

Conteúdo

1 Grafos	1
1.1 Árvore de Steiner	1
1.2 Árvore Geradora de Diâmetro Mínimo	2
1.3 Árvore Geradora de Diâmetro Mínimo (+ eficiente)	2
1.4 Árvore Geradora Mínima (Prim)	3
1.5 Bellman Ford	3
1.6 Caminho Mínimo em um DAG	4
1.7 Centro de uma Árvore	4
1.8 Ciclo Hamiltoniano de Custo Mínimo	5
1.9 Circuito e Passeio de Euler	5
1.10 Cliques Maximais	6
1.11 Cobertura Mínima por Caminhos em DAG	6
1.12 Componentes Fortemente Conexas	6
1.13 Corte Mínimo Geral (Stoer-Wagner)	7
1.14 Dijkstra	7
1.15 Emparelhamento Bipartido de Custo Máximo	8
1.16 Emparelhamento Máximo Bipartido $O(\sqrt{n} * m)$	8
1.17 Emparelhamento Máximo e Cobertura Mínima	9
1.18 Emparelhamento Máximo Geral (Edmonds)	10
1.19 Floyd Warshall	10
1.20 Fluxo Máximo de Custo Mínimo	11
1.21 Fluxo Máximo de Custo Mínimo (Uso Geral) - Corte Mínimo	11
1.22 Fluxo Máximo Push-Relabel	12
1.23 Fluxo Mínimo	13
1.24 Intersecção de Matróides	13
1.25 Isomorfismo de Árvores	14
1.26 Menor Ancestral Comum (LCA)	15
1.27 Pontes, Pontos de Articulação e Componentes Biconexas	15
1.28 Stable Marriage	16
1.29 Topological Sort	16
1.30 Two Satisfiability	17
1.31 Union Find e Árvore Geradora Mínima (Kruskal)	17
2 Programação Dinâmica	17
2.1 Longest Common Subsequence (LCS)	17
2.2 Longest Increasing Subsequence (LIS)	18
2.3 Mochila binária (knapsack)	18
3 Geométricos	19
3.1 Algoritmos Básicos para Circunferência	19
3.2 Algoritmos Básicos para Geométricos	19
3.3 Algoritmos de Intersecções	19
3.4 Círculo Gerador Mínimo	20
3.5 Convex Hull (Graham Scan)	20
3.6 Diâmetro de Pontos e Polígono	21
3.7 Distância Esférica	21
3.8 Estrutura e Base para Geométricos	21
3.9 Intersecção de Polígonos Convexos	22
3.10 Par de Pontos Mais Próximos	22
3.11 Verificações de Ponto em Polígono	23

4 Numéricos	23
4.1 Binomial Modular (e não modular)	23
4.2 Crivo de Eratóstenes	23
4.3 Eliminação de Gauss	24
4.4 Estrutura de Polinômio	24
4.5 Euclides Extendido	25
4.6 Exponenciação de Matrizes - Fibonnaci	25
4.7 Exponenciação modular rápida	25
4.8 Fatoração de Número Inteiro	25
4.9 Gray Code	26
4.10 Inverso Modular	26
4.11 Log Discreto	26
4.12 Operações com Frações	26
4.13 Operações com Matrizes	27
4.14 Phi de Euler	28
4.15 Raízes de Polinômio	28
4.16 Simplex	29
4.17 Teorema Chinês do Resto	30
5 Strings	30
5.1 Aho-Corasick	30
5.2 Algoritmo Z	31
5.3 Array de Sufixos	31
5.4 Array de Sufixos $n * \lg(n)$	32
5.5 Árvore de Sufixos	32
5.6 Busca de Strings (KMP)	33
5.7 Hash de Strings	33
5.8 Split	33
6 Miscelânea	34
6.1 Árvore de Intervalos	34
6.2 Árvore de Intervalos (c/ Lazy Propagation)	34
6.3 BigInteger em Java	34
6.4 Binary Indexed Tree/Fenwick Tree	35
6.5 Binary Indexed Tree/Fenwick Tree 2D	35
6.6 Binary Indexed Tree with range update	35
6.7 Calculador de Expressões	36
6.8 Convex Hull Trick	36
6.9 Decomposição Heavy Light	37
6.10 Funções para Datas	37
6.11 Knight Distance	38
6.12 Maior Retângulo em um Histograma	38
6.13 Operações com Bits	38
6.14 Operações com Matriz de Bits	39
6.15 Range Minimum Query 2D	40
6.16 Range Minimum Query (RMQ)	40
6.17 Rope (via árvore cartesiana)	41
7 Matemática	41
7.1 Geometria	41
7.2 Relações Binomiais	42
7.3 Equações Diofantinas	42
7.4 Fibonacci	42
7.5 Problemas clássicos	42
7.6 Séries Numéricas	43
7.7 Matrizes e Determinantes	43
7.8 Probabilidades	43

7.9 Teoria dos Números	43
----------------------------------	----

1 Grafos**1.1 Árvore de Steiner**

Autor: Douglas Oliveira Santos

Complexidade: $O(3^t)$, t = número de terminais

Dependencias: Floyd-Warshall

Descricao: Encontra o grafo de menor custo que conecta todos os vértices terminais, podendo utilizar os demais vértices.

```
#include <vector>
#include <cstring>
#include <algorithm>
using namespace std;

#define INF 0x3f3f3f3f
#define MAXN 120
#define MAXT 10

/* FILL ME */
int adj[MAXN][MAXN]; /* matriz de adj com custos */
int tt[MAXT]; /* vértices terminais */
int n, nt; /* número de vértices e de terminais */

int memo[1<<MAXT][MAXT];

vector<int> mask[MAXT];

/* FLOYD AQUI */

void getMask(int mask, int e, int& x, int& y, int n) {
    int j = 0;
    x = 0;
    y = 0;
    for (int i = 0; i < n; i++) {
        while (!(mask & (1 << j))) {
            j++;
        }
        if (e & (1 << i)) {
            x = x | (1 << j);
        }
        else {
            y = y | (1 << j);
        }
        j++;
    }
}

int minstree() {
    floyd();
    if (nt == 2) return d[tt[0]][tt[1]];
    for (int t = 0; t < nt-1; t++) {
        mask[t].clear();
        for (int j = 0; j < n; j++) {
            memo[(1<<t)][j] = d[j][tt[t]];
        }
    }
}
```

```

}
for (int i = 1; i <= (1 << (nt-1)) - 1; i++) {
    int x = __builtin_popcount(i);
    if (x > 1) {
        mask[x].push_back(i);
    }
}
for (int m = 2; m <= nt-2; m++) {
    for (int k = 0; k < mask[m].size(); k++) {
        int msk = mask[m][k];
        for (int i = 0; i < n; i++) {
            memo[msk][i] = INF;
        }
        for (int j = 0; j < n; j++) {
            int u = INF;
            int e;
            for (e = 0; e < (1 << (m-1)) - 1; e++) {
                e = e | (1 << (m-1));
                int x, y;
                getMask(msk, e, x, y, m);
                u = min(u, memo[x][j] + memo[y][j]);
            }
            for (int i = 0; i < n; i++) {
                memo[msk][i] = min(memo[msk][i], d[i][j] + u);
            }
        }
    }
}
int v = INF;
int q = tt[nt-1];
for (int j = 0; j < n; j++) {
    int u = INF;
    for (int e = 1; e < (1 << (nt - 1)) - 1; e++) {
        u = min(u, memo[e][j] +
            memo[e ^ ((1 << (nt - 1)) - 1)][j]);
    }
    v = min(v, d[q][j] + u);
}
return v;
}

/**** Exemplo simples de uso ****/
int main() {
    scanf("%d %d", &n, &nt);
    for (int i = 0; i < nt; i++) {
        int u;
        scanf("%d", &u);
        u--;
        tt[i] = u;
    }
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &adj[i][j]);
        }
    }
    printf("%d\n", minstree());
    return 0;
}

```

1.2 Árvore Geradora de Diâmetro Mínimo

Autor: Igor Assis

Complexidade: $O(n^2 + mn^2)$

Tempo de implementacao: ?

Testes: spoj.MDST

Descricao: Encontra o diametro da arvore geradora de diametro minimo e um absolute 1-center a partir do qual da para se obter a arvore.

```

#include <cstdio>
#include <cstring>
#include <utility>
#include <algorithm>
#include <queue>
#include <vector>

```

```
using namespace std;
```

```

#define MAXN 1012
#define INF 0x3f3f3f3f

```

```
/* FILL ME */
```

```

int n, m;
int adj[MAXN][MAXN], peso[MAXN][MAXN], nadj[MAXN];

```

```

int d[MAXN][MAXN], pred[MAXN][MAXN], t[2][MAXN], k[MAXN];
int U, L;

```

```
int mark[MAXN];
```

```

/* a arvore geradora de diametro minimo é a
   arvore de caminhos minimos a partir do vértices CAC: que é
   um vértice criado a distancia d/2 de _j_ na aresta (i,j) */
struct _CAC {int i, j, d;}; CAC;

```

```
/* trocar para floyd-warshall se tem pesos nas arestas */
```

```

int asp() {
    // all-pairs-shortest-path
    int vc = -1;
    memset(d, 0, sizeof(d));
    memset(k, -1, sizeof(k));
    for (int j = 0; j < n; j++) {
        queue<int> Q;
        memset(mark, 0, sizeof(mark));
        mark[j] = 1;
        pred[j][j] = -1;
        Q.push(j);
        while (!Q.empty()) {
            int u = Q.front(); Q.pop();
            if (k[u] == -1 || d[j][u] > d[j][k[j]]) k[j] = u;
            for (int i = 0; i < nadj[u]; i++)
                if (!mark[adj[u][i]]) {
                    Q.push(adj[u][i]);
                    pred[j][adj[u][i]] = u;
                    d[j][adj[u][i]] = d[j][u] + 1;
                    mark[adj[u][i]] = 1;
                }
        }
    }
    if (vc == -1 || d[j][k[j]] < d[vc][k[vc]]) vc = j;
}

```

```

return vc;
}

int update(int i, int j, int k) {
    int c, delta = t[j][k];
    for (c = 0; c < n; c++) t[j][c] -= delta;
    for (c = 0; c < n; c++)
        if (t[j][c] > 0 && t[i][c] > 0) break;
    if (c == n) {
        U = L;
        CAC.i = i; CAC.j = j; CAC.d = abs(U-2*d[j][k]);
        return 1;
    }
    return 0;
}

int mdst() {
    int j, vc;

    vc = asp();
    U = 2*d[vc][k[vc]];
    CAC.i = -1; CAC.j = vc; CAC.d = 0;

    for (int r = 0; r < n; r++)
        for (int u = 0; u < nadj[r]; u++) if (r < adj[r][u]) {
            int s = adj[r][u];
            if (k[r] == k[s]) continue;
            if ((L = peso[r][u] + d[r][k[s]] + d[s][k[r]]) >= U)
                continue;
            memcpy(t[0], d[r], sizeof(t[0]));
            memcpy(t[1], d[s], sizeof(t[1]));
            if (update(1, 0, k[s]) || update(0, 1, k[r]))
                continue;
            for (;;) {
                int maxv = -1, maxi, maxj;
                for (j = 0; j < n; j++)
                    if ((t[0][j] > 0 && t[1][j] > 0)) {
                        if (maxv == -1 || t[0][j] > maxv)
                            maxv = t[0][j], maxi = 0, maxj = j;
                        if (maxv == -1 || t[1][j] > maxv)
                            maxv = t[1][j], maxi = 1, maxj = j;
                    }
                L = L + t[1-maxi][maxj];
                if (L >= U) break;
                if(update(maxi, 1-maxi, maxj)) break;
            }
        }
    return U;
}

/**** Exemplo simples de uso ****/
int main(){
    return 0;
}

```

1.3 Árvore Geradora de Diâmetro Mínimo (+ eficiente)

Autor: Igor Assis

Complexidade: $O(n^3 + n^2 \log n)$

Tempo de implementacao: ?

Testes: spoj.MDST

Descricao: Encontra o diametro da arvore geradora de diametro minimo, mais rapido se só precisa do diametro.

```
#include <cstdio>
#include <cstring>
#include <utility>
#include <algorithm>
#include <queue>
#include <vector>
```

```
using namespace std;
```

```
#define MAXN 1123
#define INF 0x3f3f3f3f
```

```
/* FILL ME */
int n, m;
int adj[MAXN][MAXN], peso[MAXN][MAXN], nadj[MAXN];
```

```
int d[MAXN][MAXN], t[MAXN];
int mark[MAXN];
```

```
int mdst() {
    int i, j, k, u;
```

```
    if (n == 1)
        return 0;
```

```
    // all-pairs-shortest-path
    for (k = 0; k < n; k++) {
        queue<int> Q;
        memset(mark, 0, sizeof(mark));
        d[k][k] = 0;
        mark[k] = 1;
        Q.push(k);
        while (!Q.empty()) {
            u = Q.front(); Q.pop();
            for (i = 0; i < nadj[u]; i++)
                if (!mark[adj[u][i]]) {
                    Q.push(adj[u][i]);
                    d[k][adj[u][i]] = d[k][u] + 1;
                    mark[adj[u][i]] = 1;
                }
        }
    }
}
```

```
    // absolute local 1-center
```

```
int H = INF;
i = 0;
memset(t, 0, sizeof(t));
for (u = 0; u < n; u++)
    for (j = 0; j < nadj[u]; j++)
        if (u < adj[u][j]) {
            for (k = 0; k < n; k++)
                t[i] = max(t[i], min(d[u][k], d[adj[u][j]][k]));
            H = min(H, peso[u][j] + 2*t[i++]);
        }
}
```

```
int value = INF;
i = 0;
for (u = 0; u < n; u++)
    for (j = 0; j < nadj[u]; j++)
        if (u < adj[u][j] && 2*t[i++] <= H) {
            vector<pair<int, int> > L;
            for (int k = 0; k < n; k++)
                L.push_back(make_pair(d[u][k], d[adj[u][j]][k]));
            sort(L.begin(), L.end(), greater<pair<int, int> > ());
            int p = 0;
            value = min(value, 2*L[0].first);
            for (int k = 0; k < n; k++)
                if (L[p].second < L[k].second)
                    value = min(value, peso[u][j] +
                                L[p].second + L[k].first), p = k;
            value = min(value, 2*L[p].second);
        }

    return value;
}
```

**** Exemplo simples de uso ****

```
int main(){
    return 0;
}
```

1.4 Árvore Geradora Mínima (Prim)

Autor: Davi Costa

Complexidade: $O(m \log n)$

Tempo de implementacao: ?

Testes: uva.10034

Descricao: Encontra Arvore Geradora Minima

```
#include <queue>
#include <limits.h>
#include <cstdio>
#include <cstring>
```

```
using namespace std;
```

```
#define MAXN 101 //numero maximo de vertices
#define INF INT_MAX //nao ha perigo de overflow
```

```
/* FILL ME */
int adj[MAXN][MAXN]; //lista de adj
int custo[MAXN][MAXN]; //tamanho das arestas de adj
int nadj[MAXN]; //grau de cada vertice
```

```
int pai[MAXN]; //para reconstruir o caminho
int dist[MAXN]; //distancia de cada vertice a arvore
bool used[MAXN];
```

```
/*
    n: numero de vertices, s: origem (opcional)
    retorna peso total da arvore
*/
int prim(int n, int s = 0) {
    priority_queue<pair<int, int> > q;
```

```
int a, v;
int cost, nv = 0;
int ret = 0;
memset(pai, -1, sizeof(pai));
memset(used, 0, sizeof(used));
for (int i = 0; i < n; i++) dist[i] = INF;
dist[s] = 0;
pai[s] = s;
q.push(make_pair(0, s));
while (!q.empty() && nv < n) {
    a = q.top().second; q.pop();
    if (used[a]) continue;
    ret += dist[a];
    used[a] = true;
    nv++;
    for (int i = 0; i < nadj[a]; i++) {
        v = adj[a][i];
        if (!used[v]) {
            cost = custo[a][i];
            if (cost >= dist[v]) continue;
            dist[v] = cost;
            q.push(make_pair(-1*cost, v));
            pai[v] = a;
        }
    }
}
return ret;
}
```

**** Exemplo simples de uso ****

```
int main() {
    int n, m;
    int from, to, cost;
    while (scanf("%d %d", &n, &m) == 2 && n != 0) {
        memset(nadj, 0, sizeof(nadj));
        for (int i = 0; i < m; i++) {
            scanf("%d %d %d", &from, &to, &cost);
            custo[from][nadj[from]] = custo[to][nadj[to]] = cost;
            adj[from][nadj[from]++] = to;
            adj[to][nadj[to]++] = from;
        }
        printf("%d\n", prim(n));
    }
    return 0;
}
```

1.5 Bellman Ford

Autor: Igor Assis/Davi Costa

Complexidade: $O(n*m)$

Tempo de implementacao: ?

Testes: uva.10806, uva.423

Dependencias: Sem dependencias

Descricao: Caminho minimo com pesos negativos

```
#define MAXN 100 //Numero maximo de vertices
#define MAXM 10000 //Numero maximo de arestas
#define INF 0x3f3f3f3f
```

```
/* aresta (u,v) com peso w:
```

```

    orig[i] = u, dest[i] = v, peso[i] = w
    d[u], distancia da origem s ao vertice u
*/

/* FILL ME */
int orig[MAXM], dest[MAXM], peso[MAXM];

int d[MAXN], pai[MAXN];
/*
s: origem, n: numero de vertices, m: numero de arestas
retorna 1 se o grafo nao tem ciclo negativo alcançavel
a partir de s, 0 c.c
*/

int bellman_ford(int s, int n, int m) {
    int i, j;
    memset(pai,-1,sizeof(pai));
    pai[s] = s;
    for (i = 0; i < n; i++)
        d[i] = INF;
    d[s] = 0;
    for (i = 0; i < n-1; i++)
        for (j = 0; j < m; j++) {
            int u = orig[j], v = dest[j], w = peso[j];
            if (d[u] != INF && d[v] > d[u]+w) {
                d[v] = d[u]+w;
                pai[v] = u;
            }
        }
    for (j = 0; j < m; j++) {
        int u = orig[j], v = dest[j], w = peso[j];
        if (d[u] != INF && d[v] > d[u]+w) return 0;
    }
    return 1;
}

/***** Exemplo simples de uso *****/
int main() {
    int n, m;
    int from, to, cost;
    int origem, destino;
    while (scanf("%d %d",&n,&m) == 2 && n != 0) {
        int k = 0;
        for (int i = 0; i < m; i++) {
            scanf("%d %d %d",&from,&to,&cost);
            orig[k] = from;
            dest[k] = to;
            peso[k] = cost;
            k++;
            orig[k] = to;
            dest[k] = from;
            peso[k] = cost;
            k++;
        }
        scanf("%d %d",&origem,&destino);
        bellman_ford(origem,n,m);
        printf("%d\n",d[destino]);
    }
    return 0;
}

```

1.6 Caminho Mínimo em um DAG

Autor: NBMundial / Davi Costa
 Complexidade: $O(E + V)$
 Tempo de implementacao: ?
 Testes: uva-10350
 Dependencias: Topological Sort
 Descricao: Caminho minimo em DAG

```

#define MAXN 100
#define INF 0x3f3f3f

/* FILL ME */
int nadj[MAXN]; //grau de cada vertice
int adj[MAXN][MAXN]; //lista de adj.
int custo[MAXN][MAXN]; //refere-se a aresta de adj

int pai[MAXN]; //se precisar reconstruir o caminho
int d[MAXN]; //distancia de s ateh cada vertice
int tops[MAXN]; //topological sort
int path[MAXN]; //caminho ate t
bool used[MAXN];
int ip;

/*
n: numero de vertices, s: origem
*/
int calc_path(int n, int s) {
    topsort(n);
    memset(pai,-1,sizeof(pai));
    for (int i = 0; i < n; i++) d[i] = INF;
    d[s] = 0;
    pai[s] = 0;
    for (int i = 0; i < n; i++) {
        int x = tops[i];
        if (pai[x] == -1) continue;
        for (int j = 0; j < nadj[x]; j++) {
            int v = adj[x][j];
            int cost = custo[x][j];
            if (d[v] > d[x] + cost) {
                d[v] = d[x] + cost;
                pai[v] = x;
            }
        }
    }
}

/*Exemplo de uso*/

int main() {
    int n, m;
    int origem, destino;
    int from, to, cost;
    while (scanf("%d %d",&n,&m) == 2 && n != 0) {
        scanf("%d %d",&origem,&destino);
        memset(nadj,0,sizeof(nadj));
        for (int i = 0; i < m; i++) {
            scanf("%d %d %d",&from,&to,&cost);
            custo[from][nadj[from]] = custo[to][nadj[to]] = cost;
            adj[from][nadj[from]++] = to;
            adj[to][nadj[to]++] = from;
        }
    }
}

```

```

    }
    shortdag(n,origem);
    printf("%d\n",d[destino]);
}
return 0;
}

```

1.7 Centro de uma Árvore

Autor: Douglas Santos
 Complexidade: $O(n)$
 Testes: uva.12489
 Descricao: Encontra o centro de uma árvore.
 O centro de uma árvore é formado por um ou dois vértices tal que esses vértices são os mais distantes de um folha.

```

#include <cstdio>
#include <vector>
using namespace std;

```

```

#define MAXN 100010

```

```

/* FILL ME */
int n;
vector<int> adj[MAXN];

```

```

int de[MAXN];
int x[2][MAXN];

```

```

/* Retorna o par de vértice que forma o centro da árvore.
Se o centro tiver apenas um vértice, retorna -1 pro
segundo vértice do par */
pair<int, int> cTree(int n, vector<int> adj[]) {
    int c[2] = {0, 0};
    int ind = 0;
    int r = n;
    for (int i = 0; i < n; i++) {
        de[i] = adj[i].size();
        if (de[i] <= 1) {
            x[ind][c[ind]++] = i;
            r = r - 1;
        }
    }
    while (r > 0) {
        int ot = (ind + 1) % 2;
        c[ot] = 0;
        for (int i = 0; i < c[ind]; i++) {
            int u = x[ind][i];
            for (int j = 0; j < adj[u].size(); j++) {
                int v = adj[u][j];
                de[v] = de[v] - 1;
                if (de[v] == 1) {
                    x[ot][c[ot]++] = v;
                    r = r - 1;
                }
            }
        }
        ind = ot;
    }
}

```

```

    if (c[ind] == 1) return make_pair(x[ind][0], -1);
    return make_pair(x[ind][0], x[ind][1]);
}

/**** Exemplo simples de uso ****/
int main() {

    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        adj[i].clear();
    }
    int u, v;
    for (int i = 0; i < n-1; i++) {
        scanf("%d %d", &u, &v);
        u--;
        v--;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    pair<int, int> p;
    p = cTree(n, adj);
    printf("Centro(s) da arvore: %d ", p.first + 1);
    if (p.second != -1) printf("%d", p.second + 1);
    printf("\n");
    return 0;
}

```

1.8 Ciclo Hamiltoniano de Custo Mínimo

Autor: Alexandre Kunieda + Prefield
 Complexidade: $O(n^2 \cdot 2^n)$
 Testes: uva.11643
 Descrição: Resolve o problema do caixeiro viajante.
 Para obter um ciclo de custo mínimo, descomente os trechos comentados do código.

```

#include <vector>
using namespace std;

#define MAXN 18
#define INF 0x3f3f3f3f

/* FILL ME */
int n;
int graph[MAXN][MAXN]; //matriz de adjacências

int X[MAXN][1<<MAXN];
// int Y[MAXN][1<<MAXN];
// vector<int> path;

// void rec(int i, int S) {
//     if(S) rec(Y[i][S], S & ~(1<<i));
//     path.push_back(i);
// }

int tsp(int s=0) {
    int N = 1<<n;

    for(int i=0 ; i<n ; i++)
        for(int j=0 ; j<N ; j++) {

```

```

            X[i][j] = INF;
            //Y[i][j] = -1;
        }

    for(int i=0 ; i<n ; i++) {
        X[i][1<<i] = graph[s][i];
        //Y[i][1<<i] = s;
    }

    for(int S=1 ; S<N ; S++)
        for(int i=0 ; i<n ; i++) {
            if(!(S & (1<<i))) continue;
            for(int j=0 ; j<n ; j++) {
                if(S & (1<<j)) continue;
                if(X[j][S|(1<<j)] > X[i][S]+graph[i][j]) {
                    X[j][S|(1<<j)] = X[i][S]+graph[i][j];
                    //Y[j][S|(1<<j)] = i;
                }
            }
        }

    //path.clear();
    //rec(s, N-1);
    return X[s][N-1];
}

/**** Exemplo simples de uso ****/
#include <stdio>
#include <string>

```

```

int main() {
    memset(graph, 0x3f, sizeof(graph));

    n = 3;
    graph[0][1] = 1;
    graph[1][2] = 3;
    graph[2][0] = 2;

    printf("%d\n", tsp());

    return 0;
}

```

1.9 Circuito e Passeio de Euler

Autor: Marcelo Galvão Póvoa
 Complexidade: $O(n+m)$
 Testes: uva.10054, codeforces.267B
 Descrição: Verifica se há e encontra um circuito/passeio de Euler em grafo não-direcionado contendo todas arestas podendo ser paralelas ou laços e o grafo conexo ou não

```

#include <queue>
#include <string>
#include <vector>
#include <algorithm>
#include <stack>

using namespace std;
#define MAXM 100100

```

```

#define MAXN 100100

/* FILL ME */
int ea[MAXM], eb[MAXM], n, m;

vector<int> vtour; // resposta: lista de vértices
vector<int> g[MAXN];
int mrk[MAXN];

/* Retorna 1 se há circuito, 2 se há passeio ou 0 c.c */
int euler() {
    for (int i=0; i<n; i++) g[i].clear();

    for (int i=0; i<m; i++) {
        g[ea[i]].push_back(i);
        g[eb[i]].push_back(i);
        mrk[i] = 0;
    }

    int qi = 0, v0;
    for (int i=0; i<n; i++) {
        if (!qi && g[i].size()) v0 = i;
        if (g[i].size() % 2) v0 = i, qi++;
    }

    if (qi > 2) return 0;

    stack<int> st;
    st.push(v0);

    vtour.clear();
    while (!st.empty()) {
        int v = st.top();

        while (g[v].size() && mrk[g[v].back()]++)
            g[v].pop_back();

        if (g[v].empty()) {
            vtour.push_back(v);
            st.pop();
        }
        else {
            int k = g[v].back();
            st.push(v == ea[k] ? eb[k] : ea[k]);
        }
    }

    return (vtour.size() == m+1) ? (1+qi/2) : 0;
}

```

```

/**** Exemplo simples de uso ****/
int main() {
    scanf("%d%d", &n, &m);

    for (int i=0; i<m; i++)
        scanf("%d%d", &ea[i], &eb[i]);

    int p = euler();
    if (!p) printf("No tour\n");
    else {

```

```

    printf("%s\n", p==2 ? "Path" : "Tour");

    for (int i=0; i<vtour.size(); i++)
        printf("%d ", vtour[i]);
    printf("\n");
}
}

```

1.10 Cliques Maximais

Autor: Douglas Santos
 Complexidade: $O(3^{n/3})$
 Testes: SRM571-div1-550
 Descricao: Acha todas as cliques maximais de um grafo.

```

#include <cstring>
#include <algorithm>
using namespace std;

```

```

typedef long long int int64;

```

```

#define MAXN 55
#define INF 0x3f3f3f3f

```

```

/* FILL ME */
int n;
/* matriz de adj representada por mascara de bits */
int64 adj[MAXN];

```

```

void clique(int64 r, int64 p, int64 x) {
    if (p == 0 && x == 0) {
        /* r é uma clique maximal */
        return;
    }
    int pivot = -1;
    int menor = INF;
    for (int i = 0; i < n; i++) {
        if ( ((1LL << i) & p) || ((1LL << i) & x) ) {
            int x = __builtin_popcountll(p & ~(adj[i]));
            if (x < menor) {
                pivot = i;
                menor = x;
            }
        }
    }
}

for (int i = 0; i < n; i++) {
    if ((1LL << i) & p) {
        if (pivot != -1 && adj[pivot] & (1LL << i)) continue;
        clique(r | (1LL << i), p & adj[i], x & adj[i]);
        p = p ^ (1LL << i);
        x = x | (1LL << i);
    }
}
}

```

```

/**** Exemplo simples de uso ****/
int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {

```

```

        adj[i] = 0;
        for (int j = 0; j < n; j++) {
            int x;
            scanf("%d", &x);
            if (x == 1) adj[i] |= (1LL << j);
        }
    }
    clique(0, (1LL << n) - 1, 0);
    return 0;
}

```

1.11 Cobertura Mínima por Caminhos em DAG

Autor: Douglas Santos
 Complexidade: $O(n*m)$
 Dependências: Emparelhamento Máximo Bipartido
 Testes: SRM557-div1-550
 Descricao: Dado uma DAG encontra o menor número de caminhos necessários para cobrir todos os vértices. Cada vértice é coberto exatamente uma vez.

```

#include <vector>
#include <cstring>
#include <algorithm>
using namespace std;

```

```

#define MAXNDAG 130
#define MAXN 2*MAXNDAG

```

```

/* BPM AQUI */

```

```

/* FILL ME */
int n;
vector<int> dag[MAXNDAG];

```

```

int minpcover() {
    memset(nadj, 0, sizeof(nadj));
    nU = nV = n;

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < (int) dag[i].size(); j++) {
            int v = dag[i][j];
            adj[i][nadj[i]++] = v+n;
            adj[v+n][nadj[v+n]++] = i;
        }
    }
}

```

```

return n - maxbpm();
}

```

```

/* Abaixo apenas se for necessário imprimir a solução*/
vector<int> path[MAXNDAG];

```

```

void DFS(int u, int c) {
    path[c].push_back(u);
    if (conj[u+n] == -1) return;
    DFS(conj[u+n], c);
}

```

```

int getPaths() {

```

```

    int res = 0;
    for (int i = 0; i < n; i++) {
        if (conj[i] == -1) {
            path[res].clear();
            DFS(i, res);
            reverse(path[res].begin(), path[res].end());
            res++;
        }
    }
    return res;
}

```

```

/**** Exemplo simples de uso ****/

```

```

int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        dag[i].clear();
        int d;
        scanf("%d", &d);
        for (int j = 0; j < d; j++) {
            int v;
            scanf("%d", &v);
            v--;
            dag[i].push_back(v);
        }
    }
}

```

```

int res = minpcover();
getPaths();
printf("%d\n", res);

```

```

for (int i = 0; i < res; i++) {
    for (int j = 0; j < (int) path[i].size(); j++) {
        printf("%d ", path[i][j] + 1);
    }
    printf("\n");
}
return 0;
}

```

1.12 Componentes Fortemente Conexas

Autor: Igor Assis
 Complexidade: $O(n+m)$
 Testes: uva.247 uva.10510 SPOJ.CARPDAPIO
 Descricao: Encontra as componentes fortemente conexas de um grafo orientado. Componentes nomeadas de 1 à ncomp.

```

#include <cstring>
#include <algorithm>
#include <vector>
using namespace std;

```

```

#define MAXN 1024

```

```

/* Input - FILL ME */
vector<int> adj[MAXN];

```

```

/* Output */
int ncomp;        // numero de componentes fortemente conexas

```

```
int comp[MAXN]; // comp[i] = componente do vertice i

/* Variaveis auxiliares */
int vis[MAXN], stck[MAXN], t, high[MAXN];
int num;
```

```
void dfscc(int u) {
    int i, v;
    high[u] = vis[u] = num--;
    stck[t++] = u;
    for (i = 0; i < (int) adj[u].size(); i++) {
        v = adj[u][i];
        if (!vis[v]) {
            dfscc(v);
            high[u] = max(high[u], high[v]);
        } else if (vis[v] > vis[u] && !comp[v])
            high[u] = max(high[u], vis[v]);
    }
    if (high[u] == vis[u]) {
        ncomp++;
        do {
            v = stck[--t];
            comp[v] = ncomp;
        } while (v != u);
    }
}
```

```
void scc(int n) {
    ncomp = t = 0; num = n;
    memset(vis, 0, sizeof(vis));
    memset(comp, 0, sizeof(comp));
    for (int i = 0; i < n; i++)
        if (!vis[i]) dfscc(i);
}
```

```
/* Exemplo simples de uso */
#include <stdio>
```

```
int main(void){
    int i, u, v, n, m;

    scanf("%d%d", &n, &m);
    for (int i = 0; i < n; i++) adj[i].clear();
    for (i = 0; i < m; i++) {
        scanf("%d%d", &u, &v);
        adj[u].push_back(v);
    }

    scc(n);

    printf("Numero componentes: %d\n", ncomp);
    for (i = 0; i < n; i++)
        printf("componente[%d] = %d\n", i, comp[i]);

    return 0;
}
```

1.13 Corte Mínimo Geral (Stoer-Wagner)

Autor: Igor Naverniouk, Igor Assis

Complexidade: $O(n^3)$
 Testes: uva.10989 spojbr.EINSTEIN
 Descricao: Algoritmo que encontra o valor do corte mínimo
 dentre todos de um grafo nao-orientado com peso na aresta.

```
#include <algorithm>
using namespace std;

// Maximum number of vertices in the graph
#define MAXN 256

// Maximum edge weight (MAXW * NN * NN must fit into an int)
#define MAXW 1000

// Adjacency matrix and some internal arrays
int adj[MAXN][MAXN], v[MAXN], w[MAXN], na[MAXN];
bool a[MAXN];

int mincut(int n) {
    // init the remaining vertex set
    for(int i = 0; i < n; i++) v[i] = i;

    // run Stoer-Wagner
    int best = MAXW * n * n;
    while(n > 1) {
        // initialize the set A and vertex weights
        a[v[0]] = true;
        for( int i = 1; i < n; i++ ) {
            a[v[i]] = false;
            na[i - 1] = i;
            w[i] = adj[v[0]][v[i]];
        }

        // add the other vertices
        int prev = v[0];
        for(int i = 1; i < n; i++) {
            // find the most tightly connected non-A vertex
            int zj = -1;
            for(int j = 1; j < n; j++)
                if(!a[v[j]] && (zj < 0 || w[j] > w[zj]))
                    zj = j;

            // add it to A
            a[v[zj]] = true;

            // last vertex?
            if(i == n - 1) {
                // remember the cut weight
                best = min(best, w[zj]);

                // merge prev and v[zj]
                for(int j = 0; j < n; j++)
                    adj[v[j]][prev]=adj[prev][v[j]] += adj[v[zj]][v[j]];
                v[zj] = v[--n];
                break;
            }
            prev = v[zj];
        }

        // update the weights of its neighbours
        for(int j = 1; j < n; j++)
```

```
        if(!a[v[j]])
            w[j] += adj[v[zj]][v[j]];
    }
}

return best;
}

/* Exemplo simples de uso */
int main() {
    // preencha a matriz de adjacencias adj[][]
    // se nao existe a aresta (u,v) coloque *zero*
    int n, answer = mincut( n );
    return 0;
}
```

1.14 Dijkstra

Autor: Davi Costa
 Complexidade: $O(m \log n)$
 Tempo de implementacao: ?
 Testes: spoj.SHPATH, uva.423
 Descricao: Encontra caminho minimo em grafos com pesos ≥ 0

```
#include <queue>
#include <cstring>
using namespace std;

#define MAXN 101
#define INF INT_MAX //nao ha perigo de overflow

/* FILL ME */
int adj[MAXN][MAXN], nadj[MAXN]; //lista de adj
int cus[MAXN][MAXN]; //tamanho das arestas de adj

int dist[MAXN]; //distancia da origem ateh cada vertice
bool used[MAXN];
//int pai[MAXN]; //Caso queira reconstruir o caminho

// preenche o vetor de distancias dist
void dijkstra(int n, int s) {
    priority_queue<pair<int, int> > q;

    //memset(pai,-1,sizeof(pai));
    memset(used,0,sizeof(used));
    for (int i = 0; i < n; i++) dist[i] = INF;
    dist[s] = 0;
    //pai[s] = s;

    q.push(make_pair(0,s));
    while (!q.empty()) {
        int a = q.top().second;
        q.pop();

        if (used[a]) continue;
        used[a] = true;

        for (int i = 0; i < nadj[a]; i++) {
            int v = adj[a][i];
            int cost = dist[a] + cus[a][i];
            if (cost >= dist[v]) continue;
```

```

        dist[v] = cost;
        q.push(make_pair(-1*cost,v));
        //pai[v] = a;
    }
}

/**** Exemplo simples de uso ****/
int main() {
    int n, m;
    int origem,destino;
    int from, to, cost;
    while (scanf("%d %d",&n,&m) == 2 && n != 0) {
        scanf("%d %d",&origem,&destino);
        memset(nadj,0,sizeof(nadj));
        for (int i = 0; i < m; i++) {
            scanf("%d %d %d",&from,&to,&cost);
            cus[from][nadj[from]] = cus[to][nadj[to]] = cost;
            adj[from][nadj[from]++] = to;
            adj[to][nadj[to]++] = from;
        }
        dijkstra(n,origem);

        printf("%d\n",dist[destino]);
    }
    return 0;
}

```

1.15 Emparelhamento Bipartido de Custo Máximo

Autor: Chines/Davi Costa
 Complexidade: $O(n^3)$
 Testes: nuevo-3987
 Dependencias: Sem dependencias
 Descricao: Encontra o emparelhamento maximo de custo maximo, para custo minimo insira as arestas com peso negativo. Se uma aresta nao existe o valor na matriz deve ser $-1*INF$.
 Cuidado: NAO UTILIZAR MEMSET PARA O $-1*INF$ necessariamente $n \leq m$

```

#include <algorithm>
#include <vector>
#include <cstring>
using namespace std;

#define INF 0x3f3f3f3f
#define MAXN 351

/* FILL ME */
int n, m; //# de vertices em cada lado
int adj[MAXN][MAXN]; //Matriz de Adj

int labelx[MAXN], usedx[MAXN], lnk[MAXN];
int labely[MAXN], usedy[MAXN];
int mat; //Tamanho to match

//Auxiliar Caminho Aumentante
bool path(int i) {
    usedx[i] = 1;

```

```

    for (int j = 0; j < m; j++) {
        if (!usedy[j] && adj[i][j] != -INF &&
            !abs(adj[i][j] - labelx[i] - labely[j])) {
            usedy[j] = 1;
            if (lnk[j] == -1 || path(lnk[j])) {
                lnk[j] = i;
                return true;
            }
        }
    }
    return false;
}

//Apos preencher adj chamar match()
int match() {
    mat = 0;
    memset(lnk,-1,sizeof(lnk));
    memset(labely,0,sizeof(labely));
    for (int i = 0; i < n; i++) {
        labelx[i] = 0;
        for (int j = 0; j < m; j++)
            if (adj[i][j] > labelx[i]) labelx[i] = adj[i][j];
    }
    for (int k = 0; k < n; k++) {
        while (1) {
            memset(usedx,0,sizeof(usedx));
            memset(usedy,0,sizeof(usedy));
            if (path(k)) { mat++; break; }
            int del = INF;
            for (int i = 0; i < n; i++)
                if (usedx[i])
                    for (int j = 0; j < m; j++)
                        if (!usedy[j] && adj[i][j] != -INF)
                            del = min(del,labelx[i]+labely[j]-adj[i][j]);
            if (del == 0 || del == INF) break;
            for (int i = 0; i < n; i++)
                if (usedx[i]) labelx[i] -= del;
            for (int j = 0; j < m; j++)
                if (usedy[j]) labely[j] += del;
        }
    }
    int sum = 0;
    for (int i = 0; i < n; i++) sum += labelx[i];
    for (int j = 0; j < m; j++) sum += labely[j];
    return sum;
}

```

```

/**** Exemplo simples de uso ****/
int main() {
    int k, e;
    int from, to, cost;
    scanf("%d",&k);
    for (int z = 0; z < k; z++) {
        if (z != 0) printf("\n");
        scanf("%d %d",&n,&m);
        for (int i = 0; i < n; i++)
            for (int j = 0; j < m; j++)
                adj[i][j] = -INF;
        scanf("%d",&e);
        for (int i = 0; i < e; i++) {

```

```

            scanf("%d %d %d",&from,&to,&cost);
            adj[from][to] = -cost;
        }
        int r = -match();
        printf("%d\n",r);
    }
    return 0;
}

```

1.16 Emparelhamento Máximo Bipartido $O(\sqrt{n} * m)$

Autor: Douglas Santos
 Complexidade: $O(\sqrt{n} * m)$
 Testes: timus-1229, timus-1389
 Descricao: Encontra um emparelhamento máximo * em grafo bipartido.

```

#include <cstdio>
#include <cstring>
#include <queue>
#include <vector>

using namespace std;

#define MAXN 100010
#define NIL 0
#define INF 0x3f3f3f3f

/* FILL ME */
/* U = {1, 2, ..., nu}, V = {nu+1, nu+2, ..., nv} */
int nu, nv;
vector< int > G[MAXN];

int match[MAXN], dist[MAXN];

bool bfs() {
    int i, u, v, len;
    queue< int > Q;
    for(i=1; i<=nu; i++) {
        if(match[i]==NIL) {
            dist[i] = 0;
            Q.push(i);
        }
        else dist[i] = INF;
    }
}

dist[NIL] = INF;
while(!Q.empty()) {
    u = Q.front(); Q.pop();
    if(u!=NIL) {
        len = G[u].size();
        for(i=0; i<len; i++) {
            v = G[u][i];
            if(dist[match[v]]==INF) {
                dist[match[v]] = dist[u] + 1;
                Q.push(match[v]);
            }
        }
    }
}

```



```

    }
}

return (dist[NIL]!=INF);
}

bool dfs(int u) {
    int i, v, len;
    if(u!=NIL) {
        len = G[u].size();
        for(i=0; i<len; i++) {
            v = G[u][i];
            if(dist[match[v]]==dist[u]+1) {
                if(dfs(match[v])) {
                    match[v] = u;
                    match[u] = v;
                    return true;
                }
            }
        }
    }

    dist[u] = INF;
    return false;
}

return true;
}

int hopcroft_karp() {
    int matching = 0, i;
    while(bfs())
        for(i=1; i<=nu; i++)
            if(match[i]==NIL && dfs(i))
                matching++;
    return matching;
}

/* Exemplo simples de uso */
int main() {
    int m,a,b;
    scanf("%d %d %d",&nu,&nv,&m);
    for (int i=0; i<m; i++) {
        scanf("%d %d",&a,&b);
        b += nu;
        G[a].push_back(b);
        G[b].push_back(a);
    }

    printf("%d\n",hopcroft_karp());

    /* Imprime MATCH */
    for (int u = 1; u <= nu; u++) {
        if (match[u]) printf("%d %d\n", u, match[u]);
    }
    return 0;
}

```

1.17 Emparelhamento Máximo e Cobertura Mínima

Autor: Igor Assis / Douglas Santos

Complexidade: $O(n*m)$

Testes: uva.11419, uva.10080, cf-227d

Descricao: Encontra um emparelhamento máximo e uma cobertura de vértice mínima em grafo bipartido.

Para um grafo bipartido:

- |cobertura mínima de vertices| = |emparelhamento máximo|
 - |conj. independente máximo| = $n - |\text{emparelhamento máximo}|$
 - |cobertura mínima de arestas| = $n - |\text{emparelhamento máximo}|$

```

#include <cstring>
#include <vector>
using namespace std;
#define MAXN 2024

/* Input - FILL ME */
vector<int> adj[MAXN];
/* Output */
int conj[MAXN]; // conj[u] = v, se v for emparelhado com u
int cor[MAXN]; // cor[u] = partição do vertice u (0 ou 1)
/* Variavel auxiliar */
int vis[MAXN];

```

```

/* Função auxiliar */
bool DFS(int u, int c) {
    cor[u] = c;
    for (int i = 0; i < (int) adj[u].size(); i++) {
        int v = adj[u][i];
        if (cor[v] == c) return false;
        if (cor[v] == -1) {
            if (!DFS(v, c^1)) return false;
        }
    }
    return true;
}

```

```

/* Função auxiliar */
int aumenta(int u) {
    int i;
    for (i = 0; i < (int) adj[u].size(); i++) {
        int v = adj[u][i];
        if (vis[v]) continue; vis[v] = 1;
        if (conj[v] == -1 || aumenta(conj[v])) {
            conj[v] = u;
            conj[u] = v;
            return 1;
        }
    }
    return 0;
}

```

```

int maxbpm(int n) {
    int i;
    int res = 0;
    memset(cor, -1, sizeof(cor));
    for (int i = 0; i < n; i++) {
        if (cor[i] == -1)
            if (!DFS(i, 0)) return -1; // grafo não é bipartido
    }
}

```

```

memset(conj, -1, sizeof(conj));
for (i = 0; i < n; i++) {
    if (cor[i]) continue;
    memset(vis, 0, sizeof(vis));
    if (aumenta(i)) res++;
}
return res;
}

```

/* Código apenas para cobertura mínima de vertice */

/* Output */
 bool cover[MAXN]; //cover[i] -> vertice i pertence a cobertura

```

/* Função auxiliar */
void reach(int u) {
    int i;
    vis[u] = 1;
    for (i = 0; i < (int) adj[u].size(); i++) {
        int v = adj[u][i];
        if (!vis[v] && ((conj[u] != v && !cor[u]) ||
                        (conj[u] == v && cor[u]))) reach(v);
    }
}

```

```

int minbpec(int n) {
    int i;
    int res = maxbpm(n);
    memset(vis, 0, sizeof(vis));
    for (i = 0; i < n; i++) {
        if (cor[i]) continue;
        if (!vis[i] && conj[i] == -1) reach(i);
    }
    /* C = (U \ Rm) \ (V /\ Rm) */
    memset(cover, false, sizeof(cover));
    for (i = 0; i < n; i++) {
        if (!vis[i] && !cor[i]) cover[i] = true;
        if (vis[i] && cor[i]) cover[i] = true;
    }
    return res;
}

```

**** Exemplo simples de uso ****/
 #include <cstdio>

```

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

    scanf("%d %d", &n, &m);
    for(int i = 0; i < n; i++) adj[i].clear();
    for (i = 0; i < m; i++) {
        scanf("%d%d", &u, &v);
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
}

```

```

printf("Matching Maximo/Cobertura Minima: %d\n", minbpec(n));
printf("Arestas do Matching:\n");
for (i = 0; i < n; i++)
    if (conj[i] != -1)

```

```

    printf("%d %d ", i, conj[i]);
    printf("\nVertices na cobertura:\n");
    for (i = 0; i < n; i++)
        if (cover[i])
            printf("%d ", i);
    printf("\n");

    return 0;
}

```

1.18 Emparelhamento Máximo Geral (Edmonds)

Autor: Davi Costa
 Complexidade: $O(n^3)$
 Testes: uva-11439, nuevo-4130, spojbr-ENGENHAR
 Uso: CUIDADO - Nao utilizar o vertice 0
 - Para cada aresta 'i' (sem direcao) u-v,
 faca from[i] = u, to[i] = v e coloque i
 na lista de adjacencia de ambos u e v.
 - n e m devem ser utilizados obrigatoriamente.
 - E() retorna o tamanho do emparelhamento (# de casais).
 - mate[v] quando diferente de 0 indica que o vertice v esta
 casado com mate[v]

```

#include <algorithm>
#include <queue>
#include <cstring>
using namespace std;

```

```

#define MAXN 110
#define MAXM MAXN*MAXN

```

```

/* FILL ME */
int n,m;
int adj[MAXN][MAXN], nadj[MAXN], from[MAXM], to[MAXM];

```

```

int mate[MAXN], first[MAXN], label[MAXN];
queue<int> q;

```

```

#define OUTER(x) (label[x] >= 0)

```

```

void L(int x, int y, int nxy) {
    int join, v, r = first[x], s = first[y];
    if (r == s) return;
    nxy += n + 1;
    label[r] = label[s] = -nxy;
    while (1) {
        if (s != 0) swap(r,s);
        r = first[label[mate[r]]];
        if (label[r] != -nxy) label[r] = -nxy;
        else {
            join = r;
            break;
        }
    }
    v = first[x];
    while (v != join) {
        if (!OUTER(v)) q.push(v);
        label[v] = nxy; first[v] = join;
        v = first[label[mate[v]]];
    }
}

```

```

}
v = first[y];
while (v != join) {
    if (!OUTER(v)) q.push(v);
    label[v] = nxy; first[v] = join;
    v = first[label[mate[v]]];
}
for (int i = 0; i <= n; i++) {
    if (OUTER(i) && OUTER(first[i])) first[i] = join;
}
}
}

```

```

void R(int v, int w) {
    int t = mate[v]; mate[v] = w;
    if (mate[t] != v) return;
    if (label[v] >= 1 && label[v] <= n) {
        mate[t] = label[v];
        R(label[v],t);
        return;
    }
    int x = from[label[v]-n-1];
    int y = to[label[v]-n-1];
    R(x,y); R(y,x);
}

```

```

int E() {
    memset(mate,0,sizeof(mate));
    int r = 0;
    bool e7;
    for (int u = 1; u <= n; u++) {
        memset(label,-1,sizeof(label));
        while (!q.empty()) q.pop();
        if (mate[u]) continue;
        label[u] = first[u] = 0;
        q.push(u); e7 = false;
        while (!q.empty() && !e7) {
            int x = q.front(); q.pop();
            for (int i = 0; i < nadj[x]; i++) {
                int y = from[adj[x][i]];
                if (y == x) y = to[adj[x][i]];
                if (!mate[y] && y != u) {
                    mate[y] = x; R(x,y);
                    r++; e7 = true;
                    break;
                }
            }
            else if (OUTER(y)) L(x,y,adj[x][i]);
            else {
                int v = mate[y];
                if (!OUTER(v)) {
                    label[v] = x; first[v] = y;
                    q.push(v);
                }
            }
        }
        label[0] = -1;
    }
    return r;
}

```

```

/**** Exemplo simples de uso ****/
int main() {
    int f,t;
    while (scanf("%d %d",&n,&m) == 2 && (n || m)) {
        memset(nadj,0,sizeof(nadj));
        for (int i = 0; i < m; i++) {
            scanf("%d %d",&f,&t);
            f++; t++; //nao utilizar o vertice 0
            adj[f][nadj[f]++] = i;
            adj[t][nadj[t]++] = i;
            from[i] = f; to[i] = t;
        }
        printf("0 emparelhamento tem tamanho %d\n",E());
        for (int i = 1; i <= n; i++) {
            if (mate[i] > i) { //para nao imprimir 2 vezes
                printf("%d casa com %d\n",i-1,mate[i]-1);
            }
        }
    }
    return 0;
}

```

1.19 Floyd Warshall

Autor: Douglas Oliveira Santos
 Complexidade: $O(n^3)$
 Testes: nuevo-5785
 Descricao: Encontra o caminho
 mínimo entre todos os pares de vértices

```

#include <cstdio>
#include <cstring>
using namespace std;

```

```

typedef long long int64;

```

```

#define MAXN 150
#define INF 0x3f3f3f3f

```

```

/* FIILL ME */
int adj[MAXN][MAXN]; // matriz de adj com os custos

```

```

int d[MAXN][MAXN];
int pai[MAXN][MAXN]; // pai de j nos caminhos a partir de i */

```

```

void floyd(int n) {
    memset(d, 0x3f, sizeof(d));
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            if (adj[i][j] < INF) {
                d[i][j] = adj[i][j];
                pai[i][j] = i;
            }

    for (int k = 0; k < n; k++)
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                if (d[i][j] > d[i][k] + d[k][j]) {
                    d[i][j] = d[i][k] + d[k][j];
                    pai[i][j] = pai[k][j];
                }
}

```

```

    }
}

/**** Exemplo simples de uso ****/
int main() {
    int n;
    while (1) {
        scanf("%d", &n);
        if (n == 0) return 0;
        memset(adj, 0, sizeof(adj));
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                scanf("%d", &adj[i][j]);
            }
        }
        floyd(n);
    }
    return 0;
}

```

1.20 Fluxo Máximo de Custo Mínimo

Autor: Frank Chu, Igor Naverniouk, Igor Assis

Complexidade: $O(n^2 \cdot \text{flow} \cdot \text{cost})$

Tempo de implementacao: ?

Testes: uva.10594 uva.10806

Descricao: Fluxo maximo de custo minimo entre dois vertices s e t usando algoritmo de caminhos aumentantes minimos.

```

#include <cstring>
#include <climits>
#include <algorithm>

```

```
using namespace std;
```

```

// the maximum number of vertices + 1
#define MAXN 1024

```

```

/* FILL ME */
int cap[MAXN][MAXN];
int cost[MAXN][MAXN]; // cost per unit of flow matrix
int n, s, t;

```

```

// flow network and adjacency list
int x[MAXN][MAXN], adj[MAXN][MAXN], nadj[MAXN];

```

```

// Dijkstra's successor and depth
int par[MAXN], d[MAXN]; // par[source] = source;

```

```

// Labelling function
int pi[MAXN];

```

```
#define INF (0x3f3f3f3f)
```

```

// Dijkstra's using non-negative edge weights (cost+potential)
#define Pot(u,v) (d[u] + pi[u] - pi[v])
bool dijkstra(int n, int s, int t) {
    for (int i = 0; i < n; i++) d[i] = INF, par[i] = -1;
    d[s] = 0;

```

```

    par[s] = -n - 1;

    for (;;) {
        // find u with smallest d[u]
        int u = -1, bestD = INF;
        for (int i = 0; i < n; i++) if (par[i] < 0 && d[i] < bestD)
            bestD = d[u = i];

        if (bestD == INF) break;

        // relax edge (u,i) or (i,u) for all i;
        par[u] = -par[u] - 1;
        for (int i = 0; i < nadj[u]; i++) {
            // try undoing edge v->u
            int v = adj[u][i];
            if (par[v] >= 0) continue;
            if (x[v][u] && d[v] > Pot(u,v) - cost[v][u])
                d[v] = Pot(u, v) - cost[v][u], par[v] = -u-1;

            // try edge u->v
            if (x[u][v] < cap[u][v] && d[v] > Pot(u,v) + cost[u][v])
                d[v] = Pot(u,v) + cost[u][v], par[v] = -u - 1;
        }
    }

    for (int i = 0; i < n; i++) if (pi[i] < INF) pi[i] += d[i];

    return par[t] >= 0;
}
#undef Pot

int mfmc(int &fcost) {
    // build the adjacency list
    memset(nadj, 0, sizeof(nadj));
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            if (cap[i][j] || cap[j][i]) adj[i][nadj[i]++] = j;

    memset(x, 0, sizeof(x));
    memset(pi, 0, sizeof(pi));
    int flow = fcost = 0;

    // repeatedly, find a cheapest path from s to t
    while (dijkstra(n, s, t)) {
        // get the bottleneck capacity
        int bot = INT_MAX;
        for (int v = t, u = par[v]; v != s; u = par[v = u])
            bot = min(bot, x[v][u] ? x[v][u] : (cap[u][v] - x[u][v]));

        // update the flow network
        for (int v = t, u = par[v]; v != s; u = par[v = u])
            if (x[v][u] { x[v][u] -= bot; fcost -= bot * cost[v][u]; }
            else { x[u][v] += bot; fcost += bot * cost[u][v]; }

        flow += bot;
    }

    return flow;
}

```

```

/**** Exemplo simples de uso ****/

```

```

#include <iostream>
#include <stdio.h>

```

```
using namespace std;
```

```

/*
 * PARAMETERS:
 *   - cap (global): adjacency matrix where
 *       cap[u][v] is the capacity of the edge u->v.
 *       cap[u][v] is 0 for non-existent edges.
 *   - cost (global): a matrix where cost[u][v] is the cost
 *       per unit of flow along the edge u->v.
 *       If cap[u][v] == 0, cost[u][v] is
 *       ignored. ALL COSTS MUST BE NON-NEGATIVE!
 *   - n: the number of vertices
 *   - s: source vertex.
 *   - t: sink.
 * RETURNS:
 *   - the flow
 *   - the total cost through 'fcost'
 *   - fnet contains the flow network. Careful:
 *       both fnet[u][v] and fnet[v][u] could be positive.
 *       Take the difference.
 */

```

```

int main() {
    cin >> n;
    memset(cap, 0, sizeof(cap));

    int m, a, b, c, cp;
    cin >> m;
    cin >> s >> t;

    // fill up cap with existing capacities.
    // if the edge u->v has capacity 6, set cap[u][v] = 6.
    // for each cap[u][v] > 0, set cost[u][v] to the
    // cost per unit of flow along the edge u->v
    for (int i=0; i<m; i++) {
        cin >> a >> b >> cp >> c;
        cost[a][b] = c; // cost[b][a] = c;
        cap[a][b] = cp; // cap[b][a] = cp;
    }

    int fcost;
    int flow = mfmc(fcost);
    cout << "flow: " << flow << endl;
    cout << "cost: " << fcost << endl;

    return 0;
}

```

1.21 Fluxo Máximo de Custo Mínimo (Uso Geral) - Corte Mínimo

Autor: Codeforces / Marcelo Galvão Póvoa / Douglas Santos

Complexidade: $O(m \cdot \text{Flow})$, em média, $O(m \cdot n \cdot \text{Flow})$ pior caso

Testes: uva.10594, cf.277E, cf.164C, cf.78E, uva.10480

Descricao: Calcula fluxo usando caminhos aumentantes usando SPFA (um Bellman-Ford otimizado), suporta arestas múltiplas

e não direcionadas (usar `add()` 2x), usa lista de adjacências eficiente em um único vetor e sem STL. Para max-cost, use arestas de custo negativo (mas sem ciclo negativo). Se todos os custos são iguais, o algoritmo equivale ao Edmonds Karp. Se quiser obter o fluxo em cada aresta `i`, use `re[2*i+1]` e para obter o residual use `re[2*i]`. Aresta `i`: `ve[2*i+1] -> ve[2*i]`

```
#include <algorithm>
using namespace std;
```

```
#define N 201
#define M (2*1010) // dobro do número de arestas
#define INF 0x3f3f3f3f

int vt, ve[M], re[M], ze[M], next[M];
int in[N], head[N], path[N], dis[N], qu[N], lim[N];
```

```
void init() {
    vt = 1;
    memset(head, 0, sizeof(head));
}
```

```
void add(int x, int y, int cap, int wei = 0) {
    // aresta x->y é armazenada em [vt+1] e [vt+2]
    ve[++vt] = y; re[vt] = cap; ze[vt] = wei;
    next[vt] = head[x]; head[x] = vt;
    ve[++vt] = x; re[vt] = 0; ze[vt] = -wei;
    next[vt] = head[y]; head[y] = vt;
}
```

```
int mfmc(int s, int t, int n, int &fcost) {
    int flow = fcost = 0;
    while (1) {
        int qt = 0, k = 0;
        qu[qt++] = s;
        for (int i = 0; i < n; i++)
            dis[i] = lim[i] = INF;
        dis[s] = 0;
```

```
        while (k != qt) {
            if (k == N) k = 0;
            int x = qu[k++];
            for (int i = head[x]; i; i = next[i]) // ve[i]: adjs de x
                if (re[i] && dis[x] + ze[i] < dis[ve[i]]) {
                    dis[ve[i]] = dis[x] + ze[i];
                    path[ve[i]] = i;
                    lim[ve[i]] = min(lim[x], re[i]);
                    if (!in[ve[i]]) {
                        if (qt == N) qt = 0;
                        qu[qt++] = ve[i];
                        in[ve[i]] = 1;
                    }
                }
            in[x] = 0;
        }
    }
```

```
    if (dis[t] == INF) break;
    int f = lim[t];
    for (int p = t; p != s; p = ve[path[p] ^ 1]) {
        re[path[p]] -= f; re[path[p] ^ 1] += f; // novo residual
```

```
    }
    fcost += f * dis[t];
    flow += f;
}
return flow;
}
```

```
/** Código abaixo apenas para Min Cut **/
```

```
/* mark[v] = true, se v esta no mesmo lado de u no corte */
bool mark[N];
```

```
void DFS(int u) {
    mark[u] = true;
    for (int i = head[u]; i; i = next[i]) {
        if (i % 2) continue;
        int v = ve[i];
        if (!mark[v] && re[i] > 0) {
            DFS(v);
        }
    }
}
```

```
void mincut(int s, int t, int n) {
    memset(mark, 0, sizeof(mark));
    DFS(s);
    /* Arestas do corte */
    for (int i = 0; i < n; i++)
        for (int j = head[i]; j; j = next[j]) {
            if (j % 2) continue;
            int v = ve[j];
            if (mark[i] && !mark[v])
                printf("%d %d\n", i, v);
        }
}
```

```
/***** Exemplo simples de uso *****/
```

```
int main() {
    int n, m;
    while (scanf("%d %d", &n, &m) == 2) {
        int from, to, cp, co;
        init();
        for (int i = 0; i < m; i++) {
            scanf("%d %d %d", &from, &to, &cp, &co);
            add(from, to, cp, co);
        }
        int s=0, t=1;
        int fcost, flow=mfmc(s,t,n,fcost);
        printf("Fluxo: %d\nCusto: %d\n", flow, fcost);

        for (int i = 1; i <= m; i++) {
            printf("f(%d->%d) = %d\n", ve[2*i+1], ve[2*i], re[2*i+1]);
            printf("r(%d->%d) = %d\n", ve[2*i+1], ve[2*i], re[2*i]);
        }

        mincut(s, t, n);
    }
    return 0;
}
```

1.22 Fluxo Máximo Push-Relabel

Autor: Felipe Sodré, Igor Assis

Complexidade: $O(n^3)$

Tempo de implementacao: ?

Testes: uva.820 uva.10330 uva.10480

Descricao: Algoritmo para encontrar o fluxo máximo de `s` a `t`.

```
#include <cstdio>
```

```
#include <cstring>
#include <queue>
```

```
using namespace std;
```

```
#define MAXN 100
```

```
/* FILL ME */
```

```
int n, adj[MAXN][MAXN], nadj[MAXN], cap[MAXN][MAXN];
```

```
int x[MAXN][MAXN], r[MAXN][MAXN];
int e[MAXN], d[MAXN], s, t;
```

```
queue<int> Q;
```

```
#define adm(u, v) (d[u] == d[v] + 1)
```

```
void push(int u, int v, int c) {
    x[u][v] += c; x[v][u] -= c;
    r[u][v] -= c; r[v][u] += c;
    e[u] -= c; e[v] += c;
}
```

```
void preprocess() {
    memset(x, 0, sizeof(x));
    memset(e, 0, sizeof(e));
    memset(d, 0, sizeof(d));
    for (int i = 0; i < nadj[s]; i++) {
        int v = adj[s][i];
        push(s, v, cap[s][v]);
        if (v != s && v != t) Q.push(v);
    }
    d[s] = n;
}
```

```
void push_relabel(int u) {
    int j = -1;
    for (int i = 0; i < nadj[u]; i++) {
        int v = adj[u][i];
        if (e[u] <= 0) break;
        if (adm(u, v) && r[u][v] > 0) {
            int delta = min(e[u], r[u][v]);
            push(u, v, delta);
            if (e[v] > 0 && v != s && v != t) Q.push(v);
        }
        if (r[u][v] > 0 && (j == -1 || d[v] < d[j])) j = v;
    }
    if (e[u] > 0) {
        d[u] = d[j] + 1;
        Q.push(u);
    }
}
```

```

}

int maxflow() {
    int flow = 0;
    memcpy(r, cap, sizeof(r));
    preprocess();
    while (!Q.empty()) {
        int u = Q.front(); Q.pop();
        push_relabel(u);
    }
    for (int i = 0; i < nadj[s]; i++)
        flow += x[s][adj[s][i]];
    return flow;
}

/* funcoes para encontrar um s-t-corte minimo */
#define MAXM MAXN*MAXN

int mark[MAXN], cut[MAXM];

void dfs(int u) {
    mark[u] = 1;
    for (int i = 0; i < nadj[u]; i++)
        if (!mark[adj[u][i]] && r[u][adj[u][i]] > 0)
            dfs(adj[u][i]);
}

void mincut() {
    memset(mark, 0, sizeof(mark));
    dfs(s);
    for (int i = 0; i < n; i++)
        if (mark[i])
            for (int j = 0; j < nadj[i]; j++)
                if (!mark[adj[i][j]]) printf("%d %d\n", i, adj[i][j]);
}

/**** Exemplo simples de uso *****/
int main(void){
    return 0;
}

```

1.23 Fluxo Mínimo

Autor: Douglas Santos
Complexidade: $O(m \cdot \text{Flow})$
Testes: pku-3870

Descricao: Calcula o fluxo mínimo viável entre s e t , onde cada aresta e do grafo tem um capacidade mínima $lb[e]$ e máxima $ub[e]$, de forma que o fluxo: $lb[e] \leq f[e] \leq ub[e]$.

```

#include <algorithm>
#include <cstring>
using namespace std;

/* Fluxo Máximo SPFA AQUI */

/* FILL ME */
int lb[N][N], ub[N][N];

int f[N][N];

```

```

int minflow(int n, int s, int t) {
    lb[t][s] = 0;
    ub[t][s] = INF;
    init();
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            if (ub[i][j])
                add(i, j, ub[i][j] - lb[i][j], 0);

    for (int i = 0; i < n; i++) {
        int b = 0;
        for (int j = 0; j < n; j++) {
            b += lb[j][i];
            b -= lb[i][j];
        }
        if (b >= 0)
            add(n, i, b);
        else
            add(i, n+1, -b);
    }

    int fcost;
    mfmc(n, n+1, n+2, fcost);

    for (int i = head[n]; i; i = next[i]) {
        if (i & 1) continue;
        if (re[i] > 0) return -1;
    }

    for (int i = 0; i < n; i++)
        for (int j = head[i]; j; j = next[j]) {
            if (j & 1 || ve[j] != n+1) continue;
            if (re[j] > 0) return -1;
        }

    memcpy(f, lb, sizeof(lb));
    mfmc(t, s, n, fcost);
    int res = 0;
    for (int i = 0; i < n; i++)
        for (int j = head[i]; j; j = next[j]) {
            if (j & 1) continue;
            int v = ve[j];
            f[i][v] += re[j+1];
            if (i == s) res += f[s][v];
        }
    return res;
}

/**** Exemplo simples de uso *****/
#include <stdio>
int main() {
    int n, m;
    memset(lb, 0, sizeof(lb));
    memset(ub, 0, sizeof(ub));
    scanf("%d %d", &n, &m);
    for (int i = 0; i < m; i++) {
        int u, v, a, b;
        scanf("%d %d %d %d", &u, &v, &a, &b);
        lb[u][v] = a;
        ub[u][v] = b;
    }
}

```

```

}

int s = 0, t = n-1;
int res = minflow(s, t, n);

if (res == -1)
    printf("Fluxo Inviável\n");
else
    printf("Fluxo Mínimo Viável: %d\n", res);

return 0;
}

```

1.24 Intersecção de Matróides

Autor: Igor Assis
Complexidade: $O(|I| \cdot B(n, m))$, $|I|$ = tamanho interseccao
 $B(n, m)$ = complexidade da busca
Tempo de implementacao: ?
Testes: spojbr.HONESTID ($|I| = n$, $B(n, m) = O(m \log^* n)$)
Descricao: Encontra a interseccao de dois matroides.
Esta implementado o caso especifico de matroide floresta e matroide cor de aresta unica.

```

#include <stdio>
#include <string>
#include <queue>
#include <algorithm>

```

```
using namespace std;
```

```

#define MAXN 128
#define MAXM 128*128

```

```
#define MAXK 2*MAXN
```

```

/* FILL ME */
int n, m, ncor;
int orig[MAXM], dest[MAXM], firma[MAXM];

```

```

int p[MAXM], mark[MAXM], comp[MAXN], rank[MAXN];
int nX1, nX2, nY, nX_Y, cor[MAXK];

```

```

int X2[MAXM], X1[MAXM], inX2[MAXM], inX1[MAXM];
int Y[MAXM], X_Y[MAXM], inY[MAXM], inX_Y[MAXM];

```

```

int find(int u) {
    if (u == comp[u])
        return u;
    return comp[u] = find(comp[u]);
}

```

```

void unite(int u, int v) {
    if (rank[u] > rank[v])
        comp[v] = u;
    else {
        comp[u] = v;
        if (rank[u] == rank[v])
            rank[v]++;
    }
}

```

```

int caminho() {
    int i;
    queue<int>Q;

    memset(mark, 0, sizeof(mark));
    for (i = 0; i < nX1; i++) {
        p[X1[i]] = -1;
        mark[X1[i]] = 1;
        if (inX2[X1[i]] != -1)
            return X1[i];
        Q.push(X1[i]);
    }

    while (!Q.empty()) {
        int u = Q.front(); Q.pop();
        if (inX2[u] != -1)
            return u;
        if (inY[u] == -1) {
            /* monta ciclo em uma componente */
            if (find(orig[u]) == find(dest[u])) {

                for (i = 0; i < nY; i++)
                    if (!mark[Y[i]] &&
                        find(orig[Y[i]]) == find(orig[Y[u]])) {
                        p[Y[i]] = u;
                        mark[Y[i]] = 1;
                        if (inX2[Y[i]] != -1)
                            return Y[i];
                        Q.push(Y[i]);
                    }
            } else {
                for (i = 0; i < nY; i++)
                    if (!mark[Y[i]]) {
                        p[Y[i]] = u;
                        mark[Y[i]] = 1;
                        if (inX2[Y[i]] != -1)
                            return Y[i];
                        Q.push(Y[i]);
                    }
            }
        } else {
            for (i = 0; i < nX_Y; i++)
                if (!mark[X_Y[i]] &&
                    (firma[u] == firma[X_Y[i]] || !cor[firma[u]])) {
                    p[X_Y[i]] = u;
                    mark[X_Y[i]] = 1;
                    if (inX2[X_Y[i]] != -1)
                        return X_Y[i];
                    Q.push(X_Y[i]);
                }
        }
    }

    return -1;
}

int matroide() {
    int i, u, res;
    nX2 = nX1 = m; nY = 0;

```

```

    for (i = 0; i < m; i++) {
        inX2[i] = X2[i] = inX1[i] = X1[i] = i;
        inY[i] = -1;
    }
    for (i = 0; i < n; i++) {comp[i] = i; rank[i] = 1;}
    memset(cor, 0, sizeof(cor));

    res = 0;
    while ((u = caminho()) != -1) {
        while (u != -1) {
            /* ou-exclusivo */
            if (inY[u] == -1) {
                Y[nY] = u;
                inY[u] = nY++;
                X_Y[inX_Y[u]] = X_Y[--nX_Y];
                inX_Y[X_Y[u]] = inX_Y[u];
                cor[firma[u]] = 1; /* marca firma */
            } else {
                X_Y[nX_Y] = u;
                inX_Y[u] = nX_Y++;
                Y[inY[u]] = Y[--nY];
                inY[Y[nY]] = inY[u];
                cor[firma[u]] = 0; /* desmarca firma */
            }
            u = p[u];
        }
        /* atualiza componentes */
        for (i = 0; i < n; i++) {comp[i] = i; rank[i] = 1;}
        for (i = 0; i < nY; i++)
            unite(find(orig[Y[i]]), find(dest[Y[i]]));
        /* atualiza X2 e X1 */
        nX2 = nX1 = 0;
        memset(inX2, -1, sizeof(inX2));
        memset(inX1, -1, sizeof(inX1));
        for (i = 0; i < m; i++) {
            if (inY[i] == -1 && find(orig[i]) != find(dest[i])) {
                X2[nX2] = i;
                inX2[i] = nX2++;
            }
            if (inY[i] == -1 && !cor[firma[i]]) {
                X1[nX1] = i;
                inX1[i] = nX1++;
            }
        }
        res++;
    }
    return res;
}

/* Exemplo e' o problema HONESTID do spojbr */
int main() {
    int i, cases = 1;

    while (scanf("%d%d%d", &n, &m, &ncor) == 3) {
        for (i = 0; i < m; i++) {
            scanf("%d%d%d", &orig[i], &dest[i], &firma[i]);
            orig[i]--; dest[i]--;
        }
        printf("Instancia %d\n", cases++);
        if (matroide() == n-1)

```

```

        printf("sim\n\n");
        else printf("nao\n\n");
    }

    return 0;
}

```

1.25 Isomorfismo de Árvores

Autor: Douglas Santos
 Complexidade: $O(n \log n)$
 Testes: uva.12489
 Dependencias: Centro de Árvore
 Descricao: Encontra o isomorfismo entre duas árvores

```

#include <queue>
#include <cstring>
#include <algorithm>
#include <vector>
using namespace std;
#define MAXN 10010
#define INF 0x3f3f3f3f

/* FILL ME */
int n; // número de vértices
vector<int> adj[2][MAXN]; // listas de adj das árvores

```

```
/* Tree Center AQUI */
```

```

typedef struct No {
    int v, lb, pai;
    vector<int> chld, lchld;
} No;

```

```

bool comp(No a, No b) {
    return a.lchld < b.lchld;
}

```

```

vector<No> niveis[MAXN];
int d[MAXN];

```

```

int add(int h, int v, int pai) {
    No no;
    no.v = v;
    no.lb = 0;
    no.lchld.clear();
    no.pai = pai;
    no.chld.clear();
    niveis[h].push_back(no);
    return niveis[h].size() - 1;
}

```

```

int BFS(int r1, int r2, int a, int b) {
    int h = 0;
    queue<pair<int, int>> q;
    for (int i = 0; i < n; i++) niveis[i].clear();
    for (int i = 0; i < 2*n; i++) {
        d[i] = INF;
    }
    d[r1] = 0;

```

```

d[r2+n] = 0;
add(0, r1, -1);
add(0, r2, -1);
q.push(make_pair(r1, 0));
q.push(make_pair(r2+n, 1));
while (!q.empty()) {
    int u = q.front().first;
    int ind = q.front().second;
    int k;
    q.pop();

    h = max(h, d[u]);

    if (u < n) k = a;
    else {
        k = b;
        u = u - n;
    }

    for (int i = 0; i < adj[k][u].size(); i++) {
        int v = adj[k][u][i];
        v = v + min(k-a, 1) * n;
        if (d[v] > d[u + min(k-a, 1) * n] + 1) {
            d[v] = d[u + min(k-a, 1) * n] + 1;
            No no;
            int tam;
            tam = add(d[v], v, ind);
            q.push(make_pair(v, tam));
        }
    }
}
return h;
}

bool isoR(int r1, int r2, int a, int b) {
    int h = BFS(r1, r2, a, b);

    for (int i = h-1; i >= 0; i--) {
        for (int j = 0; j < niveis[i+1].size(); j++) {
            No v = niveis[i+1][j];

            niveis[i][v.pai].lchld.push_back(v.lb);
            niveis[i][v.pai].chld.push_back(j);
        }
        sort(niveis[i].begin(), niveis[i].end(), comp);
        niveis[i][0].lb = 0;

        for (int j = 1; j < niveis[i].size(); j++)
            if (niveis[i][j].lchld == niveis[i][j-1].lchld) {
                niveis[i][j].lb = niveis[i][j-1].lb;
            }
            else niveis[i][j].lb = niveis[i][j-1].lb + 1;
    }
    if (niveis[0][0].lb == niveis[0][1].lb) return true;
    return false;
}

/* Retorna true se existe um isomorfismo entre
as arvores de índice a e b, a < b*/
bool isoTree(int a, int b) {

```

```

pair<int, int> c1, c2;
c1 = cTree(n, adj[a]);
c2 = cTree(n, adj[b]);
if (isoR(c1.first, c2.first, a, b)) return true;
if (c2.second != -1)
    return isoR(c1.first, c2.second, a, b);
return false;
}

/**** Exemplo simples de uso ****/
int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        adj[0][i].clear();
        adj[1][i].clear();
    }
    int u, v;
    for (int k = 0; k < 2; k++) {
        for (int i = 0; i < n-1; i++) {
            scanf("%d %d", &u, &v);
            u--; v--;
            adj[k][u].push_back(v);
            adj[k][v].push_back(u);
        }
    }
    if (isoTree(0, 1)) printf("S\n");
    else printf("N\n");
    return 0;
}

```

1.26 Menor Ancestral Comum (LCA)

Autor: Igor Assis / Marcelo Galvão Póvoa
Complexidade: $O(n \log n) + O(1)$ por query
Testes: nuevo.2045
Dependencias: Range Minimum Query
Descricao: Dada uma arvore preprocessa de forma a realizar queries da forma LCA(u,v) que retornam o menor ancestral (mais longe da raiz) comum de u e v na arvore.

```

#include <cstring>
#include <vector>

using namespace std;

#define N (2*1024) // usar o dobro do limite

/* FILL ME */
int adj[N][N], nadj[N];

int nE, nL, E[N], L[N], R[N], vis[N];

void euler(int u, int el) {
    E[nE++] = u; L[nL++] = el;
    vis[u] = 1;
    for (int i = 0; i < nadj[u]; i++)
        if (!vis[adj[u][i]]) {
            euler(adj[u][i], el+1);
            E[nE++] = u; L[nL++] = el;
        }
}

```

```

}
}

void preprocess(int root, int n) {
    int i;
    nE = nL = 0;
    memset(vis, 0, sizeof(vis));
    euler(root, 0);
    for (i = 2*n-2; i >= 0; i--) R[E[i]] = i;
    init(L, nL);
}

int lca(int u, int v) {
    return E[query(min(R[u], R[v]), max(R[u], R[v]))];
}

#include <cstdio>

/**** Exemplo simples de uso ****/
int main(){
    int i, u, v, n;
    scanf("%d", &n);
    memset(nadj, 0, sizeof(nadj));
    for (i = 0; i < n-1; i++) {
        scanf("%d%d", &u, &v);
        adj[u][nadj[u]++] = v;
        adj[v][nadj[v]++] = u;
    }

    preprocess(0, n);
    printf("%d\n", lca(2, 3));

    return 0;
}

```

1.27 Pontes, Pontos de Articulação e Componentes Biconexas

Autor: Igor Assis / Douglas Santos
Complexidade: $O(n+m)$
Testes: uva.796 spojbr.TUBOS
Descricao: Encontra as pontes, os pontos de articulacao e as componentes biconexas (comecando em 1) em um grafo não direcionado. Não permite arestas múltiplas.

```

#include <cstring>
#include <stack>
#include <algorithm>
#include <vector>
using namespace std;

#define MAXN 1024
#define MAXM 1024*1024

#define VIZ(u, i) (orig[inc[u][i]] != (u) ? \
                    orig[inc[u][i]] : dest[inc[u][i]])

/* Input - FILL ME */
/* aresta i = (u, v); orig[i] = u, dest[i] = v,
inc[u].push_back(i), inc[v].push_back(i) */

```

```

vector<int> inc[MAXN];
int orig[MAXM], dest[MAXM];

/* Output */
int ponte[MAXM]; // ponte[i] -> aresta i é ponte
int part[MAXN]; // part[u] -> vertice u é de articulação
int ncomp; // numero de componentes biconexas
int comp[MAXM]; // comp[i] = componente da aresta i

/* Variaveis auxiliares */
int low[MAXN], vis[MAXN], dt;
stack<int> stck;

/* Função auxiliar */
int dfsbcc(int u, int p) {
    int ch = 0;
    vis[u] = dt++;
    low[u] = vis[u];
    for (int i = 0; i < (int) inc[u].size(); i++) {
        int e = inc[u][i], v = VIZ(u, i);
        if (!vis[v]) {
            stck.push(e);
            dfsbcc(v, u); ch++;
            low[u] = min(low[u], low[v]);
            if (low[v] >= vis[u]) {
                part[u] = 1;
                ncomp++;
                while (stck.top() != e) {
                    comp[stck.top()] = ncomp;
                    stck.pop();
                }
                comp[stck.top()] = ncomp; stck.pop();
            }
            if (low[v] == vis[v]) ponte[e] = 1;
        } else if (v != p) {
            if (vis[v] < vis[u]) stck.push(e);
            low[u] = min(low[u], vis[v]);
        }
    }
    return ch;
}

void bcc(int n) {
    memset(low, 0, sizeof(low));
    memset(vis, 0, sizeof(vis));
    memset(part, 0, sizeof(part));
    memset(ponte, 0, sizeof(ponte));
    memset(comp, 0, sizeof(comp));
    dt = 1;
    ncomp = 0;
    for (int i = 0; i < n; i++)
        if (!vis[i])
            part[i] = dfsbcc(i, -1) >= 2;
}

#include <stdio.h>

int main(){
    int i;
    int n, m;

```

```

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

for (int i = 0; i < m; i++) inc[i].clear();
for (i = 0; i < m; i++) {
    int u, v;
    scanf("%d%d", &u, &v);
    orig[i] = u; dest[i] = v;
    inc[u].push_back(i);
    inc[v].push_back(i);
}

bcc(n);

printf("Pontos de Articulacao:");
for (i = 0; i < n; i++)
    if (part[i]) printf(" %d", i);
printf("\n");

printf("Pontes:");
for (i = 0; i < m; i++)
    if (ponte[i])
        printf(" (%d %d)", orig[i], dest[i]);
printf("\n");

printf("Componentes:\n");
for (i = 0; i < m; i++)
    printf("comp[%d] = %d\n", i, comp[i]);
printf("\n");

return 0;
}

1.28 Stable Marriage

Autor: Igor Naverniuk, Igor Assis
Complexidade:  $O(m^2)$ , m = numero de homens
Tempo de implementacao: ?
Testes: uva.11119
Descricao:
Takes a set of m men and n women, where each person has
an integer preference for each of the persons of opposite
sex. Produces a matching of each man to some woman.
The matching will have the following properties:
- Each man is assigned a different woman
(n must be at least m).
- No two couples M1W1 and M2W2 will be unstable.
- The solution is man-optimal.
Two couples are unstable if
- M1 prefers W2 over W1 and
- W1 prefers M2 over M1.

#include <cstring>

#define MAXM 1024
#define MAXN 1024

int m, n; // number of men and women, n>=m
// the list of women in order of decreasing preference (man i)
int L[MAXM][MAXN];
int R[MAXN][MAXM]; // the attractiveness of man i to woman j

```

```

int L2R[MAXM]; // the mate of man i (always between 0 and n-1)
int R2L[MAXN]; // the mate of woman j (or -1 if single)

int p[MAXM];

void stableMarriage() {
    memset(R2L, -1, sizeof(R2L));
    memset(p, 0, sizeof(p));

    // Each man proposes...
    for (int i = 0; i < m; i++) {
        int man = i;
        while (man >= 0) {
            // to the next woman on his list in order
            // of decreasing preference, until one of them accepts;
            int wom;
            while (1) {
                wom = L[man][p[man]++];
                if (R2L[wom] < 0 || R[wom][man] > R[wom][R2L[wom]])
                    break;
            }

            // Remember the old husband of wom.
            int hubby = R2L[wom];

            // Marry man and wom.
            R2L[L2R[man] = wom] = man;

            // If a guy was dumped in the process, remarry him now.
            man = hubby;
        }
    }
}

```

1.29 Topological Sort

```

Autor: Alexandre Kunieda
Complexidade:  $O(E + V)$ 
Tempo de implementacao: ?
Testes: uva-10350 auxiliar para shortdag, spojbr-ORKUT
Dependencias: Nenhuma
Descricao: Ordena Topologicamente, ou verifica que não há
ordenação topológica. Para verificação de existência da
ordenação, implemente os trechos comentados do código;
além disso, as duas funções podem ser declaradas como void.

int n;
int adj[MAXN][MAXN]; /* lista de adj */
int nadj[MAXN]; /* grau de cada vertice */

int foi[MAXN], ip; /* auxiliar */
/* int foi2[MAXN]; */
int tops[MAXN]; /* resposta */

int DFS(int k) {
    int i, j;

    foi[k] = /* foi2[k] = */ 1;
    for (j = 0; j < nadj[k]; j++) {
        i = adj[k][j];
    }
}

```



```

    /* if(foi2[i]) return 0; */
    if(!foi[i] && !DFS(i)) return 0;
}

tops[--ip] = k;
/* foi2[k] = 0; */

return 1;
}

/*
popular n: numero de vertices
apos chamar ord_top() "tops" tera a solucao
*/
int ord_top() {
    memset(foi, 0, n*sizeof(int));
    /* memset(foi2, 0, n*sizeof(int)); */
    ip = n;

    for(int i=0 ; i<n ; i++)
        if(!foi[i] && !DFS(i)) return 0;

    return 1;
}

/**** Exemplo simples de uso ****/
int main() {
    int m,i, from,to;
    while (scanf("%d %d",&n,&m) == 2 && n != 0) {
        memset(nadj,0,sizeof(nadj));
        for (i = 0; i < m; i++) {
            scanf("%d %d",&from,&to);
            adj[from][nadj[from]++] = to;
        }
        if(!ord_top()) puts("não há ordenação topológica");
        else for (i = 0; i < n; i++) printf("%d ",tops[i]);
    }
    return 0;
}

```

1.30 Two Satisfiability

Autor: Davi Costa
 Complexidade: $O(E + V)$
 Tempo de implementacao: ?
 Testes: spoj.CARDAPI0 nuevo-2886
 Dependencias: SCC
 Descricao:
 Determina se existe uma atribuicao
 que satisfaca a expressao $(X_i \vee X_k) \wedge (X_j \vee !X_l) \dots$
 Para cada clausula deve haver uma aresta
 no grafo da forma $!X_i \rightarrow X_k$ e $!X_k \rightarrow X_i$
 A funcao "clau" gera as arestas automaticamente
 dada a clausula, ver exemplo.

```

#define N(x) (2*x + 1)
#define Y(x) (2*x)
#define NEG(x) (x%2 == 1 ? x-1 : x+1)

```

/*n deve ser o numero total de literais p (nao 2*p)*/

```

bool two_sat(int n) {
    n *= 2;
    bool ok = true;
    scc(n);
    for (int i=0; i<n/2 && ok; i++)
        ok &= (comp[2*i] != comp[2*i+1]);
    return ok;
}

/*Solucao da literal x apos rodar two_sat() == true */
int getsol(int x) {
    return comp[2*x] < comp[2*x+1];
}

/*Insira a clausula como descrito na main*/
void clau(int x, int y) {
    int negx = NEG(x), negy = NEG(y);
    adj[negx][nadj[negx]++] = y;
    adj[negy][nadj[negy]++] = x;
}

/**** Exemplo simples de uso ****/
int main() {
    /* Nunca use N e Y recursivamente, se precisar
    use NEG apos a primeira negacao ou afirmacao */
    /* Exemplo de uso, cada pessoa entra com x, y, w, z,
    sendo que x ou y deve ser atendido e w ou z nao pode */
    int k, n;
    memset(nadj,0,sizeof(nadj));
    scanf("%d %d",&k,&n);
    for (int i = 0; i < k; i++) {
        int x,y,w,z;
        scanf("%d %d %d %d",&x,&y,&w,&z);
        clau(Y(x),Y(y)); // x ou y
        clau(N(w),N(z)); // nao(w) ou nao(z)
    }
    printf("%s\n",two_sat(n)?"yes":"no");
}

```

1.31 Union Find e Árvore Geradora Mínima (Kruskal)

Autor: Douglas Santos / topcoder forum
 Complexidade: Union-Find: $O(1)$, Kruskal: $O(m \lg n)$
 Descricao: Estrutura de dados union find e uma aplicacao
 para encontrar a árvore geradora mínima

```

#include <algorithm>
using namespace std;

#define MAXN 1010

int id[MAXN], sz[MAXN]; //uf auxiliar

void ufini(int n) {
    for (int i = 0; i < n; i++)
        id[i] = i, sz[i] = 1;
}

int uffind(int i) {

```

```

    if (i == id[i]) return i;
    return id[i] = uffind(id[i]);
}

void ufuni(int v, int w) {
    v = uffind(v); w = uffind(w);
    if (v == w) return;
    if (sz[v] > sz[w]) swap(v,w);
    id[v] = w;
    if (sz[v] == sz[w]) sz[w]++;
}

```

```

/* MST - Kruskal a partir daqui */
#define MAXM 10010

```

```

/* FILL ME */
struct edge {
    int u, v, w;
    int ind;
} ed[MAXN];

```

```

bool comp(edge a, edge b) {
    return a.w < b.w;
}

```

```

/* Para cada aresta i,
diz se está ou não na árvore */
bool used[MAXN];

```

```

int kruskal(int n, int m) {
    sort(ed, ed+m, comp);
    ufini(n);
    int res = 0;
    for (int i = 0; i < m; i++) {
        int u = uffind(ed[i].u);
        int v = uffind(ed[i].v);
        if (u == v) {
            used[ed[i].ind] = false;
            continue;
        }
        used[ed[i].ind] = true;
        ufuni(u, v);
        res += ed[i].w;
    }
    return res;
}

```

2 Programação Dinâmica

2.1 Longest Common Subsequence (LCS)

Autor: Davi Costa/Marcelo Galvão Póvoa
 Complexidade: $O(n*m)$ -calcula e $O(n+m)$ -reconstrucao
 Tempo de implementacao: 5 min
 Testes: UVA.10066, UVA.10405
 Descricao: Calcula o tamanho de uma LCS entre duas strings e
 reconstrói uma de tamanho qualquer (nao maior que
 o maximo).

```

#include <stdio.h>

```

```
#include <string.h>
#define MAX 1234

/*Primeira e segunda strings (tamanhos m e n)*/
char seq[2][MAX+1];
int pd[MAX+1][MAX+1];

/*guarda o caminho*/
enum { cima, lado, diag } way[MAX+1][MAX+1];

int lcs(int m, int n) {
    int i,j;
    for (i = 0; i <= m; i++) pd[i][0] = 0;
    for (i = 0; i <= n; i++) pd[0][i] = 0;

    for (i = 1; i <= m; i++)
        for (j = 1; j <= n; j++) {
            if (seq[0][i-1] == seq[1][j-1]) {
                pd[i][j] = pd[i-1][j-1] + 1;
                way[i][j] = diag;
            }
            else if (pd[i-1][j] > pd[i][j-1]) {
                pd[i][j] = pd[i-1][j];
                way[i][j] = cima;
            }
            else {
                pd[i][j] = pd[i][j-1];
                way[i][j] = lado;
            }
        }
    return pd[m][n];
}

/*reconstroi uma CS, deve ser chamada com (m-1,n-1,tam CS)*/
void printway(int i, int j, int k) {
    if (i==0 || j==0 || k==0) printf("\n");
    else if (way[i][j] == diag) {
        printway(i-1, j-1, k-1);
        printf("%c",seq[0][i]);
    }
    else if (way[i][j]==cima) printway(i-1, j, k-1);
    else printway(i, j-1, k-1);
}

/***** Exemplo simples de uso *****/
int main(){
    int n,m;
    while (1) {
        scanf(" %s",seq[0]);
        scanf(" %s",seq[1]);
        m=strlen(seq[0]);
        n=strlen(seq[1]);

        printf("%d\n",lcs(m,n));
    }
    return 0;
}
```

2.2 Longest Increasing Subsequence (LIS)

Autor: Marcelo Galvão Póvoa
 Complexidade: $O(n \lg k)$, sendo k o tamanho da LIS
 Testes: UVA.231
 Descricao: Determina o tamanho da LIS do vetor v , que pode ter numeros negativos, inclusive. Os trechos de codigo comentados são relativos apenas a parte de reconstrucao de uma LIS. Esse algoritmo so funciona quando a relacao entre dois elementos eh transitiva ($a < b$ e $b < c \Rightarrow a < c$), como acontece com numeros, strings, etc.

```
#include <stdio.h>
#include <string.h>
#define MAXN 1000
#define INF 0x3f3f3f3f

int v[MAXN+1] /*,ant[MAXN+1],li[MAXN+1]*/ ;
int pd[MAXN+1] /*,ipd[MAXN+1]*/ ;
/*pd armazena o menor elemento que lide-
ra uma IS de tamanho i ate o momento*/

int lis(int n) {
    int es,di,m,mx=0;

    memset(pd,0x3f,sizeof(pd));
    pd[0]=-INF;

    for (int i=0;i<n;i++) {
        es=0; di=i;
        while (es<di) {
            m=(es+di+1)/2;
            if (pd[m]<v[i]) es=m;
            else di=m-1;
        }

        if (pd[es]<v[i] && pd[es+1]>v[i]) {
            pd[es+1]=v[i];
            if (es+1>mx) mx=es+1;
            /* ipd[es+1]=i;
            ant[i]=ipd[es];*/
        }
    }
    return mx;
}

/*reconstroi uma IS de tamanho tam depois de chamar lis(n)*/
/*void build(int tam) {
    int p=ipd[tam];

    if (pd[tam]==INF) printf("-1\n");
    else if (tam>0) {
        for (int i=0;i<tam;i++) {
            li[i]=v[p];
            p=ant[p];
        }

        for (int i=tam-1;i>0;i--) printf("%d ",li[i]);
        printf("%d\n",li[0]);
    }
}
```

```
else printf("\n");
}*/

/***** Exemplo simples de uso *****/
int main() {
    int n,i,k;

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

    k=lis(n);
    printf("%d\n",k);
    /*build(k);*/

    return 0;
}
```

2.3 Mochila binária (knapsack)

Autor: Micael Carvalho
 Complexidade: $O(nW)$
 Teste: CF.248A (433/A)
 Descrição: Encontra uma possível solução para o problema da mochila binária.

```
#include <bits/stdc++.h>
using namespace std;

int knapsack(int W, int wt[], int val[], int n) {
    int i, w;
    int K[n+1][W+1];
    for (i = 0; i <= n; i++) {
        for (w = 0; w <= W; w++) {
            if (i==0 || w==0)
                K[i][w] = 0;
            else if (wt[i-1] <= w)
                K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]],
                               K[i-1][w]);
            else
                K[i][w] = K[i-1][w];
        }
    }
    return K[n][W];
}

int main() {
    int n, i, w[100], sum;
    scanf("%d", &n);
    sum = 0;
    for(i = 0; i < n; ++i) {
        scanf("%d", &w[i]);
        sum += w[i];
    }
    sum /= 2;
    i = knapsack(sum, w, w, n); // peso = valor
    if(i == sum) { // é possível separar em 2 grupos?
        printf("YES\n");
    } else {
        printf("NO\n");
    }
}
```

```
    return 0;
}
```

3 Geométricos

3.1 Algoritmos Básicos para Circunferência

Autor: Douglas Santos

Testes: poj-3831

Descricao: Calcula interseção entre duas circunferência, e a área dessa interseção.

Dependências: ccw

```
/**
Estrutura de ponto, circunferencia e reta aqui
**/
```

```
#include <cmath>
#include <algorithm>
#define F first
#define S second
using namespace std;
```

```
double sqr(double x) { return x*x; }
```

```
/* retorna 0, 1 ou 2 intersecções entre dois círculos e
 * se houver 1, em ia; se houver 2, em ia e ib */
int inter_circ(pt &ia, pt &ib, circ c1, circ c2) {
    int c = cmp(norma(c1.F-c2.F), c1.S+c2.S);
    if (c > 0) return 0;
```

```
    double dx=c2.F.x-c1.F.x;
    double dy=c2.F.y-c1.F.y;
    double d=sqr(dx)+sqr(dy);
    double r=sqrt((sqr(c1.S+c2.S)-d)*(d-sqr(c2.S-c1.S)));
```

```
    ia.x=ib.x+0.5*((c2.F.x+c1.F.x)+dx*(sqr(c1.S)-sqr(c2.S))/d);
    ia.y=ib.y+0.5*((c2.F.y+c1.F.y)+dy*(sqr(c1.S)-sqr(c2.S))/d);
    ia.x+=dy*r/(2*d); ib.x-=dy*r/(2*d);
    ia.y-=dx*r/(2*d); ib.y+=dx*r/(2*d);
    return 1-c;
}
```

```
/* Calcula a menor área quando o círculo é dividido pela
corda dos pontos (a, b) */
double area_corda(circ c, pt a, pt b) {
    double d = norma(a - b);
    double r = c.S;
    double h = sqrt(r*r - (d*d)/4.0);
    double at = (d*h) / 2.0;
    double ang = 2 * acos(h / r);
    double ac = (ang * r * r) / 2.0;
    return ac - at;
}
```

```
/* Calcula a area da intersecção entre dois círculos */
double area_inter(circ c1, circ c2) {
    if (c1.S > c2.S) swap(c1, c2);
    c2.F.x = norma(c1.F - c2.F); c2.F.y = 0;
    c1.F.x = 0; c1.F.y = 0;
```

```
    pt ia, ib;
    int it = inter_circ(ia, ib, c1, c2);
    if (it == 1) return 0.0;
    if (it == 0) {
        if (cmp(c2.F.x, c1.S + c2.S) > 0) return 0.0;
        else return pi * c1.S * c1.S;
    }
}
```

```
double a1 = area_corda(c1, ia, ib);
double a2 = area_corda(c2, ia, ib);
if (ccw(ia, ib, c1.F) == ccw(ia, ib, c2.F)) {
    return a2 + pi * c1.S * c1.S - a1;
}
return a1 + a2;
}
```

```
/* Exemplo simples de uso */
int main() {
    circ c1, c2;
    pt ia, ib;
    int res;

    c1 = circ(pt(0, 0), 1);
    c2 = circ(pt(0, 2), 2);
    res = inter_circ(ia, ib, c1, c2);
    printf("%d (%lf %lf) (%lf %lf) (%lf\n", res, ia.x, ia.y,
        ib.x, ib.y, area_inter(c1, c2));
```

```
    return 0;
}
```

3.2 Algoritmos Básicos para Geométricos

Autor: Alexandre Kunieda (+ PUC-Rio)

Tempo de implementacao: 2 minutos

Descricao: Contem algoritmos simples para geometricos

```
/**
Estrutura de ponto e poligono aqui
**/
```

```
double polyarea(poly& p){ /* area com sinal */
    int i, n=p.size();
    double area = 0.0;
```

```
    for(i=0; i<n; i++){
        area += p[i]*p[(i+1)%n];
    }
```

```
    return area/2.0; /* area>0 = ccw ; area<0 = cw */
}
```

```
/* ponto p entre segmento [qr] */
int between3(pt p, pt q, pt r){
    if(cmp((q-p)*(r-p)) == 0) /* colinear */
        if(cmp((q-p)*(r-p)) <= 0) /* < para nao contar extremos */
            return 1;
```

```
    return 0;
}
```

```
/* rotaciona pt p em ang radianos, em torno do ponto q
se q nao especificado, rotaciona em torno da origem */
pt rotate(pt p, double ang, pt q = pt(0,0)) {
    double s = sin(ang), c = cos(ang);
    p = p-q;
    return q + pt( p.x*c - p.y*s, p.x*s + p.y*c );
}
```

```
/***** Exemplo simples de uso *****/
```

```
int main() {
    pt v,w;

    while(scanf("%lf %lf", &v.x,&v.y)==2){
        w = rotate(v,pi/4);
        printf("%lf %lf\n", w.x,w.y);

        w = rotate(v,pi/4, pt(1,1));
        printf("%lf %lf\n", w.x,w.y);
    }
```

```
    return 0;
}
```

3.3 Algoritmos de Intersecções

Autor: Alexandre Kunieda + (PUC-Rio)

Testes:

- UVa 11068 [intersect] [acha] t=0.010s
- UVa 866 [intersect_seg] [intersect_seg_2] [acha] t=0.000s
- UVa 378 [intersect] [acha] t=0.010s
- UVa 191 [intersect_seg] [intersect_seg_2] t=0.000s
- POJ 3819 [inter_reta_circ]

Dependências:

- comparacoes na estrutura de ponto - soh intersect_seg()
- norma() - distPR(), inter_reta_circ()
- projecao() - distPR(), inter_reta_circ()
- between3() - distPR(), intersect_seg_2(), inter_reta_circ()
- ccw() - soh intersect_seg_2()

Descricao: Determina se há interseção ou o ponto de interseção entre segmentos de reta ou retas. Acha intersecções entre segmento de reta ou reta e circunferência. Também contém função que devolve a distância de um ponto a uma reta.

```
/**
Estruturas aqui
**/
```

```
int intersect(reta p0, reta q0){ /*intersecção de retas*/
    eq_reta p(p0), q(q0);
```

```
    if(cmp(p.A*q.B , p.B*q.A)==0){ /*paralelos*/
        if(cmp(p.A*q.C , p.C*q.A)==0 &&
            cmp(p.B*q.C , p.C*q.B)==0) return 2; /*reta*/
        else return 0; /*nada*/
    }
    return 1; /*ponto*/
}
```

```
/* intersecção nos extremos dos segmentos tbm é contada! */
```

```

bool intersect_seg(pt p, pt q, pt r, pt s) {
    pt A = q - p, B = s - r, C = r - p, D = s - q;
    int a = cmp(A % C) + 2 * cmp(A % D);
    int b = cmp(B % C) + 2 * cmp(B % D);
    if (a == 3 || a == -3 || b == 3 || b == -3) return false;
    if (a || b || p==r || p==s || q==r || q==s) return true;
    int t = (p < r) + (p < s) + (q < r) + (q < s);
    return t != 0 && t != 4;
}

bool intersect_seg_2(pt p, pt q, pt r, pt s) {
    int a = ccw(p,q,r)*ccw(p,q,s);
    int b = ccw(r,s,p)*ccw(r,s,q);

    if(a<0 && b<0) return true;
    else return false;

    // tire o 'else' para verificar intersecção nos extremos
    if(a>0 || b>0) return false;

    return ( between3(p,r,s) ||
             between3(q,r,s) ||
             between3(r,p,q) ||
             between3(s,p,q) );
}

/*acha intersecção de duas retas*/
pt acha(pt a, pt b, pt c, pt d){
    /* pressupoe que haja intersecção! */
    eq_reta p(reta(a,b)), q(reta(c,d));
    pt k;

    k.x = (q.C*p.B - p.C*q.B)/(p.A*q.B - q.A*p.B);
    k.y = (q.C*p.A - p.C*q.A)/(p.B*q.A - q.B*p.A);
    return k;
}

/*acha intersecção de duas retas - da PUC*/
pt acha_(pt p, pt q, pt r, pt s){
    pt a = q-p, b = s-r, c = pt(p%q,r%s);
    return pt(pt(a.x, b.x)%c, pt(a.y, b.y)%c) / (a%b);
}

/* distância de um ponto a uma reta */
double distPR(pt p, reta r){
    pt v = p - r.ini;
    pt w = r.fim - r.ini;

    pt proj = projecao(v,w);
    /* (proj+r.ini) é o ponto mais proximo de p,
       e que pertence à reta r */

    /* para segmentos de reta
       *
       * if( !between3(proj+r.ini, r.ini, r.fim) )
       * return min( norma(p-r.ini), norma(p-r.fim) );
       */

    return norma(v - proj);
}

```

```

/* retorna 0, 1 ou 2 intersecções entre segmento/reta e
 * círculo: se houver 1, em ia; se houver 2, em ia e ib */
int inter_reta_circ(pt &ia, pt &ib, reta r, circ c) {
    pt p = r.ini + projecao(c.first - r.ini, r.fim - r.ini);
    double d = norma(p - c.first);

    if (cmp(d, c.second) > 0) return 0;

    pt v = cmp(norma(r.ini - p)) ? r.ini : r.fim;
    v = versor(v - p) * sqrt(max(0.0,c.second*c.second - d*d));

    ia = p + v; ib = p - v;

    /* para segmentos de reta, descomente
       * int ba = between3(ia, r.ini, r.fim);
       * int bb = between3(ib, r.ini, r.fim);
       * if (!ba) {
       *     ia = ib;
       *     return bb;
       * }
       */
    return (cmp(norma(ia - ib))* && bb*) + 1;
}

/***** Exemplo simples de uso *****/
int main(){
    /***** Especifico do problema UVa 378 *****/
    int n, i,aux;
    reta r[2];
    pt p;

    puts("INTERSECTING LINES OUTPUT");

    scanf(" %d", &n);
    while(n--){
        for(i=0 ; i<2 ; i++)
            scanf(" %lf %lf %lf %lf",
                  &r[i].ini.x, &r[i].ini.y,
                  &r[i].fim.x, &r[i].fim.y);

        aux = intersect(r[0], r[1]);

        if(aux == 0) puts("NONE");
        if(aux == 1){
            p = acha(r[0].ini,r[0].fim, r[1].ini,r[1].fim);
            printf("POINT %.2lf %.2lf\n", p.x,p.y);
        }
        if(aux == 2) puts("LINE");
    }

    puts("END OF OUTPUT");

    return 0;
}

```

3.4 Círculo Gerador Mínimo

Autor: PUC-Rio
Complexidade: $O(n^3)$

Tempo de implementacao: 3 min
Testes:
- SPOJbr ICPC (n<=100) t=0.35s
- UVa 10005 (n<=100) t=0.002s
Dependencias:
- norma()
Descricao: O algoritmo devolve o circulo de raio minimo que
contem todos os pontos dados

```

bool in_circle(circ C, pt p){
    return cmp(norma(p - C.first), C.second) <= 0;
}

pt circumcenter(pt p, pt q, pt r) {
    pt a = p - r, b = q - r,
        c = pt(a * (p + r) / 2, b * (q + r) / 2);

    return pt(c % pt(a.y, b.y),pt(a.x, b.x)%c) / (a%b);
}

circ spanning_circle(vector<pt>& T) {
    int n = T.size();

    circ C(pt(), -INFINITY);
    for (int i = 0; i < n; i++) if (!in_circle(C, T[i])) {
        C = circ(T[i], 0);
        for (int j = 0; j < i; j++) if (!in_circle(C, T[j])) {
            C = circ((T[i] + T[j]) / 2, norma(T[i] - T[j]) / 2);
            for (int k = 0; k < j; k++) if (!in_circle(C, T[k])) {
                pt o = circumcenter(T[i], T[j], T[k]);
                C = circ(o, norma(o - T[k]));
            }
        }
    }

    return C;
}

/**
Estrutura de ponto e poligono aqui
**/

```

3.5 Convex Hull (Graham Scan)

Autor: Alexandre Kunieda (+ PUC-Rio)
Complexidade: $O(n \lg n)$
Testes: uva.218, uva.596, uva.10065, uva.11096, nuevo.3655
Dependencias:
- norma()
- ccw()
Descricao: Algoritmo de Graham, para obter o Convex Hull de um
dado conjunto de pontos

```

/**
Estrutura de ponto e poligono aqui
**/

pt pivo;
/* ordena em sentido horario */
bool cmp_radial(pt a, pt b){
    int aux = ccw(pivo, a,b);
    return ((aux<0) || (aux==0 && norma(a-pivo)<norma(b-pivo)));
}

bool cmp_pivo(pt p, pt q){ /* pega o de menor x e y */

```

```

    int aux = cmp( p.x, q.x );
    return ((aux<0) || (aux==0 && cmp( p.y, q.y )<0));
}
/* usar poly& p reduz tempo, mas desordena o conj de pontos */
poly graham(poly p){
    int i,j,n = p.size();
    poly g;

    /* ordena e torna o conj de pontos um poligono estrelado */
    pivo = *min_element(p.begin(), p.end(), cmp_pivo);
    sort(p.begin(), p.end(), cmp_radial);

    /* para pegar colineares do final do poligono
    *
    * for(i=n-2 ; i>=0 && ccw(p[0], p[i], p[n-1])==0 ; i--);
    * reverse(p.begin()+i+1, p.end());
    */

    for(i=j=0 ; i<n ; i++) {
        /* trocar ccw>=0 por ccw>0 para pegar colineares */
        while( j>=2 && ccw(g[j-2], g[j-1], p[i]) >= 0 ){
            g.pop_back(); j--;
        }
        g.push_back(p[i]); j++;
    }

    return g;
}

/**** Exemplo simples de uso ****/
int main() {
    int i,n;
    poly p;
    pt k;

    scanf(" %d", &n);
    while(n--){
        scanf(" %lf %lf", &k.x, &k.y);
        p.push_back(k);
    }

    poly g = graham(p);

    for(i=0 ; i<g.size() ; i++)
        printf("%lf,%lf\n", g[i].x, g[i].y);

    return 0;
}

```

3.6 Diâmetro de Pontos e Polígono

Autor: Marcelo Póvoa

Complexidade: $O(n)$ polígono convexo, $O(n \lg n)$ pontos

Testes: SPOJ.TFOSS (usar int64)

Dependências: Graham (exceto para polígono convexo)

Descrição: Calcula a maior distância entre um par de pontos de um polígono ou de um conjunto de pontos em posição geral

```

#include <cmath>
#include <vector>

```

```

#include <algorithm>
using namespace std;

typedef pair<int,int> pii;

int cmpa(poly &p, int i, int j) {
    int n = p.size();
    return cmp(triarea(p[i], p[(i+1) % n], p[(j+1) % n]),
        triarea(p[i], p[(i+1) % n], p[j]));
}

/* Retorna os O(n) pares de vértices de um poligono convexo
pelos quais passam um par de retas de suporte paralelas. O
polígono deve ser anti-horário e pode ter pontos colineares */
vector<pii> antipodals(poly &p) {
    int n = p.size();
    int i = n - 1, j;

    for (j = 0; cmpa(p, i, j) >= 0; j++) {}

    vector<pii> res(1, pii(i, j));
    int k = j;
    while (j) {
        i = (i+1) % n;
        res.push_back(pii(i, j));
        while (j && cmpa(p, i, j) >= 0) {
            j = (j+1) % n;
            if (i != k || j != 0)
                res.push_back(pii(i, j));
        }
        if (!cmpa(p, i, j) && (i != k || j != n-1))
            res.push_back(pii(i, (j+1) % n));
    }
    return res;
}

double diam_convex(poly p) { // p em sentido horário
    double res = 0;
    reverse(p.begin(), p.end());

    vector<pii> c = antipodals(p);
    for (int i = 0; i < c.size(); i++)
        res = max(res, norma(p[c[i].first]-p[c[i].second]));

    return res;
}

double diam_points(poly &p) {
    if (p.size() <= 1) return 0;
    if (p.size() == 2) return norma(p[0] - p[1]);
    return diam_convex(graham(p));
}

/**** Exemplo simples de uso ****/
int main() {
    int n;
    poly p;
    pt k;

```

```

    scanf(" %d", &n);
    while(n--){
        scanf(" %lf %lf", &k.x, &k.y);
        p.push_back(k);
    }

    printf("%.2lf\n", diam_convex(p));
    return 0;
}

```

3.7 Distância Esférica

Autor: Guilherme Kunigami

Complexidade: $O(1)$

Tempo de implementacao: 1min

Testes: Uva 10075, nuevo 4153

Descricao: Calcula a distância entre 2 pontos em uma esfera

```

#include <cmath>

double torad;
double r = 6378;

struct geo {
    double lat, lon;
    geo(double lat1 = 0.0, double lon1 = 0.0) {
        lat = lat1 * torad;
        lon = lon1 * torad;
    };

    double geoDist(geo a, geo b)
    {
        return acos(sin(a.lat) * sin(b.lat) +
            cos(a.lat)*cos(b.lat)*cos(fabs(a.lon - b.lon)))*r;
    }

    /**** Exemplo simples de uso ****/
    int main(void)
    {
        torad = acos(-1) / 180.0;
        /* Calcula a distancia entre os pontos a e b, dadas
        sua latitude/longitude no planeta de raio r */
        geo a(23.8500, 90.4000);
        geo b(22.2500, 91.8333);
        printf("Distancia esferica: %.3lf\n", geoDist(a, b));

        return 0;
    }
}

```

3.8 Estrutura e Base para Geométricos

Autor: Alexandre Kunieda (+ PUC-Rio)

Tempo de implementacao: 5 minutos

Descricao: Contem estrutura de ponto, reta, poligono e algumas operacoes-base para os algoritmos geometricos

```

#include <cmath>
#include <vector>

using namespace std;

```

```

const double pi = acos(-1);

int cmp(double a, double b = 0){
    if (fabs(a-b)<1e-8) return 0;
    if (a<b) return -1;
    return 1;
}

struct pt {
    double x,y;
    explicit pt(double x = 0, double y = 0): x(x), y(y) {}

    pt operator +(pt q){ return pt(x + q.x, y + q.y); }
    pt operator -(pt q){ return pt(x - q.x, y - q.y); }
    pt operator *(double t){ return pt(x * t, y * t); }
    pt operator /(double t){ return pt(x / t, y / t); }
    double operator *(pt q){ return x * q.x + y * q.y; }
    double operator %(pt q){ return x * q.y - y * q.x; }

    int cmp(pt q) const {
        if (int t = ::cmp(x, q.x)) return t;
        return ::cmp(y, q.y);
    }
    bool operator ==(pt q) const { return cmp(q) == 0; }
    bool operator !=(pt q) const { return cmp(q) != 0; }
    bool operator < (pt q) const { return cmp(q) < 0; }
};

struct reta {
    pt ini,fim;
    reta(){}
    reta(pt ini, pt fim): ini(ini), fim(fim) {}
};

struct eq_reta {
    double A,B,C; /* Ax + By + C = 0 */

    void init(reta p){
        pt aux = p.ini - p.fim;
        A = aux.y;
        B = -aux.x;
        C = -A*p.ini.x - B*p.ini.y;
    }
    eq_reta(reta p){ init(p); }
};

typedef vector<pt> poly;
typedef pair<pt,double> circ;

pt normal(pt v){ return pt(-v.y,v.x); }
double norma(pt v){ return hypot(v.x, v.y); }
pt versor(pt v){ return v/norma(v); }
double anglex(pt v){ return atan2(v.y, v.x); }
double angle(pt v1, pt v2){ /* angulo orientado ccw */
    return atan2(v1%v2, v1*v2);
}
double triarea(pt a, pt b, pt c){ /* area c/ sinal */
    return ((b-a)%(c-a))/2.0; /* area>0 = ccw ; area<0 = cw */
}

```

```

int ccw(pt a, pt b, pt c){ /* b-a em relacao a c-a */
    return cmp((b-a)%(c-a)); /* ccw=1 ; cw=-1 ; colinear=0 */
    /* equivalente a cmp(triarea(a,b,c)), mas evita divisao */
}
pt projecao(pt v, pt w){ /* proj de v em w */
    double alfa = (v*w)/(w*w);
    return w*alfa;
}

```

3.9 Intersecção de Polígonos Convexos

Autor: PUC-Rio
 Complexidade: $O(n+m)$
 Tempo de implementacao: 8 min
 Testes: uva.137
 Dependencias:
 - ccw()
 - between3()
 - intersect_seg()
 - acha()
 - inpoly()
 Descricao: O algoritmo devolve a interseccao de dois poligonos convexos, orientados em sentido anti-horario.
 Pode ser utilizado `inpoly_convex()`, sem verificacao de ponto na borda do poligono, ja que os poligonos sao convexos.

```

#define all(x) (x).begin(),(x).end()

/* os poligonos P e Q devem estar orientados em
    sentido anti-horario! */
poly poly_intersect(poly& P, poly& Q) {
    int m = Q.size(), n = P.size();
    int a = 0, b = 0, aa = 0, ba = 0, inflag = 0;
    poly R;
    while ((aa < n || ba < m) && aa < 2*n && ba < 2*m) {
        pt p1 = P[a], p2 = P[(a+1) % n],
            q1 = Q[b], q2 = Q[(b+1) % m];
        pt A = p2 - p1, B = q2 - q1;
        int cross = cmp(A % B), ha = ccw(p2, q2, p1),
            hb = ccw(q2, p2, q1);
        if (cross == 0 && ccw(p1, q1, p2) == 0 && cmp(A*B) < 0) {
            if (between3(p1, q1, p2)) R.push_back(q1);
            if (between3(p1, q2, p2)) R.push_back(q2);
            if (between3(q1, p1, q2)) R.push_back(p1);
            if (between3(q1, p2, q2)) R.push_back(p2);
            if (R.size() < 2) return poly();
            inflag = 1; break;
        } else if (cross != 0 && intersect_seg(p1, p2, q1, q2)) {
            if (inflag == 0) aa = ba = 0;
            R.push_back(acha(p1, p2, q1, q2));
            inflag = (hb > 0) ? 1 : -1;
        }
        if (cross == 0 && hb < 0 && ha < 0) return R;
        bool t = cross == 0 && hb == 0 && ha == 0;
        if (t ? (inflag==1) : (cross>0) ? (ha<0) : (hb>0)) {
            if (inflag == -1) R.push_back(q2);
            ba++; b++; b % m;
        } else {
            if (inflag == 1) R.push_back(p2);
            aa++; a++; a % n;
        }
    }
}

```

```

    }
}
if (inflag == 0) {
    if (inpoly(P[0], Q)) return P;
    if (inpoly(Q[0], P)) return Q;
}
R.erase(unique(all(R)), R.end());
if (R.size() > 1 && R.front() == R.back()) R.pop_back();
return R;
}

```

```

/**** Exemplo simples de uso ****/
int main() {
    return 0;
}

```

3.10 Par de Pontos Mais Próximos

Autor: Notebook Unicamp (Mundial)
 Complexidade: $O(n \lg(n))$
 Tempo de implementacao: 2 minutos
 Testes:
 - UVa 10245 ($n \leq 10000$) $t = 0.290$ s
 Dependencias:
 - norma()
 Descricao: Obtem a menor distancia entre pontos de um conjunto de pontos. Eh preciso que o conjunto contenha pelo menos 2 pontos para o algoritmo funcionar

```

#include <set>
#define foreach(it, a,b) for(typeof(a)it=(a) ; it!=(b) ; it++)
#define all(x) (x).begin(), (x).end()

bool ycmp(pt a, pt b) {
    if (a.y!=b.y) return a.y<b.y;
    return a.x<b.x;
}

double closest_pair (poly &P) {
    int n = P.size();
    double d = norma(P[0]-P[1]);
    set<pt, bool(*)>(pt,pt)> s(&ycmp);

    sort(all(P));
    for(int i=0,j=0 ; i<n ; i++) {
        pt lower(0, P[i].y - d), upper(0, P[i].y + d);
        while(P[i].x - P[j].x > d)
            s.erase(P[j++]);

        foreach(p, s.lower_bound(lower), s.upper_bound(upper))
            /* os pontos mais proximos sao tirados de P[i] e *p */
            d = min(d, norma(P[i] - *p));
        s.insert(P[i]);
    }
    return d;
}

/**** Exemplo simples de uso ****/
int main(){
    /**** especifico para o problema UVa 10245 ****/
    pt i;
}

```

```

poly p;
int k,n;
double d;

while(scanf(" %d", &n)==1 && n) {
    p.clear();

    for(k=0 ; k<n ; k++) {
        scanf(" %lf %lf", &i.x,&i.y);
        p.push_back(i);
    }

    if(n==1) d = 15000.0;
    else d = closest_pair(p);

    if(d>10000) puts("INFINITY");
    else printf("%.4lf\n", d);
}

return 0;
}

```

3.11 Verificações de Ponto em Polígono

Autor: Alexandre Kunieda (+ PUC-Rio + note da Mundial)

Complexidade: $O(n)$, $O(\lg(n))$

Tempo de implementacao: 1 minuto (cada)

Testes:

- inpoly(): uva.634

- inpoly_convex(): testes gerados na mão

Dependencias:

- ccw()

- between3()

- intri() - soh inpoly_convex()

```

/**
Estrutura de ponto e poligono aqui
**/

```

```

int intri(pt k, pt a, pt b, pt c){
    int a1,a2,a3;

    a1 = ccw(a,k,b);
    a2 = ccw(b,k,c);
    a3 = ccw(c,k,a);

    if((a1*a2)>0 && (a2*a3)>0) return 1; /*dentro*/
    if(between3(k,a,b) || between3(k,b,c) || between3(k,c,a))
        return 2; /*borda*/
    return 0; /*fora*/
}

```

```

int inpoly(pt k, poly &p){
    int n = p.size();
    int cross = 0;

    for(int i=1; i<=n ; i++) {
        pt q=p[i-1], r=p[i%n];

        if( between3(k,q,r) ) return 2;
    }
}

```

```

    if( q.y>r.y ) swap(q,r);
    if( q.y<k.y && r.y>=k.y && ccw(k,q,r)>0 ) cross++;
}

return cross%2;
}

/* 0(lg(n)) - só para poligonos convexos */
int inpoly_convex(pt k, poly& p){
    /* 'val' indica o sentido do poligono */
    int val = ccw(p[0],p[1],p[2]);
    /* tomar cuidado para o caso em que o poligono
       começa com pontos colineares, 'val' receberá 0 */

    int esq,dir,meio, n = p.size();

    esq = 1; dir = n-1;
    while(dir>esq+1) {
        meio = (esq+dir)/2;

        if(ccw(p[0],p[meio],k) == val) esq = meio;
        else dir = meio;
    }

    return intri(k, p[0],p[esq],p[dir]);

    /* caso seja preciso verificar se está na borda,
       * substituir o return por:
       *
       * if(between3(k,p[esq],p[dir]) ||
       *    between3(k,p[0],p[1]) ||
       *    between3(k,p[0],p[n-1])) return 2; //BORDA
       * return intri(k, p[0],p[esq],p[dir])?1:0; //DENTRO:FORA
       */
}

/**** Exemplo simples de uso ****/
int main() {
    pt k;
    poly p;
    int n;
    int val;

    scanf(" %d", &n);
    while(n--) {
        scanf(" %lf %lf", &k.x,&k.y);
        p.push_back(k);
    }

    scanf(" %lf %lf", &k.x,&k.y);

    val = inpoly(k,p);
    /*val = inpoly_convex(k,p);*/
    printf("%d\n", val);

    return 0;
}

```

4 Numéricos

4.1 Binomial Modular (e não modular)

Autor: Alexandre Kunieda

Complexidade: binomial: $O(n)*O(\lg \text{MOD})$; binomial_: $O(n)$

Testes: binomial: DIOFANTO; binomial_: uva.530, uva.369

Dependencias: binomial: inverso modular; binomial_: nenhuma

Descricao: binomial(n,k,M) calcula o binomial $C(n,k) \% M$ sem overflow, utilizando inverso modular; binomial_(n,k) determina qualquer $C(n,k)$ que caiba em um int (menor que $2^{31}-1$)

```

#define MOD 1300031
typedef long long int64;

```

```

int64 fat(int64 n, int64 M = MOD) {
    int64 i, fat = 1;
    for(i=2 ; i<=n ; i++)
        fat = (fat*i)%M;
    return fat;
}

```

```

int64 binomial(int64 n, int64 k, int64 M = MOD) {
    int64 a = fat(n)*invmod(fat(k),M);
    int64 b = invmod(fat(n-k),M);
    return ( a%M*b )%M;
}

```

```

int64 binomial_(int n, int k) {
    if(n-k < k) k = n-k;
    if(k == 0) return 1;
    return (n-k+1)*binomial_(n,k-1)/k;
}

```

**** Exemplo simples de uso ****

```

#include <stdio.h>
#define MAX 10

```

```

int main() {
    for(int i=0 ; i<MAX ; ++i) {
        for(int j=0 ; j<=i ; ++j)
            printf("%lld ", binomial(i,j));
        putchar('\n');
    }
}

```

```

    for(int i=0 ; i<MAX ; ++i) {
        for(int j=0 ; j<=i ; ++j)
            printf("%lld ", binomial_(i,j));
        putchar('\n');
    }
}

```

```

    return 0;
}

```

4.2 Crivo de Erastótenes

Autor: Felipe Sodre

Complexidade: $O(N \log \log N)$

Tempo de implementacao: 2 minutos

Testes: todo(fsodre)

Descricao: Popula o array pr, de tal forma que pr[i] eh

verdadeiro se i eh primo.

```
#include <iostream>
#include <cstdio>

// Numero maximo a ser analisado
#define MAXN 1123123

// se pr[i] == true, i eh primo
bool pr[MAXN+1];

// algum divisor primo de i. Para fatoracao.
int divisor[MAXN+1];

// Analisa primalidade no intervalo [1,n]
void crivo(int n) {
    memset(pr, true, n * sizeof(bool));
    pr[0] = pr[1] = false;
    for(int i = 2; i*i <= n; i++){
        if( !pr[i] || !(i&1) && i > 2) continue;
        int k = i*i;
        divisor[i] = i;
        while(k <= n){
            pr[k] = false;
            divisor[k] = i;
            k += i;
        }
    }
}

**** Exemplo simples de uso ****/
int main(void){
    crivo(500);
    if(pr[2]) printf("2 eh primo\n");
    if(pr[9]) printf("9 eh primo\n");

    return 0;
}
```

4.3 Eliminação de Gauss

Autor: Marcelo Galvão Póvoa
Complexidade: $O(n^3)$
Testes: poj-3756
Descricao: Calcula, se existir, a inversa mi da matriz ma com pivoteamento, sendo inicialmente mi a identidade. Pode ser usado colocando uma matriz-coluna de constantes em mi para se obter a solução do sistema linear.

```
#include <cmath>
#include <algorithm>
using namespace std;
#define MAXN 100

double ma[MAXN][MAXN], mi[MAXN][MAXN];

bool invert(int n) {
    for (int k=0; k<n; k++) {
        int imax=k;
        for (int i=k+1; i<n; i++)
```

```
        if (fabs(ma[i][k]) > fabs(ma[imax][k])) imax=i;

        double p = ma[imax][k];
        if (fabs(p) < 1e-8) return false;

        for (int j=0; j<n; j++) {
            swap(ma[k][j], ma[imax][j]);
            swap(mi[k][j], mi[imax][j]);
            ma[k][j] /= p; mi[k][j] /= p;
        }

        for (int i=0; i<n; i++) {
            if (i == k) continue;
            double mul = ma[i][k];
            for (int j=0; j<n; j++) {
                ma[i][j] -= ma[k][j]*mul;
                mi[i][j] -= mi[k][j]*mul;
            }
        }
    }
    return true;
}

**** Exemplo simples de uso ****/
int main() {
    int n;

    scanf("%d",&n);
    for (int i=0; i<n; i++)
        for (int j=0; j<n; j++) {
            scanf("%lf",&ma[i][j]);
            mi[i][j] = (i==j)?1.0:0.0;
        }

    if (!invert(n)) printf("singular\n");
    else {
        for (int i=0; i<n; i++) {
            for (int j=0; j<n-1; j++)
                printf("%.1lf ", mi[i][j]);
            printf("%.1lf\n", mi[i][n-1]);
        }
    }
    return 0;
}
```

4.4 Estrutura de Polinômio

Autor: Alexandre Kunieda/Marcelo Póvoa
Testes: uva.10719, uva.10326, nuevo.4460
Descricao: Estrutura e operações com polinômios

```
#include <math.h>
#include <vector>
#include <algorithm>

using namespace std;

const double EPS = 1e-8;

typedef vector<double> vd;
```

```
struct polin {
    vd p; // expoentes decrescentes

    polin(){}
    polin(double x) { p.push_back(x); }
    polin(vd v): p(v) { fix(); }

    int grau() { return p.size()-1; }
    double& operator [](int i) { return p[i]; }

    void fix() {
        int i;
        for(i=0; i<=grau() && fabs(p[i])<EPS; i++) ;
        p.erase(p.begin(), p.begin()+i);
    }

    polin operator + (polin &q) {
        int k = q.grau() - grau();
        polin sum = (k>0)? q : p;

        for(int i=sum.grau(); i>=abs(k); i--)
            sum[i] += (k>0)? p[i-k] : q[i+k];

        sum.fix();
        return sum;
    }

    polin operator - (polin &q) {
        return q*(-1) + *this;
    }

    polin operator * (double x) {
        polin prod(p);
        for(int i=0; i<=grau(); i++)
            prod.p[i] *= x;
        return prod;
    }

    polin operator * (polin &q) {
        polin prod;

        for(int i=0; i<=grau(); i++) {
            polin aux = q * p[i];
            aux.p.resize(q.grau()+p.size()-i, 0);
            prod = prod + aux;
        }
        return prod;
    }

    pair<polin,polin> operator / (polin &d) {
        polin resto(p);
        polin q;
        int g = grau(), dg = d.grau();
        for(int i=0; i<=g-dg; i++) {
            double a = resto[i] / d[0];
            for(int j=0; j<=dg; j++)
                resto[i+j] -= a*d[j];
            q.p.push_back(a);
        }

        resto.fix();
        return make_pair(q, resto);
    }
}
```



```

polin operator ~ () { // derivada
    polin dp;
    int g = grau();
    for(int i=0; i<g; i++)
        dp.p.push_back(p[i] * (g-i));
    return dp;
}

double eval(double x) {
    double res=0, pw=1;
    for (int i=grau(); i>=0; i--, pw*=x)
        res += pw*p[i];
    return res;
}
};

```

```

polin mdc(polin &a, polin &b) {
    if(b.grau() == -1) return a;
    polin resto = (a/b).second;
    return mdc(b, resto);
}

```

```

/**** Exemplo simples de uso ****/
#include <stdio.h>

```

```

int main(){
    vd a0(3), b0(2);
    //x^2 + 4x - 3
    a0[0] = 1; a0[1] = 4; a0[2] = -3;
    //2x - 1
    b0[0] = 2; b0[1] = -1;

    polin a(a0), b(b0);
    polin x = ((a * a)/b).first;
    printf("quotient(a^2/b) =");

    if (x.grau() == -1) puts("0");
    else {
        for(int i=0; i<=x.grau(); i++)
            printf(" %+.1lfx^%d", x[i], x.grau()-i);
        putchar('\n');
    }

    return 0;
}

```

4.5 Euclides Extendido

Autor: NU 2/Marcelo Galvão Póvoa
 Complexidade: $O(\lg x)$
 Tempo de implementacao: 1 min
 Testes: SPOJ.DIOFANTO
 Descricao: Calcula um par x, y tal que $a*x+b*y=mdc(a,b)$

```

#include <algorithm>
using namespace std;

typedef pair<int,int> pii;

pii mdc(int a, int b){

```

```

    if (b == 0) return pii(1,0);
    pii u = mdc(b,a%b);
    return pii(u.second, u.first - (a/b)*u.second);
}

```

/**** Exemplo simples de uso ****/

```

int main(){
    int a,b;
    pii euext;

    scanf("%d %d",&a,&b);
    euext=mdc(a,b);

    printf("%d %d\n",euext.first,euext.second);
    return 0;
}

```

4.6 Exponenciação de Matrizes - Fibonnaci

Autor: Marcelo Galvão Póvoa / Douglas Santos
 Complexidade: $O(d^3 \log n)$
 Testes: SPOJ.RABBIT1
 Descricao: Exponenciação rápida modular de matriz
 e como exemplo cálculo do fibonnaci

```

#include <cstring>
#define MOD 1000000
#define MAXD 2
using namespace std;

typedef long long int64;
typedef int64 mat[MAXD][MAXD];

```

```

void mul(mat &c, mat &a, mat &b, int d) {
    mat r = {{0,0}};

    for (int i=0;i<d;i++)
        for (int j=0;j<d;j++)
            for (int k=0;k<d;k++) {
                r[i][j] += (a[i][k] * b[k][j]) % MOD;
                r[i][j] %= MOD;
            }

    memcpy(c,r,sizeof(r));
}

```

```

/* res = (a^n) % mod, d = dimensão da matriz */
void expo(mat &res, mat &a, int64 n, int d = MAXD) {
    mat y, x;
    memcpy(y, a, sizeof(a));
    memset(x, 0, sizeof(x));
    for (int i = 0; i < d; i++) x[i][i] = 1;

    for (n--; n>0; n/=2) {
        if (n % 2) mul(x,x,y,d);
        mul(y,y,y,d);
    }
    memcpy(res, x, sizeof(x));
}

```

```

/**** Exemplo simples de uso ****/
/* Cálculo do fibonacci */
int main() {
    int64 n;
    mat a = {{1, 1}, {1, 0}};
    while (scanf("%lld",&n)==1) {
        mat f;
        expo(f, a, n);
        printf("%lld\n",(f[0][0] + f[0][1]) % MOD);
    }
    return 0;
}

```

4.7 Exponenciação modular rápida

Autor: Douglas Oliveira Santos
 Complexidade: $O(\log(b))$
 Descricao: Dado a, b e m , calcula $a^b \bmod m$

```

#include <stdio>
#include <cstring>

```

```

using namespace std;

```

```

typedef long long int int64;

int64 expo(int64 a, int64 b, int64 m) {
    int64 y = a % m, x = 1;
    while (b > 0) {
        if (b % 2 == 1) {
            x = (x*y) % m;
        }
        y = (y*y) % m;
        b = b/2;
    }
    return x % m;
}

```

```

/**** Exemplo simples de uso ****/
int main()
{
    int64 res;
    res = expo(2, 1000000007-2, 1000000007);
    printf("%lld\n", res);
    return 0;
}

```

4.8 Fatoração de Número Inteiro

Autor: Felipe Sodre
 Complexidade: $O(\log N)$
 Tempo de implementacao: 1 minuto
 Testes: todo(fsodre)
 Dependencias:
 - Crivo de Erastotenes
 Descricao:
 fatora(n, arr) coloca no array arr todos os fatores
 primos de n, nao necessariamente em ordem. Retorna
 a quantidade de fatores primos.

```
#include <iostream>
#include <cstdio>

/**
    Crivo aqui
**/

inline int div(int n){
    if(pr[n]) return n;
    return divisor[n];
}

int fatora(int n, int fatores[]){
    if(n <= 1){
        fatores[0] = n;
        return 0;
    }

    int k = 0;
    while(n > 1){
        fatores[k++] = div(n);
        n /= div(n);
    }
    return k;
}

/**** Exemplo simples de uso ****/
int main(void){
    int nums[15];

    crivo(500);

    int qt = fatora(444, nums);
    for(int i = 0; i < qt; i++){
        printf("%d eh um fator de 444.\n",nums[i]);
    }

    return 0;
}
```

4.9 Gray Code

Autor: Douglas Santos
Complexidade: $O(1)$
Testes: uva-12447.cpp
Descricao: Calcula o código de gray, o inverso do código de gray e o código de gray invertido.

```
/* Retorna o i-ésimo elemento do código de gray */
int gray(int i) {
    return i ^ ( i >> 1 );
}

/* Retorna a posição de um elemento no código de gray */
int inverso_gray(int g) {
    int n = 0;
    for (; g >= 1)
        n ^= g;
    return n;
}
```

```
/* Retorna o i-ésimo elemento do código de gray invertido
   de n bits. No código de gray invertido, o sucessor de
   um elemento tem apenas 1 bit igual a ele */
int gray_invertido(int i, int n) {
    int y = i;
    if (i % 2) y = i ^ ((1 << n) - 1);
    int x = ((i/2) | (i & ((n % 2) * (1 << (n-1))))) ^ y;
    return x;
}
```

4.10 Inverso Modular

Autor: NU2/Marcelo Galvão Póvoa
Complexidade: $O(\lg x)$
Tempo de implementacao: 3 min
Testes: SPOJ.DIOFANTO
Descricao: Calcula um x tal que $a \cdot x \equiv 1 \pmod{M}$
Para a e M coprimos, eh garantido que x eh unico
Nesse caso, pode ser usado para determinar
a divisao modular como exemplificado.

```
#include <algorithm>
using namespace std;

typedef pair<int,int> pii;

pii mdc(int a, int b){
    if (b == 0) return pii(1,0);
    pii u = mdc(b,a%b);
    return pii(u.second, u.first - (a/b)*u.second);
}

int invmod(int a, int M) {
    pii r=mdc(a,M);
    if (r.first * a + r.second * M == 1)
        return (r.first + M) % M;
    return 0;
}
```

```
#include <cstdio>
/**** Exemplo simples de uso ****/
int main() {
    int x,m;

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

    /*retorna 36/x (mod m), se x eh divisor de 36*/
    //printf("%d\n",36*invmod(x,m) % m);
    printf("%d\n",invmod(x,m));
    return 0;
}
```

4.11 Log Discreto

Autor: Marcelo Galvão Póvoa
Complexidade: $O(\sqrt{m} \cdot \lg m)$
Testes: uva10225
Descricao: Dados a , b e m (a e m devem ser coprimos),
calcula o menor x tal que $a^x \equiv b \pmod{m}$

```
#include <cmath>
#include <map>
using namespace std;

int baby_giant(int a, int b, int m) {
    int n = ceil(sqrt(m));
    int an = 1;
    for (int i = 0; i < n; i++)
        an = (an * 1LL * a) % m;

    map<int, int> vals;
    for (int i = 0, cur = b; i <= n; i++) {
        vals[cur] = i;
        cur = (cur * 1LL * a) % m; // baby step
    }

    for (int i = 1, cur = an; i <= n; i++) {
        if (vals.count(cur)) {
            int ans = i*n - vals[cur];
            if (ans < m)
                return ans;
        }
        cur = (cur * 1LL * an) % m; // giant step
    }
    return -1;
}
```

```
/**** Exemplo simples de uso ****/
int main() {
    int p, b, n;
    while (scanf("%d %d %d", &p, &b, &n) != EOF) {
        int x = baby_giant(b, n, p);

        if (x == -1) printf("no solution\n");
        else printf("%d\n", x);
    }
    return 0;
}
```

4.12 Operações com Frações

Autor: Davi Costa
Tempo de implementacao: 7min
Testes: uva-10808, uva-684
Descricao: Realiza todas as operacoes com
numeros racionais menos MOD. Eh fortem-
ente aconselhavel usar fra<long long>
A fracao eh sempre irredutivel e o
denominador eh sempre positivo.

```
#define ABS(x) (x < (typeof(x))0 ? -x : x)
#define F fra<T>
```

```
template<class T>
class fra {
public:
    T n,d;
    T gcds(const T &a, const T &b) const {
        if (b == T(0)) return a;
```

```

    return gcds(b,a%b);
}
T gcd(T a, T b) const {
    a = ABS(a); b = ABS(b);
    if (a < b) return gcds(b,a);
    return gcds(a,b);
}
fra(): n(T(0)), d(T(1)) {}
fra(T num, T den = 1): n(num), d(den) {
    if (d < T(0)) d=-d, n=-n;
    T g = gcd(n,d); n /= g; d /= g;
}
F operator+(const F &f) const {
    // + rapido + overflow descomente a linha abaixo
    //return F(n*f.d + f.n*d,d*f.d);
    T g = gcd(d,f.d);
    return F(f.d/g*n + d/g*f.n,d/g*f.d);
}
F operator-(const F &f) const { return F(-n,d); }
F operator-(const F &f) const { return -f + *this; }
F operator*(const F &f) const {
    // + rapido + overflow descomente a linha abaixo
    //return F(n*f.n,d*f.d);
    F f1(n,f.d), f2(f.n,d);
    return F(f1.n*f2.n,f1.d*f2.d);
}
F operator/(const F &f) const { return *this*f(f.d,f.n); }

F &operator+=(const F &f){ *this = *this+f; return *this; }
F &operator-=(const F &f){ *this = *this-f; return *this; }
F &operator*=(const F &f){ *this = *this*f; return *this; }
F &operator/=(const F &f){ *this = *this/f; return *this; }

bool operator==(const F &f) const {return n==f.n && d==f.d;}
bool operator!=(const F &f) const {return n!=f.n || d!=f.d;}
bool operator< (const F &f) const {return (*this-f).n<T(0);}
bool operator> (const F &f) const {return f < *this;}
};

/** Exemplo simples de uso */
typedef fra<long long> fll;

int main() {
    fll f1= 3;
    fll f2(2,3);
    fll f3 = f1/f2;
    long long numerador = f3.n;
    long long denominador = f3.d;
    return 0;
}

```

4.13 Operações com Matrizes

Autor: Davi Costa

Complexidade:

pot() - $O(\log n)$ multiplicacoes

*, linsys, inv, det - $O(n^3)$

Dependencias:

Todas as funcoes precisam da parte Obrigatoria

pot -> *; - -> +

```

linsys -> diag, det -> diag, inv -> diag
Tempo de implementacao:
Cada bloco Independente ~ 3-5min, Total ~ 12min
Testes:
spoj-vampiros, gcj-CollectinCards, (linsys, inversa)
uva-10808 (linsys com racional)
nuevo-4332(Exponenciacao modular), uva-684(Determinante)
Soh foram feitos testes locais de sistema impossivel
Descricao:
Realiza diversas operacoes com matrizes
- Modular: Descomente MOD, lembre de corrigir
valores negativos posteriormente
- det, inv e linsys soh funcionam com double
ou fracao, se usar int sera convertido tudo
pra double antes. Se usar fracao troque
todos os "double" por "T" e "EPS" por "0"
- 0 resolvidor de sistemas lineares seta
duas flags no vetor resposta, mulsol indica
multiplas solucoes e nosol indica que nao ha solucao.
Pode-se descomentar alguns trechos abaixo para que
caso existam multiplas solucoes o programa escolha
valor 1.0 para algumas variaveis, porem o programa
nao indicara se o sistema era inicialmente impossivel.

#include <iostream>
#include <algorithm>
#include <math.h>
#include <cstring>
#include <vector>

#define FOR(i,n) for(int i=0; i<n; ++i)
#define ABS(x) (x < (typeof(x))0 ? -x : x)
#define EPS (1e-8)

using namespace std;

template<class T>
class matrix {
public:
    /*Obrigatorio*/
    int n,m,sz;
    T *data;
    double deter;
    bool nosol, mulsol;
    matrix() { n = m = sz = 0; data = NULL; }
    matrix(int n, int m): n(n), m(m) {
        sz = n*m; data = (T*)malloc(sizeof(T)*sz);
    }
    matrix(const matrix<T> &mtx){data = NULL;*this = mtx;}
    ~matrix() { free(data);}
    const matrix<T> &operator=(const matrix<T> &mtx) {
        if (&mtx == this) return *this;
        n = mtx.n; m = mtx.m; sz = mtx.sz;
        deter = mtx.deter; nosol = mtx.nosol, mulsol = mtx.mulsol;
        data = (T*)realloc(data,sizeof(T)*sz);
        memcpy(data,mtx.data,sizeof(T)*sz);
        return *this;
    }
    T *operator[](int i) { return data + i*m; }
    void zera() { memset(data,0,sizeof(T)*sz); }

```

```

//Fracao void zera() { FOR(i,sz) data[i] = 0; }
/*Facilita a insercao manual de elementos*/
void setall(T* v) { memcpy(data,v,sizeof(T)*sz); }
void setl(int i,T* v) { memcpy(data + i*m,v,sizeof(T)*m); }
/*Exponenciacao*/
matrix<T> operator * (matrix<T> &mtx) {
    matrix<T> res(n,mtx.m);
    FOR(i,n) FOR(j,m) {
        res[i][j] = 0;
        FOR(k,m)
            res[i][j] = (res[i][j]+
                        (*this)[i][k]*mtx[k][j])/MOD%;
    }
    return res;
}
matrix<T> pot(int x) {
    matrix<T> res(n,m);
    if (x == 0)
        FOR(i,n) FOR(j,m) res[i][j] = (i == j);
    else pot(x,res); return res;
}
void pot(int x, matrix<T> &res) {
    if (x == 1) { res = *this; return; }
    pot(x/2,res); res = res*res;
    if (x%2) res = res*(*this);
}
/*Soma e Subtracao*/
matrix<T> operator + (matrix<T> &mtx) {
    matrix<T> res(n,m);
    FOR(i,sz) res.data[i] = (data[i] + mtx.data[i])/MOD%;
    return res;
}
matrix<T> operator -(const matrix<T> &mtx) {
    matrix<T> res(n,m);
    FOR(i,sz) res.data[i] = -this->data[i];
    return res;
}
matrix<T> operator -(matrix<T> &mtx) {
    return -mtx + *this;
}
/*Transpota*/
matrix<T> transp() {
    matrix<T> res(m,n);
    FOR(i,n) FOR(j,m) res[j][i] = (*this)[i][j];
    return res;
}
/*Funcao necessaria para determinante
sistema linear e inversa*/
matrix<double> diag(int mp = -1) {
#define M(i,j) (mat[idx[i]][j])
    matrix<double> ret(min(n,mp),m-mp), mat(n,m);
    if (mp == -1) mp = m;
    deter = 1; ret.nosol = false; ret.mulsol = false;
    FOR(i,sz) mat.data[i] = (double)data[i];
    vector<int> idx(n); FOR(i,n) idx[i]=i;
    int step=0;
    FOR(j,mp){
        int p = -1; double pivot = 0;
        for(int i=step;i<n;i++)
            if (ABS(M(i,j)) > ABS(pivot) + EPS)

```

```

    p = i, pivot = M(i,j);
    if(p!=-1) {
        /*Para obemultiplas solucoes comente continue
        e descomente as linhas abaixo */
        deter = 0; ret.mulsol = true;
        continue;
        //for(int jj = j+1; jj < m; jj++) M(step,jj) = 0.0;
        //M(step,j) = 1; M(step,m-1) = 1; p = step; pivot = 1;
    }
    if (p != step) deter *= -1, swap(idx[step],idx[p]);
    deter *= pivot;
    FOR(jj,m) M(step,jj) /= pivot;
    FOR(i,n) if (step!=i) {
        double w=M(i,j);
        FOR(k,m) M(i,k) -= w * M(step,k);
    }
    step++;
}
for (int i = step; i < n; i++)
    if (ABS(M(i,m))>EPS) {
        ret.nosol = true; ret.mulsol = false; break;
    }
FOR(i,min(n,mp)) FOR(j,m-mp) ret[i][j] = M(i,j+mp);
return ret;
#endif M
}
matrix<double> linsys(matrix<T> b) {
    //pra solucao viavel troque n por max(n,m) na linha abaixo
    matrix<double> msys(n,m+1);
    FOR(i,n) FOR(j,m) msys[i][j] = (double)(*this)[i][j];
    FOR(i,n) msys[i][m] = (double)b[i][0];
    for(int i = n; i < msys.n; i++) FOR(j,m+1) msys[i][j] = 0;
    return msys.diag(m);
}
double det() { diag(); return deter; }
matrix<double> inv() {
    matrix<double> minv(n,2*m);
    FOR(i,n) FOR(j,m) {
        minv[i][j] = (double)(*this)[i][j];
        minv[i][j+m] = (i == j);
    }
    return minv.diag(m);
}
};

template< class T >
ostream &operator<< ( ostream &out, matrix< T > mtx ) {
    FOR(i,mtx.n) {
        FOR(j,mtx.m) {
            if(j) out << " ";
            out << mtx[i][j];
        }
        out << endl;
    }
    out << endl;
    return out;
}
}

typedef matrix<double> md;

```

```

typedef matrix<double> md;

```

```

int main() {
    //inicializar a matriz
    md m(2,2);
    /* 0 1
       1 2 */
    m.setall((double[]){ 0, 1, 1, 2});
    cout << m;
    /*ou*/
    m.setl(0,(double[]){ 0, 1});
    m.setl(1,(double[]){ 1, 2});
    cout << m;
    /*ou*/
    m[0][0] = 0; m[0][1] = 1; m[1][0] = 1; m[1][1] = 2;
    cout << m;
    //negacao
    md neg = -m; cout << neg;
    //soma
    md msum = m + m; cout << msum;
    //subtracao
    md msub = m - m; cout << msub;
    //Exponenciacao
    md mpot = m.pot(15); cout << mpot;
    //identidade
    md ident = m.pot(0); cout << ident;
    //inversa
    md inv = m.inv(); cout << inv;
    //transposta
    md transp = m.transp(); cout << transp;
    //determinante
    double det = m.det(); cout << det << endl << endl;
    //sistema linear
    /* 0x + 1y = 1
       1x + 2y = -3 */
    md b(2,1);
    b.setall((double[]){ 1, -3 });
    cout << b;
    md ans = m.linsys(b);
    if (ans.mulsol) cout << "multiplas solucoes" << endl;
    if (ans.nosol) cout << "Sistema impossivel" << endl;
    cout << ans;
    /*ou sem informacoes sobre as solucoes*/
    ans = inv*b;
    cout << ans;
    double x = ans[0][0], y = ans[1][0];
    return 0;
}

```

4.14 Phi de Euler

Autor: Douglas Santos
 Complexidade: $O(\sqrt{n})$
 Testes: uva-12493, uva-12425
 Descricao: Calcula o número de elementos
 coprimos a n, no intervalo [1, n].

```

typedef long long int int64;

```

```

int64 phi(int64 n) {
    int64 res = n;
    for (int64 i = 2; i * i <= n; ++i)

```

```

    if (n % i == 0) {
        while (n % i == 0)
            n /= i;
        res -= res / i;
    }
    if (n > 1)
        res -= res / n;
    return res;
}

```

4.15 Raízes de Polinômio

Autor: Crbondy/Davi Costa
 Tempo de implementacao: 15~20 minutos
 Testes: uva-10428 (apenas raizes reais melhor tempo),
 nuevo-4460 (raizes complexas e reais)
 KMSL4B (Raizes complexas e rais melhor tempo)
 Descricao:
 Encontra todas as raizes reais e complexas
 de um polimio INCLUSIVE repetidas. Ver exemplo de uso
 OBS: Nao eh aconselhavel mudar o EPS

```

#include <math.h>
#include <stdlib.h>
#include <stdio>

```

```

using namespace std;

```

```

#define EPS (1e-10)
#define maxiter 500
#define MAX 21 //MAXDGREE + 1

```

```

inline int cmp(double a, double b = 0){
    if(fabs(a-b)<=EPS) return 0;
    if(a<b) return -1;
    return 1;
}

```

```

int roots(double *a,int n,double *wr,double *wi) {
    double sq,b2,c,disc;
    int m,numroots;
    m = n; numroots = 0;
    while (m > 1) {
        b2 = -0.5*a[m-2]; c = a[m-1]; disc = b2*b2-c;
        if (!cmp(disc/(b2*b2+fabs(c)))) disc = 0.0;
        if (disc < 0.0) {
            sq = sqrt(-disc);
            wr[m-2] = b2; wi[m-2] = sq;
            wr[m-1] = b2; wi[m-1] = -sq;
            numroots+=2;
        }
        else {
            sq = sqrt(disc);
            wr[m-2] = fabs(b2)+sq;
            if (b2 < 0.0) wr[m-2] = -wr[m-2];
            if (wr[m-2] == 0) wr[m-1] = 0;
            else {
                wr[m-1] = c/wr[m-2];
                numroots+=2;
            }
        }
    }
}

```

```

        wi[m-2] = wi[m-1] = 0.0;
    }
    m -= 2;
}
if (m == 1) {
    wr[0] = -a[0]; wi[0] = 0.0;
    numroots++;
}
return numroots;
}

void defl(double *a,int n,double *b,double *quad,double &err){
    double r,s,c[MAX];
    int i;
    r = quad[1]; s = quad[0];
    b[1] = a[1] - r; c[1] = b[1] - r;
    for (i=2;i<=n;i++){
        b[i] = a[i] - r * b[i-1] - s * b[i-2];
        c[i] = b[i] - r * c[i-1] - s * c[i-2];
    }
    err = fabs(b[n])+fabs(b[n-1]);
}

void find_quad(double *a,int n,double *b,
               double *quad,double &err, int &iter) {
    double c[MAX],dn,dr,ds,drn,dsn,eps,r,s;
    int i;
    c[0] = 1.0;
    r = quad[1]; s = quad[0];
    dr = 1.0; ds = 0; eps = EPS;
    iter = 1;
    while (cmp(fabs(dr)+fabs(ds))) {
        if (iter > maxiter) break;
        if ((iter % 200) == 0)eps*=10.0;
        b[1] = a[1] - r; c[1] = b[1] - r;
        for (i=2;i<=n;i++){
            b[i] = a[i] - r * b[i-1] - s * b[i-2];
            c[i] = b[i] - r * c[i-1] - s * c[i-2];
        }
        dn = c[n-1] * c[n-3] - c[n-2] * c[n-2];
        drn = b[n] * c[n-3] - b[n-1] * c[n-2];
        dsn = b[n-1] * c[n-1] - b[n] * c[n-2];
        if (!cmp(dn)) {
            if (dn < 0.0) dn = -EPS/100.;
            else dn = EPS/100.;
        }
        dr = drn / dn; ds = dsn / dn;
        r += dr; s += ds;
        iter++;
    }
    quad[0] = s; quad[1] = r;
    err = fabs(ds)+fabs(dr);
}

void diff_poly(double *a,int n,double *b) {
    double coef;
    int i;
    coef = (double)n; b[0] = 1.0;
    for (i=1;i<=n;i++)
        b[i] = a[i]*((double)(n-i))/coef;
}

```

```

}

void recurse(double *a,int n,double *b,int m,double *quad,
             double &err,int &iter) {
    double c[MAX],x[MAX], rs[2],tst;
    if (!cmp(b[m])) m--; // this bypasses roots at zero
    if (m == 2) {
        quad[0] = b[2]; quad[1] = b[1];
        err = iter = 0;
        return;
    }
    c[0] = x[0] = 1.0;
    rs[0] = quad[0];rs[1] = quad[1];
    iter = 0;
    find_quad(b,m,c,rs,err,iter);
    tst = fabs(rs[0]-quad[0])+fabs(rs[1]-quad[1]);
    if (!cmp(err)) {
        quad[0] = rs[0]; quad[1] = rs[1];
    }
    // tst will be 'large' if we converge to wrong root
    if ((iter > 5 && tst < 1e-4) || (iter > 20 && tst < 1e-1)) {
        diff_poly(b,m,c);
        recurse(a,n,c,m-1,rs,err,iter);
        quad[0] = rs[0]; quad[1] = rs[1];
    }
}

void get_quads(double *a,int n,double *quad,double *x) {
    double b[MAX],z[MAX],err,tmp;
    int iter,i,m;
    if ((tmp = a[0]) != 1.0) {
        a[0] = 1.0;
        for (i=1;i<=n;i++) {
            a[i] /= tmp;
        }
    }
    if (n == 2) {
        x[0] = a[1]; x[1] = a[2];
        return;
    }
    else if (n == 1) {
        x[0] = a[1];
        return;
    }
    m = n; b[0] = 1.0;
    for (i=0;i<=n;i++)
        z[i] = a[i], x[i] = 0.0;
    do {
        if (n > m)
            quad[0] = 3.14159e-1, quad[1] = 2.78127e-1;
        loop:
            find_quad(z,m,b,quad,err,iter);
            if ((err > 1e-7) || (iter > maxiter)) {
                diff_poly(z,m,b);
                iter = 0;
                recurse(z,m,b,m-1,quad,err,iter);
            }
        defl(z,m,b,quad,err);
        //mais seguro
        if (err > 0.01) {

```

```

            quad[0] = -0.003*rand();
            quad[1] = -0.002*rand();
        }
        if (err > 1) goto loop;
        x[m-2] = quad[1]; x[m-1] = quad[0];
        m -= 2;
        for (i=0;i<=m;i++) {
            z[i] = b[i];
        }
    } while (m > 2);
    if (m == 2) x[0] = b[1], x[1] = b[2];
    else x[0] = b[1];
}

/**** Exemplo simples de uso ****/
int main() {
    //sempre precisa declarar
    double a[MAX],x[MAX],wr[MAX],wi[MAX],quad[2];
    int n = 2, numr;
    //pos 0 tem o coeficiente do grau maximo
    //que precisa ser diferente de 0
    //Polinomio 2x^2 + 0x -8 = 0
    a[0] = 2; a[1] = 0; a[2] = -8;
    //bons valores iniciais
    quad[0] = 0.271828;
    quad[1] = 0.314159;
    get_quads(a,n,quad,x);
    numr = roots(x,n,wr,wi);
    //se numr for menor que n nao encontrou todas as raizes
    printf("Raiz 0: %lf + %lfi\n",wr[0],wi[0]);
    printf("Raiz 1: %lf + %lfi\n",wr[1],wi[1]);
    return 0;
}

```

4.16 Simplex

Descricao:

Resolve o problema de programação linear:

minimizar cx

sujeito a $Ax = b$

$x \geq 0$

// UVA 10498, Happiness

#include <iostream>

#include <cstdio>

#include <cmath>

#include <vector>

using namespace std;

typedef vector<double> array;

typedef vector<array> matrix;

const double EPS = 1e-8;

enum { OPTIMAL, UNBOUNDED, NOSOLUTION, UNKNOWN };

struct two_stage_simplex {

int N, M, st;

matrix a;

vector<int> s;

two_stage_simplex(const matrix &A, const array &b,

```

        const array &c)
: N(A.size()), M(A[0].size()), a(N+2, array(M+N+1)),
s(N+2), st(UNKNOWN) {
for (int j = 0; j < M; ++j) a[N+1][j] = c[j];
// make simplex table
for (int i = 0; i < N; ++i)
    for (int j = 0; j < M; ++j) a[i+1][j] = A[i][j];
for (int i = 0; i < N; ++i) a[i+1][M+N] = b[i];
// add helper table
for (int i = 0; i < N; ++i) a[0][i+M] = 1;
for (int i = 0; i < N; ++i) a[i+1][i+M] = 1;
for (int i = 0; i < N; ++i) s[i+1] = i+M;
for (int i = 1; i <= N; ++i)
    for (int j = 0; j <= N+M; ++j) a[0][j] += a[i][j];
st = solve();
}
int status() const { return st; }
double solution() const { return -a[0][M]; }
double solution(array &x) const {
    x.resize(M, 0);
    for (int i = 0; i < N; ++i)
        x[s[i+1]] = a[i+1].back();
    return -a[0][M];
}
int solve() {
    M += N; N += 1;
    solve_sub(); // solve stage one
    if (solution() > EPS) return NOSOLUTION;
    N -= 1; M -= N;
    swap(a[0], a.back()); a.pop_back(); // modify table
    for (int i = 0; i <= N; ++i) {
        swap(a[i][M], a[i].back());
        a[i].resize(M+1);
    }
    return solve_sub(); // solve stage two
}
int solve_sub() {
    int p, q;
    while (1) {
        //print();
        for (q = 0; q <= M && a[0][q] >= -EPS; ++q);
        for (p = 0; p <= N && a[p][q] <= EPS; ++p);
        if (q >= M || p > N) break;
        for (int i = p+1; i <= N; ++i)
            // bland's care for cyclation
            if (a[i][q] > EPS)
                if (a[i][M]/a[i][q] < a[p][M]/a[p][q] ||
                    (a[i][M]/a[i][q] == a[p][M]/a[p][q] &&
                     s[i] < s[q])) p = i;
        pivot(p, q);
    }
    if (q >= M) return OPTIMAL;
    else return UNBOUNDED;
}
void pivot(int p, int q) {
    for (int j = 0; j <= N; ++j)
        for (int k = M; k >= 0; --k)
            if (j != p && k != q)
                a[j][k] -= a[p][k]*a[j][q]/a[p][q];
    for (int j = 0; j <= N; ++j)

```

```

        if (j != p) a[j][q] = 0;
    for (int k = 0; k <= M; ++k)
        if (k != q) a[p][k] = a[p][k]/a[p][q];
    a[p][q] = 1.0;
    s[p] = q;
}
};
int main() {
    for (int n, m; cin >> n >> m; ) {
        array c(n+m), b(m);
        for (int i = 0; i < n; ++i)
            cin >> c[i], c[i] *= -1;
        matrix A(m, array(n+m));
        for (int i = 0; i < m; ++i) {
            for (int j = 0; j < n; ++j)
                cin >> A[i][j];
            A[i][n+i] = 1;
            cin >> b[i];
        }
        two_stage_simplex tss(A, b, c);
        double ans = -tss.solution() * m;
        printf("Nasa can spend %.0f taka.\n", ans + 0.5 - EPS);
    }
}

```

4.17 Teorema Chinês do Resto

Autor: NU 2/Marcelo Galvão Póvoa
 Complexidade: $O(n * \lg X)$
 Tempo de implementacao: 4 min
 Testes: Testes proprios
 Dependencias: Algoritmo de Euclides Extendido
 Descricao: Resolve o conj de eqs: $a[i]*x \equiv b[i] \pmod{m[i]}$
 para $0 \leq i < n$ com a restricao $m[i] > 1$
 Se $a[i] \equiv 1$ para todo i , existe solucao sse
 $b[i] \equiv b[j] \pmod{\gcd(m[i], m[j])}$ para todo i e j

```

#include <algorithm>
#include <vector>
#define MAXN 1000
using namespace std;

int n, a[MAXN], b[MAXN], m[MAXN];
typedef pair<int, int> pii;

int chines() {
    int x = 0, M = 1;
    for (int i = 0; i < n; ++i) {
        int b2 = b[i] - a[i]*x;
        pii bizu = mdc(a[i]*M, m[i]);
        int g = a[i]*M * bizu.first + m[i] * bizu.second;

        if (b2 % g) return -1;
        x += M * (bizu.first * (b2/g) % (m[i]/g));
        M *= (m[i]/g);
    }
    return (x%M+M)%M;
}

```

```

/**** Exemplo simples de uso ****/
int main(){
    int i;

    scanf("%d",&n);
    for (i=0; i<n; i++) scanf(" %d %d %d",&a[i], &b[i], &m[i]);

    printf("%d\n", chines());
    return 0;
}

```

5 Strings

5.1 Aho-Corasick

Autor: UFPE (com modificações)
 Complexidade: $O(\text{texto} + \text{padrões} + \text{ocorrências})$
 Testes: liveArchive.4811
 Descrição: Dado um conjunto de padrões (strings) e um texto, encontra todas as ocorrências dos padrões no texto

```

#include <map>
#include <vector>
#include <queue>
using namespace std;
typedef pair<int, int> pii;

/* Tamanho total dos padrões */
#define MAXST (1000100)

struct No {
    vector<pii> out; // num e tamanho do pad
    map<char, int> lis;
    int fail;
    int nxt; // aponta para o próx. sufixo com out.size > 0
};
No t[MAXST];
int qNo, qPad;

void init() {
    t[0].fail = t[0].nxt = -1;
    t[0].lis.clear();
    t[0].out.clear();
    qNo = 1;
    qPad = 0;
}

void add(const char *pad) {
    int no = 0, len = 0;
    for (int i = 0; pad[i]; i++, len++) {
        if (t[no].lis.find(pad[i]) == t[no].lis.end()) {
            t[qNo].lis.clear(); t[qNo].out.clear();
            t[no].lis[pad[i]] = qNo;
            no = qNo++;
        }
        else no = t[no].lis[pad[i]];
    }
    t[no].out.push_back(pii(qPad++, len));
}

```

```
// Ativar aho-corasick, ajustando funções de falha
void preprocess() {
    int no, v, f, w;
    queue<int> fila;
    for (map<char,int>::iterator it = t[0].lis.begin();
        it != t[0].lis.end(); it++) {
        t[no = it->second].fail = 0;
        t[no].nxt = t[0].out.size() ? 0 : -1;
        fila.push(no);
    }
    while (!fila.empty()) {
        no = fila.front(); fila.pop();
        for (map<char,int>::iterator it = t[no].lis.begin();
            it != t[no].lis.end(); it++) {
            char c = it->first;
            v = it->second;
            fila.push(v);
            f = t[no].fail;
            while (t[f].lis.find(c) == t[f].lis.end()) {
                if (f == 0) { t[0].lis[c] = 0; break; }
                f = t[f].fail;
            }
            w = t[f].lis[c];
            t[v].fail = w;
            t[v].nxt = t[w].out.size() ? w : t[w].nxt;
        }
    }
}

// descomente p/ obter só 1 ocorrência por padrão (+rápido)
// int mark[MAXST];

// Busca em text devolve pares (índice do padrão, posição)
void find(const char *text, vector<pii> &res) {
    int v, no = 0;

    // memset(mark,0,sizeof(mark));
    for (int i = 0; text[i]; i++) {
        while (t[no].lis.find(text[i]) == t[no].lis.end()) {
            if (no == 0) { t[0].lis[text[i]] = 0; break; }
            no = t[no].fail;
        }
        for (v = no = t[no].lis[text[i]]; v != -1; v = t[v].nxt) {
            // if (mark[v]++) break;
            for (int k = 0 ; k < (int)t[v].out.size() ; k++) {
                // encontrado padrao t[v].out[k].first no
                // intervalo (i-t[v].out[k].second+1)..i
                res.push_back(pii(t[v].out[k].first,
                    i - t[v].out[k].second + 1));
            }
        }
    }
}

int main(){
    char text[10010], pat[10010];
    int qpat;

    scanf("%s %d", text, &qpat);
    init();
}
```

```
for (int i=0; i<qpat; i++) {
    scanf("%s",pat);
    add(pat);
}

preprocess();
vector<pii> oc;
find(text, oc);

for (int i=0; i<(int)oc.size(); i++)
    printf("Padrão %d em %d\n", oc[i].first, oc[i].second);
return 0;
}
```

5.2 Algoritmo Z

Autor: paladin8/Micael Carvalho
 Complexidade: $O(n)$
 Tempo de implementacao: 2 min
 Testes: CodeForces.126B
 Descricao: Encontra o tamanho do maior prefixo de S que comeca em S[i], para todo S[i] (resultados em z[i]).

```
#define MAXN 1000000
using namespace std;

/** fill me **/
int z[MAXN];
char s[MAXN+1];
int n;

void run_z() {
    int L = 0, R = 0;
    for (int i = 1; i < n; i++) {
        if (i > R) {
            L = R = i;
            while (R < n && s[R-L] == s[R]) R++;
            z[i] = R-L; R--;
        } else {
            int k = i-L;
            if (z[k] < R-i+1) z[i] = z[k];
            else {
                L = i;
                while (R < n && s[R-L] == s[R]) R++;
                z[i] = R-L; R--;
            }
        }
    }
}
```

```
/* Exemplo simples de uso */
int main() {
    int maxx = 0, res = 0;
    scanf("%s", s);
    n = strlen(s);
    run_z();
    for (int i = 1; i < n; i++) {
        if (z[i] == n-i && maxx >= n-i) { res = n-i; break; }
        maxx = max(maxx, z[i]);
    }
}
```

```
}
if(res > 0) {
    s[res] = '\0';
    printf("%s\n", s);
} else {
    printf("Just a legend\n");
}
}
```

5.3 Array de Sufixos

Autor: Marcelo Galvão Póvoa
 Complexidade: $O(n \lg^2 n)$
 Descricao: Gera o índice de cada sufixo quando ordenados lexicograficamente. A matriz p[i][j] contem os indices para prefixos de sufixos em j de tamanho (1<i) A funcao lcp(i,j) calcula o maior prefixo comum dos suf i e j

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <vector>
using namespace std;
#define MAX 201000
#define LOGM 19

int p[LOGM][MAX],n,ps;
pair< pair<int,int> , int> vp[MAX];
char s[MAX];

//requer precalculo de p[][]
int lcp(int x, int y) {
    int res=0;

    if (x==y) return n-x;

    for (int k=ps-1; k>=0; k--)
        if (p[k][x+res]==p[k][y+res]) {
            res+=(1<k);
            if (x+res>=n || y+res>=n) break;
        }
    return res;
}
```

```
void suffarr() {
    for (int i=0; i<n; i++) p[0][i]=s[i];
    ps=1;

    for (int pow=1, k=0; pow<n; pow*=2, k++) {
        for (int i=0; i<n; i++) {
            int x;
            if (i+pow>=n) x=-1;
            else x=p[k][i+pow];

            vp[i]=make_pair(make_pair(p[k][i],x), i);
        }

        sort(vp,vp+n);
        int id=0;
        p[k+1][vp[0].second]=0;
    }
}
```

```

    for (int i=1; i<n; i++) {
        if (vp[i].first!=vp[i-1].first) id++;
        p[k+1][vp[i].second]=id;
    }
    ps=k+2; //qtde de linhas da tabela
}

int main() {
    scanf(" %d",&n);
    scanf(" %s",s);
    suffarr();

    int res=0;
    vector<pair<int, int> > vp;
    for (int i=0; i<n; i++)
        vp.push_back(make_pair(p[ps-1][i], i));
    sort(vp.begin(), vp.end()); //guarda o suffarr ordenado

    for (int i=0; i<n-1; i++)
        res=max(res,lcp(vp[i].second, vp[i+1].second));
    printf("%d\n",res);
}

```

5.4 Array de Sufixos $n \cdot \lg(n)$

Autor: Douglas Oliveira Santos

Complexidade: $O(n \cdot \lg(n))$

Testes: nuevo-4477

Descricao: Gera o indice de cada sufixo quando ordenados lexicograficamente em $O(n \cdot \lg(n))$. No entanto, só calcula LCP de sufixos adjacentes.

```

#include <cstring>
#include <algorithm>

```

```
using namespace std;
```

```
#define N 150000
```

```

char str[N], inp[N];
int H, Bucket[N], nBucket[N], Rank[N], Height[N], c;

```

```

struct Suffix {
    int idx; // Suffix starts at idx, i.e. it's str[ idx .. L-1 ]
    bool operator<(const Suffix& sfx) const
    // Compares two suffixes based on their first 2H symbols,
    // assuming we know the result for H symbols.
    {
        if(H == 0) return str[idx] < str[sfx.idx];
        else if(Bucket[idx] == Bucket[sfx.idx])
            return (Bucket[idx+H] < Bucket[sfx.idx+H]);
        else
            return (Bucket[idx] < Bucket[sfx.idx]);
    }
    bool operator==(const Suffix& sfx) const {
        return !(*this < sfx) && !(sfx < *this);
    }
} Pos[N];

```

```

int UpdateBuckets(int L) {
    int start = 0, id = 0, c = 0;
    for(int i = 0; i < L; i++) {
        if(i != 0 && !(Pos[i] == Pos[i-1])) {
            start = i;
            id++;
        }
        if(i != start)
            c = 1;
        nBucket[Pos[i].idx] = id;
    }
    memcpy(Bucket, nBucket, 4 * L);
    return c;
}

```

```

void SuffixSort(int L) {
    H = 0;
    for(int i = 0; i < L; i++) Pos[i].idx = i;
    sort(Pos, Pos + L);
    c = UpdateBuckets(L);
    for(H=1;c;H *= 2) {
        // Sort based on first 2*H symbols, assuming
        // that we have sorted based on first H character
        sort(Pos, Pos+L);
        // Update Buckets based on first 2*H symbols
        c = UpdateBuckets(L);
    }
}

```

// Must compute the suffix array Pos first

```

void ComputeLCP(int L) {
    for (int i = 0; i < L; i++) Rank[Pos[i].idx] = i;
    int h = 0;
    for (int i = 0; i < L; i++)
        if (Rank[i] > 0) {
            int k = Pos[Rank[i] - 1].idx;
            while (str[i+h] == str[k+h])
                ++h;
            Height[Rank[i]] = h;
            if (h > 0) --h;
        }
}

```

```

int main() {
    scanf("%s",str);

    /* e necessario colocar o tamanho + 1 */
    int n = strlen(str) + 1;
    SuffixSort(n);

    ComputeLCP(n);

    /* Pos[i].idx guarda a posicao na string original */
    for (int i = 0; i < n; i++) {
        printf("%d\n", Pos[i].idx);
    }

    /* Height[i] tem o LCP da posicao i com a posicao i-1 */
    for (int i = 0; i < n; i++) {
        printf("%d\n", Height[i]);
    }
}

```

```

}

return 0;
}

```

5.5 Árvore de Sufixos

```

#define FOR(i,n) for(int i = 0; i < n; i++)
#define MAXL 27
#define cor(x) (str[x] - 'a'+1)

```

```

class Stree {
public:
    int n,l,r;
    Stree *ch[MAXL], *slink;
    string str;
    Stree(int i = 0, int f = 0):l(i),r(f) {
        slink = NULL;
        memset(ch,0,sizeof(ch));
    }
    ~Stree() {
        FOR(i,MAXL) if (ch[i]) delete ch[i];
    }
    void canonize(Stree *&anode, int &al, int ar) {
        Stree *next;
        while (al <= ar) {
            next = anode->ch[cor(al)];
            if (next->r - next->l > ar - al) break;
            al += next->r - next->l+1; anode = next;
        }
    }
    bool testsplit(Stree *&anode, int al,
                    int ar, char t, Stree *&mid) {
        if (al > ar) {
            mid = anode;
            return anode->ch[cor(al)] != NULL;
        }
        Stree *next = anode->ch[cor(al)];
        int p = ar - al + next->l+1;
        if (t == cor(p)) return true;
        mid = new Stree(next->l,p-1);
        next->l = p; mid->ch[cor(p)] = next;
        anode->ch[cor(al)] = mid;
        return false;
    }
    void update(Stree *&anode, int &al, int ar) {
        Stree *old = this, *mid;
        while(!testsplit(anode,al,ar-1,cor(ar),mid)) {
            mid->ch[cor(ar)] = new Stree(ar,n-1);
            if (old != this) old->slink = mid;
            old = mid; anode = anode->slink;
            canonize(anode,al,ar-1);
        }
        if (old != this) old->slink = anode;
    }
    void buildtree(string &str) {
        int al, ar; //active node
        Stree *anode = this;
        this->str = str;
    }
}

```



```

    n = str.size(); l=-1,r=-1;
    Stree dummy;
    FOR(i,MAXL) dummy.ch[i] = this;
    slink = &dummy;
    for (al = ar = 0; ar < n; ar++) {
        update(anode,al,ar);
        canonize(anode,al,ar);
    }
    memset(dummy.ch,0,sizeof(dummy.ch));
}
};

int longest, last;
int n,m;

int dfs(Stree *t, int deep) {
    int sz = min(t->r,n-1) - t->l + 1;
    if (t->r == n && sz == 0) return 1;
    if (t->l != -1) deep += sz;
    int freq = 0;
    if (t->r == n) freq++;
    FOR(i,MAXL) {
        if (t->ch[i]) freq += dfs(t->ch[i],deep);
    }
    int start = min(n-1, t->r) - deep+2;
    if (t->l != -1 && freq >= m && (deep > longest ||
        (deep == longest && start < last))) {
        longest = deep;
        last = start;
    }
    return freq;
}

int main(int argv, char *argv[]) {
    string str;
    while(scanf("%d",&m) == 1 && m) {
        Stree t;
        cin >> str;
        reverse(str.begin(),str.end());
        str += 'a'-1;
        n = str.size()-1;
        t.buildtree(str);
        longest = last = 0;
        dfs(&t,0);
        if (longest == 0) printf("none\n");
        else printf("%d %d\n",longest,n-longest-last+1);
    }
}

```

5.6 Busca de Strings (KMP)

Autor: NU 2/Marcelo Galvão Póvoa

Complexidade: $O(n \cdot m)$

Tempo de implementacao: 3 min

Testes: SPOJ.NHAY

Descricao: Acha todas as ocorrencias do padrao p num texto t

```
#define MAXNP 1000
```

```
int b[MAXNP+1];
```

```

void buildb(char *p, int psize) {
    int i = 0, j = b[0] = -1;
    while(i < psize) {
        while (j >= 0 && p[i] != p[j]) j = b[j];
        b[++i] = ++j;
    }
}

void kmp(char *text, char *pattern) {
    int i = -1, j = 0, m = strlen(pattern), n = strlen(text);
    buildb(pattern, m);
    while(++i < n) {
        while (j >= 0 && pattern[j] != text[i]) j = b[j];
        if (++j >= m){
            j = b[j];
            printf("Achou em %d\n", i-m+1);
        }
    }
}

/***** Exemplo simples de uso *****/
int main(){
    char p[MAXNP+1], t[100000];
    while(scanf("%s",p)==1) {
        scanf("%s",t);
        kmp(t, p);
    }
    return 0;
}

```

5.7 Hash de Strings

Autor: Douglas Santos / Marcelo Póvoa

Complexidade: $O(n)$

Testes: Codeforces.7D, uva 12494, sgu 439

Descricao: Após preprocessar uma string, calcula o hash de qualquer substring sua em tempo constante.

```

#include <algorithm>
#define MAXN 100100
#define B 33
using namespace std;
typedef unsigned long long hash;

```

```

hash h[MAXN], pwr[MAXN];
char s[MAXN];

```

```

void gen(char *s) {
    h[0] = 0;
    pwr[0] = 1;
    for (int i = 0; s[i]; i++) {
        h[i+1] = h[i] * B + s[i];
        pwr[i+1] = pwr[i] * B;
    }
}

```

```

// Calcula o hash da substring s[a..b]
hash sect(int a, int b) {
    if (a > b) return 0;

```

```

    return h[b+1] - h[a] * pwr[b - a + 1];
}

// Maior prefixo comum das substrings s[a..n-1], s[b..n-1]
int lcp(int a, int b, int n) {
    int es = 0, di = min(n-b, n-a);

    while (es < di) {
        int me = (es+di+1)/2;
        if (sect(a, a+me-1) == sect(b, b+me-1)) es = me;
        else di = me-1;
    }
    return es;
}

/***** Exemplo simples de uso *****/
int main() {
    while (scanf("%s",s)==1) {
        gen(s);
        printf("%llu\n", sect(0, strlen(s) - 1));
    }
    return 0;
}

```

5.8 Split

Autor: Davi Costa

Complexidade: $O(n)$

Tempo de implementacao: 30s

Testes: nuevo-4427

Descricao: Divide uma sentenca utilizando delimitadores.

Delimitadores duplos sao ignorados e nunca aparece no vetor retornado strings vazias.

```

#include <cstring>
#include <vector>
#include <string>
#include <iostream>

```

```
using namespace std;
```

```

#define pb push_back
typedef vector<string> vs;

```

```

//Espaco eh usado como padrao
vs split(string ss, string delim = " ") {
    vs buf;
    char *s = (char *)ss.c_str();
    char *d = (char *)delim.c_str();
    char *p = strtok(s,d);
    while (p) {
        buf.pb((const char *)p);
        p = strtok(0,d);
    }
    return buf;
}

```

```

/***** Exemplo simples de uso *****/
int main() {
    char buf[1000];

```

```

while (scanf("%[^\n]s",buf)) {
    getchar(); //Nao esquecer
    if (buf[0] == '\0') break;
    /*Usa espaço ou $ como delimitadores de palavras*/
    vs v = split(buf,"$ ");
    for (int i = 0; i < v.size(); i++)
        cout << v[i] << " ";
    cout << endl;
}
return 0;
}

```

6 Miscelânea

6.1 Árvore de Intervalos

Autor: Marcelo Galvão Póvoa

Complexidade: $O(\lg n)$ por update/query

Testes: SPOJ.QTREE

Descricao: Modelo de segtree que deve ser adaptado ao problema desejado. Suporta query em intervalo e update em ponto. O código abaixo é de RMQ, ou seja, dá o máximo elemento em um intervalo

```

#include <cstring>
#include <algorithm>
#define MAXN 100100
using namespace std;

```

```
int t[4*MAXN];
```

```

/* Obtém o RMQ em [a,b]; chamar com root=0, rl=0, rr=n-1 */
int query(int root, int rl, int rr, int a, int b) {
    if (a > b) return 0;
    if (rl==a && rr==b) return t[root];

```

```

    int rm = (rl+rr)/2;
    return max(query(2*root+1, rl, rm, a, min(b, rm)),
               query(2*root+2, rm+1, rr, max(a, rm+1), b));
}

```

```

/* Muda posição x para vx; chamar com root=0, rl=0, rr=n-1 */
void update(int root, int rl, int rr, int x, int vx) {
    if (rl==rr) t[root] = vx;
    else {
        int rm = (rl+rr)/2;
        if (x <= rm) update(2*root+1, rl, rm, x, vx);
        else update(2*root+2, rm+1, rr, x, vx);

        t[root] = max(t[2*root+1], t[2*root+2]);
    }
}

```

```

/**** Exemplo simples de uso ****/
int main() {

```

```
    int n,c,op,x,y;
```

```

    while (scanf("%d%d",&n,&c)==2) {
        memset(t, 0, sizeof(t));

```

```
        while (c--) {
```

```

            scanf("%d %d %d",&op,&x,&y);
            if (op==0) update(0, 0, n-1, x, y);
            else printf("%d\n", query(0, 0, n-1, x, y));
        }
    }
    return 0;
}

```

6.2 Árvore de Intervalos (c/ Lazy Propagation)

Autor: Marcelo Galvão Póvoa

Complexidade: $O(\lg n)$ por update/query

Testes: SPOJ.LITE, SPOJ.MULTQ3, SPOJ.GSS3

Descricao: Modelo de segtree que deve ser adaptado ao problema desejado. Suporta query e update em ponto ou intervalo. O código abaixo representa um vetor de bits com update(a,b) sendo "toggle bits entre a e b" e query(a,b) "quantos bits 1 entre a e b"

```

#include <algorithm>
#define MAXN 100100
using namespace std;

```

```

struct tr {
    int ql; /* Qtd de bits 1 no intervalo */
    int sz; /* Tam do intervalo */
    int prop; /* Qtd de updates a propagar nos filhos */
} tree[4*MAXN];

```

```

/* Aplica q vezes o update no nó t (q pode ser zero) */
void apply(tr &t, int q) {
    if (q % 2) t.ql = t.sz - t.ql;
}

```

```

/* Faz x updates (chamando acc=up=x) e retorna a query a
 * partir da sub-árvore root = [rl,rr] no intervalo [a,b]
 * Para obter apenas a query, use acc=up=0 */
int go(int root, int rl, int rr,
       int a, int b, int acc, int up) {
    /* acc = acúmulo de updates dos pais mais o up original */
    /* devemos aplicar acc updates na sub-árvore */
    tree[root].prop+=acc;

```

```

    if (a > b) { /* [a,b] não está nesse nó raiz */
        /* aplica na raiz e agenda para os filhos
         * apenas os updates dos pais (sem up) */
        tree[root].prop -= up;
        apply(tree[root], acc - up);
        return 0; /* elemento nulo */
    }

```

```

    if (rl == a && rr == b) { /* [a,b] == nó raiz */
        /* basta aplicar updates na raiz e devolvê-la
         * a propagação será feita posteriormente */
        apply(tree[root], acc);
        return tree[root].ql;
    }

```

```

    int rm = (rl + rr) / 2;
    int ls = 2*root + 1, rs = 2*root + 2;

```

```
    /* res = a combinacao das queries dos filhos (p ex soma) */
```

```

    int res = go(ls, rl, rm, a, min(b,rm), tree[root].prop, up)
    + go(rs, rm + 1, rr, max(a,rm+1), b, tree[root].prop, up);

```

```

    /* nova raiz é a combinação dos filhos atualizados */
    tree[root].ql = tree[ls].ql + tree[rs].ql;
    tree[root].prop=0; /* propagação feita na raiz */
    return res;
}

```

```

/* Inicializar árvore (ou pode usar memset se tudo for 0) */
void init(int root, int rl, int rr) {
    tree[root].ql = 0;
    tree[root].sz = rr-rl+1;
    tree[root].prop = 0;

```

```

    if (rl < rr) {
        int rm = (rl+rr) / 2;
        init(2*root+1, rl, rm);
        init(2*root+2, rm+1, rr);
    }
}

```

```

/**** Exemplo simples de uso ****/
int main() {

```

```
    int n,c;
```

```

    while (scanf("%d%d",&n,&c)==2) {
        init(0,0,n-1);
        for (int i=0; i<c; i++) {
            int op, p, q;
            scanf("%d%d%d",&op,&p,&q);
            p--; q--;
            if (!op) /* update +1 em [p,q] */
                go(0,0,n-1,p,q,1,1);
            else /* query [p,q] */
                printf("%d\n", go(0,0,n-1,p,q,0,0));
        }
    }
    return 0;
}

```

6.3 BigInteger em Java

Autor: Douglas Oliveira Santos

Complexidade: --

Testes: uva10176, uva10183, uva10334, uva113
uva424, uva495, uva763

Dependencias: Nenhuma

Descricao: Operações com BigInteger em Java

```

import java.io.*;
import java.math.BigInteger;
import java.util.*;

```

```

public class BigInt {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        BigInteger res;
        res = BigInteger.valueOf(0);
        BigInteger a, b;

```

```

a = sc.nextBigInteger();
b = sc.nextBigInteger();

res = a.add(b); // a + b
res = a.subtract(b); // a - b
res = a.multiply(b); // a * b
res = a.divide(b); // a / b
res = a.max(b); // a / b
res = a.abs(); // abs(a)
res = a.mod(b); // a mod b
}
/* OBS: Scanner tambem pode ser usado
 * para ler int (sc.nextInt) e double (sc.nextDouble), */
}

```

6.4 Binary Indexed Tree/Fenwick Tree

Autor: Marcelo Galvão Póvoa
 Complexidade: $O(\lg \text{MAX})$ - atualizacao e consulta
 Testes: SPOJ.ORDERS, SPOJ.INCDSEQ
 Descricao: Dada um vetor inicial vazio, eh possivel fazer incremento no conteudo v[x] e consultar a soma do subvetor v[0..x] de forma eficiente. Para o calculo da soma de um subvetor qualquer usa-se a relacao $\text{sum}(a..b) = \text{sum}(1..b) - \text{sum}(1..a-1)$

Observacao: Zerar o vetor tree[] antes de utilizar

```

#define MAXN 1000

int tree[MAXN+1];

int query(int x) {
    int sum=0;

    for (x++; x>0; x=x & (-x))
        sum+=tree[x];

    return sum;
}

/*by representa um inc/decremento em x*/
void update(int x, int by) {
    if (x<0) return;

    for (x++; x<=MAXN; x+=x & (-x))
        tree[x]+=by;
}

/**** Exemplo simples de uso ****/
#include <cstdio>
#include <cstring>

int main(){
    memset(tree,0,sizeof(tree));

    update(3,2);
    update(2,-1);

```

```

printf("%d\n",query(10));
return 0;
}

```

6.5 Binary Indexed Tree/Fenwick Tree 2D

Autor: Marcelo Galvão Póvoa
 Complexidade: $O(\lg^2 \text{MAX})$ - atualizacao e consulta
 Tempo de implementacao: 3 min
 Testes: SPOJ.MATSUM
 Descricao: Dada uma matriz inicial vazia, eh possivel fazer incremento no conteudo m[x,y] e consultar a soma da submatriz m[1..x,1..y] de forma eficiente. Para o calculo da soma de uma submatriz qualquer usa-se a relacao $\text{sum}(a..b,c..d) = \text{sum}(1..b,1..d) + \text{sum}(1..a-1,1..c-1) - \text{sum}(1..b,1..c-1) - \text{sum}(1..a-1,1..d)$

Observacao: Zerar a matriz tree[][] antes de utilizar

```

#include <stdio.h>
#include <string.h>
#define MAX 1000

int tree[MAX+1][MAX+1]; /*nao usar posicao em 0*/

int query(int x, int y) {
    int sum=0,yy=y;
    if (x==0 || y==0) return 0;

    while (x) {
        while (y) {
            sum+=tree[x][y];
            y=y & (-y);
        }
        x=x & (-x);
        yy=yy;
    }
    return sum;
}

/*v representa um inc/decremento em x,y!*/
void update(int x, int y, int v) {
    int yy=y;
    if (x==0 || y==0) return;

    while (x<=MAX) {
        while (y<=MAX) {
            tree[x][y]+=v;
            y+=y & (-y);
        }
        x+=x & (-x);
        yy=yy;
    }
}

/**** Exemplo simples de uso ****/
int main(){
    memset(tree,0,sizeof(tree));

    update(3,5,2);

```

```

update(2,3,-1);

printf("%d\n",query(10,10));
return 0;
}

```

6.6 Binary Indexed Tree with range update

Autor: Douglas Oliveira Santos
 Complexidade: $O(\lg n)$ - atualizacao e consulta
 Testes: poj-3468
 Descricao: Dada um vetor inicial vazio, eh possivel fazer incremento no conteudo v[a, b] e consultar a soma do subvetor v[0..x] de forma eficiente. Para soma acumulada do intervalo [a, b], utilize $\text{query}(b) - \text{query}(a-1)$

Observacao: Zerar o vetor tr[] antes de utilizar
 Pode usar o índice 0

```

typedef long long int int64;
#define MAXN 100010

int64 tr[MAXN][2];
int n;

/* Função interna, não deve ser chamada externamente */
void up(int at, int64 mul, int64 add) {
    while (at < n) {
        tr[at][0] += mul;
        tr[at][1] += add;
        at |= (at + 1);
    }
}

/* Soma by em todos elementos no intervalo [a, b] */
void update(int a, int b, int64 by) {
    up(a, by, -by * (a-1));
    up(b, -by, by * b);
}

/* Retorna a soma acumulada do intervalo [0, at] */
int64 query(int at) {
    int64 mul = 0;
    int64 add = 0;
    int s = at;
    while (at >= 0) {
        mul += tr[at][0];
        add += tr[at][1];
        at = (at & (at + 1)) - 1;
    }
    return mul * s + add;
}

/**** Exemplo simples de uso ****/
#include <cstdio>
#include <cstring>

```

```
int main(){
    n = 10;
    memset(tr,0,sizeof(tr));

    /* soma 10 no intervalo [0, 5] */
    update(0, 5, 10);
    /* soma 2 no intervalo [6, 6] */
    update(6, 6, 2);
    /* imprime soma acumulada [0, 6] */
    printf("%lld\n",query(6));
    /* imprime soma acumulada [1, 6] */
    printf("%lld\n",query(6) - query(0));
    return 0;
}
```

6.7 Calculador de Expressões

Autor: Alexandre Kunieda

Complexidade: $O(n)$

Tempo de implementacao: 4 minutos

Testes:

- SPOJbr Calculadora (n<=?) t=0.00s

Descricao:

Calcula expressoes que envolvam parenteses e operacoes binarias de +, -, *, /, alem de operacao unaria de - e variaveis de nome longo

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <map>
#include <string>
#include <algorithm>
```

```
using namespace std;
```

```
/* tamanho maximo que as variaveis/numeros assumem */
#define MAXvar 10
```

```
char expr[1000];
```

```
map<string,double> val;
char seq[MAXvar];
char *l;
```

```
double solve();
double read() {
    int n;
```

```
    while(isspace(*l)) l++;
    if(*l=='-'){
        l++;
        return -read();
    }
    else if(*l=='('){
        l++;
        return solve();
    }
    else{
        sscanf(l," %[a-zA-Z0-9]\n",seq,&n);
```

```
        l+=n;
        if( isalpha(seq[0]) ) return val[ string(seq) ];
        else return atof(seq);
    }
}
```

```
double solve(){
    char op;
    double acc=0,ult,x;

    ult=read();

    while(*l!='\0'){
        while(isspace(*l)) l++;
        op = *(l++);
        if(op=='+' || op=='\0') break;
```

```
        x = read();
```

```
        if (op=='+') { acc+=ult; ult=x; }
        if (op=='-') { acc+=ult; ult=-x; }
        if (op=='*') ult*=x;
        if (op=='/') ult/=x;
```

```
    }
```

```
    return acc+ult;
}
```

```
/* Exemplo de uso */
```

```
int main(){
    int i;
```

```
    while(gets(expr)!=NULL){
        /* busca por um caracter '=' na string expr[] */
        for(i=0 ; expr[i]!='\0' && expr[i]!='=' ; i++);

        /* caso tenha achado caracter '=', seta a variavel */
        if(expr[i]=='='){
            l=expr+i+1; expr[i]='\0';
            val[ string(expr) ]=solve();
        }
        /* senao, apenas calcula a expressao */
        else{
            l = expr;
            printf("%.2lf\n", solve());
        }
    }
```

```
    return 0;
}
```

6.8 Convex Hull Trick

Autor: Douglas Oliveira Santos

Complexidade: $O(\lg n)$ - insert e query

Testes: LA5133, LA6131, SPOJ.ACQUIRE, SPOJ.API010A

Descrição: Estrutura de dados que permite inserir retas não-verticais na forma $y(x)=A*x+B$ e consultar, dado x , qual é o maior $y(x)$ entre as retas existentes. Para se obter o menor no lugar do maior, troque os sinais de A , B e do resultado da query.

Funciona para int e double.

```
#include <set>
#include <algorithm>
#include <cmath>
using namespace std;
```

```
// Caso queira double, troque as 2 linhas abaixo
typedef long long int T;
#define INF 0x3f3f3f3f3f3f3fLL
```

```
struct line {
    T a, b, xmax;
    line(T a, T b) : a(a), b(b) {};
};
```

```
bool compa(line a, line b) {
    return a.a < b.a;
}
```

```
bool compx(line a, line b) {
    return a.xmax < b.xmax;
}
```

```
bool(*fcompa)(line,line) = compa;
bool(*fcompx)(line,line) = compx;
```

```
set<line, bool(*)>(line, line)> sa(fcompa);
set<line, bool(*)>(line, line)> sx(fcompx);
```

```
set<line, bool(*)>(line, line)>::iterator it;
```

```
/* Funcao auxiliar */
```

```
void add(line r) {
    sa.insert(r);
    sx.insert(r);
}
```

```
/* Funcao auxiliar */
```

```
void remove(line r) {
    sa.erase(r);
    sx.erase(r);
}
```

```
/* Funcao auxiliar */
```

```
T getMax(line r, line s) {
    //return (s.b - r.b) / (r.a - s.a); // para double
    return floor((double) (s.b - r.b) / (double) (r.a - s.a));
}
```

```
/* Funcao auxiliar */
```

```
T gety(line r, T x) {
    return r.a * x + r.b;
}
```

```
void init() {
    sa.clear();
    sx.clear();
}
```

```

T query(T x) {
    if (sx.empty()) return -INF;
    line r(0, 0);
    r.xmax = x;
    it = sx.lower_bound(r);
    return gety(*it, x);
}

bool insert(T a, T b) {
    line r(a, b);
    it = sa.lower_bound(r);
    if (it != sa.end() && it->a == r.a) {
        if (it->b >= r.b) return false;
        remove(*it);
    }
    it = sa.lower_bound(r);
    if (it != sa.end() && it != sa.begin()) {
        line s = *it;
        it--;
        if (getxMax(r, s) <= getxMax(r, *it)) return false;
    }
    while (1) {
        it = sa.lower_bound(r);
        if (it == sa.end()) break;
        if (getxMax(r, *it) >= it->xmax) {
            remove(*it);
        }
        else {
            break;
        }
    }
    while (1) {
        it = sa.lower_bound(r);
        if (it == sa.begin()) break;
        it--;
        line s = *it;
        if (it == sa.begin()) {
            remove(s);
            s.xmax = getxMax(s, r);
            add(s);
            break;
        }
        it--;
        line t = *it;
        remove(s);
        if (getxMax(s, r) > t.xmax) {
            s.xmax = getxMax(s, r);
            add(s);
            break;
        }
    }
    it = sa.lower_bound(r);
    if (it == sa.end()) r.xmax = INF;
    else r.xmax = getxMax(r, *it);
    add(r);
    return true;
}

/**** Exemplo simples de uso ****/
int main() {

```

```

init();
insert(0, -10);
insert(-2, -0);
insert(-1, -7);
insert(-2, -3);
insert(-2, -2);
printf("%lld\n", query(5));
return 0;
}

```

6.9 Decomposição Heavy Light

Autor: Marcelo Galvão Póvoa
 Complexidade: $O(\lg n)$ LCA / $O(\lg^2 n)$ queries
 Testes: SPOJ.QTREE, uva.12424
 Descrição: Particiona os vértices de uma árvore em chains (sequência de vértices ancestrais) de modo que qualquer caminho usa um número logarítmico de chains, que podem ser incrementadas para responder queries em caminhos (ver ex.)

```

#include <vector>
using namespace std;
#define MAXN 100100

vector<int> g[MAXN];
/* Vértice do topo da chain i, tam da chain i e qtd delas */
int head[MAXN], chsz[MAXN], nch;
/* Chain do vértice i e seu índice nela (cresce pra raiz) */
int chain[MAXN], chidx[MAXN];
/* Altura do vértice i, seu antecessor e tam da subárvore */
int depth[MAXN], pai[MAXN], size[MAXN];

/* Adiciona um vértice v no topo da chain c */
void chadd(int v, int c) {
    chidx[v] = chsz[c]++;
    chain[v] = c;
    head[c] = v;
}

/* Gera as chains e vetores associados */
void dfshl(int x) {
    size[x] = 1;

    for (int i = 0; i < g[x].size(); i++) {
        int v = g[x][i];
        if (pai[x] != v) {
            depth[v] = depth[x] + 1;
            pai[v] = x;

            dfshl(v);
            size[x] += size[v];
        }
    }

    chain[x] = -1;
    for (int i = 0; i < g[x].size(); i++)
        if (g[x][i] != pai[x] && size[g[x][i]] > size[x] / 2)
            chadd(x, chain[g[x][i]]);

    if (chain[x] == -1) chadd(x, nch++);
}

```

```

}

/* Exemplo de LCA. Percorre as chains no caminho entre a e b
Pode ser alterado para responder query usando uma estrutura
de dados de intervalos por chain (por ex. BIT, segtree) */
int lca(int a, int b) {
    while (chain[a] != chain[b]) {
        if (depth[head[chain[a]]] > depth[head[chain[b]]])
            // query chain[a] em [chidx[a], chsz[chain[a]]-1]
            a = pai[head[chain[a]]];
        else
            // query chain[b] em [chidx[b], chsz[chain[b]]-1]
            b = pai[head[chain[b]]];
    }

    if (depth[a] < depth[b]) {
        // query chain[a] em [chidx[b], chidx[a]]
        return a;
    }
    // query chain[a] em [chidx[a], chidx[b]]
    return b;
}

/**** Exemplo simples de uso ****/
int main() {
    int n, m, a, b;

    while (scanf("%d%d", &n, &m) == 2) {
        nch = 0;
        memset(chsz, 0, sizeof(chsz));

        for (int i = 0; i < n; i++)
            g[i].clear();

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

        dfshl(0);

        while (m--) {
            scanf("%d%d", &a, &b);
            printf("%d\n", lca(a, b));
        }
        return 0;
    }
}

```

6.10 Funções para Datas

Autor: Alexandre Kunieda/Marcelo Póvoa
 Complexidade: $O(1)$
 Testes: uva.602/nuevo.4306

Zeller: Eh capaz de calcular o dia da semana para o calendario gregoriano (atual) - chame zeller() -, ou calendario juliano (antigo, considerava bissexto todo ano multiplo de 4, sem as regras de multiplo de 100 e 400) - chame zeller_julian().

Getdate: Retorna o numero de dias a partir do ano 0 ate a data

```
bool bissex(int y) { return (y%4==0 && (y%100 || y%400==0)); }
```

```
int zeller(int d, int m, int y) {
    if(m<3) --y, m+=12;
    return (d + ((m+1)*13)/5 + y + y/4 +
            6*(y/100) + y/400 + 6) % 7;
}
```

```
int zeller_julian(int d, int m, int y) {
    if(m<3) --y, m+=12;
    return (d + ((m+1)*13)/5 + y + y/4 + 4) % 7;
}
```

```
int getdate(int d, int m, int y) { //mes e dia a partir de 1
    int qm[]={0,31,28,31,30,31,30,31,31,30,31,30,31};
    int s=0;
    for (int i=1; i<m; i++) s+=qm[i];

    int res=365*y+s+d+(y/4-y/100+y/400);
    if (m<3 && bissex(y)) res--;
    return res;
}
```

```
/***** Exemplo simples de uso *****/
#include <stdio>
```

```
char wkday[]="DSTQQSS";
```

```
int main() {
    int d,m,y;

    while(scanf(" %d %d %d", &d, &m, &y)==3) {
        printf("%d/%d/%d is a %c\n", d,m,y,wkday[zeller(d,m,y)]);
        printf("%d/%d/%d = %d days\n", d,m,y,getdate(d,m,y));
    }

    return 0;
}
```

6.11 Knight Distance

Autor: Alexandre Kunieda + TopCoder Forum

Complexidade: O(1)

Testes: uva.11643, uva.439, tju.1736

Descricao: Determina em O(1) a distância (em movimentos de cavalo) entre 2 pontos de um tabuleiro (infinito ou finito). Se o tabuleiro for finito, deve ter tamanho n x m com n >= 4 e m >= 4.

```
#include <algorithm>
using namespace std;
```

```
int knightdist_inf(int x1, int y1, int x2, int y2) {
    int dx=abs(x2-x1);
    int dy=abs(y2-y1);
    if (abs(dx)==1 && dy==0) return 3;
    if (abs(dy)==1 && dx==0) return 3;
    if (abs(dx)==2 && abs(dy)==2) return 4;
```

```
    int lb=max((dx+1)/2, (dy+1)/2);
    lb = max(lb, (dx+dy+2)/3);
    if ((lb%2)!=(dx+dy)%2) lb++;
    return lb;
}
```

```
int n,m; //tamanho do tabuleiro
```

```
int knightdist(int x1, int y1, int x2, int y2) {
    if(x1==n || x2==n) {
        x1 = n+1 - x1;
        x2 = n+1 - x2;
    }
    if(y1==m || y2==m) {
        y1 = m+1 - y1;
        y2 = m+1 - y2;
    }
    if((x1==1 && y1==1) || (x2==1 && y2==1)) {
        int a=abs(x1-x2), b=abs(y1-y2);
        if(a==0 && b==3 && m==4) return 5;
        if(b==0 && a==3 && n==4) return 5;
        if(a==1 && b==1) return 4;
    }
    return knightdist_inf(x1,y1,x2,y2);
}
```

```
/***** Exemplo simples de uso *****/
#include <stdio>
```

```
int main() {
    n = m = 4;

    //as coordenadas estão indexadas em 1
    printf("%d\n", knightdist(1,1, 3,3));
    printf("%d\n", knightdist(1,1, 4,1));

    return 0;
}
```

6.12 Maior Retângulo em um Histograma

Autor: Marcelo Galvão Póvoa

Complexidade: O(n)

Testes: SPOJ.HISTOGRA

Descricao: Dado um vetor que contem alturas (>=0) das barras de um histograma de largura fixa = 1, calcula a área do maior retangulo contido no histograma

```
#include <algorithm>
#define MAX 100100
using namespace std;

int sh[MAX], sp[MAX];
```

```
long long histogram(int *v, int n) {
    int qs=1, curh=0;
    long long res=0;

    sh[0]=-1; sp[0]=0;
```

```
v[n]=-1;
```

```
for (int i=0; i<n+1; i++) {
    if (i<n && v[i]>=curh) {
        sh[qs]=v[i];
        sp[qs++]=i;
    }
    else {
        while (sh[qs-1]>v[i]) {
            qs--;
            res=max(res, (long long) sh[qs]*(i-sp[qs]));
        }
        sh[qs++]=v[i];
    }
    curh=v[i];
}

return res;
}
```

```
/***** Exemplo simples de uso *****/
#include <stdio>
```

```
int main() {
    int n;
    int v[MAX];

    while (scanf("%d",&n)==1 && n) {
        for (int i=0; i<n; i++)
            scanf(" %d", &v[i]);

        printf("%lld\n",histogram(v,n));
    }
    return 0;
}
```

6.13 Operações com Bits

Autor: Guilherme Kunigami

Complexidade:

Tempo de implementacao: 7 min

Testes:

- TCCC 2006, Round 1B Medium
- TCO 2006, Round 1 Easy
- SRM 320, Division 1 Hard

```
#include <string>
#include <stdlib.h>
using namespace std;
```

```
void print_bit(int a, int k){
    for (int i=0, j = (1 << (k-1)); i<k; i++){
        printf("%d", (j&a)?1:0);
        a <<= 1;
    }
    printf("\n");
}

int bitmask(string s){
    return strtol(s.c_str(), NULL, 2);
}
```

```

/* Operacoes entre as mascaras de bits a e b, de tamanho k */
void operacoes_bits(int a, int b, int k){

    print_bit(a, k);
    print_bit(b, k);

    int ALL_BITS = (1 << k) - 1;
    /* bit = {0,...,k-1} */
    int bit = 6;

    print_bit(ALL_BITS, k);

    /* Uniao */
    print_bit(a | b, k);
    /* Intersecao */    a & b;
    print_bit(a & b, k);
    /* Subtracao */    a & ~b;
    print_bit(a & ~b, k);
    /* Negacao */
    print_bit(ALL_BITS ^ a, 32);
    /* Limpar o bit setado menos significativo */
    print_bit(a & (a - 1), k);
    /* Seta o 4º bit-esimo menos significativo */
    print_bit(a |= 1 << bit, k);
    /* Limpar o bit-esimo menos significativo */
    print_bit(a &= ~(1 << bit), k);
    /* Testar o bit-esimo menos significativo */
    if (a & 1 << bit)
        printf("Bit setado\n");
    else
        printf("Bit nao setado\n");
}

/* Itera sobre todos os subconjuntos do conjunto
 * representado pela mascara mask. Gera os subconjuntos
 * maiores (em valor de mascara) primeiro */
void subset(int mask, int k){
    for (int i = mask; i > 0; i = (i - 1) & mask)
        print_bit(i, k);
    print_bit(0, k);
}

/* Gera todas as mascaras de tamanho n, com m bits setados */
void comb(int dep, int from, int mask, int n, int m) {
    if (dep == m){
        print_bit(mask, n);
        return;
    }
    // Seta o i-esimo bit e desce
    for (int i = from; i < n; i++)
        comb(dep+1, i+1, mask | (1<<i), n, m);
}

/* Funcoes de bits do gcc */
void builtin(int mask){
    printf("# Bits setados: %d\n", __builtin_popcount(mask));
    //PS.: Depende do tipo do numero!
    printf("# Zeros no comeco: %d\n", __builtin_clz(mask));
    printf("# Zeros no final: %d\n", __builtin_ctz(mask));
}

```

```

*** Exemplo Simples de uso ***
int main (){
    printf("Principais operacoes de bits\n");
    operacoes_bits(389, 454, 10);

    printf("Gera todos os subconjuntos de 1000110101:\n");
    subset(bitmask("1000110101"), 10);

    printf("Todas as combinacoes 10 escolhe 3:\n");
    comb(0, 0, 0, 10, 3);

    printf("Teste das funcoes builtin do GCC");
    builtin(bitmask("011101011000"));

    return 0;
}

```

6.14 Operações com Matriz de Bits

Autor: Davi Costa

Complexidade:

Tempo de implementacao: 5~7 min

Testes: uva-11862

Descricao: Suporta todas as operacoes comuns de matrizes utilizando um vetor de inteiros. OBS: A coluna 0 é representada na direita (bit menos significativo) e nao na esquerda como estamos acostumados.

```

#include <stdio>
#include <cstring>
using namespace std;

```

```
#define MAX 32
```

```
typedef unsigned int ui;
```

```

class bm {
public:
    ui n, m;
    ui t[MAX];
    bm(ui nn, ui mm) {
        n = nn;
        m = mm;
        for (ui i = 0; i < nn; i++) t[i] = 0;
    }
    bm & operator=(const bm &b) { //copia
        n = b.n;
        m = b.m;
        for (ui i = 0; i < n; i++) t[i] = b.t[i];
        return *this;
    }
    ui& operator [] (ui r) { return t[r]; } //pega toda linha r
    //pega o bit r,c
    bool operator () (ui r, ui c) {return (t[r] & (1<<c)) != 0;}
    void set(ui r, ui c) { t[r] |= 1<<c; } //seta bit r,c
    void clear(ui r, ui c) { t[r] &= ~(1<<c); } //limpa bit r,c
    bm transp() { //transpoe uma matriz
        bm res(m,n);

```

```

        for (ui i = 0; i < n; i++) {
            ui a = t[i];
            for (int j = m-1; j >= 0; j--, a>>=1)
                if (a&1) res.set(j,n-i-1);
        }
        return res;
    }
    bm operator * (bm &m2) { //multiplica 2 matrizes
        bm trans = m2.transp();
        ui m = m2.m; bm res(n,m);
        for (ui i = 0; i < n; i++)
            for (ui j = 0; j < m; j++) {
                ui r = __builtin_popcount(t[i]&trans[j])%2;
                if (r) res.set(i,m-j-1);
            }
        return res;
    }
    ui mul2(ui a) { //multiplica matriz por um vetor Mx1
        ui res = 0;
        for (ui i = 0; i < n; i++) {
            ui r = __builtin_popcount(t[i]&a)%2;
            res |= r<<(n-i-1);
        }
        return res;
    }
    bm pot(ui k) { //exponenciacao em logn
        bm res(n,n);
        if (k == 0) {
            for (ui i = 0; i < n; i++) res.set(n-i-1,i);
            return res;
        }
        if (k == 1) return *this;
        ui l = k/2;
        res = pot(l);
        if (k%2) return res*res*(*this);
        return res*res;
    }
    void print() { //imprime
        for (ui i = 0; i < n; i++) {
            for (ui j = m; j > 0; j--) {
                printf("%d",(*this)(i,j-1));
            }
            printf("\n");
        }
        printf("\n");
    }
};

**** Exemplo simples de uso ****
int main() {
    int n = 5, m = 5;
    bm mat(n,m);
    for (int i = 0; i < n; i++) mat[i] = 1<<i;
    bm mattransp = mat.transp();
    bm mat2 = mat*mat;
    mat.print();
    mattransp.print();
    mat2.print();
    return 0;
}

```

6.15 Range Minimum Query 2D

Autor: André Linhares

Complexidade:

inicialização: $O(nm)$

atualização e consulta: $\log(nm)$

Teste: UVA 11297

Descrição: encontra a posição do menor elemento de uma submatriz. A função best pode ser adaptada para Range Maximum Query ou para definir um critério de desempate.

```
#include <iostream>
#include <vector>
#include <utility>
#include <cstdlib>
#include <algorithm>

#define pii pair<int,int>

using namespace std;

#define for_to(i,j,k) for(i=j; i<=k; ++i)

#define xm (x1+x2)/2
#define ym (y1+y2)/2

#define r0 x1,xm,y1,ym
#define r1 xm+1,x2,y1,ym
#define r2 x1,xm,ym+1,y2
#define r3 xm+1,x2,ym+1,y2

#define G0 go(0,r0); go(1,r1); go(2,r2); go(3,r3);

template <class T>
struct RMQ
{
    public:
    void init(vector<vector<T> > w)
    {
        R=(int)w.size()-1, C=(int)w[0].size()-1;
        v=w;
        son.assign(1,vector<int> (4,0));
        ans.resize(1,pii(0,0));
        init(0,0,R,0,C);
    }

    void update(T t,int i,int j=0)
    {
        x=i, y=j;
        v[x][y]=t;
        update(0,0,R,0,C);
    }

    pii query(int a,int b,int c=0,int d=0)
    {
        xi=a, xf=b, yi=c, yf=d;
        ret=pii(xi,yi);
        query(0,0,R,0,C);
        return ret;
    }
}
```

```
int R,C,x,y,xi,xf,yi,yf;
vector<vector<int> > son;
vector<vector<T> > v;
vector<pii> ans;
pii ret;

pii best(pii a,pii b)
{
    if (v[a.first][a.second] < v[b.first][b.second]) return a;
    else if (v[a.first][a.second] > v[b.first][b.second])
        return b;
    return min(a,b);
}

#define _go(i,a,b,c,d) if (a<=b && c<=d) \
{ son[node][i]=son.size(); son.push_back(vector<int> (4,0));\
ans.push_back(pii(a,c)); init(son[node][i],a,b,c,d); \
ans[node]=best(ans[node],ans[son[node][i]]); }

#define go(a,b) _go(a,b)

void init(int node,int x1,int x2,int y1,int y2)
{ if (x1!=x2 || y1!=y2) G0; }

#undef _go
#define _go(i,a,b,c,d) if (a<=x && x<=b && c<=y && y<=d) \
update(son[node][i],a,b,c,d); if (son[node][i]) \
ans[node]=best(ans[node],ans[son[node][i]]);

void update(int node,int x1,int x2,int y1,int y2)
{ if (x1!=x2 || y1!=y2) G0; }

#undef _go
#define _go(i,a,b,c,d) if (son[node][i] && !(a>xf || b<xi \
|| c>yf || d<yi)) { query(son[node][i],a,b,c,d); }

void query(int node,int x1,int x2,int y1,int y2)
{
    if (x1==x2 && y1==y2 || (xi<=x1 && x2<=xf && yi<=y1
        && y2<=yf))
    {
        ret=best(ret,ans[node]);
        return;
    }
    G0;
}

RMQ<int> T1,T2;
vector<vector<int> > v1,v2;
int i,j,k,n;
int m,t,x,y,a,x1,x2,y1,y2;
char c;
int a1,a2,lixo;
pii P;

int main()
{
    scanf("%d %d",&n,&lixo);
```

```
v1=v2=vector<vector<int> > (n,vector<int>(n));
for_to(i,0,n-1)
    for_to(j,0,n-1)
    {
        scanf("%d",&v1[i][j]);
        v2[i][j]=-v1[i][j];
    }
T1.init(v1);
T2.init(v2);
scanf("%d",&m);
for_to(t,1,m)
{
    scanf(" %c",&c);
    if (c=='c')
    {
        scanf("%d %d %d",&x,&y,&a);
        --x, --y;
        T1.update(a,x,y);
        T2.update(-a,x,y);
    }
    else
    {
        scanf("%d %d %d %d",&x1,&y1,&x2,&y2);
        --x1, --x2, --y1, --y2;
        P=T1.query(x1,x2,y1,y2);
        a1=T1.v[P.first][P.second];
        P=T2.query(x1,x2,y1,y2);
        a2=-T2.v[P.first][P.second];
        printf("%d %d\n",a2,a1);
    }
}
return 0;
}
```

6.16 Range Minimum Query (RMQ)

Autor: NU 2/Marcelo Galvão Póvoa

Complexidade: $O(n \lg n)$ -preprocessamento e $O(1)$ -consulta

Tempo de implementacao: 7 min

Testes: Testes proprios aleatorios

Descricao: Apos o preprocessamento de um vetor v, o algoritmo responde de forma eficiente o indice do elemento minimo em v[a..b]. Em caso de empate, é devolvido o maior indice. Caso queira o menor, descomente o = em pairmin()

```
using namespace std;
#define N 100100
#define LOG 16 // piso de log2(N)

int *va, Log2[N], p[LOG+1][N];

int pairmin(int i1, int i2) {
    return va[i1]</**/va[i2] ? i1:i2;
}

void init(int *a, int n) {
    va = a;
    for (int i=1,k=0; i<=n; i++) {
        Log2[i] = k;
        if (1 << (k+1) == i) k++;
    }
```



```

}

int ln = Log2[n];
for (int i=0; i<n; i++) p[0][i]=i;
for (int i=1; i<=ln; i++)
    for (int j=0; j + (1 << i) - 1 < n; j++) {
        int i1 = p[i-1][j];
        int i2 = p[i-1][j+ (1 << i-1)];
        p[i][j] = pairmin(i1, i2);
    }
}

int query(int b, int e) {
    int ln = Log2[e - b + 1];
    int i1 = p[ln][b];
    int i2 = p[ln][e - (1 << ln) + 1];

    return pairmin(i1,i2);
}

/**** Exemplo simples de uso ****/
#include <cstdio>
int main(){
    int i,n,x;
    int vet[1000];

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

    init(vet, n);
    printf("%d\n",query(0,n-1));
    /*imprime o maior indice de um menor elemento*/

    return 0;
}

```

6.17 Rope (via árvore cartesiana)

Autor: Marcelo Galvão Póvoa

Complexidade: $O(\lg n)$ por operação

Tempo de implementacao: ?

Testes: LiveArchive.5902

Descricao: Estrutura para manipular cadeias, suporta merge e split em qualquer ponto. Implementada com árvore cartesiana com Y's aleatórios, o que a torna balanceada.

```

#include <cstdio>
#include <cstring>
#include <algorithm>
#define MAXN 100100
using namespace std;

```

```

struct _node {
    int c;        // Contagem de nós na subárvore (inclui raiz)
    int v, sum;    // "Valor" da raiz e a soma deles na subárvore
    int id;        // Índice do nó na cadeia original
    int y;

```

```

    _node *l,*r;
} mem[MAXN], nil;

typedef _node* node;

// Atualiza o nó T dado que seus filhos já o fizeram
// pode ser estendido se houver outras variáveis de interesse
void fix(node T) {
    T->sum = T->v+T->l->sum+T->r->sum;
    T->c = 1+T->l->c+T->r->c;
}

// Divide subárvore T em [L,R], deixando L com x elementos
void split(node T, int x, node &L, node &R) {
    if (T==&nil) L = R = &nil;
    else if (x <= T->l->c) {
        split(T->l, x, L, T->l);
        fix(T);
        R = T;
    }
    else {
        split(T->r, x-T->l->c-1, T->r, R);
        fix(T);
        L = T;
    }
}

node merge(node L, node R) {
    if (L == &nil) return R;
    if (R == &nil) return L;
    if (L->y > R->y) {
        L->r = merge(L->r, R);
        fix(L);
        return L;
    }
    R->l = merge(L, R->l);
    fix(R);
    return R;
}

node add(node T, node N) {
    if (T == &nil) return N;
    if (T->y < N->y) {
        split(T, N->id, N->l, N->r);
        fix(N);
        return N;
    }
    if (N->id < T->id) T->l = add(T->l,N);
    else T->r = add(T->r,N);

    fix(T);
    return T;
}

```

```

// Uso como árvore de segmentos da variável sum
int query(node T, int ll, int rr, int a, int b) {
    if (T == &nil || a > b) return 0;
    if (a == ll && b == rr) return T->sum;

    int me = ll+T->l->c;

```

```

    int res = query(T->l, ll, me-1, a, min(b, me-1))+
        query(T->r, me+1, rr, max(a, me+1), b);

    if (a<=me && b>=me) res += T->v;

    return res;
}

// Devolve o nó na x-ésima posição de T
node getid(node T, int x) {
    if (T->l->c == x) return T;
    if (T->l->c > x) return getid(T->l, x);
    return getid(T->r, x-T->l->c-1);
}

int main() {
    int n,x;

    while (scanf(" %d",&n)==1 && n) {
        node t = &nil;

        for (int i=0; i<n; i++) {
            scanf(" %d",&x);

            mem[i].y=rand()%123456789;
            mem[i].v=mem[i].sum=x;
            mem[i].c=1;
            mem[i].l=mem[i].r=&nil;
            mem[i].id=i;

            t=add(t, &mem[i]);
        }

        // troca de ordem as duas metades da sequência
        node p1, p2;
        split(t, n/2, p1, p2);
        t=merge(p2, p1);

        for (int i=0; i<n; i++)
            printf("%d\n",getid(t, i)->v);
    }
    return 0;
}

```

7 Matemática

7.1 Geometria

Matriz de rotação

$$\begin{bmatrix} x_\theta \\ y_\theta \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Fórmula de Brahmagupta Sendo a, b, c, d os lados do quadrilátero, $s = \frac{1}{2}(a + b + c + d)$:

$$A = \sqrt{(s-a)(s-b)(s-c)(s-d) - k}$$

E sendo θ a soma do ângulo de dois lados opostos, ou p e q os

comprimentos das diagonais do quadrilátero, temos:

$$\begin{aligned} k &= abcd \cos^2 \frac{\theta}{2} \\ &= \frac{1}{4}(ac + bd + pq)(ac + bd - pq) \end{aligned}$$

Calota Esférica Sendo R o raio da esfera, r o raio da base, e h a altura da calota:

$$A_{calota} = 2\pi Rh$$

$$V_{calota} = \frac{\pi h}{6} (3r^2 + h^2) = \frac{\pi h^2}{3} (3R - h)$$

Área de Segmento Circular Sendo α o ângulo formado pelo segmento circular, temos:

$$A_{segmento} = \frac{r^2}{2} (\alpha - \text{sen } \alpha)$$

Se tivermos h , a altura do segmento circular, ao invés de α :

$$\alpha = 2\text{acos} \left(\frac{h}{r} \right)$$

Centróide de um Polígono

$$c_x = \frac{1}{6A} \sum_{i=0}^{n-1} (x_i + x_{i+1})(x_i y_{i+1} - x_{i+1} y_i)$$

$$c_y = \frac{1}{6A} \sum_{i=0}^{n-1} (y_i + y_{i+1})(x_i y_{i+1} - x_{i+1} y_i)$$

Área de triângulo Sendo R o raio da circunferência circunscrita, e r da inscrita, temos:

$$A_{\Delta} = \frac{abc}{4R} = \frac{(a + b + c)r}{2}$$

Fórmula de Euler para Poliedros Convexos V vértices, A arestas, F faces: $V - A + F = 2$

Teorema de Pick Sendo A a área de um polígono e i e b a quantidade de pontos de coordenadas inteiras no interior e na borda no polígono,respectivamente, temos:

$$A = i + b/2 - 1$$

Quantidade de pontos de coordenas inteiras num segmento Sendo (x_1, y_1) e (x_2, y_2) pontos de coordenadas inteiras nos extremos de um segmento:

$$q = \text{mdc}(|x_1 - x_2|, |y_1 - y_2|) + 1$$

7.2 Relações Binomiais

Relação de Stifel:

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$$

Absorções:

$$\binom{n}{k} = \frac{n-k+1}{k} \binom{n}{k-1} = \frac{n}{k} \binom{n-1}{k-1} = \frac{n}{n-k} \binom{n-1}{k}$$

Soma de quadrados de binomiais:

$$\sum_{k=0}^n \binom{n}{k}^2 = \binom{2n}{n}$$

7.3 Equações Diofantinas

Dados inteiros $a, b > 0$ e c , a equação $ax + by = c$ tem soluções sse $g = \text{gcd}(a, b)$ é divisor de c .

Sejam x_g e y_g a solução de $a \cdot x_g + b \cdot y_g = g$ obtida por Euclides. Então:

$$\begin{cases} x = x_g(c/g) + k \cdot b/g \\ y = y_g(c/g) - k \cdot a/g \end{cases} \quad k \in Z$$

7.4 Fibonacci

Fórmula em $lg(n)$:
 $f(0) = 1$ e $f(1) = 1$

$$\begin{aligned} f(n) &= f(x)f(n-x) + f(x-1)f(n-x-1) \\ &= f(\lfloor \frac{n}{2} \rfloor)f(n - \lfloor \frac{n}{2} \rfloor) + f(\lfloor \frac{n}{2} \rfloor + 1)f(n - \lfloor \frac{n}{2} \rfloor - 1) \\ &= f(\lfloor \frac{n}{2} \rfloor)f(\lceil \frac{n}{2} \rceil) + f(\lfloor \frac{n}{2} \rfloor + 1)f(\lceil \frac{n}{2} \rceil - 1) \end{aligned}$$

Fórmula com potência de matrizes:

$$\begin{bmatrix} f(n+1) \\ f(n) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n \begin{bmatrix} f(1) \\ f(0) \end{bmatrix}$$

Propriedades:

- $f(n+1)f(n-1) - f(n)^2 = (-1)^n$
- $f(m)$ múltiplo de $f(n)$ sse m múltiplo de n
- $\text{mdc}(f(m), f(n)) = f(\text{mdc}(m, n))$

7.5 Problemas clássicos

Fila do cinema: Sendo n pessoas com \$5 e m com \$10, temos: $K_{0,m} = 0$ e $K_{n,0} = 1$
 $K_{n,m} = K_{n-1,m} + K_{n,m-1}$

$$K_{n,m} = \binom{n+m}{n} - \binom{n+m}{n+1} = \frac{n-m+1}{n+1} \binom{n+m}{n}$$

Números de Catalan: É um caso do problema da *Fila de cinema*, com $n = m$.

$$C_n = \binom{2n}{n} - \binom{2n}{n+1} = \frac{1}{n+1} \binom{2n}{n}$$

Aplicações: 1) Número de expressões com n pares de parênteses, todos abrindo e fechando corretamente. Exemplo: $(()) () ()$; 2) Número de maneiras de parentizar completamente $n + 1$ fatores. Exemplo: $(ab)c$ $a(bc)$; 3) Número de árvores binárias completas com $n + 1$ folhas; 4) Número de maneiras de triangularizar um polígono convexo de $n + 2$ lados;

Número de somas $x_1 + x_2 + \dots + x_n = p$
- Soluções não negativas: $CR_n^p = \binom{n+p-1}{p}$

- Soluções positivas $CP_n^p = \binom{p-1}{n-1}$

Variáveis com restrições: Quando alguns x_i têm restrições do tipo $x_i \geq 3$, adotamos um y_i tal que $x_i = 3 + y_i$.

Assim, seguindo a restrição de que $y_i \geq 0$, teremos $x_i \geq 3$. A soma fica, então:

$$\begin{aligned} x_1 + x_2 + \dots + x_i + \dots + x_n &= p \\ x_1 + x_2 + \dots + y_i + \dots + x_n &= p - 3 \end{aligned}$$

De forma geral, teremos:

$$CR_n^p = \binom{n+p-(b_1+b_2+\dots+b_n)-1}{p}$$

Sendo b_i o decremento (pode ser negativo) na variável x_i .
 $x_1 + x_2 + \dots + x_n \leq p$
Definimos uma variável de *folga*, $f = p - (x_1 + x_2 + \dots + x_n)$, e obtemos:

$$\begin{aligned} f &\geq 0 \\ x_1 + x_2 + \dots + x_n + f &= p \end{aligned}$$

Permutações Caóticas: O número de permutações caóticas para n elementos é dado por: $D_0 = 1; D_n = (-1)^n + nD_{n-1} = (n-1)(D_{n-1} + D_{n-2})$

Triângulos de Lados em $\{1, 2, \dots, n\}$

$$f_{n+1} = f_n + \begin{cases} \frac{(n-2)^2}{2} & , \text{ n par} \\ \lceil \frac{(n-2)(n-4)}{4} \rceil & , \text{ n ímpar} \end{cases}$$

Problema de Josephus: Sendo n pessoas em circulo, eliminando-se de k em k , temos a recorrência:

$$\begin{aligned} f(1, k) &= 0 \\ f(n, k) &= (f(n-1, k) + k)(\text{mod } n) \end{aligned}$$

Formas de Conectar um Grafo: Seja um grafo com k componentes com tamanhos s_1, \dots, s_k . O número de maneiras de adicionar $k - 1$ arestas de modo a conectá-lo é: $s_1 \dots s_k n^{k-2}$

Código de Gray: $\text{gray}(i) = i \text{ xor } \frac{i}{2}$

Código de Gray Invertido (n-bits):

$$\overline{\text{gray}}_n(i) = (\frac{i}{2} \text{ or } (i \text{ and } ((n\%2)2^{n-1}))) \text{ xor } \begin{cases} \frac{i}{2} & , \text{ i par} \\ \frac{i}{2} & , \text{ i ímpar} \end{cases}$$

7.6 Séries Numéricas

$$\sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{i=1}^n i^3 = \frac{n^2(n+1)^2}{4}$$

PA de 2ª ordem:

$$a_n = a_1 + b_1(n-1) + \frac{r}{2}(n-2)(n-1)$$

$$S_n = a_1 n + \frac{b_1 n(n-1)}{2} + \frac{r}{6} n(n-2)(n-1)$$

PA de nª ordem:

$$S_k = a_1 \binom{k}{1} + \sum_{i=1}^n \Delta_i \binom{k}{i+1}$$

Δ_i : Primeiro elemento considerando a i-ésima PA.

Exemplo: $n = 5$; seq = (1,32,243,1024,3125,7776,...)

$$\Delta_1 = 32 - 1 = 31$$

$$\Delta_2 = 211 - 31 = 180$$

$$\Delta_3 = 570 - 180 = 390$$

$$\Delta_4 = 750 - 390 = 360$$

$$\Delta_5 = 480 - 360 = 120$$

Para a PA de 2ª ordem ficaria:

$$\Delta_1 = b_1$$

$$\Delta_2 = b_2 - b_1 = r$$

7.7 Matrizes e Determinantes

Determinante de Vandermonde:

$$V_n = \begin{vmatrix} 1 & 1 & \cdots & 1 \\ a_1 & a_2 & \cdots & a_n \\ a_1^2 & a_2^2 & \cdots & a_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ a_1^{n-1} & a_2^{n-1} & \cdots & a_n^{n-1} \end{vmatrix} = \prod_{i>j} (a_i - a_j)$$

7.8 Probabilidades

Probabilidade Condicional:

$$P(B|A) = \frac{n(A \cap B)}{n(A)} = \frac{P(A \cap B)}{P(A)}$$

Experimentos Repetidos: Seja um experimento que se repete n vezes, e em qualquer um deles temos $P(A) = p$ e, portanto, $P(\bar{A}) = 1 - p$. A probabilidade do evento A ocorrer k das n vezes é:

$$P_k = \binom{n}{k} p^k (1-p)^{n-k}$$

7.9 Teoria dos Números

Teorema de Fermat-Euler: Se p é primo, temos, para todo inteiro a : $a^{p-1} \equiv 1 \pmod{p}$. Se temos a e n coprimos: $a^{\phi(n)} \equiv 1 \pmod{n}$ onde $\phi(n) = n \prod_{p|n} \left(1 - \frac{1}{p}\right)$, p é fator primo de n , é a quantidade de números entre 1 e n que são coprimos com n .

Teorema de Wilson: n é primo sse $(n-1)! \equiv -1 \pmod{n}$

Soma dos Divisores: A soma dos divisores de n elevados à x -ésima potência, sendo p_i os fatores primos e a_i os expoentes correspondentes:

$$\sigma_x(n) = \prod_{i=1}^r \frac{p_i^{(a_i+1)x} - 1}{p_i^x - 1}$$

Divisibilidade

Considere o número como: $a_n a_{n-1} a_{n-2} \dots a_2 a_1 a_0$

Por 3: A soma dos dígitos deve ser divisível por 3

Por 4: O número formado por $a_1 a_0$ deve ser divisível por 4

Por 7: A soma $a_2 a_1 a_0 - a_5 a_4 a_3 + a_8 a_7 a_6 - \dots$ deve ser divisível por 7

Por 8: O número formado por $a_2 a_1 a_0$ deve ser divisível por 8

Por 9: A soma dos dígitos deve ser divisível por 9

Por 11: A soma $a_0 - a_1 + a_2 - a_3 + a_4 - \dots$ deve ser divisível por 11

Por 13: A soma $a_2 a_1 a_0 - a_5 a_4 a_3 + a_8 a_7 a_6 - \dots$ deve ser divisível por 13

Equação Modular Linear: Dada equação $ax \equiv b \pmod{m}$, se $b \equiv 0 \pmod{g}$ onde $g = \gcd(a, m)$, então as soluções são:

$$x = \frac{b}{g} * \text{invmod}\left(\frac{a}{g}, \frac{m}{g}\right) + k \frac{m}{g}, k \in \mathbb{Z}$$