



Pedro Augusto Ulisses Andrade Adjailson Freire(POV)

Adoradores do Xandão

Contents		
1 Utils	2	
1.1 Files	2	4.7 Combinacao Simples 7
1.2 Limites	2	4.8 Permutacao Circular 7
1.3 Makefile	2	4.9 Permutacao Com Repeticao 7
1.4 Template Cpp	2	4.10 Permutacao Simples 7
1.5 Template Python	3	
2 Informações	4	5 DP 7
2.1 Bitmask	4	5.1 Dp 7
2.2 Priority Queue	4	5.2 Livro Caixa 8
2.3 Set	4	5.3 Mochila 8
2.4 Sort	5	5.4 Mochila Eduardo 8
2.5 String	5	
2.6 Vector	5	6 Estruturas 9
3 .vscode	6	6.1 Bittree 9
4 Combinatoria	6	6.2 Fenwick Tree 9
4.1 @ Factorial	6	6.3 Seg Tree 10
4.2 @ Tabela	6	6.4 Sparse Table Disjunta 11
4.3 Arranjo Com Repeticao	6	6.5 Tabuleiro 11
4.4 Arranjo Simples	6	6.6 Union Find 13
4.5 Catalan	6	
4.6 Combinacao Com Repeticao	6	7 Geometria 13
		7.1 3D - Distancia Entre 2 Poliedros 13
		7.2 Andrew 14
		7.3 Circulo 15
		7.4 Closestpair Otimizado 15
		7.5 Geometricosgerai 16
		7.6 Leis 17

7.7	Linha	17	10 Matematica	34
7.8	Maior Poligono Convexo	17	10.1 Casas	34
7.9	Minkowski Sum	19	10.2 Ciclo Em Funcao	34
7.10	Ponto	20	10.3 Contar Quanti Solucoes Eq 2 Variaveis	34
7.11	Triangulos	20	10.4 Conversao De Bases	35
7.12	Vetor	21	10.5 Decimal Para Fracao	35
8	Grafos	21	10.6 Dois Primos Somam Num	35
8.1	Bfs - Matriz	21	10.7 Factorial	35
8.2	Bfs - Por Niveis	22	10.8 Fast Exponentiation	35
8.3	Bfs - String	22	10.9 Fast Fibonacci	35
8.4	Bfs - Tradicional	22	10.10Fatorial Grande	36
8.5	Dfs	23	10.11Fibonacci Modulo	36
8.6	Articulation	23	10.12Mmc Mdc - Euclides Extendido	36
8.7	Bipartido	24	10.13Mmc Mdc - Mdc	37
8.8	Caminho Minimo - @Tabela	24	10.14Mmc Mdc - Mdc Multiplo	37
8.9	Caminho Minimo - Bellman Ford	25	10.15Mmc Mdc - Mmc	37
8.10	Caminho Minimo - Checar I J (In)Diretamente Conectados	25	10.16Mmc Mdc - Mmc Multiplo	37
8.11	Caminho Minimo - Diametro Do Grafo	25	10.17Modulo - @Info	37
8.12	Caminho Minimo - Dijkstra	26	10.18Modulo - Divisao E Potencia Mod M	37
8.13	Caminho Minimo - Floyd Warshall	26	10.19Modulo - Fibonacci Modulo	37
8.14	Caminho Minimo - Minimax	27	10.20N Fibonacci	38
8.15	Cycle Check	27	10.21Numeros Grandes	38
8.16	Encontrar Ciclo	28	10.22Primos - Divisores De N - Listar	38
8.17	Euler Tree	28	10.23Primos - Divisores De N - Somar	39
8.18	Kosaraju	28	10.24Primos - Fatores Primos - Contar Diferentes	39
8.19	Kruskal	29	10.25Primos - Fatores Primos - Listar	39
8.20	Labirinto	30	10.26Primos - Fatores Primos - Somar	39
8.21	Pontos Articulacao	30	10.27Primos - Is Prime	39
8.22	Prufer Code To Tree	31	10.28Primos - Lowest Prime Factor	39
8.23	Successor Graph	31	10.29Primos - Miller Rabin	39
8.24	Topological Sort	31	10.30Primos - Numero Fatores Primos De N	40
9	Grafos Especiais	32	10.31Primos - Primo Grande	40
9.1	Arvore - @Info	32	10.32Primos - Primos Relativos De N	40
9.2	Bipartido - @Info	32	10.33Primos - Sieve	40
9.3	Dag - @Info	32	10.34Primos - Sieve Linear	40
9.4	Dag - Sslp	32	10.35Tabela Verdade	41
9.5	Dag - Sssp	33	11 Matriz	41
9.6	Dag - Fishmonger	33	11.1 Fibonacci Matricial	41
9.7	Dag - Numero De Caminhos 2 Vertices	33	11.2 Maior Retangulo Binario Em Matriz	41
9.8	Eulerian - @Info	34	11.3 Maxsubmatrixsum	43
9.9	Eulerian - Euler Path	34	11.4 Max 2D Range Sum	43
			11.5 Potencia Matriz	43
			11.6 Verifica Se E Quadrado Magico	44
			11.7 Verificar Se Retangulo Cabe Em Matriz Binaria	44

12 Strings	45
12.1 Kmp	45
12.2 Aro Corasick	46
12.3 Calculadora Posfixo	47
12.4 Chaves Colchetes Parenteses	48
12.5 Infixo Para Posfixo	48
12.6 Is Subsequence	48
12.7 Levenshtein	49
12.8 Lexico E Sintatico	49
12.9 Lexicograficamente Minima	50
12.10Longest Common Substring	50
12.11Lower Upper	50
12.12Numeros E Char	50
12.13Ocorrencias	51
12.14Palindromo	51
12.15Permutacao	51
12.16Remove Acento	51
12.17Split Cria	51
12.18String Hashing	51
13 Vector	52
13.1 Contar Menores Elementos A Direita	52
13.2 Contar Subarrays Somam K	52
13.3 Elemento Mais Frequente	53
13.4 K Maior Elemento	53
13.5 Longest Common Subsequence	54
13.6 Maior Retangulo Em Histograma	54
13.7 Maior Sequencia Subsequente	54
13.8 Maior Subsequencia Comum	54
13.9 Maior Subsequência Crescente	55
13.10Maior Triangulo Em Histograma	55
13.11Remove Repetitive	55
13.12Soma Maxima Sequencial	56
13.13Subset Sum	56
13.14Troco	56
14 Outros	56
14.1 Dp	56
14.2 Binario	57
14.3 Binary Search	57
14.4 Fibonacci	57
14.5 Horário	57
14.6 Intervalos	57

1 Utils

1.1 Files

```
1 #!/bin/bash
2
3 for c in {a..f}; do
4     cp temp.cpp "$c.cpp"
5     echo "$c" > "$c.txt"
6     if [ "$c" = "$letter" ]; then
7         break
8     fi
9 done
```

1.2 Limites

```
1 // LIMITES DE REPRESENTACAO DE DADOS
```

tipo	bits	minimo .. maximo	precisao decim.
char	8	0 .. 127	2
signed char	8	-128 .. 127	2
unsigned char	8	0 .. 255	2
short	16	-32.768 .. 32.767	4
unsigned short	16	0 .. 65.535	4
int	32	-2 x 10 ⁹ .. 2 x 10 ⁹	9
unsigned int	32	0 .. 4 x 10 ⁹	9
int64_t	64	-9 x 10 ¹⁸ .. 9 x 10 ¹⁸	18
uint64_t	64	0 .. 18 x 10 ¹⁸	19
float	32	1.2 x 10 ⁻³⁸ .. 3.4 x 10 ³⁸	6-9
double	64	2.2 x 10 ⁻³⁰⁸ .. 1.8 x 10 ³⁰⁸	15-17
long double	80	3.4 x 10 ⁻⁴⁹³² .. 1.1 x 10 ⁴⁹³²	18-19
BigInt/Dec(java)	1 x 10 ⁻²¹⁴⁷⁴⁸³⁶⁴⁸	.. 1 x 10 ²¹⁴⁷⁴⁸³⁶⁴⁷	0

```
19 // LIMITES DE MEMORIA
```

```
21 1MB = 1,048,576 bool
22 1MB = 524,288 char
23 1MB = 262,144 int32_t
24 1MB = 131,072 int64_t
25 1MB = 65,536 float
26 1MB = 32,768 double
27 1MB = 16,384 long double
28 1MB = 16,384 BigInteger / BigDecimal
```

```
30 // ESTOURAR TEMPO
```

input size	complexidade para 1 s
[10,11]	0(n!), 0(n ⁶)
[17,19]	0(2 ⁿ * n ²)
[18, 22]	0(2 ⁿ * n)
[24,26]	0(2 ⁿ)
... 100	0(n ⁴)
... 450	0(n ³)
... 1500	0(n ^{2.5})

```
41 ... 2500      | 0(n^2 * log n)
42 ... 10^4      | 0(n^2)
43 ... 2*10^5    | 0(n^1.5)
44 ... 4.5*10^6  | 0(n log n)
45 ... 10^7      | 0(n log log n)
46 ... 10^8      | 0(n), 0(log n), 0(1)
47
48
49 // FATORIAL
50
51 12! = 479.001.600 [limite do (u)int]
52 20! = 2.432.902.008.176.640.000 [limite do (u)int64_t]
```

1.3 Makefile

```
1 CXX = g++
2 CXXFLAGS = -fsanitize=address,undefined -fno-omit-frame-pointer -g -Wall
               -Wshadow -std=c++17 -Wno-unused-result -Wno-sign-compare -Wno-char-
               subscripts #-fuse-ld=gold
3
4 q:
5     cp temp.cpp $(f).cpp
6     touch $(f).txt
7     code $(f).txt
8     code $(f).cpp
9     clear
10 compile:
11     g++ -g $(f).cpp $(CXXFLAGS) -o $(f)
12 exe:
13     ./$(f) < $(f).txt
14
15 runc: compile
16 runci: compile exe
17
18 clearexe:
19     find . -maxdepth 1 -type f -executable -exec rm {} +
20 cleartxt:
21     find . -type f -name "*.txt" -exec rm -f {} \;
22 clear: clearexe cleartxt
23 clear
```

1.4 Template Cpp

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #define _ std::ios::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);
5 #define all(a) a.begin(), a.end()
6 #define int long long int
7 #define double long double
8 #define vi vector<int>
9 #define pii pair<int,int>
10 #define endl "\n"
11 #define print_v(a) for(auto x : a) cout<<x<<" ";cout<<endl
12 #define print_vp(a) for(auto x : a) cout<<x.first<<" "<<x.second<< endl
13 #define f(i,s,e) for(int i=s;i<e;i++)
```

```

14 #define rf(i,e,s)      for(int i=e-1;i>=s;i--)
15 #define CEIL(a, b)    ((a) + (b - 1))/b
16 #define TRUNC(x, n)   floor(x * pow(10, n))/pow(10, n)
17 #define ROUND(x, n)   round(x * pow(10, n))/pow(10, n)
18 #define dbg(x) cout << #x << " = " << x << " ";
19 #define dbg1(x) cout << #x << " = " << x << endl;
20
21 const int INF = 1e9;      // 2^31-1
22 const int LLINF = 4e18;  // 2^63-1
23 const double EPS = 1e-9;
24 const int MAX = 1e6+10;  // 10^6 + 10
25
26 void solve() {
27
28 }
29
30 int32_t main() { _
31
32
33     clock_t z = clock();
34     int t = 1; // cin >> t;
35     while (t--) {
36         solve();
37     }
38     cerr << fixed << "Run Time : " << ((double)(clock() - z) /
39     CLOCKS_PER_SEC) << endl;
40     return 0;
41 }

```

1.5 Template Python

```

1 import sys
2 import math
3 import bisect
4 from sys import stdin, stdout
5 from math import gcd, floor, sqrt, log
6 from collections import defaultdict as dd
7 from bisect import bisect_left as bl, bisect_right as br
8
9 sys.setrecursionlimit(100000000)
10
11 inp      = lambda: int(input())
12 strng    = lambda: input().strip()
13 jn       = lambda x, l: x.join(map(str, l))
14 strl     = lambda: list(input().strip())
15 mul      = lambda: map(int, input().strip().split())
16 mulf     = lambda: map(float, input().strip().split())
17 seq      = lambda: list(map(int, input().strip().split()))
18
19 ceil     = lambda x: int(x) if x==int(x) else int(x)+1
20 ceildiv  = lambda x, d: x//d if (x%d==0) else x//d+1
21
22 flush    = lambda: stdout.flush()
23 stdstr   = lambda: stdin.readline()
24 stdint   = lambda: int(stdin.readline())
25 stdpr    = lambda x: stdout.write(str(x))
26

```

```

27 mod=1000000007
28
29 #main code
30
31 a = None
32 b = None
33 lista = None
34
35 def ident(*args):
36     if len(args) == 1:
37         return args[0]
38     return args
39
40
41 def parsin(*, l=1, vpl=1, s=" "):
42     if l == 1:
43         if vpl == 1: return ident(input())
44         else: return list(map(ident, input().split(s)))
45     else:
46         if vpl == 1: return [ident(input()) for _ in range(l)]
47         else: return [list(map(ident, input().split(s))) for _ in range(l)]
48
49
50 def solve():
51     pass
52
53 # if __name__ == '__main__':
54 def main():
55     st = clk()
56
57     escolha = "in"
58     #escolha = "num"
59
60     match escolha:
61         case "in":
62             # ãl infinitas linhas agrupadas de 2 em 2
63             # pra infinitos valores em 1 linha pode armazenar em uma lista
64             while True:
65                 global a, b
66                 try: a, b = input().split()
67                 except (EOFError): break #permite ler todas as linhas
68             dentro do .txt
69             except (ValueError): pass # consegue ler éat linhas em
70             branco
71             else:
72                 a, b = int(a), int(b)
73                 solve()
74
75         case "num":
76             global lista
77             # int l; cin >> l; while(l--){for(i=0; i<vpl; i++){
78             # retorna listas com inputs de cada linha
79             # leia l linhas com vpl valores em cada uma delas
80             # caseo seja mais de uma linha, retorna lista com listas
81             de inputs
82             lista = parsin(l=2, vpl=5)

```

```

80         solve()
81
82     sys.stderr.write(f"Run Time : {(clk() - st):.6f} seconds\n")
83
84 main()

```

2 Informações

2.1 Bitmask

```

1  int n = 11, ans = 0, k = 3;
2
3  // Operacoes com bits
4  ans = n & k; // AND bit a bit
5  ans = n | k; // OR bit a bit
6  ans = n ^ k; // XOR bit a bit
7  ans = ~n;    // NOT bit a bit
8
9  // Operacoes com 2^k em O(1)
10 ans = n << k; // ans = n * 2^k
11 ans = n >> k; // ans = n / 2^k
12
13 int j;
14
15 // Ativa j-esimo bit (0-based)
16 ans |= (1<<j);
17
18 // Desativa j-esimo bit (0-based)
19 ans &= (1<<j);
20
21 // Inverte j-esimo bit (0-based)
22 ans ^= (1<<j);
23
24 // verificar se j-esimo bit esta ativo (0-based)
25 ans = n & (1<<j);
26
27 // Pegar valor do bit menos significativo | Retorna o maior divisor
28 ans = n & -n;
29
30 // Ligar todos os n bits
31 ans = (1<<n) - 1;
32
33 // Contar quantos 1's tem no binario de n
34 ans = __builtin_popcount(n);
35
36 // Contar quantos 0's tem no final do binario de n
37 ans = __builtin_ctz(n);

```

2.2 Priority Queue

```

1 // HEAP CRESCENTE {5,4,3,2,1}
2 priority_queue<int> pq; // max heap
3     // maior elemento:
4     pq.top();
5

```

```

6 // HEAP DECRESCENTE {1,2,3,4,5}
7 priority_queue<int, vector<int>, greater<int>> pq; // min heap
8     // menor elemento:
9     pq.top();
10
11 // REMOVER ELEMENTO
12 // Complexidade: O(n)
13 // Retorno: true se existe, false se não existe
14 pq.remove(x);
15
16 // INSERIR ELEMENTO
17 // Complexidade: O(log(n))
18 pq.push(x);
19
20 // REMOVER TOP
21 // Complexidade: O(log(n))
22 pq.pop();
23
24 // TAMANHO
25 // Complexidade: O(1)
26 pq.size();
27
28 // VAZIO
29 // Complexidade: O(1)
30 pq.empty();
31
32 // LIMPAR
33 // Complexidade: O(n)
34 pq.clear();
35
36 // ITERAR
37 // Complexidade: O(n)
38 for (auto x : pq) {}
39
40 // çãOrdemao por çãfuno customizada passada por parametro ao criar a pq
41 // Complexidade: O(n log(n))
42 auto cmp = [](int a, int b) { return a > b; };
43 priority_queue<int, vector<int>, decltype(cmp)> pq(cmp);

```

2.3 Set

```

1 set<int> st;
2
3 // Complexidade: O(log(n))
4 st.insert(x);
5 st.erase(x);
6 st.find(x);
7 st.erase(st.find(x));
8
9
10 // Complexidade: O(1)
11 st.size();
12 st.empty();
13
14 // Complexidade: O(n)
15 st.clear();
16 for (auto x : st) {}

```

17						
18	priority_queue			set		
19	-----+-----+-----+-----+-----+-----					
20	op	call	compl	call	compl	melhor
21	-----+-----+-----+-----+-----+-----					
22	insert	push	log(n)	insert	log(n)	pq
23	erase_menor	pop	log(n)	erase	log(n)	pq
24	get_menor	top	1	begin	1	set
25	get_maior	-	-	rbegin	1	set
26	erase_number	remove	n	erase	log(n)	set
27	find_number	-	-	find	log(n)	set
28	find_>=	-	-	lower	log(n)	set
29	find_<=	-	-	upper	log(n)	set
30	iterate	for	n	for	n	set
31	-----+-----+-----+-----+-----+-----					

2.4 Sort

```

1 vector<int> v;
2 // Sort Crescente:
3 sort(v.begin(), v.end());
4 sort(all(v));
5
6 // Sort Decrescente:
7 sort(v.rbegin(), v.rend());
8 sort(all(v), greater<int>());
9
10 // Sort por uma função:
11 auto cmp = [](int a, int b) { return a > b; }; // { 2, 3, 1 } -> { 3,
2, 1 }
12 auto cmp = [](int a, int b) { return a < b; }; // { 2, 3, 1 } -> { 1,
2, 3 }
13 sort(v.begin(), v.end(), cmp);
14 sort(all(v), cmp);
15
16 // Sort por uma função (comparação de pares):
17 auto cmp = [](pair<int, int> a, pair<int, int> b) { return a.second >
b.second; };
18
19 // Sort parcial:
20 partial_sort(v.begin(), v.begin() + n, v.end()); // sorta com n menos
elementos
21 partial_sort(v.rbegin(), v.rbegin() + n, s.rend()) // sorta com n
maiores elementos
22
23 // SORT VS SET
24 * para um input com elementos distintos, sort é mais rápido que set

```

2.5 String

```

1 // INICIALIZAR
2 string s; // string vazia
3 string s(n, c); // n cópias de c
4 string s(s); // cópia de s
5 string s(s, i, n); // cópia de s[i..i+n-1]
6

```

```

7 // SUBSTRING
8 // Complexidade: O(n)
9 s.substr(i, n); // substring de s[i..i+n-1]
10 s.substr(i, j - i + 1); // substring de s[i..j]
11
12 // TAMANHO
13 // Complexidade: O(1)
14 s.size(); // tamanho da string
15 s.empty(); // true se vazia, false se não vazia
16
17 // MODIFICAR
18 // Complexidade: O(n)
19 s.push_back(c); // adiciona c no final
20 s.pop_back(); // remove o último
21 s += t; // concatena t no final
22 s.insert(i, t); // insere t a partir da posição i
23 s.erase(i, n); // remove n caracteres a partir da posição i
24 s.replace(i, n, t); // substitui n caracteres a partir da posição i por t
25 s.swap(t); // troca o conteúdo com t
26
27 // COMPARAR
28 // Complexidade: O(n)
29 s == t; // igualdade
30 s != t; // diferença
31 s < t; // menor que
32 s > t; // maior que
33 s <= t; // menor ou igual
34 s >= t; // maior ou igual
35
36 // BUSCA
37 // Complexidade: O(n)
38 s.find(t); // posição da primeira ocorrência de t, ou string::npos se não
existe
39 s.rfind(t); // posição da última ocorrência de t, ou string::npos se não
existe
40 s.find_first_of(t); // posição da primeira ocorrência de um caractere de t
, ou string::npos se não existe
41 s.find_last_of(t); // posição da última ocorrência de um caractere de t,
ou string::npos se não existe
42 s.find_first_not_of(t); // posição do primeiro caractere que não está em t
, ou string::npos se não existe
43 s.find_last_not_of(t); // posição do último caractere que não está em t, ou
string::npos se não existe
44
45 // SUBSTITUIR
46 // Complexidade: O(n)
47 s.replace(i, n, t); // substitui n caracteres a partir da posição i por t
48 s.replace(s.begin() + i, s.begin() + i + n, t.begin(), t.end()); //
substitui n caracteres a partir da posição i por t
49 s.replace(s.begin() + i, s.begin() + i + n, t); // substitui n caracteres
a partir da posição i por t
50 s.replace(s.begin() + i, s.begin() + i + n, n, c); // substitui n
caracteres a partir da posição i por n cópias de c

```

2.6 Vector

```

1 // INICIALIZAR

```

```

2 vector<int> v (n); // n ócpias de 0
3 vector<int> v (n, v); // n ócpias de v
4
5 // PUSH_BACK
6 // Complexidade: O(1) amortizado (O(n) se realocar)
7 v.push_back(x);
8
9 // REMOVER
10 // Complexidade: O(n)
11 v.erase(v.begin() + i);
12
13 // INSERIR
14 // Complexidade: O(n)
15 v.insert(v.begin() + i, x);
16
17 // ORDENAR
18 // Complexidade: O(n log(n))
19 sort(v.begin(), v.end());
20 sort(all(v));
21
22 // BUSCA BINARIA
23 // Complexidade: O(log(n))
24 // Retorno: true se existe, false se ão existe
25 binary_search(v.begin(), v.end(), x);
26
27 // FIND
28 // Complexidade: O(n)
29 // Retorno: iterador para o elemento, v.end() se ão existe
30 find(v.begin(), v.end(), x);
31
32 // CONTAR
33 // Complexidade: O(n)
34 // Retorno: úmnero de êocorrncias
35 count(v.begin(), v.end(), x);

```

3 .vscode

4 Combinatoria

4.1 @ Factorial

```

1 // Calcula o fatorial de um úmnero n
2 // Complexidade: O(n)
3
4 int factdp[20];
5
6 int fact(int n) {
7     if (n < 2) return 1;
8     if (factdp[n] != 0) return factdp[n];
9     return factdp[n] = n * fact(n - 1);
10 }

```

4.2 @ Tabela

```

1 // Sequencia de p elementos de um total de n
2
3 ORDEM \ REPETIC | COM | SEM
4 -----+-----+-----
5 IMPORTA | ARRANJO COM REPETICAO | ARRANJO SIMPLES
6 NAO | COMBINACAO COM REPETICAO | COMBINACAO SIMPLES

```

4.3 Arranjo Com Repeticao

```

1 int arranjoComRepeticao(int p, int n) {
2     return pow(n, p);
3 }

```

4.4 Arranjo Simples

```

1 int arranjoSimples(int p, int n) {
2     return fact(n) / fact(n - p);
3 }

```

4.5 Catalan

```

1 const int MAX_N = 100010;
2 const int p = 1e9+7; // p is a prime > MAX_N
3
4 int mod(int a, int m) {
5     return ((a%m) + m) % m;
6 }
7
8 int inv(int a) {
9     return modPow(a, p-2, p);
10 }
11
12 int modPow(int b, int p, int m) {
13     if (p == 0) return 1;
14     int ans = modPow(b, p/2, m);
15     ans = mod(ans*ans, m);
16     if (p&1) ans = mod(ans*b, m);
17     return ans;
18 }
19
20 int Cat[MAX_N];
21
22 void solve() {
23     Cat[0] = 1;
24     for (int n = 0; n < MAX_N-1; ++n) // O(MAX_N * log p)
25         Cat[n+1] = ((4*n+2)%p * Cat[n]%p * inv(n+2)) % p;
26     cout << Cat[100000] << "\n"; // the answer is
27                                     945729344
28 }

```

4.6 Combinacao Com Repeticao

```

1 int combinacaoComRepeticao(int p, int n) {
2     return fact(n + p - 1) / (fact(p) * fact(n - 1));
3 }

```


4.7 Combinacao Simples

```
1 // Description: Calcula o valor de comb(n, k) % p, onde p é um primo > n.
2 // Complexidade: O(n)
3 const int MAX_N = 100010;
4 const int p = 1e9+7; // p is a prime > MAX_N
5
6 int mod(int a, int m) {
7     return ((a%m) + m) % m;
8 }
9
10 int modPow(int b, int p, int m) {
11     if (p == 0) return 1;
12     int ans = modPow(b, p/2, m);
13     ans = mod(ans*ans, m);
14     if (p&1) ans = mod(ans*b, m);
15     return ans;
16 }
17
18 int inv(int a) {
19     return modPow(a, p-2, p);
20 }
21
22 int fact[MAX_N];
23
24 int comb(int n, int k) {
25     if (n < k) return 0;
26     return (((fact[n] * inv(fact[k])) % p) * inv(fact[n-k])) % p;
27 }
28
29 void solve() {
30     fact[0] = 1;
31     for (int i = 1; i < MAX_N; ++i)
32         fact[i] = (fact[i-1]*i) % p;
33     cout << comb(3, 3) << "\n";
34 }
```

4.8 Permutacao Circular

```
1 // Permutacao objetos em posicao simetrica em um circulo
2
3 int permutacaoCircular(int n) {
4     return fact(n - 1);
5 }
```

4.9 Permutacao Com Repeticao

```
1 // Trocar elementos de lugar quando ha termos repetidos (ANAGRAMA)
2 int permutacaoComRepeticao(string s) {
3     int n = s.size();
4     int ans = fact(n);
5     map<char, int> freq;
6     for (char c : s) {
7         freq[c]++;
8     }
9     for (auto [c, f] : freq) {
```

```
10         ans /= fact(f);
11     }
12     return ans;
13 }
```

4.10 Permutacao Simples

```
1 // Agrupamentos distintos entre si pela ordem (FILA)
2 // Diferenca do arranjo: usa todos os elementos para o calculo
3 // SEM repeticao
4
5 int permutacaoSimples(int n) {
6     return fact(n);
7 }
```

5 DP

5.1 Dp

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 const int MAX_gm = 30; // up to 20 garments at most and 20 models/garment
5 const int MAX_M = 210; // maximum budget is 200
6
7 int M, C, price[MAX_gm][MAX_gm]; // price[g (<= 20)][k (<= 20)]
8 int memo[MAX_gm][MAX_M]; // TOP-DOWN: dp table [g (< 20)][money
9                             (<= 200)]
10
11 int dp(int g, int money) {
12     if (money < 0) return -1e9;
13     if (g == C) return M - money;
14     if (memo[g][money] != -1)
15         return memo[g][money]; // avaliar linha g com dinheiro money (cada
16                                 caso pensavel)
17     int ans = -1;
18     for (int k = 1; k <= price[g][0]; ++k)
19         ans = max(ans, dp(g + 1, money - price[g][k]));
20     return memo[g][money] = ans;
21 }
22
23 int main() {
24     int TC;
25     scanf("%d", &TC);
26     while (TC--)
27     {
28         scanf("%d %d", &M, &C);
29         for (int g = 0; g < C; ++g)
30         {
31             scanf("%d", &price[g][0]); // store k in price[g][0]
32             for (int k = 1; k <= price[g][0]; ++k)
33                 scanf("%d", &price[g][k]);
34         }
35         memset(memo, -1, sizeof memo); // TOP-DOWN: init memo
```

```

35     if (dp(0, M) < 0)
36         printf("no solution\n"); // start the top-down DP
37     else
38         printf("%d\n", dp(0, M));
39 }
40 return 0;
41 }

```

5.2 Livro Caixa

```

1 // Description: Problema do livro caixa: retorna a expressao que resulta
  em f
2 // Complexidade: O(n*f)
3 // Explicacao: parecido com mochila => dados n numeros, dita se existe uma
  expressao que resulta em f, dado que cada numero pode ser positivo,
  negativo ou nao utilizado
4 /*
5 5 7
6 1 2 3 4 5
7 => ?+??+
8 */
9
10 int f=1, n=1, entrada[MAX];
11 map<pii, bool> memo;
12 bool positivo[MAX], negativo[MAX];
13
14 bool dp(int id, int soma) {
15
16     if(id == n) return soma == f;
17     if(memo.count({soma, id})) return memo[{soma, id}];
18
19     bool pos = dp(id+1, soma+entrada[id]);
20     bool neg = dp(id+1, soma-entrada[id]);
21
22     if(pos and !neg) positivo[id] = true;
23     else if(!pos and neg) negativo[id] = true;
24     else if(pos and neg) positivo[id] = negativo[id] = true;
25     return memo[{soma, id}] = (pos or neg);
26 }
27
28 void solve() {
29
30     cin >> n >> f;
31     memo.clear();
32
33     f(i, 0, n) {
34         positivo[i] = negativo[i] = false;
35         cin >> entrada[i];
36     }
37
38     bool ans = dp(0, 0);
39
40     if(!ans) cout << "*";
41     else {
42         f(i, 0, n) {
43             bool pos = positivo[i], neg = negativo[i];
44             if(pos and neg) cout << "?";

```

```

45         else if(pos) cout << "+";
46         else cout << "-";
47     }
48 }
49 }

```

5.3 Mochila

```

1 // Description: Problema da mochila 0-1: retorna o valor maximo que pode
  ser carregado
2 // Complexidade: O(n*capacidade)
3
4 const int MAX_QNT_OBJETOS = 60; // 50 + 10
5 const int MAX_PESO_OBJETO = 1010; // 1000 + 10
6
7 int n, memo[MAX_QNT_OBJETOS][MAX_PESO_OBJETO];
8 vi valor, peso;
9
10 int mochila(int id, int remW) {
11     if ((id == n) || (remW == 0)) return 0;
12     int &ans = memo[id][remW];
13     if (ans != -1) return ans;
14     if (peso[id] > remW) return ans = mochila(id+1, remW);
15     return ans = max(mochila(id+1, remW), valor[id]+mochila(id+1, remW-
  peso[id]));
16 }
17
18 void solve() {
19
20     memset(memo, -1, sizeof memo);
21
22     int capacidadeMochila; cin >> capacidadeMochila;
23
24     f(i, 0, capacidadeMochila) { memo[0][i] = 0; } // testar com e sem essa
  linha
25
26     cin >> n;
27
28     valor.assign(n, 0);
29     peso.assign(n, 0);
30
31     f(i, 0, n) {
32         cin >> peso[i] >> valor[i];
33     }
34
35     cout << mochila(0, capacidadeMochila) << endl;
36 }

```

5.4 Mochila Eduardo

```

1 // Description: çãImplementao da mochila com çãreconstruo de çãsoluo
2 // Complexidade: O(n*capacidade)
3
4 int t[100][100]; // Defina os tamanhos conforme seu problema, pode usar
  vector
5

```

```

6 unordered_set<int> selecionados; // conjunto dos indices do itens que
   asero selecionados
7 int numItens;
8
9 // ps = pesos, vals = valores, fiz isso por legibilidade
10 int knapsack(int i, int cap, int ps[], int vals[]) {
11     if(cap < 0) return -0x3f3f3f3f;
12     if(i == numItens) return 0;
13     if(t[i][cap] > -1) return t[i][cap];
14
15     int x = knapsack(i + 1, cap, ps, vals);
16     int y = knapsack(i + 1, cap - ps[i], ps, vals) + vals[i];
17     return t[i][cap] = max(x, y);
18 }
19
20 // ps = pesos, vals = valores, fiz isso por legibilidade
21 void retrieve(int i, int cap, int ps[], int vals[]) {
22     if(i == numItens) return;
23
24     if(cap >= ps[i]) { // Dividi o if para legibilidade
25         if(knapsack(i + 1, cap, ps, vals) < knapsack(i + 1, cap - ps[i],
26 ps, vals) + vals[i]){
27             selecionados.insert(i);
28             return retrieve(i + 1, cap - ps[i], ps, vals);
29         }
30
31     return retrieve(i + 1, cap, ps, vals);
32 }
33
34 int main() {
35     memset(t, -1, sizeof t);
36
37     int capacidade = 6;
38     int pesos[] = {5, 4, 2}, valores[] = {500, 300, 250};
39     numItens = 3;
40
41     cout << knapsack(0, capacidade, pesos, valores) << endl;
42
43     retrieve(0, 6, pesos, valores);
44     for(auto i : selecionados) cout << i << ' ';
45     cout << endl;
46 }

```

6 Estruturas

6.1 Bittree

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 /*      n --> No. of elements present in input array.
5     BITree[0..n] --> Array that represents Binary Indexed Tree.
6     arr[0..n-1] --> Input array for which prefix sum is evaluated. */
7
8 // Returns sum of arr[0..index]. This function assumes

```

```

9 // that the array is preprocessed and partial sums of
10 // array elements are stored in BITree[].
11 int getSum(vector<int>& BITree, int index) {
12     int sum = 0;
13     index = index + 1;
14     while (index > 0) {
15         sum += BITree[index];
16         index -= index & (-index);
17     }
18     return sum;
19 }
20
21 void updateBIT(vector<int>& BITree, int n, int index, int val) {
22     index = index + 1;
23
24     while (index <= n) {
25         BITree[index] += val;
26         index += index & (-index);
27     }
28 }
29
30 vector<int> constructBITree(vector<int>& arr, int n) {
31     vector<int> BITree(n+1, 0);
32
33     for (int i=0; i<n; i++)
34         updateBIT(BITree, n, i, arr[i]);
35
36     return BITree;
37 }
38
39 void solve() {
40     vector<int> freq = {2, 1, 1, 3, 2, 3, 4, 5, 6, 7, 8, 9};
41     int n = freq.size();
42     vector<int> BITree = constructBITree(freq, n);
43     cout << "Sum of elements in arr[0..5] is" << getSum(BITree, 5);
44     // Let use test the update operation
45     freq[3] += 6;
46     updateBIT(BITree, n, 3, 6); // BIT[4] = 6
47
48     cout << "\nSum of elements in arr[0..5] after update is "
49         << getSum(BITree, 5);
50 }
51
52 int main() {
53     solve();
54     return 0;
55 }

```

6.2 Fenwick Tree

```

1 // BIT com update em range (Binary Indexed Tree)
2 //
3 // Operacoes 0-based
4 // query(l, r) retorna a soma de v[l..r]
5 // update(l, r, x) soma x em v[l..r]
6 //
7 // Complexidades:

```

```

8 // build - O(n)
9 // query - O(log(n))
10 // update - O(log(n))
11 namespace bit {
12     int bit[2][MAX+2];
13     int n;
14
15     void build(int n2, vector<int>& v) {
16         n = n2;
17         for (int i = 1; i <= n; i++)
18             bit[1][min(n+1, i+(i&-i))] += bit[1][i] += v[i];
19     }
20     int get(int x, int i) {
21         int ret = 0;
22         for (; i; i -= i&-i) ret += bit[x][i];
23         return ret;
24     }
25     void add(int x, int i, int val) {
26         for (; i <= n; i += i&-i) bit[x][i] += val;
27     }
28     int get2(int p) {
29         return get(0, p) * p + get(1, p);
30     }
31     int query(int l, int r) { // zero-based
32         return get2(r+1) - get2(l);
33     }
34     void update(int l, int r, int x) {
35         add(0, l+1, x), add(0, r+2, -x);
36         add(1, l+1, -x*l), add(1, r+2, x*(r+1));
37     }
38 };
39
40 void solve() {
41
42     vector<int> v {0,1,2,3,4,5}; // v[0] eh inutilizada
43     bit::build(v.size(), v);
44
45     int a = 0, b = 3;
46     bit::query(a, b); // v[a] + v[a+1] + ... + v[b] = 6 | 1+2+3 = 6 |
47     // zero-based
48     bit::update(a, b, 2); // v[a...b] += 2 | zero-based
49 }

```

6.3 Seg Tree

```

1 // SegTree
2 //
3 // Recursiva com Lazy Propagation
4 // Query: soma do range [a, b]
5 // Update: soma x em cada elemento do range [a, b]
6 // Pode usar a seguinte funcao para indexar os nohs:
7 // f(l, r) = (l+r)|(l!=r), usando 2N de memoria
8 //
9 // Complexidades:
10 // build - O(n)
11 // query - O(log(n))
12 // update - O(log(n))

```

```

13
14 #include<bits/stdc++.h>
15 using namespace std;
16
17 const int MAX = 1e5+10;
18
19 namespace SegTree {
20     int seg[4*MAX], lazy[4*MAX];
21     int n, *v;
22
23     int op(int a, int b) { return a + b; }
24
25     int build(int p=1, int l=0, int r=n-1) {
26         lazy[p] = 0;
27         if (l == r) return seg[p] = v[l];
28         int m = (l+r)/2;
29         return seg[p] = op(build(2*p, l, m), build(2*p+1, m+1, r));
30     }
31     void build(int n2, int* v2) {
32         n = n2, v = v2;
33         build();
34     }
35     void prop(int p, int l, int r) {
36         seg[p] += lazy[p]*(r-l+1);
37         if (l != r) lazy[2*p] += lazy[p], lazy[2*p+1] += lazy[p];
38         lazy[p] = 0;
39     }
40     int query(int a, int b, int p=1, int l=0, int r=n-1) {
41         prop(p, l, r);
42         if (a <= l and r <= b) return seg[p];
43         if (b < l or r < a) return 0;
44         int m = (l+r)/2;
45         return op(query(a, b, 2*p, l, m), query(a, b, 2*p+1, m+1, r));
46     }
47     int update(int a, int b, int x, int p=1, int l=0, int r=n-1) {
48         prop(p, l, r);
49         if (a <= l and r <= b) {
50             lazy[p] += x;
51             prop(p, l, r);
52             return seg[p];
53         }
54         if (b < l or r < a) return seg[p];
55         int m = (l+r)/2;
56         return seg[p] = op(update(a, b, x, 2*p, l, m), update(a, b, x, 2*p
57 +1, m+1, r));
58     }
59
60     // Se tiver uma seg de max, da pra descobrir em O(log(n))
61     // o primeiro e ultimo elemento >= val numa range:
62
63     // primeira posicao >= val em [a, b] (ou -1 se nao tem)
64     int get_left(int a, int b, int val, int p=1, int l=0, int r=n-1) {
65         prop(p, l, r);
66         if (b < l or r < a or seg[p] < val) return -1;
67         if (r == l) return l;
68         int m = (l+r)/2;
69         int x = get_left(a, b, val, 2*p, l, m);

```

```

69     if (x != -1) return x;
70     return get_left(a, b, val, 2*p+1, m+1, r);
71 }
72
73 // ultima posicao >= val em [a, b] (ou -1 se nao tem)
74 int get_right(int a, int b, int val, int p=1, int l=0, int r=n-1) {
75     prop(p, l, r);
76     if (b < l or r < a or seg[p] < val) return -1;
77     if (r == l) return l;
78     int m = (l+r)/2;
79     int x = get_right(a, b, val, 2*p+1, m+1, r);
80     if (x != -1) return x;
81     return get_right(a, b, val, 2*p, l, m);
82 }
83
84 // Se tiver uma seg de soma sobre um array nao negativo v, da pra
85 // descobrir em O(log(n)) o maior j tal que v[i]+v[i+1]+...+v[j-1] <
86 // val
87 int lower_bound(int i, int& val, int p, int l, int r) {
88     prop(p, l, r);
89     if (r < i) return n;
90     if (i <= l and seg[p] < val) {
91         val -= seg[p];
92         return n;
93     }
94     if (l == r) return l;
95     int m = (l+r)/2;
96     int x = lower_bound(i, val, 2*p, l, m);
97     if (x != n) return x;
98     return lower_bound(i, val, 2*p+1, m+1, r);
99 }
100
101 void solve() {
102     int n = 10;
103     int v[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
104     SegTree::build(n, v);
105
106     cout << SegTree::query(0, 9) << endl; // seg[0] + seg[1] + ... + seg
107     [9] = 55
108     SegTree::update(0, 9, 1); // seg[0,...,9] += 1
109 }

```

6.4 Sparse Table Disjunta

```

1 // Description: Sparse Table Disjunta para soma de intervalos
2 // Complexity Temporal: O(n log n) para construir e O(1) para consultar
3 // Complexidade Espacial: O(n log n)
4
5 #include <bits/stdc++.h>
6 using namespace std;
7
8 #define MAX 100010
9 #define MAX2 20 // log(MAX)
10
11 namespace SparseTable {
12     int m[MAX2][2*MAX], n, v[2*MAX];

```

```

13     int op(int a, int b) { return a + b; }
14     void build(int n2, int* v2) {
15         n = n2;
16         for (int i = 0; i < n; i++) v[i] = v2[i];
17         while (n&(n-1)) n++;
18         for (int j = 0; (1<<j) < n; j++) {
19             int len = 1<<j;
20             for (int c = len; c < n; c += 2*len) {
21                 m[j][c] = v[c], m[j][c-1] = v[c-1];
22                 for (int i = c+1; i < c+len; i++) m[j][i] = op(m[j][i-1],
23                     v[i]);
24                 for (int i = c-2; i >= c-len; i--) m[j][i] = op(v[i], m[j]
25                     ][i+1]);
26             }
27         }
28     int query(int l, int r) {
29         if (l == r) return v[l];
30         int j = __builtin_clz(1) - __builtin_clz(l^r);
31         return op(m[j][l], m[j][r]);
32     }
33
34 void solve() {
35     int n = 9;
36     int v[] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
37     SparseTable::build(n, v);
38     cout << SparseTable::query(0, n-1) << endl; // sparse[0] + sparse[1] +
39     ... + sparse[n-1] = 45

```

6.5 Tabuleiro

```

1 // Description: Estrutura que simula um tabuleiro M x N, sem realmente
2 // criar uma matriz
3 // Permite atribuir valores a linhas e colunas, e consultar a
4 // çãoposio mais frequente
5 // Complexidade Atribuir: O(log(N))
6 // Complexidade Consulta: O(log(N))
7 // Complexidade verificar frequencia geral: O(N * log(N))
8 #define MAX_VAL 5 // maior valor que pode ser adicionado na matriz + 1
9
10 class BinTree {
11     protected:
12         vector<int> mBin;
13     public:
14         explicit BinTree(int n) { mBin = vector(n + 1, 0); }
15
16         void add(int p, const int val) {
17             for (auto size = mBin.size(); p < size; p += p & -p)
18                 mBin[p] += val;
19         }
20
21         int query(int p) {
22             int sumToP {0};
23             for (; p > 0; p -= p & -p)
24                 sumToP += mBin[p];

```

```

23         return sumToP;
24     }
25 };
26
27 class ReverseBinTree : public BinTree {
28 public:
29     explicit ReverseBinTree(int n) : BinTree(n) {};
30
31     void add(int p, const int val) {
32         BinTree::add(static_cast<int>(mBin.size()) - p, val);
33     }
34
35     int query(int p) {
36         return BinTree::query(static_cast<int>(mBin.size()) - p);
37     }
38 };
39
40 class Tabuleiro {
41 public:
42     explicit Tabuleiro(const int m, const int n, const int q) : mM(m),
43         mN(n), mQ(q) {
44         mLinhas = vector<pair<int, int8_t>>(m, {0, 0});
45         mColunas = vector<pair<int, int8_t>>(n, {0, 0});
46
47         mAtribuiçoesLinhas = vector(MAX_VAL, ReverseBinTree(mQ)); //
48         mAtribuiçoesColunas = vector(MAX_VAL, ReverseBinTree(mQ));
49
50     void atribuirLinha(const int x, const int8_t r) {
51         mAtribuirFileira(x, r, mLinhas, mAtribuiçoesLinhas);
52     }
53
54     void atribuirColuna(const int x, const int8_t r) {
55         mAtribuirFileira(x, r, mColunas, mAtribuiçoesColunas);
56     }
57
58     int maxPosLinha(const int x) {
59         return mMaxPosFileira(x, mLinhas, mAtribuiçoesColunas, mM);
60     }
61
62     int maxPosColuna(const int x) {
63         return mMaxPosFileira(x, mColunas, mAtribuiçoesLinhas, mN);
64     }
65
66     vector<int> frequenciaElementos() {
67         vector<int> frequenciaGlobal(MAX_VAL, 0);
68         for(int i=0; i<mM; i++) {
69             vector<int> curr = frequenciaElementos(i,
70 mAtribuiçoesColunas);
71             for(int j=0; j<MAX_VAL; j++)
72                 frequenciaGlobal[j] += curr[j];
73         }
74         return frequenciaGlobal;
75     }
76

```

```

77 private:
78     int mM, mN, mQ, mMoment {0};
79
80     vector<ReverseBinTree> mAtribuiçoesLinhas, mAtribuiçoesColunas;
81     vector<pair<int, int8_t>> mLinhas, mColunas;
82
83     void mAtribuirFileira(const int x, const int8_t r, vector<pair<int
84 , int8_t>>& fileiras,
85         vector<ReverseBinTree>& atribuiçoes) {
86         if (auto& [oldQ, oldR] = fileiras[x]; oldQ)
87             atribuiçoes[oldR].add(oldQ, -1);
88
89         const int currentMoment = ++mMoment;
90         fileiras[x].first = currentMoment;
91         fileiras[x].second = r;
92         atribuiçoes[r].add(currentMoment, 1);
93     }
94
95     int mMaxPosFileira(const int x, const vector<pair<int, int8_t>>&
96 fileiras, vector<ReverseBinTree>& atribuiçoesPerpendiculares, const
97 int& currM) const {
98         auto [momentoAtribuiçãoFileira, rFileira] = fileiras[x];
99
100         vector<int> fileiraFrequencia(MAX_VAL, 0);
101         fileiraFrequencia[rFileira] = currM;
102
103         for (int8_t r {0}; r < MAX_VAL; ++r) {
104             const int frequenciaR = atribuiçoesPerpendiculares[r].
105 query(momentoAtribuiçãoFileira + 1);
106             fileiraFrequencia[rFileira] -= frequenciaR;
107             fileiraFrequencia[r] += frequenciaR;
108         }
109
110         return MAX_VAL - 1 - (max_element(fileiraFrequencia.cbegin(),
111 fileiraFrequencia.crend()) - fileiraFrequencia.cbegin());
112     }
113
114     vector<int> frequenciaElementos(int x, vector<ReverseBinTree>&
115 atribuiçoesPerpendiculares) const {
116         vector<int> fileiraFrequencia(MAX_VAL, 0);
117
118         auto [momentoAtribuiçãoFileira, rFileira] = mLinhas[x];
119
120         fileiraFrequencia[rFileira] = mN;
121
122         for (int8_t r {0}; r < MAX_VAL; ++r) {
123             const int frequenciaR = atribuiçoesPerpendiculares[r].
124 query(momentoAtribuiçãoFileira + 1);
125             fileiraFrequencia[rFileira] -= frequenciaR;
126             fileiraFrequencia[r] += frequenciaR;
127         }
128
129         return fileiraFrequencia;
130     }
131 };

```

```

127 void solve() {
128
129     int L, C, q; cin >> L >> C >> q;
130
131     Tabuleiro tabuleiro(L, C, q);
132
133     int linha = 0, coluna = 0, valor = 10; // linha e coluna sao 0 based
134     tabuleiro.atribuirLinha(linha, static_cast<int8_t>(valor)); // f(i,0,C)
135     matriz[linha][i] = valor
136     tabuleiro.atribuirColuna(coluna, static_cast<int8_t>(valor)); // f(i
137     ,0,L) matriz[i][coluna] = valor
138
139     // Frequencia de todos os elementos, de 0 a MAX_VAL-1
140     vector<int> frequenciaGeral = tabuleiro.frequenciaElementos();
141
142     int a = tabuleiro.maxPosLinha(linha); // retorna a posicao do elemento
143     mais frequente na linha
144     int b = tabuleiro.maxPosColuna(coluna); // retorna a posicao do
145     elemento mais frequente na coluna

```

6.6 Union Find

```

1 // Description: Union-Find (Disjoint Set Union)
2
3 typedef vector<int> vi;
4
5 struct UnionFind {
6     vi p, rank, setSize;
7     int numSets;
8     UnionFind(int N) {
9         p.assign(N, 0);
10        for (int i = 0; i < N; ++i)
11            p[i] = i;
12        rank.assign(N, 0);
13        setSize.assign(N, 1);
14        numSets = N;
15    }
16
17    // Retorna o numero de sets disjuntos (separados)
18    int numDisjointSets() { return numSets; }
19    // Retorna o tamanho do set que contem o elemento i
20    int sizeOfSet(int i) { return setSize[find(i)]; }
21
22    int find(int i) { return (p[i] == i) ? i : (p[i] = find(p[i])); }
23    bool same(int i, int j) { return find(i) == find(j); }
24    void uni(int i, int j) {
25        if (same(i, j))
26            return;
27        int x = find(i), y = find(j);
28        if (rank[x] > rank[y])
29            swap(x, y);
30        p[x] = y;
31        if (rank[x] == rank[y])
32            ++rank[y];
33        setSize[y] += setSize[x];

```

```

34        --numSets;
35    }
36 };
37
38 void solve() {
39     int n; cin >> n;
40     UnionFind UF(n);
41     UF.uni(0, 1);

```

7 Geometria

7.1 3D - Distancia Entre 2 Poliedros

```

1 // Description: Calcula a menor distancia entre dois poliedros convexos
2 // Complexidade:  $O(n^2 * m^2)$ , onde n e m sao o numero de vertices dos
3 // poliedros
4 // OBS: apenas testado para tetraedros
5
6 const double EPS = 1e-9;
7 const double INF = 1e50;
8
9 int cmpD(double a, double b = 0.0) { return a+EPS < b ? -1 : a-EPS > b; }
10
11 struct Point {
12     double x, y, z;
13     Point(double a=0.0, double b=0.0, double c=0.0) { x=a, y=b, z=c; }
14     Point operator+(const Point &P) const { return Point(x+P.x, y+P.y, z+P.z); }
15     Point operator-(const Point &P) const { return Point(x-P.x, y-P.y, z-P.z); }
16     Point operator*(double c) const { return Point(x*c, y*c, z*c); }
17     Point operator/(double c) const { return Point(x/c, y/c, z/c); }
18     double operator!() const { return sqrt(x*x+y*y+z*z); } // modulo
19
20 double produtoEscalar(Point A, Point B) { return A.x*B.x + A.y*B.y + A.z*B.z; }
21
22 Point produtoVetorial(Point A, Point B) { return Point(A.y*B.z-A.z*B.y, A.z*B.x-A.x*B.z, A.x*B.y-A.y*B.x); }
23
24 Point projWEmV(Point W, Point V) { return V * produtoEscalar(W,V) / produtoEscalar(V,V); }
25
26 // check if segments AB and CD have an intersection
27 bool checkIfSegmentsIntercept(Point A, Point B, Point C, Point D) {
28     return cmpD(produtoEscalar(produtoVetorial(A-B,C-B), produtoVetorial(A-B,D-B))) <= 0 &&
29     cmpD(produtoEscalar(produtoVetorial(C-D,A-D), produtoVetorial(C-D,B-D))) <= 0;
30 }
31
32 // distance between point P and segment AB
33 double dist_Point_seg(Point P, Point A, Point B) {
34     Point PP = A + projWEmV(P-A,B-A);

```

```

35 if (cmpD(!(A-PP)+!(PP-B),!(A-B)) == 0) return !(P-PP); //distance Points2
36 -line!
37 return min(!(P-A),!(P-B));
38 }
39 // distance between segments AB and CD
40 double dist_seg_seg(Point A, Point B, Point C, Point D) {
41     Point E = projWEmV(A-D,produtoVetorial(B-A,D-C));
42     if (checkIfSegmentsIntercept(A,B,C+E,D+E)) return !E;
43     return min( min( dist_Point_seg(A,C,D), dist_Point_seg(B,C,D) ),
44         min( dist_Point_seg(C,A,B), dist_Point_seg(D,A,B) ) );
45 }
46
47 // distance between point P and triangle ABC
48 double dist_Point_tri(Point P, Point A, Point B, Point C) {
49     Point N = produtoVetorial(A-C,B-C);
50     Point PP = P + projWEmV(C-P,N);
51     Point V1 = produtoVetorial(PP-A,B-A);
52     Point V2 = produtoVetorial(PP-B,C-B);
53     Point V3 = produtoVetorial(PP-C,A-C);
54     if (cmpD(produtoEscalar(V1,V2)) >= 0 && cmpD(produtoEscalar(V1,V3)) >= 0
55         && cmpD(produtoEscalar(V2,V3)) >= 0)
56         return !(PP-P); // distance Point-plane!
57     return min(dist_Point_seg(P,A,B),min(dist_Point_seg(P,A,C),
58         dist_Point_seg(P,B,C)));
59 }
60 // Calcula a menor distancia entre dois poliedros
61 // Complexidade: O(sz1^2 * sz2^2), onde sz1 e sz2 sao o numero de vertices
62 // dos poliedros
63 double distanciaPoliedroPoliedro(Point T1[], int sz1, Point T2[], int sz2)
64 {
65     double ans = INF;
66
67     // itera por todos os pares de arestas dos dois tetraedros, e calcula
68     // a distancia entre os segmentos gerados por estes pares
69     for (int i=0; i < sz1; i++) // arestas -> arestas
70         for (int j=i+1; j < sz1; j++)
71             for (int ii=0; ii < sz2; ii++)
72                 for (int jj=ii+1; jj < sz2; jj++)
73                     ans = min( ans, dist_seg_seg(T1[i],T1[j],T2[ii],T2[jj]
74 )) );
75
76     // itera por todos os pontos de um tetraedro e calcula a distancia
77     // entre estes pontos e as faces triangulares do outro tetraedro
78     for (int i=0; i < sz1; i++)
79         for (int j=i+1; j < sz1; j++)
80             for (int k=j+1; k < sz1; k++)
81                 for (int x=0; x < sz2; x++)
82                     ans = min( ans, dist_Point_tri(T1[x],T2[i],T2[j],T2[k]
83 )) ),
84                     ans = min( ans, dist_Point_tri(T2[x],T1[i],T1[j],T1[k]
85 )) );
86     return ans;
87 }
88
89 void solve() {
90     Point poliedro1[5], poliedro2[5];
91
92     f(i,0,4) { auto& [x,y,z] = poliedro1[i]; scanf("%lf %lf %lf", &x, &y,
93         &z); }
94     f(i,0,4) { auto& [x,y,z] = poliedro2[i]; scanf("%lf %lf %lf", &x, &y,
95         &z); }
96
97     printf("%.2lf\n",distanciaPoliedroPoliedro(poliedro1, 4, poliedro2, 4)
98 );
99 }

```

7.2 Andrew

```

1 // Nome: Convex Hull - Andrew's Monotone Chain
2 // Description: Calcula o perimetro do menor poligono convexo que contem
3 // todos os pontos
4 // Complexidade: O(n log n)
5
6 int produto_vetoril(pair<int,int> a,pair<int,int> b,pair<int,int> novo){
7     return (b.first - a.first)*(novo.second-b.second) -(b.second - a.
8         second)*(novo.first - b.first);
9 }
10 double distancia(pair<int,int> a,pair<int,int> b){
11     return sqrt(pow((a.first - b.first), 2) + pow((a.second - b.second),
12         2));
13 }
14 double andrew(pair<int,int> pontos[], int n) {
15     vector<pair<int,int>> hull;
16
17     pair<int,int> ponto;
18     int k=0;
19     f(i,0,n) {
20         while(k>=2 and produto_vetoril(hull[k-2],hull[k-1],pontos[i]) <=
21             0) {
22             hull.pop_back();
23             k--;
24         }
25         hull.push_back(pontos[i]);
26         k++;
27     }
28     for(int i=n-1, tam = k+1 ; i>=0 ; i--) {
29         while(k>=tam && produto_vetoril(hull[k-2],hull[k-1],pontos[i])<=0)
30         {
31             hull.pop_back();
32             k--;
33         }
34         hull.push_back(pontos[i]);
35         k++;
36     }
37     double perimetro = 0;
38

```



```

39     f(i,1,hull.size()) {
40         perimetro += distancia(hull[i-1],hull[i]);
41     }
42
43     return perimetro;
44 }
45
46 void solve() {
47
48     int n; scanf("%lld",&n);
49     pair<int,int> pontos[n];
50
51     for(auto& [x, y] : pontos)
52         scanf("%lld %lld",&x,&y);
53
54     sort(pontos,pontos+n);
55
56     double perimetro = andrew(pontos,n);
57 }

```

7.3 Circulo

```

1 #include <bits/stdc++.h>
2 #include "ponto.cpp"
3 using namespace std;
4
5 // Retorna se o ponto p esta dentro, fora ou na circunferencia de centro c
   e raio r
6 int insideCircle(const point_i &p, const point_i &c, int r) {
7     int dx = p.x-c.x, dy = p.y-c.y;
8     int Euc = dx*dx + dy*dy, rSq = r*r; // all integer
9     return Euc < rSq ? 1 : (Euc == rSq ? 0 : -1); // in/border/out
10 }
11
12 // Determina o centro e raio de um circulo que passa por 3 pontos
13 bool circle2PtsRad(point p1, point p2, double r, point &c) {
14     double d2 = (p1.x-p2.x) * (p1.x-p2.x) +
15             (p1.y-p2.y) * (p1.y-p2.y);
16     double det = r*r / d2 - 0.25;
17     if (det < 0.0) return false;
18     double h = sqrt(det);
19     c.x = (p1.x+p2.x) * 0.5 + (p1.y-p2.y) * h;
20     c.y = (p1.y+p2.y) * 0.5 + (p2.x-p1.x) * h;
21     return true;
22 }

```

7.4 Closestpair Otimizado

```

1
2 // A structure to represent a Point in 2D plane
3 struct Point
4 {
5     int x, y;
6 };
7 // Needed to sort array of points according to X coordinate
8 int compareX(const void* a, const void* b)

```

```

9 {
10     Point *p1 = (Point *)a, *p2 = (Point *)b;
11     return (p1->x != p2->x) ? (p1->x - p2->x) : (p1->y - p2->y);
12 }
13 // Needed to sort array of points according to Y coordinate
14 int compareY(const void* a, const void* b)
15 {
16     Point *p1 = (Point *)a, *p2 = (Point *)b;
17     return (p1->y != p2->y) ? (p1->y - p2->y) : (p1->x - p2->x);
18 }
19 // A utility function to find the distance between two points
20 float dist(Point p1, Point p2)
21 {
22     return sqrt( (p1.x - p2.x)*(p1.x - p2.x) +
23                 (p1.y - p2.y)*(p1.y - p2.y)
24                 );
25 }
26 // A Brute Force method to return the smallest distance between two points
27 // in P[] of size n
28 float bruteForce(Point P[], int n){
29     float min = FLT_MAX;
30     for (int i = 0; i < n; ++i)
31         for (int j = i+1; j < n; ++j)
32             if (dist(P[i], P[j]) < min)
33                 min = dist(P[i], P[j]);
34     return min;
35 }
36 // A utility function to find a minimum of two float values
37 float min(float x, float y)
38 {
39     return (x < y)? x : y;
40 }
41 // A utility function to find the distance between the closest points of
42 // strip of a given size. All points in strip[] are sorted according to
43 // y coordinate. They all have an upper bound on minimum distance as d.
44 // Note that this method seems to be a O(n^2) method, but it's a O(n)
45 // method as the inner loop runs at most 6 times
46 float stripClosest(Point strip[], int size, float d){
47     float min = d; // Initialize the minimum distance as d
48     // Pick all points one by one and try the next points till the
49     // difference
50     // between y coordinates is smaller than d.
51     // This is a proven fact that this loop runs at most 6 times
52     for (int i = 0; i < size; ++i)
53         for (int j = i+1; j < size && (strip[j].y - strip[i].y) < min; ++j)
54             if (dist(strip[i],strip[j]) < min)
55                 min = dist(strip[i], strip[j]);
56     return min;
57 }
58 // A recursive function to find the smallest distance. The array Px
59 // contains
60 // all points sorted according to x coordinates and Py contains all points
61 // sorted according to y coordinates
62 float closestUtil(Point Px[], Point Py[], int n){
63     // If there are 2 or 3 points, then use brute force

```

```

63 if (n <= 3)
64     return bruteForce(Px, n);
65 // Find the middle point
66 int mid = n/2;
67 Point midPoint = Px[mid];
68 // Divide points in y sorted array around the vertical line.
69 // Assumption: All x coordinates are distinct.
70 Point Pyl[mid]; // y sorted points on left of vertical line
71 Point Pyr[n-mid]; // y sorted points on right of vertical line
72 int li = 0, ri = 0; // indexes of left and right subarrays
73 for (int i = 0; i < n; i++)
74 {
75     if ((Py[i].x < midPoint.x || (Py[i].x == midPoint.x && Py[i].y <
midPoint.y)) && li < mid)
76         Pyl[li++] = Py[i];
77     else
78         Pyr[ri++] = Py[i];
79 }
80 // Consider the vertical line passing through the middle point
81 // calculate the smallest distance dl on left of middle point and
82 // dr on right side
83 float dl = closestUtil(Px, Pyl, mid);
84 float dr = closestUtil(Px + mid, Pyr, n-mid);
85 // Find the smaller of two distances
86 float d = min(dl, dr);
87 // Build an array strip[] that contains points close (closer than d)
88 // to the line passing through the middle point
89 Point strip[n];
90 int j = 0;
91 for (int i = 0; i < n; i++)
92     if (abs(Py[i].x - midPoint.x) < d)
93         strip[j] = Py[i], j++;
94 // Find the closest points in strip. Return the minimum of d and
closest
95 // distance is strip[]
96 return stripClosest(strip, j, d);
97 }
98 // The main function that finds the smallest distance
99 // This method mainly uses closestUtil()
100 float closest(Point P[], int n){
101     Point Px[n];
102     Point Py[n];
103     for (int i = 0; i < n; i++)
104     {
105         Px[i] = P[i];
106         Py[i] = P[i];
107     }
108     qsort(Px, n, sizeof(Point), compareX);
109     qsort(Py, n, sizeof(Point), compareY);
110     // Use recursive function closestUtil() to find the smallest distance
111     return closestUtil(Px, Py, n);
112 }
113
114 int main(){
115     Point P[] = {{2, 3}, {12, 30}, {40, 50}, {5, 1}, {12, 10}, {3, 4}};
116     int n = sizeof(P) / sizeof(P[0]);
117     cout << "The smallest distance is " << closest(P, n);

```

```

118     return 0;
119 }

```

7.5 Geometricosgerai

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 struct Point {
5     double x, y;
6 };
7 //checa se dado ponto esta dentro de um poligno.
8 //tempo O(NxM) sendo N=numero de pontos do poligno, M= a quantia de pontos
que vc quer checar
9 bool point_in_polygon(Point point, vector<Point> polygon){
10     int num_vertices = polygon.size();
11     double x = point.x, y = point.y;
12     bool inside = false;
13     Point p1 = polygon[0], p2;
14     for (int i = 1; i <= num_vertices; i++) {
15         p2 = polygon[i % num_vertices];
16         if (y > min(p1.y, p2.y)) {
17             if (y <= max(p1.y, p2.y)) {
18                 if (x <= max(p1.x, p2.x)) {
19                     double x_intersection
20                         = (y - p1.y) * (p2.x - p1.x)
21                           / (p2.y - p1.y)
22                       + p1.x;
23                     if (p1.x == p2.x
24                         || x <= x_intersection) {
25                         inside = !inside;
26                     }
27                 }
28             }
29         }
30         p1 = p2;
31     }
32     return inside;
33 }
34 //dado N pontos ordenados, encontre a area do poligno
35 double polygonArea(vector<pair<double,double>> vec )
36 {
37     // Initialize area
38     double area = 0.0;
39     // Calculate value of shoelace formula
40     int j = vec.size() - 1;
41     for (int i = 0; i < vec.size(); i++)
42     {
43         area += (vec[j].first + vec[i].first) * (vec[j].second - vec[i].
second);
44         j = i; // j is previous vertex to i
45     }
46     // Return absolute value
47     return abs(area / 2.0);
48 }
49
50 //encontrar area de intersecao entre dois circulos

```

```

51 //(x,y)posicao do centro + raio
52 long long int intersectionArea(long double X1, long double Y1,
53                               long double R1, long double X2,
54                               long double Y2, long double R2){
55     long double Pi = 3.14;
56     long double d, alpha, beta, a1, a2;
57     long long int ans;
58
59     // Calculate the euclidean distance
60     // between the two points
61     d = sqrt((X2 - X1) * (X2 - X1) + (Y2 - Y1) * (Y2 - Y1));
62
63     if (d > R1 + R2)
64         ans = 0;
65
66     else if (d <= (R1 - R2) && R1 >= R2)
67         ans = floor(Pi * R2 * R2);
68
69     else if (d <= (R2 - R1) && R2 >= R1)
70         ans = floor(Pi * R1 * R1);
71
72     else {
73         alpha = acos((R1 * R1 + d * d - R2 * R2)
74                     / (2 * R1 * d))
75                 * 2;
76         beta = acos((R2 * R2 + d * d - R1 * R1)
77                   / (2 * R2 * d))
78                * 2;
79         a1 = 0.5 * beta * R2 * R2
80             - 0.5 * R2 * R2 * sin(beta);
81         a2 = 0.5 * alpha * R1 * R1
82             - 0.5 * R1 * R1 * sin(alpha);
83         ans = floor(a1 + a2);
84     }
85
86     return ans;
87 }

```

7.6 Leis

```

1 // Lei dos Cossenos: a^2 = b^2 + c^2 - 2bc*cos(A)
2 // Lei dos Senos: a/sen(A) = b/sen(B) = c/sen(C) = 2R
3 // Pitagoras: a^2 = b^2 + c^2

```

7.7 Linha

```

1 #include <bits/stdc++.h>
2 #include "ponto.cpp"
3 using namespace std;
4
5 // const int EPS = 1e-9;
6
7 struct line { double a, b, c; }; // ax + by + c = 0
8
9 // Gera a equacao da reta que passa por 2 pontos
10 void pointsToLine(point p1, point p2, line &l) {

```

```

11     if (fabs(p1.x-p2.x) < EPS)
12         l = {1.0, 0.0, -p1.x};
13     else {
14         double a = -(double)(p1.y-p2.y) / (p1.x-p2.x);
15         l = {a, 1.0, -(double)(a*p1.x) - p1.y};
16     }
17 }
18
19 // Gera a equacao da reta que passa por um ponto e tem inclinacao m
20 void pointSlopeToLine(point p, double m, line &l) { // m < Inf
21     l = {m, 1.0, -((m * p.x) + p.y)};
22 }
23
24 // Checa se 2 retas sao paralelas
25 bool areParallel(line l1, line l2) {
26     return (fabs(l1.a-l2.a) < EPS) and (fabs(l1.b-l2.b) < EPS);
27 }
28
29 // Checa se 2 retas sao iguais
30 bool areSame(line l1, line l2) {
31     return areParallel(l1, l2) and (fabs(l1.c-l2.c) < EPS);
32 }
33
34 // Retorna se 2 retas se intersectam e o ponto de interseccao (referencia)
35 bool areIntersect(line l1, line l2, point &p) {
36     if (areParallel(l1, l2)) return false;
37
38     p.x = (l2.b*l1.c - l1.b*l2.c) / (l2.a*l1.b - l1.a*l2.b);
39     if (fabs(l1.b) > EPS) p.y = -(l1.a*p.x + l1.c);
40     else
41         p.y = -(l2.a*p.x + l2.c);
42     return true;
43 }

```

7.8 Maior Poligono Convexo

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 const double EPS = 1e-9;
5
6 double DEG_to_RAD(double d) { return d*M_PI / 180.0; }
7
8 double RAD_to_DEG(double r) { return r*180.0 / M_PI; }
9
10 struct point {
11     double x, y;
12     point() { x = y = 0.0; }
13     point(double _x, double _y) : x(_x), y(_y) {}
14     bool operator == (point other) const {
15         return (fabs(x-other.x) < EPS && (fabs(y-other.y) < EPS));
16     }
17
18     bool operator <(const point &p) const {
19         return x < p.x || (abs(x-p.x) < EPS && y < p.y);
20     }
21 };
22

```

```

23 struct vec {
24     double x, y;
25     vec(double _x, double _y) : x(_x), y(_y) {}
26 };
27
28 vec toVec(point a, point b) { return vec(b.x-a.x, b.y-a.y); }
29
30 double dist(point p1, point p2) { return hypot(p1.x-p2.x, p1.y-p2.y); }
31
32 // returns the perimeter of polygon P, which is the sum of Euclidian
33 // distances of consecutive line segments (polygon edges)
34 double perimeter(const vector<point> &P) {
35     double ans = 0.0;
36     for (int i = 0; i < (int)P.size()-1; ++i)
37         ans += dist(P[i], P[i+1]);
38     return ans;
39 }
40
41 // returns the area of polygon P
42 double area(const vector<point> &P) {
43     double ans = 0.0;
44     for (int i = 0; i < (int)P.size()-1; ++i)
45         ans += (P[i].x*P[i+1].y - P[i+1].x*P[i].y);
46     return fabs(ans)/2.0;
47 }
48
49 double dot(vec a, vec b) { return (a.x*b.x + a.y*b.y); }
50 double norm_sq(vec v) { return v.x*v.x + v.y*v.y; }
51
52 // returns angle aob in rad
53 double angle(point a, point o, point b) {
54     vec oa = toVec(o, a), ob = toVec(o, b);
55     return acos(dot(oa, ob) / sqrt(norm_sq(oa) * norm_sq(ob)));
56 }
57
58 double cross(vec a, vec b) { return a.x*b.y - a.y*b.x; }
59
60 // returns the area of polygon P, which is half the cross products of
61 // vectors defined by edge endpoints
62 double area_alternative(const vector<point> &P) {
63     double ans = 0.0; point O(0.0, 0.0);
64     for (int i = 0; i < (int)P.size()-1; ++i)
65         ans += cross(toVec(O, P[i]), toVec(O, P[i+1]));
66     return fabs(ans)/2.0;
67 }
68
69 // note: to accept collinear points, we have to change the '> 0'
70 // returns true if point r is on the left side of line pq
71 bool ccw(point p, point q, point r) { return cross(toVec(p, q), toVec(p, r)) > 0; }
72
73 // returns true if point r is on the same line as the line pq
74 bool collinear(point p, point q, point r) { return fabs(cross(toVec(p, q), toVec(p, r))) < EPS; }
75
76 // while examining all the edges of the polygon one by one
77
78 bool isConvex(const vector<point> &P) {
79     int n = (int)P.size();
80     // a point/sz=2 or a line/sz=3 is not convex
81     if (n <= 3) return false;
82     bool firstTurn = ccw(P[0], P[1], P[2]); // remember one result,
83     for (int i = 1; i < n-1; ++i) // compare with the others
84         if (ccw(P[i], P[i+1], P[i+2]) == n ? 1 : i+2) != firstTurn)
85             return false; // different -> concave
86     return true; // otherwise -> convex
87 }
88
89 // returns 1/0/-1 if point p is inside/on (vertex/edge)/outside of
90 // either convex/concave polygon P
91 int insidePolygon(point pt, const vector<point> &P) {
92     int n = (int)P.size();
93     if (n <= 3) return -1; // avoid point or line
94     bool on_polygon = false;
95     for (int i = 0; i < n-1; ++i) // on vertex/edge?
96         if (fabs(dist(P[i], pt) + dist(pt, P[i+1]) - dist(P[i], P[i+1])) < EPS)
97             on_polygon = true;
98     if (on_polygon) return 0; // pt is on polygon
99     double sum = 0.0; // first = last point
100     for (int i = 0; i < n-1; ++i) {
101         if (ccw(pt, P[i], P[i+1]))
102             sum += angle(P[i], pt, P[i+1]); // left turn/ccw
103         else
104             sum -= angle(P[i], pt, P[i+1]); // right turn/cw
105     }
106     return fabs(sum) > M_PI ? 1 : -1; // 360d->in, 0d->out
107 }
108
109 // compute the intersection point between line segment p-q and line A-B
110 point lineIntersectSeg(point p, point q, point A, point B) {
111     double a = B.y-A.y, b = A.x-B.x, c = B.x*A.y - A.x*B.y;
112     double u = fabs(a*p.x + b*p.y + c);
113     double v = fabs(a*q.x + b*q.y + c);
114     return point((p.x*v + q.x*u) / (u+v), (p.y*v + q.y*u) / (u+v));
115 }
116
117 // cuts polygon Q along the line formed by point A->point B (order matters)
118 // (note: the last point must be the same as the first point)
119 vector<point> cutPolygon(point A, point B, const vector<point> &Q) {
120     vector<point> P;
121     for (int i = 0; i < (int)Q.size(); ++i) {
122         double left1 = cross(toVec(A, B), toVec(A, Q[i])), left2 = 0;
123         if (i != (int)Q.size()-1) left2 = cross(toVec(A, B), toVec(A, Q[i+1]));
124         if (left1 > -EPS) P.push_back(Q[i]); // Q[i] is on the left
125         if (left1*left2 < -EPS) // crosses line AB
126             P.push_back(lineIntersectSeg(Q[i], Q[i+1], A, B));
127     }
128     if (!P.empty() && !(P.back() == P.front()))
129         P.push_back(P.front()); // wrap around
130     return P;
131 }

```

```

130 vector<point> CH_Graham(vector<point> &Pts) { // overall O(n log n)
131     vector<point> P(Pts); // copy all points
132     int n = (int)P.size();
133     if (n <= 3) { // point/line/triangle
134         if (!(P[0] == P[n-1])) P.push_back(P[0]); // corner case
135         return P; // the CH is P itself
136     }
137
138     // first, find P0 = point with lowest Y and if tie: rightmost X
139     int P0 = min_element(P.begin(), P.end())-P.begin();
140     swap(P[0], P[P0]); // swap P[P0] with P[0]
141
142     // second, sort points by angle around P0, O(n log n) for this sort
143     sort(++P.begin(), P.end(), [&](point a, point b) {
144         return ccw(P[0], a, b); // use P[0] as the pivot
145     });
146
147     // third, the ccw tests, although complex, it is just O(n)
148     vector<point> S({P[n-1], P[0], P[1]}); // initial S
149     int i = 2; // then, we check the
150     rest
151     while (i < n) { // n > 3, O(n)
152         int j = (int)S.size()-1;
153         if (ccw(S[j-1], S[j], P[i])) // CCW turn
154             S.push_back(P[i++]); // accept this point
155         else // CW turn
156             S.pop_back(); // pop until a CCW turn
157     }
158     return S; // return the result
159 }
160
161 vector<point> CH_Andrew(vector<point> &Pts) { // overall O(n log n)
162     int n = Pts.size(), k = 0;
163     vector<point> H(2*n);
164     sort(Pts.begin(), Pts.end()); // sort the points by x/y
165     for (int i = 0; i < n; ++i) { // build lower hull
166         while ((k >= 2) && !ccw(H[k-2], H[k-1], Pts[i])) --k;
167         H[k++] = Pts[i];
168     }
169     for (int i = n-2, t = k+1; i >= 0; --i) { // build upper hull
170         while ((k >= t) && !ccw(H[k-2], H[k-1], Pts[i])) --k;
171         H[k++] = Pts[i];
172     }
173     H.resize(k);
174     return H;
175 }
176
177 int main() {
178     // 6(+1) points, entered in counter clockwise order, 0-based indexing
179     vector<point> P;
180     P.emplace_back(1, 1); // P0
181     P.emplace_back(3, 3); // P1
182     P.emplace_back(9, 1); // P2
183     P.emplace_back(12, 4); // P3
184     P.emplace_back(9, 7); // P4
185     P.emplace_back(1, 7); // P5
186     P.push_back(P[0]); // loop back, P6 = P0
187
188     printf("Perimeter = %.2lf\n", perimeter(P)); // 31.64
189     printf("Area = %.2lf\n", area(P)); // 49.00
190     printf("Area = %.2lf\n", area_alternative(P)); // also 49.00
191     printf("Is convex = %d\n", isConvex(P)); // 0 (false)
192
193     point p_out(3, 2); // outside this (concave) polygon
194     printf("P_out is inside = %d\n", insidePolygon(p_out, P)); // -1
195     printf("P1 is inside = %d\n", insidePolygon(P[1], P)); // 0
196     point p_on(5, 7); // on this (concave) polygon
197     printf("P_on is inside = %d\n", insidePolygon(p_on, P)); // 0
198     point p_in(3, 4); // inside this (concave) polygon
199     printf("P_in is inside = %d\n", insidePolygon(p_in, P)); // 1
200
201     P = cutPolygon(P[2], P[4], P);
202     printf("Perimeter = %.2lf\n", perimeter(P)); // smaller now, 29.15
203     printf("Area = %.2lf\n", area(P)); // 40.00
204
205     P = CH_Graham(P); // now this is a
206     rectangle
207     printf("Perimeter = %.2lf\n", perimeter(P)); // precisely 28.00
208     printf("Area = %.2lf\n", area(P)); // precisely 48.00
209     printf("Is convex = %d\n", isConvex(P)); // true
210     printf("P_out is inside = %d\n", insidePolygon(p_out, P)); // 1
211     printf("P_in is inside = %d\n", insidePolygon(p_in, P)); // 1
212     return 0;
213 }

```

7.9 Minkowski Sum

```

1 // Nome: Minkowski Sum
2 // Complexidade: O(n + m)
3
4 struct pt{
5     long long x, y;
6     pt operator + (const pt & p) const {
7         return pt{x + p.x, y + p.y};
8     }
9     pt operator - (const pt & p) const {
10        return pt{x - p.x, y - p.y};
11    }
12    long long cross(const pt & p) const {
13        return x * p.y - y * p.x;
14    }
15 };
16
17 void reorder_polygon(vector<pt> & P){
18     size_t pos = 0;
19     for(size_t i = 1; i < P.size(); i++){
20         if(P[i].y < P[pos].y || (P[i].y == P[pos].y && P[i].x < P[pos].x))
21             pos = i;
22     }
23     rotate(P.begin(), P.begin() + pos, P.end());
24 }
25

```

```

26 vector<pt> minkowski(vector<pt> P, vector<pt> Q){
27     // the first vertex must be the lowest
28     reorder_polygon(P);
29     reorder_polygon(Q);
30     // we must ensure cyclic indexing
31     P.push_back(P[0]);
32     P.push_back(P[i]);
33     Q.push_back(Q[0]);
34     Q.push_back(Q[i]);
35     // main part
36     vector<pt> result;
37     size_t i = 0, j = 0;
38     while(i < P.size() - 2 || j < Q.size() - 2){
39         result.push_back(P[i] + Q[j]);
40         auto cross = (P[i + 1] - P[i]).cross(Q[j + 1] - Q[j]);
41         if(cross >= 0 && i < P.size() - 2)
42             ++i;
43         if(cross <= 0 && j < Q.size() - 2)
44             ++j;
45     }
46     return result;
47 }

```

7.10 Ponto

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int EPS = 1e-9;
4 // Ponto 2D
5 struct point_i {
6     int x, y;
7     point_i() { x = y = 0; }
8     point_i(int _x, int _y) : x(_x), y(_y) {}
9 };
10
11 // Ponto 2D com precisao
12 struct point {
13     double x, y;
14     point() { x = y = 0.0; }
15     point(double _x, double _y) : x(_x), y(_y) {}
16
17     bool operator < (point other) const {
18         if (fabs(x-other.x) > EPS)
19             return x < other.x;
20         return y < other.y;
21     }
22
23     bool operator == (const point &other) const {
24         return (fabs(x-other.x) < EPS) and (fabs(y-other.y) < EPS);
25     }
26 };
27
28 // Distancia entre 2 pontos
29 double dist(const point &p1, const point &p2) {
30     return hypot(p1.x-p2.x, p1.y-p2.y);
31 }
32

```

```

33 double DEG_to_RAD(double d) { return d*M_PI / 180.0; }
34 double RAD_to_DEG(double r) { return r*180.0 / M_PI; }
35
36 // Rotaciona o ponto p em theta graus em sentido anti-horario em relacao a
    origem (0, 0)
37 point rotate(const point &p, double theta) {
38     double rad = DEG_to_RAD(theta);
39     return point(p.x*cos(rad) - p.y*sin(rad),
40                 p.x*sin(rad) + p.y*cos(rad));
41 }

```

7.11 Triangulos

```

1 #include <bits/stdc++.h>
2 #include "vetor.cpp"
3 #include "linha.cpp"
4
5 using namespace std;
6
7 // Condicao Existencia
8 bool existeTriangulo(double a, double b, double c) {
9     return (a+b > c) && (a+c > b) && (b+c > a);
10 }
11
12 // Area de um triangulo de lados a, b e c
13 int area(int a, int b, int c) {
14     if (!existeTriangulo(a, b, c)) return 0;
15     double s = (a+b+c)/2.0;
16     return sqrt(s*(s-a)*(s-b)*(s-c));
17 }
18
19 double perimeter(double ab, double bc, double ca) {
20     return ab + bc + ca;
21 }
22
23 double perimeter(point a, point b, point c) {
24     return dist(a, b) + dist(b, c) + dist(c, a);
25 }
26
27 // ===== CIRCULO INSCRITO =====
28
29 // Retorna raio de um circulo inscrito em um triangulo de lados a, b e c
30 double rInCircle(double ab, double bc, double ca) {
31     return area(ab, bc, ca) / (0.5 * perimeter(ab, bc, ca));
32 }
33
34 double rInCircle(point a, point b, point c) {
35     return rInCircle(dist(a, b), dist(b, c), dist(c, a));
36 }
37
38 // Calcula o centro e o raio do circulo inscrito em um triangulo dados
    seus pontos
39 bool inCircle(point p1, point p2, point p3, point &ctr, double &r) {
40     r = rInCircle(p1, p2, p3);
41     if (fabs(r) < EPS) return false;
42     line l1, l2;
43     double ratio = dist(p1, p2) / dist(p1, p3);
44     point p = translate(p2, scale(toVec(p2, p3), ratio / (1+ratio)));

```

```

44 pointsToLine(p1, p, l1);
45 ratio = dist(p2, p1) / dist(p2, p3);
46 p = translate(p1, scale(toVec(p1, p3), ratio / (1+ratio)));
47 pointsToLine(p2, p, l2);
48 areIntersect(l1, l2, ctr);
49 return true;
50 }
51
52 // ===== CIRCULO CIRCUNSCRITO =====
53
54 double rCircumCircle(double ab, double bc, double ca) {
55     return ab * bc * ca / (4.0 * area(ab, bc, ca));
56 }
57
58 double rCircumCircle(point a, point b, point c) {
59     return rCircumCircle(dist(a, b), dist(b, c), dist(c, a));
60 }

```

7.12 Vetor

```

1 #include <bits/stdc++.h>
2 #include "ponto.cpp"
3 using namespace std;
4
5 struct vec {
6     double x, y;
7     vec(double _x, double _y) : x(_x), y(_y) {}
8 };
9
10 double dot(vec a, vec b) { return (a.x*b.x + a.y*b.y); }
11 double norm_sq(vec v) { return v.x*v.x + v.y*v.y; }
12 double cross(vec a, vec b) { return a.x*b.y - a.y*b.x; }
13
14 // Converte 2 pontos em um vetor
15 vec toVec(const point &a, const point &b) {
16     return vec(b.x-a.x, b.y-a.y);
17 }
18
19 // Soma 2 vetores
20 vec scale(const vec &v, double s) {
21     return vec(v.x*s, v.y*s);
22 }
23
24 // Resultado do ponto p + vetor v
25 point translate(const point &p, const vec &v) {
26     return point(p.x+v.x, p.y+v.y);
27 }
28
29 // Angulo entre 2 vetores (produto escalar) em radianos
30 double angle(const point &a, const point &o, const point &b) {
31     vec oa = toVec(o, a), ob = toVec(o, b);
32     return acos(dot(oa, ob) / sqrt(norm_sq(oa) * norm_sq(ob)));
33 }
34
35 // Retorna se o ponto r esta a esquerda da linha pq (counter-clockwise)
36 bool ccw(point p, point q, point r) {
37     return cross(toVec(p, q), toVec(p, r)) > EPS;
38 }

```

```

39 // Retorna se sao colineares
40 bool collinear(point p, point q, point r) {
41     return fabs(cross(toVec(p, q), toVec(p, r))) < EPS;
42 }
43
44 // Distancia ponto-linha
45 double distToLine(point p, point a, point b, point &c) {
46     vec ap = toVec(a, p), ab = toVec(a, b);
47     double u = dot(ap, ab) / norm_sq(ab);
48     c = translate(a, scale(ab, u));
49     return dist(p, c);
50 }
51
52 // Distancia ponto p - segmento ab
53 double distToLineSegment(point p, point a, point b, point &c) {
54     vec ap = toVec(a, p), ab = toVec(a, b);
55     double u = dot(ap, ab) / norm_sq(ab);
56     if (u < 0.0) { // closer to a
57         c = point(a.x, a.y);
58         return dist(p, a); // dist p to a
59     }
60     if (u > 1.0) { // closer to b
61         c = point(b.x, b.y);
62         return dist(p, b); // dist p to b
63     }
64     return distToLine(p, a, b, c); // use distToLine
65 }

```

8 Grafos

8.1 Bfs - Matriz

```

1 // Description: BFS para uma matriz (n x m)
2 // Complexidade: O(n * m)
3
4 vector<vi> mat;
5 vector<vector<bool>> vis;
6 vector<pair<int,int>> mov = {{0, 1}, {0, -1}, {1, 0}, {-1, 0}};
7 int l, c;
8
9 bool valid(int x, int y) {
10     return (0 <= x and x < l and 0 <= y and y < c and !vis[x][y] /*and mat[x][y]*/);
11 }
12
13 void bfs(int i, int j) {
14
15     queue<pair<int,int>> q; q.push({i, j});
16
17     while(!q.empty()) {
18
19         auto [u, v] = q.front(); q.pop();
20         vis[u][v] = true;
21
22         for(auto [x, y]: mov) {
23             if(valid(u+x, v+y)) {

```

```

24         q.push({u+x,v+y});
25         vis[u+x][v+y] = true;
26     }
27 }
28 }
29 }
30
31 void solve() {
32     cin >> l >> c;
33     mat.resize(l, vi(c));
34     vis.resize(l, vector<bool>(c, false));
35     /*preenche matriz*/
36     bfs(0,0);
37 }

```

8.2 Bfs - Por Niveis

```

1 // Description: Encontrar distancia entre S e outros pontos em que pontos
  // estao agrupados (terminais)
2 // EXTRA: BFS diferenciado para armazenar distancias sem VIS
3
4 int n;
5 vi dist;
6 vector<vi> niveisDoNode, itensDoNivel;
7
8 void bfs(int s) {
9
10     queue<pair<int, int>> q; q.push({s, 0});
11
12     while (!q.empty()) {
13         auto [v, dis] = q.front(); q.pop();
14
15         for(auto nivel : niveisDoNode[v]) {
16             for(auto u : itensDoNivel[nivel]) {
17                 if (dist[u] == 0) {
18                     q.push({u, dis+1});
19                     dist[u] = dis + 1;
20                 }
21             }
22         }
23     }
24 }
25
26 void solve() {
27
28     int n, ed; cin >> n >> ed;
29     dist.clear(), itensDoNivel.clear(), niveisDoNode.clear();
30     itensDoNivel.resize(n);
31
32     f(i,0,ed) {
33         int q; cin >> q;
34         while(q--) {
35             int v; cin >> v;
36             niveisDoNode[v].push_back(i);
37             itensDoNivel[i].push_back(v);
38         }
39     }

```

```

40
41     bfs(0);
42 }

```

8.3 Bfs - String

```

1 // Description: BFS para listas de adjacencia por nivel
2 // Complexidade: O(V + E)
3
4 int n;
5 unordered_map<string, int> dist;
6 unordered_map<string, vector<int>> niveisDoNode;
7 vector<vector<string>> itensDoNivel;
8
9 void bfs(string s) {
10
11     queue<pair<string, int>> q; q.push({s, 0});
12
13     while (!q.empty()) {
14         auto [v, dis] = q.front(); q.pop();
15
16         for(auto linha : niveisDoNode[v]) {
17             for(auto u : itensDoNivel[linha]) {
18                 if (dist[u] == 0) {
19                     q.push({u, dis+1});
20                     dist[u] = dis + 1;
21                 }
22             }
23         }
24     }
25 }
26
27 void solve() {
28
29     int n, ed; cin >> n >> ed;
30     dist.clear(), itensDoNivel.clear(), niveisDoNode.clear();
31     itensDoNivel.resize(n);
32
33     f(i,0,ed) {
34         int q; cin >> q;
35         while(q--) {
36             string str; cin >> str;
37             niveisDoNode[str].push_back(i);
38             itensDoNivel[i].push_back(str);
39         }
40     }
41
42     string src; cin >> src;
43     bfs(src);
44 }

```

8.4 Bfs - Tradicional

```

1 // BFS com informacoes adicionais sobre a distancia e o pai de cada
  // vertice
2 // Complexidade: O(V + E), onde V eh o numero de vertices e E o numero de
  // areqas

```



```

3
4 int n;
5 vector<bool> vis;
6 vector<int> d, p;
7 vector<vector<int>> adj;
8
9 void bfs(int s) {
10
11     queue<int> q; q.push(s);
12     vis[s] = true, d[s] = 0, p[s] = -1;
13
14     while (!q.empty()) {
15         int v = q.front(); q.pop();
16         vis[v] = true;
17
18         for (int u : adj[v]) {
19             if (!vis[u]) {
20                 vis[u] = true;
21                 q.push(u);
22                 // d[u] = d[v] + 1;
23                 // p[u] = v;
24             }
25         }
26     }
27 }
28
29 void solve() {
30     cin >> n;
31     adj.resize(n); d.resize(n, -1);
32     vis.resize(n); p.resize(n, -1);
33
34     for (int i = 0; i < n; i++) {
35         int u, v; cin >> u >> v;
36         adj[u].push_back(v);
37         adj[v].push_back(u);
38     }
39
40     bfs(0);
41 }

```

42
43 // OBS: Pode ser usado para encontrar o menor caminho entre dois vertices
em um grafo sem pesos

8.5 Dfs

```

1 vector<int> adj[MAXN], parent;
2 int visited[MAXN];
3
4 // DFS com informacoes adicionais sobre o pai de cada vertice
5 // Complexidade: O(V + E), onde V eh o numero de vertices e E o numero de
   areqas
6 void dfs(int p) {
7     memset(visited, 0, sizeof visited);
8     stack<int> st;
9     st.push(p);
10
11     while (!st.empty()) {

```

```

12         int v = st.top(); st.pop();
13
14         if (visited[v]) continue;
15         visited[v] = true;
16
17         for (int u : adj[v]) {
18             if (!visited[u]) {
19                 parent[u] = v;
20                 st.push(u);
21             }
22         }
23     }
24 }
25
26 // DFS com informacoes adicionais sobre o pai de cada vertice
27 // Complexidade: O(V + E), onde V eh o numero de vertices e E o numero de
   areqas
28 void dfs(int v) {
29     visited[v] = true;
30     for (int u : adj[v]) {
31         if (!visited[u]) {
32             parent[u] = v;
33             dfs(u);
34         }
35     }
36 }
37
38 void solve() {
39     int n; cin >> n;
40     for (int i = 0; i < n; i++) {
41         int u, v; cin >> u >> v;
42         adj[u].push_back(v);
43         adj[v].push_back(u);
44     }
45     dfs(0);
46 }

```

8.6 Articulation

```

1 // Description: encontra os pontos de çãarticulao de um grafo
2 // Complexidade: O(V+E)
3
4 const int MAX = 410;
5
6 vector <int> adj[MAX];
7
8 void APUtil(int u, bool visited[], int disc[], int low[], int& time, int
   parent, bool isAP[]) {
9     int children = 0;
10
11     visited[u] = true;
12
13     disc[u] = low[u] = ++time;
14
15     for (auto v : adj[u]) {
16         if (!visited[v]) {
17             children++;

```

```

18     APUtil(v, visited, disc, low, time, u, isAP);
19
20     low[u] = min(low[u], low[v]);
21
22     if (parent != -1 && low[v] >= disc[u])
23         isAP[u] = true;
24 }
25
26 else if (v != parent)
27     low[u] = min(low[u], disc[v]);
28 }
29
30 if (parent == -1 && children > 1)
31     isAP[u] = true;
32 }
33
34 void AP(int V) {
35     int disc[V] = { 0 };
36     int low[V];
37     bool visited[V] = { false };
38     bool isAP[V] = { false };
39     int time = 0, par = -1;
40
41     for (int u = 0; u < V; u++)
42         if (!visited[u])
43             APUtil(u, visited, disc, low, time, par, isAP);
44
45     bool printed = false;
46
47     for (int u = 0; u < V; u++) {
48         if (isAP[u] == true) {
49             cout << u+1 << " ";
50             printed = true;
51         }
52     }
53
54     if (!printed) cout << "nenhum" << endl;
55     else cout << endl;
56 }
57
58 void solve() {
59
60     int n, ed; cin >> n >> ed;
61
62     for(int i = 0; i < n; i++)
63         adj[i].clear();
64
65     while(ed--) {
66         int a, b; cin >> a >> b; a--, b--;
67         adj[a].push_back(b);
68         adj[b].push_back(a);
69     }
70
71     AP(n);
72 }

```

8.7 Bipartido

```

1 // Description: Determina se um grafo eh bipartido ou nao
2 // Complexidade: O(V+E)
3
4 vector<vi> AL;
5
6 bool bipartido(int n) {
7
8     int s = 0;
9     queue<int> q; q.push(s);
10
11     vi color(n, INF); color[s] = 0;
12     bool ans = true;
13     while (!q.empty() && ans) {
14         int u = q.front(); q.pop();
15
16         for (auto &v : AL[u]) {
17             if (color[v] == INF) {
18                 color[v] = 1 - color[u];
19                 q.push(v);
20             }
21             else if (color[v] == color[u]) {
22                 ans = false;
23                 break;
24             }
25         }
26     }
27
28     return ans;
29 }
30
31 void solve() {
32
33     int n, edg; cin >> n >> edg;
34     AL.resize(n, vi());
35
36     while(edg--) {
37         int a, b; cin >> a >> b;
38         AL[a].push_back(b);
39         AL[b].push_back(a);
40     }
41
42     cout << bipartido(n) << endl;
43 }

```

8.8 Caminho Minimo - @Tabela

1	Critério	BFS (V + E)	Dijkstra (E*log V)	Bellman-Ford (V*E)	Floyd-Warshall (V ³)
2		-----+	-----+	-----+	-----+
3	Max Size	V + E <= 100M	V + E <= 1M	V * E <= 100M	V <= 450
4	Sem-Peso no geral	CRIA	Ok	Ruim	Ruim

5	Peso		WA			Melhor			Ok			Ruim
	no geral											
6	Peso Neg		WA			Modificado	Ok			Ok		
	no geral											
7	Neg-Cic		Nao Detecta			Nao Detecta			Detecta			
	Detecta											
8	Grafo Pequeno		WA se peso			Overkill			Overkill			
	Melhor											

8.9 Caminho Minimo - Bellman Ford

```

1 // Description: Encontra menor caminho em grafos com pesos negativos
2 /* Complexidade:
3     Conexo: O(VE)
4     Desconexo: O(EV^2)
5 */
6 // Classe: Single Source Shortest Path (SSSP)
7
8 vector<tuple<int,int,int>> edg; // edge: u, v, w
9 vi dist;
10
11 int bellman_ford(int n, int src) {
12     dist.assign(n+1, INT_MAX);
13
14     f(i,0,n+2) {
15         for(auto& [u, v, w] : edg) {
16             if(dist[u] != INT_MAX and dist[v] > w + dist[u])
17                 dist[v] = dist[u] + w;
18         }
19     }
20
21     // Possivel checar ciclos negativos (ciclo de peso total negativo)
22     for(auto& [u, v, w] : edg) {
23         if(dist[u] != INT_MAX and dist[v] > w + dist[u])
24             return 1;
25     }
26
27     return 0;
28 }
29
30 int main() {
31
32     int n, edges; cin >> n >> edges;
33     f(i,0,edges) {
34         int u, v, w; cin >> u >> v >> w;
35         edg.push_back({u, v, w});
36     }
37     bellman_ford(n, 1);
38 }

```

8.10 Caminho Minimo - Checar I J (In)Diretamente Conectados

```

1 // Description: Verifica se o vertice i esta diretamente conectado ao
  vertice j
2 // Complexity: O(n^3)

```

```

3
4 const int INF = 1e9;
5 const int MAX_V = 450;
6 int adj[MAX_V][MAX_V];
7
8 void transitive_closure(int n) {
9
10     for (int k = 0; k < n; ++k)
11         for (int i = 0; i < n; ++i)
12             for (int j = 0; j < n; ++j)
13                 adj[i][j] |= (adj[i][k] & adj[k][j]);
14 }
15
16 void solve() {
17
18     int n, ed; cin >> n >> ed;
19     f(u,0,n) {
20         f(v,0,n) {
21             adj[u][v] = INF;
22         }
23         adj[u][u] = 0;
24     }
25
26     f(i,0,ed) {
27         int u, v, w; cin >> u >> v >> w;
28         adj[u][v] = w;
29     }
30
31     transitive_closure(n);
32
33     int i = 0, j = 0; cin >> i >> j;
34     cout << (adj[i][j] == INF ? "Nao" : "Sim") << endl;
35 }

```

8.11 Caminho Minimo - Diametro Do Grafo

```

1 // Description: Encontra o diametro de um grafo
2 // => maximum shortest path between any two vertices
3 // Complexidade: O(n^3)
4
5 int adj[MAX_V][MAX_V];
6
7 int diameter(int n) {
8     int ans = 0;
9     f(u,0,n) {
10         f(v,0,n) {
11             if (adj[u][v] != INF) {
12                 ans = max(ans, adj[u][v]);
13             }
14         }
15     }
16     return ans;
17 }
18
19 void floyd_warshall(int n) {
20
21     for (int k = 0; k < n; ++k)

```

```

22     for (int u = 0; u < n; ++u)
23     for (int v = 0; v < n; ++v)
24         adj[u][v] = min(adj[u][v], adj[u][k]+adj[k][v]);
25 }
26
27 void solve() {
28
29     int n, ed; cin >> n >> ed;
30     f(u,0,n) {
31         f(v,0,n) {
32             adj[u][v] = INF;
33         }
34         adj[u][u] = 0;
35     }
36
37     f(i,0,ed) {
38         int u, v, w; cin >> u >> v >> w;
39         adj[u][v] = w;
40     }
41
42     floyd_warshall(n);
43     cout << diameter(n) << endl;
44 }

```

8.12 Caminho Minimo - Dijkstra

```

1 // Description: Algoritmo de Dijkstra para caminho mínimo em grafos.
2 // Complexity:  $O(E \log V)$ 
3 // Classe: Single Source Shortest Path (SSSP)
4
5 vi dist;
6 vector<vector<pii>> adj;
7
8 void dijkstra(int s) {
9
10     dist[s] = 0; // se eventualmente puder voltar pra ca, tipo ciclo |
11     salesman | remover essa linha
12
13     priority_queue<pii, vector<pii>, greater<pii>> pq; pq.push({0, s});
14
15     while (!pq.empty()) {
16         auto [d, u] = pq.top(); pq.pop();
17
18         if (d > dist[u]) continue;
19
20         // if(u == s and dist[u] < INF) break; | pra quando tiver que
21         // fazer um ciclo
22
23         for (auto &[v, w] : adj[u]) {
24             if (dist[u] + w >= dist[v]) continue;
25             dist[v] = dist[u]+w;
26             pq.push({dist[v], v});
27         }
28     }
29
30 void solve() {

```

```

30
31     int n, ed; cin >> n >> ed;
32     adj.assign(n, vector<pii>());
33     dist.assign(n, INF); // INF = 1e9
34
35     while (ed-->0) {
36         int u, v, w; cin >> u >> v >> w;
37         adj[u].emplace_back(v, w);
38     }
39
40     int s; cin >> s;
41     dijkstra(s);
42 }

```

8.13 Caminho Minimo - Floyd Warshall

```

1 // Description: Caminho minimo entre todos os pares de vertices em um
2 // grafo
3 // Complexity:  $O(n^3)$ 
4 // Classe: All Pairs Shortest Path (APSP)
5
6 const int INF = 1e9;
7 const int MAX_V = 450;
8 int adj[MAX_V][MAX_V];
9
10 void printAnswer(int n) {
11     for (int u = 0; u < n; ++u)
12     for (int v = 0; v < n; ++v)
13         cout << "APSP("<u<<u<<v<<v<<") = " << adj[u][v] << endl;
14 }
15
16 void prepareParent() {
17     f(i,0,n) {
18         f(j,0,n) {
19             p[i][j] = i;
20         }
21     }
22
23     for (int k = 0; k < n; ++k)
24     for (int i = 0; i < n; ++i)
25     for (int j = 0; j < n; ++j)
26         if (adj[i][k] + adj[k][j] < adj[i][j]) {
27             adj[i][j] = adj[i][k]+adj[k][j];
28             p[i][j] = p[k][j];
29         }
30 }
31
32 vi restorePath(int u, int v) {
33
34     if (adj[u][v] == INF) return {};
35     vi path;
36     for (; v != u; v = p[u][v]) {
37         if (v == -1) return {};
38         path.push_back(v);
39     }
40     path.push_back(u);
41     reverse(path.begin(), path.end());

```

```

41     return path;
42 }
43
44 void floyd_warshall(int n) {
45     for (int k = 0; k < n; ++k)
46         for (int u = 0; u < n; ++u)
47             for (int v = 0; v < n; ++v)
48                 adj[u][v] = min(adj[u][v], adj[u][k]+adj[k][v]);
49 }
50
51 void solve() {
52
53     int n, ed; cin >> n >> ed;
54     f(u,0,n) {
55         f(v,0,n) {
56             adj[u][v] = INF;
57         }
58         adj[u][u] = 0;
59     }
60
61     f(i,0,ed) {
62         int u, v, w; cin >> u >> v >> w;
63         adj[u][v] = w;
64     }
65
66     floyd_warshall(n);
67
68     // prepareParent();
69     // vi path = restorePath(0, 3);
70 }

```

8.14 Caminho Mínimo - Minimax

```

1 // Description: MiniMax problem: encontrar o menor caminho mais longo
  // entre todos os pares de vertices em um grafo
2 // Complexity: O(n^3)
3
4 const int INF = 1e9;
5 const int MAX_V = 450;
6 int adj[MAX_V][MAX_V];
7
8 void miniMax(int n) {
9     for (int k = 0; k < V; ++k)
10         for (int i = 0; i < V; ++i) // reverse min and max
11             for (int j = 0; j < V; ++j) // for MaxiMin problem
12                 AM[i][j] = min(AM[i][j], max(AM[i][k], AM[k][j]));
13 }
14
15 void solve() {
16
17     int n, ed; cin >> n >> ed;
18     f(u,0,n) {
19         f(v,0,n) {
20             adj[u][v] = INF;
21         }
22         adj[u][u] = 0;

```

```

23     }
24
25     f(i,0,ed) {
26         int u, v, w; cin >> u >> v >> w;
27         adj[u][v] = w;
28     }
29
30     transitive_closure(n);
31
32     int i = 0, j = 0; cin >> i >> j;
33     cout << (adj[i][j] == INF ? "Nao" : "Sim") << endl;
34 }

```

8.15 Cycle Check

```

1 // Description: Checa se um grafo direcionado possui ciclos e imprime os
  // tipos de arestas.
2 // Complexidade: O(V + E)
3
4 vector<vector<pii>> adj;
5 vi dfs_num, dfs_parent;
6
7 void cycleCheck(int u) {
8     dfs_num[u] = -2;
9     for (auto &[v, w] : adj[u]) {
10         if (dfs_num[v] == -1) {
11             dfs_parent[v] = u;
12             cycleCheck(v);
13         }
14         else if (dfs_num[v] == -2) {
15             if (v == dfs_parent[u])
16                 cout << " Bidirectional Edge (" << u << ", " << v << ")-("
17                 << v << ", " << u << ")\n";
18             else
19                 cout << "Back Edge (" << u << ", " << v << ") (Cycle)\n";
20         }
21         else if (dfs_num[v] == -3)
22             cout << " Forward/Cross Edge (" << u << ", " << v << ")\n";
23     }
24     dfs_num[u] = -3;
25
26 void solve() {
27     int n, ed; cin >> n >> ed;
28     adj.assign(ed, vector<pii>());
29
30     for (int i = 0; i < ed; ++i) {
31         int u, v, w; cin >> u >> v >> w;
32         adj[u].emplace_back(v, w);
33     }
34
35     cout << "Graph Edges Property Check\n";
36     dfs_num.assign(ed, -1);
37     dfs_parent.assign(ed, -1);
38     for (int u = 0; u < n; ++u)
39         if (dfs_num[u] == -1)
40             cycleCheck(u);

```

41 }

8.16 Encontrar Ciclo

```
1 // Description: Encontrar ciclo em grafo nao direcionado
2 // Complexidade: O(n + m)
3
4 int n;
5 vector<vector<int>> adj;
6 vector<bool> vis;
7 vector<int> p;
8 int cycle_start, cycle_end;
9
10 bool dfs(int v, int par) {
11     vis[v] = true;
12     for (int u : adj[v]) {
13         if(u == par) continue;
14         if(vis[u]) {
15             cycle_end = v;
16             cycle_start = u;
17             return true;
18         }
19         p[u] = v;
20         if(dfs(u, p[u]))
21             return true;
22     }
23     return false;
24 }
25
26 vector<int> find_cycle() {
27     cycle_start = -1;
28
29     for (int v = 0; v < n; v++)
30         if (!vis[v] and dfs(v, p[v]))
31             break;
32
33     if (cycle_start == -1) return {};
34
35     vector<int> cycle;
36     cycle.push_back(cycle_start);
37     for (int v = cycle_end; v != cycle_start; v = p[v])
38         cycle.push_back(v);
39     cycle.push_back(cycle_start);
40     return cycle;
41 }
42
43 void solve() {
44     int edg; cin >> n >> edg;
45     adj.assign(n, vector<int>());
46     vis.assign(n, false), p.assign(n, -1);
47     while(edg--) {
48         int a, b; cin >> a >> b;
49         adj[a].push_back(b);
50         adj[b].push_back(a);
51     }
52     vector<int> ans = find_cycle();
53 }
```

8.17 Euler Tree

```
1 // Descricao: Encontra a euler tree de um grafo
2 // Complexidade: O(n)
3 vector<vector<int>> adj(MAX);
4 vector<int> vis(MAX, 0);
5 vector<int> euTree(MAX);
6
7 void eulerTree(int u, int &index) {
8     vis[u] = 1;
9     euTree[index++] = u;
10    for (auto it : adj[u]) {
11        if (!vis[it]) {
12            eulerTree(it, index);
13            euTree[index++] = u;
14        }
15    }
16 }
17
18 void solve() {
19
20     f(i,0,n-1) {
21         int a, b; cin >> a >> b;
22         adj[a].push_back(b);
23         adj[b].push_back(a);
24     }
25
26     int index = 0; eulerTree(1, index);
27 }
```

8.18 Kosaraju

```
1 // Description: Encontra o numero de componentes fortemente conexas em um
2 // grafo direcionado
3 // Complexidade: O(V + E)
4 int dfsNumberCounter, numSCC;
5 vector<vii> adj, adj_t;
6 vi dfs_num, dfs_low, S, visited;
7 stack<int> St;
8
9 void kosarajuUtil(int u, int pass) {
10     dfs_num[u] = 1;
11     vii &neighbor = (pass == 1) ? adj[u] : adj_t[u];
12     for (auto &[v, w] : neighbor)
13         if (dfs_num[v] == -1)
14             kosarajuUtil(v, pass);
15     S.push_back(u);
16 }
17
18 bool kosaraju(int n) {
19
20     S.clear();
21     dfs_num.assign(n, -1);
22
23     f(u,0,n) {
```

```

24     if (dfs_num[u] == -1)
25         kosarajuUtil(u, 1);
26 }
27
28 int numSCC = 0;
29 dfs_num.assign(n, -1);
30 f(i,n-1,-1) {
31     if (dfs_num[S[i]] == -1)
32         numSCC++, kosarajuUtil(S[i], 2);
33 }
34
35 return numSCC == 1;
36 }
37
38 void solve() {
39
40     int n, ed; cin >> n >> ed;
41     adj.assign(n, vii());
42     adj_t.assign(n, vii());
43
44     while (ed--) {
45         int u, v, w; cin >> u >> v >> w;
46         AL[u].emplace_back(v, 1);
47         adj_t[v].emplace_back(u, 1);
48     }
49
50     // Printa se o grafo eh fortemente conexo
51     cout << kosaraju(n) << endl;
52
53     // Printa o numero de componentes fortemente conexas
54     cout << numSCC << endl;
55
56     // Printa os vertices de cada componente fortemente conexa
57     f(i,0,n){
58         if (dfs_num[i] == -1) cout << i << ": " << "Nao visitado" << endl;
59         else cout << i << ": " << dfs_num[i] << endl;
60     }
61 }

```

8.19 Kruskal

```

1 // DDescricao: Encontra a arvore geradora minima de um grafo
2 // Complexidade: O(E log V)
3
4 vector<int> id, sz;
5
6 int find(int a){ // O(a(N)) amortizado
7     return id[a] = (id[a] == a ? a : find(id[a]));
8 }
9
10 void uni(int a, int b) { // O(a(N)) amortizado
11     a = find(a), b = find(b);
12     if(a == b) return;
13
14     if(sz[a] > sz[b]) swap(a,b);
15     id[a] = b, sz[b] += sz[a];
16 }

```

```

17 pair<int, vector<tuple<int, int, int>>> kruskal(vector<tuple<int, int, int>
18 >>& edg) {
19
20     sort(edg.begin(), edg.end()); // Minimum Spanning Tree
21
22     int cost = 0;
23     vector<tuple<int, int, int>> mst; // opcional
24     for (auto [w,x,y] : edg) if (find(x) != find(y)) {
25         mst.emplace_back(w, x, y); // opcional
26         cost += w;
27         uni(x,y);
28     }
29     return {cost, mst};
30 }
31
32 void solve() {
33
34     int n, ed;
35
36     id.resize(n); iota(all(id), 0);
37     sz.resize(n, -1);
38     vector<tuple<int, int, int>> edg;
39
40     f(i,0,ed) {
41         int a, b, w; cin >> a >> b >> w;
42         edg.push_back({w, a, b});
43     }
44
45     auto [cost, mst] = kruskal(edg);
46 }
47
48 // VARIANTES
49
50 // Maximum Spanning Tree: sort(edg.rbegin(), edg.rend());
51
52 /* 'Minimum' Spanning Subgraph:
53 - Algumas arestas ja foram adicionadas (maior prioridade - Questao das
   rodovias)
54 - Arestas que nao foram adicionadas (menor prioridade - ferrovias)
55 -> kruskal(rodovias); kruskal(ferrovias);
56 */
57
58 /* Minimum Spanning Forest:
59 - Queremos uma floresta com k componentes
60 -> kruskal(edg); if(mst.sizer() == k) break;
61 */
62
63 /* MiniMax
64 - Encontrar menor caminho entre dois vertices com maior quantidade de
   arestas
65 -> kruskal(edg); dijkstra(mst);
66 */
67
68 /* Second Best MST
69 - Encontrar a segunda melhor arvore geradora minima
70 -> kruskal(edg);

```

```

71     -> flag mst[i] = 1;
72     -> sort(cmp(edg.flag != -1)) => da prioridade para outras arestas
73 */

```

8.20 Labirinto

```

1 // Verifica se eh possivel sair de um labirinto
2 // Complexidade:  $O(4^{(n*m)})$ 
3
4 vector<pair<int,int>> mov = {{1,0}, {0,1}, {-1,0}, {0,-1}};
5 vector<vector<int>> labirinto, sol;
6 vector<vector<bool>> visited;
7 int L, C;
8
9 bool valid(const int& x, const int& y) {
10     return x >= 0 and x < L and y >= 0 and y < C and labirinto[x][y] != 0
11     and !visited[x][y];
12 }
13
14 bool condicaoSaida(const int& x, const int& y) {
15     return labirinto[x][y] == 2;
16 }
17
18 bool search(const int& x, const int& y) {
19     if(!valid(x, y))
20         return false;
21
22     if(condicaoSaida(x,y)) {
23         sol[x][y] = 2;
24         return true;
25     }
26
27     sol[x][y] = 1;
28     visited[x][y] = true;
29
30     for(auto [dx, dy] : mov)
31         if(search(x+dx, y+dy))
32             return true;
33
34     sol[x][y] = 0;
35     return false;
36 }
37
38 int main() {
39
40     labirinto = {
41         {1, 0, 0, 0},
42         {1, 1, 0, 0},
43         {0, 1, 0, 0},
44         {1, 1, 1, 2}
45     };
46
47     L = labirinto.size(), C = labirinto[0].size();
48     sol.resize(L, vector<int>(C, 0));
49     visited.resize(L, vector<bool>(C, false));
50

```

```

51     cout << search(0, 0) << endl;
52 }

```

8.21 Pontos Articulacao

```

1 // Description: Encontra os pontos de articulo de um grafo ão
2 // direcionado
3 // Complexidade:  $O(V*(V+E))$ 
4
5 int V;
6 vector<vi> adj;
7 vi ans;
8
9 void dfs(vector<bool>& vis, int i, int curr) {
10     vis[curr] = 1;
11     for (auto x : adj[curr]) {
12         if (x != i) {
13             if (!vis[x]) {
14                 dfs(vis, i, x);
15             }
16         }
17     }
18 }
19
20 void AP() {
21     f(i,1,V+1) {
22         int components = 0;
23         vector<bool> vis(V + 1, 0);
24         f(j,1, V+1) {
25             if (j != i) {
26                 if (!vis[j]) {
27                     components++;
28                     dfs(vis, i, j);
29                 }
30             }
31         }
32         if (components > 1) {
33             ans.push_back(i);
34         }
35     }
36 }
37
38 void solve() {
39
40     V = n;
41     adj.clear(), ans.clear();
42     adj.resize(V+1);
43
44     while(edg--) {
45         int a, b; cin >> a >> b;
46         adj[a].push_back(b);
47         adj[b].push_back(a);
48     }
49
50     AP();
51

```



```

52 // Vertices articulacao: ans
53 }

```

8.22 Prufer Code To Tree

```

1 bool vis [MAX];
2 vector<int> adj [MAX];
3 int freq [MAX];
4
5 void dfs (int a) {
6     vis[a] = true;
7     cout << "(" << a;
8     for (const auto& p : adj[a]) {
9         if (!vis[p]) {
10             cout << " ";
11             dfs(p);
12         }
13     }
14
15     cout << ")";
16 }
17
18 // Description: Dado um código de Prufer, construir a árvore
19 // correspondente, preenchendo a lista de adjacência
20 // Complexidade:  $O(V^2)$ 
21 void pruferCodeToTree(queue<int>& q, int V) {
22     f(j,1,V) {
23         f(i,1,V+1) {
24             if (freq[i] == 0) {
25
26                 int front = q.front(); q.pop();
27
28                 freq[i] = -1; // mark as visited
29                 freq[front]--; // decrease the frequency of the front
30
31                 element
32
33                 adj[front].push_back(i);
34                 adj[i].push_back(front);
35
36                 break;
37             }
38         }
39     }
40
41 void solve(string s) {
42     int testNum = s[0];
43
44     if(!('0' <= testNum and testNum <= '9')) {
45         cout << "(1)" << endl;
46         return;
47     }
48
49     memset(freq, 0, sizeof(freq));
50     memset(vis, 0, sizeof(vis));

```

```

51     for (int i = 0; i < MAX; i++) adj[i].clear(); //
52
53     stringstream ss(s);
54     int v;
55
56     queue<int> q;
57     while (ss >> v) {
58         freq[v]++;
59         q.push(v);
60     }
61
62     int V = q.back(); // quantidade de vertices
63
64     pruferCodeToTree(q, V);
65
66     dfs(V);
67
68     cout << endl;
69 }

```

8.23 Successor Graph

```

1 // Encontra sucessor de um vertice dentro de um grafo direcionado
2 // Pre calcular:  $O(n \log n)$ 
3 // Consulta:  $O(\log n)$ 
4
5 vector<vector<int>> adj;
6
7 int succ(int x, int u) {
8     if(k == 1) return adj[x][0];
9     return succ(succ(x, k/2), k/2);
10 }

```

8.24 Topological Sort

```

1 // Description: Retorna ordenacao topologica de adj, e vazio se nao for
2 // DAG
3 // Complexidade:  $O(V+E)$ 
4 // Explicacao: usado para ordenar vertices de um DAG de forma que para cada
5 // aresta direcionada uv, o vértice u aparece antes do vértice v na
6 // ordenacao
7
8 #define MAXN 50010
9
10 int grauEntrada[MAXN];
11 vi adj[MAXN];
12
13 vi topologicalSort(int n) {
14     priority_queue<int, vi, greater<int>> pq;
15
16     f(i,0,n) {
17         if(!grauEntrada[i])
18             pq.push(i);
19     }
20 }

```

```

19 vi ans;
20
21 while (!pq.empty()) {
22     int node = pq.top(); pq.pop();
23
24     for(auto x : adj[node]) {
25         grauEntrada[x]--;
26         if (!grauEntrada[x])
27             pq.push(x);
28     }
29
30     ans.push_back(node);
31 }
32
33 return ans.size() == n ? ans : vi();
34 }
35
36 void solve() {
37
38     int n, ed; cin >> n >> ed;
39
40     memset(grauEntrada, 0, sizeof grauEntrada);
41
42     while(ed--) {
43         int a, b; cin >> a >> b;
44         grauEntrada[b]++;
45         adj[a].push_back(b);
46     }
47
48     vi ans = topologicalSort(n);
49 }

```

9 Grafos Especiais

9.1 Arvore - @Info

```

1 Arvore (NDAG):
2
3 * Definicao
4     - é contm V vertices e V-1 arestas (E = V-1)
5     - todo algoritmo O(V+E) numa arvore eh O(V)
6     - nao direcionado
7     - sem ciclo
8     - conexa
9     - um unico caminho para todo par de vertices
10
11 * Aplicacoes
12
13 -> TREE TRAVERSAL
14     pre-order(v):          in-order(v):          post-order(v):
15         visit(v)           in-order(left(v))      post-order(
16 left(v))
17         pre-order(left(v)) visit(v)             post-order(
18 right(v))
19         pre-order(right(v)) in-order(right(v)) visit(v)

```

```

19 -> Pontos de Articulacao / Pontes
20     - todo vertice eh ponto de articulacao
21
22 -> Single Source Shortest Path (SSSP)
23     - O(V) para achar o caminho minimo de um vertice para todos os
    outros
24     - BFS ou DFS funcionam, mesmo com pesos
25
26 -> All Pairs Shortest Path (APSP)
27     - O(V^2) para achar o caminho minimo de todos para todos
28     - V * SSSP
29
30 -> Diametro
31     - greatest 'shortest path length' between any pair of vertices
32     - 2 * SSSP:
33         1. BFS/DFS de qualquer vertice
34         2. BFS/DFS do vertice mais distante => diametro = maior
    distancia
35
36 -> Lowest Common Ancestor (LCA)
37     - O(V) para achar o LCA de 2 vertices
38     - O(V) para pre-processar

```

9.2 Bipartido - @Info

```

1 Grafo Bipartido
2
3 * Definicao
4     - vertices podem ser divididos em 2 conjuntos disjuntos
5     - todas as arestas conectam vertices de conjuntos diferentes
6     - nao ha arestas entre vertices do mesmo conjunto
7     - nao ha ciclos de tamanho impar
8     > EX: arvores sao bipartidas
9
10 * Aplicacoes

```

9.3 Dag - @Info

```

1 Grafo Direcionado Aciclico (DAG):
2 * Definicao
3     - tem direcao
4     - nao tem ciclos
5     - problemas com ele => usar DP (estados da DP = vertices DAG)
6     - so tem um topological sort
7 * Aplicacoes
8     - Single Source (Shortest / Longest) Path na DAG => O(V + E)
9     - Numero de caminhos entre 2 vertices => O(V + E)
10    - Numero de caminhos de um vertice para todos os outros => O(V + E)
11    - DP de 'minimizacao', 'maximizacao', 'contar algo' => menor | maior |
    contar numero de caminhos na recursao de DP na DAG
12 * Exemplos
13     - mochila
14     - troco

```

9.4 Dag - Sslp

```

1 // Description: Finds SSLP (Single Source Longest Path) in a directed
    acyclic graph.
2 // Complexity: O(V + E)
3 // OBS: Not tested
4 vector<vector<pair<int,int>>> adj;
5
6 vector<int> dagLongestPath(int s, int n) {
7
8     vector<int> topsort = topologicalSort();
9     vector<int> dist(n, INT_MIN);
10    dist[s] = 0;
11
12    for (int i = 0; i < n; i++) {
13        int nodeIndex = topsort[i];
14        if (dist[nodeIndex] != INT_MIN) {
15            auto adjacentEdges = adj[nodeIndex];
16            for (auto [u, w] : adjacentEdges) {
17                int newDist = dist[nodeIndex] + w;
18                if (dist[u] == INT_MIN) dist[u] = newDist;
19                else dist[u] = max(dist[u], newDist);
20            }
21        }
22    }
23
24    return dist;
25 }

```

9.5 Dag - Sssp

```

1 // Description: Encontra SSSP (Single Source Shortest Path) em um grafo
    iacclico direcionado.
2 // Complexity: O(V + E)
3 // OBS: Nao testado
4 vector<vector<pair<int,int>>> adj;
5
6 vector<int> dagShortestPath(int s, int n) {
7
8     vector<int> topsort = topologicalSort();
9     vector<int> dist(n, INT_MAX);
10    dist[s] = 0;
11
12    for (int i = 0; i < n; i++) {
13        int nodeIndex = topsort[i];
14        if (dist[nodeIndex] != nodeIndex) {
15            auto adjacentEdges = adj[nodeIndex];
16            for (auto [u, w] : adjacentEdges) {
17                int newDist = dist[nodeIndex] + w;
18                if (dist[u] == INT_MAX) dist[u] = newDist;
19                else dist[u] = min(dist[u], newDist);
20            }
21        }
22    }
23
24    return dist;
25 }

```

9.6 Dag - Fishmonger

```

1 // Given the number of cities 3 <= n <= 50, available time 1 <= t <= 1000,
    and two n x n matrices (one gives travel times and another gives
    tolls between two cities), choose a route from the port city (vertex
    0) in such a way that the fishmonger has to pay as little tolls as
    possible to arrive at the market city (vertex n-1) within a certain
    time t
2
3 // Cada estado eh um vertice da DAG (node, tempoRestante)
4
5 pii dp(int cur, int t_left) {
6     if (t_left < 0) return {INF, INF};
7     if (cur == n-1) return {0, 0};
8     if (memo[cur][t_left] != {-1, -1}) return memo[cur][t_left];
9     pii ans = {INF, INF};
10    for (int X = 0; X < n; ++X)
11        if (cur != X) {
12            auto &[tollpaid, timeneeded] = dp(X, t_left - travelTime[cur][X]);
13            if (tollpaid + toll[cur][X] < ans.first) {
14                ans.first = tollpaid + toll[cur][X];
15                ans.second = timeneeded + travelTime[cur][X];
16            }
17        }
18    return memo[cur][t_left] = ans;
19 }

```

9.7 Dag - Numero De Caminhos 2 Vertices

```

1 // Description: Encontra o únmero de caminhos entre dois évrtes em um
    grafo iacclico direcionado.
2 // Complexity: O(V + E)
3
4 const int MAXN = 1e5 + 5;
5
6 int dp[MAXN],
7 int mod = 1e9 + 7, n;
8 vector<vector<int>> adj;
9
10 int countPaths(int s, int d) {
11     if (s == d) return 1;
12     if (dp[s] != -1) return dp[s];
13
14     int c = 0;
15     for (int& neigh : adj[s]) {
16         int x = countPaths(neigh, d);
17         if (x != -1)
18             c = (c % mod + x % mod) % mod;
19     }
20     return (dp[s] = (c == 0) ? -1 : c);
21 }
22
23 int countPossiblePaths(int s, int d) {
24     memset(dp, -1, sizeof dp);
25     int c = countPaths(s, d);
26     if (c == -1) return 0;

```

```

27     return c;
28 }
29
30 void solve() {
31     int n, ed; cin >> n >> ed;
32     adj.resize(n);
33
34     for (int i = 0; i < ed; i++) {
35         int u, v; cin >> u >> v;
36         adj[u].push_back(v);
37     }
38
39     int src, end; cin >> src >> end; // 0-based
40     cout << countPossiblePaths(src, end) << endl;
41 }

```

9.8 Eulerian - @Info

```

1 Eulerian Graph:
2
3 * Eulerian Path (Eulerian Tour):
4     - caminho que atravessa grafo apenas 1 vez
5     - Grafo Nao direcionado: tem um se e somente se tiver 0 ou 2 vertices
    de grau impar
6     - Grafo Direcionado: tem um se e somente se
7         1. todos os vertices tiverem o mesmo numero de arestas entrando e
            saindo
8         2. eh 'conexo' (considerando arestas bidirecionadas)
9 * Definicao
10     - nao direcionado
11     - conexo
12     - grau de todos os vertices par

```

9.9 Eulerian - Euler Path

```

1 // Description: Encontra um caminho euleriano em um grafo direcionado
2 // Complexidade: O(E)
3 // OBS: testar com bidirecionado / encontrar versao que aceita
    bidirecionado
4
5 int N;
6 vector<vi> adj;
7 vi hierholzer(int s) {
8     vi ans, idx(N, 0), st;
9     st.push_back(s);
10    while (!st.empty()) {
11        int u = st.back();
12        if (idx[u] < (int)adj[u].size()) {
13            st.push_back(adj[u][idx[u]]);
14            ++idx[u];
15        }
16        else {
17            ans.push_back(u);
18            st.pop_back();
19        }
20    }

```

```

21     reverse(ans.begin(), ans.end());
22     return ans;
23 }

```

10 Matematica

10.1 Casas

```

1 // Descriptiuon: Conta quantas casas decimais certo numero tem
2
3 int casas(double a) {
4     return (int)floor(1 + log10((double)a))
5 }

```

10.2 Ciclo Em Funcao

```

1 // Description: Encontra o tamanho do ciclo de uma çãfuno  $f(x) = (Z \cdot x + I) \% M$  a partir de um valor inicial L.
2 // Complexidade:  $O(\text{lambda} + \mu)$  | lambda = tamanho do ciclo | mu = tamanho
    do prefixo antes do ciclo.
3 // Return: pair<int, int> = {mu, lambda} | mu = tamanho do prefixo antes
    do ciclo | lambda = tamanho do ciclo.
4 // Parameters: x0 = valor inicial para encontrar o ciclo.
5 int f(int x); // f(x) do problema
6
7 pii floydCycleFinding(int x0) {
8     int t = f(x0), h = f(f(x0));
9     while (t != h) { t = f(t); h = f(f(h)); }
10    int mu = 0; h = x0;
11    while (t != h) { t = f(t); h = f(h); ++mu; }
12    int lambda = 1; h = f(t);
13    while (t != h) { h = f(h); ++lambda; }
14    return {mu, lambda};
15 }

```

10.3 Contar Quanti Solucoes Eq 2 Variaveis

```

1 // Description: Dada uma equacao de 2 variaveis, calcula quantas
    combinacoes {x,y}
2 // inteiras que resolvem essa equacao
3 // Complexidade:  $O(\sqrt{c})$ 
4 // y = numerador / denominador
5 int numerador(int x) { return c - x; } // expressao do numerador
6 int denominador(int x) { return 2 * x + 1; } // expressao do denominador
7
8 int count2VariableIntegerEquationAnswers() {
9
10    unordered_set<pair<int,int>, PairHash> ans; int lim = sqrt(c);
11    for(int i=1; i<= lim; i++) {
12        if (numerador(i) % denominador(i) == 0) {
13            int x = i, y = numerador(i) / denominador(i);
14            if(!ans.count({x,y}) and !ans.count({y,x}))
15                ans.insert({x,y});
16        }
17    }

```

```

18     return ans.size();
19 }
20 }

```

10.4 Conversao De Bases

```

1 // Converter um decimal (10) para base n [2, 8, 10, 16]
2 // Complexidade: O(log n)
3 char charForDigit(int digit) {
4     if (digit > 9) return digit + 87;
5     return digit + 48;
6 }
7
8 string decimalToBase(int n, int base = 10) {
9     if (not n) return "0";
10    stringstream ss;
11    for (int i = n; i > 0; i /= base) {
12        ss << charForDigit(i % base);
13    }
14    string s = ss.str();
15    reverse(s.begin(), s.end());
16    return s;
17 }
18
19 // Converter um numero de base [2, 8, 10, 16] para decimal (10)
20 // Complexidade: O(n)
21 int intForDigit(char digit) {
22     int intDigit = digit - 48;
23     if (intDigit > 9) return digit - 87;
24     return intDigit;
25 }
26
27 int baseToDecimal(const string& n, int base = 10) {
28     int result = 0;
29     int basePow = 1;
30     for (auto it = n.rbegin(); it != n.rend(); ++it, basePow *= base)
31         result += intForDigit(*it) * basePow;
32     return result;
33 }

```

10.5 Decimal Para Fracao

```

1 // Converte um decimal para fracao irredutivel
2 // Complexidade: O(log n)
3 pair<int, int> toFraction(double n, unsigned p) {
4     const int tenP = pow(10, p);
5     const int t = (int) (n * tenP);
6     const int rMdc = mdc(t, tenP);
7     return {t / rMdc, tenP / rMdc};
8 }

```

10.6 Dois Primos Somam Num

```

1 // Description: Verifica se dois numeros primos somam um numero n.
2 // Complexity: O(sqrt(n))
3 bool twoNumsSumPrime(int n) {

```

```

4
5     if(n % 2 == 0) return true;
6     return isPrime(n-2);
7 }

```

10.7 Factorial

```

1 unordered_map<int, int> memo;
2
3 // Factorial
4 // Complexidade: O(n), onde n eh o numero a ser fatorado
5 int factorial(int n) {
6     if (n == 0 || n == 1) return 1;
7     if (memo.find(n) != memo.end()) return memo[n];
8     return memo[n] = n * factorial(n - 1);
9 }

```

10.8 Fast Exponentiation

```

1 const int mod = 1e9 + 7;
2
3 // Fast Exponentiation: retorna a^b % mod
4 // Quando usar: quando precisar calcular a^b % mod
5 int fexp(int a, int b)
6 {
7     int ans = 1;
8     while (b)
9     {
10         if (b & 1)
11             ans = ans * a % mod;
12         a = a * a % mod;
13         b >>= 1;
14     }
15     return ans;
16 }

```

10.9 Fast Fibonacci

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #define _ std::ios::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);
5 #define all(a) a.begin(), a.end()
6 #define int long long int
7 #define double long double
8 #define vi vector<int>
9 #define pii pair<int,int>
10 #define endl "\n"
11 #define print_v(a) for(auto x : a) cout<<x<<" ";cout<<endl
12 #define print_vp(a) for(auto x : a) cout<<x.first<<" "<<x.second<< endl
13 #define f(i,s,e) for(int i=s;i<e;i++)
14 #define rf(i,e,s) for(int i=e-1;i>=s;i--)
15 #define CEIL(a, b) ((a) + (b - 1))/b
16 #define TRUNC(x, n) floor(x * pow(10, n))/pow(10, n)
17 #define ROUND(x, n) round(x * pow(10, n))/pow(10, n)
18 #define dbg(x) cout << #x << " = " << x << " ";

```

```

19 #define dbg1(x) cout << #x << " = " << x << endl;
20 using namespace std;
21
22 string decimal_to_bin(int n) {
23     string bin = bitset<sizeof(int) * 8>(n).to_string();
24     auto loc = bin.find('1');
25     // remove leading zeros
26     if (loc != string::npos)
27         return bin.substr(loc);
28     return "0";
29 }
30
31 int fastfib(int n) {
32     string bin_of_n = decimal_to_bin(n);
33
34     int f[] = { 0, 1 };
35
36     for (auto b : bin_of_n) {
37         int f2i1 = f[1] * f[1] + f[0] * f[0];
38         int f2i = f[0] * (2 * f[1] - f[0]);
39
40         if (b == '0') {
41             f[0] = f2i;
42             f[1] = f2i1;
43         } else {
44             f[0] = f2i1;
45             f[1] = f2i1 + f2i;
46         }
47     }
48
49     return f[0];
50 }
51
52 int main() {
53     int n = 13;
54     int fib = fastfib(n);
55     cout << "F(" << n << ") = " << fib << "\n";
56 }

```

10.10 Fatorial Grande

```

1 static BigInteger[] dp = new BigInteger[1000000];
2
3 public static BigInteger factorialDP(BigInteger n) {
4     dp[0] = BigInteger.ONE;
5     for (int i = 1; i <= n.intValue(); i++) {
6         dp[i] = dp[i - 1].multiply(BigInteger.valueOf(i));
7     }
8     return dp[n.intValue()];
9 }

```

10.11 Fibonacci Modulo

```

1 long pisano(long m)
2 {
3     long prev = 0;

```

```

4     long curr = 1;
5     long res = 0;
6
7     for(int i = 0; i < m * m; i++)
8     {
9         long temp = 0;
10        temp = curr;
11        curr = (prev + curr) % m;
12        prev = temp;
13
14        if (prev == 0 && curr == 1)
15            res = i + 1;
16    }
17    return res;
18 }
19
20 // Calculate Fn mod m
21 long fibonacciModulo(long n, long m)
22 {
23
24     // Getting the period
25     long pisanoPeriod = pisano(m);
26
27     n = n % pisanoPeriod;
28
29     long prev = 0;
30     long curr = 1;
31
32     if (n == 0)
33         return 0;
34     else if (n == 1)
35         return 1;
36
37     for(int i = 0; i < n - 1; i++)
38     {
39         long temp = 0;
40         temp = curr;
41         curr = (prev + curr) % m;
42         prev = temp;
43     }
44     return curr % m;
45 }

```

10.12 Mmc Mdc - Euclides Extendido

```

1 // Description: Retorna mdc(a, b) e referencia inteiros x, y t.q ax + by =
   mdc(a, b).
2 // Complexidade: O(log(min(a, b)))
3
4 int extEuclid(int a, int b, int &x, int &y) {
5     int xx = y = 0;
6     int yy = x = 1;
7     while (b) {
8         int q = a/b;
9         tie(a, b) = tuple(b, a%b);
10        tie(x, xx) = tuple(xx, x-q*xx);
11        tie(y, yy) = tuple(yy, y-q*yy);

```

```

12     }
13     return a;
14 }

```

10.13 Mmc Mdc - Mdc

```

1 // Description: Calcula o mdc de dois numeros inteiros.
2 // Complexidade: O(logn) onde n eh o maior numero
3 int mdc(int a, int b) {
4     for (int r = a % b; r; a = b, b = r, r = a % b);
5     return b;
6 }

```

10.14 Mmc Mdc - Mdc Multiplo

```

1 // Description: Calcula o MDC de um vetor de inteiros.
2 // Complexidade: O(nlogn) onde n eh o tamanho do vetor
3 int mdc_many(vector<int> arr) {
4     int result = arr[0];
5
6     for (int& num : arr) {
7         result = mdc(num, result);
8
9         if(result == 1) return 1;
10    }
11    return result;
12 }

```

10.15 Mmc Mdc - Mmc

```

1 // Description: Calcula o mmc de dois números inteiros.
2 // Complexidade: O(logn) onde n eh o maior numero
3 int mmc(int a, int b) {
4     return a / mdc(a, b) * b;
5 }

```

10.16 Mmc Mdc - Mmc Multiplo

```

1 // Description: Calcula o mmc de um vetor de inteiros.
2 // Complexidade: O(nlogn) onde n eh o tamanho do vetor
3 int mmc_many(vector<int> arr)
4 {
5     int result = arr[0];
6
7     for (int &num : arr)
8         result = (num * result / mdc(num, result));
9     return result;
10 }

```

10.17 Modulo - @Info

```

1 SOMA
2 (a + b) % m = ((a % m) + (b % m)) % m
3
4 SUBTRAO

```

```

5 (a - b) % m = ((a % m) - (b % m) + m) % m
6
7 MULTIPLICAO
8 (a * b) % m = ((a % m) * (b % m)) % m
9
10 DIVISAO
11 (a / b) % m = (a * b^-1) % m
12 // se m eh primo = ((a % m) * (b^(m-2) % m)) % m.
13 // else = (a * modInverse(b, m)) % m
14
15 POTENCIA
16 (a ^ b) % m = ((a % m) ^ b) % m = modPow(a, b, m)

```

10.18 Modulo - Divisao E Potencia Mod M

```

1 // Retorna a % m (garante que o resultado é positivo)
2 int mod(int a, int m) {
3     return ((a%m) + m) % m;
4 }
5
6 // Description: retorna b^(-1) mod m, ou -1 se não existir.
7 // Complexidade: O(log(min(b, m)))
8 int modInverse(int b, int m) {
9     int x, y;
10    int d = extEuclid(b, m, x, y);
11    if (d != 1) return -1;
12    return mod(x, m);
13 }
14
15 // Description: retorna b^p mod m
16 // Complexidade: O(log(p))
17 int modPow(int b, int p, int m) {
18     if (p == 0) return 1;
19     int ans = modPow(b, p/2, m);
20     ans = mod(ans*ans, m);
21     if (p&1) ans = mod(ans*b, m);
22     return ans;
23 }

```

10.19 Modulo - Fibonacci Modulo

```

1 // Descricao: Calcula o n-esimo numero de Fibonacci modulo P
2 // Complexidade: O(log(n))
3
4 int mostSignificantBitPosition(int n) {
5     int msb_position = 63;
6     while (!(1 << (msb_position-1) & n)) && msb_position >= 0)
7         msb_position--;
8     return msb_position;
9 }
10
11 int fib (int n, int P) {
12
13     int msb_position = mostSignificantBitPosition(n);
14
15     int a=0, b=1;

```

```

16
17 for (int i=msb_position; i>=0;--i) {
18     int d = (a%P) * ((b%P)*2 - (a%P) + P),
19         e = (a%P) * (a%P) + (b%P)*(b%P);
20     a = d % P;
21     b = e % P;
22
23     if (((n >> i) & 1) != 0) {
24         int c = (a + b) % P;
25         a = b;
26         b = c;
27     }
28 }
29 return a;
30 }

```

10.20 N Fibonacci

```

1 int dp[MAX];
2
3 int fibonacciDP(int n) {
4     if (n == 0) return 0;
5     if (n == 1) return 1;
6     if (dp[n] != -1) return dp[n];
7     return dp[n] = fibonacciDP(n-1) + fibonacciDP(n-2);
8 }
9
10 int nFibonacci(int minus, int times, int n) {
11     if (n == 0) return 0;
12     if (n == 1) return 1;
13     if (dp[n] != -1) return dp[n];
14     int aux = 0;
15     for(int i=0; i<times; i++) {
16         aux += nFibonacci(minus, times, n-minus);
17     }
18 }

```

10.21 Numeros Grandes

```

1 public static void BBigInteger() {
2
3     BigInteger a = BigInteger.valueOf(10000000000);
4     a = new BigInteger("10000000000");
5
6     // çð0peraes com inteiros grandes
7     BigInteger arit = a.add(a);
8     arit = a.subtract(a);
9     arit = a.multiply(a);
10    arit = a.divide(a);
11    arit = a.mod(a);
12
13    // çãComparao
14    boolean bool = a.equals(a);
15    bool = a.compareTo(a) > 0;
16    bool = a.compareTo(a) < 0;
17    bool = a.compareTo(a) >= 0;

```

```

18        bool = a.compareTo(a) <= 0;
19
20    // ãConverso para string
21    String m = a.toString();
22
23    // ãConverso para inteiro
24    int _int = a.intValue();
25    long _long = a.longValue();
26    double _doub = a.doubleValue();
27
28    // @Potncia
29    BigInteger _pot = a.pow(10);
30    BigInteger _sqr = a.sqrt();
31
32 }
33
34 public static void BigDecimal() {
35
36     BigDecimal a = new BigDecimal("10000000000");
37     a = new BigDecimal("1000000000.0000000000");
38     a = BigDecimal.valueOf(1000000000, 10);
39
40
41    // çð0peraes com reais grandes
42    BigDecimal arit = a.add(a);
43    arit = a.subtract(a);
44    arit = a.multiply(a);
45    arit = a.divide(a);
46    arit = a.remainder(a);
47
48    // çãComparao
49    boolean bool = a.equals(a);
50    bool = a.compareTo(a) > 0;
51    bool = a.compareTo(a) < 0;
52    bool = a.compareTo(a) >= 0;
53    bool = a.compareTo(a) <= 0;
54
55    // ãConverso para string
56    String m = a.toString();
57
58    // ãConverso para inteiro
59    int _int = a.intValue();
60    long _long = a.longValue();
61    double _doub = a.doubleValue();
62
63    // @Potncia
64    BigDecimal _pot = a.pow(10);
65 }

```

10.22 Primos - Divisores De N - Listar

```

1 // Description: Retorna o numero de divisores de N
2 // Complexidade: O(log(N))
3 // Exemplo: numDiv(60) = 12 {1,2,3,4,5,6,10,12,15,20,30,60}
4 int numDiv(int N) {
5     int ans = 1;
6     for (int i = 0; i < p.size() and p[i]*p[i] <= N; ++i) {

```



```

7         int power = 0;
8         while (N%p[i] == 0) { N /= p[i]; ++power; }
9         ans *= power+1;
10    }
11    return (N != 1) ? 2*ans : ans;
12 }

```

10.23 Primos - Divisores De N - Somar

```

1 // Description: Retorna a soma dos divisores de N
2 // Complexidade: O(log(N))
3 // Exemplo: sumDiv(60) = 168 : 1+2+3+4+5+6+10+12+15+20+30+60
4 int sumDiv(int N) {
5     int ans = 1;
6     for (int i = 0; i < p.size() and p[i]*p[i] <= N; ++i) {
7         int multiplier = p[i], total = 1;
8         while (N%p[i] == 0) {
9             N /= p[i];
10            total += multiplier;
11            multiplier *= p[i];
12        }
13        ans *= total;
14    }
15    if (N != 1) ans *= (N+1);
16    return ans;
17 }

```

10.24 Primos - Fatores Primos - Contar Diferentes

```

1 // Description: Retorna o numero de fatores primos diferentes de N
2 // Complexidade: O(sqrt(N))
3 // Exemplo: numDiffPF(60) = 3 {2, 3, 5}
4
5 int numDiffPF(int N) {
6     int ans = 0;
7     for (int i = 0; i < p.size() && p[i]*p[i] <= N; ++i) {
8         if (N%p[i] == 0) ++ans; // count this prime
9         factor
10        while (N%p[i] == 0) N /= p[i]; // only once
11    }
12    if (N != 1) ++ans;
13    return ans;
14 }

```

10.25 Primos - Fatores Primos - Listar

```

1 // Fatora um únmero em seus fatores primos
2 // Complexidade: O(sqrt(n))
3 // Ex: factorize(1200) = {2: 4, 3: 1, 5: 2}
4
5 map<int, int> factorize(int n) {
6     map<int, int> factorsOfN;
7     int lpf = 2;
8
9     while (n != 1) {
10        lpf = lowestPrimeFactor(n, lpf);

```

```

11        factorsOfN[lpf] = 1;
12        n /= lpf;
13        while (not (n % lpf)) {
14            factorsOfN[lpf]++;
15            n /= lpf;
16        }
17    }
18
19    return factorsOfN;
20 }

```

10.26 Primos - Fatores Primos - Somar

```

1 // Description: Retorna a soma dos fatores primos de N
2 // Complexidade: O(log(N))
3 // Exemplo: sumPF(60) = sumPF(2^2 * 3^1 * 5^1) = 2 + 2 + 3 + 5 = 12
4
5 int sumPF(int N) {
6     int ans = 0;
7     for (int i = 0; i < p.size() && p[i]*p[i] <= N; ++i)
8         while (N%p[i] == 0) { N /= p[i]; ans += p[i]; }
9     if (N != 1) ans += N;
10    return ans;
11 }

```

10.27 Primos - Is Prime

```

1 // Descricao: Funcao que verifica se um numero n eh primo.
2 // Complexidade: O(sqrt(n))
3 bool isPrime(int n) {
4     return n > 1 and lowestPrimeFactor(n) == n;
5 }

```

10.28 Primos - Lowest Prime Factor

```

1 // Description: Funcao auxiliar que retorna o menor fator primo de n.
2 // Complexidade: O(sqrt(n))
3
4 int lowestPrimeFactor(int n, int startPrime = 2) {
5     if (startPrime <= 3) {
6         if (not (n & 1)) return 2;
7         if (not (n % 3)) return 3;
8         startPrime = 5;
9     }
10
11    for (int i = startPrime; i * i <= n; i += (i + 1) % 6 ? 4 : 2)
12        if (not (n % i))
13            return i;
14    return n;
15 }

```

10.29 Primos - Miller Rabin

```

1 // Teste de primalidade de Miller-Rabin
2 // Complexidade: O(k*log^3(n)), onde k eh o numero de testes e n eh o
   numero a ser testado

```

```

3 // Descricao: Testa se um numero eh primo com uma probabilidade de erro de
  1/4^k
4
5 int mul(int a, int b, int m) {
6     int ret = a*b - int((long double)1/m*a*b+0.5)*m;
7     return ret < 0 ? ret+m : ret;
8 }
9
10 int pow(int x, int y, int m) {
11     if (!y) return 1;
12     int ans = pow(mul(x, x, m), y/2, m);
13     return y%2 ? mul(x, ans, m) : ans;
14 }
15
16 bool prime(int n) {
17     if (n < 2) return 0;
18     if (n <= 3) return 1;
19     if (n % 2 == 0) return 0;
20     int r = __builtin_ctzint(n - 1), d = n >> r;
21
22     // com esses primos, o teste funciona garantido para n <= 2^64
23     // funciona para n <= 3*10^24 com os primos ate 41
24     for (int a : {2, 325, 9375, 28178, 450775, 9780504, 795265022}) {
25         int x = pow(a, d, n);
26         if (x == 1 or x == n - 1 or a % n == 0) continue;
27
28         for (int j = 0; j < r - 1; j++) {
29             x = mul(x, x, n);
30             if (x == n - 1) break;
31         }
32         if (x != n - 1) return 0;
33     }
34     return 1;
35 }

```

10.30 Primos - Numero Fatores Primos De N

```

1 // Description: Retorna o numero de fatores primos de N
2 // Complexidade: O(log(N))
3
4 vi p; // vetor de primos p (sieve(10000000))
5
6 int numPF(int N) {
7     int ans = 0;
8     for (int i = 0; i < (int)p.size() && p[i]*p[i] <= N; ++i)
9         while (N%p[i] == 0) { N /= p[i]; ++ans; }
10    return ans + (N != 1);
11 }

```

10.31 Primos - Primo Grande

```

1 // Description: verificar se um numero > 9e18 eh primo
2 public static boolean isProbablePrime(BigInteger num, int certainty) {
3     return num.isProbablePrime(certainty);
4 }

```

10.32 Primos - Primos Relativos De N

```

1 // Description: Conta o numero de primos relativos de N
2 // Complexidade: O(log(N))
3 // Exemplo: countPrimosRelativos(60) = 16
  {1,7,11,13,17,19,23,29,31,37,41,43,47,49,53,59}
4 // Definicao: Dois numeros sao primos relativos se o mdc entre eles eh 1
5
6 int countPrimosRelativos(int N) {
7     int ans = N;
8     for (int i = 0; i < (int)p.size() && p[i]*p[i] <= N; ++i) {
9         if (N%p[i] == 0) ans -= ans/p[i];
10        while (N%p[i] == 0) N /= p[i];
11    }
12    if (N != 1) ans -= ans/N;
13    return ans;
14 }

```

10.33 Primos - Sieve

```

1 // Description: Gera todos os primos do intervalo [1,lim]
2 // Complexidade: O(n log log n)
3
4 int _sieve_size;
5 bitset<10000010> bs;
6 vi p;
7
8 void sieve(int lim) {
9     _sieve_size = lim+1;
10    bs.set();
11    bs[0] = bs[1] = 0;
12    f(i,2,_sieve_size) {
13        if (bs[i]) {
14            for (int j = i*i; j < _sieve_size; j += i) bs[j] = 0;
15            p.push_back(i);
16        }
17    }
18 }

```

10.34 Primos - Sieve Linear

```

1 // Sieve de Eratosthenes com linear sieve
2 // Encontra todos os números primos no intervalo [2, N]
3 // Complexidade: O(N)
4
5 vector<int> sieve(const int N) {
6
7     vector<int> lp(N + 1); // lp[i] = menor fator primo de i
8     vector<int> pr;
9
10    for (int i = 2; i <= N; ++i) {
11        if (lp[i] == 0) {
12            lp[i] = i;
13            pr.push_back(i);
14        }
15        for (int j = 0; i * pr[j] <= N; ++j) {

```

```

16         lp[i * pr[j]] = pr[j];
17         if (pr[j] == lp[i])
18             break;
19     }
20 }
21
22     return pr;
23 }

```

10.35 Tabela Verdade

```

1 // Gerar tabela verdade de uma expressão booleana
2 // Complexidade:  $O(2^n)$ 
3
4 vector<vector<int>> tabelaVerdade;
5 int indexTabela = 0;
6
7 void backtracking(int posicao, vector<int>& conj_bool) {
8
9     if(posicao == conj_bool.size()) { // Se chegou ao fim da BST
10         for(size_t i=0; i<conj_bool.size(); i++) {
11             tabelaVerdade[indexTabela].push_back(conj_bool[i]);
12         }
13         indexTabela++;
14
15     } else {
16         conj_bool[posicao] = 1;
17         backtracking(posicao+1, conj_bool);
18         conj_bool[posicao] = 0;
19         backtracking(posicao+1, conj_bool);
20     }
21 }
22
23 int main() {
24
25     int n = 3;
26
27     vector<int> linhaBool (n, false);
28     tabelaVerdade.resize(pow(2,n));
29
30     backtracking(0, linhaBool);
31 }

```

11 Matriz

11.1 Fibonacci Matricial

```

1
2
3 #include <bits/stdc++.h>
4 using namespace std;
5
6 typedef long long ll;
7
8 ll MOD;

```

```

9
10 const int MAX_N = 2;
11
12 struct Matrix { ll mat[MAX_N][MAX_N]; };
13
14 ll mod(ll a, ll m) { return ((a%m)+m) % m; }
15
16 Matrix matMul(Matrix a, Matrix b) {
17     Matrix ans;
18     for (int i = 0; i < MAX_N; ++i)
19         for (int j = 0; j < MAX_N; ++j)
20             ans.mat[i][j] = 0;
21     for (int i = 0; i < MAX_N; ++i)
22         for (int k = 0; k < MAX_N; ++k) {
23             if (a.mat[i][k] == 0) continue;
24             for (int j = 0; j < MAX_N; ++j) {
25                 ans.mat[i][j] += mod(a.mat[i][k], MOD) * mod(b.mat[k][j], MOD);
26
27                 ans.mat[i][j] = mod(ans.mat[i][j], MOD);
28             }
29         }
30     return ans;
31 }
32
33 Matrix matPow(Matrix base, int p) {
34     Matrix ans;
35     for (int i = 0; i < MAX_N; ++i)
36         for (int j = 0; j < MAX_N; ++j)
37             ans.mat[i][j] = (i == j);
38     while (p) {
39         if (p&1)
40             ans = matMul(ans, base);
41         base = matMul(base, base);
42         p >>= 1;
43     }
44     return ans;
45 }
46
47 int main() {
48     int n, m;
49     while (scanf("%d %d", &n, &m) == 2) {
50         Matrix ans;
51         ans.mat[0][0] = 1; ans.mat[0][1] = 1;
52         ans.mat[1][0] = 1; ans.mat[1][1] = 0;
53         MOD = 1LL << m;
54         ans = matPow(ans, n);
55         printf("%lld\n", ans.mat[0][1]);
56     }
57     return 0;
58 }

```

11.2 Maior Retângulo Binário Em Matriz

```

1 // Description: Encontra o maior retângulo binário em uma matriz.
2 // Time:  $O(n*m)$ 
3 // Space:  $O(n*m)$ 
4 tuple<int, int, int> maximalRectangle(vector<vector<int>>& mat) {

```

```

5  int r = mat.size();
6  if(r == 0) return {0, 0, 0};
7  int c = mat[0].size();
8
9  vector<vector<int>> dp(r+1, vector<int>(c));
10
11 int mx = 0;
12 int area = 0, height = 0, length = 0;
13 for(int i=1; i<r; ++i) {
14     int leftBound = -1;
15     stack<int> st;
16     vector<int> left(c);
17
18     for(int j=0; j<c; ++j) {
19         if(mat[i][j] == 1) {
20             mat[i][j] = 1+mat[i-1][j];
21             while(!st.empty() and mat[i][st.top()] >= mat[i][j])
22                 st.pop();
23
24             int val = leftBound;
25             if(!st.empty())
26                 val = max(val, st.top());
27
28             left[j] = val;
29         } else {
30             leftBound = j;
31             left[j] = 0;
32         }
33         st.push(j);
34     }
35     while(!st.empty()) st.pop();
36
37     int rightBound = c;
38     for(int j=c-1; j>=0; j--) {
39         if(mat[i][j] != 0) {
40
41             while(!st.empty() and mat[i][st.top()] >= mat[i][j])
42                 st.pop();
43
44             int val = rightBound;
45             if(!st.empty())
46                 val = min(val, st.top());
47
48             dp[i][j] = (mat[i][j]) * (((val-1)-(left[j]+1)+1));
49             if (dp[i][j] > mx) {
50                 mx = dp[i][j];
51                 area = mx;
52                 height = mat[i][j];
53                 length = (val-1)-(left[j]+1)+1;
54             }
55             st.push(j);
56         } else {
57             dp[i][j] = 0;
58             rightBound = j;
59         }
60     }
61 }

```

```

62
63 return {area, height, length};
64 }
65
66 int r = mat.size();
67 if(r == 0) return make_tuple(0, 0, 0);
68 int c = mat[0].size();
69
70 vector<vector<int>> dp(r+1, vector<int>(c));
71
72 int mx = 0;
73 int area = 0, height = 0, length = 0;
74 for(int i=1; i<r; ++i) {
75     int leftBound = -1;
76     stack<int> st;
77     vector<int> left(c);
78
79     for(int j=0; j<c; ++j) {
80         if(mat[i][j] == 1) {
81             mat[i][j] = 1+mat[i-1][j];
82             while(!st.empty() and mat[i][st.top()] >= mat[i][j])
83                 st.pop();
84
85             int val = leftBound;
86             if(!st.empty())
87                 val = max(val, st.top());
88
89             left[j] = val;
90         } else {
91             leftBound = j;
92             left[j] = 0;
93         }
94         st.push(j);
95     }
96     while(!st.empty()) st.pop();
97
98     int rightBound = c;
99     for(int j=c-1; j>=0; j--) {
100         if(mat[i][j] != 0) {
101
102             while(!st.empty() and mat[i][st.top()] >= mat[i][j])
103                 st.pop();
104
105             int val = rightBound;
106             if(!st.empty())
107                 val = min(val, st.top());
108
109             dp[i][j] = (mat[i][j]+1) * (((val-1)-(left[j]+1)+1)+1);
110             if (dp[i][j] > mx) {
111                 mx = dp[i][j];
112                 area = mx;
113                 height = mat[i][j];
114                 length = (val-1)-(left[j]+1)+1;
115             }
116             st.push(j);
117         } else {
118             dp[i][j] = 0;
119             rightBound = j;
120         }
121     }
122 }

```

```

119     }
120 }
121 }
122
123     return make_tuple(area, height, length);
124 }

```

11.3 Maxsubmatrixsum

```

1 // Description: Calcula a maior soma de uma submatriz MxN de uma matriz
    lxc
2 // Complexidade: O(l*c)
3
4 const int MAX = 1010; // 10^6 + 10
5
6 int mat[MAX][MAX];
7
8 int maxSubmatrixSum(int l, int c, int M, int N) {
9     int dp[l+1][c+1];
10
11     f(i,0,l+1) {
12         dp[i][0] = 0;
13         dp[0][i] = 0;
14     }
15
16     f(i,1,l+1) {
17         f(j,1,c+1) {
18             dp[i][j] = dp[i-1][j]
19                 + dp[i][j-1]
20                 - dp[i-1][j-1]
21                 + mat[i][j];
22         }
23     }
24
25     int ans = 0;
26     f(i,M,l+1) {
27         f(j,N,c+1) {
28             int ponto =
29                 dp[i][j]
30                 - dp[i-M][j]
31                 - dp[i][j-N]
32                 + dp[i-M][j-N];
33             ans = max(ans, ponto);
34         }
35     }
36     return ans;
37 }
38
39 void solve() {
40     int l, c, M, N; cin >> l >> c >> M >> N;
41
42     f(i,1,l+1) {
43         f(j,1,c+1) {
44             cin >> mat[i][j];
45         }
46     }
47

```

```

48     int ans = maxSubmatrixSum(l, c, M, N);
49
50     cout << ans << endl;
51 }

```

11.4 Max 2D Range Sum

```

1 // Maximum Sum
2 // O(n^3) 1D DP + greedy (Kadane's) solution, 0.000s in UVa
3
4 #include <bits/stdc++.h>
5 using namespace std;
6
7 #define f(i,s,e)      for(int i=s;i<e;i++)
8 #define MAX_n 110
9
10 int A[MAX_n][MAX_n];
11
12 int maxMatrixSum(vector<vector<int>> mat) {
13
14     int n = mat.size();
15     int m = mat[0].size();
16
17     f(i,0,n) {
18         f(j,0,m) {
19             if (j > 0)
20                 mat[i][j] += mat[i][j - 1];
21         }
22     }
23
24     int maxSum = INT_MIN;
25     f(l,0,m) {
26         f(r,l,m) {
27             vector<int> sum(n, 0);
28             f(row,0,n) {
29                 sum[row] = mat[row][r] - (l > 0 ? mat[row][l - 1] : 0);
30             }
31             int maxSubRect = sum[0];
32             f(i,1,n) {
33                 if (sum[i - 1] > 0)
34                     sum[i] += sum[i - 1];
35                 maxSubRect = max(maxSubRect, sum[i]);
36             }
37             maxSum = max(maxSum, maxSubRect);
38         }
39     }
40
41     return maxSum;
42 }

```

11.5 Potencia Matriz

```

1 // Description: Calcula a potencia de uma matriz quadrada A elevada a um
    expoente n
2
3 int MOD;

```

```

4  const int MAX_N = 2;
5
6  struct Matrix { int mat[MAX_N][MAX_N]; };
7
8  int mod(int a, int m) { return ((a%m)+m) % m; }
9
10 Matrix matMul(Matrix a, Matrix b) {
11     Matrix ans;
12     for (int i = 0; i < MAX_N; ++i)
13         for (int j = 0; j < MAX_N; ++j)
14             ans.mat[i][j] = 0;
15
16     for (int i = 0; i < MAX_N; ++i)
17         for (int k = 0; k < MAX_N; ++k) {
18             if (a.mat[i][k] == 0) continue;
19             for (int j = 0; j < MAX_N; ++j) {
20                 ans.mat[i][j] += mod(a.mat[i][k], MOD) * mod(b.mat[k][j],
21                     MOD);
22                 ans.mat[i][j] = mod(ans.mat[i][j], MOD);
23             }
24         }
25     return ans;
26
27 Matrix matPow(Matrix base, int p) {
28     Matrix ans;
29     for (int i = 0; i < MAX_N; ++i)
30         for (int j = 0; j < MAX_N; ++j)
31             ans.mat[i][j] = (i == j);
32     while (p) {
33         if (p&1)
34             ans = matMul(ans, base);
35         base = matMul(base, base);
36         p >>= 1;
37     }
38     return ans;
39 }
40
41 void solve() {
42
43 }

```

11.6 Verifica Se E Quadrado Magico

```

1 // Description: Verifica se uma matriz é um quadrado mágico.
2 // Complexidade: O(n^2)
3
4 int isMagicSquare(vector<vi> mat, int n) {
5     int i=0,j=0;
6     int sumd1 = 0, sumd2=0;
7     f(i,0,n) {
8         sumd1 += mat[i][i];
9         sumd2 += mat[i][n-1-i];
10    }
11    if(sumd1!=sumd2) return 0;
12
13    int ans = 0;

```

```

14
15    f(i,0,n) {
16        int rowSum = 0, colSum = 0;
17        f(j,0,n) {
18            rowSum += mat[i][j];
19            colSum += mat[j][i];
20        }
21        if (rowSum != colSum || colSum != sumd1) return 0;
22        ans = rowSum;
23    }
24    return ans;
25 }

```

11.7 Verificar Se Retangulo Cabe Em Matriz Binaria

```

1 // Description: Verifica se um retangulo C X L cabe em uma matriz binaria
2 // N X M
3 // Complexidade: O(N*M)
4 // OBS: comprParaAltura[i] = maior comprimento de retangulo de 1's com
5 // altura i que caiba na matriz
6 void histogram(int alturasHistograma[], int colunas, int comprParaAltura
7 []) {
8     int stack_top, width;
9     stack<int> st;
10
11     int i = 0;
12     while (i < colunas) {
13         if (st.empty() || alturasHistograma[st.top()] <= alturasHistograma
14             [i]) {
15             st.push(i++);
16         } else {
17             stack_top = alturasHistograma[st.top()];
18             st.pop();
19             width = i;
20
21             if (!st.empty())
22                 width = i - st.top() - 1;
23
24             if (comprParaAltura[stack_top] < width)
25                 comprParaAltura[stack_top] = width;
26         }
27     }
28
29     while (!st.empty()) {
30         stack_top = alturasHistograma[st.top()];
31         st.pop();
32         width = i;
33
34         if (!st.empty())
35             width = i - st.top() - 1;
36
37         if (comprParaAltura[stack_top] < width)
38             comprParaAltura[stack_top] = width;
39     }
40 }
41
42 bool fits(int c, int l, int comprParaAltura[], int maxRectSize) {

```

```

39     return (c <= maxRectSize and l <= comprimParaAltura[c]) or (l <=
maxRectSize and c <= comprimParaAltura[l]);
40 }
41
42 void solve() {
43
44     int n, m; cin >> n >> m; // dimensioes da matriz
45
46     int mat[n][m]; memset(mat, 0, sizeof(mat));
47
48     char str[m];
49     f(i,0,n) {
50         cin >> str;
51         f(j,0,m) {
52             if (str[j] == ',')
53                 mat[i][j] = 1;
54         }
55     }
56
57     int maxRectSize = min((int)500, max(n, m)); // ãdimenso maxima do
retangulo (max(comprimentoMaximo, larguraMaxima))
58
59     int comprimParaAltura[maxRectSize + 1];
60     memset(comprimParaAltura, -1, sizeof(comprimParaAltura));
61
62     int histogramaAux[m]; memset(histogramaAux, 0, sizeof(histogramaAux));
63
64     f(i,0,n) {
65         f(j,0,m) {
66             histogramaAux[j] = (mat[i][j] ? 1 + histogramaAux[j] : 0);
67         }
68         histogram(histogramaAux, m, comprimParaAltura);
69     }
70
71     int comprimentoRetangulo, larguraRetangulo; cin >>
comprimentoRetangulo >> larguraRetangulo;
72
73     if(fits(comprimentoRetangulo, larguraRetangulo, comprimParaAltura,
maxRectSize)) {
74         /* retangulo de comprimento comprimentoRetangulo e largura
larguraRetangulo cabe na matriz */
75     }
76 }

```

12 Strings

12.1 Kmp

```

1 #include <bits/stdc++.h>
2
3 void computeLPSArray(char* pat, int M, int* lps);
4
5 // Prints occurrences of pat[] in txt[]
6 void KMPSearch(char* pat, char* txt)
7 {
8     int M = strlen(pat);

```

```

9     int N = strlen(txt);
10
11     // create lps[] that will hold the longest prefix suffix
12     // values for pattern
13     int lps[M];
14
15     // Preprocess the pattern (calculate lps[] array)
16     computeLPSArray(pat, M, lps);
17
18     int i = 0; // index for txt[]
19     int j = 0; // index for pat[]
20     while ((N - i) >= (M - j)) {
21         if (pat[j] == txt[i]) {
22             j++;
23             i++;
24         }
25
26         if (j == M) {
27             printf("Found pattern at index %d ", i - j);
28             j = lps[j - 1];
29         }
30
31         // mismatch after j matches
32         else if (i < N && pat[j] != txt[i]) {
33             // Do not match lps[0..lps[j-1]] characters,
34             // they will match anyway
35             if (j != 0)
36                 j = lps[j - 1];
37             else
38                 i = i + 1;
39         }
40     }
41 }
42
43 // Fills lps[] for given pattern pat[0..M-1]
44 void computeLPSArray(char* pat, int M, int* lps)
45 {
46     // length of the previous longest prefix suffix
47     int len = 0;
48
49     lps[0] = 0; // lps[0] is always 0
50
51     // the loop calculates lps[i] for i = 1 to M-1
52     int i = 1;
53     while (i < M) {
54         if (pat[i] == pat[len]) {
55             len++;
56             lps[i] = len;
57             i++;
58         }
59         else // (pat[i] != pat[len])
60         {
61             // This is tricky. Consider the example.
62             // AAACAAAA and i = 7. The idea is similar
63             // to search step.
64             if (len != 0) {
65                 len = lps[len - 1];

```

```

66         // Also, note that we do not increment
67         // i here
68     }
69     else // if (len == 0)
70     {
71         lps[i] = 0;
72         i++;
73     }
74 }
75 }
76 }
77 }
78 // Driver code
79 int main()
80 {
81     char txt[] = "ABABDABACDABABCABAB";
82     char pat[] = "ABABCABAB";
83     KMPSearch(pat, txt);
84     return 0;
85 }
86

```

12.2 Aro Corasick

```

1 // C++ program for implementation of Aho Corasick algorithm
2 // for string matching
3 using namespace std;
4 #include <bits/stdc++.h>
5
6 // Max number of states in the matching machine.
7 // Should be equal to the sum of the length of all keywords.
8 const int MAXS = 500;
9
10 // Maximum number of characters in input alphabet
11 const int MAXC = 26;
12
13 // OUTPUT FUNCTION IS IMPLEMENTED USING out[]
14 // Bit i in this mask is one if the word with index i
15 // appears when the machine enters this state.
16 int out[MAXS];
17
18 // FAILURE FUNCTION IS IMPLEMENTED USING f[]
19 int f[MAXS];
20
21 // GOTO FUNCTION (OR TRIE) IS IMPLEMENTED USING g[][]
22 int g[MAXS][MAXC];
23
24 // Builds the string matching machine.
25 // arr - array of words. The index of each keyword is important:
26 // "out[state] & (1 << i)" is > 0 if we just found word[i]
27 // in the text.
28 // Returns the number of states that the built machine has.
29 // States are numbered 0 up to the return value - 1, inclusive.
30 int buildMatchingMachine(string arr[], int k)
31 {
32     // Initialize all values in output function as 0.
33     memset(out, 0, sizeof out);
34

```

```

34
35     // Initialize all values in goto function as -1.
36     memset(g, -1, sizeof g);
37
38     // Initially, we just have the 0 state
39     int states = 1;
40
41     // Construct values for goto function, i.e., fill g[][]
42     // This is same as building a Trie for arr[]
43     for (int i = 0; i < k; ++i)
44     {
45         const string &word = arr[i];
46         int currentState = 0;
47
48         // Insert all characters of current word in arr[]
49         for (int j = 0; j < word.size(); ++j)
50         {
51             int ch = word[j] - 'a';
52
53             // Allocate a new node (create a new state) if a
54             // node for ch doesn't exist.
55             if (g[currentState][ch] == -1)
56                 g[currentState][ch] = states++;
57
58             currentState = g[currentState][ch];
59         }
60
61         // Add current word in output function
62         out[currentState] |= (1 << i);
63     }
64
65     // For all characters which don't have an edge from
66     // root (or state 0) in Trie, add a goto edge to state
67     // 0 itself
68     for (int ch = 0; ch < MAXC; ++ch)
69         if (g[0][ch] == -1)
70             g[0][ch] = 0;
71
72     // Now, let's build the failure function
73
74     // Initialize values in fail function
75     memset(f, -1, sizeof f);
76
77     // Failure function is computed in breadth first order
78     // using a queue
79     queue<int> q;
80
81     // Iterate over every possible input
82     for (int ch = 0; ch < MAXC; ++ch)
83     {
84         // All nodes of depth 1 have failure function value
85         // as 0. For example, in above diagram we move to 0
86         // from states 1 and 3.
87         if (g[0][ch] != 0)
88         {
89             f[g[0][ch]] = 0;
90             q.push(g[0][ch]);
91

```



```

91     }
92 }
93
94 // Now queue has states 1 and 3
95 while (q.size())
96 {
97     // Remove the front state from queue
98     int state = q.front();
99     q.pop();
100
101     // For the removed state, find failure function for
102     // all those characters for which goto function is
103     // not defined.
104     for (int ch = 0; ch <= MAXC; ++ch)
105     {
106         // If goto function is defined for character 'ch'
107         // and 'state'
108         if (g[state][ch] != -1)
109         {
110             // Find failure state of removed state
111             int failure = f[state];
112
113             // Find the deepest node labeled by proper
114             // suffix of string from root to current
115             // state.
116             while (g[failure][ch] == -1)
117                 failure = f[failure];
118
119             failure = g[failure][ch];
120             f[g[state][ch]] = failure;
121
122             // Merge output values
123             out[g[state][ch]] |= out[failure];
124
125             // Insert the next level node (of Trie) in Queue
126             q.push(g[state][ch]);
127         }
128     }
129 }
130
131 return states;
132 }
133
134 // Returns the next state the machine will transition to using goto
135 // and failure functions.
136 // currentState - The current state of the machine. Must be between
137 // 0 and the number of states - 1, inclusive.
138 // nextInput - The next character that enters into the machine.
139 int findNextState(int currentState, char nextInput)
140 {
141     int answer = currentState;
142     int ch = nextInput - 'a';
143
144     // If goto is not defined, use failure function
145     while (g[answer][ch] == -1)
146         answer = f[answer];
147

```

```

148     return g[answer][ch];
149 }
150
151 // This function finds all occurrences of all array words
152 // in text.
153 void searchWords(string arr[], int k, string text)
154 {
155     // Preprocess patterns.
156     // Build machine with goto, failure and output functions
157     buildMatchingMachine(arr, k);
158
159     // Initialize current state
160     int currentState = 0;
161
162     // Traverse the text through the built machine to find
163     // all occurrences of words in arr[]
164     for (int i = 0; i < text.size(); ++i)
165     {
166         currentState = findNextState(currentState, text[i]);
167
168         // If match not found, move to next state
169         if (out[currentState] == 0)
170             continue;
171
172         // Match found, print all matching words of arr[]
173         // using output function.
174         for (int j = 0; j < k; ++j)
175         {
176             if (out[currentState] & (1 << j))
177             {
178                 cout << "Word " << arr[j] << " appears from "
179                     << i - arr[j].size() + 1 << " to " << i << endl;
180             }
181         }
182     }
183 }
184
185 // Driver program to test above
186 int main()
187 {
188     string arr[] = {"he", "she", "hers", "his"};
189     string text = "ahishers";
190     int k = sizeof(arr)/sizeof(arr[0]);
191
192     searchWords(arr, k, text);
193
194     return 0;
195 }

```

12.3 Calculadora Posfixo

```

1 // Description: Calculadora de expressoes posfixas
2 // Complexidade: O(n)
3 int posfixo(string s) {
4     stack<int> st;
5     for (char c : s) {
6         if (isdigit(c)) {

```

```

7         st.push(c - '0');
8     } else {
9         int b = st.top(); st.pop();
10        int a = st.top(); st.pop();
11        if (c == '+') st.push(a + b);
12        if (c == '-') st.push(a - b);
13        if (c == '*') st.push(a * b);
14        if (c == '/') st.push(a / b);
15    }
16 }
17 return st.top();
18 }

```

12.4 Chaves Colchetes Parenteses

```

1 // Description: Verifica se s tem uma sequência valida de {}, [] e ()
2 // Complexidade: O(n)
3 bool brackets(string s) {
4     stack<char> st;
5
6     for (char c : s) {
7         if (c == '(' || c == '[' || c == '{') {
8             st.push(c);
9         } else {
10            if (st.empty()) return false;
11            if (c == ')') and st.top() != '(' return false;
12            if (c == ']') and st.top() != '[' return false;
13            if (c == '}') and st.top() != '{' return false;
14            st.pop();
15        }
16    }
17
18    return st.empty();
19 }

```

12.5 Infixo Para Posfixo

```

1 // Description: Converte uma expressao matematica infixa para posfixa
2 // Complexidade: O(n)
3 int prec(char c) {
4     if (c == '^')
5         return 3;
6     else if (c == '/' || c == '*')
7         return 2;
8     else if (c == '+' || c == '-')
9         return 1;
10    else
11        return -1;
12 }
13
14 char associativity(char c) {
15     if (c == '^')
16         return 'R';
17     return 'L';
18 }
19

```

```

20 string infixToPostfix(string s) {
21     stack<char> st;
22     string result;
23
24     for (int i = 0; i < s.length(); i++) {
25         char c = s[i];
26
27         if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') || (c >= '0'
&& c <= '9'))
28             result += c;
29
30         else if (c == '(')
31             st.push('(');
32
33         else if (c == ')') {
34             while (st.top() != '(') {
35                 result += st.top();
36                 st.pop();
37             }
38             st.pop(); // Pop '('
39         }
40
41         else {
42             while (!st.empty() && prec(s[i]) < prec(st.top()) ||
43                 !st.empty() && prec(s[i]) == prec(st.top()) &&
44                 associativity(s[i]) == 'L') {
45                 result += st.top();
46                 st.pop();
47             }
48             st.push(c);
49         }
50     }
51
52     while (!st.empty()) {
53         result += st.top();
54         st.pop();
55     }
56
57     return result;
58 }

```

12.6 Is Subsequence

```

1 // Description: Verifica se a string s eh subsequencia da string t
2 // Complexidade Temporal: O(n)
3 // Complexidade Espacial: O(n)
4 bool isSubsequence(string& s, string& t) {
5     queue<char> q;
6     int cnt = 0;
7     for (int i = 0; i < t.size(); i++) {
8         q.push(t[i]);
9     }
10    int i = 0;
11    while (!q.empty()) {
12        if (s[i] == q.front()) {
13            cnt++;
14            i++;
15        }
16    }
17
18    return cnt == s.length();
19 }

```

```

15     }
16     q.pop();
17 }
18
19     return cnt == s.size();
20 }

```

12.7 Levenshtein

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 //a distância Levenshtein ou distância de edição entre dois "strings" é
   dada
4 //pelo número mínimo de operações necessárias para transformar um string no
   outro.
5 //Entendemos por "operações" a inserção, deleção ou substituição de um
   caractere.
6 int levenshteinDist(string word1, string word2) {
7     int size1 = word1.size();
8     int size2 = word2.size();
9     int verif[size1 + 1][size2 + 1]; // Verification matrix i.e. 2D array
   which will store the calculated distance.
10
11     // If one of the words has zero length, the distance is equal to the
   size of the other word.
12     if (size1 == 0)
13         return size2;
14     if (size2 == 0)
15         return size1;
16
17     // Sets the first row and the first column of the verification matrix
   with the numerical order from 0 to the length of each word.
18     for (int i = 0; i <= size1; i++)
19         verif[i][0] = i;
20     for (int j = 0; j <= size2; j++)
21         verif[0][j] = j;
22
23     // Verification step / matrix filling.
24     for (int i = 1; i <= size1; i++) {
25         for (int j = 1; j <= size2; j++) {
26             // Sets the modification cost.
27             // 0 means no modification (i.e. equal letters) and 1 means
   that a modification is needed (i.e. unequal letters).
28             int cost = (word2[j - 1] == word1[i - 1]) ? 0 : 1;
29
30             // Sets the current position of the matrix as the minimum
   value between a (deletion), b (insertion) and c (substitution).
31             // a = the upper adjacent value plus 1: verif[i - 1][j] + 1
32             // b = the left adjacent value plus 1: verif[i][j - 1] + 1
33             // c = the upper left adjacent value plus the modification
   cost: verif[i - 1][j - 1] + cost
34             verif[i][j] = min(
35                 min(verif[i - 1][j] + 1, verif[i][j - 1] + 1),
36                 verif[i - 1][j - 1] + cost
37             );
38         }
39     }

```

```

40
41     // The last position of the matrix will contain the Levenshtein
   distance.
42     return verif[size1][size2];
43 }
44
45 int main() {
46     string word1, word2;
47
48     cout << "Please input the first word: " << endl;
49     cin >> word1;
50     cout << "Please input the second word: " << endl;
51     cin >> word2;
52
53     cout << "The number of modifications needed in order to make one word
   equal to the other is: " << levenshteinDist(word1, word2) << endl;
54
55     system("pause");
56     return 0;
57 }

```

12.8 Lexico E Sintatico

```

1 unordered_map<char, int> precedence = {{'|', 1}, {'.', 2}, {'>', 3}, {'<',
   3}, {'=', 3}, {'#', 3}, {'+', 4}, {'-', 4}, {'*', 5}, {'/', 5}, {'^',
   6}};
2 string ops = "+-*/^>=<#.|";
3
4 bool isOp(char a) {
5     return ops.find(a) != string::npos;
6 }
7
8 bool checkIfSyntaxError(const string &str) {
9     int len = str.size();
10    stack<char> pilha;
11    for (int i = 0; i < len; i++) {
12        if (isOp(str[i])) {
13            if (i + 1 == len || i == 0) return true;
14            if (!isalnum(str[i - 1]) and str[i - 1] != ')') return true;
15            if (!isalnum(str[i + 1]) and str[i + 1] != '(') return true;
16            while (!pilha.empty() and pilha.top() != '(' and precedence[
   pilha.top()] >= precedence[str[i]]) {
17                pilha.pop();
18            }
19            pilha.push(str[i]);
20        }
21        else if (str[i] == '(') {
22            if (i > 0 and (isalnum(str[i - 1]) || str[i - 1] == ')'))
   return true;
23            pilha.push('(');
24        }
25        else if (str[i] == ')') {
26            if (i > 0 and str[i - 1] == '(') return true;
27            if (pilha.empty()) {
28                return true;
29            }
30            while (!pilha.empty() and pilha.top() != '(') {

```

```

31         pilha.pop();
32         if (pilha.empty()) {
33             return true;
34         }
35     }
36     if (!pilha.empty()) pilha.pop();
37 }
38 else if (isalnum(str[i])) {
39     if (i > 0 and isalnum(str[i - 1]) and !isOp(str[i - 1]) and
str[i - 1] != '(') return true;
40 }
41 }
42 while (!pilha.empty()) {
43     if (pilha.top() == '(') {
44         return true;
45     }
46     pilha.pop();
47 }
48 return false;
49 }
50
51 bool checkIfLexicalError(const string &str) {
52     for (char ch : str)
53         if (!isalnum(ch) and !isOp(ch) and ch != '(' and ch != ')')
54             return true;
55     return false;
56 }
57
58 void solve() {
59     string str; cin >> str;
60     if (checkIfLexicalError(str))
61         cout << "Lexical Error!" << endl;
62     else if (checkIfSyntaxError(str))
63         cout << "Syntax Error!" << endl;
64 }

```

12.9 Lexicograficamente Minima

```

1 // Descricao: Retorna a menor rotacao lexicografica de uma string.
2 // Complexidade:  $O(n * \log(n))$  onde n eh o tamanho da string
3 string minLexRotation(string str) {
4     int n = str.length();
5
6     string arr[n], concat = str + str;
7
8     for (int i = 0; i < n; i++)
9         arr[i] = concat.substr(i, n);
10
11     sort(arr, arr+n);
12
13     return arr[0];
14 }

```

12.10 Longest Common Substring

```

1 // Description: Encontra o comprimento da maior substring em comum entre 2 4
strings

```

```

2 // Complexidade Temporal:  $O(n * m)$ 
3 // Complexidade Espacial:  $O(\min(m, n))$ 
4 int LCSSubStr(string s, string t, int n, int m)
5 {
6     vector<vector<int>> dp(n + 1, vector<int>(m + 1, 0));
7     int ans = 0;
8
9     for (int i = 1; i <= n; i++) {
10         for (int j = 1; j <= m; j++) {
11             if (s[i - 1] == t[j - 1]) {
12                 dp[i][j] = dp[i - 1][j - 1] + 1;
13                 if (dp[i][j] > ans)
14                     ans = dp[i][j];
15             }
16             else
17                 dp[i][j] = 0;
18         }
19     }
20     return ans;
21 }
22
23 void solve() {
24     string x, y; cin >> x >> y;
25     cout << LCSSubStr(x, y, x.size(), y.size()) << endl;
26 }

```

12.11 Lower Upper

```

1 // Description: çãFunco que transforma uma string em lowercase.
2 // Complexidade:  $O(n)$  onde n é o tamanho da string.
3 string to_lower(string a) {
4     for (int i=0; i<(int)a.size(); ++i)
5         if (a[i]>='A' && a[i]<='Z')
6             a[i]+='a'-'A';
7     return a;
8 }
9
10 // para checar se é lowercase: islower(c);
11
12 // Description: çãFunco que transforma uma string em uppercase.
13 // Complexidade:  $O(n)$  onde n é o tamanho da string.
14 string to_upper(string a) {
15     for (int i=0; i<(int)a.size(); ++i)
16         if (a[i]>='a' && a[i]<='z')
17             a[i]-='a'-'A';
18     return a;
19 }
20
21 // para checar se e uppercase: isupper(c);

```

12.12 Numeros E Char

```

1 char num_to_char(int num) { // 0 -> '0'
2     return num + '0';
3 }
4
5 int char_to_num(char c) { // '0' -> 0

```

```

6     return c - '0';
7 }
8
9 char int_to_ascii(int num) { // 97 -> 'a'
10     return num;
11 }
12
13 int ascii_to_int(char c) { // 'a' -> 97
14     return c;
15 }

```

12.13 Ocorrencias

```

1 // Description: Função que retorna um vetor com as posições de todas as
2 // ocorrências de uma substring em uma string.
3 // Complexidade: O(n * m) onde n é o tamanho da string e m é o tamanho da
4 // substring.
5 vector<int> ocorrencias(string str, string sub){
6     vector<int> ret;
7     int index = str.find(sub);
8     while(index != -1){
9         ret.push_back(index);
10        index = str.find(sub, index+1);
11    }
12    return ret;
13 }

```

12.14 Palindromo

```

1 // Descrição: Função que verifica se uma string é um palindromo.
2 // Complexidade: O(n) onde n é o tamanho da string.
3 bool isPalindrome(string str) {
4     for (int i = 0; i < str.length() / 2; i++) {
5         if (str[i] != str[str.length() - i - 1]) {
6             return false;
7         }
8     }
9     return true;
10 }

```

12.15 Permutacao

```

1 // Função para gerar todas as permutações de uma string
2 // Complexidade: O(n!)
3
4 void permute(string& s, int l, int r) {
5     if (l == r)
6         permutacoes.push_back(s);
7     else {
8         for (int i = l; i <= r; i++) {
9             swap(s[l], s[i]);
10            permute(s, l+1, r);
11            swap(s[l], s[i]);
12        }
13    }

```

```

14 }
15
16 int main() {
17
18     string str = "ABC";
19     int n = str.length();
20     permute(str, 0, n-1);
21 }

```

12.16 Remove Acento

```

1 // Descrição: Função que remove acentos de uma string.
2 // Complexidade: O(n * m) onde n é o tamanho da string e m é o tamanho
3 // do alfabeto com acento.
4 string removeAcento(string str) {
5     string comAcento = "áéíóúâêôãõä";
6     string semAcento = "aeiouaeoaoa";
7
8     for(int i = 0; i < str.size(); i++){
9         for(int j = 0; j < comAcento.size(); j++){
10            if(str[i] == comAcento[j]){
11                str[i] = semAcento[j];
12                break;
13            }
14        }
15    }
16
17    return str;
18 }

```

12.17 Split Cria

```

1 // Descrição: Função que divide uma string em um vetor de strings.
2 // Complexidade: O(n * m) onde n é o tamanho da string e m é o tamanho
3 // do delimitador.
4 vector<string> split(string s, string del = " ") {
5     vector<string> retorno;
6     int start, end = -1*del.size();
7     do {
8         start = end + del.size();
9         end = s.find(del, start);
10        retorno.push_back(s.substr(start, end - start));
11    } while (end != -1);
12    return retorno;
13 }

```

12.18 String Hashing

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 struct Hash {
5     const int p1 = 31, m1 = 1e9 + 7;
6     const int p2 = 37, m2 = 1e9 + 9;
7     int hash1 = 0, hash2 = 0;

```

```

8 Hash(const string& s) {
9     compute_hash1(s);
10    compute_hash2(s);
11 }
12
13 void compute_hash1(const string& s) {
14     long p_pow = 1;
15     for(char ch: s) {
16         hash1 = (hash1 + (ch + 1 - 'a') * p_pow) % m1;
17         p_pow = (p_pow * p1) % m1;
18     }
19 }
20
21 void compute_hash2(const string& s) {
22     long p_pow = 1;
23     for(char ch: s) {
24         hash2 = (hash2 + (ch + 1 - 'a') * p_pow) % m2;
25         p_pow = (p_pow * p2) % m2;
26     }
27 }
28
29 // For two strings to be equal
30 // they must have same hash1 and hash2
31 bool operator==(const Hash& other) {
32     return (hash1 == other.hash1 && hash2 == other.hash2);
33 }
34 };
35
36 int main() {
37     const string s = "geeksforgeeks";
38     Hash h(s);
39     cout << "Hash values of " << s << " are: ";
40     cout << "(" << h.hash1 << ", " << h.hash2 << ")" << '\n';
41     return 0;
42 }

```

13 Vector

13.1 Contar Menores Elementos A Direita

```

1 // Description: Conta quantos elementos menores que o elemento atual
  // existem a direita do mesmo.
2 // Complexity: O(nlogn)
3
4 const int MAX = 100010; // Tamanho maximo do array de entrada
5
6 int ansArr[MAX]; // Array que armazena a resposta
7
8 void merge(pair<int, int> a[], int start, int mid, int end) {
9     pair<int, int> f[mid - start + 1], s[end - mid];
10
11     int n = mid - start + 1;
12     int m = end - mid;
13
14     for(int i = start; i <= mid; i++)
15         f[i - start] = a[i];

```

```

16     for(int i = mid + 1; i <= end; i++)
17         s[i - mid - 1] = a[i];
18
19     int i = 0, j = 0, k = start;
20     int cnt = 0;
21
22     while(i < n and j < m) {
23         if (f[i].second <= s[j].second) {
24             ansArr[f[i].first] += cnt;
25             a[k++] = f[i++];
26         } else {
27             cnt++;
28             a[k++] = s[j++];
29         }
30     }
31
32     while(i < n) {
33         ansArr[f[i].first] += cnt;
34         a[k++] = f[i++];
35     }
36
37     while(j < m) {
38         a[k++] = s[j++];
39     }
40 }
41
42 void mergesort(pair<int, int> item[], int low, int high) {
43     if (low >= high) return;
44
45     int mid = (low + high) / 2;
46     mergesort(item, low, mid);
47     mergesort(item, mid + 1, high);
48     merge(item, low, mid, high);
49 }
50
51 void solve() {
52     int n; cin >> n;
53     int arr[n];
54     f(i,0,n) { cin >> arr[i]; }
55
56     pair<int, int> a[n];
57     memset(ansArr, 0, sizeof(ansArr));
58
59     f(i,0,n) {
60         a[i].second = arr[i];
61         a[i].first = i;
62     }
63
64     mergesort(a, 0, n - 1);
65
66     int ans = 0;
67     f(i,0,n) { ans += ansArr[i]; }
68     cout << ans << endl;
69 }

```

13.2 Contar Subarrays Somam K

```

1 // Descricao: Conta quantos subarrays de um vetor tem soma igual a k
2 // Complexidade: O(n)
3 int contarSomaSubarray(vector<int>& v, int k) {
4     unordered_map<int, int> prevSum; // map to store the previous sum
5
6     int ret = 0, currentSum = 0;
7
8     for(int& num : v) {
9         currentSum += num;
10
11         if (currentSum == k) ret++; /// Se a soma atual for igual a k,
            encontramos um subarray
12
13         if (prevSum.find(currentSum - k) != prevSum.end()) // se subarray
            com soma (currentSum - k) existir, sabe que [0:n] eh um subarray com
            soma k
14             ret += (prevSum[currentSum - k]);
15
16         prevSum[currentSum]++;
17     }
18
19     return ret;
20 }

```

13.3 Elemento Mais Frequente

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 // Encontra o unico elemento mais frequente em um vetor
5 // Complexidade: O(n)
6 int maxFreq1(vector<int> v) {
7     int res = 0;
8     int count = 1;
9
10    for(int i = 1; i < v.size(); i++) {
11
12        if(v[i] == v[res])
13            count++;
14        else
15            count--;
16
17        if(count == 0) {
18            res = i;
19            count = 1;
20        }
21    }
22
23    return v[res];
24 }
25
26 // Encontra os elemento mais frequente em um vetor
27 // Complexidade: O(n)
28 vector<int> maxFreqn(vector<int> v)
29 {
30     unordered_map<int, int> hash;
31     for (int i = 0; i < v.size(); i++)

```

```

32         hash[v[i]]++;
33
34     int max_count = 0, res = -1;
35     for (auto i : hash) {
36         if (max_count < i.second) {
37             res = i.first;
38             max_count = i.second;
39         }
40     }
41
42     vector<int> ans;
43     for (auto i : hash) {
44         if (max_count == i.second) {
45             ans.push_back(i.first);
46         }
47     }
48
49     return ans;
50 }

```

13.4 K Maior Elemento

```

1 // Description: Encontra o k-ésimo maior elemento de um vetor
2 // Complexidade: O(n)
3
4 int Partition(vector<int>& A, int l, int r) {
5     int p = A[l];
6     int m = l;
7     for (int k = l+1; k <= r; ++k) {
8         if (A[k] < p) {
9             ++m;
10            swap(A[k], A[m]);
11        }
12    }
13    swap(A[l], A[m]);
14    return m;
15 }
16
17 int RandPartition(vector<int>& A, int l, int r) {
18     int p = l + rand() % (r-l+1);
19     swap(A[l], A[p]);
20     return Partition(A, l, r);
21 }
22
23 int QuickSelect(vector<int>& A, int l, int r, int k) {
24     if (l == r) return A[l];
25     int q = RandPartition(A, l, r);
26     if (q+1 == k)
27         return A[q];
28     else if (q+1 > k)
29         return QuickSelect(A, l, q-1, k);
30     else
31         return QuickSelect(A, q+1, r, k);
32 }
33
34 void solve() {
35     vector<int> A = { 2, 8, 7, 1, 5, 4, 6, 3 };

```

```

36     int k = 1;
37     cout << QuickSelect(A, 0, A.size()-1, k) << endl;
38 }

```

13.5 Longest Common Subsequence

```

1 // Description: Encontra o tamanho da maior subsequencia comum entre duas
  strings
2 // Complexidade Temporal: O(n*m)
3 // Complexidade Espacial: O(m)
4 // Exemplo: "ABCDG" e "ADEB" => 2 ("AB")
5
6 int longestCommonSubsequence(string& text1, string& text2) {
7     int n = text1.size();
8     int m = text2.size();
9
10    vector<int> prev(m + 1, 0), cur(m + 1, 0);
11
12    for (int idx2 = 0; idx2 < m + 1; idx2++)
13        cur[idx2] = 0;
14
15    for (int idx1 = 1; idx1 < n + 1; idx1++) {
16        for (int idx2 = 1; idx2 < m + 1; idx2++) {
17            if (text1[idx1 - 1] == text2[idx2 - 1])
18                cur[idx2] = 1 + prev[idx2 - 1];
19
20            else
21                cur[idx2]
22                    = 0 + max(cur[idx2 - 1], prev[idx2]);
23        }
24        prev = cur;
25    }
26
27    return cur[m];
28 }

```

13.6 Maior Retangulo Em Histograma

```

1 // Calcula area do maior retangulo em um histograma
2 // Complexidade: O(n)
3 int maxHistogramRect(const vector<int>& hist) {
4     stack<int> s;
5     int n = hist.size();
6
7     int ans = 0, tp, area_with_top;
8
9     int i = 0;
10    while (i < n) {
11
12        if (s.empty() || hist[s.top()] <= hist[i])
13            s.push(i++);
14
15        else {
16            tp = s.top(); s.pop();
17
18            area_with_top = hist[tp] * (s.empty() ? i : i - s.top() - 1);

```

```

19
20            if (ans < area_with_top)
21                ans = area_with_top;
22        }
23    }
24
25    while (!s.empty()) {
26        tp = s.top(); s.pop();
27        area_with_top = hist[tp] * (s.empty() ? i : i - s.top() - 1);
28
29        if (ans < area_with_top)
30            ans = area_with_top;
31    }
32
33    return ans;
34 }
35
36 void solve() {
37     vector<int> hist = { 6, 2, 5, 4, 5, 1, 6 };
38     cout << maxHistogramRect(hist) << endl;
39 }

```

13.7 Maior Sequencia Subsequente

```

1 // Maior sequencia subsequente
2 // {6, 2, 5, 1, 7, 4, 8, 3} => {2, 5, 7, 8}
3
4 int maiorCrescente(vector<int> v) {
5     vector<int> lenght(v.size());
6     for(int k=0; k<v.size(); k++) {
7         lenght[k] = ;
8         for(int i=0; i<k; i++) {
9             if(v[i] < v[k]) {
10                 lenght[i] = max(lenght[k], lenght[i]+1)
11             }
12         }
13     }
14     return lenght.back();
15 }

```

13.8 Maior Subsequencia Comum

```

1 int s1[MAXN], s2[MAXN], tab[MAXN][MAXN];
2
3 // Description: Retorna o tamanho da maior subsequencia comum entre s1 e
  s2
4 // Complexidade: O(n*m)
5 int lcs(int a, int b){
6
7     if(tab[a][b]>=0) return tab[a][b];
8     if(a==0 or b==0) return tab[a][b]=0;
9     if(s1[a]==s2[b]) return 1 + lcs(a-1, b-1);
10    return tab[a][b] = max(lcs(a-1, b), lcs(a, b-1));
11 }
12
13 void solve() {

```



```

14
15 s1 = {1, 3, 2, 5, 4, 2, 3, 4, 5};
16 s2 = {1, 2, 3, 4, 5};
17 int n = s1.size(), m = s2.size();
18 memset(tab, -1, sizeof(tab));
19 cout << lcs(n, m) << endl; // 5
20 }

```

13.9 Maior Subsequência Crescente

```

1 // Retorna o tamanho da maior subsequencia crescente de v
2 // Complexidade: O(n log(n))
3 int maiorSubCrescSize(vector<int> &v) {
4
5     vector<int> pilha;
6     for (int i = 0; i < v.size(); i++) {
7         auto it = lower_bound(pilha.begin(), pilha.end(), v[i]);
8         if (it == pilha.end())
9             pilha.push_back(v[i]);
10        else
11            *it = v[i];
12    }
13
14    return pilha.size();
15 }
16
17 // Retorna a maior subsequencia crescente de v
18 // Complexidade: O(n log(n))
19 vector<int> maiorSubCresc(vector<int> &v) {
20
21     vector<int> pilha, resp;
22     int pos[MAXN], pai[MAXN];
23     for (int i = 0; i < v.size(); i++) {
24         auto it = lower_bound(pilha.begin(), pilha.end(), v[i]);
25         int p = it - pilha.begin();
26         if (it == pilha.end())
27             pilha.push_back(v[i]);
28         else
29             *it = v[i];
30         pos[p] = i;
31         if (p == 0)
32             pai[i] = -1; // seu pai é -1
33         else
34             pai[i] = pos[p - 1];
35     }
36
37     int p = pos[pilha.size() - 1];
38     while (p >= 0) {
39         resp.push_back(v[p]);
40         p = pai[p];
41     }
42     reverse(resp.begin(), resp.end());
43
44     return resp;
45 }
46
47 void solve() {

```

```

48     vector<int> v = {1, 3, 2, 5, 4, 2, 3, 4, 5};
49     cout << maiorSubCrescSize(v) << endl // 5
50     /*****
51     vector<int> ans = maiorSubCresc(v); // {1,2,3,4,5}
52     */

```

13.10 Maior Triângulo Em Histograma

```

1 // Calcula o maior triângulo em um histograma
2 // Complexidade: O(n)
3 int maiorTrianguloEmHistograma(const vector<int>& histograma) {
4
5     int n = histograma.size();
6     vector<int> esquerda(n), direita(n);
7
8     esquerda[0] = 1;
9     f(1, 1, n) {
10         esquerda[i] = min(histograma[i], esquerda[i - 1] + 1);
11     }
12
13     direita[n - 1] = 1;
14     rf(i, n - 1, 0) {
15         direita[i] = min(histograma[i], direita[i + 1] + 1);
16     }
17
18     int ans = 0;
19     f(1, 0, n) {
20         ans = max(ans, min(esquerda[i], direita[i]));
21     }
22
23     return ans;
24 }
25 }

```

13.11 Remove Repetitive

```

1 // Remove repetitive elements from a vector
2 // Complexity: O(n)
3 vector<int> removeRepetitive(const vector<int>& vec) {
4
5     unordered_set<int> s;
6     s.reserve(vec.size());
7
8     vector<int> ans;
9
10    for (int num : vec) {
11        if (s.insert(num).second)
12            v.push_back(num);
13    }
14
15    return ans;
16 }
17
18 void solve() {
19     vector<int> v = {1, 3, 2, 5, 4, 2, 3, 4, 5};
20     vector<int> ans = removeRepetitive(v); // {1, 3, 2, 5, 4}
21 }

```

13.12 Soma Maxima Sequencial

```
1 // Description: Soma maxima sequencial de um vetor
2 // Complexidade: O(n)
3 int max_sum(vector<int> s) {
4
5     int ans = 0, maior = 0;
6
7     for(int i = 0; i < s.size(); i++) {
8         maior = max(0, maior+s[i]);
9         ans = max(resp, maior);
10    }
11
12    return ans;
13 }
14
15 void solve() {
16     vector<int> v = {1,-3,5,-1,2,-1};
17     cout << max_sum(v) << endl; // 6 = {5,-1,2}
18 }
```

13.13 Subset Sum

```
1 // Description: Verifica se algum subset dentro do array soma igual a sum
2 // Complexidade Temporal: O(sum * n)
3 // Complexidade Espacial: O(sum * n)
4
5 bool isSubsetSum(vi set, int n, int sum) {
6     bool subset[n + 1][sum + 1];
7
8     for (int i = 0; i <= n; i++)
9         subset[i][0] = true;
10
11     for (int i = 1; i <= sum; i++)
12         subset[0][i] = false;
13
14     for (int i = 1; i <= n; i++) {
15         for (int j = 1; j <= sum; j++) {
16             if (j < set[i - 1])
17                 subset[i][j] = subset[i - 1][j];
18             if (j >= set[i - 1])
19                 subset[i][j]
20                     = subset[i - 1][j]
21                     || subset[i - 1][j - set[i - 1]];
22         }
23     }
24
25     return subset[n][sum];
26 }
```

13.14 Troco

```
1 // Description: Retorna o menor únmero de moedas para formar um valor n
2 // Complexidade: O(n*m)
3 vector<int> troco(vector<int> coins, int n) {
4     int first[n];
```

```
5     value[0] = 0;
6     for(int x=1; x<=n; x++) {
7         value[x] = INF;
8         for(auto c : coins) {
9             if(x-c >= 0 and value[x-c] + 1 < value[x]) {
10                 value[x] = value[x-c]+1;
11                 first[x] = c;
12             }
13         }
14     }
15
16     vector<int> ans;
17     while(n>0) {
18         ans.push_back(first[n]);
19         n -= first[n];
20     }
21     return ans;
22 }
23
24 void solve() {
25     vector<int> coins = {1, 3, 4};
26     vector<int> ans = troco(coins, 6); // {3,3}
27 }
```

14 Outros

14.1 Dp

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 const int MAX_gm = 30; // up to 20 garments at most and 20 models/garment
5 const int MAX_M = 210; // maximum budget is 200
6
7 int M, C, price[MAX_gm][MAX_gm]; // price[g (<= 20)][k (<= 20)]
8 int memo[MAX_gm][MAX_M]; // TOP-DOWN: dp table [g (< 20)][money
9                             (<= 200)]
10
11 int dp(int g, int money) {
12     if (money < 0) return -1e9;
13     if (g == C) return M - money;
14     if (memo[g][money] != -1)
15         return memo[g][money]; // avaliar linha g com dinheiro money (cada
16                                 caso pensavel)
17     int ans = -1;
18     for (int k = 1; k <= price[g][0]; ++k)
19         ans = max(ans, dp(g + 1, money - price[g][k]));
20     return memo[g][money] = ans;
21 }
22
23 int main() {
24     int TC;
25     scanf("%d", &TC);
26     while (TC--)
```

```

27     scanf("%d %d", &M, &C);
28     for (int g = 0; g < C; ++g)
29     {
30         scanf("%d", &price[g][0]); // store k in price[g][0]
31         for (int k = 1; k <= price[g][0]; ++k)
32             scanf("%d", &price[g][k]);
33     }
34     memset(memo, -1, sizeof memo); // TOP-DOWN: init memo
35     if (dp(0, M) < 0)
36         printf("no solution\n"); // start the top-down DP
37     else
38         printf("%d\n", dp(0, M));
39 }
40 return 0;
41 }

```

14.2 Binario

```

1 // Descricao: conversao de decimal para binario
2 // Complexidade: O(logn) onde n eh o numero decimal
3 string decimal_to_binary(int dec) {
4     string binary = "";
5     while (dec > 0) {
6         int bit = dec % 2;
7         binary = to_string(bit) + binary;
8         dec /= 2;
9     }
10    return binary;
11 }
12
13 // Descricao: conversao de binario para decimal
14 // Complexidade: O(logn) onde n eh o numero binario
15 int binary_to_decimal(string binary) {
16     int dec = 0;
17     int power = 0;
18     for (int i = binary.length() - 1; i >= 0; i--) {
19         int bit = binary[i] - '0';
20         dec += bit * pow(2, power);
21         power++;
22     }
23     return dec;
24 }

```

14.3 Binary Search

```

1 // Description: Implementao do algoritmo de busca binaria.
2 // Complexidade: O(logn) onde n eh o tamanho do vetor
3 int BinarySearch(<vector>int arr, int x){
4     int k = 0;
5     int n = arr.size();
6
7     for (int b = n/2; b >= 1; b /= 2) {
8         while (k+b < n && arr[k+b] <= x) k += b;
9     }
10    if (arr[k] == x) {
11        return k;

```

```

12    }
13 }

```

14.4 Fibonacci

```

1 vector<int> memo(MAX, -1);
2
3 // Descricao: Funcao que retorna o n-esimo termo da sequencia de Fibonacci
4 // utilizando programacao dinamica.
5 // Complexidade: O(n) onde n eh o termo desejado
6 int fibPD(int n) {
7     if (n <= 1) return n;
8     if (memo[n] != -1) return memo[n];
9     return memo[n] = fibPD(n - 1) + fibPD(n - 2);
10 }

```

14.5 Horario

```

1 // Descricao: Funcoes para converter entre horas e segundos.
2 // Complexidade: O(1)
3 int cts(int h, int m, int s) {
4     int total = (h * 3600) + (m * 60) + s;
5     return total;
6 }
7
8 tuple<int, int, int> cth(int total_seconds) {
9     int h = total_seconds / 3600;
10    int m = (total_seconds % 3600) / 60;
11    int s = total_seconds % 60;
12    return make_tuple(h, m, s);
13 }

```

14.6 Intervalos

```

1 // Conta quantos intervalos nao tem overlap ([a,b] e [b,c] nao geram)
2
3 bool cmp(const pair<int,int>& p1, const pair<int,int>& p2) {
4     if(p1.second != p2.second) return p1.second < p2.second;
5     return p1.first < p2.first;
6 }
7
8 int countNonOverlappingIntervals(vector<pair<int,int>> intervals) {
9     sort(all(intervals), cmp);
10    int firstTermino = intervals[0].second;
11    int ans = 1;
12    f(i,1,intervals.size()) {
13        if(intervals[i].first >= firstTermino) {
14            ans++;
15            firstTermino = intervals[i].second;
16        }
17    }
18
19    return ans;
20 }

```