt = 2; }
        else if ( LCP[i] == MAX )
    }
    int j = 0;
    for( ; j < MAX; ++j ) if ( s[SA[i]+j] != s[SA[pos]+j] ) break;
    if ( j == MAX ) ++cnt;
}
if ( MAX == 0 ) return make_pair( "", 0 );
return make_pair( s.substr( SA[pos], MAX ), cnt );
}
// 3 - Dada uma string s, retorna o numero de substrings distintas em s
// Solução: contar o numero de nos com excesso da raiz na suffix tree. Isso pode
// ser feito através da representação em suffix array
int number_of_substrings( const string & s ) {
    int asw = 0;
    initialize( s ); suffix_array( v, SA, N, CUR_ALF ); lcp( s );
    asw += N - SA[0];
    for(i,N-1) asw += ( N - SA[i+1] ) - LCP[i];
    return asw;
}
// 4 - Dada uma string s, retorna o numero de substrings distintas em s que
// aparecem pelo menos duas vezes
int number_of_substrings_twice( const string & s ) {
    int asw = 0, last = 0;
    initialize( s ); suffix_array( v, SA, N, CUR_ALF ); lcp( s );
    for(i,N-1) if ( LCP[i] > last ) asw += LCP[i] - last; last = LCP[i];
    return asw;
}
// 5 - Dado um vetor de strings vs, retorna a maior substring comum de todas as
// strings armazenadas em vs
int longest_common_substring( const vector< string > & vs ) {
    int asw = 0, cnt = 0, num_words = vs.sz, end = 0, sum = 0, aux = 0;
    char c = '!';
    string s = "";
    vector< int > init, occur( num_words, 0 );
    for(i,num_words) {
        for(j,vs[i].sz) s.append( 1, c );
        for(j,vs[i].sz) init.pb( cnt );
        lcp( s );
        init.pb( cnt );
        aux++;
        c++;
    }
    initialize( s );
    suffix_array( v, SA, N, CUR_ALF );
    lcp( s );
    // o intervalo [start,end] deve conter pelo menos um sufixo
    // de cada uma das strings de vs
    for(i,start,N-1) {
        if ( start != 0 ) {
            occur[init[SA[start-1]]]--;
            if ( occur[init[SA[start-1]]] == 0 ) sum--;
        }
        while ( sum != num_words && end < N ) {
            if ( occur[init[SA[end]]] == 0 ) occur[init[SA[end]]] = 1; sum++;
            else occur[init[SA[end]]]++;
        }
        if ( sum == num_words && ( aux = lcp( start, end - 1 ) ) > asw ) asw = aux;
    }
    return asw;
}

```

```

    /**
     * String matching - Algoritmo KMP - O(n+m)
     * // F[i] - size of the largest prefix of pattern[0..i] that is also a
     * // suffix of pattern[1..i]. Ex: pattern = {a,b,a,c,a,b}, F = {0,0,1,0,1,2}
     */
    void build_failure_function( const string & pattern ) {
        int m = pattern.size();
        F[0] = -1;
        for(i,m) {
            F[i+1] = F[i] + 1;
            if( i+1 > 0 && pattern[i] != pattern[ F[i+1]-1 ] )
                F[i+1] = F[ F[i+1]-1 ] + 1;
        }
    }

    /**
     * retorna a posicao inicial de cada ocorrencia de pattern em text
     */
    vector<int> KMP( const string & text, const string & pattern ) {
        build_failure_function( pattern );
        vector<int> start_positions;
        int j = 0, m = pattern.size(), n = text.size();
        for(i,n) while ( true ) {
            if ( text[i] == pattern[j] ) {
                if ( ++j == m ) { start_positions.push_back( i - m + 1 ); } break;
                if ( j == 0 ) break;
            }
        }
        return start_positions;
    }

    /**
     * String matching - Algoritmo aho-corasick
     */
    const int NULO = -1, MAX_NO = 100010, MAX_PAD = 10010;
    typedef map<char, int> mapach;
    map<string, int> mapastr;
    struct automato {
        mapach trans[MAX_NO];
        mapastr pad;
        list<int> pos[MAX_PAD];
        int falha[MAX_NO], final[MAX_NO], tam[MAX_PAD], numNos;
        automato(): numNos(0) {}

        // Funcao que adiciona padroes ao automato. 1 chamada por instancia, antes das outras funcoes
        void init() {
            memset(falha, NULO, sizeof(falha));
            memset(final, NULO, sizeof(final));
            for(i,numNos) trans[i].clear();
            pad.clear(); numNos = 1;
        }

        // Funcao que adiciona um padrao ao automato. Uma chamada por padrao, depois
        // da inicializacao. Retorna o ind. de acesso a variavel global pos.
        int adiciona_padrao(const char* s) {
            Pair<mapach:iterator, bool> pch;
            if ( pad.count(s) ) return pad[s];
            else pad.insert(make_pair(s, numPad));
            for (i = 0; s[i]; i++) {
                if ( (pch = trans[numPad].insert(make_pair(s[i], numNos))).second ) numNos++;
                no = pch.first->second;
                numPad = no;
            }
            tan[numPad] = i ? i : 1;
            return finalno;
        }

        // Funcao que gera o tratamento de falhas.
        void gera_falhas() {
            for(i,fila) {
                fila.pop();
                if( !fila.empty() ) {
                    tr(it, trans[0].begin(), trans[0].end()) {
                        fila.push_back(it);
                    }
                }
            }
        }
    };

```

```

        while ( !fila.empty() ) {
            int atual = fila.front(); fila.pop();
            tr(it, trans[atual].begin(), trans[atual].end());
            char c = it->first; filho = it->second;
            int ret = falha[atual];
            while ( ret != NULO ) {
                if ( ret != NULO ) {
                    falha[filho] = trans[ret][c];
                    if ( final[filho] == NULO && final[falha[filho]] != NULO )
                        final[filho] = final[falha[filho]];
                }
                else if ( trans[0].count(c) ) falha[filho] = trans[0][c];
                fila.push_back(filho);
            }
        }

        /**
         * Funcao que busca os padroes em uma texto de consulta.
         */
        // Uma chamada por consulta, depois da geracao do tratamento de falhas.
        // Preenche a variavel global pos com posicoes iniciais de cada padrao.
        void consulta(const char* s) {
            int ret, atual = 0, i = 0;
            int N = pad.size();
            for (int j = 0; j < N; j++) pos[j].clear();
            do {
                while( atual != NULO && !trans[atual].count(s[i]) ) atual = falha[atual];
                if ( atual == NULO ) ? 0 : trans[atual][s[i]];
                for (ret = atual; ret != NULO && final[ret] != NULO; ret = falha[ret]) {
                    pos[final[ret]].push_back(1 - tam[final[ret]] + 1);
                }
                while ( (falha[ret] != NULO && final[falha[ret]] == final[ret]) ||
                        ret == falha[ret] ) {
                    i += 1;
                }
            } while ( s[i++]);
        }

        /**
         * funcao main para algoritmo aho-corasick
         */
        // IMPORTANTE: usar "aut.pad[patterns[j]]" para indexar padrao j
        // chamar 'nasta ordem: inici, adiciona_padrao, gera_falhas, consulta
        int main() {
            int main() {
                string text, pattern;
                cin >> k; // numero de instancias
                for(i,k) {
                    cin >> text >> q; // texto, numero de padroes
                    vector<string> patterns( q ); // padroes a serem pesquisados
                    automato aut; aut.inic(); // cria e inicia automato
                    for(j,q) {
                        cin >> pattern;
                        patterns[j] = pattern;
                        aut.adiciona_padrao(pattern.c_str()); // adiciona padrao
                        aut.gera_falhas(); // tratamento de falhas
                        aut.consulta(text.c_str()); // realiza pesquisa em todos
                    }
                    for(j,q) {
                        // imprime se o j-esimo padrao existe ou nao no texto
                        if( aut.pos[ aut.pad[patterns[j]] ].empty() ) puts("n");
                        else cout << "y" << endl;
                    }
                    //for(j,q) {
                    //cout << patterns[j] << ":" << endl;
                    //tr(it, aut.pos[ aut.pad[patterns[j]] ].begin(),
                    //   aut.pos[ aut.pad[patterns[j]] ].end() ) cout << *it << " ";
                    //}
                    cout << endl;
                }
            }
            return 0;
        }
    };

```

Parte VI

Physical Cheat

FORMULAS DE FÍSICA

ARMANDO CRUZ Versão RC

CINEMÁTICA	
Grandezas Básicas $V/m = \frac{\Delta x}{\Delta t}$ $a = \frac{\Delta v}{\Delta t}$	Queda livre $h_{max} = \frac{v_0^2}{2g}$ $t_{h,max} = \frac{v_0}{g}$ $A = \frac{v_0^2 \operatorname{sen}(2\theta)}{g}$
M.R.U.V. $x = x_0 + v_0 t + \frac{at^2}{2}$ $v^2 = v_0^2 + 2a\Delta x$	$\varphi = \varphi_0 + \omega_0 t + \frac{at^2}{2}$ $\omega^2 = \omega_0^2 + 2a\Delta\varphi$
M.C.U. $\dot{\varphi} = \frac{s}{R}$ $\omega = \frac{v}{R}$ $\alpha = \frac{a}{r}$ $\omega_m = \frac{\Delta\varphi}{\Delta t}$ $\alpha_c = \frac{\Delta\omega}{t}$	Acoplamento de polias $f_a R_a = f_b R_b$ $\omega_a R_a = \omega_b R_b$ Por eixo $\omega_a = \omega_b$ $\frac{v_a}{R_a} = \frac{v_b}{R_b}$
DINÂMICA	
2ª Lei de Newton $\vec{F}_r = m\vec{a}$	Energia cinética $E_c = \frac{mv^2}{2}$
Lei de Hooke $F = kx$	Gravitação Força Gravitacional $F = G \frac{m_1 m_2}{d^2}$ 3º lei de Kepler $T^2 = kr^3$
Força de atrito $F_{fric} \leq \mu_e N$ $F_{fric} = \mu_e N$	Energia potencial gravitacional $E_{pg} = mg\ell$
Momento de uma força (Torque) $M = Fd$	Energia potencial elástica $E_{pe} = \frac{kx^2}{2}$
Resultante centrípeta $F_p = ma_c$ $a_c = \frac{v^2}{R}$	Fluidos Velo. de um satélite $v = \sqrt{\frac{GM}{r}}$
Quantidade de movimento $\vec{Q} = mv$	Pressão $p = \frac{F}{s}$
Trabalho $\vec{T} = F\Delta x \cdot \cos\theta$ $\vec{T} = \Delta E$	Densidade ou massa específica $\mu = \frac{m}{V}$
Potência mecânica $\mathcal{P} = \frac{\mathcal{J}}{\Delta t} = \frac{F\Delta x}{\Delta t} = FV$	Impulso $\vec{I} = \vec{F}\Delta t$ $\vec{i} = \Delta \vec{Q}$
Rendimento $\eta = \frac{P_u}{P_t}$	Vasos comunicantes $\mu_a h_a = \mu_b h_b$
TERMODINÂMICA	
Termometria Fusão Ebulição	Leis da termodinâmica Capacidade calorífica $C = \frac{\partial Q}{\partial T}$ Transformação: Isobárica $\vec{T} = p \cdot \Delta V$ Isotérmica $\Delta E_i = \Delta Q$ Adiabáticas $\Delta E_i = -\mathcal{T}$ Cíclica $\vec{T} = \Delta Q = Q_1 - Q_2$
Queda livre $h_{max} = \frac{v_0^2}{2g}$ $t_{h,max} = \frac{v_0}{g}$ $A = \frac{v_0^2 \operatorname{sen}(2\theta)}{g}$	Calor específico $c = \frac{C}{m} = \frac{Q}{m\Delta T}$ Transferência de calor por condução $\phi = \frac{\Delta Q}{\Delta t} = k A(T_2 - T_1)$ Lei Geral dos gases perfeitos $\frac{P_0 V_0}{T_0} = \frac{P V}{T}$
Acoplamento de polias $f_a R_a = f_b R_b$ $\omega_a R_a = \omega_b R_b$	Dilatação Linear $\Delta L = \alpha L_0 \Delta T$ Superficial $\Delta S = \beta S_0 \Delta T$ Volumétrica $\Delta V = \gamma V_0 \Delta T$ $\gamma \cong 3\alpha$
Por eixo $\omega_a = \omega_b$ $\frac{v_a}{R_a} = \frac{v_b}{R_b}$	Calorimetria Calor latente $L = \frac{Q}{m}$ Equação de Clapeyron $\frac{PV}{T} = nR$
ÓPTICA	
Movimento harmônico simples $x = A \cos(\omega t + \theta_0)$ $v = -\omega A \operatorname{sen}(\omega t + \theta_0)$ $a = -\omega^2 x$	Índice de refração $n_{2,1} = v_1/v_2$ Lei de Snell-Descartes $\frac{\operatorname{sen} i}{\operatorname{sen} r} = \frac{n_2}{n_1} = \frac{v_1}{v_2} = \frac{\lambda_1}{\lambda_2}$
Velocidade das ondas $v = \frac{\lambda}{T}$	Lâmina de faces paralelas $d = e \frac{\sin(\hat{i} - \hat{r})}{\cos \hat{r}}$ Desvio produzido por um prisma $\alpha = \hat{i} + \hat{r} - \hat{A}$
Ampliação da imagem $A = \frac{y'}{y} = -\frac{x'}{x}$	Reflexão interna total $\operatorname{sen} \hat{L} = \frac{n_2}{n_1}$ Convergência ou vergência $V = \frac{1}{f}$
ONDULATORIA	
Movimento harmônico simples $x = A \cos(\omega t + \theta_0)$ $v = -\omega A \operatorname{sen}(\omega t + \theta_0)$ $a = -\omega^2 x$	Cordas e tubos sonoros Frequência de uma corda ou tubo sonoro $f = n \cdot f_n$ corda ou tubo sonoro aberto $f_n = \frac{v}{2L}$ tubo sonoro fechado $f_n = \frac{v}{4L}$
Velocidade angular de um sistema massa mola $\omega = \sqrt{k/m}$	Efeito Doppler $f = f_0 \left(\frac{v \pm v_r}{v \pm v_f} \right)$ Experiência de Young $\lambda = d/L$
Centro de massa $X_{CM} = \frac{m_1 x_1 + m_2 x_2 + \dots + m_n x_n}{m_1 + m_2 + \dots + m_n}$	Empuxo $E = \mu_U V/g$
Nível sonoro $\beta = 10 \log \frac{i}{i_0}$	Empuxo $E = \mu_U V/g$

MOVIMENTO HARMÔNICO SIMPLES	
Potência mecânica $\mathcal{P} = \frac{\mathcal{J}}{\Delta t} = \frac{F\Delta x}{\Delta t} = FV$	Velocidade das ondas $v = \frac{\lambda}{T}$
Rendimento $\eta = \frac{P_u}{P_t}$	Cordas e tubos sonoros Frequência de uma corda ou tubo sonoro $f = n \cdot f_n$ corda ou tubo sonoro aberto $f_n = \frac{v}{2L}$ tubo sonoro fechado $f_n = \frac{v}{4L}$
ACÚSTICA	
Centro de massa $X_{CM} = \frac{m_1 x_1 + m_2 x_2 + \dots + m_n x_n}{m_1 + m_2 + \dots + m_n}$	Intensidade sonora $i = \frac{\Delta E}{S \cdot \Delta t} = \frac{\mathcal{P}}{S}$
Empuxo $E = \mu_U V/g$	Nível sonoro $\beta = 10 \log \frac{i}{i_0}$
Velocidade angular de um pendulo massa mola $\omega = \sqrt{k/m}$	Experiência de Young $\lambda = d/L$

ELETROSTÁTICA

Carga elétrica de um corpo $Q = n \cdot e$	Energia potencial elétrica $E_{pe} = k \frac{Q_1 \cdot Q_2}{x}$	Capacitância $C = \frac{Q}{U}$
Lei de Coulomb $F = k \frac{ Q_1 \cdot Q_2 }{x^2}$		$V = \frac{Q_1 + Q_2 + \dots + Q_n}{C_1 + C_2 + \dots + C_n}$
 vetor intensidade campo elétrico $\vec{E} = \frac{\vec{F}}{q} = \frac{kQ}{x^2}$		Em um condutor Estérico $C_{est} = R/k$
Trabalho da força elétrica $T_{AB} = Uq$		Energia elétrica armazenada $E_{pe} = QU/2$
ddp em campo elétrico uniforme $U = ED$		Capacitor de placas paralelas $C = \epsilon \frac{A}{D}$ $\bar{E} = \frac{Q}{\epsilon A}$
Aquecimento por efeito Joule $Q = i^2 \cdot R \cdot \Delta t$		
Potência elétrica $\bar{P} = UI$		

Unidades do SI

Unidades fundamentais			
Grandezas	Unidade	Símbolo	Observações e definições (simplificado)
Comprimento	metro	m	Comprimento percorrido pela luz no vácuo no intervalo de 1/299 792 458 segundos.
Massa	quilograma	kg	Massa do protótipo internacional
Tempo	segundo	s	Duração de 9 192 631 770 períodos da radiação correspondente à transição entre dois níveis hiperfino do átomo de césio 133
Corrente elétrica	ampère	A	Corrente mantida em dois condutores paralelos, situados no vácuo a 1 metro de distância um do outro, que produz uma força entre esses condutores igual a $2 \cdot 10^{-7}$ newtons.
Temperatura	kelvin	K	Fração 1/273,16 da temperatura termodinâmica do ponto triplice da água.
Quantidade de matéria	mol	mol	Quantidade de matéria contida em 0,012 kg de carbono 12. Equivalendo a $6,02 \cdot 10^{23}$
Intensidade luminosa	candela	cd	Intensidade luminosa de uma fonte emissora de radiação monocromática na frequência de 540 1012 hertz, com uma intensidade energética de 1/683 watts por esteroradiano.
Unidades derivadas			
Área	metro quadrado	m^2	
Volume	metro cúbico	m^3	
Ângulo	radianos	rad	
Densidade	quilograma por m^3	kg/m^3	
Velocidade	metro por segundo	m/s	
Aceleração	metro por s^2	m/s^2	
Força	newton	N	$1N = 1kg \cdot m/s^2$
Pressão	pascal	Pa	N/m^2
Trabalho, energia...	joule	J	$1J = N \cdot m$
Potência	watt	W	$W = J/s$ ou $W = N \cdot v$
Intensidade sonora	potencia por área	W/m^2	
Nível sonoro	decibéis	dB	
Frequência	hertz	Hz	Quantidade de ciclos em um segundo (s^{-1})
Convergência ou vergencia	dioptria	di	$di = m^{-1}$
Carga elétrica	coulomb	C	
Diferença de potencial (ddp)	volt	V	$1/C$
Capacitância	farad	F	C/V
Resistência elétrica	ohm	Ω	V/A
Fluxo magnético	weber	Wb	$1Wb = 1T \cdot m^2$
Indução magnética	tesla	T	$1T = 1N/(C \cdot m/s)$ ou $1N/(A \cdot m)$

PRINCIPAIS RELAÇÕES MATEMÁTICAS E DE UNIDADES

$\frac{1}{s} = 3,6 \frac{km}{h}$	$1\ell = dm^3$	Carga elétrica de um elétron (e) $e = 1,6 \cdot 10^{-19} C$
$dm^3 \xrightarrow{\times 1000} m^3 \xleftarrow{\div 1000} dm^3$		Constante universal dos gases (R) $8,31 \frac{J}{mol \cdot K} = 0,082 \frac{atm \cdot l}{mol \cdot K} = 2 \frac{cal}{mol \cdot K}$
$1atm = 760mmHg \cong 10^5 N/m^2$		Teorema do paralelogramo $a^2 = b^2 + c^2 + 2bc \cdot \cos \alpha$
$1cal = 4,186J$		Indução magnética tesla

CONSTANTES FÍSICAS

Constante	Símbolo	Valor para cálculo	Valor + (incerteza) + unidade
Velocidade da luz no vácuo	c	$3 \cdot 10^8 \text{ m/s}$	$2,997\,924\,58 (\text{exato})$
Carga elementar	e	$1,6 \cdot 10^{-19} \text{ C}$	$1,602\,177\,33(49) \cdot 10^{-19} \text{ C}$
Número de Avogadro	N_A	$6,02 \cdot 10^{23}$	$6,022\,136\,7(36) \cdot 10^{23}$
Constante da gravitação universal	G	$6,67 \cdot 10^{-11} \frac{\text{N} \cdot \text{m}^2}{\text{kg}^2}$	$6,672\,59(85) \cdot 10^{-11} \frac{\text{N} \cdot \text{m}^2}{\text{kg}^2}$
Permeabilidade elétrica do vácuo	ϵ_0	$8,8 \cdot 10^{-12} \frac{\text{C}^2}{\text{Nm}^2}$	$8,854\,187\,817 \cdot 10^{-12} \frac{\text{C}^2}{\text{Nm}^2} (\text{exato})$
Permeabilidade magnética do vácuo	μ_0	$4\pi \cdot 10^{-7} \frac{\text{T} \cdot \text{m}}{\text{A}}$	$4\pi \cdot 10^{-7} \frac{\text{T} \cdot \text{m}}{\text{A}} (\text{exato})$
Constante eletrostática do vácuo ou Constante de Coulomb	k_0	$9 \cdot 10^9 \frac{\text{N} \cdot \text{m}^2}{\text{C}^2}$	$8,987\,551\,787 \cdot 10^9 \frac{\text{N} \cdot \text{m}^2}{\text{C}^2} (\text{exato})$
Unidade de massa atômica	u	$1,66 \cdot 10^{-12} \text{ Kg}$	$1,660\,540\,2(10) \cdot 10^{-12} \text{ Kg}$
Constante dos gases	R	$8,31 \frac{\text{J}}{\text{mol} \cdot \text{K}}$	$8,314\,510(70) \frac{\text{J}}{\text{mol} \cdot \text{K}}$
Constante de Planck	h	$6,63 \cdot 10^{-34} \text{ J} \cdot \text{s}$	$6,626\,075(40) 10^{-34} \text{ J} \cdot \text{s} (\text{exato})$

SIGNIFICADOS E UNIDADES DAS FÓRMULAS

CINEMÁTICA

Símbolo	Significado	Unidade	Símbolo	Significado	Unidade
\vec{F}_r	Força resultante	N	Δx	Variação de posição	m
m	Massa	kg	E	Energia	J
\ddot{a}	Aceleração	m/s^2	P	Potência	W
p	Peso de um corpo	N	Δt	Variação de tempo	s
g	Aceleração da gravidade	m/s^2	η	Rendimento	*
F	Força	N	P_u	Potência útil	W
k	Coeficiente elástico da mola	N/m	P_t	Potência total	W
x	Elongação da mola	m	E_c	Energia cinética	J
F_{ae}	Força de atrito estático	N	E_{pg}	Energia potencial gravitacional	J
μ_e	Coeficiente de atrito estático	*	h	Altura	m
F_{ac}	Força de atrito cinético	N	E_{pe}	Energia potencial elástica	J
μ_c	Coeficiente de atrito cinético	*	E_m	Energia mecânica	J
N	Força normal	N	\vec{Q}	Quantidade de movimento	$\text{kg} \cdot \text{m/s}$
M	Momento de uma força	$\text{N} \cdot \text{m}$	\vec{I}	Impulso	$\text{N} \cdot \text{s}$
d	Distância	m	e	Coeficiente de restituição	*
F_{cp}	Força centrípeta	N	V_{af}	Velocidade de afastamento	m/s
a_c	Aceleração centrípeta	m/s^2	V_{ap}	Velocidade de aproximação	m/s
v	Velocidade	m/s	X_{cm}	Ponto do centro de m em x	m
R	Raio do círculo	m	$m_i; m_n$	Massa dos corpos	kg
\mathcal{T}	Trabalho	J	$x_1; x_n$	Posição em x dos corpos	m

DINÂMICA

Símbolo	Significado	Unidade	Símbolo	Significado	Unidade
\vec{F}_r	Força resultante	N	Δx	Variação de posição	m
m	Massa	kg	E	Energia	J
\ddot{a}	Aceleração	m/s^2	P	Potência	W
p	Peso de um corpo	N	Δt	Variação de tempo	s
g	Aceleração da gravidade	m/s^2	η	Rendimento	*
F	Força	N	P_u	Potência útil	W
k	Coeficiente elástico da mola	N/m	P_t	Potência total	W
x	Elongação da mola	m	E_c	Energia cinética	J
F_{ae}	Força de atrito estático	N	E_{pg}	Energia potencial gravitacional	J
μ_e	Coeficiente de atrito estático	*	h	Altura	m
F_{ac}	Força de atrito cinético	N	E_{pe}	Energia potencial elástica	J
μ_c	Coeficiente de atrito cinético	*	E_m	Energia mecânica	J
N	Força normal	N	\vec{Q}	Quantidade de movimento	$\text{kg} \cdot \text{m/s}$
M	Momento de uma força	$\text{N} \cdot \text{m}$	\vec{I}	Impulso	$\text{N} \cdot \text{s}$
d	Distância	m	e	Coeficiente de restituição	*
F_{cp}	Força centrípeta	N	V_{af}	Velocidade de afastamento	m/s
a_c	Aceleração centrípeta	m/s^2	V_{ap}	Velocidade de aproximação	m/s
v	Velocidade	m/s	X_{cm}	Ponto do centro de m em x	m
R	Raio do círculo	m	$m_i; m_n$	Massa dos corpos	kg
\mathcal{T}	Trabalho	J	$x_1; x_n$	Posição em x dos corpos	m

* Unidade adimensional

GRAVITAÇÃO E FLUIDOS

Símbolo	Significado	Unidade	Símbolo	Significado	Unidade
T	Período orbital	s	F	Força	N
k	Constante		S	Área da superfície	m^2
r	Raio médio da órbita	m	V	Volume do corpo	m^3
F_g	Força gravitacional	N	m	Massa do corpo	kg
$G^\#$	Constante de gravitação universal	$\frac{Nm^2}{kg^2}$	μ	Densidade	$\frac{kg}{m^3}$
M_1M_2	Massa dos corpos	kg	h	Altura	m
d	Distância dos corpos	m	E	Empuxo	N
v	Velocidade	m/s	P_r	Peso real	N
p	Pressão	Pa	P_a	Peso aparente	N

Quando aparecer o símbolo “ $\#$ ” ver a tabela: “CONSTANTES FÍSICAS”

TERMODINÂMICA

Símbolo	Significado	Unidade	Símbolo	Significado	Unidade
T	Temperatura	K	m	Massa	kg
T_{F_x}	Temperatura de fusão de “x”	K	C	Capacidade calorífica	J/K
T_{F_x}	Temperatura de ebulição de “x”	K	c	Calor específico	J/kg K
α	Coeficiente de dilatação linear	K^{-1}	P	Pressão	Pa
β	C. de dilatação superficial	K^{-1}	n	Quantidade de <i>mol</i> do gás	mol
γ	C. de dilatação volumétrica	K^{-1}	$R^\#$	Constante universal dos gases	J/mol K
L	Comprimento	m	ϕ	Fluxo de calor por condução	J/s
S	Superfície	m^2	k	Coeficiente de condutibilidade	$J/s \cdot m \cdot K$
V	Volume	m^3	x	Distância	m
ΔV_r	Variação de volume real	m^3	A	Área da secção transversal	m^2
ΔV_{ap}	Varição de volume aparente	m^3	J	Trabalho	
ΔV_{rec}	Variação de volume do recipiente	m^3	E_i	Energia interna	J
L	Calor latente	J/K	η	Rendimento	*
Q	Quantidade de energia	J			

** Número natural (N)

ELETROSTÁTICA

Símbolo	Significado	Unidade	Símbolo	Significado	Unidade
Q	Carga elétrica de um corpo	C	V	Potencial elétrico	V
n	Nº de cargas “e” em excesso	N	T_{AB}	Trabalho de A para B	V
$e^\#$	Carga elétrica do elétron	C	U	Diferença de potencial (ΔV)	V
F	Força	N	C	Capacitância	F
$k^\#$	Constante eletrostática do meio	$N \cdot m^2/C^2$	R	Raio da esfera	m
$Q_1 e Q_2$	Cargas dos corpos	C	E_{pe}	Energia potencial elétrica	J
x	Distância	m	$\epsilon^\#$	Permitividade do meio	N/m
E	Vetor campo elétrico	N/C	A	Área das armaduras	m^2
E_{pe}	Energia potencia elétrica	J	D	Distância entre as armaduras	m

ELETRODINÂMICA

Símbolo	Significado	Unidade	Símbolo	Significado	Unidade
i	Intensidade da corrente elétrica	C/s	P	Potência elétrica	W
Δq	Quantidade de cargas	C	E_{ele}	Energia elétrica	$Ws = J$
t	Tempo	s	ξ	Força eletromotriz	V
f	Foco	sen \hat{i}	Seno do ângulo de incidência	rad	
x	Posição relativa ao eixo “x”	m	sen \hat{r}	Seno do ângulo refletido	rad
y	Posição relativa ao eixo “y”	m	sen \hat{L}	Seno do ângulo limite de refração	rad
n	Índice de refração	*	d	Desvio do raios luminoso	m
λ	Comprimento de onda	m	e	Espessura da lâmina	m
v_1	Velocidade da onda incidente	m/s	α	Desvio produzido por um prisma	rad
v_2	Velocidade da onda refratada	m/s	V	Convergência ou vergencia	di

ONDULATORIA

Símbolo	Significado	Unidade	Símbolo	Significado	Unidade
x	Elongação	m	μ_t	Densidade linear da corda	kg/m
A	Amplitude	m	B	Constante do meio	
θ	Ângulo	rad	n	Número de harmônicos	N^{**}
ω	Velocidade angular	rad/s	f_n	Frequência natural	Hz
t	Tempo	s	v	Velocidade da onda	m/s
v	Velocidade	m/s	F	Força	N
a	Aceleração	m/s^2	v_r	Velocidade do receptor	m/s
m	Massa	kg	v_f	Velocidade da fonte	m/s
k	Constante	i	I	Intensidade sonora	W/m^2
L	Comprimento	m	ΔE	Variiação de energia	J
g	Gravidade	m/s^2	S	Superfície	m^2
μ	Densidade do ar	kg/m^3	β	Nível sonoro	dB

MAGNÉTISMO

MAGNÉTISMO					
Símbolo	Significado	Unidade	Símbolo	Significado	Unidade
B	Intensidade do campo magnético	T	q	Carga do corpo	C
i	Intensidade da corrente elétrica	A	v	Velocidade do corpo	m/s
$\mu^{\#}$	Permeabilidade magnética do meio	$T \cdot m/A$	θ	Ângulo entre \vec{B} e o \vec{v}	rad
r	Raio da circunferência	m	d	Distância entre os fios	m
N	Numero de espiras	N	Φ	Fluxo magneto	Wb
l	Comprimento	m	A	Área da superfície	m^2
F	Força magnética	N	ε	Força eletromotriz induzida	V

Parte VII

Math Cheat

Theoretical Computer Science Cheat Sheet

Definitions		Series
$f(n) = O(g(n))$	iff \exists positive c, n_0 such that $0 \leq f(n) \leq cg(n) \forall n \geq n_0$.	$\sum_{i=1}^n i = \frac{n(n+1)}{2}, \quad \sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}, \quad \sum_{i=1}^n i^3 = \frac{n^2(n+1)^2}{4}$. In general: $\sum_{i=1}^n i^m = \frac{1}{m+1} \left[(n+1)^{m+1} - 1 - \sum_{i=1}^n ((i+1)^{m+1} - i^{m+1} - (m+1)i^m) \right]$ $\sum_{i=1}^{n-1} i^m = \frac{1}{m+1} \sum_{k=0}^m \binom{m+1}{k} B_k n^{m+1-k}.$
$f(n) = \Omega(g(n))$	iff \exists positive c, n_0 such that $f(n) \geq cg(n) \geq 0 \forall n \geq n_0$.	
$f(n) = \Theta(g(n))$	iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.	
$f(n) = o(g(n))$	iff $\lim_{n \rightarrow \infty} f(n)/g(n) = 0$.	
$\lim_{n \rightarrow \infty} a_n = a$	iff $\forall \epsilon > 0, \exists n_0$ such that $ a_n - a < \epsilon, \forall n \geq n_0$.	
$\sup S$	least $b \in \mathbb{R}$ such that $b \geq s, \forall s \in S$.	
$\inf S$	greatest $b \in \mathbb{R}$ such that $b \leq s, \forall s \in S$.	
$\liminf_{n \rightarrow \infty} a_n$	$\liminf_{n \rightarrow \infty} \{a_i \mid i \geq n, i \in \mathbb{N}\}$.	
$\limsup_{n \rightarrow \infty} a_n$	$\limsup_{n \rightarrow \infty} \{a_i \mid i \geq n, i \in \mathbb{N}\}$.	
$\binom{n}{k}$	Combinations: Size k subsets of a size n set.	
$[n]_k$	Stirling numbers (1st kind): Arrangements of an n element set into k cycles.	1. $\binom{n}{k} = \frac{n!}{(n-k)!k!}, \quad 2. \sum_{k=0}^n \binom{n}{k} = 2^n, \quad 3. \binom{n}{k} = \binom{n}{n-k}$,
$\{n\}_k$	Stirling numbers (2nd kind): Partitions of an n element set into k non-empty sets.	4. $\binom{n}{k} = \frac{n}{k} \binom{n-1}{k-1}, \quad 5. \binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1},$
$\langle n \rangle_k$	1st order Eulerian numbers: Permutations $\pi_1 \pi_2 \dots \pi_n$ on $\{1, 2, \dots, n\}$ with k ascents.	6. $\binom{n}{m} \binom{m}{k} = \binom{n}{k} \binom{n-k}{m-k}, \quad 7. \sum_{k=0}^n \binom{r+k}{k} = \binom{r+n+1}{n},$
$\langle\langle n \rangle\rangle_k$	2nd order Eulerian numbers.	8. $\sum_{k=0}^n \binom{k}{m} = \binom{n+1}{m+1}, \quad 9. \sum_{k=0}^n \binom{r}{k} \binom{s}{n-k} = \binom{r+s}{n},$
C_n	Catalan Numbers: Binary trees with $n+1$ vertices.	10. $\binom{n}{k} = (-1)^k \binom{k-n-1}{k}, \quad 11. \begin{Bmatrix} n \\ 1 \end{Bmatrix} = \begin{Bmatrix} n \\ n \end{Bmatrix} = 1,$
14. $\begin{Bmatrix} n \\ 1 \end{Bmatrix} = (n-1)!$,	15. $\begin{Bmatrix} n \\ 2 \end{Bmatrix} = (n-1)!H_{n-1}$,	16. $\begin{Bmatrix} n \\ n \end{Bmatrix} = 1, \quad 17. \begin{Bmatrix} n \\ k \end{Bmatrix} \geq \begin{Bmatrix} n \\ k \end{Bmatrix},$
18. $\begin{Bmatrix} n \\ k \end{Bmatrix} = (n-1) \begin{Bmatrix} n-1 \\ k \end{Bmatrix} + \begin{Bmatrix} n-1 \\ k-1 \end{Bmatrix}, \quad 19. \begin{Bmatrix} n \\ n-1 \end{Bmatrix} = \begin{Bmatrix} n \\ n-1 \end{Bmatrix} = \binom{n}{2}, \quad 20. \sum_{k=0}^n \begin{Bmatrix} n \\ k \end{Bmatrix} = n!, \quad 21. C_n = \frac{1}{n+1} \binom{2n}{n},$		
22. $\begin{Bmatrix} n \\ 0 \end{Bmatrix} = \begin{Bmatrix} n \\ n-1 \end{Bmatrix} = 1, \quad 23. \begin{Bmatrix} n \\ k \end{Bmatrix} = \begin{Bmatrix} n \\ n-1-k \end{Bmatrix}, \quad 24. \begin{Bmatrix} n \\ k \end{Bmatrix} = (k+1) \begin{Bmatrix} n-1 \\ k \end{Bmatrix} + (n-k) \begin{Bmatrix} n-1 \\ k-1 \end{Bmatrix},$		
25. $\begin{Bmatrix} 0 \\ k \end{Bmatrix} = \begin{cases} 1 & \text{if } k=0, \\ 0 & \text{otherwise} \end{cases}$,	26. $\begin{Bmatrix} n \\ 1 \end{Bmatrix} = 2^n - n - 1, \quad 27. \begin{Bmatrix} n \\ 2 \end{Bmatrix} = 3^n - (n+1)2^n + \binom{n+1}{2},$	
28. $x^n = \sum_{k=0}^n \begin{Bmatrix} n \\ k \end{Bmatrix} \binom{x+k}{n}, \quad 29. \begin{Bmatrix} n \\ m \end{Bmatrix} = \sum_{k=0}^m \binom{n+1}{k} (m+1-k)^n (-1)^k, \quad 30. m! \begin{Bmatrix} n \\ m \end{Bmatrix} = \sum_{k=0}^n \begin{Bmatrix} n \\ k \end{Bmatrix} \binom{k}{n-m},$		
31. $\begin{Bmatrix} n \\ m \end{Bmatrix} = \sum_{k=0}^n \begin{Bmatrix} n \\ k \end{Bmatrix} \binom{n-k}{m} (-1)^{n-k-m} k!, \quad 32. \begin{Bmatrix} n \\ 0 \end{Bmatrix} = 1, \quad 33. \begin{Bmatrix} n \\ n \end{Bmatrix} = 0 \quad \text{for } n \neq 0,$		
34. $\begin{Bmatrix} n \\ k \end{Bmatrix} = (k+1) \begin{Bmatrix} n-1 \\ k \end{Bmatrix} + (2n-1-k) \begin{Bmatrix} n-1 \\ k-1 \end{Bmatrix}, \quad 35. \sum_{k=0}^n \begin{Bmatrix} n \\ k \end{Bmatrix} = \frac{(2n)^n}{2^n},$		
36. $\begin{Bmatrix} x \\ x-n \end{Bmatrix} = \sum_{k=0}^n \begin{Bmatrix} n \\ k \end{Bmatrix} \binom{x+n-1-k}{2n}, \quad 37. \begin{Bmatrix} n+1 \\ m+1 \end{Bmatrix} = \sum_k \binom{n}{k} \binom{k}{m} = \sum_{k=0}^n \binom{k}{m} (m+1)^{n-k},$		

Theoretical Computer Science Cheat Sheet

Identities Cont.		Trees
38. $\begin{bmatrix} n+1 \\ m+1 \end{bmatrix} = \sum_k \begin{bmatrix} n \\ k \end{bmatrix} \binom{k}{m} = \sum_{k=0}^n \begin{bmatrix} k \\ m \end{bmatrix} n^{n-k} = n! \sum_{k=0}^n \frac{1}{k!} \binom{k}{m}$,	39. $\begin{bmatrix} x \\ x-n \end{bmatrix} = \sum_{k=0}^n \begin{Bmatrix} n \\ k \end{Bmatrix} \binom{x+k}{2n}$,	Every tree with n vertices has $n-1$ edges.
40. $\begin{Bmatrix} n \\ m \end{Bmatrix} = \sum_k \binom{n}{k} \begin{Bmatrix} k+1 \\ m+1 \end{Bmatrix} (-1)^{n-k}$,	41. $\begin{bmatrix} n \\ m \end{bmatrix} = \sum_k \begin{bmatrix} n+1 \\ k+1 \end{bmatrix} \binom{k}{m} (-1)^{m-k}$,	Kraft inequality: If the depths of the leaves of a binary tree are d_1, \dots, d_n : $\sum_{i=1}^n 2^{-d_i} \leq 1,$
42. $\begin{Bmatrix} m+n+1 \\ m \end{Bmatrix} = \sum_{k=0}^m k \begin{Bmatrix} n+k \\ k \end{Bmatrix}$,	43. $\begin{bmatrix} m+n+1 \\ m \end{bmatrix} = \sum_{k=0}^m k(n+k) \binom{n+k}{k}$,	and equality holds only if every internal node has 2 sons.
44. $\binom{n}{m} = \sum_k \begin{Bmatrix} n+1 \\ k+1 \end{Bmatrix} \binom{k}{m} (-1)^{m-k}$, 45. $(n-m)! \binom{n}{m} = \sum_k \begin{Bmatrix} n+1 \\ k+1 \end{Bmatrix} \binom{k}{m} (-1)^{m-k}$, for $n \geq m$,	46. $\begin{Bmatrix} n \\ n-m \end{Bmatrix} = \sum_k \binom{m-n}{m+k} \binom{m+n}{n+k} \binom{m+k}{k}$,	47. $\begin{bmatrix} n \\ n-m \end{bmatrix} = \sum_k \binom{m-n}{m+k} \binom{m+n}{n+k} \binom{m+k}{k}$,
48. $\begin{Bmatrix} n \\ \ell+m \end{Bmatrix} \binom{\ell+m}{\ell} = \sum_k \begin{Bmatrix} k \\ \ell \end{Bmatrix} \begin{Bmatrix} n-k \\ m \end{Bmatrix} \binom{n}{k}$,	49. $\begin{bmatrix} n \\ \ell+m \end{bmatrix} \binom{\ell+m}{\ell} = \sum_k \begin{Bmatrix} k \\ \ell \end{Bmatrix} \begin{Bmatrix} n-k \\ m \end{Bmatrix} \binom{n}{k}$.	
Recurrences		
<p>Master method: $T(n) = aT(n/b) + f(n)$, $a \geq 1, b > 1$</p> <p>If $\exists \epsilon > 0$ such that $f(n) = O(n^{\log_b a - \epsilon})$ then $T(n) = \Theta(n^{\log_b a})$.</p> <p>If $f(n) = \Theta(n^{\log_b a})$ then $T(n) = \Theta(n^{\log_b a} \log_2 n)$.</p> <p>If $\exists \epsilon > 0$ such that $f(n) = \Omega(n^{\log_b a + \epsilon})$, and $\exists c < 1$ such that $af(n/b) \leq cf(n)$ for large n, then $T(n) = \Theta(f(n))$.</p> <p>Substitution (example): Consider the following recurrence</p> $T_{i+1} = 2^{2^i} \cdot T_i^2, \quad T_1 = 2.$ <p>Note that T_i is always a power of two. Let $t_i = \log_2 T_i$. Then we have $t_{i+1} = 2^i + 2t_i, \quad t_1 = 1.$</p> <p>Let $u_i = t_i/2^i$. Dividing both sides of the previous equation by 2^{i+1} we get</p> $\frac{t_{i+1}}{2^{i+1}} = \frac{2^i}{2^{i+1}} + \frac{t_i}{2^i}.$ <p>Substituting we find $u_{i+1} = \frac{1}{2} + u_i, \quad u_1 = \frac{1}{2},$</p> <p>which is simply $u_i = i/2$. So we find that T_i has the closed form $T_i = 2^{i2^{i-1}}$.</p> <p>Summing factors (example): Consider the following recurrence $T(n) = 3T(n/2) + n, \quad T(1) = 1.$</p> <p>Rewrite so that all terms involving T are on the left side $T(n) - 3T(n/2) = n.$</p> <p>Now expand the recurrence, and choose a factor which makes the left side “telescope”</p>	$\begin{aligned} 1(T(n) - 3T(n/2) = n) \\ 3(T(n/2) - 3T(n/4) = n/2) \\ \vdots \quad \vdots \quad \vdots \\ 3^{\log_2 n-1}(T(2) - 3T(1) = 2) \end{aligned}$ <p>Let $m = \log_2 n$. Summing the left side we get $T(n) - 3^m T(1) = T(n) - 3^m = T(n) - n^k$ where $k = \log_2 3 \approx 1.58496$. Summing the right side we get</p> $\sum_{i=0}^{m-1} \frac{n}{2^i} 3^i = n \sum_{i=0}^{m-1} \left(\frac{3}{2}\right)^i.$ <p>Let $c = \frac{3}{2}$. Then we have</p> $\begin{aligned} n \sum_{i=0}^{m-1} c^i &= n \left(\frac{c^m - 1}{c - 1} \right) \\ &= 2n(c^{\log_2 n} - 1) \\ &= 2n(c^{(k-1)\log_2 n} - 1) \\ &= 2n^k - 2n, \end{aligned}$ <p>and so $T(n) = 3n^k - 2n$. Full history recurrences can often be changed to limited history ones (example): Consider</p> $T_i = 1 + \sum_{j=0}^{i-1} T_j, \quad T_0 = 1.$ <p>Note that</p> $T_{i+1} = 1 + \sum_{j=0}^i T_j.$ <p>Subtracting we find</p> $\begin{aligned} T_{i+1} - T_i &= 1 + \sum_{j=0}^i T_j - 1 - \sum_{j=0}^{i-1} T_j \\ &= T_i. \end{aligned}$ <p>And so $T_{i+1} = 2T_i = 2^{i+1}$.</p>	<p>Generating functions:</p> <ol style="list-style-type: none"> Multiply both sides of the equation by x^i. Sum both sides over all i for which the equation is valid. Choose a generating function $G(x)$. Usually $G(x) = \sum_{i=0}^{\infty} x^i g_i$. Rewrite the equation in terms of the generating function $G(x)$. Solve for $G(x)$. The coefficient of x^i in $G(x)$ is g_i. <p>Example: $g_{i+1} = 2g_i + 1, \quad g_0 = 0.$</p> <p>Multiply and sum: $\sum_{i \geq 0} g_{i+1} x^i = \sum_{i \geq 0} 2g_i x^i + \sum_{i \geq 0} x^i.$</p> <p>We choose $G(x) = \sum_{i \geq 0} x^i g_i$. Rewrite in terms of $G(x)$:</p> $\frac{G(x) - g_0}{x} = 2G(x) + \sum_{i \geq 0} x^i.$ <p>Simplify: $\frac{G(x)}{x} = 2G(x) + \frac{1}{1-x}.$</p> <p>Solve for $G(x)$:</p> $G(x) = \frac{x}{(1-x)(1-2x)}.$ <p>Expand this using partial fractions:</p> $\begin{aligned} G(x) &= x \left(\frac{2}{1-2x} - \frac{1}{1-x} \right) \\ &= x \left(2 \sum_{i \geq 0} 2^i x^i - \sum_{i \geq 0} x^i \right) \\ &= \sum_{i \geq 0} (2^{i+1} - 1) x^{i+1}. \end{aligned}$ <p>So $g_i = 2^i - 1$.</p>

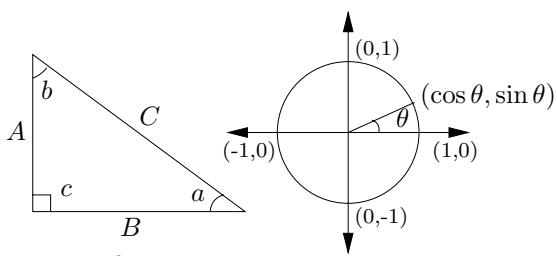
Theoretical Computer Science Cheat Sheet

$$\pi \approx 3.14159, \quad e \approx 2.71828, \quad \gamma \approx 0.57721, \quad \phi = \frac{1+\sqrt{5}}{2} \approx 1.61803, \quad \hat{\phi} = \frac{1-\sqrt{5}}{2} \approx -.61803$$

i	2^i	p_i	General	Probability
1	2	2	Bernoulli Numbers ($B_i = 0$, odd $i \neq 1$): $B_0 = 1, B_1 = -\frac{1}{2}, B_2 = \frac{1}{6}, B_4 = -\frac{1}{30},$ $B_6 = \frac{1}{42}, B_8 = -\frac{1}{30}, B_{10} = \frac{5}{66}.$	Continuous distributions: If $\Pr[a < X < b] = \int_a^b p(x) dx,$ then p is the probability density function of X . If $\Pr[X < a] = P(a),$
2	4	3	Change of base, quadratic formula: $\log_b x = \frac{\log_a x}{\log_a b}, \quad \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$	then P is the distribution function of X . If P and p both exist then $P(a) = \int_{-\infty}^a p(x) dx.$
3	8	5	Euler's number e : $e = 1 + \frac{1}{2} + \frac{1}{6} + \frac{1}{24} + \frac{1}{120} + \dots$ $\lim_{n \rightarrow \infty} \left(1 + \frac{x}{n}\right)^n = e^x.$	Expectation: If X is discrete $E[g(X)] = \sum_x g(x) \Pr[X = x].$
4	16	7	$(1 + \frac{1}{n})^n < e < (1 + \frac{1}{n})^{n+1}.$	If X continuous then $E[g(X)] = \int_{-\infty}^{\infty} g(x)p(x) dx = \int_{-\infty}^{\infty} g(x) dP(x).$
5	32	11	$(1 + \frac{1}{n})^n = e - \frac{e}{2n} + \frac{11e}{24n^2} - O\left(\frac{1}{n^3}\right).$	Variance, standard deviation: $\text{VAR}[X] = E[X^2] - E[X]^2,$ $\sigma = \sqrt{\text{VAR}[X]}.$
6	64	13	Harmonic numbers: $1, \frac{3}{2}, \frac{11}{6}, \frac{25}{12}, \frac{137}{60}, \frac{49}{20}, \frac{363}{140}, \frac{761}{280}, \frac{7129}{2520}, \dots$	For events A and B : $\Pr[A \vee B] = \Pr[A] + \Pr[B] - \Pr[A \wedge B]$ $\Pr[A \wedge B] = \Pr[A] \cdot \Pr[B],$ iff A and B are independent. $\Pr[A B] = \frac{\Pr[A \wedge B]}{\Pr[B]}$
7	128	17	$\ln n < H_n < \ln n + 1,$ $H_n = \ln n + \gamma + O\left(\frac{1}{n}\right).$	For random variables X and Y : $E[X \cdot Y] = E[X] \cdot E[Y],$ if X and Y are independent.
8	256	19	Factorial, Stirling's approximation: $1, 2, 6, 24, 120, 720, 5040, 40320, 362880, \dots$	$E[X + Y] = E[X] + E[Y],$ $E[cX] = cE[X].$
9	512	23	$n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right).$	Bayes' theorem: $\Pr[A_i B] = \frac{\Pr[B A_i] \Pr[A_i]}{\sum_{j=1}^n \Pr[A_j] \Pr[B A_j]}.$
10	1,024	29	Ackermann's function and inverse: $a(i, j) = \begin{cases} 2^j & i = 1 \\ a(i-1, 2) & j = 1 \\ a(i-1, a(i, j-1)) & i, j \geq 2 \end{cases}$ $\alpha(i) = \min\{j \mid a(j, j) \geq i\}.$	Inclusion-exclusion: $\Pr\left[\bigvee_{i=1}^n X_i\right] = \sum_{i=1}^n \Pr[X_i] + \sum_{k=2}^n (-1)^{k+1} \sum_{i_1 < \dots < i_k} \Pr\left[\bigwedge_{j=1}^k X_{i_j}\right].$
11	2,048	31	Binomial distribution: $\Pr[X = k] = \binom{n}{k} p^k q^{n-k}, \quad q = 1 - p,$ $E[X] = \sum_{k=1}^n k \binom{n}{k} p^k q^{n-k} = np.$	Moment inequalities: $\Pr[X \geq \lambda E[X]] \leq \frac{1}{\lambda},$ $\Pr[X - E[X] \geq \lambda \cdot \sigma] \leq \frac{1}{\lambda^2}.$
12	4,096	37	Poisson distribution: $\Pr[X = k] = \frac{e^{-\lambda} \lambda^k}{k!}, \quad E[X] = \lambda.$	Geometric distribution: $\Pr[X = k] = pq^{k-1}, \quad q = 1 - p,$ $E[X] = \sum_{k=1}^{\infty} kpq^{k-1} = \frac{1}{p}.$
13	8,192	41	Normal (Gaussian) distribution: $p(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-(x-\mu)^2/2\sigma^2}, \quad E[X] = \mu.$	
14	16,384	43	The “coupon collector”: We are given a random coupon each day, and there are n different types of coupons. The distribution of coupons is uniform. The expected number of days to pass before we collect all n types is	
15	32,768	47	$nH_n.$	
16	65,536	53		
17	131,072	59		
18	262,144	61		
19	524,288	67		
20	1,048,576	71		
21	2,097,152	73		
22	4,194,304	79		
23	8,388,608	83		
24	16,777,216	89		
25	33,554,432	97		
26	67,108,864	101		
27	134,217,728	103		
28	268,435,456	107		
29	536,870,912	109		
30	1,073,741,824	113		
31	2,147,483,648	127		
32	4,294,967,296	131		
Pascal's Triangle				
1				
1 1				
1 2 1				
1 3 3 1				
1 4 6 4 1				
1 5 10 10 5 1				
1 6 15 20 15 6 1				
1 7 21 35 35 21 7 1				
1 8 28 56 70 56 28 8 1				
1 9 36 84 126 126 84 36 9 1				
1 10 45 120 210 252 210 120 45 10 1				

Theoretical Computer Science Cheat Sheet

Trigonometry



Pythagorean theorem:

$$C^2 = A^2 + B^2.$$

Definitions:

$$\sin a = A/C, \quad \cos a = B/C,$$

$$\csc a = C/A, \quad \sec a = C/B,$$

$$\tan a = \frac{\sin a}{\cos a} = \frac{A}{B}, \quad \cot a = \frac{\cos a}{\sin a} = \frac{B}{A}.$$

Area, radius of inscribed circle:

$$\frac{1}{2}AB, \quad \frac{AB}{A+B+C}.$$

Identities:

$$\sin x = \frac{1}{\csc x}, \quad \cos x = \frac{1}{\sec x},$$

$$\tan x = \frac{1}{\cot x}, \quad \sin^2 x + \cos^2 x = 1,$$

$$1 + \tan^2 x = \sec^2 x, \quad 1 + \cot^2 x = \csc^2 x,$$

$$\sin x = \cos(\frac{\pi}{2} - x), \quad \sin x = \sin(\pi - x),$$

$$\cos x = -\cos(\pi - x), \quad \tan x = \cot(\frac{\pi}{2} - x),$$

$$\cot x = -\cot(\pi - x), \quad \csc x = \cot \frac{x}{2} - \cot x,$$

$$\sin(x \pm y) = \sin x \cos y \pm \cos x \sin y,$$

$$\cos(x \pm y) = \cos x \cos y \mp \sin x \sin y,$$

$$\tan(x \pm y) = \frac{\tan x \pm \tan y}{1 \mp \tan x \tan y},$$

$$\cot(x \pm y) = \frac{\cot x \cot y \mp 1}{\cot x \pm \cot y},$$

$$\sin 2x = 2 \sin x \cos x, \quad \sin 2x = \frac{2 \tan x}{1 + \tan^2 x},$$

$$\cos 2x = \cos^2 x - \sin^2 x, \quad \cos 2x = 2 \cos^2 x - 1,$$

$$\cos 2x = 1 - 2 \sin^2 x, \quad \cos 2x = \frac{1 - \tan^2 x}{1 + \tan^2 x},$$

$$\tan 2x = \frac{2 \tan x}{1 - \tan^2 x}, \quad \cot 2x = \frac{\cot^2 x - 1}{2 \cot x},$$

$$\sin(x+y) \sin(x-y) = \sin^2 x - \sin^2 y,$$

$$\cos(x+y) \cos(x-y) = \cos^2 x - \sin^2 y.$$

$$\text{Euler's equation: } e^{ix} = \cos x + i \sin x, \quad e^{i\pi} = -1.$$

v2.02 ©1994 by Steve Seiden

sseiden@acm.org

<http://www.csc.lsu.edu/~seiden>

Matrices

Multiplication:

$$C = A \cdot B, \quad c_{i,j} = \sum_{k=1}^n a_{i,k} b_{k,j}.$$

Determinants: $\det A \neq 0$ iff A is non-singular.

$$\det A \cdot B = \det A \cdot \det B,$$

$$\det A = \sum_{\pi} \prod_{i=1}^n \text{sign}(\pi) a_{i,\pi(i)}.$$

2×2 and 3×3 determinant:

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc,$$

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = g \begin{vmatrix} b & c \\ e & f \end{vmatrix} - h \begin{vmatrix} a & c \\ d & f \end{vmatrix} + i \begin{vmatrix} a & b \\ d & e \end{vmatrix}$$

$$= ae i + b f g + c d h - c e g - f h a - i b d.$$

Permanents:

$$\text{perm } A = \sum_{\pi} \prod_{i=1}^n a_{i,\pi(i)}.$$

Hyperbolic Functions

Definitions:

$$\sinh x = \frac{e^x - e^{-x}}{2}, \quad \cosh x = \frac{e^x + e^{-x}}{2},$$

$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \quad \csch x = \frac{1}{\sinh x},$$

$$\sech x = \frac{1}{\cosh x}, \quad \coth x = \frac{1}{\tanh x}.$$

Identities:

$$\cosh^2 x - \sinh^2 x = 1, \quad \tanh^2 x + \sech^2 x = 1,$$

$$\coth^2 x - \csch^2 x = 1, \quad \sinh(-x) = -\sinh x,$$

$$\cosh(-x) = \cosh x, \quad \tanh(-x) = -\tanh x,$$

$$\sinh(x+y) = \sinh x \cosh y + \cosh x \sinh y,$$

$$\cosh(x+y) = \cosh x \cosh y + \sinh x \sinh y,$$

$$\sinh 2x = 2 \sinh x \cosh x,$$

$$\cosh 2x = \cosh^2 x + \sinh^2 x,$$

$$\cosh x + \sinh x = e^x, \quad \cosh x - \sinh x = e^{-x},$$

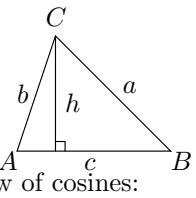
$$(\cosh x + \sinh x)^n = \cosh nx + \sinh nx, \quad n \in \mathbb{Z},$$

$$2 \sinh^2 \frac{x}{2} = \cosh x - 1, \quad 2 \cosh^2 \frac{x}{2} = \cosh x + 1.$$

θ	$\sin \theta$	$\cos \theta$	$\tan \theta$
0	0	1	0
$\frac{\pi}{6}$	$\frac{1}{2}$	$\frac{\sqrt{3}}{2}$	$\frac{\sqrt{3}}{3}$
$\frac{\pi}{4}$	$\frac{\sqrt{2}}{2}$	$\frac{\sqrt{2}}{2}$	1
$\frac{\pi}{3}$	$\frac{\sqrt{3}}{2}$	$\frac{1}{2}$	$\sqrt{3}$
$\frac{\pi}{2}$	1	0	∞

... in mathematics you don't understand things, you just get used to them.
- J. von Neumann

More Trig.



Law of cosines:

$$c^2 = a^2 + b^2 - 2ab \cos C.$$

Area:

$$\begin{aligned} A &= \frac{1}{2}hc, \\ &= \frac{1}{2}ab \sin C, \\ &= \frac{c^2 \sin A \sin B}{2 \sin C}. \end{aligned}$$

Heron's formula:

$$A = \sqrt{s \cdot s_a \cdot s_b \cdot s_c},$$

$$s = \frac{1}{2}(a+b+c),$$

$$s_a = s - a,$$

$$s_b = s - b,$$

$$s_c = s - c.$$

More identities:

$$\sin \frac{x}{2} = \sqrt{\frac{1 - \cos x}{2}},$$

$$\cos \frac{x}{2} = \sqrt{\frac{1 + \cos x}{2}},$$

$$\tan \frac{x}{2} = \sqrt{\frac{1 - \cos x}{1 + \cos x}},$$

$$= \frac{1 - \cos x}{\sin x},$$

$$= \frac{\sin x}{1 + \cos x},$$

$$\cot \frac{x}{2} = \sqrt{\frac{1 + \cos x}{1 - \cos x}},$$

$$= \frac{1 + \cos x}{\sin x},$$

$$= \frac{\sin x}{1 - \cos x},$$

$$\sin x = \frac{e^{ix} - e^{-ix}}{2i},$$

$$\cos x = \frac{e^{ix} + e^{-ix}}{2},$$

$$\tan x = -i \frac{e^{ix} - e^{-ix}}{e^{ix} + e^{-ix}},$$

$$= -i \frac{e^{2ix} - 1}{e^{2ix} + 1},$$

$$\sinh x = \frac{\sinh ix}{i},$$

$$\cosh x = \cosh ix,$$

$$\tan x = \frac{\tanh ix}{i}.$$

Theoretical Computer Science Cheat Sheet

Number Theory	Graph Theory								
<p>The Chinese remainder theorem: There exists a number C such that:</p> $C \equiv r_1 \pmod{m_1}$ $\vdots \quad \vdots \quad \vdots$ $C \equiv r_n \pmod{m_n}$ <p>if m_i and m_j are relatively prime for $i \neq j$.</p> <p>Euler's function: $\phi(x)$ is the number of positive integers less than x relatively prime to x. If $\prod_{i=1}^n p_i^{e_i}$ is the prime factorization of x then</p> $\phi(x) = \prod_{i=1}^n p_i^{e_i-1} (p_i - 1).$ <p>Euler's theorem: If a and b are relatively prime then</p> $1 \equiv a^{\phi(b)} \pmod{b}.$ <p>Fermat's theorem:</p> $1 \equiv a^{p-1} \pmod{p}.$ <p>The Euclidean algorithm: if $a > b$ are integers then</p> $\gcd(a, b) = \gcd(a \bmod b, b).$ <p>If $\prod_{i=1}^n p_i^{e_i}$ is the prime factorization of x then</p> $S(x) = \sum_{d x} d = \prod_{i=1}^n \frac{p_i^{e_i+1} - 1}{p_i - 1}.$ <p>Perfect Numbers: x is an even perfect number iff $x = 2^{n-1}(2^n - 1)$ and $2^n - 1$ is prime.</p> <p>Wilson's theorem: n is a prime iff</p> $(n-1)! \equiv -1 \pmod{n}.$ <p>Möbius inversion:</p> $\mu(i) = \begin{cases} 1 & \text{if } i = 1. \\ 0 & \text{if } i \text{ is not square-free.} \\ (-1)^r & \text{if } i \text{ is the product of } r \text{ distinct primes.} \end{cases}$ <p>If</p> $G(a) = \sum_{d a} F(d),$ <p>then</p> $F(a) = \sum_{d a} \mu(d) G\left(\frac{a}{d}\right).$ <p>Prime numbers:</p> $p_n = n \ln n + n \ln \ln n - n + n \frac{\ln \ln n}{\ln n} + O\left(\frac{n}{\ln n}\right),$ $\pi(n) = \frac{n}{\ln n} + \frac{n}{(\ln n)^2} + \frac{2!n}{(\ln n)^3} + O\left(\frac{n}{(\ln n)^4}\right).$	<p>Definitions:</p> <ul style="list-style-type: none"> Loop: An edge connecting a vertex to itself. Directed: Each edge has a direction. Simple: Graph with no loops or multi-edges. Walk: A sequence $v_0 e_1 v_1 \dots e_\ell v_\ell$. Trail: A walk with distinct edges. Path: A trail with distinct vertices. Connected: A graph where there exists a path between any two vertices. Component: A maximal connected subgraph. Tree: A connected acyclic graph. Free tree: A tree with no root. DAG: Directed acyclic graph. Eulerian: Graph with a trail visiting each edge exactly once. Hamiltonian: Graph with a cycle visiting each vertex exactly once. Cut: A set of edges whose removal increases the number of components. Cut-set: A minimal cut. Cut edge: A size 1 cut. k-Connected: A graph connected with the removal of any $k-1$ vertices. k-Tough: $\forall S \subseteq V, S \neq \emptyset$ we have $k \cdot c(G-S) \leq S$. k-Regular: A graph where all vertices have degree k. k-Factor: A k-regular spanning subgraph. Matching: A set of edges, no two of which are adjacent. Clique: A set of vertices, all of which are adjacent. Ind. set: A set of vertices, none of which are adjacent. Vertex cover: A set of vertices which cover all edges. Planar graph: A graph which can be embedded in the plane. Plane graph: An embedding of a planar graph. <p>Notation:</p> <ul style="list-style-type: none"> $E(G)$: Edge set $V(G)$: Vertex set $c(G)$: Number of components $G[S]$: Induced subgraph $\deg(v)$: Degree of v $\Delta(G)$: Maximum degree $\delta(G)$: Minimum degree $\chi(G)$: Chromatic number $\chi_E(G)$: Edge chromatic number G^c: Complement graph K_n: Complete graph K_{n_1, n_2}: Complete bipartite graph $r(k, \ell)$: Ramsey number <p>Geometry</p> <p>Projective coordinates: triples (x, y, z), not all x, y and z zero.</p> $(x, y, z) = (cx, cy, cz) \quad \forall c \neq 0.$ <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 50%; text-align: center;">Cartesian</td> <td style="width: 50%; text-align: center;">Projective</td> </tr> <tr> <td>(x, y)</td> <td>$(x, y, 1)$</td> </tr> <tr> <td>$y = mx + b$</td> <td>$(m, -1, b)$</td> </tr> <tr> <td>$x = c$</td> <td>$(1, 0, -c)$</td> </tr> </table> <p>Distance formula, L_p and L_∞ metric:</p> $\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2},$ $[x_1 - x_0 ^p + y_1 - y_0 ^p]^{1/p},$ $\lim_{p \rightarrow \infty} [x_1 - x_0 ^p + y_1 - y_0 ^p]^{1/p}.$ <p>Area of triangle (x_0, y_0), (x_1, y_1) and (x_2, y_2):</p> $\frac{1}{2} \operatorname{abs} \begin{vmatrix} x_1 - x_0 & y_1 - y_0 \\ x_2 - x_0 & y_2 - y_0 \end{vmatrix}.$ <p>Angle formed by three points:</p> $\cos \theta = \frac{(x_1, y_1) \cdot (x_2, y_2)}{\ell_1 \ell_2}.$ <p>Line through two points (x_0, y_0) and (x_1, y_1):</p> $\begin{vmatrix} x & y & 1 \\ x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \end{vmatrix} = 0.$ <p>Area of circle, volume of sphere:</p> $A = \pi r^2, \quad V = \frac{4}{3} \pi r^3.$ <p>If I have seen farther than others, it is because I have stood on the shoulders of giants. – Isaac Newton</p>	Cartesian	Projective	(x, y)	$(x, y, 1)$	$y = mx + b$	$(m, -1, b)$	$x = c$	$(1, 0, -c)$
Cartesian	Projective								
(x, y)	$(x, y, 1)$								
$y = mx + b$	$(m, -1, b)$								
$x = c$	$(1, 0, -c)$								

Theoretical Computer Science Cheat Sheet

π

Wallis' identity:

$$\pi = 2 \cdot \frac{2 \cdot 2 \cdot 4 \cdot 4 \cdot 6 \cdot 6 \cdots}{1 \cdot 3 \cdot 3 \cdot 5 \cdot 5 \cdot 7 \cdots}$$

Brouncker's continued fraction expansion:

$$\frac{\pi}{4} = 1 + \cfrac{1^2}{2 + \cfrac{3^2}{2 + \cfrac{5^2}{2 + \cfrac{7^2}{\cdots}}}}$$

Gregory's series:

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots$$

Newton's series:

$$\frac{\pi}{6} = \frac{1}{2} + \frac{1}{2 \cdot 3 \cdot 2^3} + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5 \cdot 2^5} + \cdots$$

Sharp's series:

$$\frac{\pi}{6} = \frac{1}{\sqrt{3}} \left(1 - \frac{1}{3^1 \cdot 3} + \frac{1}{3^2 \cdot 5} - \frac{1}{3^3 \cdot 7} + \cdots \right)$$

Euler's series:

$$\frac{\pi^2}{6} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \frac{1}{5^2} + \cdots$$

$$\frac{\pi^2}{8} = \frac{1}{1^2} + \frac{1}{3^2} + \frac{1}{5^2} + \frac{1}{7^2} + \frac{1}{9^2} + \cdots$$

$$\frac{\pi^2}{12} = \frac{1}{1^2} - \frac{1}{2^2} + \frac{1}{3^2} - \frac{1}{4^2} + \frac{1}{5^2} - \cdots$$

Partial Fractions

Let $N(x)$ and $D(x)$ be polynomial functions of x . We can break down $N(x)/D(x)$ using partial fraction expansion. First, if the degree of N is greater than or equal to the degree of D , divide N by D , obtaining

$$\frac{N(x)}{D(x)} = Q(x) + \frac{N'(x)}{D(x)},$$

where the degree of N' is less than that of D . Second, factor $D(x)$. Use the following rules: For a non-repeated factor:

$$\frac{N(x)}{(x-a)D(x)} = \frac{A}{x-a} + \frac{N'(x)}{D(x)},$$

where

$$A = \left[\frac{N(x)}{D(x)} \right]_{x=a}.$$

For a repeated factor:

$$\frac{N(x)}{(x-a)^m D(x)} = \sum_{k=0}^{m-1} \frac{A_k}{(x-a)^{m-k}} + \frac{N'(x)}{D(x)},$$

where

$$A_k = \frac{1}{k!} \left[\frac{d^k}{dx^k} \left(\frac{N(x)}{D(x)} \right) \right]_{x=a}.$$

The reasonable man adapts himself to the world; the unreasonable persists in trying to adapt the world to himself. Therefore all progress depends on the unreasonable.
— George Bernard Shaw

Calculus

Derivatives:

1. $\frac{d(cu)}{dx} = c \frac{du}{dx},$
2. $\frac{d(u+v)}{dx} = \frac{du}{dx} + \frac{dv}{dx},$
3. $\frac{d(uv)}{dx} = u \frac{dv}{dx} + v \frac{du}{dx},$
4. $\frac{d(u^n)}{dx} = nu^{n-1} \frac{du}{dx},$
5. $\frac{d(u/v)}{dx} = \frac{v \left(\frac{du}{dx} \right) - u \left(\frac{dv}{dx} \right)}{v^2},$
6. $\frac{d(e^{cu})}{dx} = ce^{cu} \frac{du}{dx},$
7. $\frac{d(c^u)}{dx} = (\ln c) c^u \frac{du}{dx},$
9. $\frac{d(\sin u)}{dx} = \cos u \frac{du}{dx},$
11. $\frac{d(\tan u)}{dx} = \sec^2 u \frac{du}{dx},$
13. $\frac{d(\sec u)}{dx} = \tan u \sec u \frac{du}{dx},$
15. $\frac{d(\arcsin u)}{dx} = \frac{1}{\sqrt{1-u^2}} \frac{du}{dx},$
17. $\frac{d(\arctan u)}{dx} = \frac{1}{1+u^2} \frac{du}{dx},$
19. $\frac{d(\text{arcsec } u)}{dx} = \frac{1}{u\sqrt{1-u^2}} \frac{du}{dx},$
21. $\frac{d(\sinh u)}{dx} = \cosh u \frac{du}{dx},$
23. $\frac{d(\tanh u)}{dx} = \text{sech}^2 u \frac{du}{dx},$
25. $\frac{d(\text{sech } u)}{dx} = -\text{sech } u \tanh u \frac{du}{dx},$
27. $\frac{d(\text{arsinh } u)}{dx} = \frac{1}{\sqrt{1+u^2}} \frac{du}{dx},$
29. $\frac{d(\text{arctanh } u)}{dx} = \frac{1}{1-u^2} \frac{du}{dx},$
31. $\frac{d(\text{arcsech } u)}{dx} = \frac{-1}{u\sqrt{1-u^2}} \frac{du}{dx},$

Integrals:

1. $\int cu \, dx = c \int u \, dx,$
2. $\int (u+v) \, dx = \int u \, dx + \int v \, dx,$
3. $\int x^n \, dx = \frac{1}{n+1} x^{n+1}, \quad n \neq -1,$
4. $\int \frac{1}{x} \, dx = \ln x,$
5. $\int e^x \, dx = e^x,$
6. $\int \frac{dx}{1+x^2} = \arctan x,$
8. $\int \sin x \, dx = -\cos x,$
10. $\int \tan x \, dx = -\ln |\cos x|,$
12. $\int \sec x \, dx = \ln |\sec x + \tan x|,$
14. $\int \arcsin \frac{x}{a} \, dx = \arcsin \frac{x}{a} + \sqrt{a^2 - x^2}, \quad a > 0,$
7. $\int u \frac{dv}{dx} \, dx = uv - \int v \frac{du}{dx} \, dx,$
9. $\int \cos x \, dx = \sin x,$
11. $\int \cot x \, dx = \ln |\cos x|,$
13. $\int \csc x \, dx = \ln |\csc x + \cot x|,$

Theoretical Computer Science Cheat Sheet

Calculus Cont.

- 15.** $\int \arccos \frac{x}{a} dx = \arccos \frac{x}{a} - \sqrt{a^2 - x^2}, \quad a > 0,$
- 16.** $\int \arctan \frac{x}{a} dx = x \arctan \frac{x}{a} - \frac{a}{2} \ln(a^2 + x^2), \quad a > 0,$
- 17.** $\int \sin^2(ax) dx = \frac{1}{2a} (ax - \sin(ax) \cos(ax)),$
- 18.** $\int \cos^2(ax) dx = \frac{1}{2a} (ax + \sin(ax) \cos(ax)),$
- 19.** $\int \sec^2 x dx = \tan x,$
- 20.** $\int \csc^2 x dx = -\cot x,$
- 21.** $\int \sin^n x dx = -\frac{\sin^{n-1} x \cos x}{n} + \frac{n-1}{n} \int \sin^{n-2} x dx,$
- 22.** $\int \cos^n x dx = \frac{\cos^{n-1} x \sin x}{n} + \frac{n-1}{n} \int \cos^{n-2} x dx,$
- 23.** $\int \tan^n x dx = \frac{\tan^{n-1} x}{n-1} - \int \tan^{n-2} x dx, \quad n \neq 1,$
- 24.** $\int \cot^n x dx = -\frac{\cot^{n-1} x}{n-1} - \int \cot^{n-2} x dx, \quad n \neq 1,$
- 25.** $\int \sec^n x dx = \frac{\tan x \sec^{n-1} x}{n-1} + \frac{n-2}{n-1} \int \sec^{n-2} x dx, \quad n \neq 1,$
- 26.** $\int \csc^n x dx = -\frac{\cot x \csc^{n-1} x}{n-1} + \frac{n-2}{n-1} \int \csc^{n-2} x dx, \quad n \neq 1,$
- 27.** $\int \sinh x dx = \cosh x,$
- 28.** $\int \cosh x dx = \sinh x,$
- 29.** $\int \tanh x dx = \ln |\cosh x|,$
- 30.** $\int \coth x dx = \ln |\sinh x|,$
- 31.** $\int \operatorname{sech} x dx = \arctan \sinh x,$
- 32.** $\int \operatorname{csch} x dx = \ln |\tanh \frac{x}{2}|,$
- 33.** $\int \sinh^2 x dx = \frac{1}{4} \sinh(2x) - \frac{1}{2}x,$
- 34.** $\int \cosh^2 x dx = \frac{1}{4} \sinh(2x) + \frac{1}{2}x,$
- 35.** $\int \operatorname{sech}^2 x dx = \tanh x,$
- 36.** $\int \operatorname{arcsinh} \frac{x}{a} dx = x \operatorname{arcsinh} \frac{x}{a} - \sqrt{x^2 + a^2}, \quad a > 0,$
- 37.** $\int \operatorname{arctanh} \frac{x}{a} dx = x \operatorname{arctanh} \frac{x}{a} + \frac{a}{2} \ln |a^2 - x^2|,$
- 38.** $\int \operatorname{arccosh} \frac{x}{a} dx = \begin{cases} x \operatorname{arccosh} \frac{x}{a} - \sqrt{x^2 + a^2}, & \text{if } \operatorname{arccosh} \frac{x}{a} > 0 \text{ and } a > 0, \\ x \operatorname{arccosh} \frac{x}{a} + \sqrt{x^2 + a^2}, & \text{if } \operatorname{arccosh} \frac{x}{a} < 0 \text{ and } a > 0, \end{cases}$
- 39.** $\int \frac{dx}{\sqrt{a^2 + x^2}} = \ln \left(x + \sqrt{a^2 + x^2} \right), \quad a > 0,$
- 40.** $\int \frac{dx}{a^2 + x^2} = \frac{1}{a} \arctan \frac{x}{a}, \quad a > 0,$
- 41.** $\int \sqrt{a^2 - x^2} dx = \frac{x}{2} \sqrt{a^2 - x^2} + \frac{a^2}{2} \arcsin \frac{x}{a}, \quad a > 0,$
- 42.** $\int (a^2 - x^2)^{3/2} dx = \frac{x}{8} (5a^2 - 2x^2) \sqrt{a^2 - x^2} + \frac{3a^4}{8} \arcsin \frac{x}{a}, \quad a > 0,$
- 43.** $\int \frac{dx}{\sqrt{a^2 - x^2}} = \arcsin \frac{x}{a}, \quad a > 0,$
- 44.** $\int \frac{dx}{a^2 - x^2} = \frac{1}{2a} \ln \left| \frac{a+x}{a-x} \right|,$
- 45.** $\int \frac{dx}{(a^2 - x^2)^{3/2}} = \frac{x}{a^2 \sqrt{a^2 - x^2}},$
- 46.** $\int \sqrt{a^2 \pm x^2} dx = \frac{x}{2} \sqrt{a^2 \pm x^2} \pm \frac{a^2}{2} \ln \left| x + \sqrt{a^2 \pm x^2} \right|,$
- 47.** $\int \frac{dx}{\sqrt{x^2 - a^2}} = \ln \left| x + \sqrt{x^2 - a^2} \right|, \quad a > 0,$
- 48.** $\int \frac{dx}{ax^2 + bx} = \frac{1}{a} \ln \left| \frac{x}{a+bx} \right|,$
- 49.** $\int x \sqrt{a+bx} dx = \frac{2(3bx-2a)(a+bx)^{3/2}}{15b^2},$
- 50.** $\int \frac{\sqrt{a+bx}}{x} dx = 2\sqrt{a+bx} + a \int \frac{1}{x\sqrt{a+bx}} dx,$
- 51.** $\int \frac{x}{\sqrt{a+bx}} dx = \frac{1}{\sqrt{2}} \ln \left| \frac{\sqrt{a+bx} - \sqrt{a}}{\sqrt{a+bx} + \sqrt{a}} \right|, \quad a > 0,$
- 52.** $\int \frac{\sqrt{a^2 - x^2}}{x} dx = \sqrt{a^2 - x^2} - a \ln \left| \frac{a + \sqrt{a^2 - x^2}}{x} \right|,$
- 53.** $\int x \sqrt{a^2 - x^2} dx = -\frac{1}{3} (a^2 - x^2)^{3/2},$
- 54.** $\int x^2 \sqrt{a^2 - x^2} dx = \frac{x}{8} (2x^2 - a^2) \sqrt{a^2 - x^2} + \frac{a^4}{8} \arcsin \frac{x}{a}, \quad a > 0,$
- 55.** $\int \frac{dx}{\sqrt{a^2 - x^2}} = -\frac{1}{a} \ln \left| \frac{a + \sqrt{a^2 - x^2}}{x} \right|,$
- 56.** $\int \frac{x dx}{\sqrt{a^2 - x^2}} = -\sqrt{a^2 - x^2},$
- 57.** $\int \frac{x^2 dx}{\sqrt{a^2 - x^2}} = -\frac{x}{2} \sqrt{a^2 - x^2} + \frac{a^2}{2} \arcsin \frac{x}{a}, \quad a > 0,$
- 58.** $\int \frac{\sqrt{a^2 + x^2}}{x} dx = \sqrt{a^2 + x^2} - a \ln \left| \frac{a + \sqrt{a^2 + x^2}}{x} \right|,$
- 59.** $\int \frac{\sqrt{x^2 - a^2}}{x} dx = \sqrt{x^2 - a^2} - a \arccos \frac{a}{|x|}, \quad a > 0,$
- 60.** $\int x \sqrt{x^2 \pm a^2} dx = \frac{1}{3} (x^2 \pm a^2)^{3/2},$
- 61.** $\int \frac{dx}{x\sqrt{x^2 + a^2}} = \frac{1}{a} \ln \left| \frac{x}{a + \sqrt{a^2 + x^2}} \right|,$

Theoretical Computer Science Cheat Sheet

Calculus Cont.

62. $\int \frac{dx}{x\sqrt{x^2 - a^2}} = \frac{1}{a} \arccos \frac{a}{|x|}, \quad a > 0,$ **63.** $\int \frac{dx}{x^2\sqrt{x^2 \pm a^2}} = \mp \frac{\sqrt{x^2 \pm a^2}}{a^2 x},$
64. $\int \frac{x dx}{\sqrt{x^2 \pm a^2}} = \sqrt{x^2 \pm a^2},$ **65.** $\int \frac{\sqrt{x^2 \pm a^2}}{x^4} dx = \mp \frac{(x^2 + a^2)^{3/2}}{3a^2 x^3},$
66. $\int \frac{dx}{ax^2 + bx + c} = \begin{cases} \frac{1}{\sqrt{b^2 - 4ac}} \ln \left| \frac{2ax + b - \sqrt{b^2 - 4ac}}{2ax + b + \sqrt{b^2 - 4ac}} \right|, & \text{if } b^2 > 4ac, \\ \frac{2}{\sqrt{4ac - b^2}} \arctan \frac{2ax + b}{\sqrt{4ac - b^2}}, & \text{if } b^2 < 4ac, \end{cases}$
67. $\int \frac{dx}{\sqrt{ax^2 + bx + c}} = \begin{cases} \frac{1}{\sqrt{a}} \ln \left| 2ax + b + 2\sqrt{a} \sqrt{ax^2 + bx + c} \right|, & \text{if } a > 0, \\ \frac{1}{\sqrt{-a}} \arcsin \frac{-2ax - b}{\sqrt{b^2 - 4ac}}, & \text{if } a < 0, \end{cases}$
68. $\int \sqrt{ax^2 + bx + c} dx = \frac{2ax + b}{4a} \sqrt{ax^2 + bx + c} + \frac{4ax - b^2}{8a} \int \frac{dx}{\sqrt{ax^2 + bx + c}},$
69. $\int \frac{x dx}{\sqrt{ax^2 + bx + c}} = \frac{\sqrt{ax^2 + bx + c}}{a} - \frac{b}{2a} \int \frac{dx}{\sqrt{ax^2 + bx + c}},$
70. $\int \frac{dx}{x\sqrt{ax^2 + bx + c}} = \begin{cases} \frac{-1}{\sqrt{c}} \ln \left| \frac{2\sqrt{c}\sqrt{ax^2 + bx + c} + bx + 2c}{x} \right|, & \text{if } c > 0, \\ \frac{1}{\sqrt{-c}} \arcsin \frac{bx + 2c}{|x|\sqrt{b^2 - 4ac}}, & \text{if } c < 0, \end{cases}$
71. $\int x^3 \sqrt{x^2 + a^2} dx = (\frac{1}{3}x^2 - \frac{2}{15}a^2)(x^2 + a^2)^{3/2},$
72. $\int x^n \sin(ax) dx = -\frac{1}{a} x^n \cos(ax) + \frac{n}{a} \int x^{n-1} \cos(ax) dx,$
73. $\int x^n \cos(ax) dx = \frac{1}{a} x^n \sin(ax) - \frac{n}{a} \int x^{n-1} \sin(ax) dx,$
74. $\int x^n e^{ax} dx = \frac{x^n e^{ax}}{a} - \frac{n}{a} \int x^{n-1} e^{ax} dx,$
75. $\int x^n \ln(ax) dx = x^{n+1} \left(\frac{\ln(ax)}{n+1} - \frac{1}{(n+1)^2} \right),$
76. $\int x^n (\ln ax)^m dx = \frac{x^{n+1}}{n+1} (\ln ax)^m - \frac{m}{n+1} \int x^n (\ln ax)^{m-1} dx.$

$$\begin{array}{llll}
x^1 & x^1 & = & x^{\bar{1}} \\
x^2 & x^2 + x^1 & = & x^{\bar{2}} - x^{\bar{1}} \\
x^3 & x^3 + 3x^2 + x^1 & = & x^{\bar{3}} - 3x^{\bar{2}} + x^{\bar{1}} \\
x^4 & x^4 + 6x^3 + 7x^2 + x^1 & = & x^{\bar{4}} - 6x^{\bar{3}} + 7x^{\bar{2}} - x^{\bar{1}} \\
x^5 & x^5 + 15x^4 + 25x^3 + 10x^2 + x^1 & = & x^{\bar{5}} - 15x^{\bar{4}} + 25x^{\bar{3}} - 10x^{\bar{2}} + x^{\bar{1}} \\
x^{\bar{1}} & x^1 & x^{\bar{1}} = & x^1 \\
x^{\bar{2}} & x^2 + x^1 & x^{\bar{2}} = & x^2 - x^1 \\
x^{\bar{3}} & x^3 + 3x^2 + 2x^1 & x^{\bar{3}} = & x^3 - 3x^2 + 2x^1 \\
x^{\bar{4}} & x^4 + 6x^3 + 11x^2 + 6x^1 & x^{\bar{4}} = & x^4 - 6x^3 + 11x^2 - 6x^1 \\
x^{\bar{5}} & x^5 + 10x^4 + 35x^3 + 50x^2 + 24x^1 & x^{\bar{5}} = & x^5 - 10x^4 + 35x^3 - 50x^2 + 24x^1
\end{array}$$

Finite Calculus

Difference, shift operators:

$$\Delta f(x) = f(x+1) - f(x),$$

$$\mathrm{E} f(x) = f(x+1).$$

Fundamental Theorem:

$$f(x) = \Delta F(x) \Leftrightarrow \sum f(x) \delta x = F(x) + C.$$

$$\sum_a^b f(x) \delta x = \sum_{i=a}^{b-1} f(i).$$

Differences:

$$\Delta(cu) = c\Delta u, \quad \Delta(u+v) = \Delta u + \Delta v,$$

$$\Delta(uv) = u\Delta v + \mathrm{E} v \Delta u,$$

$$\Delta(x^n) = nx^{n-1},$$

$$\Delta(H_x) = x^{-1}, \quad \Delta(2^x) = 2^x,$$

$$\Delta(c^x) = (c-1)c^x, \quad \Delta(\binom{x}{m}) = \binom{x}{m-1}.$$

Sums:

$$\sum cu \delta x = c \sum u \delta x,$$

$$\sum (u+v) \delta x = \sum u \delta x + \sum v \delta x,$$

$$\sum u \Delta v \delta x = uv - \sum \mathrm{E} v \Delta u \delta x,$$

$$\sum x^n \delta x = \frac{x^{n+1}}{m+1}, \quad \sum x^{-1} \delta x = H_x,$$

$$\sum c^x \delta x = \frac{c^x}{c-1}, \quad \sum \binom{x}{m} \delta x = \binom{x}{m+1}.$$

Falling Factorial Powers:

$$x^n = x(x-1)\cdots(x-n+1), \quad n > 0,$$

$$x^0 = 1,$$

$$x^n = \frac{1}{(x+1)\cdots(x+|n|)}, \quad n < 0,$$

$$x^{n+m} = x^m (x-m)^n.$$

Rising Factorial Powers:

$$x^{\bar{n}} = x(x+1)\cdots(x+n-1), \quad n > 0,$$

$$x^{\bar{0}} = 1,$$

$$x^{\bar{n}} = \frac{1}{(x-1)\cdots(x-|n|)}, \quad n < 0,$$

$$x^{\bar{n+m}} = x^{\bar{m}} (x+m)^{\bar{n}}.$$

Conversion:

$$x^{\bar{n}} = (-1)^n (-x)^{\bar{n}} = (x-n+1)^{\bar{n}} = 1/(x+1)^{\bar{-n}},$$

$$x^{\bar{n}} = (-1)^n (-x)^n = (x+n-1)^n = 1/(x-1)^{\bar{-n}},$$

$$x^n = \sum_{k=1}^n \binom{n}{k} x^k = \sum_{k=1}^n \binom{n}{k} (-1)^{n-k} x^{\bar{k}},$$

$$x^{\bar{n}} = \sum_{k=1}^n \binom{n}{k} (-1)^{n-k} x^k,$$

$$x^{\bar{n}} = \sum_{k=1}^n \binom{n}{k} x^k.$$

Theoretical Computer Science Cheat Sheet

Series

Taylor's series:

$$f(x) = f(a) + (x-a)f'(a) + \frac{(x-a)^2}{2}f''(a) + \dots = \sum_{i=0}^{\infty} \frac{(x-a)^i}{i!} f^{(i)}(a).$$

Expansions:

$\frac{1}{1-x}$	$= 1 + x + x^2 + x^3 + x^4 + \dots$	$= \sum_{i=0}^{\infty} x^i,$
$\frac{1}{1-cx}$	$= 1 + cx + c^2x^2 + c^3x^3 + \dots$	$= \sum_{i=0}^{\infty} c^i x^i,$
$\frac{1}{1-x^n}$	$= 1 + x^n + x^{2n} + x^{3n} + \dots$	$= \sum_{i=0}^{\infty} x^{ni},$
$\frac{x}{(1-x)^2}$	$= x + 2x^2 + 3x^3 + 4x^4 + \dots$	$= \sum_{i=0}^{\infty} ix^i,$
$x^k \frac{d^n}{dx^n} \left(\frac{1}{1-x} \right)$	$= x + 2^n x^2 + 3^n x^3 + 4^n x^4 + \dots = \sum_{i=0}^{\infty} i^n x^i,$	
e^x	$= 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \dots$	$= \sum_{i=0}^{\infty} \frac{x^i}{i!},$
$\ln(1+x)$	$= x - \frac{1}{2}x^2 + \frac{1}{3}x^3 - \frac{1}{4}x^4 - \dots$	$= \sum_{i=1}^{\infty} (-1)^{i+1} \frac{x^i}{i},$
$\ln \frac{1}{1-x}$	$= x + \frac{1}{2}x^2 + \frac{1}{3}x^3 + \frac{1}{4}x^4 + \dots$	$= \sum_{i=1}^{\infty} \frac{x^i}{i},$
$\sin x$	$= x - \frac{1}{3!}x^3 + \frac{1}{5!}x^5 - \frac{1}{7!}x^7 + \dots$	$= \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)!},$
$\cos x$	$= 1 - \frac{1}{2!}x^2 + \frac{1}{4!}x^4 - \frac{1}{6!}x^6 + \dots$	$= \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i}}{(2i)!},$
$\tan^{-1} x$	$= x - \frac{1}{3}x^3 + \frac{1}{5}x^5 - \frac{1}{7}x^7 + \dots$	$= \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)},$
$(1+x)^n$	$= 1 + nx + \frac{n(n-1)}{2}x^2 + \dots$	$= \sum_{i=0}^{\infty} \binom{n}{i} x^i,$
$\frac{1}{(1-x)^{n+1}}$	$= 1 + (n+1)x + \binom{n+2}{2}x^2 + \dots$	$= \sum_{i=0}^{\infty} \binom{i+n}{i} x^i,$
$\frac{x}{e^x - 1}$	$= 1 - \frac{1}{2}x + \frac{1}{12}x^2 - \frac{1}{720}x^4 + \dots$	$= \sum_{i=0}^{\infty} \frac{B_i x^i}{i!},$
$\frac{1}{2x}(1 - \sqrt{1-4x})$	$= 1 + x + 2x^2 + 5x^3 + \dots$	$= \sum_{i=0}^{\infty} \frac{1}{i+1} \binom{2i}{i} x^i,$
$\frac{1}{\sqrt{1-4x}}$	$= 1 + x + 2x^2 + 6x^3 + \dots$	$= \sum_{i=0}^{\infty} \binom{2i}{i} x^i,$
$\frac{1}{\sqrt[3]{1-4x}} \left(\frac{1-\sqrt{1-4x}}{2x} \right)^n$	$= 1 + (2+n)x + \binom{4+n}{2}x^2 + \dots$	$= \sum_{i=0}^{\infty} \binom{2i+n}{i} x^i,$
$\frac{1}{1-x} \ln \frac{1}{1-x}$	$= x + \frac{3}{2}x^2 + \frac{11}{6}x^3 + \frac{25}{12}x^4 + \dots$	$= \sum_{i=1}^{\infty} H_i x^i,$
$\frac{1}{2} \left(\ln \frac{1}{1-x} \right)^2$	$= \frac{1}{2}x^2 + \frac{3}{4}x^3 + \frac{11}{24}x^4 + \dots$	$= \sum_{i=2}^{\infty} \frac{H_{i-1} x^i}{i},$
$\frac{x}{1-x-x^2}$	$= x + x^2 + 2x^3 + 3x^4 + \dots$	$= \sum_{i=0}^{\infty} F_i x^i,$
$\frac{F_n x}{1 - (F_{n-1} + F_{n+1})x - (-1)^n x^2}$	$= F_n x + F_{2n} x^2 + F_{3n} x^3 + \dots$	$= \sum_{i=0}^{\infty} F_{ni} x^i.$

Ordinary power series:

$$A(x) = \sum_{i=0}^{\infty} a_i x^i.$$

Exponential power series:

$$A(x) = \sum_{i=0}^{\infty} a_i \frac{x^i}{i!}.$$

Dirichlet power series:

$$A(x) = \sum_{i=1}^{\infty} \frac{a_i}{i^x}.$$

Binomial theorem:

$$(x+y)^n = \sum_{k=0}^n \binom{n}{k} x^{n-k} y^k.$$

Difference of like powers:

$$x^n - y^n = (x-y) \sum_{k=0}^{n-1} x^{n-1-k} y^k.$$

For ordinary power series:

$$\alpha A(x) + \beta B(x) = \sum_{i=0}^{\infty} (\alpha a_i + \beta b_i) x^i,$$

$$x^k A(x) = \sum_{i=k}^{\infty} a_{i-k} x^i,$$

$$\frac{A(x) - \sum_{i=0}^{k-1} a_i x^i}{x^k} = \sum_{i=0}^{\infty} a_{i+k} x^i,$$

$$A(cx) = \sum_{i=0}^{\infty} c^i a_i x^i,$$

$$A'(x) = \sum_{i=0}^{\infty} (i+1) a_{i+1} x^i,$$

$$xA'(x) = \sum_{i=1}^{\infty} ia_i x^i,$$

$$\int A(x) dx = \sum_{i=1}^{\infty} \frac{a_{i-1}}{i} x^i,$$

$$\frac{A(x) + A(-x)}{2} = \sum_{i=0}^{\infty} a_{2i} x^{2i},$$

$$\frac{A(x) - A(-x)}{2} = \sum_{i=0}^{\infty} a_{2i+1} x^{2i+1}.$$

Summation: If $b_i = \sum_{j=0}^i a_j$ then

$$B(x) = \frac{1}{1-x} A(x).$$

Convolution:

$$A(x)B(x) = \sum_{i=0}^{\infty} \left(\sum_{j=0}^i a_j b_{i-j} \right) x^i.$$

God made the natural numbers;
all the rest is the work of man.

– Leopold Kronecker

Theoretical Computer Science Cheat Sheet

Series	Escher's Knot																																																																																																				
<p>Expansions:</p> $\frac{1}{(1-x)^{n+1}} \ln \frac{1}{1-x} = \sum_{i=0}^{\infty} (H_{n+i} - H_n) \binom{n+i}{i} x^i,$ $x^{\bar{n}} = \sum_{i=0}^{\infty} \binom{n}{i} x^i,$ $\left(\ln \frac{1}{1-x}\right)^n = \sum_{i=0}^{\infty} \binom{i}{n} \frac{n! x^i}{i!},$ $\tan x = \sum_{i=1}^{\infty} (-1)^{i-1} \frac{2^{2i}(2^{2i}-1)B_{2i}x^{2i-1}}{(2i)!},$ $\frac{1}{\zeta(x)} = \sum_{i=1}^{\infty} \frac{\mu(i)}{i^x},$ $\zeta(x) = \prod_p \frac{1}{1-p^{-x}},$ $\zeta^2(x) = \sum_{i=1}^{\infty} \frac{d(i)}{x^i} \quad \text{where } d(n) = \sum_{d n} 1,$ $\zeta(x)\zeta(x-1) = \sum_{i=1}^{\infty} \frac{S(i)}{x^i} \quad \text{where } S(n) = \sum_{d n} d,$ $\zeta(2n) = \frac{2^{2n-1} B_{2n} }{(2n)!} \pi^{2n}, \quad n \in \mathbb{N},$ $\frac{x}{\sin x} = \sum_{i=0}^{\infty} (-1)^{i-1} \frac{(4^i-2)B_{2i}x^{2i}}{(2i)!},$ $\left(\frac{1-\sqrt{1-4x}}{2x}\right)^n = \sum_{i=0}^{\infty} \frac{n(2i+n-1)!}{i!(n+i)!} x^i,$ $e^x \sin x = \sum_{i=1}^{\infty} \frac{2^{i/2} \sin \frac{i\pi}{4}}{i!} x^i,$ $\sqrt{\frac{1-\sqrt{1-x}}{x}} = \sum_{i=0}^{\infty} \frac{(4i)!}{16^i \sqrt{2}(2i)!(2i+1)!} x^i,$ $\left(\frac{\arcsin x}{x}\right)^2 = \sum_{i=0}^{\infty} \frac{4^i i!^2}{(i+1)(2i+1)!} x^{2i}.$																																																																																																					
	Stieltjes Integration																																																																																																				
	<p>If G is continuous in the interval $[a, b]$ and F is nondecreasing then</p> $\int_a^b G(x) dF(x)$ <p>exists. If $a \leq b \leq c$ then</p> $\int_a^c G(x) dF(x) = \int_a^b G(x) dF(x) + \int_b^c G(x) dF(x).$ <p>If the integrals involved exist</p> $\int_a^b (G(x) + H(x)) dF(x) = \int_a^b G(x) dF(x) + \int_a^b H(x) dF(x),$ $\int_a^b G(x) d(F(x) + H(x)) = \int_a^b G(x) dF(x) + \int_a^b G(x) dH(x),$ $\int_a^b c \cdot G(x) dF(x) = \int_a^b G(x) d(c \cdot F(x)) = c \int_a^b G(x) dF(x),$ $\int_a^b G(x) dF(x) = G(b)F(b) - G(a)F(a) - \int_a^b F(x) dG(x).$ <p>If the integrals involved exist, and F possesses a derivative F' at every point in $[a, b]$ then</p> $\int_a^b G(x) dF(x) = \int_a^b G(x) F'(x) dx.$																																																																																																				
Cramer's Rule	Fibonacci Numbers																																																																																																				
<p>If we have equations:</p> $a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n = b_1$ $a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n = b_2$ $\vdots \quad \vdots \quad \vdots$ $a_{n,1}x_1 + a_{n,2}x_2 + \cdots + a_{n,n}x_n = b_n$ <p>Let $A = (a_{i,j})$ and B be the column matrix (b_i). Then there is a unique solution iff $\det A \neq 0$. Let A_i be A with column i replaced by B. Then</p> $x_i = \frac{\det A_i}{\det A}.$	<table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <tr><td>00</td><td>47</td><td>18</td><td>76</td><td>29</td><td>93</td><td>85</td><td>34</td><td>61</td><td>52</td></tr> <tr><td>86</td><td>11</td><td>57</td><td>28</td><td>70</td><td>39</td><td>94</td><td>45</td><td>02</td><td>63</td></tr> <tr><td>95</td><td>80</td><td>22</td><td>67</td><td>38</td><td>71</td><td>49</td><td>56</td><td>13</td><td>04</td></tr> <tr><td>59</td><td>96</td><td>81</td><td>33</td><td>07</td><td>48</td><td>72</td><td>60</td><td>24</td><td>15</td></tr> <tr><td>73</td><td>69</td><td>90</td><td>82</td><td>44</td><td>17</td><td>58</td><td>01</td><td>35</td><td>26</td></tr> <tr><td>68</td><td>74</td><td>09</td><td>91</td><td>83</td><td>55</td><td>27</td><td>12</td><td>46</td><td>30</td></tr> <tr><td>37</td><td>08</td><td>75</td><td>19</td><td>92</td><td>84</td><td>66</td><td>23</td><td>50</td><td>41</td></tr> <tr><td>14</td><td>25</td><td>36</td><td>40</td><td>51</td><td>62</td><td>03</td><td>77</td><td>88</td><td>99</td></tr> <tr><td>21</td><td>32</td><td>43</td><td>54</td><td>65</td><td>06</td><td>10</td><td>89</td><td>97</td><td>78</td></tr> <tr><td>42</td><td>53</td><td>64</td><td>05</td><td>16</td><td>20</td><td>31</td><td>98</td><td>79</td><td>87</td></tr> </table> <p>The Fibonacci number system: Every integer n has a unique representation</p> $n = F_{k_1} + F_{k_2} + \cdots + F_{k_m},$ <p>where $k_i \geq k_{i+1} + 2$ for all i, $1 \leq i < m$ and $k_m \geq 2$.</p>	00	47	18	76	29	93	85	34	61	52	86	11	57	28	70	39	94	45	02	63	95	80	22	67	38	71	49	56	13	04	59	96	81	33	07	48	72	60	24	15	73	69	90	82	44	17	58	01	35	26	68	74	09	91	83	55	27	12	46	30	37	08	75	19	92	84	66	23	50	41	14	25	36	40	51	62	03	77	88	99	21	32	43	54	65	06	10	89	97	78	42	53	64	05	16	20	31	98	79	87
00	47	18	76	29	93	85	34	61	52																																																																																												
86	11	57	28	70	39	94	45	02	63																																																																																												
95	80	22	67	38	71	49	56	13	04																																																																																												
59	96	81	33	07	48	72	60	24	15																																																																																												
73	69	90	82	44	17	58	01	35	26																																																																																												
68	74	09	91	83	55	27	12	46	30																																																																																												
37	08	75	19	92	84	66	23	50	41																																																																																												
14	25	36	40	51	62	03	77	88	99																																																																																												
21	32	43	54	65	06	10	89	97	78																																																																																												
42	53	64	05	16	20	31	98	79	87																																																																																												