acm International Collegiate Programming Contest

ACM ICPC Reference

Os Feras do Paranauê

Notebook de Algoritmos

ÍNDICE

1. DICAS E MASSETES PARA UMA BOA PROVA

1.1 Seguir os 1010 passos para resolver um problema

```
0) Ler, reler e Dividir em subproblemas
1) Verificar o score e ver a quantidade de submissões erradas no exercício
2) Traçar a estratégia a ser usada para cada subproblema
3) Perguntar a opinião dos outros membros do time sobre as estratégias
4) Escrever o código das sub-rotinas do que solucionam os subproblemas
5) Testar os casos de teste
6) Testar nos limites (avaliar o intervalo de dados)
7) Se o computador estiver livre, ocupa-lo para digitar o código
8) Testar o programa com os casos de teste e nos limites do intervalo
9) Submeter o problema e mandar imprimir imediatamente
```

1.2 Não desobedecer aos 1010 mandamentos

```
0) Não dividirás por zero.
1) Não alocarás dinamicamente.
2) Compararás números de ponto flutuante usando cmp().
3) Verificarás se o grafo pode ser desconexo.
4) Verificarás se as arestas do grafo podem ter peso negativo.
5) Verificarás se pode haver mais de uma aresta ligando dois vértices.
6) Conferirás todos os índices de uma programação dinâmica.
7) Reduzirás o branching factor da DFS.
8) Farás todos os cortes possíveis em uma DFS.
9) Tomarás cuidado com pontos coincidentes e com pontos colineares.
```

2. TEMPLATE

2.1 C++

```cpp
#include <cstdio>
#include <string>
#include <cstring>
#include <vector>
#include <algorithm>
#include <map>
#include <utility>
#include <cmath>
#include <queue>
#include <stack>
#include <set>
#include <deque>
#include <iostream>
#include <math.h>
#include <sstream>
#include <assert.h>
#include <numeric>
#include <fstream>
#include <limits>
#define INF 0x3f3f3f3f

using namespace std;

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

2.2 JAVA

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

public class template
{
    public static void main(String[] args)
    {
        Scanner teclado = new Scanner(System.in);

        teclado.next();
        teclado.nextBoolean();
        teclado.nextInt();
        teclado.nextLong();
        teclado.nextDouble();
        teclado.nextFloat();
        teclado.nextLine();
        teclado.nextBigDecimal();
        teclado.nextBigInteger();

        Calendar g = new GregorianCalendar(2000, 11, 25);

        g.get(GregorianCalendar.DAY_OF_WEEK);
        g.get(GregorianCalendar.DAY_OF_MONTH);
        g.get(GregorianCalendar.MONTH);
        g.get(GregorianCalendar.YEAR);

        BigInteger b = new BigInteger("10");

        b.add(new BigInteger("1"));
        b.divide(new BigInteger("1"));
        b.multiply(new BigInteger("1"));
        b.subtract(new BigInteger("1"));

        System.out.println("OK");
    }
}
```

3. GRAFOS

3.1 BREADTH-FIRST SEARCH (BFS)

```
#include <stdio.h>
#include <string.h>
#include <queue>
#define MAXN 1001
#define INF 0x3F3F3F3F

using namespace std;

/*

    BREADTH-FIRST SEARCH (BFS)

    Aplicacoes:

        Menor caminho em grafos sem peso
        Verificar se um grafo eh bipartido
        Verificar se um grafo eh bicoloravel
        Alcancabilidade
        Conectividade
        Etc.

    Como chamar a funcao:

        1) Zerar a matriz de adjacencias (graph[][])

        2) Armazenar em n o numero de vertices do grafo

        3) Preencher a matriz de adjacencias com as
           informacoes referentes ao grafo.

        4) Chamar a funcao bfs(s), onde s eh o vertice
           de origem da busca.

    Resultado da funcao:

        A funcao armazena no vetor dist[] a distancia
        do vertice s ate o vertice i, se dist[i] == INF
        o vertice s nao pode alcancar o vertice i

    Complexidade do algoritmo:

        O(n^2)

    Problemas resolvidos:

        MESA (SPOJ BR)
        MOEDAS (SPOJ BR)
        DUENDE (SPOJ BR)

    Adicionado por:

        Jorge Gabriel Siqueira

*/
```

```
bool graph[MAXN][MAXN];
int dist[MAXN];
int n;

void bfs (int s)
{
    memset (dist, INF, sizeof(int)*n);
    dist[s] = 0;
    queue <int> q;
    q.push (s);
    while (!q.empty())
    {
        int u = q.front();
        q.pop();
        for (int v = 0; v < n; ++v)
        {
            if (dist[v] == INF && graph[u][v])
            {
                dist[v] = dist[u] + 1;
                q.push(v);
            }
        }
    }
}
```

3.2 DEPTH-FIRST SEARCH (DFS)

```
#include <stdio.h>
#define MAXN 1001

using namespace std;

/*

    DEPTH-FIRST SEARCH (DFS)

    Aplicacoes:

        Deteccao de ciclos
        Determinacao de pontes e articulacoes
        Caminho/ciclo Euleriano
        Decomposicao em componentes fortemente conexas
        Teste de conectividade
        Bipartite matching
        Etc.

    Como chamar a funcao:

        1) Zerar a matriz de adjacencias (graph[][]), color[]
           e a variavel num.

        2) Armazenar em n o numero de vertices do grafo

        3) Preencher a matriz de adjacencias com as
           informacoes referentes ao grafo.

        4) Chamar a funcao dfs(u,p), onde u eh o vertice
           de origem (source) para a busca e p eh o
           vertice de destino da busca. Se a intencao
           for realizar uma busca completa deve-se passar
           o valor -1 no parametro p.

    Resultado da funcao:

        A funcao armazena em dfsNum[] a ordem de visita
        dos vertices durante a busca. E o vetor color[]
        indica se a busca encontrou (2) ou nao (0) o
        vertice.

    Complexidade do algoritmo:

        O(n^2)

    Problemas resolvidos:

        DEPENDEN (SPOJ BR)
        FORRO (SPOJ BR)
        IREVIR (SPOJ BR)
        NATUREZA (SPOJ BR)

    Adicionado por:

        Jorge Gabriel Siqueira
```

```
*/

bool graph[MAXN][MAXN];
int color[MAXN];
int dfsNum[MAXN];
int num;
int n;

void dfs (int u, int p)
{
    color[u] = 1; //gray
    dfsNum[u] = ++num;
    for (int v = 0; v < n; ++v)
    {
        if (graph[u][v] && v != p)
        {
            if (color[v] == 0)   //forward edge
            {
                dfs (v, u);
            }
        }
        else if (color[v] == 1)   //back edge
        {
        }
        else   //cross edge, somente em grafos direcionados
        {
        }
    }
    color[u] = 2; //black
}
```

3.3 ARTICULATION POINTS AND BRIDGES

```
#include <stdio.h>
#include <string.h>
#include <vector>
#define MAXN 400

using namespace std;

/*

    ARTICULATION POINTS AND BRIDGES DETECTION ALGORITHM

    Aplicacoes:

        Deteccao de pontes e pontos de articulacao
        simultaneamente.

    Como chamar a funcao:

        1) Armazenar em n o numero de vertices do grafo

        2) Armazenar o grafo em forma de lista de adjacencias
           no vector graph[]

        3) Chamar a funcao articulations_and_bridges()

    Resultado da funcao:

        A funcao armazena em vector<int> articulacoes os
        vertices de corte do grafo e em
        vector< pair<int,int> > pontes as arestas pontes
        do grafo.

    Complexidade do algoritmo:

        O(n+m) onde m eh o numero de arestas do grafo

    Problemas resolvidos:

        MANUT (SPOJ BR)

    Adicionado por:

        Jorge Gabriel Siqueira

*/

vector <int> graph[MAXN];
vector <int> articulacoes;
vector <pair <int, int> > pontes;
bool visited[MAXN];
int timer, d[MAXN], lowpt[MAXN], pi[MAXN];
int n;

void dfs (int u)
{
    int sons = 0, v;
    bool art = false;
```

```
    visited[u] = true;
    lowpt[u] = d[u] = timer++;
    int k = graph[u].size();
    for (int i = 0; i < k; ++i)
    {
        v = graph[u][i];
        if (!visited[ graph[u][i]])
        {
            ++sons;
            pi[v] = u;
            dfs(v);
            lowpt[u] = min (lowpt[u], lowpt[v]);
            if ((pi[u] != -1) && (lowpt[v] >= d[u]))
                art = true;
            if (lowpt[v] > d[u])
                pontes.push_back(make_pair(u,v));
        }
        else if (v!= pi[u])
            lowpt[u] = min (lowpt[u], d[v]);
    }
    if (art)
        articulacoes.push_back(u);
    else if (pi[u] == -1 && sons > 1)
        articulacoes.push_back(u);
}

void articulations_and_bridges()
{
    memset (visited, 0, sizeof(visited));
    memset (pi, -1, sizeof(pi));
    articulacoes.clear();
    pontes.clear();
    timer = 0;
    for (int i = 0; i < n; ++i)
        if (!visited[i])
            dfs(i);
}
```

3.4 TARJAN'S STRONGLY CONNECTED COMPONENTS ALGORITHMS

```
#include <stdio.h>
#include <string.h>
#include <vector>
#define MAXN 1000

using namespace std;

/*

    TARJAN'S STRONGLY CONNECTED COMPONENTS ALGORITHMS

    Aplicacoes:

        Decomposicao de um grafo direcionado desconexo
        em suas componentes fortemente conexas.
        2-SAT
        Etc.

    Como chamar a funcao:

        1) Armazenar o grafo em forma de lista de adjacencias
           no vector graph[]

        2) Chamar a funcao scc_decomposition(n), onde n eh o
           numero de vertices do grafo

        Observacao: Para o problema 2-SAT, criar um vertice
        para cada variavel e sua respectiva negacao. Para cada
        afirmacao da forma (a OU b), criar uma aresta (~b -> a)
        e outra (~a -> b). Todos os vertices de uma mesma
        componente fortemente conexa sao verdadeiros ou falsos.
        Portanto, se uma variavel e sua respectiva negacao
        pertencerem a uma mesma componente o problema nao possui
        solucao. Resultado da decomposicao em componentes
        fortemente conexas: um DAG (digrafo aciclico)

    Resultado da funcao:

        A funcao armazena em component[] : component[i] = o indice
        da componente ao qual o vertice i pertence
        A funcao tambem armazena em scc o numero de componentes
        fortemente conexas do grafo.

    Complexidade do algoritmo:

        O (n+m)

    Problemas resolvidos:

        DESVRUA (SPOJ BR)
        CARDAPIO (SPOJ BR)

    Adicionado por:

        Jorge Gabriel Siqueira

*/
```

```
vector <int> g[MAXN];
int component[MAXN];
int scc;

vector <int> S;
bool in_stack[MAXN];
int indices[MAXN], lowlinks[MAXN], indice;

void tarjan (int v)
{
    indices[v] = indice;
    lowlinks[v] = indice;
    ++indice;
    S.push_back(v);
    in_stack[v] = true;
    for (int i = 0; i < g[v].size(); ++i)
    {
        int w = g[v][i];
        if (indices[w] == -1)
        {
            tarjan(w);
            lowlinks[v] = min (lowlinks[v], lowlinks[w]);
        }
        else if (in stack[w])
            lowlinks[v]=min(lowlinks[v], indices[w]);
    }
    if (lowlinks[v] == indices[v])
    {
        int w = S[S.size() - 1];
        S.pop_back();
        while (w != v)
        {
            in_stack[w] = false;
            component[w] = scc;
            w = S[S.size() - 1];
            S.pop_back();
        }
        component[v] = scc++;
        in_stack[v] = false;
    }
}

void scc_decomposition (int n)
{
    for (int i = 0; i < n; ++i)
    {
        indices[i] = lowlinks[i] = component[i] = -1;
        in_stack[i] = 0;
    }
    S.clear();
    indice = scc = 0;
    for (int i = 0; i < n; ++i)
        if (component[i] == -1)
            tarjan(i);
}
```

3.5 DIJKSTRA'S ALGORITHM

```
#include <stdio.h>
#include <string.h>
#include <iostream>
#define INF 0x3F3F3F3F
#define MAXN 1000

using namespace std;

/*

    DIJKSTRA'S ALGORITHM (SINGLE-SOURCE SHORTEST PATH)

    Aplicacoes:

       Menor caminho em grafos densos (m > n2/log(n))
       SEM ARESTAS NEGATIVAS!

    Como chamar a funcao:

        1) Preencher a matriz de adjacencias (grafo[][])
           com INF.

        2) Armazenar em n o numero de vertices do grafo

        3) Preencher a matriz de adjacencias com as
           informacoes referentes ao grafo.

        4) Chamar a funcao dijkstra(p), onde p eh o
           vertice de origem (source) de busca

    Resultado da funcao:

        A funcao armazena em dist[]: dist[i] a distancia
        de p (source) ate i

    Complexidade do algoritmo:

        O(n^2)

    Problemas resolvidos:

        PONTES09 (SPOJ BR)
        QUASEMEN (SPOJ BR)
        DESVIO (SPOJ BR)

    Adicionado por:

        Jorge Gabriel Siqueira

*/

int grafo[MAXN][MAXN];
int dist[MAXN];
int pred[MAXN];
bool visitado[MAXN];
int n;
```

```
void dijkstra(int p)
{
    register int i, v, c;

    memset(dist, INF, sizeof(dist));
    memset(visitado, 0, sizeof(visitado));
    memset(pred, -1, sizeof(pred));

    dist[p] = 0;
        v = p;

        while(!visitado[v])
        {
                visitado[v] = true;
                for(i=0; i<n; i++)
                {
                        if(grafo[v][i] != INF)
                        {
                                c = grafo[v][i];

                                if(dist[i] > dist[v]+c)
                                {
                                        dist[i] = dist[v]+c;
                                        pred[i] = v;
                                }
                        }
                }

                v = 0;
                c = INF;

                for(i=1; i<n; i++)
                {
                        if(visitado[i] == false && c > dist[i])
                        {
                                c = dist[i];
                                v = i;
                        }
                }
        }
}
```

3.6 BELLMAN-FORD

```
#include <stdio.h>
#include <string.h>
#include <vector>
#define INF 0x3F3F3F3F
#define MAXN 150

using namespace std;

/*

    BELLMAN-FORD (SINGLE-SOURCE SHORTEST PATH)

    Aplicacoes:

        Menor caminho em grafos com arestas negativas,
        maior caminho em digrafo aciclico (adicionar os
        custos das arestas com sinal invertido)

    Como chamar a funcao:

        1) Armazenar em n o numero de vertices do grafo

        2) Zerar a matriz de adjacencias (graph) e
           limpar o vetor de arestas (edges.clean())

        3) Armazenar o grafo na matriz de adjacencias e
           nao se esquecer de tambem adicionar as arestas
           no vetor edges. Cada pair<int,int> em edges deve
           representar uma aresta que liga first em second

        4) Chamar a funcao bellman_ford(s) onde s eh o
           vertice de origem (source) da busca.

    Resultado da funcao:

        A funcao armazena em dist[]: dist[i] a distancia
        de s (source) ate i. A funcao retorna false se o
        grafo possui ciclos negativos ou true caso contrario.

    Complexidade do algoritmo:

        O(n*m), onde m eh o numero de arestas do grafo

    Problemas resolvidos:

        INCIDENT (SPOJ BR)
        PASSEIO (SPOJ BR)

    Adicionado por:

        Jorge Gabriel Siqueira

*/

vector <pair<int, int> > edges; //lista de arestas do grafo (vertices de 0 a n-1)
int graph[MAXN][MAXN]; //matriz de adjacencia (custos): vertices de 0 a n-1
int dist[MAXN]; //distancia de s (source) ate os demais vertices alcancaveis
```

```
int n; //numero de vertices do grafo

bool bellman_ford (int s)
{
    int m = edges.size();
    memset (dist, INF, sizeof(int)*n);
    dist[s] = 0;
    for (int k = 0; k < n-1; ++k)
    {
        for (int j = 0; j < m; ++j)
        {
            int u = edges[j].first;
            int v = edges[j].second;
            if (dist[u] < INF && dist[v] > dist[u] + graph[u][v])
                dist[v] = dist[u] + graph[u][v];
        }
    }
    //detecta ciclos negativos
    for (int j = 0; j < m; ++j)
    {
        int u = edges[j].first, v = edges[j].second;
        if (dist[u] < INF && dist[v] > dist[u] + graph[u][v])
            return false;
    }
    return true;
}
```

3.7 FLOYD-WARSHALL (ALL-PAIRS SHORTEST PATH)

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

#define INF 0x3F3F3F3F
#define MAXN 150

using namespace std;

/*

    FLOYD-WARSHALL (ALL-PAIRS SHORTEST PATH)

    Aplicacoes:

        Menor caminho entre todos os pares de vertices do grafo
        Fecho transitivo
        Deteccao de ciclos negativos
        Conectividade
        Etc.

    Como chamar a funcao:

        1) Armazenar em n o numero de vertices do grafo

        2) Inicializar graph[][] da seguinte forma:
           - graph[i][i] = 0, para todo i
           - graph[i][j] = INF, se nao existir uma aresta entre i e j
           - graph[i][j] = c, se existir uma aresta entre i e j de custo c

        4) Chamar a funcao floyd_warshall ()

    Resultado da funcao:

        ANTECAO: A funcao armazena o resultado na propria
        matriz de adjacencias. graph[i][j] represetara no
        final da execucao do algoritmo a distancia minima
        do vertice i ate o vertice j.

    Complexidade do algoritmo:

        O(n^3)

    Problemas resolvidos:

        DENGUE (SPOJ BR)
        IREVIR (SPOJ BR)

    Adicionado por:

        Jorge Gabriel Siqueira

*/

int graph[MAXN][MAXN];
int n;
```

3.7.1 Algoritimo Clássico

```
/*

    Algoritmo Classico

*/
void floyd_warshall ()
{
    for (int k = 0; k < n; ++k)
        for (int i = 0; i < n; ++i)
            for (int j = 0; j < n; ++j)
                graph[i][j] = min (graph[i][j], graph[i][k] +
graph[k][j]);
}
```

3.7.2 Adaptação para detecção de ciclos negativos

```
/*

    Deteccao de Ciclos Negativos

*/
bool floyd_warshall ()
{
    for (int k = 0; k < n; ++k)
    {
        for (int i = 0; i < n; ++i)
        {
            for (int j = 0; j < n; ++j)
            {
                graph[i][j] = min (graph[i][j], graph[i][k] +
graph[k][j]);
            }
            if (graph[i][i] < 0)
                return true;
        }
    }
    return false;
}
```

3.7.3 Adaptação para encontrar o fecho transitivo do grafo

```
/*

    Fecho Transitivo

    Apos a execucao do algoritmo a matriz de adjacencia serq dada por:

         graph[i][j] = 1, se existir um caminho direto ou indireto entre i
e j
         graph[i][j] = 0, se nao existir um caminho entre i e j no grafo

    obs: inicializar graph[i][j] com 1 se existir um
    caminho direto entre os vertices i e j

*/
void floyd_warshall ()
{
    for (int k = 0; k < n; ++k)
    {
        for (int i = 0; i < n; ++i)
        {
            if (graph[i][k])
            {
                for (int j = 0; j < n; ++j)
                {
                    graph[i][j] |= (graph[i][k] & graph[k][j]);
                }
            }
        }
    }
}
```

3.7.4 Adaptação para resolver o problema do Mini-Max

```
/*

    Mini-Max

    O problema consiste em determinar de
    todos os caminhos possiveis entre dois
    vertices o caminho que possui a menor
    maior aresta.

*/
void floyd_warshall ()
{
    for (int k = 0; k < n; ++k)
        for (int i = 0; i < n; ++i)
            for (int j = 0; j < n; ++j)
                graph[i][j] = min(graph[i][j], max(graph[i][k],
graph[k][j]));
}
```

3.7.5 Adaptação para resolver o problema do Max-Mini

```
/*

    Max-Mini

    O problema consiste em determinar de
    todos os caminhos possiveis entre dois
    vertices o caminho que possui a maior
    menor aresta.

*/
void floyd_warshall ()
{
    for (int k = 0; k < n; ++k)
        for (int i = 0; i < n; ++i)
            for (int j = 0; j < n; ++j)
                graph[i][j] = max(graph[i][j], min(graph[i][k],
graph[k][j]));
}
```

3.7.6 Adaptação para resolução do problema Safest-Path

```
/*

    Safest Path

    inicializar graph[i][j] com a probabilidade de
    sobreviver movendo-se do vertice i ao vertice j.

    inicializar graph[i][i] com 0.

    Ao final do processo graph[i][j] contera a probabilidade
    de sobreviver movendo-se do vertice i ao vertice j
    utilizando-se do caminho mais seguro possivel

*/
void floyd_warshall ()
{
    for (int k = 0; k < n; ++k)
        for (int i = 0; i < n; ++i)
            for (int j = 0; j < n; ++j)
                graph[i][j] = max(graph[i][j], graph[i][k] *
graph[k][j]));
}
```

3.8 KRUSKAL'S MINIMUM SPANNING FOREST ALGORITHM

```
#include <stdio.h>
#include <string.h>
#include <algorithm>
#define MAXN 1000
#define MAXM 2000000

using namespace std;

/*

    KRUSKAL MINIMUM SPANNING FOREST

    Aplicacoes:

        Floresta geradora minima em grafos esparsos
        nao direcionados.

    Como chamar a funcao:

        1) Armazenar em n o numero de vertices do grafo

        2) Armazenar em m o numero de arestas do grafo

        3) Armazenar em edges[] as arestas do grafo

            edge.s -> vertice de origem da aresta
            edge.t -> vertice de destino da aresta
            edge.w -> peso da aresta

        4) Chamar a funcao kruskal()

    Resultado da funcao:

        A funcao armazena em result o custo da floresta
        geradora minima e armazena em pa[] o vetor de
        predecessores dos vertices.

    Complexidade do algoritmo:

        O(m(log(m)))

    Problemas resolvidos:

        CIPO (SPOJ BR)

    Adicionado por:

        Jorge Gabriel Siqueira

*/


struct edge
{
    int s, t, w;
    bool operator < (const edge& e) const
    {
```

```
        return w < e.w;
    }
};

edge edges[MAXM]; //lista de arestas do grafo
int pa[MAXN], comp_sz[MAXN];
int n, m; //numero de vertices e arestas do grafo

long long int kruskal()
{
    int u, v;
    long long int result = 0;
    edge e;
    for (int i = 0; i < n; ++i)
    {
        comp_sz[i] = 1;
        pa[i] = i;
    }
    sort (edges, edges+m); //ordenar em tempo linear, se possivel!!!
    for (int i = 0, k = 0; i < m && k < n - 1; ++i)
    {
        e = edges[i];
        //union-find
        for (u = e.s; u != pa[u]; u = pa[u]);
        for (v = e.t; v != pa[v]; v = pa[v]);
        if (u == v) //se a aresta gera um ciclo
            continue;
        if (comp_sz[u] < comp_sz[v])
        {
            pa[u] = v;
            comp_sz[v] += comp_sz[u];
        }
        else
        {
            pa[v] = u;
            comp_sz[u] += comp_sz[v];
        }
        result += e.w;
        ++k;
    }
    return result;
}
```

3.9 PRIM'S MINIMUM SPANNING TREE ALGORITHM

```
#include <stdio.h>
#include <string.h>
#define INF 0x3F3F3F3F
#define MAXN 500

using namespace std;

/*

    PRIM'S MINIMUM SPANNING TREE ALGORITHM

    Aplicacoes:

        Arvore geradora minima em grafos densos
        nao-direcionados

    Como chamar a funcao:

        1) Preencher a matriz de adjacencias com INF

            memset(graph, INF, sizeof(graph));

        2) Armazenar em n o numero de vertices do grafo

        3) Preencher a matriz de adjacencias com as
           informacoes referentes ao grafo.

        4) Chamar a funcao prim()

    Resultado da funcao:

        A funcao retorna o custo da arvore geradora
        minima. A funcao armazena em pa[] o vetor de
        predecessores

    Complexidade do algoritmo:

        O(n^2)

    Problemas resolvidos:

        RMAPA11 (SPOJ BR)
        REDOTICA (SPOJ BR)

    Adicionado por:

        Jorge Gabriel Siqueira

*/


int graph[MAXN][MAXN];
int tree[MAXN], dist[MAXN];
int pa[MAXN];
int n;
```

```
int prim ()
{
    int v0, cost = 0;
    for (int i = 1; i < n; ++i)
    {
        pa[i] = -1;
        tree[i] = 0;
        dist[i] = graph[0][i];
    }
    pa[0] = 0;
    while (1)
    {
        int mincost = INF;
        for (int i = 0; i < n; ++i)
            if (pa[i] == -1 && mincost > dist[i])
                mincost = dist[v0 = i];
        if (mincost == INF)
            break;
        pa[v0] = tree[v0];
        cost += mincost;
        for (int i = 0; i < n; ++i)
            if (pa[i] == -1 && dist[i] > graph[v0][i])
            {
                dist[i] = graph[v0][i];
                tree[i] = v0;
            }
    }
    return cost;
}
```

3.10 EULER CIRCUIT

```
#include <stdio.h>
#include <string.h>
#include <vector>
#include <iostream>
#define INF 0x3F3F3F3F
#define MAXN 1000

using namespace std;

/*

    EULERIAN CIRCUIT

    Aplicacoes:

        Determinar um caminho euleriano em um grafo nao-direcionado.

    Como chamar a funcao:

        1) Preencher a matriz de adjacencias com INF

            memset(graph, INF, sizeof(graph));

        2) Armazenar em n o numero de vertices do grafo

        3) Preencher a matriz de adjacencias com as
           informacoes referentes ao grafo.

        4) Chamar a funcao print_eulerian_circuit()

        Obs.:

        Um caminho euleriano eh um caminho que visita cada aresta do
        grafo uma unica vez.

        Um grafo possui um caminho euleriano se e somente ele eh conexo
        e possui 2 vertices de grau impar. Se o grafo for conexo e nao
        possuir nenhum vertice de grau impar, entao tem-se um circuito
        euleriano (um caminho que inicia e termina no mesmo vertice).

    Resultado da funcao:

        A funcao imprime o circuito euleriano do grafo

        Obs.: Verifique antes se o grafo satisfaz as restricoes
        propostas em 4

    Complexidade do algoritmo:

        O(n)

    Problemas resolvidos:

        OBIDOMIN (SPOJ BR)
        UVA 10054

    Adicionado por:
```

```
        Jorge Gabriel Siqueira

*/


int graph[MAXN][MAXN];
int g[MAXN][MAXN];
int n;

void dfs (int v, vector<int>& path)
{
    for (int i = 0; i < n; ++i)
        if(g[v][i] != INF)
        {
            g[v][i] = INF;
            //g[i][v] = INF -> caso o grafo seja nao-direcionado!
            dfs(i, path);
        }
    path.push_back(v);
}

void print_eulerian_circuit ()
{
    memcpy (g, graph, sizeof(graph));
    vector <int> path;
    dfs(0, path);
    for (int i = 0; i < path.size() - 1; ++i)
        cout << path[i] << ' ' << path[i + 1] << endl;
}
```

3.11 DINIC'S MAXIMUM FLOW ALGORITHM

```
#include <stdio.h>
#include <string.h>
#include <vector>
#define INF 0x3F3F3F3F
#define MAXN 200

using namespace std;

/*

    DINIC MAX FLOW ALGORITHM

    Aplicacoes:

        Fluxo maximo entre dois vertices de um grafo,
        valor do corte minimo entre dois vertices de
        um grafo (max flow = weight of min cut).

    Como chamar a funcao:

        1) Zerar a matriz de adjacencias (cap[][])

        2) Armazenar nas variaveis:

            n: Numero de vertices do grafo
            source: Origem do fluxo
            sink: Destino do fluxo

        3) Preencher a matriz de adjacencias com as
           informacoes referentes ao grafo.

        4) Chamar a funcao maxflow()

    Resultado da funcao:

        A funcao maxflow() retorna um inteiro referente ao
        fluxo maximo do grafo.

    Complexidade do algoritmo:

        O(n^2 * m)

    Problemas resolvidos:

        CAVALOS (SPOJ BR)
        ENGENHAR (SPOJ BR)
        POTHOLE (SPOJ)

    Adicionado por:

        Jorge Gabriel Siqueira

*/

int cap[MAXN][MAXN]; //matriz de adjacencia (capacidades): vertices de 0 a
n-1;
vector <int> adj[MAXN];
```

```
int q[MAXN], prev[MAXN];
int n, source, sink; //numero de vertices, origem e destino do fluxo

int dinic ()
{
    int flow = 0;
    while (1)
    {
        memset (prev, -1, sizeof(int)*n);
        int qf = 0, qb = 0;
        prev[q[qb++] = source] = -2;
        while (qb > qf && prev[sink] == -1)
        {
            for (int u = q[qf++], m = adj[u].size(), i = 0, v; i < m; ++i)
                if (prev[v = adj[u][i]] == -1 && cap[u][v])
                    prev[q[qb++] = v] = u;
        }
        if (prev[sink] == -1)
            break;
        for (int z = 0; z < n; ++z)
            if (cap[z][sink] && prev[z] != -1)
            {
                int bot = cap[z][sink];
                for (int v = z, u = prev[v]; u >= 0; v = u, u = prev[v])
                    bot = min (bot, cap[u][v]);
                if (!bot)
                    continue;
                cap[z][sink] -= bot;
                cap[sink][z] += bot;
                for (int v = z, u = prev[v]; u >= 0; v = u, u = prev[v])
                {
                    cap[u][v] -= bot;
                    cap[v][u] += bot;
                }
                flow += bot;
            }
    }
    return flow;
}

int maxflow ()
{
    for (int i = 0; i < n; ++i)
        adj[i].clear();
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            if(cap[i][j] || cap[j][i])
                adj[i].push_back (j);
    return dinic();
}
```

3.12 FORD FULKERSON'S MAX FLOW ALGORITHM

```
#include <stdio.h>
#include <string.h>
#define INF 0x3f3f3f3f
#define MAX 1001

using namespace std;

/*

    FORD FULKERSON MAX FLOW ALGORITHM

    Aplicacoes:

        Fluxo maximo de um vertice s(origem) para um
        vertice d(destino)

    Como chamar a funcao:

        1) Zerar a matriz de adjacencias (grafo[][])

        2) Armazenar em n o numero de vertices do grafo

        3) Preencher a matriz de adjacencias com as
           informacoes referentes ao grafo.

        4) Chamar a funcao fordF(s,d), onde s eh o vertice
           de origem e d eh o vertice de destino do fluxo

    Resultado da funcao:

        A funcao retorna o fluxo maximo de s ate d.
        A funcao armazena em pred[]: pred[i] o predecessor
        do vertice i (-1 caso nao possua).

    Complexidade do algoritmo:

         O(m*f), onde m eh o numero de arestas e f eh o valor
                 do fluxo maximo.

    Problemas resolvidos:

        CAVALOS (SPOJ BR)

    Adicionado por:

        Jorge Gabriel Siqueira

*/

int grafo[MAX][MAX], pred[MAX], n;
bool visitado[MAX];

bool busca(int s, int d)
{
    register int i, x, pilha[MAX], topo;

    memset(visitado, 0, sizeof(visitado));
```

```
    memset(pred, -1, sizeof(pred));

    topo = -1;

    pilha[++topo] = s;
    pred[s] = s;

    while(topo >= 0)
    {
        x = pilha[topo];
        topo--;
        visitado[x] = 1;
        if(x == d) break;

        for(i=0; i<n; i++)
        {
            if(grafo[x][i] && !visitado[i])
            {
                pilha[++topo] = i;
                pred[i] = x;
            }
        }
    }

    if(pred[d] == -1) return false;
    else return true;
}

int calculaMinimo(int s, int d)
{
    int minimo = INF;
    while(d != s)
    {
        if(minimo > grafo[pred[d]][d]) minimo = grafo[pred[d]][d];
        d = pred[d];
    }
    return minimo;
}

int fordF(int s, int d)
{
    int minimo, total=0, x;
    while(busca(s,d))
    {
        minimo = calculaMinimo(s,d);
        total += minimo;

        x = d;

        while(x != s)
        {
            grafo[pred[x]][x] -= minimo;
            grafo[x][pred[x]] += minimo;
            x = pred[x];
        }
    }
    return total;
}
```

3.13 EDMOND'S KARP'S MINIMUM-COST MAXIMUM-FLOW ALGORITHM

```
#include <stdio.h>
#include <string.h>
#include <iostream>
#define INF 0x3F3F3F3F
#define NMAX 100

using namespace std;

/*

    EDMONDS KARP MIN CUT - MAX FLOW

    Aplicacoes:

        Fluxo maximo de custo minimo em grafos.

    Como chamar a funcao:

        1) Zerar a matriz de fluxo (cap[][])

        2) Preencher a matriz de custo cost[][] com INF

        3) Armazenar nas variaveis:

            n: Numero de vertices do grafo
            source: Origem do fluxo
            sink: Destino do fluxo

        4) Preencher a matriz de fluxo e de custo com as
           informacoes referentes ao grafo.

        5) Usar o dijkstra mais recomendado para o grafo
           em questao.

        6) Chamar a funcao mincost maxflow()

    Resultado da funcao:

        A funcao mincost_maxflow() retorna um pair<long long, long long>:
            first: Custo minimo
            second: Fluxo maximo

    Complexidade do algoritmo:

        O(?)

    Problemas resolvidos:

        FUTURE (SPOJ BR)
        UVA 10594

    Adicionado por:

        Jorge Gabriel Siqueira

*/
```

```
int cap[NMAX][NMAX], cost[NMAX][NMAX]; //capacidades e custos: vertices de
0 a n-1
int adj[NMAX][NMAX], fnet[NMAX][NMAX], deg[NMAX], par[NMAX], d[NMAX],
pi[NMAX];
int n, source, sink; //numero de vertices, origem e destino do fluxo

//Dijkstra para grafos densos
bool dijkstra (int s, int t)
{
    for (int i = 0; i < n; ++i)
        d[i] = INF, par[i] = -1;
    d[s] = 0;
    par[s] = -n - 1;
    while (1)
    {
        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;
        par[u] = -par[u] - 1;
        for (int i = 0; i < deg[u]; ++i)
        {
            int v = adj[u][i];
            if (par[v] >= 0)
                continue;
            if (fnet[v][u] && d[v] > d[u] + pi[u] - pi[v] - cost[v][u])
            {
                d[v] = d[u] + pi[u] - pi[v] - cost[v][u];
                par[v] = -u - 1;
            }
            if (fnet[u][v] < cap[u][v] && d[v] > d[u] + pi[u] - pi[v] +
cost[u][v])
            {
                d[v] = d[u] + pi[u] - pi[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;
}



//Dijkstra para grafos esparsos
#define BUBL { \
t = q[i]; q[i] = q[j]; q[j] = t; \
t = inq[q[i]]; inq[q[i]] = inq[q[j]]; inq[q[j]] = t; }
int q[NMAX], inq[NMAX], qs;
bool dijkstra (int s, int t)
{
    memset (d, INF, sizeof(d));
    memset (par, -1, sizeof(par));
    memset (inq, -1, sizeof(inq));
```

```
    d[s] = qs = 0;
    inq[q[qs++] = s] = 0;
    par[s] = n;
    while (qs)
    {
        int u = q[0];
        inq[u] = -1;
        q[0] = q[--qs];
        if (qs) inq[q[0]] = 0;
        for (int i = 0, j = 2*i + 1, t; j < qs; i = j, j = 2*i + 1)
        {
            if (j + 1 < qs && d[q[j + 1]] < d[q[j]])
                ++j;
            if (d[q[j]] >= d[q[i]])
                break;
            BUBL;
        }
        for (int k = 0, v = adj[u][k]; k < deg[u]; v = adj[u][++k])
        {
            if (fnet[v][u] && d[v] > d[u] + pi[u] - pi[v] - cost[v][u])
                d[v] = d[u] + pi[u] - pi[v] - cost[v][par[v] = u];
            if (fnet[u][v] < cap[u][v] && d[v] > d[u] + pi[u] - pi[v] +
cost[u][v])
                d[v] = d[u] + pi[u] - pi[v] + cost[par[v] = u][v];
            if (par[v] == u)
            {
                if(inq[v] < 0)
                {
                    inq[q[qs] = v] = qs;
                    qs++;
                }
                for (int i = inq[v], j = (i - 1)/2, t;
                        d[q[i]] < d[q[j]]; i = j, j = (i - 1)/2)
                    BUBL;
            }
        }
    }
    for (int i = 0; i < n; ++i)
        if (pi[i] < INF)
            pi[i] += d[i];
    return par[t] >= 0;
}

pair <long long int, long long int> mincost_maxflow()
{
    memset (deg, 0, sizeof(deg));
    memset (fnet, 0, sizeof(fnet));
    memset (pi, 0, sizeof(pi));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            if (cap[i][j] || cap[j][i])
                adj[i][deg[i]++] = j;
    long long int flow = 0, fcost = 0;
    while (dijkstra (source, sink))
    {
        int bot = INF;
        for (int v = sink, u = par[v]; v != source; u = par[v = u])
            bot = min (bot, fnet[v][u] ? fnet[v][u] : (cap[u][v]-
```

```
fnet[u][v]));
        for (int v = sink, u = par[v]; v != source; u = par[v = u])
            if(fnet[v][u])
            {
                fnet[v][u] -= bot;
                fcost -= (long long int) (bot * cost[v][u]);
            }
            else
            {
                fnet[u][v] += bot;
                fcost += (long long int) (bot * cost[u][v]);
            }
        flow += (long long int) bot;
    }
    return make_pair (fcost,flow);
}
```

3.14 KÖNIG'S MAXIMUM BIPARTITE MATCHING ALGORITHM

```
#include <stdio.h>
#include <string.h>
#define MAXN 100
#define MAXM 100

using namespace std;

/*

    KONIG MAXIMUM BIPARTITE MATCHING

    Aplicacoes:

        Maximum bipartite matching.

    Como chamar a funcao:

        1) Zerar a matriz de adjacencias (graph[][])

        2) Armazenar em m o numero de vertices do lado
           esquerdo do grafo e em n o numero de vertices
           do lado direito do grafo.

        3) Adicionar as arestas que ligam o lado esquerdo
           ao lado direito chamando fazendo:

                graph[u][v] = true;

           onde u eh o vertice do lado esquerdo do grafo
           e v eh o vertice do lado direito do grafo.

        4) Chamar a funcao bpm()

    Resultado da funcao:

        A funcao bpm() retorna o numero maximo de
        casamentos possiveis.

        A funcao armazena em matchL[]: matchL[m] o
        indice do vertice do conjunto da direita ao
        qual o vertice m da esquerda se liga (-1 se nao existir)

        A funcao armazena em matchR[]: matchR[n] o
        indice do vertice do conjunto da esquerda ao
        qual o vertice n da direita se liga (-1 se nao existir)

    Complexidade do algoritmo:

        O(n^2 * m)

    Problemas resolvidos:

        ENGENHAR (SPOJ BR)
        URI 1208
        URI 1056

    Adicionado por:
```

```
        Jorge Gabriel Siqueira

*/

bool graph[MAXM][MAXN];
bool seen[MAXN];
int matchL[MAXM], matchR[MAXN];
int n, m;

bool dfs (int u)
{
    for (int v = 0; v < n; ++v)
        if (graph[u][v])
        {
            if (seen[v])
                continue;
            seen[v] = true;
            if (matchR[v] < 0 || dfs(matchR[v]))
            {
                matchL[u] = v;
                matchR[v] = u;
                return true;
            }
        }
    return false;
}

int bpm ()
{
    memset (matchL, -1, sizeof(matchL));
    memset (matchR, -1, sizeof(matchR));
    int cnt = 0;
    for (int i = 0; i < m; ++i)
    {
        memset (seen, 0, sizeof(seen));
        if (dfs (i))
            ++cnt;
    }
    return cnt;
}
```

3.15 TREE ISOMORPHISM

```
#include <iostream>
#include <cmath>
#include <string>
#include <cstring>
#include <iomanip>
#include <vector>
#include <algorithm>
#define MAX 10006

using namespace std;

/*

    TREE ISOMORPHISM

    Aplicacoes:

        Isomorfismo em arvores. Determinar se duas
        arvores com indices de nos distintos tem a
        mesma forma.

    Como chamar a funcao:

        1) Armazenar em n o numero de vertices do grafo

        2) Limpar as listas de adjacencias A e B da
           posicao 0 ate a posicao n.

           for(int i=0; i<=n; i++)
           {
                A[i].clear();
                B[i].clear();
           }


        3) Armazenar a arvore 1 em A e a arvore 2 em B
           como lista de adjacencias.

           Importante: o range de vertices eh de 1 a n (inclusive)

        4) Chamar a funcao treeIsomorphism()

    Resultado da funcao:

        A funcao treeIsomorphism() retorna:

            true: Caso a arvore a seja isomorfica com b
            false: caso contrario

    Complexidade do algoritmo:

        O(n*log(n))

    Problemas resolvidos:

        UVA 12489
        URI 1229
```

```
    Adicionado por:

        Jorge Gabriel Siqueira

*/


vector<int> A[MAX], B[MAX];
vector<int> NA[MAX], NB[MAX];
int n;

bool comp(const vector<int>& a, const vector<int>& b)
{
    if (a.size() != b.size()) return a.size() < b.size();
    for(int i=0; i<a.size(); i++)
    {
        if (a[i] != b[i]) return a[i] < b[i];
    }
    return false;
}

bool eq(const vector<int>& a, const vector<int>& b)
{
    if (a.size() != b.size()) return false;
    for(int i=0; i<a.size(); i++)
    {
        if (a[i] != b[i]) return false;
    }
    return true;
}

bool treeIsomorphism()
{
    memset(NA, 0, sizeof(NA));
    memset(NB, 0, sizeof(NB));

    for(int i=1; i<=n; i++)
    {
        for(int j=0; j<A[i].size(); j++)
            NA[i].push_back(A[A[i][j]].size());
        sort(NA[i].begin(), NA[i].end());

        for(int j=0; j<B[i].size(); j++)
            NB[i].push_back(B[B[i][j]].size());
        sort(NB[i].begin(), NB[i].end());
    }

    sort(NA+1, NA+n+1, comp);
    sort(NB+1, NB+n+1, comp);

    bool equals = true;
    for(int i=1; i<=n; i++) equals &= eq(NA[i], NB[i]);
    return equals;
}
```

3.16 STOER-WAGNER'S MINIMUM GLOBAL CUT ALGORITHM

```
#include <stdio.h>
#include <string.h>
#include <algorithm>

#define INF 0x3f3f3f3f

using namespace std;

/*

    STOER WAGNER MINIMUM CUT

    Aplicacoes:

        Dado um grafo nao orientado e ponderado,
        retorna o peso do corte minimo no grafo.
        Um corte eh um conjunto de arestas que,
        quando removido, desconecta o grafo.
        Um corte minimo eh um corte de peso total minimo.

    Como chamar a funcao:

        1) Zerar a matriz de adjacencias (graph[][])

        2) Armazenar em n o numero de vertices do grafo

        3) Preencher a matriz de adjacencias com as
           informacoes referentes ao grafo.

        4) Chamar a funcao stoer_wagner()

    Resultado da funcao:

        A funcao stoer_wagner() retorna o
        valor do corte minimo no grafo

    Complexidade do algoritmo:

        O(n^3)

    Problemas resolvidos:

        UVA 10989

    Adicionado por:

        Jorge Gabriel Siqueira

*/

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

// Maximum edge weight (MAXW * NN * NN must fit into an int)
#define MAXW 1000000

// Adjacency matrix and some internal arrays
```

```
int g[NN][NN], v[NN], w[NN], na[NN];
bool a[NN];
int n;

int stoer_wagner()
{
    // 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] = g[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++ )
                    g[v[j]][prev] = g[prev][v[j]] += g[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] += g[v[zj]][v[j]];
        }
    }
    return best;
}
```

3.17 GOMORY-HU TREE (ALL-PAIRS MAXIMUM FLOW)

```
#include <stdio.h>
#include <string.h>
#include <queue>
#define MAXN 1000
#define MAXM 100000

using namespace std;

/*

    GOMORY-HU TREE

    Aplicacoes:

        Max-Flow/Min-Cut entre todos os vertices do grafo.

    Como chamar a funcao:

        1) Armazenar em n o numero de vertices do grafo

        2) Chamar a funcao d_init()

        3) Adicionar as arestas no grafo chamando:

            d_edge(s, t, capacity), onde:

                s: Vertice de origem
                t: Vertice de destino
                capacity: Capacidade da aresta

        4) Chamar a funcao gomory_hu()

    Resultado da funcao:

        A funcao gomory hu() armazena em ans[][] a matriz
        de adjacencias que contem o max-flow/min-cut entre
        todos os vertices do grafo.

    Complexidade do algoritmo:

        O(?)

    Problemas resolvidos:

        UVA 11549
        UVA 11603

    Adicionado por:

        Jorge Gabriel Siqueira

*/


int last_edge[MAXN], cur_edge[MAXN], dist[MAXN];
int prev_edge[MAXM], cap[MAXM], flow[MAXM], adj[MAXM];
int up[MAXN], val[MAXN];
```

```
bool cut[MAXN];
int nedges;
int ans[MAXN][MAXN];
int n;

void d_init ()
{
    nedges = 0;
    memset(last_edge, -1, sizeof last_edge);
}

void d_edge (int v, int w, int capacity, bool r = false)
{
    prev_edge[nedges] = last_edge[v];
    cap[nedges] = capacity;
    adj[nedges] = w;
    flow[nedges] = 0;
    last_edge[v] = nedges++;
    if (!r)
        d_edge(w, v, 0, true);
}

bool d_auxflow (int source, int sink)
{
    queue<int> q;
    q.push(source);
    memcpy(cur_edge, last_edge, sizeof last_edge);
    memset(dist, -1, sizeof dist);
    dist[source] = 0;
    while (!q.empty())
    {
        int v = q.front();
        q.pop();
        for (int i = last_edge[v]; i != -1; i = prev_edge[i])
        {
            if (cap[i] - flow[i] == 0)
                continue;
            if (dist[adj[i]] == -1)
            {
                dist[adj[i]] = dist[v] + 1;
                q.push(adj[i]);
                if (adj[i] == sink)
                    return true;
            }
        }
    }
    return false;
}

int d_augmenting (int v, int sink, int c)
{
    if (v == sink)
        return c;
    for (int& i = cur_edge[v]; i != -1; i = prev_edge[i])
    {
        if (cap[i] - flow[i] == 0 || dist[adj[i]] != dist[v] + 1)
            continue;
        int val;
```

```
        if (val = d_augmenting(adj[i], sink, min(c, cap[i] - flow[i])))
        {
            flow[i] += val;
            flow[i^1] -= val;
            return val;
        }
    }
    return 0;
}

int dinic (int source, int sink)
{
    int ret = 0;
    while (d_auxflow(source, sink))
    {
        int flow;
        while (flow = d_augmenting(source, sink, 0x3f3f3f3f))
            ret += flow;
    }
    return ret;
}
int mincut (int s, int t)
{
    memset(flow, 0, sizeof flow);
    memset(cut, 0, sizeof cut);
    int ret = dinic(s, t);
    queue <int> q;
    q.push(s);
    cut[s] = true;
    while (!q.empty())
    {
        int v = q.front();
        q.pop();
        for (int i = last_edge[v]; i != -1; i = prev_edge[i])
        {
            int w = adj[i];
            if (cap[i] - flow[i] && !cut[w])
                cut[w] = true, q.push(w);
        }
    }
    return ret;
}
void gomory_hu ()
{
    memset(up, 0, sizeof up);
    memset(ans, 0x3f3f3f3f, sizeof ans);
    for (int i = 1; i < n; ++i)
    {
        val[i] = mincut(i, up[i]);
        for (int j = i+1; j < n; ++j)
            if (cut[j] && up[j] == up[i])
                up[j] = i;
        ans[i][up[i]] = ans[up[i]][i]=val[i];
        for (int j = 0; j < i; ++j)
            ans[i][j] = ans[j][i] = min(val[i], ans[up[i]][j]);
    }
}
```

3.18 HUNGARIAN ALGORITHM (MAXIMUM WEIGHTED BIPARTITE MATCHING)

```
#include <stdio.h>
#include <string.h>
#include <vector>
#include <math.h>
#define INF 0x3f3f3f3f
#define MAXN 150

using namespace std;

/*

    HUNGARIAN ALGORITHM

    Aplicacoes:

        Maximum bipartite matching de peso maximo/minimo.

        o algoritmo resolve o problema do maximum
        bipartite matching de peso maximo. Para
        matching de peso minimo, inicializar a matriz
        de custos com custos negativos.

    Como chamar a funcao:

            1) Armazenar em n o numero de vertices em cada lado.
               Caso o numero de vertices seja diferente entre os
               dois lados, inicializar n com o meior valor entre
               eles.

            2) Inicializar cost[][] com valores que nao interfiram
               no algoritmo (geralmente 0 para maximo, INF para
               minimo)

            3) Preencher cost[][] com os valores referentes ao grafo:
               Para representar uma aresta que liga o vertice
               x ao vertice y com custo c faca:

                   cost[x][y] = c;

           4) Chame a funcao hungarian()

    Resultado da funcao:

        A funcao hungarian() retorna o valor do peso
        do maximum bipartite matching do grafo.

        A funcao deixa armazenado em xy[]: xy[x] o
        vertice casado com x, e em yx[]: yx[y] o vertice
        casado com y.

        O tamanho do maximum bipartite matching fica
        armazenado em max_match.

    Complexidade do algoritmo:

        O(n^3)
```

```
    Problemas resolvidos:

        SCITIES (SPOJ)

    Adicionado por:

        Jorge Gabriel Siqueira

*/


int cost[MAXN][MAXN];
int xy[MAXN];
int yx[MAXN];
bool S[MAXN], T[MAXN];
int lx[MAXN], ly[MAXN], slack[MAXN], slackx[MAXN], prev[MAXN];
int max_match;
int n;

void init_labels ()
{
    memset(lx, 0, sizeof(lx));
    memset(ly, 0, sizeof(ly));
    for (int x = 0; x < n; ++x)
        for (int y = 0; y < n; ++y)
            lx[x] = max(lx[x], cost[x][y]);
}

void add (int x, int prevx)
{
    S[x] = true;
    prev[x] = prevx;
    for (int y = 0; y < n; ++y)
        if (lx[x] + ly[y] - cost[x][y] < slack[y])
        {
            slack[y] = lx[x] + ly[y] - cost[x][y];
            slackx[y] = x;
        }
}

void update_labels ()
{
    int x, y, delta = INF;
    for (y = 0; y < n; ++y)
        if (!T[y])
            delta = min(delta, slack[y]);
    for (x = 0; x < n; ++x)
        if (S[x])
            lx[x] -= delta;
    for (y = 0; y < n; ++y)
        if (T[y])
            ly[y] += delta;
    for (y = 0; y < n; ++y)
        if (!T[y])
            slack[y] -= delta;
}
```

```
void augment ()
{
    if (max_match == n)
        return;
    int x, y, root;
    int q[MAXN], wr = 0, rd = 0;
    memset(S, false, sizeof(S));
    memset(T, false, sizeof(T));
    memset(prev, -1, sizeof(prev));
    for (x = 0; x < n; ++x)
        if (xy[x] == -1)
        {
            q[wr++] = root = x;
            prev[x] = -2;
            S[x] = true;
            break;
        }
    for (y = 0; y < n; ++y)
    {
        slack[y] = lx[root] + ly[y] - cost[root][y];
        slackx[y] = root;
    }
    while (true)
    {
        while (rd < wr)
        {
            x = q[rd++];
            for (y = 0; y < n; ++y)
                if (cost[x][y] == lx[x] + ly[y] && !T[y])
                {
                    if (yx[y] == -1)
                        break;
                    T[y] = true;
                    q[wr++] = yx[y];
                    add(yx[y], x);
                }
            if (y < n)
                break;
        }
        if (y < n)
            break;
        update_labels();
        wr = rd = 0;
        for (y = 0; y < n; ++y)
            if (!T[y] && slack[y] == 0)
            {
                if (yx[y] == -1)
                {
                    x = slackx[y];
                    break;
                }
                else
                {
                    T[y] = true;
                    if (!S[yx[y]])
                    {
                        q[wr++] = yx[y];
                        add(yx[y], slackx[y]);
```

```
                    }
                }
            }
        if (y < n)
            break;
    }
    if (y < n)
    {
        ++max_match;
        for (int cx = x, cy = y, ty; cx != -2; cx = prev[cx], cy = ty)
        {
            ty = xy[cx];
            yx[cy] = cx;
            xy[cx] = cy;
        }
        augment();
    }
}

int hungarian ()
{
    int ret = 0;
    max_match = 0;
    memset(xy, -1, sizeof(xy));
    memset(yx, -1, sizeof(yx));
    init_labels();
    augment();
    for (int x = 0; x < n; ++x)
        ret += cost[x][xy[x]];
    return ret;
}
```

3.19 HOPCROFT-KARP'S MAXIMUM BIPARTITE MATCHING ALGORITHM

```
#include <stdio.h>
#include <string.h>
#include <vector>
#include <queue>
#include <iostream>
#define MAXN1 50000
#define MAXN2 50000
#define MAXM 150000

using namespace std;

/*

    HOPCROFT KARP MAXIMUM BIPARTITE MATCHING

    Aplicacoes:

        Maximum bipartite matching de forma eficiente.

    Como chamar a funcao:

        1) Chamar a funcao init(_n1,_n2), onde _n1 eh o numero
           de vertices do lado esquerdo do grafo e _n2 eh o numero
           de vertices do lado direito do grafo.

        2) Adicionar as arestas que ligam o lado esquerdo ao lado
           direito chamando a funcao void addEdge(u,v), onde u
           eh o vertice do lado esquerdo do grafo e v eh o vertice
           do lado direito do grafo.

        3) Chamar a funcao maxMatching()

    Resultado da funcao:

        A funcao maxMatching() retorna o matching maximo no grafo.

        matching[]: matching[i] -> armazena o par da esquerda do
        vertice i (da direita, obviamente)

    Complexidade do algoritmo:

        O(sqrt(n) * m)

    Problemas resolvidos:

        ENGENHAR (SPOJ BR)
        URI 1208
        URI 1056
        URI 1330

    Adicionado por:

        Jorge Gabriel SIqueira

*/
```

```
int n1, n2, edges, last[MAXN1], prev[MAXM], head[MAXM];
int matching[MAXN2], dist[MAXN1], Q[MAXN1];
bool used[MAXN1], vis[MAXN1];

void init(int _n1, int _n2)
{
    n1 = _n1;
    n2 = _n2;
    edges = 0;
    fill(last, last + n1, -1);
}

void addEdge(int u, int v)
{
    head[edges] = v;
    prev[edges] = last[u];
    last[u] = edges++;
}

void bfs()
{
    fill(dist, dist + n1, -1);
    int sizeQ = 0;
    for (int u = 0; u < n1; ++u)
    {
        if (!used[u])
        {
            Q[sizeQ++] = u;
            dist[u] = 0;
        }
    }
    for (int i = 0; i < sizeQ; i++)
    {
        int u1 = Q[i];
        for (int e = last[u1]; e >= 0; e = prev[e])
        {
            int u2 = matching[head[e]];
            if (u2 >= 0 && dist[u2] < 0)
            {
                dist[u2] = dist[u1] + 1;
                Q[sizeQ++] = u2;
            }
        }
    }
}

bool dfs(int u1)
{
    vis[u1] = true;
    for (int e = last[u1]; e >= 0; e = prev[e])
    {
        int v = head[e];
        int u2 = matching[v];
        if (u2 < 0 || !vis[u2] && dist[u2] == dist[u1] + 1 && dfs(u2))
        {
            matching[v] = u1;
            used[u1] = true;
            return true;
```

```
        }
    }
    return false;
}

int maxMatching()
{
    fill(used, used + n1, false);
    fill(matching, matching + n2, -1);
    for (int res = 0;;)
    {
        bfs();
        fill(vis, vis + n1, false);
        int f = 0;
        for (int u = 0; u < n1; ++u)
            if (!used[u] && dfs(u))
                ++f;
        if (!f)
            return res;
        res += f;
    }
}
```

3.20 STABLE MARRIAGE PROBLEM

```
#include <stdio.h>
#include <string.h>
#define MAXM 600
#define MAXW 600

using namespace std;

/*

    STABLE MARRIAGE ALGORITHM

    Aplicacoes:

        Seja um conjunto de m homens e n mulheres, onde
        cada pessoa possui uma preferencia (numero inteiro)
        por pessoas do sexo oposto. o algoritmo produz um
        casamento de cada homem com uma mulher de forma que
        todos os casamentos sejam estaveis, isto eh:

            - Cada homem se casara com uma mulher diferente (n >= m)
            - Dois casais H1M1 e H2M2 nao serao instaveis.

        Dois casais H1M1 e H2M2 sao instaveis se:

            - H1 prefere M2 ao inves de M1, e
            - M1 prefere H2 ao inves de H1.

    Como chamar a funcao:

        1) Armazenar em:

                m: O numero de homens
                n: O numero de mulheres

        2) Preencher R com as preferencias das
           mulheres da seguinte forma:

            R[x][y] = i : i eh a ordem de preferencia
                          do homem y pela mulher x

            Obs.: Quanto maior o valor de i menor eh a
                  preferencia do homem y pela mulher x

        3) Preencher L com as preferencias dos
           homens da seguinte forma:

            L[x][i] = y : A mulher y eh a i-esima
                          preferencia do homem x

            Obs.: 0 <= i <= n-1, quanto menor o valor de i
                  maior eh a preferencia do homem x pela mulher y

    Resultado da funcao:

       A funcao stableMarriage() armazena em:

            L2R[i]: a mulher do homem i (sempre entre 0 e n-1)
```

```
            R2L[j]: o homem da mulher j (-1 se a mulher for solteira)

    Complexidade do algoritmo:

        O(m^2)

    Problemas resolvidos:

        STABLEMP (SPOJ)

    Adicionado por:

        Jorge Gabriel Siqueira

*/

int L[MAXM][MAXW];
int R[MAXW][MAXM];
int L2R[MAXM], R2L[MAXW];
int m, n;
int p[MAXM];

void stableMarriage ()
{
    static int p[MAXM];
    memset(R2L, -1, sizeof(R2L));
    memset(p, 0, sizeof(p));
    for (int i = 0; i < m; ++i)
    {
        int man = i;
        while (man >= 0)
        {
            int wom;
            while (1)
            {
                wom = L[man][p[man]++];
                if (R2L[wom] < 0 || R[wom][man] > R[wom][R2L[wom]])
                    break;
            }
            int hubby = R2L[wom];
            R2L[L2R[man] = wom] = man;
            man = hubby;
        }
    }
}
```

3.21 YEN'S ALGORITHM K-TH SHORTEST PATH

```
#include <iostream>
#include <cstring>
#include <vector>
#include <queue>
#include <algorithm>
#include <functional>
#define MAX_VER  50
#define MAX_LEN 99999999

using namespace std;

/*

    YEN'S ALGORITHM K-TH SHORTEST PATH

    Aplicacoes:

        Determinar o k-esimo menor caminho em um grafo
        Obs.: O k-th menor caminho pode usar arestas do
        (k-1)-th menor caminho, porem nao pode ser identico

    Como chamar a funcao:

        1) Armazenar em n o numero de vertices do grafo e
           em m o numero de arestas do grafo.

        2) Inicializar o grafo com:

            graph.init(n);

        3) Adicionar as arestas de forma invertida no grafo:

            cin >> to >> from >> len;
            graph.addEdge(to, from, len);

        4) Armazenar a origem em source e o destino em sink

        5) Armazenar em k o valor referente a k

        6) Chamar a funcao para calcular os k menores caminhos

            vector<Path> kSh = graph.yenLoopless(sink, source, k);

    Resultado da funcao:

        Com a chamada de funcao definida em (5) o
        vector<Path> kSh: kSh[i-1] contera o i-th
        menor caminho do grafo.

        Obs.: Para nao ter problemas com crash fazer
        a verificacao:

            if (kSh.size() < k)
            {
                cout << "Nao possui kth menor caminho" << endl;
            }
```

```
            else
            {
                cout << kSh[k - 1]  << " -> " << kSh[k-1].len  << endl;
            }

        kSh[i].len contem o custo do i-th menor caminho

    Complexidade do algoritmo:

        O(?)

    Problemas resolvidos:

        POJ 3137

    Adicionado por:

        Jorge Gabriel Siqueira

*/

struct Edge
{
    int to;
    int len;
    Edge* next;
    Edge(int t = 0, int l = 0, Edge* n = NULL)
    {
        to = t;
        len = l;
        next = n;
    }
};

struct Path
{
    vector<int> node;
    vector<int> block;
    int len;
    int dev;
    Path(int v = 0) : node(), block()
    {
        node.push_back(v);
        len = 0;
    }

    bool operator > (const Path& p) const
    {
        return len > p.len || len == p.len
               && lexicographical_compare(
                   p.node.rbegin(), p.node.rend(),
                   node.rbegin(), node.rend() );
    }

    friend ostream& operator << (ostream& os, const Path& p)
    {
        os << p.node[ p.node.size() - 1 ] + 1;
        for (int i = p.node.size() - 2; i >= 0; i--)
```

```
        {
            os << "-" << p.node[i] + 1;
        }
        return os;
    }
};

struct Graph
{
    Graph()
    {
        memset(m_adj, 0, sizeof(m_adj));
    }

    void addEdge(int from, int to, int len)
    {
        if ( NULL == m_edge[from][to] )
        {
            m_adj[from] = new Edge(to, len, m_adj[from]);
            m_edge[from][to] = m_adj[from];
        }
    }

    void dijkstra()
    {
        int minV;
        for (int iter = 0; iter < m_verCnt; iter++)
        {
            minV = -1;
            for (int i = 0; i < m_verCnt; i++)
            {
                if (!m_visit[i]
                        && ( -1 == minV || m_sh[i] < m_sh[minV] )
                   )
                {
                    minV = i;
                }
            }
            if (-1 == minV)
            {
                break;
            }
            m_visit[minV] = true;
            for (Edge* adj = m_adj[minV]; adj; adj = adj->next)
            {
                int to = adj->to;
                if (!m_visit[to] && !m_block[minV][to])
                {
                    relax(minV, to, adj->len);
                }
            }
        }
    }

    void init(int n = MAX_VER)
    {
        memset( m_edge, 0, sizeof(m_edge) );
        m_verCnt = n;
```

```
        Edge* p, *temp;
        for (int i = 0; i < m_verCnt; i++)
        {
            p = m_adj[i];
            while (p)
            {
                temp = p;
                p = p->next;
                delete temp;
            }
            m_adj[i] = NULL;
        }
    }

    //Get the k loopless shortest paths with YEN's algorithm.
    //If two paths have the same length, the one whose reversed path
    //has lexicographically lower value ranks first.
    vector<Path> yenLoopless(int source, int sink, int k)
    {
        vector<Path> result;
        priority_queue< Path, vector<Path>, greater<Path> > candidate;
        memset(m_block, 0, sizeof(m_block));
        initSingleSrc(source);
        dijkstra();
        if ( shortest(sink) < MAX_LEN )
        {
            Path sh = shortestPath(sink);
            sh.dev = 1;
            sh.block.push_back( sh.node[sh.dev] );
            candidate.push(sh);
        }
        while ( result.size() < k && !candidate.empty() )
        {
            Path p = candidate.top();
            candidate.pop();
            int dev = p.dev;
            while ( dev < p.node.size() )
            {
                int pre = p.node[dev - 1];
                if (dev == p.dev)
                {
                    for (int i = 0; i < p.block.size(); i++)
                    {
                        m_block[pre][ p.block[i] ] = true;
                    }
                }
                else
                {
                    m_block[pre][ p.node[dev] ] = true;
                }
                initSingleSrc(source);
                delSubpath(p, dev);
                dijkstra();
                if (shortest(sink) < MAX_LEN)
                {
                    Path newP = shortestPath(sink);
                    newP.dev = dev;
                    if (dev == p.dev)
```

```
                    {
                        newP.block = p.block;
                    }
                    else
                    {
                        newP.block.push_back( p.node[dev] );
                    }
                    newP.block.push_back( newP.node[dev] );
                    candidate.push(newP);
                }
                dev++;
            }
            memset(m_block, 0, sizeof(m_block));
            result.push_back(p);
        }
        return result;
    }

    Path shortestPath(int v) const
    {
        Path p(v);
        p.len = m_sh[v];
        for (v = m_pre[v]; -1 != v; v = m_pre[v])
        {
            p.node.push_back(v);
        }
        reverse( p.node.begin(), p.node.end() );
        return p;
    }

    //The shortest distance from the source to v
    //(after solving the single source shortest paths).
    int shortest(int v) const
    {
        return m_sh[v];
    }

    void delSubpath(const Path& p, int dev)
    {
        int pre = p.node[0];
        m_visit[pre] = true;
        int v;
        for (int i = 1; dev != i; i++)
        {
            v = p.node[i];
            m_pre[v] = pre;
            m_sh[v] = m_sh[pre] + m_edge[pre][v]->len;
            m_visit[v] = true;
            pre = v;
        }
        m_visit[pre] = false;
    }

    //Initialize the single source shortest path algorithms.
    void initSingleSrc(int source)
    {
        for (int i = 0; i < m_verCnt; i++)
        {
```

```
            m_sh[i] = MAX_LEN;
            m_pre[i] = -1;
            m_visit[i] = false;
        }
        m_sh[source] = 0;
    }

    //Help the shortest path algorithms.
    bool relax(int from, int to, int len)
    {
        if (m_sh[to] > m_sh[from] + len)
        {
            m_sh[to] = m_sh[from] + len;
            m_pre[to] = from;
            return true;
        }
        //With the following condition, the REVERSE shortest path with be
        //the LEXICOGRAPHICALLY first one.
        else if (m_sh[to] == m_sh[from] + len && from < m_pre[to])
        {
            m_pre[to] = from;
            return true;
        }
        return false;
    }

    int m_verCnt;//Number of vertices.
    Edge* m_adj[MAX_VER];//Adjacent list.
    int m_sh[MAX_VER];//Every vertex's shortest distance from the source.
    int m_pre[MAX_VER];//The previous vertex in the shortest path.
    Edge* m_edge[MAX_VER][MAX_VER];  //m_edge[i][j]: the edge from i to j.
NULL value if no edge (i, j).
    bool m_visit[MAX_VER];//Help the dijkstra.
    bool m_block[MAX_VER][MAX_VER]; //Help to make acyclic paths.
};

Graph graph;
int k,n,m;
int source, sink;
```

3.22 TOPOLOGICAL SORT

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <queue>
#define FOI -1
#define MAX 200

using namespace std;

/*

    TOPOLOGICAL SORT

    Aplicacoes:

        Calcula a ordenacao topologica de um grafo

        Obs. Nao existe ordenacao topologica em grafos
             ciclicos

    Como chamar a funcao:

        1) Zerar o vetor de tamanho da lista de
           adjacencias pos[] e o vetor grauEntrada[]:

                memset(pos, 0, sizeof(pos));
                memset(grauEntrada, 0, sizeof(grauEntrada));

        2) Armazenar em n o numero de vertices do grafo

        3) Armazenar o grafo em forna de lista de adjacencias,
           mantendo o grau de entrada de cada vertice.

            Para armazenar uma aresta que liga o vertice x a y:

                grafo[x][pos[x]++] = y;
                grauEntrada[y]++;

        4) Chamar a funcao topSort()

    Resultado da funcao:

        A funcao armazena em top[]: top[i] o i-esimo
        elemento da ordenacao topologica.

    Complexidade do algoritmo:

        O(n+m), onde m eh  numero de arestas do grafo

    Problemas resolvidos:



    Adicionado por:

        Jorge Gabriel Siqueira
```

```
*/

int grafo[MAX][MAX], pos[MAX], grauEntrada[MAX], top[MAX], n;

void topSort()
{
    queue<int> Q;

    for(int i=0; i<n; i++)
    {
        if(!grauEntrada[i]) Q.push(i);
    }

    int posic = 0;

    while(!Q.empty())
    {
        int N = Q.front();
        Q.pop();
        top[posic++] = N;
        for(int M=0; M<pos[N]; M++)
        {
            int v = grafo[N][M];
            grauEntrada[v]--;
            if(!grauEntrada[v])
            {
                Q.push(v);
            }
        }
    }
}
```

3.23 CAMINHO MINIMO EM DÍGRAFOS ACÍCLICOS

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <algorithm>
#include <iostream>
#include <queue>
#define FOI -1
#define INF 0x3f3f3f3f
#define MAX 200

using namespace std;

/*

    SHORTEST PATH ACICLIC DIGRAPHS

    Aplicacoes:

        Calcula o caminho minimo em um grafo
        orientado sem ciclos

    Como chamar a funcao:

        1) Zerar as variaveis grafo[][] e grauEntrada[]

        2) Armazenar em n o numero de vertices do grafo

        3) Armazenar o grafo em forna de matriz de adjacencias,
           mantendo o grau de entrada de cada vertice.

            Para armazenar uma aresta que liga o vertice
            x a y com custo c:

                grafo[x][y] = c;
                grauEntrada[y]++;

        4) Chamar a funcao shortest_path(p), onde p eh o
           vertice de origem da busca

    Resultado da funcao:

        A funcao armazena em dist[]: dist[i] a distancia
        minima dovertice p ao vertice i

    Complexidade do algoritmo:

        O(n^2)

    Problemas resolvidos:



    Adicionado por:

        Jorge Gabriel Siqueira
```

```
*/


int grafo[MAX][MAX], n;
int grauEntrada[MAX];
int dist[MAX];
int top[MAX];

void topSort()
{
    queue<int> Q;

    for(int i=0; i<n; i++)
    {
        if(grauEntrada[i] == 0) Q.push(i);
    }

    int posic=0;

    while( !Q.empty())
    {
        int N = Q.front();
        Q.pop();
        top[posic++] = N;
        for(int M=0; M<n; M++)
        {
            if(grafo[N][M] != 0)
            {
                grauEntrada[M]--;
                if(grauEntrada[M] == 0)
                {
                    Q.push(M);
                }
            }
        }
    }
}

void shortest_path(int p)
{
    int i,j;
    memset(dist, INF, sizeof(dist));
    topSort();
    dist[p] = 0;

    for(i = 0; top[i] != p; i++);

    for(; i<n; i++)
        {
                for(j=i+1; j<n; j++)
                {
                        if(grafo[top[i]][top[j]]) dist[top[j]] =
min(dist[top[j]], dist[top[i]] + grafo[top[i]][top[j]]);
                }
        }
}
```

3.24 CAMINHO MINIMO COM NUMERO MAXIMO DE MOVIMENTOS

```
#include <cstdio>
#include <string>
#include <cstring>
#include <vector>
#include <algorithm>
#include <map>
#include <utility>
#include <cmath>
#include <queue>
#include <stack>
#include <set>
#include <deque>
#include <iostream>
#include <sstream>

#define INF 0x3f3f3f3f

#define MAXSZ 1000
#define MAX 300
#define MAXE 100000

using namespace std;

/*

    SHORTEST PATH LIMITED NUMBER OF MOVES

    Aplicacoes:

        Calcular o caminho minimo (se o grafo for aciclico
        pode-se calcular o caminho maximo multiplicando os
        custos por -1) com numero maximo de movimentos

    Como chamar a funcao:

        1) Armazenar em:

            n: O numero de vertices do grafo
            m: O numero de arestas do grafo
            t: O numero maximo de movimentos permitidos

        2) Chamar a funcao inicializaGrafo(n)

        3) Preencher os grafos com as arestas chamando:

            insereAresta(s,t,cst), onde:

                s: Vertice de origem
                t: Vertice de destino
                cst: Peso da aresta

        4) Chamar a funcao SPLNM(s), onde s eh o
           vertice de origem da busca

    Resultado da funcao:
```

```
        A funcao armazena em dist[]: dist[i] a distancia
        minima dovertice s ao vertice i

    Complexidade do algoritmo:

        O(n*m)

    Problemas resolvidos:

        INCIDENT (SPOJ BR)

    Adicionado por:

        Jorge Gabriel Siqueira

*/

int n, t, m;

typedef struct edge
{
    int s, t;
} edge;

int nEdges;
edge edges[MAXE];
int graph[MAX][MAX];
int dist[MAX];
int nVertices;
int nVezes[MAX];

void inicializaGrafo(int n_)
{
    memset(graph,INF,sizeof(graph));
    nEdges = 0;
    nVertices = n_;
}

void insereAresta(int s, int t, int cst)
{
    graph[s][t] = cst;

    edge temp;
    temp.s = s;
    temp.t = t;

    edges[nEdges] = temp;
    nEdges++;
}

void SPLNM(int s)
{
    int k, j;

    memset(dist, INF, sizeof(dist));
    memset(nVezes, 0, sizeof(nVezes));
    dist[s] = 0;
```

```
    for (k = 0; k < nVertices-1; ++k)
    {
        for (j = 0; j < nEdges; ++j)
        {
            int u = edges[j].s;
            int v = edges[j].t;

            if (dist[u] < INF && dist[v] > dist[u] + graph[u][v])
            {
                if(nVezes[u] + 1 <= t)
                {
                    dist[v] = dist[u] + graph[u][v];
                    nVezes[v] = nVezes[u] + 1;
                }
            }
        }
    }
}
```

3.25 ERDOS GALLAI LINEAR

```
#include <cstdio>
#include <string>
#include <cstring>
#include <vector>
#include <algorithm>
#include <map>
#include <utility>
#include <cmath>
#include <queue>
#include <stack>
#include <set>
#include <deque>
#include <iostream>
#include <math.h>
#include <assert.h>
#include <sstream>
#define INF 0x3f3f3f3f
#define MAX 100000

using namespace std;


/*

    ERDOS GALLAI LINEAR

    Aplicacoes:

        Determinar se um conjunto b de n elementos pode
        representar o grau dos vertices de um grafo de n
        elementos, ou seja, b[1] == grau do vertice 1,
        b[2] == grau do vertice[2], ...

    Como chamar a funcao:

        1) Armazenar o tamanho (quantidade de elementos)
           do vetor b na variavel n.

        2) Preencher o vetor b partindo da posicao 1
           ate a posicao n (inclusive)

        3) Chamar a funcao EGH()

    Resultado da funcao:

        A funcao EGL retorna:

            true -> se b pode representar o grau dos
                    n vertices de um grafo

            false -> caso contrario

    Complexidade do algoritmo:

        O(n)
```

```
    Problemas resolvidos:

        URI 1462
        UVA 10720

    Adicionado por:

        Jorge Gabriel Siqueira

*/

long long b[MAX], n;
long long dmax,dmin,dsum,num_degs[MAX];

bool basic_graphical_tests() //Sort and perform some simple tests on the
sequence
{
    int p = n;

    memset(num_degs, 0, (n+1)*sizeof(long long));

    dmax = dsum = n = 0;
    dmin = p;

    for(int d=1; d<=p; d++)
    {
        if(b[d] < 0 || b[d] >= p)
        {
            return false;
        }
        else if(b[d] > 0)
        {
            if(dmax < b[d]) dmax = b[d];
            if(dmin > b[d]) dmin = b[d];
            dsum = dsum+b[d];
            n++;
            num_degs[b[d]]++;
        }
    }

    if(dsum%2 || dsum > n*(n-1)) return false;
    return true;
}

bool EGL()
{
    if(basic_graphical_tests() == false) return false;

    if(n == 0 || 4*dmin*n >= (dmax+dmin+1)*(dmax+dmin+1)) return true;

    long long k,sum_deg,sum_nj,sum_jnj,run_size;

    k = sum_deg = sum_nj = sum_jnj = 0;

    for(int dk = dmax; dk >= dmin; dk--)
    {
        if(dk < k+1) return true;
```

```
        if(num_degs[dk] > 0)
        {
            run_size = num_degs[dk];
            if(dk < k+run_size)
            {
                run_size = dk-k;
            }
            sum_deg += run_size * dk;

            for(int v=0; v<run_size; v++)
            {
                sum_nj += num_degs[k+v];
                sum_jnj += (k+v) * num_degs[k+v];
            }

            k += run_size;

            if(sum_deg > k*(n-1) - k*sum_nj + sum_jnj)
            {
                return false;
            }
        }
    }
    return true;
}

int main()
{
    n = 3;
    b[1] = 1; b[2] = 1; b[3] = 1;
    assert(EGL() == false);

    n = 3;
    b[1] = 2; b[2] = 2; b[3] = 2;
    assert(EGL() == true);

    n = 4;
    b[1] = 1; b[2] = 1; b[3] = 1; b[4] = 1;
    assert(EGL() == true);

    n = 4;
    b[1] = 3; b[2] = 2; b[3] = 2; b[4] = 1;
    assert(EGL() == true);

    n = 4;
    b[1] = 3; b[2] = 3; b[3] = 2; b[4] = 2;
    assert(EGL() == true);

    n = 4;
    b[1] = 3; b[2] = 3; b[3] = 1; b[4] = 1;
    assert(EGL() == false);
    return 0;
}
```

3.26 MINIMUM STEINER TREE

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

using namespace std;

/*

    MINIMUM STEINER TREE

    Aplicacoes:

        Determinar uma arvore de custo total minimo que
        interonecte um conjunto de vertices de um grafo.

    Como chamar a funcao:

        1) Armazenar em n o numero de vertices do grafo

        2) Chamar a funcao initialize()

        3) Armazenar em T os vertices que a arvore deve possuir

        4) Preencher a matriz de adjacencias (g[][]) com as
           informacoes referentes ao grafo.

        6) Chamar a funcao minimum_steiner_tree();

    Resultado da funcao:

        A funcao retorna o custo da arvore minima que interconecta
        os vertices em T

    Complexidade do algoritmo:

        O(n*log(n))

    Problemas resolvidos:

        AIZU ONLINE JUDGE 1040

    Adicionado por:

        Jorge Gabriel Siqueira

*/

typedef vector< vector<int> > matrix;

int n;
matrix g;
vector<int> T;

void initialize()
{
```

```
    g = matrix(n, vector<int>(n,INF));
    T.clear();
}

int minimum_steiner_tree()
{
    const int numT = T.size();

    if (numT <= 1) return 0;

    matrix d(g); // all-pair shortest

    for (int k = 0; k < n; ++k)
        for (int i = 0; i < n; ++i)
            for (int j = 0; j < n; ++j)
                d[i][j] = min( d[i][j], d[i][k] + d[k][j] );

    int OPT[(1 << numT)][n];

    for (int S = 0; S < (1 << numT); ++S)
        for (int x = 0; x < n; ++x)
            OPT[S][x] = INF;

    for (int p = 0; p < numT; ++p) // trivial case
        for (int q = 0; q < n; ++q)
            OPT[1 << p][q] = d[T[p]][q];

    for (int S = 1; S < (1 << numT); ++S)    // DP step
    {
        if (!(S & (S-1))) continue;
        for (int p = 0; p < n; ++p)
            for (int E = 0; E < S; ++E)
                if ((E | S) == S)
                    OPT[S][p] = min( OPT[S][p], OPT[E][p] + OPT[S-E][p] );

        for (int p = 0; p < n; ++p)
            for (int q = 0; q < n; ++q)
                OPT[S][p] = min( OPT[S][p], OPT[S][q] + d[p][q] );
    }

    int ans = INF;

    for (int S = 0; S < (1 << numT); ++S)
        for (int q = 0; q < n; ++q)
            ans = min(ans, OPT[S][q] + OPT[((1 << numT)-1)-S][q]);

    return ans;
}
```

3.27 FLOYD WARSHALL COM PATH RECONSTRUCTION

```
#include <cstdio>
#include <string>
#include <cstring>
#include <vector>
#include <algorithm>
#include <map>
#include <utility>
#include <cmath>
#include <queue>
#include <stack>
#include <set>
#include <deque>
#include <iostream>
#include <math.h>
#include <sstream>
#include <assert.h>
#include <numeric>
#include <fstream>
#include <limits>
#define INF 0x3f3f3f3f
#define MAX 105

using namespace std;

/*

    FLOYD WARSHALL WITH PATH RECONSTRUCTION

    Aplicacoes:

        Calcular o custo e imprimir o menor caminho entre quaisquer
        vertices de um grafo.

    Como chamar a funcao:

        1) Armazenar em n o numero de vertices do grafo

        2) Inicializar grafo[][] da seguinte forma:
           - grafo[i][i] = 0, para todo i
           - grafo[i][j] = INF, se nao existir uma aresta entre i e j
           - grafo[i][j] = c, se existir uma aresta entre i e j de custo c

        3) Chamar a funcao FloydWarshallWithPathReconstruction() para
           calcular o menor caminho de todos os vertices para todos
           vertices.

        4) Chamar a funcao printPath(s,t) para imprimir o menor caminho
           partindo do vertice s ate o vertice t

    Resultado da funcao:

        A funcao FloydWarshallWithPathReconstruction()
        armazena em dist[][] : dist[s][t] a menor distancia
        entre partindo do vertice s ate o vertice t

    Complexidade do algoritmo:
```

```
        O(n^3)

    Problemas resolvidos:

        URI 1427

    Adicionado por:

        Jorge Gabriel Siqueira

*/

int n;
int dist[MAX][MAX];
int next[MAX][MAX];
int grafo[MAX][MAX];

void FloydWarshallWithPathReconstruction()
{
    for(int i=0; i<n; i++)
    {
        for(int j=0; j<n; j++)
        {
            dist[i][j] = grafo[i][j];
        }
    }

    memset(next, -1, n*sizeof(int));

    for(int k=0; k<n; k++)
    {
        for(int i=0; i<n; i++)
        {
            for(int j=0; j<n; j++)
            {
                if(dist[i][k] + dist[k][j] < dist[i][j])
                {
                    dist[i][j] = dist[i][k] + dist[k][j];
                    next[i][j] = k;
                }
            }
        }
    }
}

void Path(int i, int j)
{
    if(dist[i][j] == INF) cout << "no path!";

    int intermediate = next[i][j];

    if(intermediate == -1) cout << " ";
    else
    {
        Path(i, intermediate);
        cout << intermediate;
        Path(intermediate, j);
    }
```

```
}

void printfPath(int s, int t)
{
    cout << "Path:" << s;
    Path(s,t);
    cout << t << endl;
}
```

4. MATEMÁTICA

4.1 GCD

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

using namespace std;

/*

    GCD - GREATEST COMMON DIVISOR

    Aplicacoes:

        Determinar o maior divisor comum (MDC)
        entre dois numeros inteiros positivos a e b.

    Como chamar a funcao:

        1) Chamar a funcao gcd(a,b)

    Resultado da funcao:

        A funcao retorna o maior divisor comum entre a e b

    Complexidade do algoritmo:

        O(?)

    Problemas resolvidos:

        URI 1028

    Adicionado por:

        Jorge Gabriel Siqueira

*/

inline int gcd (int a, int b)
{
    return b ? gcd(b, a % b) : abs(a);
}
```

4.2 LCM

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

/*

    LCM - LEAST COMMON MULTIPLE

    Aplicacoes:

        Determinar o menor multiplo comum (MMC)
        entre dois numeros inteiros positivos a e b

    Como chamar a funcao:

        1) Chamar a funcao lcm(a,b)

    Resultado da funcao:

        A funcao retorna o menor multiplo comum entre a e b

    Complexidade do algoritmo:

        O(?)

    Problemas resolvidos:

    Adicionado por:

        Jorge Gabriel Siqueira

*/

inline int gcd (int a, int b)
{
    return b ? gcd(b, a % b) : abs(a);
}

inline long long lcm(int a, int b)
{
    if (a && b)
        return abs(a) / gcd(a, b) * (long long) abs(b);
    else
        return (long long) abs(a | b);
}
```

4.3 EXTENDED EUCLIDEAN ALGORITHM

```
#include <stdio.h>

/*

    EXTENDED EUCLIDEAN ALGORITHM (BEZOUT THEOREM)

    Aplicacoes:

        Determinar a solucao da equacao
        a*x + b*y = gcd(a,b), onde a e b sao dois
        numeros inteiros nao negativos.

    Como chamar a funcao:

        1) Chamar a funcao egcd(a,b)

    Resultado da funcao:

        A funcao retorna a tripla(gcd(a,b), x, y)

    Complexidade do algoritmo:

        O(?)

    Problemas resolvidos:

    Adicionado por:

        Jorge Gabriel Siqueira

*/

struct Triple
{
    int d, x, y;
    Triple (int q, int w, int e) : d(q), x(w), y(e) {}
};

Triple egcd (int a, int b)
{
    if (!b)
        return Triple(a, 1, 0);

    Triple q = egcd(b, a % b);
    return Triple(q.d, q.y, q.x - a / b * q.y);
}
```

4.4 EQUAÇÕES DIOFANTINAS LINEARES

```
#include <stdio.h>

/*

    EQUACOES DIOFANTINAS LINEARES

    Aplicacoes:

        Determinar todos os valores inteiros de x e y
        que satisfazem a equacao diofantina linear:

            a*x + b*y = c.

    Como chamar a funcao:

        1) Chamar a funcao ldioph(a,b,c)

    Resultado da funcao:

        A funcao ldioph(a,b,c) retorna a tripla (d, x, y),
        onde:

            se d == 0, entao a equacao nao possui solucao.

            se d != 0, entao a equacao possui infinitas
            solucoes da forma:

                x = tripla.x + k * b/tripla.d,
                y = tripla.y - k * a/tripla.d,
                para todo inteiro k.

    Complexidade do algoritmo:

        O(?)

    Problemas resolvidos:

    Adicionado por:

        Jorge Gabriel Siqueira

*/

struct Triple
{
    int d, x, y;
    Triple (int q, int w, int e) : d(q), x(w), y(e) {}
};

Triple egcd (int a, int b)
{
    if (!b)
        return Triple(a, 1, 0);

    Triple q = egcd(b, a % b);
    return Triple(q.d, q.y, q.x - a / b * q.y);
}
```

```
Triple ldioph (int a, int b, int c)
{
    Triple t = egcd(a, b);

    if (c % t.d)
        return Triple(0, 0, 0);

    t.x *= c / t.d;
    t.y *= c / t.d;

    return t;
}
```

4.5 INVERSO MODULAR

```
#include <stdio.h>

/*

    INVERSO MODULAR

    Aplicacoes:

        Determinar a solucao da equacao a*x = 1 (mod n).
        se n eh primo, entao x = (a^n-2) % n.

    Como chamar a funcao:

        1) Chamar a funcao invMod(a,n)

    Resultado da funcao:

        A funcao invMod(a,n) retorna o valor de x,
        ou 0 caso nao possua solucao

    Complexidade do algoritmo:

        O(?)

    Problemas resolvidos:

    Adicionado por:

        Jorge Gabriel Siqueira

*/
struct Triple
{
    int d, x, y;
    Triple (int q, int w, int e) : d(q), x(w), y(e) {}
};
Triple egcd (int a, int b)
{
    if (!b)
        return Triple(a, 1, 0);

    Triple q = egcd(b, a % b);
    return Triple(q.d, q.y, q.x - a / b * q.y);
}
int invMod (int a, int n)
{
    Triple t = egcd (a, n);

    if (t.d > 1)
        return 0;

    int r = t.x % n;

    return (r < 0 ? r + n : r );
}
```

4.6 EXPONENCIAÇÃO MODULAR

```
#include <stdio.h>

/*

    EXPONENCIACAO MODULAR

    Aplicacoes:

        Determinar a^k (mod n) de forma eficiente.

    Como chamar a funcao:

        1) Chamar a funcao powMod(a,k,n)

    Resultado da funcao:

        A funcao powMod(a,k,n) retorna o valor de a^k (mod n)

    Complexidade do algoritmo:

        O(log(k))

    Problemas resolvidos:

    Adicionado por:

        Jorge Gabriel Siqueira

*/

//metodo nao recursivo
long long powMod (long long a, long long k, long long n)
{
    long long ret = 1;

    for (long long pow = a; k > 0; k >>= 1, pow = (pow * pow) % n)
        if (k & 1)
            ret = (ret * pow) % n;

    return ret;
}

//metodo recursivo
long long powMod (long long a, long long k, long long n)
{
    if (k == 0)
        return 1;
    if (k%2 == 0)
        return powMod(a * a % n , k/2, n);
    else
        return a * powMod(a, k - 1 ,n) % n;
}
```

4.7 CHINESE REMAINDER THEOREM

```
#include <cstdio>
#define MAXN 1000

/*

    CHINESE REMAINDER THEOREM

    Aplicacoes:

        Determinar x tal que x = a[i] (mod p[i]).

        Exemplo: Para a[] = {1,2,3} e p[] = {5,6,7}
        determinar x tal que:

            x = 1 (mod 5)
            x = 2 (mod 6)
            x = 3 (mod 7)

        resposta: x = 206

    Como chamar a funcao:

        1) Chamar a funcao crt()

    Resultado da funcao:

        A funcao retorna o valor de x.

    Complexidade do algoritmo:

        O(?)

    Problemas resolvidos:

        UVA 756

    Adicionado por:

        Jorge Gabriel Siqueira

*/

int n;
int a[MAXN], p[MAXN];

struct Triple
{
    int d, x, y;
    Triple (int q, int w, int e) : d(q), x(w), y(e) {}
};

Triple egcd (int a, int b)
{
    if (!b)
        return Triple(a, 1, 0);

    Triple q = egcd(b, a % b);
```

```
    return Triple(q.d, q.y, q.x - a / b * q.y);
}

int invMod (int a, int n)
{
    Triple t = egcd (a, n);

    if (t.d > 1)
        return 0;

    int r = t.x % n;

    return (r < 0 ? r + n : r );
}

int crt()
{
    int M = 1, x = 0;

    for (int i = 0; i < n; ++i)
        M *= p[i];

    for (int i = 0; i < n; ++i)
        x += a[i] * invMod(M/p[i],p[i]) * (M/p[i]);

    return (((x%M) + M)%M);
}
```

4.8 BINOMIAL

```
#include <stdio.h>
#define MAXN 80

/*

    BINOMIAL

    Aplicacoes:

        Calculo do triangulo de pascal ate ordem maxima MAXN.
        C[n][k] = valor da combinacao C(n,k), onde k <= n.

    Como chamar a funcao:

        1) Chamar a funcao pascalTriangle()

    Resultado da funcao:

        A funcao pascalTriangle() armazena em C[][]:
        C[i][j] a combinacao C[i][j], note que a combinacao
        tambem representa o triangulo de pascal.

        0  : 1 |_____
        1  : 1 | 1  |_____
        2  : 1 | 2  | 1  |_____
        3  : 1 | 3  | 3  | 1   |_____
        4  : 1 | 4  | 6  | 4   | 1   |_____
        5  : 1 | 5  | 10 | 10  | 5   | 1   |_____
        6  : 1 | 6  | 15 | 20  | 15  | 6   | 1   |_____
        7  : 1 | 7  | 21 | 35  | 35  | 21  | 7   | 1   |_____
        8  : 1 | 8  | 28 | 56  | 70  | 56  | 28  | 8   | 1   |_____
        9  : 1 | 9  | 36 | 84  | 126 | 126 | 84  | 36  | 9   | 1  |
        10 : 1 | 10 | 45 | 120 | 210 | 252 | 210 | 120 | 45  | 10 | 1 |

        C(33, 16) = 1.166.803.110 [limite do int]
        C(34, 17) = 2.333.606.220 [limite do unsigned int]
        C(66, 33) = 7.219.428.434.016.265.740 [limite do int64_t]
        C(67, 33) = 14.226.520.737.620.288.370 [limite do uint64_t]

    Complexidade do algoritmo:

        O(MAXN^2)

    Problemas resolvidos:

    Adicionado por:

        Jorge Gabriel Siqueira

*/

unsigned long long C[MAXN+1][MAXN+1];

void pascalTriangle()
{
    unsigned long long n, k;
```

```
    for (n = 0; n <= MAXN; ++n)
    {
        C[n][0] = C[n][n] = 1;

        for (k = 1; k < n; ++k)
            C[n][k] = C[n-1][k-1] + C[n-1][k];
    }
}
```

4.9 MÖBIUS FUNCTION

```
#include <cstdio>
#define MAXN 2000000 //valor maximo de n

/*

    MOBIUS FUNCTION

    Aplicacoes:

        Determinar o valor de mu(n), onde mu eh a
        funcao de Mobius:

            mu(n) = 0, se n nao eh uma raiz livre;

            mu(n) = 1, se n e uma raiz livre com um
                       numero par de fatores primos

            mu(n) = -1, se n eh uma raiz livre com um
                        numero impar de fatores primos

        Obs: uma raiz livre eh aquela que nao eh divisivel
             por um quadrado perfeito.

        Exemplo: 10 (2*5) eh uma raiz livre, mas 18 (2*3*3) nao!

    Como chamar a funcao:

        1) Chamar a funcao mu(n)

    Resultado da funcao:

        A funcao retorna o resultado da funcao Mobius

    Complexidade do algoritmo:

        O(?)

    Problemas resolvidos:

    Adicionado por:

        Jorge Gabriel Siqueira

*/

int mu(int n)
{
    static int lookup = 0, p[MAXN], f[MAXN];

    if (!lookup)
    {
        for (int i = 0; i < MAXN; ++i)
            p[i] = 1, f[i] = 1;

        for (int i = 2; i < MAXN; ++i)
        {
            if (p[i])
```

```
            {
                f[i] = -1;

                for (int j = i + i; j < MAXN; j += i)
                {
                    p[j] = 0;
                    f[j] *= (j % (i * i) == 0) ? 0 : -1;
                }
            }
        }

        lookup = 1;
    }

    return f[n];
}
```

4.10 BABY STEP GIANT STEP

```
#include <stdio.h>
#include <math.h>
#include <map>

using namespace std;

/*

    BABY STEP - GIANT STEP

    Aplicacoes:

        Determinar o menor valor de e na
        expressao b^e = n (mod p)

    Como chamar a funcao:

        1) Chamar a funcao bsgs(b,n,p)

    Resultado da funcao:

        A funcao bsgs(b,n,p) retorna o menor
        valor de e ou -1, se a equacao nao
        possuir solucao.

    Complexidade do algoritmo:

        O(?)

    Problemas resolvidos:

        UVA 10225

    Adicionado por:

        Jorge Gabriel Siqueira

*/

struct Triple
{
    int d, x, y;
    Triple (int q, int w, int e) : d(q), x(w), y(e) {}
};

Triple egcd (int a, int b)
{
    if (!b)
        return Triple(a, 1, 0);

    Triple q = egcd(b, a % b);
    return Triple(q.d, q.y, q.x - a / b * q.y);
}

int invMod (int a, int n)
{
    Triple t = egcd (a, n);
```

```
    if (t.d > 1)
        return 0;

    int r = t.x % n;

    return (r < 0 ? r + n : r );
}

long long bsgs (long long b, long long n, long long p)
{
    if (n == 1)
        return 0;

    map <long long, int> table;

    long long m = sqrt(p) + 1, pot = 1, pot2 = 1;

    for (int j = 0; j < m; ++j)
    {
        if (pot == n)
            return j;

        table[(n * invMod(pot, p)) % p] = j;
        pot = (pot * b) % p;
    }

    for (int i = 0; i < m; ++i)
    {
        if (table.find(pot2) != table.end())
            return i * m + table[pot2];
        pot2 = (pot * pot2) % p;
    }

    return -1;
}
```

4.11 PRIMOS EM UM INTERVALO

```
#include <stdio.h>
#include <vector>

using namespace std;

/*

    PRIMOS EM UM INTERVALO

    Aplicacoes:

        Determinar os numero primos contidos em um
        intervalo [n,m]

    Como chamar a funcao:

        1) Chamar a funcao primesBetween(n,m)

    Resultado da funcao:

        A funcao primesBetween(n,m) armazena em vector<int> ret
        os primos contidos no intervalo [n,m]

    Complexidade do algoritmo:

        O(?)

    Problemas resolvidos:

        PRIME1 (SPOJ)

    Adicionado por:

        Jorge Gabriel Siqueira

*/

vector<int> ret;

void primesBetween (int n, int m)
{
    ret.clear();
    int* primes = new int[m - n + 1];

    for (int i = 0; i < m - n + 1; ++i)
        primes[i] = 0;

    for (int p = 2; p*p <= m; ++p)
    {
        int less = n / p;
        less *= p;

        for (int j = less; j <= m; j += p)
            if (j != p && j >= n)
                primes[j - n] = 1;
    }
```

```
    for (int i = 0; i < m - n + 1; ++i)
    {
        if (primes[i] == 0 && n+i != 1)
            ret.push_back(n+i);
    }

    delete primes;
}
```

4.12 CRIVO DE ERASTÓTENES

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#define MAXN 10000000
#define gP(n)(prime[n>>6]&(1<<((n>>1)&31)))
#define rP(n)(prime[n>>6]&=~(1<<((n>>1)&31)))

unsigned int prime[MAXN / 64];

/*

    CRIVO DE ERASTOTENES

    Aplicacoes:

        Determinar os numeros primos no intervalo [3,MAXN]
        de forma eficiente, necessitando apenas de MAXN / 16
        bytes de memoria.

        Observacao: O algoritmo funciona apenas para numeros
        impares. Tratar numeros pares separadamente!

    Como chamar a funcao:

        1) Chamar a funcao sieve() para preencher a tabela.

        3) Para descobrir se um numero n eh primo chame
           a funcao gP(n).

    Resultado da funcao:

        A funcao retorna:

            0 se o numero nao eh primo
            Algo diferente de 0 caso contrario

    Complexidade do algoritmo:

        O(?)

    Problemas resolvidos:

        DINOSTRA (SPOJ BR)

    Adicionado por:

        Jorge Gabriel Siqueira

*/

void sieve()
{
    memset(prime, -1, sizeof(prime));
    unsigned int i;
    unsigned int sqrtN = (unsigned int) sqrt((double) MAXN) + 1;

    for (i = 3; i < sqrtN; i += 2)
```

```
        if (gP( i ))
        {
            unsigned int i2 = i + i;
            for (unsigned int j = i * i; j < MAXN; j += i2)
                rP(j);
        }
}
```

4.13 DECOMPOSIÇÃO EM FATORES PRIMOS

```cpp
#include <stdio.h>
#include <math.h>
#include <vector>

using namespace std;

/*

    DECOMPOSICAO EM FATORES PRIMOS

    Aplicacoes:

        Determinar a fatoracao em numeros primos de
        um numero n.

        Exemplo:

            factorize(18) = {2, 3, 3}

    Como chamar a funcao:

        1) Chamar a funcao factorize(n)

    Resultado da funcao:

        A funcao factorize() retorna um vector <int> que
        contem os fatores primos resultantes da fatoracao
        do numero n.

    Complexidade do algoritmo:

        O(sqrt(n))

    Problemas resolvidos:

    Adicionado por:

        Jorge Gabriel Siqueira

*/

void factorize (int n, vector <int> &v)
{
    int sq = int (sqrt((double)n));
    for (int i = 2; i <= sq; ++i)
    {
        if (n % i)
            continue;
        v.push_back (i);
        n /= i--;
        sq = int (sqrt((double) n));
    }
    if (n > 1)
        v.push_back(n);
}
```

4.14 EULER TOTIENT (PHI)

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

using namespace std;

/*

    EULER TOTIENT - PHI

    Aplicacoes:

        Dado um numero n, contar o numero de inteiros
        positivos menores ou iguais a n que sao co-primos de n.

        Dois numeros a e b sao co-primos se gcd(a,b) = 1

    Como chamar a funcao:

        1) Chamar a funcao phi(n)

    Resultado da funcao:

        A funcao retorna o numero de inteiros positivos
        menores ou iguais a n que sao co-primos de n

    Complexidade do algoritmo:

        O (sqrt(n))

    Problemas resolvidos:

        UVA 12493

    Adicionado por:

        Jorge Gabriel Siqueira

*/

void factorize (int n, vector <int> &v)
{
    int sq = int (sqrt((double)n));
    for (int i = 2; i <= sq; ++i)
    {
        if (n % i)
            continue;
        v.push_back (i);
        n /= i--;
        sq = int (sqrt((double) n));
    }
    if (n > 1)
        v.push_back(n);
}

int phi (int n)
{
```

```
    vector <int> p;

    factorize (n, p);

    for (int i = 0; i < p.size(); ++i)
    {
        if (i && p[i] == p[i - 1])
            continue;
        n /= p[i];
        n *= p[i] - 1;
    }

    return n;
}
```

4.15 NÚMERO DE DIVISORES DE UM NÚMERO

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

using namespace std;

/*

    NUMERO DE DIVISORES DE UM NUMERO

    Aplicacoes:

        Contar o numero de divisores positivos de um dado
        numero n, incluindo 1 e ele mesmo.

    Como chamar a funcao:

        1) Chamar a funcao divisors(n)

    Resultado da funcao:

        A funcao retorna o numero de divisores de n,
        incluindo 1 e ele mesmo.

    Complexidade do algoritmo:

        O(?)

    Problemas resolvidos:

    Adicionado por:

        Jorge Gabriel Siqueira

*/

void factorize (int n, vector <int> &v)
{
    int sq = int (sqrt((double)n));

    for (int i = 2; i <= sq; ++i)
    {
        if (n % i)
            continue;

        v.push_back (i);
        n /= i--;
        sq = int (sqrt((double) n));
    }

    if (n > 1)
        v.push_back(n);
}

int divisors (int n)
{
    vector <int> f;
```

```
    factorize (n, f);

    int k = f.size();

    vector <int> table(k + 1, 0);
    table[k] = 1;

    for (int i = k - 1; i >= 0; --i)
    {
        table[i] = table[i + 1];

        for (int j = i + 1; ; ++j)
            if (j == k || f[j] != f[i])
            {
                table[i] += table[j];
                break;
            }
    }

    return table[0];
}
```

4.16 SUBCOLEÇÃO DISJUNTA MÁXIMA

```
#include <stdio.h>
#include <algorithm>
#include <map>
#include <iostream>
#include <set>
#include <vector>

using namespace std;

/*

    Aplicacoes:

        Dados um conjunto de tarefas determinandas
        por seus momentos de inicio e fim, determinar a
        subcolecao maxima destas tarefas de modo a
        maximizar o numero de tarefas realizadas

        Obs. Armazenar o conjunto de tarefas na variavel
        tarefas de forma inversa, ou seja, o momento de
        fim da i-esima tarefa deve ser armazendo em
        tarefas[i].first e o momento de inicio da i-esima
        tarefa deve ser armazenado em tarefas[i].second

*/
vector< pair<int,int> > tarefas; //Armazenar em first o termino e em
second o inicio

int sdm()
{
    if(tarefas.empty()) return 0;

    sort(tarefas.begin(), tarefas.end());

    int cont=1;
    int i = 0;

    for(int k=1; k<tarefas.size(); k++)
    {
        if(tarefas[k].second > tarefas[i].first)
        {
            cont++;
            i = k;
        }
    }

    return cont;
}
```

4.17 BRENT CICLE DETECTION

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

using namespace std;

/*

    BRENT CYCLE DETECTION ALGORITHM

    Aplicacoes:

        Dada uma funcao y = f(x) e um valor inicial x0,
        determinar as caracteristicas do ciclo presente
        na sequencia (x0, f(x0), f(f(x0)), f(f(f(x0))),..)

    Como chamar a funcao:

        1) Definir a funcao em function()

        2) Chamar brent(x0)

    Resultado da funcao:

        A funcao brent() retorna um pair<int,int> :

            first -> O indice a partir do qual o ciclo
                     se inicia.

            second -> O tamanho do ciclo.

    Complexidade do algoritmo:

        O(mu + lambda), onde mu eh o indice a partir
        do qual o ciclo se inicia e lambda eh o
        tamanho do ciclo.

    Problemas resolvidos:

        UVA 350

    Adicionado por:

        Jorge Gabriel Siqueira

*/

int z, j, m;

int function (int x)
{
    return (z*x + j)%m;
}

pair<int,int> brent(int x0)
{
    int p = 1, l = 1;
    int t = x0;
```

```
    int h = function(x0);
    //determina o tamanho do ciclo
    while (t != h)
    {
        if (p == l)
        {
            t = h;
            p *= 2;
            l = 0;
        }
        h = function(h);
        ++l;
    }
    //determina o indice de inicio do ciclo

    int u = 0;
    t = h = x0;

    for (int i = l; i != 0; --i)
        h = function(h);

    while (t != h)
    {
        t = function(t);
        h = function(h);
        ++u;
    }

    return make_pair(u, l);
}
```

4.18 BASKARA

```cpp
#include <stdio.h>
#include <algorithm>
#include <math.h>
#include <iostream>
#define INF 0x3f3f3f3f

using namespace std;

/*

    BHASKARA QUADRATIC POLYNOMIALS ROOTS

    Aplicacoes:

        Encontrar as raizes de uma equacao de segundo grau

    Como chamar a funcao:

        1) Chamar a funcao baskara(a,b,c), onde:

            a -> multiplicado de x^2
            b -> multiplicado de x^1
            c -> multiplicado de x^0

    Resultado da funcao:

        A funcao retorna um pair<int,int> contendo
        em first e second as duas raizes da equacao

        Caso a equacao nao possua raizes a funcao retorna
        pair<int,int>(INF,INF);

    Complexidade do algoritmo:

        O(1)

    Problemas resolvidos:

    Adicionado por:

        Jorge Gabriel Siqueira

*/

pair<double,double> bhaskara(double a, double b, double c)
{
    double delta = b*b - 4*a*c;
    if(delta < 0) return pair<int,int>(INF,INF);
    double x1 = (-b + sqrt(delta))/(2.0*a);
    double x2 = (-b - sqrt(delta))/(2.0*a);
    return pair<double,double>(x1,x2);
}

int main()
{
    //8000 -720 12
    pair<double,double> root = bhaskara(12,-720,8000);
```

```
    cout << root.first << endl << root.second << endl;
}
```

4.19 MILLER RABIN

```
#include <stdio.h>

/*

    MILLER RABIN

    Aplicacoes:

        Determinar se um numero n eh provavelmente primo.
        Ideal para testes de primalidade de numeros
        extremamente grandes.

    Como chamar a funcao:

        1) Chamar isProbablePrime(n)

    Resultado da funcao:

        A funcao isProbablePrime() retorna:

            true: Se n eh provavelmente primo
            false: Caso contrario

    Complexidade do algoritmo:

        O (?)

    Problemas resolvidos:

        PRIMO (SPOJ BR)
        URI 1221

    Adicionado por:

        Jorge Gabriel Siqueira

*/

long long powMod (long long a, long long k, long long n)
{
    long long ret = 1;

    for (long long pow = a; k > 0; k >>= 1, pow = (pow * pow) % n)
        if (k & 1)
            ret = (ret * pow) % n;

    return ret;
}

bool miller_rabin (long long n, long long base)
{
    if (n <= 1)
        return false;

    if (n%2 == 0)
        return n == 2;
```

```
    long long s = 0, d = n - 1;

    while (d%2 == 0)
        d /= 2, ++s;

    long long base_d = powMod(base, d, n);

    if (base_d == 1)
        return true;

    long long base_2r = base_d;

    for (long long i = 0; i < s; ++i)
    {
        if (base_2r == 1)
            return false;

        if (base_2r == n - 1)
            return true;

        base_2r = (long long) base_2r * base_2r % n;
    }

    return false;
}
bool isProbablePrime (long long n)
{
    if (n == 2 || n == 7 || n == 61)
        return true;

    return miller_rabin(n, 2) && miller_rabin(n, 7) && miller_rabin(n, 61);
}
```

4.20 POLLARD RHO

```
#include <stdio.h>
#include <time.h>
#include <algorithm>
#include <iostream>

using namespace std;

/*

    POLLARD RHO

    Aplicacoes:

        Fatorar numeros grandes de forma eficiente.

        Obs: O metodo so funciona para numeros compostos
        (testar a primalidade antes!)

        Usar em conjunto com MILLER RABIN

        Para fatorar um numero, deve-se chamar a funcao
        pollard() sucessivamente ate que o seu resultado
        seja um numero primo.

    Como chamar a funcao:

        1) Chamar pollard(n)

    Resultado da funcao:

        A funcao pollard() retorna um fator nao-trivial
        do numero que esta sendo fatorado

    Complexidade do algoritmo:

        O(?)

    Problemas resolvidos:

    Adicionado por:

        Jorge Gabriel Siqueira

*/

long long pollard_r, pollard_n;

inline long long f (long long val)
{
    return (val*val + pollard_r) % pollard_n;
}

inline long long myabs (long long a)
{
    return a >= 0 ? a : -a;
}
```

```
long long pollard (long long n)
{
    srand(unsigned(time(0)));
    pollard_n = n;
    long long d = 1;

    do
    {
        d = 1;
        pollard_r = rand() % n;
        long long x = 2, y = 2;
        while (d == 1)
            x = f(x), y = f(f(y)), d = __gcd(myabs(x-y), n);
    }

    while (d == n);

    return d;
}
```

4.21 SIMPLEX

```
#include <stdio.h>
#include <vector>
#include <algorithm>
#include <iostream>

using namespace std;

/*
    3.19 Simplex

    Aplicacoes:

       determinar (x0, x1, ..., xn) de forma a maximizar
       P = c[0]*x0 + c[1]*x1 + ... + c[n]*xn, com as seguintes restricoes

       A[0][0]*x0 + A[0][1]*x1 + ... + A[0][n]*xn <= b0
       A[1][0]*x0 + A[1][1]*x1 + ... + A[1][n]*xn <= b1
       ..................................... <= ..
       A[n][0]*x0 + A[n][1]*x1 + ... + A[n][n]*xn <= bn

       x0 >= 0, x1 >= 0, ..., xn >= 0

       obs: se o problema consistir em minimizar P, sujeito a inequacoes do tipo >=,
       basta adicionar a equacao c[0]*x0 + c[1]*x1 + ... + c[n]*xn = 0
       ao final do sistema e calcular a transposta deste. a ultima equacao
       passa a ser a nova restricao do sistema (c[i]) e tem-se um novo sistema
       com A[i] e b[i] diferentes do anterior. calcula-se o simplex deste novo sistema
       e o valor maximo do sistema equivalente (transposta) consiste no valor minimo
       do sistema original.

    Retorna:

        um vetor {x0, x1, ..., xn} com os valores que maximizam o valor
        da equacao P.

    Problemas:

        UVA 802
        UVA 10498
*/

typedef long double T;
typedef vector <T> VT;
typedef vector <int> VI;
const double EPS = 1e-9;
vector <VT> A; //matriz A do sistema
VT b, c; //vetores b e c do sistema, respectivamente
VT res;
VI kt, N;
int m,n;

void initialize()
{

}

inline void pivot (int k, int l, int e)
{
    int x = kt[l];
    T p = A[l][e];
```

```
    for (int i = 0; i < k; ++i)
        A[l][i] /= p;

    b[l] /= p;
    N[e] = 0;

    for (int i = 0; i < m; ++i)
        if (i != l)
            b[i] -= A[i][e] * b[l], A[i][x] = A[i][e] * -A[l][x];

    for (int j = 0; j < k; ++j)
    {
        if (N[j])
        {
            c[j] -= c[e] * A[l][j];
            for (int i = 0; i < m; ++i)
                if (i != l)
                    A[i][j] -= A[i][e] * A[l][j];
        }
    }

    kt[l] = e;
    N[x] = 1;
    c[x] = c[e] * -A[l][x];
}

VT doit (int k)
{
    VT res;
    T best;

    while (1)
    {
        int e = -1, l = -1;

        for (int i = 0; i < k; ++i)
        {
            if (N[i] && c[i] > EPS)
            {
                e = i;
                break;
            }
        }

        if (e == -1)
            break;

        for (int i = 0; i < m; ++i)
            if (A[i][e]>EPS && (l == -1 || best > b[i] / A[i][e]))
                best = b[l = i] / A[i][e];

        if (l == -1) /*ilimitado*/
            return VT();

        pivot(k, l, e);
    }
```

```
    res.resize(k, 0);

    for (int i = 0; i < m; ++i)
        res[kt[i]] = b[i];

    return res;
}

VT simplex (vector<VT> &AA, VT &bb, VT &cc)
{
    int n = AA[0].size(), k;
    m = AA.size();
    k = n + m + 1;
    kt.resize(m);
    b = bb;
    c = cc;
    c.resize(n + m);
    A = AA;

    for (int i = 0; i < m; ++i)
    {
        A[i].resize(k);
        A[i][n + i] = 1;
        A[i][k - 1] = -1;
        kt[i] = n + i;
    }

    N = VI(k, 1);

    for (int i = 0; i < m; ++i)
        N[kt[i]] = 0;

    int pos = min_element(b.begin(), b.end()) - b.begin();

    if (b[pos] < -EPS)
    {
        c = VT(k,0);
        c[k - 1] = -1;
        pivot(k, pos, k - 1);
        res = doit(k);

        if (res[k - 1] > EPS) /*impossivel*/
            return VT();

        for (int i = 0; i < m; ++i)
            if (kt[i] == k - 1)
                for (int j = 0; i < k - 1; ++j)
                    if (N[j] && (A[i][j] < -EPS || EPS < A[i][j]))
                    {
                        pivot(k, i, j);
                        break;
                    }

        c = cc;
        c.resize(k, 0);

        for (int i = 0; i < m; ++i)
            for (int j = 0; i < k; ++j)
```

```
                if (N[j])
                    c[j] -= c[kt[i]] * A[i][j];
    }

    res = doit(k - 1);

    if (!res.empty())
        res.resize(n);

    return res;
}

/*
int main()
{
    int m = 2; //numero de linhas do sistema
    int n = 2; //numero de colunas do sistema
    vector <VT> A(m, VT(n));
    VT b(m), c(n), res;
    A[0][0] = 1;
    A[0][1] = 1;
    b[0]= 4;
    A[1][0] = 2;
    A[1][1] = 1;
    b[1]= 5;
    c[0] = 3;
    c[1] = 4;

    res = simplex (A, b, c);

    for (int i = 0; i < res.size(); ++i)
        cout << res[i] << " ";

    double maximum = 0;

    for (int i = 0; i < res.size(); ++i)
        maximum += res[i] * c[i];

    cout << endl << maximum << endl;

    return 0;
}*/
```

4.22 FRAÇÕES

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

using namespace std;

/*

    FRACOES

    Aplicacoes:

        Operacoes com Fracoes

    Como chamar a funcao:

        1) O construtor da funcao recebe os parametros:

            a) long long num -> Numerador da funcao
            b) long long den -> Denominador da funcao

    Operadores suportados:

        *= , += , -= , /=
        * , + , - , /
        < , > , <= , >= , == , !=

        << (Para imprimir de forma facilitada)

    Complexidade do algoritmo:

        O(1) para todas as operacoes suportadas

    Problemas resolvidos:

    Adicionado por:

        Jorge Gabriel Siqueira

*/

struct frac
{
    long long num, den;
    frac() : num(0), den(1) { };
    frac(long long num, long long den) { set_val(num, den); }
    frac(long long num) : num(num), den(1) { };

    void set_val(long long _num, long long _den)
    {
        num = _num/__gcd(_num, _den);
        den = _den/__gcd(_num, _den);
        if(den < 0) { num *= -1; den *= -1; }
    }

    void operator*=(frac f) { set_val(num * f.num, den * f.den); }
    void operator+=(frac f) { set_val(num * f.den + f.num * den, den *
```

```
f.den); }
    void operator-=(frac f) { set_val(num * f.den - f.num * den, den *
f.den); }
    void operator/=(frac f) { set_val(num * f.den, den * f.num); }
};

bool operator < (frac a, frac b)
{
    if((a.den < 0) ^ (b.den < 0)) return a.num * b.den > b.num * a.den;
    return a.num * b.den < b.num * a.den;
}

std::ostream& operator<<(std::ostream& o, const frac f)
{
    o << f.num << "/" << f.den;
    return o;
}

bool operator == (frac a, frac b) { return a.num * b.den == b.num * a.den;
}
bool operator != (frac a, frac b) { return !(a == b); }
bool operator <= (frac a, frac b) { return (a == b) || (a < b); }
bool operator >= (frac a, frac b) { return !(a < b); }
bool operator > (frac a, frac b) { return !(a <= b); }
frac operator / (frac a, frac b) { frac ret = a; ret /= b; return ret; }
frac operator * (frac a, frac b) { frac ret = a; ret *= b; return ret; }
frac operator + (frac a, frac b) { frac ret = a; ret += b; return ret; }
frac operator - (frac a, frac b) { frac ret = a; }
```

4.23 POLINÔMIOS

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <complex>
#include <iostream>
#include <vector>

using namespace std;

/*

    POLINOMIOS

    Aplicacoes:

        Encontrar as raizes de um polinomio

    Como chamar a funcao:

        1) Criar um polinomio:

            poly p(n+1); //n eh o grau do polinomio

        2) Setar os valores de a,b,c,... tais que:

            a*x^0 + b*x^1 + c*x^2 + ... + z*x^n = 0

            Para setar:

                p.p[0] = a;
                p.p[1] = b;
                p.p[2] = c;
                .
                .
                .
                p.p[n] = z;

        3) Para achar as raizes da equacao chame p.roots()

    Resultado da funcao:

        A funcao retorna um vector<cdouble> roots
        contendo as raizes da equacao.

        roots[i].real() -> parte real da raiz
        roots[i].imag() -> parte imaginaria da raiz (Nao funciona!)

    Complexidade do algoritmo:

        O(?)

    Problemas resolvidos:

    Adicionado por:

        Jorge Gabriel Siqueira
```

```
*/

typedef complex<double> cdouble;

const double EPS = 1e-10;

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

int cmp(cdouble x, cdouble y = 0, double tol = EPS)
{
    return cmp(abs(x), abs(y), tol);
}

const int TAM = 200;

struct poly
{
    cdouble p[TAM];
    int n;
    poly(int n = 0): n(n)
    {
        memset(p, 0, sizeof(p));
    }
    cdouble& operator [](int i)
    {
        return p[i];
    }
    poly operator ~()
    {
        poly r(n-1);
        for (int i = 1; i <= n; i++)
            r[i-1] = p[i] * cdouble(i);
        return r;
    }
    pair<poly, cdouble> ruffini(cdouble z)
    {
        if (n == 0) return make_pair(poly(), 0);
        poly r(n-1);
        for (int i = n; i > 0; i--) r[i-1] = r[i] * z + p[i];
        return make_pair(r, r[0] * z + p[0]);
    }
    cdouble operator ()(cdouble z)
    {
        return ruffini(z).second;
    }
    cdouble find_one_root(cdouble x)
    {
        poly p0 = *this, p1 = ~p0, p2 = ~p1;
        int m = 1000;
        while (m--)
        {
            cdouble y0 = p0(x);
            if (cmp(y0) == 0) break;
            cdouble G = p1(x) / y0;
            cdouble H = G * G - p2(x) - y0;
```

```
            cdouble R = sqrt(cdouble(n-1) * (H * cdouble(n) - G * G));
            cdouble D1 = G + R, D2 = G - R;
            cdouble a = cdouble(n) / (cmp(D1, D2) > 0 ? D1 : D2);
            x -= a;
            if (cmp(a) == 0) break;
        }
        return x;
    }
    vector<cdouble> roots()
    {
        poly q = *this;
        vector<cdouble> r;
        while (q.n > 1)
        {
            cdouble z(rand() / double(RAND_MAX), rand() / double(RAND_MAX));
            z = q.find_one_root(z);
            z = find_one_root(z);
            q = q.ruffini(z).first;
            r.push_back(z);
        }
        return r;
    }
};
```

4.24 BIG NUMBER

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sstream>
#include <iostream>

using namespace std;

/*

    BIG NUMBERS

    Aplicacoes:

        Numero inteiros gigantes de precisao arbitraria

    Como chamar a funcao:

        1) O construtor recebe um inteiro ou uma string
           que represente o numero desejado.

    Operadores suportados:

        *= , += , -= , /=
        * , + , - , / , %
        < , > , <= , >= , == , !=

        << (Para imprimir de forma facilitada)

    Complexidade do algoritmo:

        O(?)

    Problemas resolvidos:

        KRAKOVIA (SPOJ BR)

    Adicionado por:

        Jorge Gabriel Siqueira

*/

const int DIG = 4;
const int BASE = 10000; // BASE**3 < 2**51
const int TAM = 2048;
const double EPS = 1e-10;

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

struct bigint
{
    int v[TAM], n;
```

```
    bigint (int x = 0): n(1)
    {
        memset(v, 0, sizeof(v));
        v[n++] = x;
        fix();
    }

    bigint (char *s): n(1)
    {
        memset(v, 0, sizeof(v));
        int sign = 1;

        while (*s && !isdigit(*s))
            if (*s++ == '-')
                sign *= -1;

        char *t = strdup(s), *p = t + strlen(t);

        while (p > t)
        {
            *p = 0;
            p = max(t, p - DIG);
            sscanf(p, "%d", &v[n]);
            v[n++] *= sign;
        }

        free(t);
        fix();
    }

    bigint& fix (int m = 0)
    {
        n = max(m, n);
        int sign = 0;

        for (int i = 1, e = 0; i <= n || e && (n = i); ++i)
        {
            v[i] += e;
            e = v[i] / BASE;
            v[i] %= BASE;

            if (v[i])
                sign = (v[i] > 0) ? 1 : -1;
        }

        for (int i = n - 1; i > 0; i--)
            if (v[i] * sign < 0)
            {
                v[i] += sign * BASE;
                v[i + 1] -= sign;
            }

        while (n && !v[n])
            --n;

        return *this;
    }
```

```
    int cmp (const bigint& x = 0) const
    {
        int i = max(n, x.n), t = 0;

        while (1)
            if ((t = ::cmp(v[i], x.v[i])) || i-- == 0)
                return t;
    }

    bool operator < (const bigint& x) const
    {
        return cmp(x) < 0;
    }

    bool operator == (const bigint& x) const
    {
        return cmp(x) == 0;
    }

    bool operator != (const bigint& x) const
    {
        return cmp(x) != 0;
    }

    operator string () const
    {
        ostringstream s;
        s << v[n];

        for (int i = n - 1; i > 0; --i)
        {
            s.width(DIG);
            s.fill('0');
            s << abs(v[i]);
        }

        return s.str();
    }

    friend ostream& operator << (ostream& o, const bigint& x)
    {
        return o << (string) x;
    }

    bigint& operator += (const bigint& x)
    {
        for (int i = 1; i <= x.n; ++i)
            v[i] += x.v[i];

        return fix(x.n);
    }

    bigint operator + (const bigint& x)
    {
        return bigint(*this) += x;
    }

    bigint& operator -= (const bigint& x)
```

```
    {
        for (int i = 1; i <= x.n; ++i)
            v[i] -= x.v[i];
        return fix(x.n);
    }

    bigint operator - (const bigint& x)
    {
        return bigint(*this) -= x;
    }

    bigint operator - ()
    {
        bigint r = 0;
        return r -= *this;
    }

    void ams (const bigint& x, int m, int b)    // *this += (x * m) << b;
    {
        for (int i = 1, e = 0; (i <= x.n || e) && (n = i + b); ++i)
        {
            v[i + b] += x.v[i] * m + e;
            e = v[i + b] / BASE;
            v[i + b] %= BASE;
        }
    }

    bigint operator * (const bigint& x) const
    {
        bigint r;
        for (int i = 1; i <= n; ++i)
            r.ams(x, v[i], i - 1);
        return r;
    }

    bigint& operator *= (const bigint& x)
    {
        return *this = *this * x;
    }

    // cmp(x / y) == cmp(x) * cmp(y); cmp(x % y) == cmp(x);
    bigint div (const bigint& x)
    {
        if (x == 0)
            return 0;

        bigint q;
        q.n = max(n - x.n + 1, 0);
        int d = x.v[x.n] * BASE + x.v[x.n - 1];

        for (int i = q.n; i > 0; --i)
        {
            int j = x.n + i - 1;
            q.v[i] = int((v[j] * double(BASE) + v[j - 1]) / d);
            ams(x, -q.v[i], i - 1);

            if (i == 1 || j == 1)
                break;
```

```
            v[j - 1] += BASE * v[j];
            v[j] = 0;
        }

        fix(x.n);
        return q.fix();
    }

    bigint& operator /= (const bigint& x)
    {
        return *this = div(x);
    }

    bigint& operator %= (const bigint& x)
    {
        div(x);
        return *this;
    }

    bigint operator / (const bigint& x)
    {
        return bigint(*this).div(x);
    }

    bigint operator % (const bigint& x)
    {
        return bigint(*this) %= x;
    }

    bigint pow (int x)
    {
        if (x < 0)
            return (*this == 1 || *this == -1) ? pow(-x) : 0;

        bigint r = 1;

        for (int i = 0; i < x; ++i)
            r *= *this;

        return r;
    }

    bigint root (int x)
    {
        if (cmp() == 0 || cmp() < 0 && x % 2 == 0)
            return 0;

        if (*this == 1 || x == 1)
            return *this;

        if (cmp() < 0)
            return -(-*this).root(x);

        bigint a = 1, d = *this;

        while (d != 1)
        {
```

```
            bigint b = a + (d /= 2);

            if (cmp(b.pow(x)) >= 0)
            {
                d += 1;
                a = b;
            }
        }

        return a;
    }
};


int main()
{
    bigint a;
    char str[100];
    cin >> str;
    a = bigint(str);
    cout << a;
    return 0;
}
```

4.25 SOMA EM QUALQUER BASE

```
#include <stdio.h>
#include <string.h>
#include <iostream>
#include <algorithm>

using namespace std;

/*
    Aplicacoes:

        Dados dois numeros em uma certa base, retorna a soma
        destes numeros.

    Parametros

            a -> numero a ser somado armazenado em forma de string
            b -> numero a ser somado armazenado em forma de string
            base -> a base que os numeros a e b estao representados

    Obs. Para bases acima de 10, onde existem letras no numero,
    armazenar as letras na forma maiuscula.

    O range R de bases suportado e: 2 <= R <= 36

*/

int getVal(char n)
{
    if(n >= '0' && n <= '9') return n-'0';
    else return n-'A'+10;
}

char getSimbol(int n)
{
    if(n >= 0 && n <= 9) return n+'0';
    else return (n-10)+'A';
}

string soma(string a, string b, int base)
{
    int posA = a.size()-1;
    int posB = b.size()-1;
    int localSum;
    int vaiUm = 0;
    string resposta = "";

    while(posA >= 0 || posB >= 0)
    {
        localSum = vaiUm;
        vaiUm = 0;

        if(posA >= 0) localSum += getVal(a[posA--]);
        if(posB >= 0) localSum += getVal(b[posB--]);

        if(localSum >= base)
        {
            vaiUm = 1;
```

```
            localSum -= base;
        }
        resposta.push_back(getSimbol(localSum));
    }

    if(vaiUm == 1) resposta.push_back('1');

    reverse(resposta.begin(), resposta.end());
    return resposta;
}
```

4.26 SOMA DOS DIVISORES DE UM NUMERO

```c
#include <stdio.h>
#include <math.h>

/*

    SOMA DOS DIVISORES DE UM NUMERO

    Aplicacoes:

        Determinar a soma de todos os divisores de um numero

        Exemplo:

            somaDiv(36) -> 1 + 2 + 3 + 4 + 6 + 9 + 12 + 18 = 55

    Como chamar a funcao:

        1) Chame a funcao somaDiv(n)

    Resultado da funcao:

        A funcao somaDiv(n) retorna a soma dos divisores de n

    Complexidade do algoritmo:

        O(sqrt(n))

    Problemas resolvidos:

    Adicionado por:

        Jorge Gabriel Siqueira

*/
int somaDiv(int n)
{
    int i;
    int sum = 1;

    double limite = sqrt(n);

    if(n==1) return 0;

    for(i=2;i<limite;i++)
    {
        if(n%i==0)
        {
            sum+=i;
            sum+=n/i;
        }
    }

    if(i*i==n) sum += i;

    return sum;
}
```

4.27 ARITMÉTICA MODULAR

```
#include <stdio.h>
#include <stdlib.h>
#define NEGPOW(e) ((e) % 2 ? -1 : 1)

/*

    ARITMETICA MODULAR

    Aplicacoes:

        Resolucao de operacoes modulares

    Operacoes suportadas:

        invMod(a,m)

            Devolve o inverso modular (a^(-1))%n do numero a

        powMod(x,k,m)

            Devolve a exponenciacao modular (x^k)%m

        sqrtMod(n,p)

            Devolve a raiz modular (sqrt(n))%p do numero n

        addMod(a,b,m)

            Devolve a soma modular (a+b)%m

        subMod(a,b,m)

            Devolve a subtracao modular (a+b)%m

        mulMod(a,b,m)

            Devolve a multiplicacao modular (a+b)%m

        divMod(a,b,m)

            Devolve a divisao modular (a+b)%m

    Complexidade do algoritmo:

        O(?)

    Problemas resolvidos:

    Adicionado por:

        Jorge Gabriel Siqueira

*/

long long extgcd(long long a, long long b, long long &x, long long &y)
{
    long long g = a;
```

```
    x = 1;
    y = 0;
    if(b!=0) g = extgcd(b,a%b,y,x), y -= (a/b)*x;
    return g;
}

long long jacobi(long long a, long long m)
{
    if(a==0) return m == 1 ? 1 : 0;
    if(a%2) return NEGPOW((a-1)*(m-1)/4)*jacobi(m%a, a);
    else return NEGPOW((m*m-1)/8)*jacobi(a/2, m);
}

long long invMod(long long a, long long m)
{
    long long x, y;
    if (extgcd(a, m, x, y) == 1) return (x + m) % m;
    else return 0; //Nao Resolvivel
}

long long powMod(long long x, long long k, long long m)
{
    if(k==0) return 1;
    if(k%2==0) return powMod(x*x%m,k/2,m);
    else return x*powMod(x,k-1,m)%m;
}

long long sqrtMod(long long n, long long p)
{
    long long S, Q, W, i, m = invMod(n, p);

    for(Q=p-1,S=0; Q%2==0; Q/=2,++S);

    do
    {
        W=rand()%p;
    }
    while(W== 0||jacobi(W,p)!=-1);

    for(long long R = powMod(n,(Q+1)/2,p), V=powMod(W,Q,p);;)
    {
        long long z=R*R*m%p;
        for(i=0; i<S&&z%p!=1; z*=z,++i);
        if(i==0) return R;
        R=(R*powMod(V,1<<(S-i-1),p))%p;
    }
}

long long addMod(long long x, long long y, long long n)
{
    return ((x % n) + (y % n)) % n;
}

long long subMod(long long x, long long y, long long n)
{
    return ((x % n) - (y % n)) % n;
}
```

```
long long mulMod(long long x, long long y, long long n)
{
    return ((x % n)*(y % n)) % n;
}

long long divMod(long long x, long long y, long long n)
{
    return mulMod(x,invMod(y,n),n);
}
```

4.28 LUCAS THEOREM

```
#include <stdio.h>
#include <vector>

using namespace std;

/*

    LUCAS THEOREM

    Aplicacoes:

        Expressa o resto da divisao do binomial (m , n)
        por um numero primo p

    Como chamar a funcao:

        1) Chamar lucas_theorem(n,m,p), onde:

            m -> numero m emforma de string
            n -> numero n emforma de string
            p -> numero p primo

    Resultado da funcao:

        A funcao retorna o resto da divisao do
        binomial (m,n) por p.

    Complexidade do algoritmo:

        O(?)

    Problemas resolvidos:

    Adicionado por:

        Jorge Gabriel Siqueira

*/

long long binomial(int n, int m)
{
    if (n > m || n < 0) return 0;
    long long ans = 1, ans2 = 1;
    for (int i = 0 ; i < m ; i++)
    {
        ans *= n - i;
        ans2 *= i + 1;
    }
    return ans / ans2;
}

int lucas_theorem(const char *n, const char *m, int p)
{
    vector<int> np, mp;
    int i;
    for (i = 0 ; n[i] ; i++)
    {
```

```
        if (n[i] == '0' && np.empty()) continue;
        np.push_back(n[i] - '0');
    }
    for (i = 0 ; m[i] ; i++)
    {
        if (m[i] == '0' && mp.empty()) continue;
        mp.push_back(m[i] - '0');
    }
    int ret = 1;
    int ni = 0, mi = 0;
    while (ni < np.size() || mi < mp.size())
    {
        int nmod = 0, mmod = 0;
        for (i = ni ; i < np.size() ; i++)
        {
            if (i + 1 < np.size())
                np[i + 1] += (np[i] % p) * 10;
            else
                nmod = np[i] % p;
            np[i] /= p;
        }
        for (i = mi ; i < mp.size() ; i++)
        {
            if (i + 1 < mp.size())
                mp[i + 1] += (mp[i] % p) * 10;
            else
                mmod = mp[i] % p;
            mp[i] /= p;
        }
        while (ni < np.size() && np[ni] == 0) ni++;
        while (mi < mp.size() && mp[mi] == 0) mi++;
        ret = (ret * binomial(nmod, mmod)) % p;
    }
    return ret;
}
```

4.29 MATRIZ INVERSA

```
#include <stdio.h>
#include <vector>
#include <iostream>
#include <math.h>

using namespace std;

/*

    MATRIZ INVERSA

    Aplicacoes:

        Calcula a inversa de uma matriz.

        Dizemos que uma matriz tera uma matriz inversa
        se for quadrada e se o produto das duas matrizes
        for igual a uma matriz identidade quadrada de
        mesma ordem das outras.

        Matriz identidade eh uma matriz da forma:

        1   0   ...   0
        0   1   ...   0
        .   .   .     .
        .   .     .   .
        .   .       . .
        0   0   ...   1

    Como chamar a funcao:

        1) Armazenar em n a ordem da matriz

        2) Chamar a funcao init pra alocar a matriz
           da forma correta

        3) Popular a matriz matrix[][] com os devidos valores

        4) Chamar a funcao mat_inverse()

    Resultado da funcao:

        A funcao mat_inverse() retorna um vector<vector<double> >
        representando a matriz inversa

    Complexidade do algoritmo:

        O(n^3)

    Problemas resolvidos:

    Adicionado por:

        Jorge Gabriel Siqueira

*/
```

```
int n;
vector<vector<double> > matrix;

inline bool eq(double a, double b)
{
    static const double eps = 1e-9;
    return fabs(a - b) < eps;
}

void init()
{
    matrix.clear();
    matrix.resize(n);
    for(int i=0; i<n; i++)
    {
        matrix[i].resize(n);
    }
}

// returns empty vector if fails
vector<vector<double> > mat_inverse()
{
    int i, j, k;
    vector<vector<double> > ret;
    ret.resize(n);
    for (i = 0 ; i < n ; i++)
    {
        ret[i].resize(n);
        for (j = 0 ; j < n ; j++)
            ret[i][j] = 0;
        ret[i][i] = 1;
    }
    for (i = 0 ; i < n ; i++)
    {
        if (eq(matrix[i][i],0))
        {
            for (j = i + 1 ; j < n ; j++)
            {
                if (!eq(matrix[j][i], 0))
                {
                    for (k = 0 ; k < n ; k++)
                    {
                        matrix[i][k] += matrix[j][k];
                        ret[i][k] += ret[j][k];
                    }
                    break;
                }
            }
            if (j == n)
            {
                ret.clear();
                return ret;
            }
        }
        double tmp = matrix[i][i];
        for (k = 0 ; k < n ; k++)
        {
            matrix[i][k] /= tmp;
```

```
            ret[i][k] /= tmp;
        }
        for (j = 0 ; j < n ; j++)
        {
            if (j == i) continue;
            tmp = matrix[j][i];
            for (k = 0 ; k < n ; k++)
            {
                matrix[j][k] -= matrix[i][k] * tmp;
                ret[j][k] -= ret[i][k] * tmp;
            }
        }
    }
    return ret;
}
```

4.30 MATRIZ INVERSA MODULAR

```
#include <stdio.h>
#include <vector>
#include <iostream>
#include <math.h>

using namespace std;

/*

    MATRIZ INVERSA MODULAR

    Aplicacoes:

        Calcula a inversa de uma matriz modulo m.

        Dizemos que uma matriz tera uma matriz inversa
        modular se for quadrada e se o produto das duas
        matrizes modulo m for igual a uma matriz
        identidade quadrada de mesma ordem das outras.

        Matriz identidade eh uma matriz da forma:

        1   0   ...   0
        0   1   ...   0
        .   .   .     .
        .   .     .   .
        .   .       . .
        0   0   ...   1

    Como chamar a funcao:

        1) Armazenar em n a ordem da matriz

        2) Chamar a funcao init pra alocar a matriz
           da forma correta

        3) Popular a matriz matrix[][] com os devidos valores

        4) Chamar a funcao mat_inverse(mod), onde mod eh
           o valor do modulo

    Resultado da funcao:

        A funcao mat_inverse() retorna um vector<vector<double> >
        representando a matriz inversa

    Complexidade do algoritmo:

        O(n^3)

    Problemas resolvidos:

    Adicionado por:

        Jorge Gabriel Siqueira

*/
```

```
vector<vector<long long> > matrix;
int n;

pair<long long, long long> extended_gcd(long long a, long long b)
{
    if (b == 0) return make_pair(1, 0);
    pair<long long, long long> t = extended_gcd(b, a % b);
    return make_pair(t.second, t.first - t.second * (a / b));
}

long long modinverse(long long a, long long m)
{
    return (extended_gcd(a, m).first % m + m) % m;
}

void init()
{
    matrix.clear();
    matrix.resize(n);
    for(int i=0; i<n; i++)
    {
        matrix[i].resize(n);
    }
}

// returns empty vector if fails
vector<vector<long long> > mat_inverse(long long mod)
{
    int i, j, k;
    vector<vector<long long> > ret;
    ret.resize(n);
    for (i = 0 ; i < n ; i++)
    {
        ret[i].resize(n);
        for (j = 0 ; j < n ; j++)
            ret[i][j] = 0;
        ret[i][i] = 1 % mod;
    }
    for (i = 0 ; i < n ; i++)
    {
        if (matrix[i][i] == 0)
        {
            for (j = i + 1 ; j < n ; j++)
            {
                if (matrix[j][i] != 0)
                {
                    for (k = 0 ; k < n ; k++)
                    {
                        matrix[i][k] = (matrix[i][k] + matrix[j][k]) % mod;
                        ret[i][k] = (ret[i][k] + ret[j][k]) % mod;
                    }
                    break;
                }
            }
            if (j == n)
            {
                ret.clear();
```

```
                return ret;
            }
        }
        long long tmp = modinverse(matrix[i][i], mod);
        for (k = 0 ; k < n ; k++)
        {
            matrix[i][k] = (matrix[i][k] * tmp) % mod;
            ret[i][k] = (ret[i][k] * tmp) % mod;
        }
        for (j = 0 ; j < n ; j++)
        {
            if (j == i) continue;
            tmp = matrix[j][i];
            for (k = 0 ; k < n ; k++)
            {
                matrix[j][k] -= matrix[i][k] * tmp;
                matrix[j][k] = (matrix[j][k] % mod + mod) % mod;
                ret[j][k] -= ret[i][k] * tmp;
                ret[j][k] = (ret[j][k] % mod + mod) % mod;
            }
        }
    }
    return ret;
}
```

4.31 DETERMINANTE DE UMA MATRIZ

```
#include <stdio.h>
#include <vector>
#include <iostream>
#include <math.h>

using namespace std;

/*

    DETERMINANTE DE UMA MATRIZ

    Aplicacoes:

        Calcula o determinante de uma matriz quadrada

    Como chamar a funcao:

        1) Chamar a funcao mat_det(matrix, n) onde:

            matrix -> vector< vector<double> > representando
                      a matriz

            n -> Ordem da matriz

    Resultado da funcao:

        A funcao retorna o valor do determinante da matriz
        representada na variavel matrix.

    Complexidade do algoritmo:

        O(n^2)

    Problemas resolvidos:

    Adicionado por:

        Jorge Gabriel Siqueira

*/


inline bool eq(double a, double b)
{
    static const double eps = 1e-9;
    return (a - eps < b && b < a + eps);
}

double mat_det(vector< vector<double> > matrix, int n)
{
    int i, j, k;
    double ret = 1;
    for (i = 0 ; i < n ; i++)
    {
        if (eq(matrix[i][i], 0))
        {
            for (j = i + 1 ; j < n ; j++)
```

```
            {
                if (!eq(matrix[j][i], 0))
                {
                    for (k = 0 ; k < n ; k++)
                        matrix[i][k] += matrix[j][k];
                    break;
                }
            }
            if (j == n)
                return 0;
        }
        double tmp = matrix[i][i];
        for (k = 0 ; k < n ; k++)
            matrix[i][k] /= tmp;
        ret *= tmp;
        for (j = 0 ; j < n ; j++)
        {
            if (j == i) continue;
            tmp = matrix[j][i];
            for (k = 0 ; k < n ; k++)
                matrix[j][k] -= matrix[i][k] * tmp;
        }
    }
    return ret;
}
```

4.32 RAIZES DE UMA EQUAÇÃO PELO METODO DE NEWTON

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

#define epsilon 0.0000001

/*

    RAIZ DE UMA EQUACAO PELO METODO DE NEWTON

    Aplicacoes:

        Determinar a raiz de uma equacao

    Como chamar a funcao:

        1) Definir a funcao a se obter o zero em
           f(double x).

        2) Definir a devirava da funcao a se obter
        o zero em df(double x).

        3) Chamar a funcao newtonMethod(x), onde x
           eh uma estimativa para o zero da equacao

    Resultado da funcao:

        A funcao newtonMethod() retorna um zero para a
        equacao

    Complexidade do algoritmo:

        O(?)

    Problemas resolvidos:

    Adicionado por:

        Jorge Gabriel Siqueira

*/

/* function */
double f(double x)
{
    return(x*x - 3);
}

/* derivative of f */

double df(double x)
{
    return(2*x);
}
```

```
double newtonMethod(double initialX)
{
    double oldx, x;
    x = initialX;
    do
    {
        oldx = x;
        x = x - f(x)/df(x);
        printf("%f\n",x);
    }
    while (fabs(x-oldx) > epsilon);

    return x;
}

int main()
{
    double x;

    printf("Give an initial estimate of zero > ");
    scanf("%lf",&x);
    printf("A zero is equal to: %lf", newtonMethod(x));
}
```

4.33 INTEGRAL TRAPEZOID METHOD

```
#include <stdio.h>

/*

    INTEGRAL DEFINIDA PELO METODO TRAPEZIODAL

    Aplicacoes:

        Encontrar a integral definida de uma funcao

    Como chamar a funcao:

        1) Definir a funcao a se obter o zero em
           f(double x).

        2) Chamar a funcao trapezzoid(a,b,n), onde:

            a -> Inicio do intervalo da funcao f()
            b -> Fim do intervalo da funcao f()
            n -> Numero de iteracoes

    Resultado da funcao:

        A funcao retorna o valor da integral definida
        de f() em [a,b]

    Complexidade do algoritmo:

        O(n)

    Problemas resolvidos:

    Adicionado por:

        Jorge Gabriel Siqueira

*/

double f( double x)
{
    return(x*x);
}

double trapezzoid(double a, double b, int n)
{
    double delta = delta = (b-a)/n;
    double integral = 0.0;
    int i;

    for (i=0; i < n; i=i+1)
        integral = integral + (f(a + i*delta)+f(a+(i+1)*delta))/2*delta;

    return integral;
}

int main()
{
```

```
    printf("The integral of f from %f to %f is equal to
%.18f\n",0.0,1.0,trapezzoid(0.0,1.0,100000));
}
```

4.34 INTEGRAL SIMPSONS METHOD

```
#include <stdio.h>

/*

    INTEGRAL DEFINIDA PELO METODO DE SIMPSON

    Aplicacoes:

        Encontrar a integral definida de uma funcao

    Como chamar a funcao:

        1) Definir a funcao a se obter o zero em
           f(double x).

        2) Chamar a funcao simpsons(a,b,n), onde:

            a -> Inicio do intervalo da funcao f()
            b -> Fim do intervalo da funcao f()
            n -> Numero de iteracoes

    Resultado da funcao:

        A funcao retorna o valor da integral definida
        de f() em [a,b]

    Complexidade do algoritmo:

        O(n)

    Problemas resolvidos:

    Adicionado por:

        Jorge Gabriel Siqueira

*/

double f(double x)
{
    return(x*x) ;
}

double simpsons(double a, double b, int n)
{
    double delta = delta = (b-a)/n;
    double integral = 0.0;
    int i;


    for (i=0; i < n; i=i+1)
        integral = integral +
(f(a+i*delta)+4*f(a+(i+0.5)*delta)+f(a+(i+1)*delta))/6*delta;

    return integral;
}
```

```c
int main()
{
    printf("The integral of f from %f to %f is equal to
%.18f\n",0.0,1.0,simpsons(0.0,1.0,100000));
}
```

4.35 BASIC MATRIX

```
#include <cstdio>
#include <string>
#include <cstring>
#include <vector>
#include <algorithm>
#include <map>
#include <utility>
#include <cmath>
#include <queue>
#include <stack>
#include <set>
#include <deque>
#include <iostream>
#include <math.h>
#include <sstream>
#include <assert.h>
#include <numeric>
#include <limits>
#define INF 0x3f3f3f3f

using namespace std;

struct matriz
{
    int n,m;
    long long **mat;

    matriz(int n_, int m_)
    {
        n = n_;
        m = m_;
        mat = (long long **)calloc(n_, sizeof(long long *));
        for(int i=0; i<n_; i++) mat[i] = (long long *)calloc(m_,
sizeof(long long));
    }

    matriz operator + (matriz MAT)
    {
        if(MAT.n != n || MAT.m != m) return matriz(0,0);
        else
        {
            matriz answer(n,m);
            for(int i=0; i<n; i++)
            {
                for(int j=0; j<m; j++) answer.mat[i][j] =
mat[i][j]+MAT.mat[i][j];
            }
        }
    }

    matriz operator - (matriz MAT)
    {
        if(MAT.n != n || MAT.m != m) return matriz(0,0);
        else
        {
            matriz answer(n,m);
            for(int i=0; i<n; i++)
```

```
            {
                for(int j=0; j<m; j++) answer.mat[i][j] = mat[i][j]-
MAT.mat[i][j];
            }
        }
    }

    matriz operator * (matriz MAT)
    {
        if(m != MAT.n) { printf("Invalid Sizes...\n"); return matriz(0,0);
}
        else
        {
            matriz answer(n,MAT.m);
            for(int i=0; i<n; i++)
            {
                for(int j=0; j<MAT.m; j++)
                {
                    for(int k=0; k<m; k++)
                    {
                        answer.mat[i][j] += mat[i][k]*MAT.mat[k][j];
                    }
                }
            }
            return answer;
        }
    }

    void set(int i, int j, long long val)
    {
        mat[i][j] = val;
    }

    long long get(int i, int j)
    {
        return mat[i][j];
    }
};

std::ostream& operator<<(std::ostream& o, matriz f)
{
    for(int i=0; i<f.n; i++)
    {
        for(int j=0; j<f.m; j++)
        {
            o << f.get(i,j) << " ";
        }
        o << endl;
    }
    return o;
}

int main()
{
    matriz a(2,3);
    matriz b(3,2);

    a.set(0,0,2); a.set(0,1,5); a.set(0,2,9);
```

```
    a.set(1,0,3); a.set(1,1,6); a.set(1,2,8);

    b.set(0,0,2); b.set(0,1,7);
    b.set(1,0,4); b.set(1,1,3);
    b.set(2,0,5); b.set(2,1,2);

    matriz c = a*b;

    cout << c << endl;

    return 0;
}
```

5. STRINGS

5.1 AHO CORASICK

```
#include <stdio.h>
#include <string.h>
#include <map>
#include <list>
#include <queue>
#include <iostream>

using namespace std;

/*

    AHO CORASICK

    Aplicacoes:

        String matching de varios padroes simultaneamente

    Como chamar a funcao:

        1) Chamar a funcao initialize() para inicializar a
           estrutura.

        2) Adicionar os padroes (palavras do dicionario)
           por meio da funcao addPattern(s), onde s eh o
           padrao a ser adicionado.

        3) Chamar a funcao goFails() apos a insercao de todos
           os padroes para montar a arvore de padroes.

        4) Chamar a funcao query(s) para consultar se as
           palavras do dicionario encontram-se no texto s
           passado como parametro.

    Resultado da funcao:

        A resposta da query eh dada pelo vetor de listas
        "list<int> pos[MAX_PAD]".

        Para saber se um padrao foi encontrado no texto utilizar:

            list<int> ocorrencias = aut.pos[aut.pad[pattern]], onde
            pattern eh um dos padroes adicionados ao dicionario

        A lista contera os indices do texto onde o padrao "pattern"
        foi encontrado.

        Se a lista estiver vazia eh porque o padrao nao foi
        encontrado no texto!

    Complexidade do algoritmo:

        O(?)

    Problemas resolvidos:
```

```
        UVA 10679

    Adicionado por:

        Jorge Gabriel Siqueira

*/

#define NULO -1
#define MAX_NO 100010
#define MAX_PAD 10010 //numero maximo de padroes

typedef map <char, int> mapach;
typedef map <string, int> mapastr;

struct automata
{
    mapach trans[MAX_NO];
    mapastr pad;
    list<int> pos[MAX_PAD]; //resposta para o problema!!!
    int falha[MAX_NO], final[MAX_NO], tam[MAX_PAD], numNos;
    automata (): numNos(0) {}

    void initialize ()
    {
        memset(falha, NULO, sizeof(falha));
        memset(final, NULO, sizeof(final));

        for (int i = 0; i < numNos; ++i) trans[i].clear();

        pad.clear();
        numNos = 1;
    }

    int addPattern (const char* s)
    {
        pair <mapach::iterator, bool> pch;
        int i, no = 0, numPad = pad.size();

        if (pad.count(s)) return pad[s];
        else pad.insert(make_pair(s, numPad));

        for (i = 0; s[i]; ++i)
        {
            if ((pch = trans[no].insert(make_pair(s[i], numNos))).second)
++numNos;
            no = pch.first -> second;
        }

        tam[numPad] = i ? i : 1;
        return final[no] = numPad;
    }

    void goFails ()
    {
        queue <int> fila;
        int filho;
```

```
        for (typeof(trans[0].begin()) it = trans[0].begin(); it !=
trans[0].end(); ++it)
        {
            falha[filho = it->second] = 0;
            fila.push(filho);
        }

        while (!fila.empty())
        {
            int atual = fila.front();
            fila.pop();
            for (typeof(trans[atual].begin()) it = trans[atual].begin();
it != trans[atual].end(); ++it)
            {
                char c = it->first;
                filho = it->second;
                int ret = falha[atual];

                while (ret != NULO && !trans[ret].count(c)) ret =
falha[ret];

                if (ret != NULO)
                {
                    falha[filho] = trans[ret][c];
                    if (final[filho]==NULO && final[falha[filho]]!=NULO)
final[filho] = final[falha[filho]];
                }
                else if (trans[0].count(c)) falha[filho] = trans[0][c];

                fila.push(filho);
            }
        }
    }

    void query (const char* s)
    {
        int ret, atual = 0, i = 0;
        int N = pad.size();

        for (int j = 0; j < N; j++) pos[j].clear();

        do
        {
            while(atual != NULO && !trans[atual].count(s[i])) atual =
falha[atual];

            atual = (atual == NULO) ? 0 : trans[atual][s[i]];

            for (ret = atual; ret != NULO && final[ret] != NULO; ret =
falha[ret])
            {
                pos[final[ret]].push_back(i - tam[final[ret]] + 1);
                while (falha[ret]!=NULO && final[falha[ret]]==final[ret])
ret = falha[ret];
            }
        }
        while (s[i++]);
```

```
    }
};
```

5.2 SUFFIX ARAY COM LCP

```
#include <stdio.h>
#include <string.h>
#include <vector>
#include <math.h>
#include <iostream>

using namespace std;

/*
    4.12 Suffix Array com LCP

    Aplicacoes:

        busca de padroes em textos de forma eficiente. seja
        uma string S, o suffix array desta string e um vetor ordenado
        de sufixos da mesma.
        por exemplo: "banana" -> sufixos = {a, na, ana, nana, anana, banana}
        sufixos ordenados = {a, ana, anana, banana, na, nana}
        o suffix array guarda a posicao onde cada sufixo tem inicio. logo:
        suffix array (banana) = {5, 3, 1, 0, 4, 2}
        uma vez que se possui o vetor de sufixos ordenado, pode-se realizar
        queries de substrings em O(mlogn) com busca binaria (query_substr()).
        o LCP (longest common prefix) e um vetor de inteiros que contem
        o tamanho do maior prefixo entre dois sufixos adjacentes do suffix
        array.
        LCP (banana) {1, 3, 0, 0, 2}

    Complexidade:

        queries em O(mlogn)

    Problemas:

        UVA 10679
*/

#define MAXN 100010 //tamanho maximo do texto a ser consultado
bool k_cmp (int a1, int b1, int a2, int b2, int a3 = 0, int b3 = 0)
{
    return a1 != b1 ? a1 < b1 : (a2 != b2 ? a2 < b2 : a3 < b3);
}
int bucket[MAXN + 1], tmp[MAXN];
template <class T> void k_radix (T keys, int *in, int *out,
                                 int off, int n, int k)
{
    memset(bucket, 0, sizeof(int) * (k + 1));
    for(int j = 0; j < n; ++j)
        bucket[keys[in[j]+off]]++;
    for(int j = 0, sum = 0; j <= k; ++j)
        sum += bucket[j], bucket[j] = sum - bucket[j];
    for(int j = 0; j < n; ++j)
        out[bucket[keys[in[j]+off]]++] = in[j];
}
int m0[MAXN/3 + 1];
vector <int> k_rec (const vector <int>& v, int k)
{
    int n = v.size() - 3, sz = (n + 2)/3, sz2 = sz + n/3;
    if (n < 2)
```

```
        return vector <int>(n);
    vector<int> sub(sz2+3);
    for(int i = 1, j = 0; j < sz2; i += i%3, ++j)
        sub[j] = i;
    k_radix(v.begin(), &sub[0], tmp, 2, sz2, k);
    k_radix(v.begin(), tmp, &sub[0], 1, sz2, k);
    k_radix(v.begin(), &sub[0], tmp, 0, sz2, k);
    int last[3] = {-1, -1, -1}, unique = 0;
    for (int i = 0; i < sz2; ++i)
    {
        bool diff = false;
        for(int j = 0; j < 3; last[j] = v[tmp[i]+j], ++j)
            diff |= last[j] != v[tmp[i]+j];
        unique += diff;
        if(tmp[i]%3 == 1)
            sub[tmp[i]/3] = unique;
        else
            sub[tmp[i]/3 + sz] = unique;
    }
    vector <int> rec;
    if (unique < sz2)
    {
        rec = k_rec(sub, unique);
        rec.resize(sz2+sz);
        for(int i = 0; i < sz2; ++i)
            sub[rec[i]] = i + 1;
    }
    else
    {
        rec.resize(sz2+sz);
        for(int i = 0; i < sz2; ++i)
            rec[sub[i]-1] = i;
    }
    for (int i = 0, j = 0; j < sz; ++i)
        if (rec[i] < sz)
            tmp[j++] = 3*rec[i];
    k_radix(v.begin(), tmp, m0, 0, sz, k);
    for (int i = 0; i < sz2; ++i)
        rec[i] = rec[i] < sz ? 3*rec[i] + 1 : 3*(rec[i] - sz) + 2;
    int prec = sz2 - 1, p0 = sz - 1, pret = sz2 + sz - 1;
    while (prec >= 0 && p0 >= 0)
        if (rec[prec]%3 == 1 && k_cmp(v[m0[p0]], v[rec[prec]],
                                      sub[m0[p0]/3], sub[rec[prec]/3 +
sz]) ||
                rec[prec]%3 == 2 && k_cmp(v[m0[p0]], v[rec[prec]],
                                          v[m0[p0] + 1], v[rec[prec] + 1],
                                          sub[m0[p0]/3 + sz],
sub[rec[prec]/3 + 1]))
            rec[pret--] = rec[prec--];
        else
            rec[pret--] = m0[p0--];
    if (p0 >= 0)
        memcpy(&rec[0], m0, sizeof(int)*(p0 + 1));
    if (n%3==1)
        rec.erase(rec.begin());
    return rec;
}
vector <int> suffix_array(const string& s)
```

```
{
    int n = s.size(), cnt = 1;
    vector <int> v(n + 3);
    for (int i = 0; i < n; ++i)
        v[i] = i;
    k_radix(s.begin(), &v[0], tmp, 0, n, 256);
    for (int i = 0; i < n; cnt += (i+1 < n && s[tmp[i + 1]] != s[tmp[i]]),
++i)
        v[tmp[i]] = cnt;
    return k_rec(v, cnt);
}
vector <int> lcp(const string& s, const vector <int>& sa)
{
    int n = sa.size();
    vector <int> prm(n), ans(n - 1);
    for (int i = 0; i < n; ++i)
        prm[sa[i]] = i;
    for (int h = 0, i = 0; i < n; ++i)
        if (prm[i])
        {
            int j = sa[prm[i] - 1], ij = max(i, j);
            while (ij + h < n && s[i+h] == s[j+h])
                ++h;
            ans[prm[i] - 1] = h;
            if(h)
                --h;
        }
    return ans;
}
bool query_substr (const string& in, const vector <int>& sa,
                   const string& p)
{
    int l = 0, r = in.size() - 1;
    while (l < r)
    {
        int mid = (l + r)/2;
        if (p > in.substr (sa[mid]))
            l = mid + 1;
        else
            r = mid;
    }
    for (int i = 0, j = sa[l]; i < p.size(); ++i, ++j)
        if (j >= in.size() || p[i] != in[j])
            return false;
    return true;
}
```

5.3 KNUTH MORRIS PRATT

```
#include <cstdio>
#include <string>
#include <cstring>
#include <vector>
#include <algorithm>
#include <map>
#include <utility>
#include <cmath>
#include <iostream>
#include <sstream>

using namespace std;

/*

    KNUTH MORRIS PRATT

    Aplicacoes:

        Busca de um padrao em um texto de forma eficiente.
        obs: a funcao strstr (char* text, char* pattern) da biblioteca
        <cstring> implementa KMP (C-ANSI). a funcao retorna a primeira
        ocorrencia do padrao no texto, KMP retorna todas!

    Como chamar a funcao:

        1) Chamar a funcao kmp(text, pattern)

    Resultado da funcao:

        A funcao armazena em:

            nres -> O numero de ocorrencias do padrao no texto
            res[] -> posicoes das nres ocorrencias do padrao no texto

    Complexidade do algoritmo:

        O(n)

    Problemas resolvidos:

        QUADRADO (SPOJ BR)
        PLAGIO (SPOJ BR)

    Adicionado por:

        Jorge Gabriel Siqueira

*/

#define MAXN 100001 //tamanho maximo da string
int pi[MAXN], res[MAXN], nres;
void kmp (string text, string pattern)
{
    nres = 0;
    pi[0] = -1;
```

```
    for (int i = 1; i < pattern.size(); ++i)
    {
        pi[i] = pi[i-1];
        while (pi[i] >= 0 && pattern[pi[i] + 1] != pattern[i])
            pi[i] = pi[pi[i]];
        if (pattern[pi[i] + 1] == pattern[i])
            ++pi[i];
    }
    int k = -1;  //k + 1 eh o tamanho do match atual
    for (int i = 0; i < text.size(); ++i)
    {
        while (k >= 0 && pattern[k + 1] != text[i])
            k = pi[k];
        if (pattern[k + 1] == text[i])
            ++k;
        if (k + 1 == pattern.size())
        {
            res[nres++] = i - k;
            k = pi[k];
        }
    }
}
```

5.4 NÚMERO DE SUBSEQUENCIAS PALINDROMES

```
#include <cstdio>
#include <string>
#include <cstring>
#include <vector>
#include <algorithm>
#include <map>
#include <utility>
#include <cmath>
#include <queue>
#include <stack>
#include <set>
#include <deque>
#include <iostream>
#include <sstream>

using namespace std;

/*

    NUMERO DE SUBSEQUENCIAS PALINDROMES

    Aplicacoes:

        Determinar o numero de subsequencias de uma string
        que sao palindromos.

    Como chamar a funcao:

        1) Armazenar a string em string S

        2) Chamar a funcao ps()

    Resultado da funcao:

        A funcao ps() retorna o numero de subsequencias de
        S que sao palindromes

    Complexidade do algoritmo:

        O(?)

    Problemas resolvidos:

        CB03 (CodeChef)

    Adicionado por:

        Jorge Gabriel Siqueira

*/

int n;
string S;
vector <vector<long long> > best;
long long solve (int b, int e)
{
```

```
    long long &res = best[b][e];
    if (res >= 0)
        return res;
    if (b == e)
        return res = 1;
    if (b + 1 == e)
        return res = 2;
    res = solve (b, e - 1) + solve (b + 1, e);
    if (S[b] != S[e - 1])
        res -= solve (b + 1, e - 1);
    return res;
}
long long ps ()
{
    best.clear();
    n = S.size();
    best.resize(n + 1, vector <long long> (n + 1, -1));
    return (solve (0, n) - 1);
}
```

5.5 MAIOR SUBSEQUÊNCIA PALINDROME

```
#include <cstdio>
#include <string>
#include <cstring>
#include <vector>
#include <algorithm>
#include <map>
#include <utility>
#include <cmath>
#include <queue>
#include <stack>
#include <set>
#include <deque>
#include <iostream>
#include <sstream>

using namespace std;

/*

    MAIOR SUBSEQUENCIA PALINDROME

    Aplicacoes:

        Determinar a maior subsequencia de uma string que
        e um palindromo.

    Como chamar a funcao:

        1) Armazenar a string a ser processada em text[]

        2) Chamar a funcao longestPalindromeSubsequence()

    Resultado da funcao:

        A funcao retorna o tamanho da maior subsequencia
        palindrome de text[]

    Complexidade do algoritmo:

        O(n^2)

    Problemas resolvidos:

        UVA 11151

    Adicionado por:

        Jorge Gabriel Siqueira

*/

#include <stdio.h>
#include <string.h>

char text[1024];
int dp[1024][1024];
```

```
int z(int l, int r)
{
    if(l > r) return dp[l][r] = 0;
    if(l == r) return dp[l][r] = 1;
    if(dp[l][r] != -1) return dp[l][r];
    if(text[l] == text[r]) return dp[l][r] = z(l + 1, r - 1) + 2;
    return dp[l][r] = max(z(l + 1, r), z(l, r - 1));
}

int longestPalindromeSubsequence()
{
    int len = strlen(text);
    for(int i=0; i<len; i++) memset(dp[i],-1,len*sizeof(int));
    return z(0,strlen(text)-1);
}
```

5.6 MANACHER'S – MAIOR SUBSTRING PALINDROME

```
#include <cstdio>
#include <string>
#include <cstring>
#include <vector>
#include <algorithm>
#include <map>
#include <utility>
#include <cmath>
#include <queue>
#include <stack>
#include <set>
#include <deque>
#include <iostream>
#include <sstream>

using namespace std;

/*

    MANACHER MAIOR SUBSTRING PALINDROME

    Aplicacoes:

        Determinar a maior substring palindromo de uma
        string.

        Exemplo:

            banana -> anana

    Como chamar a funcao:

        1) Chamar a funcao longestPalindromeSubstring(s), onde s
        eh a string onde sera feito a busca.

    Resultado da funcao:

        A funcao retorna a string que representa a maior
        subsequencia palindrome de s.

    Complexidade do algoritmo:

         O(n)

    Problemas resolvidos:

        EPALIN (SPOJ)

    Adicionado por:

        Jorge Gabriel Siqueira

*/

// Transform S into T.
// For example, S = "abba", T = "^#a#b#b#a#$".
```

```
// ^ and $ signs are sentinels appended to each end to avoid bounds
checking
string preProcess(string s)
{
    int n = s.length();
    if (n == 0) return "^$";
    string ret = "^";
    for (int i = 0; i < n; i++)
        ret += "#" + s.substr(i, 1);

    ret += "#$";
    return ret;
}

string longestPalindromeSubstring(string s)
{
    string T = preProcess(s);
    int n = T.length();
    int *P = new int[n];
    int C = 0, R = 0;
    for (int i = 1; i < n-1; i++)
    {
        int i_mirror = 2*C-i; // equals to i' = C - (i-C)

        P[i] = (R > i) ? min(R-i, P[i_mirror]) : 0;

        // Attempt to expand palindrome centered at i
        while (T[i + 1 + P[i]] == T[i - 1 - P[i]])
            P[i]++;

        // If palindrome centered at i expand past R,
        // adjust center based on expanded palindrome.
        if (i + P[i] > R)
        {
            C = i;
            R = i + P[i];
        }
    }

    // Find the maximum element in P.
    int maxLen = 0;
    int centerIndex = 0;
    for (int i = 1; i < n-1; i++)
    {
        if (P[i] > maxLen)
        {
            maxLen = P[i];
            centerIndex = i;
        }
    }
    delete[] P;

    return s.substr((centerIndex - 1 - maxLen)/2, maxLen);
}
```

5.7 DISTANCIA MÍNIMA DE ROTAÇÃO

```
#include <cstdio>
#include <string>
#include <cstring>
#include <vector>
#include <algorithm>
#include <map>
#include <utility>
#include <cmath>
#include <queue>
#include <stack>
#include <set>
#include <deque>
#include <iostream>
#include <sstream>

using namespace std;

/*

    DISTANCIA MINIMA DE ROTACAO

    Aplicacoes:

        Determinar o numero minimo de vezes que
        deve-se rotacionar uma string S (ao rotacionar,
        S torna-se T = S[2..n] + S[1]) de forma a se
        obter a menor construcao lexografica possivel.

        Exemplo: Dada a string "bbaaccaadd", a menor
        string resultante da rotacao desta eh "aaccaaddbb".

    Como chamar a funcao:

        1) Chamar a funcao minimumExpression(s), onde s
           eh um char[] contendo a string a ser analisada

    Resultado da funcao:

        A funcao retorna o numero minimo de rotacoes para
        tornar s a menor construcao lexografica possivel

    Complexidade do algoritmo:

        O(n)

    Problemas resolvidos:

        MINMOVE (SPOJ)

    Adicionado por:

        Jorge Gabriel Siqueira

*/
```

```
int minimumExpression(char s[])
{
    int i, j, k, n, len, p, q;
    len = n = strlen(s), n <<= 1, i = 0, j = 1, k = 0;
    while (i + k < n && j + k < n)
    {
        p = i + k >= len ? s[i + k - len] : s[i + k];
        q = j + k >= len ? s[j + k - len] : s[j + k];
        if (p == q)
            ++k;
        else if (p > q)
        {
            i = i + k + 1;
            if (i <= j)
                i = j + 1;
            k = 0;
        }
        else if (p < q)
        {
            j = j + k + 1;
            if (j <= i)
                j = i + 1;
            k = 0;
        }
    }
    return i < j ? i : j;
}
```

5.8 SUBSTITUIR TODAS AS OCORÊNCIAS

```
#include <cstdio>
#include <string>
#include <cstring>
#include <vector>
#include <algorithm>
#include <map>
#include <utility>
#include <cmath>
#include <queue>
#include <stack>
#include <set>
#include <deque>
#include <iostream>
#include <sstream>

#define REP(i,n) for(int i=0;i<(int)n;++i)
#define EACH(i,c) for(__typeof((c).begin()) i=(c).begin(); i!=(c).end();
++i)
#define ALL(c) (c).begin(), (c).end()
#define SIZE(x) (int((x).size()))

#define MAXSZ 100

using namespace std;

/*

    SUBSTITUIR EM TODAS OCORRENCIAS

    Aplicacoes:

        Substitui todas as ocorrencias de um padrao em
        uma string.

    Como chamar a funcao:

        1) Chamar a funcao replaceAll(s,f,t), onde:

            string s -> string original
            string f -> string a ser buscada em s
            string t -> string que sera substituida
                        no lugar de f

    Resultado da funcao:

        A funcao retorna uma string com todas
        ocorrencias de f em s substituidas por t

    Complexidade do algoritmo:

        O(?)

    Problemas resolvidos:

    Adicionado por:
```

```
        Jorge Gabriel Siqueira

*/

const int INF = 0x3F3F3F3F;
const double PI = 2*acos(0);
const double EPS = 1e-10;

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

string replaceAll(string s, string f, string t)
{
    string r;

    for (int p = 0; (p = s.find(f)) != s.npos; )
    {
        r += s.substr(0, p) + t;
        s = s.substr(p + f.size());
    }

    return r + s;
}
```

5.9 SUBSTITUIR NA PRIMEIRA OCORRENCIA

```
#include <cstdio>
#include <string>
#include <cstring>
#include <vector>
#include <algorithm>
#include <map>
#include <utility>
#include <cmath>
#include <queue>
#include <stack>
#include <set>
#include <deque>
#include <iostream>
#include <sstream>

#define REP(i,n) for(int i=0;i<(int)n;++i)
#define EACH(i,c) for(__typeof((c).begin()) i=(c).begin(); i!=(c).end();
++i)
#define ALL(c) (c).begin(), (c).end()
#define SIZE(x) (int((x).size()))

#define MAXSZ 100

using namespace std;

/*

    SUBSTITUIR NA PRIMEIRA OCORRENCIA

    Aplicacoes:

        Substitui a primeira ocorrencia de um padrao em
        uma string.

    Como chamar a funcao:

        1) Chamar a funcao replace(s,f,t), onde:

            string s -> string original
            string f -> string a ser buscada em s
            string t -> string que sera substituida
                        no lugar de f

    Resultado da funcao:

        A funcao retorna uma string com a primeira
        ocorrencias de f em s substituidas por t

    Complexidade do algoritmo:

        O(?)

    Problemas resolvidos:

    Adicionado por:
```

```
        Jorge Gabriel Siqueira

*/

const int INF = 0x3F3F3F3F;
const double PI = 2*acos(0);
const double EPS = 1e-10;

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

string replace(string s, string f, string t)
{
    string r;
    int p = s.find(f);

    if (p != s.npos)
    {
        r += s.substr(0, p) + t;
        s = s.substr(p + f.size());
    }

    return r + s;
}
```

5.10 DIVIDIR EM TODAS AS OCORRENCIAS

```
#include <cstdio>
#include <string>
#include <cstring>
#include <vector>
#include <algorithm>
#include <map>
#include <utility>
#include <cmath>
#include <queue>
#include <stack>
#include <set>
#include <deque>
#include <iostream>
#include <sstream>

#define REP(i,n) for(int i=0;i<(int)n;++i)
#define EACH(i,c) for(__typeof((c).begin()) i=(c).begin(); i!=(c).end();
++i)
#define ALL(c) (c).begin(), (c).end()
#define SIZE(x) (int((x).size()))

#define MAXSZ 100

using namespace std;

/*

    DIVIDIR EM TODAS OCORRENCIAS

    Aplicacoes:

        Divide uma string em todas as ocorrencias
        de um padrao

    Como chamar a funcao:

        1) Chamar a funcao splitAll(s,t), onde:

            string s -> string original
            string t -> string a ser buscada em s

    Resultado da funcao:

        A funcao retorna um vector<string>
        contendo a string s quebrada em todas
        as ocorencias de t.

    Complexidade do algoritmo:

        O(?)

    Problemas resolvidos:

    Adicionado por:

        Jorge Gabriel Siqueira
```

```
*/

const int INF = 0x3F3F3F3F;
const double PI = 2*acos(0);
const double EPS = 1e-10;

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

vector<string> splitAll(string s, string t)
{
    vector<string> v;

    for (int p = 0; (p = s.find(t)) != s.npos; )
    {
        v.push_back(s.substr(0, p));
        s = s.substr(p + t.size());
    }

    v.push_back(s);

    return v;
}
```

5.11 DIVIDIR NA PRIMEIRA OCORRENCIA

```
#include <cstdio>
#include <string>
#include <cstring>
#include <vector>
#include <algorithm>
#include <map>
#include <utility>
#include <cmath>
#include <queue>
#include <stack>
#include <set>
#include <deque>
#include <iostream>
#include <sstream>

#define REP(i,n) for(int i=0;i<(int)n;++i)
#define EACH(i,c) for(__typeof((c).begin()) i=(c).begin(); i!=(c).end(); ++i)
#define ALL(c) (c).begin(), (c).end()
#define SIZE(x) (int((x).size()))

#define MAXSZ 100

using namespace std;

/*

    DIVIDIR NA PRIMEIRA OCORRENCIA

    Aplicacoes:

        Divide uma string na primeira ocorrencia
        de um padrao

    Como chamar a funcao:

        1) Chamar a funcao split(s,t), onde:

            string s -> string original
            string t -> string a ser buscada em s

    Resultado da funcao:

        A funcao retorna um vector<string>
        contendo a string s quebrada na primeira
        corencia de t.

    Complexidade do algoritmo:

        O(?)

    Problemas resolvidos:

    Adicionado por:

        Jorge Gabriel Siqueira

*/
```

```
const int INF = 0x3F3F3F3F;
const double PI = 2*acos(0);
const double EPS = 1e-10;

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

vector<string> split(string s, string t)
{
    vector<string> v;
    int p = s.find(t);

    if (p != s.npos)
    {
        v.push_back(s.substr(0, p));
        s = s.substr(p + t.size());
    }

    v.push_back(s);

    return v;
}
```

5.12 PALINDROME

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

#define ALL(c) (c).begin(), (c).end()

using namespace std;

/*

    PALINDROME

    Aplicacoes:

        Determinar se uma string s eh palindrome

    Como chamar a funcao:

        1) Chamar a funcao is_palindrome(s), onde;

            s -> string a ser analisada

    Resultado da funcao:

        A funcao retorna:

            true se s eh palindrome
            false se s nao eh palindrome

    Complexidade do algoritmo:

        O(n)

    Problemas resolvidos:

        UVA 11309
        UVA 353
        UVA 401
        UVA 10018
        UVA 10945
        UVA 112211

    Adicionado por:

        Jorge Gabriel Siqueira

*/

bool is_palindrome(string const &s)
{
    return equal(ALL(s),s.rbegin());
}
```

5.13 PROXIMO PALINDROME

```
#include <cstdio>
#include <string>
#include <cstring>
#include <vector>
#include <algorithm>
#include <map>
#include <utility>
#include <cmath>
#include <queue>
#include <stack>
#include <set>
#include <deque>
#include <iostream>
#include <sstream>

#define REP(i,n) for(int i=0;i<(int)n;++i)
#define EACH(i,c) for(__typeof((c).begin()) i=(c).begin(); i!=(c).end();
++i)
#define ALL(c) (c).begin(), (c).end()
#define SIZE(x) (int((x).size()))

#define MAXSZ 100

using namespace std;

/*

    PROXIMO NUMERO PALINDROME

    Aplicacoes:

       Dado um numero K encontra o proximo numero maior
       de que K que seja palindrome.

    Como chamar a funcao:

        1) Armazenar o numero na variavel K[]

        2) Chamar a funcao find_palin()

    Resultado da funcao:

        A funcao reescreve K com o proximo numero
        palindrome.

    Complexidade do algoritmo:

        O(n)

    Problemas resolvidos:

        SPOJ PALIN

    Adicionado por:

        Jorge Gabriel Siqueira
```

```
*/

const int INF = 0x3F3F3F3F;
const double PI = 2*acos(0);
const double EPS = 1e-10;

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

char K[1000002];
int flag;

void find_palin()
{
    int len,i,tmp,t,tmp1;
    len = strlen(K);
    flag = 1;

    for(i=0; i<len; i++)
    {
        if(K[i] != '9')
        {
            flag = 0;
            break;
        }
    }

    if(flag == 1)
    {
        K[0] = '1';
        for(i=1; i<len; i++)
        K[i] = '0';
        K[len] = '1';
        K[len+1] = '\0';
        return ;
    }

    flag = 0;

    for(i=0; i<len/2; i++)
    {
        if(K[i] < K[len-i-1])
        flag = -1;
        else if(K[i] > K[len-i-1])
        flag = 1;
        K[len-i-1] = K[i];
    }

    if(flag == -1 || flag==0)
    {
        t = 0;

        if(len%2 == 0) tmp1 = len/2-1;
        else tmp1 = len/2;
```

```
        while(K[tmp1-t] == '9')
        {
            K[tmp1-t] = '0';
            K[len-1-tmp1+t] = '0';
            t ++;
        }

        K[tmp1-t] ++;
        K[len-1-tmp1+t] = K[tmp1-t];
    }
}
```

5.14 BIT PARALLEL ALGORITHM

```
#include <cstdio>
#include <string>
#include <cstring>
#include <vector>
#include <algorithm>
#include <map>
#include <utility>
#include <cmath>
#include <queue>
#include <stack>
#include <set>
#include <deque>
#include <iostream>
#include <sstream>

#define REP(i,n) for(int i=0;i<(int)n;++i)
#define EACH(i,c) for(__typeof((c).begin()) i=(c).begin(); i!=(c).end();
++i)
#define ALL(c) (c).begin(), (c).end()
#define SIZE(x) (int((x).size()))

using namespace std;

const int INF = 0x3F3F3F3F;
const double PI = 2*acos(0);
const double EPS = 1e-10;


/*

    BIT PARALLEL ALGORITHM

    Aplicacoes:

        Dado uma string pattern e um texto buscar todas
        as ocorencias de substrings de pattern com ate
        n erros de edicao.

    Como chamar a funcao:

        1) Chamar a funcao BPR(pattern, text, errors)
           onde:

            pattern -> padrao a ser buscado
            text -> texto onde o padrao sera buscado
            errors -> quantidade de erros de edicao tolerados

    Resultado da funcao:

        A funcao retorna um vector<int> contendo a posicao
        onde ocorreram os matches

    Complexidade do algoritmo:

        O(n*m), onde n eh o numero de letras do texto e
        m eh o numero de erros permitidos
```

```
    Problemas resolvidos:

    Adicionado por:

        Jorge Gabriel Siqueira

*/

vector<int> BPR(string pattern, string text, int errors)
{
    int B[256];
    memset(B, 0, sizeof(B));
    vector<int> ret;

    // Initialize all characters positions
    for (int i = 0; i < pattern.size(); i++)
    {
        B[(int)pattern[i]] |= 1 << i;
    }
    // Initialize NFA states
    int states[errors+1];
    for(int i= 0; i <= errors; i++)
    {
        states[i] = (i == 0) ? 0 : (1 << (i - 1) | states[i-1]);
    }
    //
    int oldR, newR;
    int exitCriteria = 1 << pattern.size() -1;

    for (int i = 0; i < text.size(); i++)
    {
        oldR = states[0];
        newR = ((oldR << 1) | 1) & B[text[i]];
        states[0] = newR;

        for (int j = 1; j <= errors; j++)
        {
            newR = ((states[j] << 1) & B[text[i]]) | oldR | ((oldR | newR)
<< 1);

            oldR = states[j];
            states[j] = newR;
        }

        if ((newR & exitCriteria) != 0)
        {
            ret.push_back(i+1);
        }
    }
    return ret;
}

int main()
{
    string text = "pato==============pateta===============caneca";
    string pattern = "pat";
```

```
    vector<int> ret = BPR(pattern,text,0);
    for(int i=0; i<ret.size(); i++) cout << ret[i] << " ";
    return 0;
}
```

5.15 BURKHARD KELLER TREE

```
#include <assert.h>
#include <queue>
#include <stdio.h>
#include <iostream>
#include <stdlib.h>

using namespace std;

/*

    BURKHARD KELLER TREE

    Aplicacoes:

        Busca aproximada em dicionarios de forma
        eficiente.

    Como chamar a funcao:

        1) Criar a tree:

            BkTree<string, &EditDistance> tree;

        2) Inserir as palavras no dicionario:

            tree.insert("word");

        3) Para fazer uma busca aproximada no dicionario chamar:

            tree.getWithinDistance( "word", errors ); onde:

                errors -> numero de edicoes permitidas em "word"

        4) A estrutura ainda fornece as funcoes auxiliares:

            int size() -> retorna o numero de palavras no
                          dicionario.

            bool empty() -> retorna true se o dicionario esta
                            vazio ou false caso contrario.

            int count(item) -> retorna 1 caso a palavra esteja
                               contida no dicionario, ou false
                               caso contrario.

    Resultado da funcao:

        A funcao getWithinDistance() retorna o numero de
        matches encontrados no dicionario.

    Complexidade do algoritmo:

        O(?)

    Problemas resolvidos:
```

```
    Adicionado por:

        Jorge Gabriel Siqueira

*/

template<class T, int (*Distance)( const T&, const T& )>
class BkTree
{
public:
    BkTree()
    {
        root_ = NULL;
        size_ = 0;
    }

    ~BkTree()
    {
        if( root_ ) delete root_;
    }

    /** Does nothing if the tree already contains this element. **/
    void insert(T item)
    {
        if( !root  )
        {
            size_ = 1;
            root_ = new Node(item, -1);
            return;
        }

        Node *t = root_;
        while( true )
        {
            int d = Distance( t->item, item );
            if( !d ) return;
            Node *ch = t->firstChild;
            while( ch )
            {
                if( ch->distToParent == d )
                {
                    t = ch;
                    break;
                }
                ch = ch->nextSibling;
            }
            if( !ch )
            {
                Node *newChild = new Node(item, d);
                newChild->nextSibling = t->firstChild;
                t->firstChild = newChild;
                size_++;
                break;
            }
        }
    }

    int size() const
```

```
    {
        return size_;
    }

    bool empty() const
    {
        return !size_;
    }

    int count( T item ) const
    {
        return getWithinDistance( item, 0 );
    }

    /**
     * Finds all elements within a distance of 'k' of 'center', inclusive.
     * Fills 'result' and returns the number of elements found.
     * Gives no guarantees about the order in which results are returned.
     **/
    int getWithinDistance(T center, int k, vector<T> *result = NULL) const
    {
        assert( k >= 0 );
        if( !root_ ) return 0;

        int found = 0;
        queue< Node* > q;
        q.push( root_ );

        while( !q.empty() )
        {
            Node *t = q.front();
            q.pop();
            int d = Distance( t->item, center );
            if( d <= k )
            {
                if( result ) result->push_back( t->item );
                found++;
            }

            Node *ch = t->firstChild;
            while( ch )
            {
                if( d - k <= ch->distToParent && ch->distToParent <= d + k
)
                    q.push(ch);
                ch = ch->nextSibling;
            }
        }

        return found;
    }

private:
    struct Node
    {
        T item;
        int distToParent;
        Node *firstChild;
```

```
        Node *nextSibling;

        Node(T x, int dist)
        {
            item = x;
            distToParent = dist;
            firstChild = nextSibling = NULL;
        }

        ~Node()
        {
            if( firstChild ) delete firstChild;
            if( nextSibling ) delete nextSibling;
        }
    };

    Node *root_;
    int size_;
};


//------------- TESTING --------------
#include <stdio.h>
#include <string>

int EditDistance( const string &s, const string &t )
{
    int m = s.size();
    int n = t.size();
    vector< vector< int > > tab( m + 1, vector< int >( n + 1, 0 ) );
    for( int i = m - 1; i >= 0; i-- ) for( int j = n - 1; j >= 0; j-- )
        {
            if( s[i] == t[j] ) tab[i][j] = tab[i + 1][j + 1];
            else tab[i][j] = 1 + (min(tab[i + 1][j] , min(tab[i][j] ,
tab[i][j + 1])));
        }
    return tab[0][0];
}

int main()
{
    BkTree<string, &EditDistance> tree;
    assert( tree.size() == 0 );
    assert( tree.empty() );

    tree.insert( "boobs" );
    assert( tree.size() == 1 );
    assert( !tree.empty() );
    assert( tree.count( "boobs" ) );
    assert( !tree.count( "books" ) );
    assert( tree.getWithinDistance( "boobs", 0 ) == 1 );
    assert( tree.getWithinDistance( "boobs", 1 ) == 1 );
    assert( tree.getWithinDistance( "books", 0 ) == 0 );
    assert( tree.getWithinDistance( "books", 1 ) == 1 );

    tree.insert( "books" );
    assert( tree.size() == 2 );
    assert( !tree.empty() );
```

```
    assert( tree.count( "boobs" ) );
    assert( tree.count( "books" ) );
    assert( !tree.count( "boots" ) );
    assert( tree.getWithinDistance( "books", 0 ) == 1 );
    assert( tree.getWithinDistance( "books", 1 ) == 2 );
    assert( tree.getWithinDistance( "boots", 1 ) == 2 );
    assert( tree.getWithinDistance( "boobs", 1 ) == 2 );

    vector< string > pool;
    int n = 100;
    int len = 10;
    for( int i = 0; i < n; i++ )
    {
        string word;
        while( word.size() < len ) word += 'a' + rand() % 26;
        pool.push_back( word );
    }

    vector< vector< int > > dist( n, vector< int >( n ) );
    for( int i = 0; i < n; i++ ) for( int j = 0; j < n; j++ )
            if( i == j )
                dist[i][j] = 0;
            else
                assert( dist[i][j] = EditDistance( pool[i], pool[j] ) );

    BkTree< string, EditDistance > tree2;
    for( int i = 0; i < n; i++ )
    {
        assert( tree2.size() == i );
        assert( !tree2.count( pool[i] ) );
        tree2.insert( pool[i] );
        assert( tree2.count( pool[i] ) );
        for( int d = 0; d <= len; d++ )
        {
            int ans = 0;
            for( int j = 0; j <= i; j++ ) if( dist[i][j] <= d ) ans++;
            assert( tree2.getWithinDistance( pool[i], d ) == ans );
        }
    }
    assert( tree2.size() == n );

    puts( "Oll Korrekt." );
}
```

5.16 MAIOR SUBSEQUENCIA PALINDROME PARA BIG TEXTOS

```
#include <iostream>
#include <fstream>
#include <string>
#include <assert.h>

using namespace std;

/*

    LSP - LONGEST SUBSET PALINDROME

    Aplicacoes:

        Encontrar a maior subsequencia palindrome de um texto

    Como chamar a funcao:

        1) Deve-se inicialmente colocar o texto original na string "s"
           se houver restricoes (como desconsiderar maiusculas e
           minusculas ou ignorar simbols de pontuacao) deve-se tratar
           estes casos e armazenar esta string em ss, caso nao existam
           restricoes simplesmente copie a string "s" para a string "ss"

        2) Chame a funcao lsp()

        Observacao: Cuidado com a complexidade, algoritmo indicado para
        textos longos com matches de palindromes pequenos

    Resultado da funcao:

        A funcao armazena em:

            string output -> armazena a maior subsequencia palindrome
            int maxl -> armazena o tamanho de output

    Complexidade do algoritmo:

        O(n*m), onde m eh o tamanho da maior substring palindrome e
        n eh o tamanho do texto (armazenado em ss)

    Problemas resolvidos:

        USACO 1.3.3

    Adicionado por:

        Jorge Gabriel Siqueira

*/

string s, ss;
int maxp, maxq, maxl = 0;
string output;

void findpal (int p, int q)
{
```

```
    int i=-1, j=-1;
    while (p >= 0 && q < ss.size() && ss[p] == ss[q])
    {
        i = p;
        j = q;
        p--;
        q++;
    }

    if (i != -1 && j != -1 && j - i + 1 > maxl)
    {
        maxp = i;
        maxq = j;
        maxl = j - i + 1;
    }
}

void lsp()
{
    maxl = 0;
    for (int i = 1; i < ss.size(); i++)
    {
        //odd palindrome
        findpal (i - 1, i + 1);
        //even palindrome
        findpal (i - 1, i);
    }

    output = "";
    int alpha = -1;

    for (int i = 0; i < s.size() && maxq >= alpha + 1; i++)
    {
        if (isalpha (s[i]))
            alpha++;
        if (maxp <= alpha && maxq >= alpha)
            output += s[i];
    }
}

int main()
{
    s = ss = "sdabccbagfa";
    lsp();
    assert(output == "abccba" && maxl == 6);

    s = ss = "gaaasmcfjafsdmadamimadamasdagsaksdj";
    lsp();
    assert(output == "madamimadam" && maxl == 11);

    s = "Confucius say: Madam, I'm Adam.";
    ss = "confuciussaymadamimadam";
    lsp();
    assert(output == "Madam, I'm Adam" && maxl == 11);

    return 0;
}
```

5.17 SUFFIX ARRAY

```
#include <stdio.h>
#include <vector>
#include <algorithm>
#include <string.h>

#define REP(i,n) for(int i=0;i<(int)n;++i)
#define EACH(i,c) for(__typeof((c).begin()) i=(c).begin(); i!=(c).end();
++i)
#define ALL(c) (c).begin(), (c).end()
#define SIZE(x) (int((x).size()))
#define COMP(h, k) (h == m || (k+h<n && p[h]<t[k+h]))

using namespace std;

/*

    Principais Aplicacoes

        Fazer varias query's de ocorrencias de substrings numa string
        Contar o numero de ocorrencias de uma substring em uma string

    Procedimento

        3.5.1) Criar uma Sufixx Tree

            int *sa = buildSA(char *t, int n);

            t -> texto a ser usado para busca
            n -> numero de caracteres do texto

        3.5.2.1) Fazer a busca 1 O(m log n)

            int pos = find(char *t, int n, char *p, int m, int *sa);

            t -> texto a ser buscado
            n -> numero de caraceres do texto a ser buscado
            p -> substring a ser buscada em t
            m -> numero de caracteres de p
            sa -> suffix tree construida no passo 1

            pos -> posicao da ultima ocorrencia de p em t / -1 caso nao
encontre

        3.5.2.2) Fazer a busca 2 O(m + log n)

            int *lcp = buildLCP(char *t,int n,int *sa);
            int *rqm = buildRMQ(lcp,n+1);
            int pos = find(t, n, p, m, sa, rqm);

            lcp -> LCP construido a partir  texto e da suffix tree
            rqm -> range minimum query construido a partir do lcp
            t -> texto a ser buscado
            n -> numero de caraceres do texto a ser buscado
            p -> substring a ser buscada em t
            m -> numero de caracteres de p
            sa -> suffix tree construida no passo 1
```

```
            pos -> posicao da ultima ocorrencia de p em t / -1 caso nao
encontre


    Testado em

        10679 - I Love Strings!!

*/

// Larsson-Sadakane's Suffix array Construction: O(n (log n)^2)
struct SAComp
{
    const int h, *g;
    SAComp(int h, int* g) : h(h), g(g) {}

    bool operator() (int a, int b)
    {
        return a == b ? false : g[a] != g[b] ? g[a] < g[b] : g[a+h] <
g[b+h];
    }
};

int *buildSA(char* t, int n)
{
    int g[n+1], b[n+1], *v = new int[n+1];

    REP(i,n+1) v[i] = i, g[i] = t[i];

    b[0] = 0;
    b[n] = 0;
    sort(v, v+n+1, SAComp(0, g));

    for(int h = 1; b[n] != n ; h *= 2)
    {
        SAComp comp(h, g);
        sort(v, v+n+1, comp);
        REP(i, n) b[i+1] = b[i] + comp(v[i], v[i+1]);
        REP(i, n+1) g[v[i]] = b[i];
    }

    return v;
}

// Naive matching O(m log n)
int find(char *t, int n, char *p, int m, int *sa)
{
    int a = 0, b = n;

    while (a < b)
    {
        int c = (a + b) / 2;

        if (strncmp(t+sa[c], p, m) < 0) a = c+1;
        else b = c;
    }
```

```
    return strncmp(t+sa[a], p, m) == 0 ? sa[a] : -1;
}

// Kasai-Lee-Arimura-Arikawa-Park's simple LCP computation: O(n)
int *buildLCP(char *t, int n, int *a)
{
    int h = 0, b[n+1], *lcp = new int[n+1];

    REP(i, n+1) b[a[i]] = i;

    REP(i, n+1)
    {
        if (b[i])
        {
            for (int j = a[b[i]-1]; j+h<n && i+h<n && t[j+h] == t[i+h];
++h);
            lcp[b[i]] = h;
        }
        else lcp[b[i]] = -1;

        if (h > 0) --h;
    }

    return lcp;
}

// outer LCP computation: O(m - o)
int computeLCP(char *t, int n, char *p, int m, int o, int k)
{
    int i = o;
    for (; i < m && k+i < n && p[i] == t[k+i]; ++i);
    return i;
}
```

6. ESTRUTURAS DE DADOS

6.1 BIT MANIPULATION

```
#include <math.h>
#include <stdio.h>
#include <stack>
#include <iostream>

using namespace std;

/*

    BIT MANIPULATION

    Aplicacoes:

        Biblioteca de manipulacao binaria

    Funcoes suportadas:

        isOn()

            Parametros:

                S -> numero em representacao decimal
                j -> posicao do bit a ser analisada

            Retorna:

                Valor diferente de 0 -> o bit esta ativo
                0 -> o bit esta inativo

        setBit()

            Parametros:

                S -> numero em representacao decimal
                j -> posicao do bit a ser setado

            Funcao:

                Seta o j-esimo bit para 1

        clearBit()

            Parametros:

                S -> numero em representacao decimal
                j -> posicao do bit a ser setado

            Funcao:

                Seta o j-esimo bit para 0

        toggleBit()

            Parametros:
```

```
            S -> numero em representacao decimal
            j -> posicao do bit a ser setado

        Funcao:

            Seta o j-esimo bit para 0, caso ele seja 1
            Seta o j-esimo bit para 1, caso ele seja 0

    lowBit()

        Parametros:

            S -> numero em representacao decimal

        Retorna:

            2^n, onde n eh a posicao do bit menos
            significativo setado como 1

    setAll()

        Parametros:

            S -> numero em representacao decimal
            n -> numero de bits a serem setados

        Funcao:

            Seta os n primeiros bits de S para 1

    modulo()

        Parametros:

            x -> numero em representacao decimal
            N -> numero em representacao decimal

        Retorna:

            x % N, onde N eh uma potencia de 2

    isPowerOfTwo()

        Parametros:

            x -> numero em representacao decimal

        Retorna:

            true -> o numero eh uma potencia de 2
            false -> o numero nao eh uma potencia de 2

    nearestPowerOfTwo()

        Parametros:

            x -> numero em representacao decimal
```

```
            Retorna:

                A potencia de 2 mais proxima de x

    Problemas resolvidos:

    Adicionado por:

        Jorge Gabriel Siqueira

*/

#define isOn(S, j) (S & (1 << j))
#define setBit(S, j) (S |= (1 << j))
#define clearBit(S, j) (S &= ~(1 << j))
#define toggleBit(S, j) (S ^= (1 << j))
#define lowBit(S) (S & (-S))
#define setAll(S, n) (S = (1 << n) - 1)
#define modulo(x, N) ((x) & (N - 1))   //
#define isPowerOfTwo(x) ((x & (x - 1)) == 0)
#define nearestPowerOfTwo(x) ((int)pow(2.0, (int)((log((double)x) /
log(2.0)) + 0.5)))

void printSet(int _S) //Imprime na representacao binaria
{
    printf("S = %2d = ", _S);
    stack<int> st;
    while (_S)
        st.push(_S % 2), _S /= 2;
    while (!st.empty())
        printf("%d", st.top()), st.pop();
    printf("\n");
}
```

6.2 UNION FIND

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

using namespace std;

/*

    UNION FIND

    Aplicacoes:

        Estrutura de dados que trata da manipulacao
        de conjuntos, abrangindo basicamente 2 operacoes:

            Union -> Juntar dois subconjuntos em um so

            Find -> Determinar o subconjunto de um determinado
                    elemento

    Como chamar a funcao:

        1) Criar a estrutura:

            UnionFind unionFind = UnionFind(n); onde:

                n -> numero de elementos do conjunto

        2) Para unir o conjunto x com o conjunto y chamar:

            unionFind.unionSet(x,y);

        3) Para saber o conjunto do elemento x chamar:

            unionFind.root(x);

    Resultado da funcao:

        A funcao unionFind.root(x) retorna o indice do
        conjunto onde x esta contido.

    Problemas resolvidos:

    Adicionado por:

        Jorge Gabriel Siqueira

*/

struct UnionFind
{
    vector<int> data;

    /*

    Construtor
```

```
    Parametros:
        int size -> tamanho do conjunto

    */

    UnionFind(int size) : data(size, -1) { }

    /*

    unionSet - Une o conjunto do elemento x ao conjunto do elemento y

    Parametros:
        int x -> elemento de um conjunto A
        int y -> elemento de um conjunto B a ser unido em A

    Retorna:
        true se conseguiu unir, ou seja, x != y
        false se x pertence ao mesmo conjunto de y

    */

    bool unionSet(int x, int y)
    {
        x = root(x);
        y = root(y);

        if (x != y)
        {
            if (data[y] < data[x]) swap(x, y);
            data[x] += data[y];
            data[y] = x;
        }
        return x != y;
    }

    /*

    findSet - Verifica se x e y pertencem ao mesmo conjunto

    Parametros:
        int x -> elemento de um conjunto
        int y -> elemento de um conjunto

    Retorna:
        true se x pertence ao mesmo conjunto de y
        false se x nao pertence ao mesmo conjunto de y

    */

    bool findSet(int x, int y)
    {
        return root(x) == root(y);
    }

    /*

    root - retorna o conjunto de um elemento x
```

```
    Parametros:
        int x -> elemento a ser verificado

    Retorna:
        indice do conjunto que contem x

    */

    int root(int x)
    {
        return data[x] < 0 ? x : data[x] = root(data[x]);
    }

    /*

    size - retorna o tamanho do conjunto onde x esta contido
    Parametros:
        int x -> elemento a ser verificado

    Retorna:
        retorna o tamanho do conjunto onde x esta contido

    */

    int size(int x)
    {
        return -data[root(x)];
    }
};

int main()
{
    UnionFind unionFind = UnionFind(10);

    unionFind.unionSet(1,2);
    unionFind.unionSet(3,4);
    unionFind.unionSet(2,3);

    cout << unionFind.root(4);
}
```

6.3 FENWICK TREE

```
#include <stdio.h>
#include <string.h>
#define MAXSZ 100

/*

    FENWICK TREE

    Aplicacoes:

        Determinar a soma dos elementos contidos em
        um sub-intervalo de um conjunto.

    Como chamar a funcao:

        1) Armazenar em c o numero de elementos do
           conjunto

        2) Para soma o valor inc a posicao k do
           conjunto chamar:

                update(k, inc);

        3) Para saber a soma do intervalo que comeca
           em from e termina em to chamar:

                query(from, to);

    Resultado da funcao:

        A funcao query(from,to) retorna a soma do intervalo
        que comeca em from e termina em to

    Complexidade do algoritmo:

        O(log(n))

    Problemas resolvidos:

    Adicionado por:

        Jorge Gabriel Siqueira

*/

int tree[MAXSZ], c;

void create()
{
    memset(tree,0,c*sizeof(int));
}

int query(int from, int to)
{
    int sum;
```

```
    if (from != 0) return query(0, to) - query(0, from-1);

    for (sum=0; to >= 0; to = (to & (to + 1)) - 1) sum += tree[to];

    return sum;
}

void update(int k, int inc)
{
    for ( ; k < c; k |= k + 1) tree[k] += inc;
}
```

6.4 FENWICK TREE 2D

```
#include <stdio.h>
#include <string.h>
#define MAXN 1000
#define MAXM 1000

/*

    FENWICK TREE 2D

    Aplicacoes:

        Determinar a soma dos elementos contidos em
        um sub-retangulo de uma matriz nxm.

    Como chamar a funcao:

        1) Armazenar as dimensoes da matriz em n(linhas)
           e m(colunas)

        2) Chamar a funcao create() para criar a arvore

        3) Para somar o valor inc a posicao (x,y) da
           matriz chamar:

                update(x,y,inc);

        4) Para saber a soma do sub-retangulo com
           canto superior esquerdo (x1,y1) e canto inferior
           direito (x2,y2) chamar:

                query(x1,y1,x2,y2);

    Resultado da funcao:

        A funcao query(x1,y1,x2,y2) retorna a soma do intervalo
        com canto superior esquerdo (x1,y1) e canto inferior
        direito (x2,y2).

    Complexidade do algoritmo:

        Queries em O(log(n)*log(m))

    Problemas resolvidos:

    Adicionado por:

        Jorge Gabriel Siqueira

*/

int tree[MAXN][MAXM]; //arvore
int matrix[MAXN][MAXM]; //matriz original
int n, m; //dimensoes da matriz

void create ()
{
```

```
    for (int i = 0; i < n; ++i)
    {
        memset (tree[i], 0, sizeof(int)*m);
        memset (matrix[i], 0, sizeof(int)*m);
    }
}

int query2 (int x, int y)
{
    int sum;
    for (sum = 0; y >= 0; y = (y&(y+1))-1)
        sum += tree[x][y];
    return sum;
}

int query (int x1, int y1, int x2, int y2)
{
    int sum;
    if (x1 != 0)
        return query(0,y1,x2,y2) - query(0,y1,x1-1,y2);
    for (sum = 0; x2 >= 0; x2 = (x2&(x2+1))-1)
    {
        sum += query2 (x2,y2);
        if (y1)
            sum -= query2 (x2,y1-1);
    }
    return sum;
}

void update (int x, int y, int incremento)
{
    int y2;
    matrix[x][y] += incremento;
    for (; x < n; x |= x + 1)
        for (y2 = y; y2 < m; y2 |= y2 + 1)
            tree[x][y2] += incremento;
}
```

6.5 RANGE MINIMUM QUERY

```
#include <stdio.h>
#include <vector>

#define REP(i,n) for(int i=0;i<(int)n;++i)
#define EACH(i,c) for(__typeof((c).begin()) i=(c).begin(); i!=(c).end();
++i)
#define ALL(c) (c).begin(), (c).end()
#define SIZE(x) (int((x).size()))
#define COMP(h, k) (h == m || (k+h<n && p[h]<t[k+h]))

using namespace std;

/*

    RANGE MINIMUM QUERY

    Aplicacoes:

        Queries de menor elemento em um sub-intervalo de
        um conjunto.

    Como chamar a funcao:

        1) Construir a RMQ chamando a funcao:

            int *rmq = buildRMQ(a,n), onde:

                a -> vetor contendo os elementos do conjunto
                n -> numero de elementos do conjunto

            A RQM fica armazenada na variavel int *rmq

        2) Para realizar as queries chamar:

            minimum(x, y, rmq, n); onde:

                x -> inicio do intervalo
                y - fim do intervalo
                rmq -> RMQ (variavel int *rmq do passo 1)
                n -> numero de elementos do conjunto

    Resultado da funcao:

        A funcao minimum(x,y,rmq,n) retorna o valor
        do menor elemento contido no subconjunto que
        inicia em x e termina em y.

    Complexidade do algoritmo:

        O(n) -> construcao
        O(1) -> queries

    Problemas resolvidos:

    Adicionado por:
```

```
        Jorge Gabriel Siqueira

*/

int *buildRMQ(int *a, int n)
{
    int logn = 1;
    for (int k = 1; k < n; k *= 2) ++logn;
    int *r = new int[n * logn];
    int *b = r;
    copy(a, a+n, b);
    for (int k = 1; k < n; k *= 2)
    {
        copy(b, b+n, b+n);
        b += n;
        REP(i, n-k) b[i] = min(b[i], b[i+k]);
    }
    return r;
}

int minimum(int x, int y, int *rmq, int n)
{
    int z = y - x, k = 0, e = 1, s; // y - x >= e = 2^k
    s = ( (z & 0xffff0000) != 0 ) << 4;
    z >>= s;
    e <<= s;
    k |= s;
    s = ( (z & 0x0000ff00) != 0 ) << 3;
    z >>= s;
    e <<= s;
    k |= s;
    s = ( (z & 0x000000f0) != 0 ) << 2;
    z >>= s;
    e <<= s;
    k |= s;
    s = ( (z & 0x0000000c) != 0 ) << 1;
    z >>= s;
    e <<= s;
    k |= s;
    s = ( (z & 0x00000002) != 0 ) << 0;
    z >>= s;
    e <<= s;
    k |= s;
    return min( rmq[x+n*k], rmq[y+n*k-e+1] );
}
```

6.6 RANGE MAXIMUM QUERY

```
#include <stdio.h>
#include <vector>

#define REP(i,n) for(int i=0;i<(int)n;++i)
#define EACH(i,c) for(__typeof((c).begin()) i=(c).begin(); i!=(c).end();
++i)
#define ALL(c) (c).begin(), (c).end()
#define SIZE(x) (int((x).size()))
#define COMP(h, k) (h == m || (k+h<n && p[h]<t[k+h]))

using namespace std;

/*

    RANGE MAXIMUM QUERY

    Aplicacoes:

        Queries de maior elemento em um sub-intervalo de
        um conjunto.

    Como chamar a funcao:

        1) Armazenar o numero de elementos do conjunto em n

        2) Armazenar o conjunto no vetor r[].

        3) Chamar a funcao construct() para construir o RMQ

        4) Chamar a funcao getmax(a, b) para realizar uma query
           no intervalo (a,b)

    Resultado da funcao:

        A funcao getmax(a, b) retorna o valor
        do maior elemento contido no subconjunto que
        inicia em a e termina em b.

    Complexidade do algoritmo:

        O(n) -> construcao
        O(1) -> queries

    Problemas resolvidos:

        UVA 11235

    Adicionado por:

        Jorge Gabriel Siqueira

*/

int r[50010], n;
int mm[50010][18]; // Ou N x log(N) + 1
```

```
void construct()
{
    int i,j,b;

    for(i=0; i<n; i++) mm[i][0]=r[i];

    for(i=1; i<18; i++)
    {
        for(j=0; (j+(1<<i)-1)<n; j+=(1<<i)) mm[j][i] = max(mm[j][i-1],
mm[j+(1<<(i-1))][i-1]);
    }
}

int getmax(int a, int b)
{
    if(a>b) return -1;

    for(int i=17; i>=0; i--)
    {
        if((a % (1 << i)) == 0 && (a + (1 << i) - 1) <= b)
            return max(mm[a][i], getmax(a + (1 << i), b));
    }
}
```

6.7 LOWEST COMMON ANCESTOR

```
#include <iostream>
#include <stdio.h>
#include <string.h>
#include <vector>
#define MAXN 100001
#define LOGNMAX 17 //log2(MAXN)

using namespace std;

/*

    LCA - LOWEST COMMON ANCESTOR

    Aplicacoes:

        -> Menor ancestral comum entre 2 vertices de uma arvore

        -> Distancia entre 2 vertices de uma arvore (otimo quando
           se tem que realizar sucessivas consultas seguidas!)

    Como chamar a funcao:

        1) Armazenar em n o numero de nos da arvore

        2) Para armazenar um  aresta que liga o no x ao no y
           com custo c, faca:

                graph[x].push_back(no(y,c));
                graph[y].push_back(no(x,c));
                (Dupla direcao!!!)

        3) Monte a arvore chamando a funcao pre_process()

        4) Para saber o menor ancestral comum entre o vertice
           a e b chame a funcao LCA(a,b)

    Resultado da funcao:

        A funcao LCA(a,b) retona um  pair <int, long long> onde:

            first -> Menor Ancestral Comum de a e b
            second -> Distancia de a ate b

    Complexidade do algoritmo:

        O(?) -> construcao

    Problemas resolvidos:

        ANTS10 (SPOJ BR)

    Adicionado por:

        Jorge Gabriel Siqueira

*/
```

```
struct X
{
    long long int v, c; //aresta[i][j] = 20 ---> v = j, c = 20
};

X no (long long int v, long long int c)
{
    X novo;
    novo.v = v;
    novo.c = c;
    return novo;
}

vector <X> graph[MAXN];
long long int pai[MAXN][LOGNMAX], distancia[MAXN][LOGNMAX], prof[MAXN];
int n;

void monta (long long int v, long long int p, long long int pro, long long
int d)
{
    prof[v] = pro;
    pai[v][0] = p;
    distancia[v][0] = d;
    for (int i = 1; i < LOGNMAX; i++)
    {
        pai[v][i] = pai[pai[v][i-1]] [i-1];
        distancia[v][i] = distancia[v][i-1] + distancia[pai[v][i-1]][i-1];
    }
    for (int i = 0; i < graph[v].size(); ++i)
    {
        if (graph[v][i].v == p)
            continue;
        monta (graph[v][i].v, v, pro + 1, graph[v][i].c);
    }
}

void pre_process()
{
    prof[0] = 0;
    for (int i = 0; i < LOGNMAX; ++i)
    {
        pai[0][i] = 0;
        distancia[0][i] = 0;
    }
    for (int i = 0; i < graph[0].size(); ++i)
        monta(graph[0][i].v, 0, 1, graph[0][i].c);
}

pair <int, long long int> LCA (int a, int b)
{
    long long int d = 0;
    int pa = a, pb = b;
    if (prof[pb] > prof[pa])
        swap(pa, pb);
    while (prof[pa] > prof[pb])
    {
        int j = 0;
```

```
        for (int i = 0; i < LOGNMAX; ++i)
        {
            if (prof[pai[pa][i]] < prof[pb])
                break;
            j = i;
        }
        d += distancia[pa][j];
        pa = pai[pa][j];
    }
    while (pa != pb)
    {
        int j = 0;
        for (int i = 0; i < LOGNMAX; ++i)
        {
            if (pai[pa][i] == pai[pb][i])
                break;
            j=i;
        }
        d += distancia[pa][j];
        d += distancia[pb][j];
        pa = pai[pa][j];
        pb = pai[pb][j];
    }
    return make_pair(pa, d);
}
```

6.8 RANGE TREE

```
#include <stdio.h>
#include <vector>
#include <algorithm>
#include <iostream>
#define MAXN 1000 //numero maximo de pontos

using namespace std;

/*

    RANGE TREE

    Aplicacoes:

        Dado um conjunto de pontos do plano cartesiano,
        determinar de forma eficiente quantos pontos
        estao dentro do retangulo delimitado pelos pontos
        (x1,y1) e (x2,y2), onde (x1,y1) eh a coordenada
        do canto inferior esquerdo e (x2,y2) eh a
        coordenada do cando superior direito do retangulo.

    Como chamar a funcao:

        1) Armazenar os pontos em vector<pt> pts

        2) Chamar a funcao build() pra montar a arvore

        3) Realizar as queries chamando:

            query (0, 0, xs.size() - 1, x1, x2, y1, y2) onde:

            (x1,y1) -> coordenada do canto inferior esquerdo
            (x2,y2) -> coordenada do canto superior direito

    Resultado da funcao:

        A funcao query() retorna o numero de pontos contidos
        no retangulo definido pelos pontos (x1,y1) e (x2,y2)

    Complexidade do algoritmo:

        O(?)

    Problemas resolvidos:

    Adicionado por:

        Jorge Gabriel Siqueira

*/


const double EPS = 1e-9, INF = 1e9;

inline int sgn(double a)
{
```

```
    return a > EPS ? 1 : (a < -EPS ? -1 : 0);
}

inline int cmp(double a, double b)
{
    return sgn(a - b);
}

struct pt
{
    double x, y;
    pt(double x = 0, double y = 0) : x(x), y(y) { }
    bool operator==(pt p)
    {
        return cmp(x, p.x) == 0 && cmp(y, p.y) == 0;
    }
    bool operator < (pt p) const
    {
        return cmp (x, p.x) ? cmp (x, p.x) < 0 : cmp(y, p.y) < 0;
    }
    bool operator<= (pt p)
    {
        return *this < p || *this == p;
    }
    double operator|| (pt p)
    {
        return x*p.x + y*p.y;
    }
    double operator% (pt p)
    {
        return x*p.y - y*p.x;
    }
    pt operator~ ()
    {
        return pt(x, -y);
    }
    pt operator+ (pt p)
    {
        return pt(x + p.x, y + p.y);
    }
    pt operator- (pt p)
    {
        return pt(x - p.x, y - p.y);
    }
    pt operator* (pt p)
    {
        return pt(x*p.x - y*p.y, x*p.y + y*p.x);
    }
    pt operator/ (double t)
    {
        return pt(x/t, y/t);
    }
    pt operator/ (pt p)
    {
        return (*this * ~p)/(p||p);
    }
};
```

```
bool compy(pt a, pt b)
{
    return cmp(a.y, b.y) ? cmp(a.y, b.y) < 0 : cmp(a.x, b.x) < 0;
}

vector <pt> pts; //conjunto de pontos
vector <pt> tree[MAXN];
vector <double> xs;
vector <int> lnk[MAXN][2];

int recurse (int root, int left, int right)
{
    lnk[root][0].clear();
    lnk[root][1].clear();
    tree[root].clear();
    if(left == right)
    {
        vector<pt>::iterator it;
        it = lower_bound(pts.begin(), pts.end(), pt(xs[left], -INF));
        for(; it != pts.end() && cmp(it->x, xs[left]) == 0; ++it)
            tree[root].push_back(*it);
        return tree[root].size();
    }
    int mid = (left + right)/2, cl = 2*root + 1, cr = cl + 1;
    int sz1 = recurse(cl, left, mid);
    int sz2 = recurse(cr, mid + 1, right);
    lnk[root][0].reserve(sz1+sz2+1);
    lnk[root][1].reserve(sz1+sz2+1);
    tree[root].reserve(sz1+sz2);
    int l = 0, r = 0, llink = 0, rlink = 0;
    pt last;
    while (l < sz1 || r < sz2)
    {
        if (r == sz2 || (l < sz1 && compy(tree[cl][l], tree[cr][r])))
            tree[root].push_back(last = tree[cl][l++]);
        else
            tree[root].push_back(last = tree[cr][r++]);
        while (llink < sz1 && compy(tree[cl][llink], last))
            ++llink;
        while (rlink < sz2 && compy(tree[cr][rlink], last))
            ++rlink;
        lnk[root][0].push_back(llink);
        lnk[root][1].push_back(rlink);
    }
    lnk[root][0].push_back(tree[cl].size());
    lnk[root][1].push_back(tree[cr].size());
    return tree[root].size();
}

void build()
{
    sort(pts.begin(), pts.end());
    xs.clear();
    for (int i = 0; i < pts.size(); ++i)
        xs.push_back(pts[i].x);
    xs.erase(unique(xs.begin(), xs.end()), xs.end());
    recurse(0, 0, xs.size() - 1);
}
```

```
//query (0, 0, xs.size() - 1, x1, x2, y1, y2)
int query (int root, int l, int r, double x1, double x2, double y1, double
y2,
           int posl = -1, int posr = -1)
{
    if (root == 0 && posl == -1)
    {
        posl = lower_bound(tree[0].begin(), tree[0].end(), pt(x1, y1),
compy)
               - tree[0].begin();
        posr = upper_bound(tree[0].begin(), tree[0].end(), pt(x2, y2),
compy)
               - tree[0].begin();
    }
    if (posl == posr)
        return 0;
    if (x1 <= xs[l] && xs[r] <= x2)
        return posr - posl;
    int mid = (l+r)/2, ret = 0;
    if (cmp(x1, xs[mid]) <= 0)
        ret += query(2*root+1, l, mid, x1, x2, y1, y2,
                     lnk[root][0][posl], lnk[root][0][posr]);
    if (cmp(xs[mid+1], x2) <= 0)
        ret += query(2*root+2, mid+1, r, x1, x2, y1, y2,
                     lnk[root][1][posl], lnk[root][1][posr]);
    return ret;
}
```

6.9 TREAP

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <vector>
#include <queue>
#include <stack>
#include <map>
#include <set>
#include <iostream>
#include <algorithm>

using namespace std;

#define INF (1<<30)

/*

    TREAP

    Aplicacoes:

        Determinar de forma eficiente:

            -> O n-esimo menor elemento de um conjunto S (FindKth(n))
            -> O numero de elementos de um conjunto S menores do que x
(Count(x))

    Como chamar a funcao:

        1) Criar uma Treap:

            Treap t;

        2) Inserir um elemento de valor x na Treap:

            t.Insert(x);

        3) Remover um elemento de valor x na Treap:

            t.Delete(x);

        4) Consultar o i-esimo menor elemento do conjunto:

            t.FindKth(i); (0 <= i < n)

        5) Consultar o numero de elementos menores que x

            t.Count(x);

    Resultado da funcao:

        A funcao FindKth(i) retorna o valor do i-esimo
        menor elemento no conjunto.
```

```
        A funcao Count(x) retorna o numero de elementos
        menores do que x no conjunto.

    Complexidade do algoritmo:

        O(log n) para todas as operacoes

    Problemas resolvidos:

        ORDERSET (SPOJ)
        KFSTD (SPOJ)

    Adicionado por:

        Jorge Gabriel Siqueira

*/

struct Node {
    int key;
    int cnt;
    int priority;

    Node *left, *right;

    Node(){cnt = 0; priority = 0; left = right = NULL;}
    Node(int _key){cnt = 1; key = _key; priority = rand(); left = right =
NULL;}
    Node(int _key, int pr){cnt = 1; key = _key; priority = pr; left =
right = NULL;}
};

struct Treap {
    Node* root;

    Treap(){root = NULL; srand(time(NULL));}

    int TreeSize(Node* T)
    {
        return T==NULL?0:T->cnt;
    }

    void UpdateCnt(Node* &T)
    {
        if(T)
        {
            T->cnt = 1 + TreeSize(T->left) + TreeSize(T->right);
        }
    }

    void LeftRotate(Node* &T)
    {
        Node* temp = T->left;
        T->left = temp->right;
        temp->right = T;
        T = temp;

        UpdateCnt(T->right);
```

```
        UpdateCnt(T);
    }

    void RightRotate(Node* &T)
    {
        Node* temp = T->right;
        T->right = temp->left;
        temp->left = T;
        T = temp;

        UpdateCnt(T->left);
        UpdateCnt(T);
    }

    void Insert(Node* &T, int _key)
    {
        if(T == NULL)
        {
            T = new Node(_key);
            return;
        }

        if(T->key > _key)
        {
            Insert(T->left, _key);

            if(T->priority < T->left->priority)
                LeftRotate(T);
        }
        else if(T->key < _key)
        {
            Insert(T->right, _key);

            if(T->priority < T->right->priority)
                RightRotate(T);
        }

        UpdateCnt(T);
    }

    void Insert(int _key)
    {
        Insert(root, _key);
    }

    void Delete(Node* &T, int _key)
    {
        if(T == NULL)
            return;

        if(T->key > _key)
            Delete(T->left, _key);
        else if(T->key < _key)
            Delete(T->right, _key);
        else
        {
            if(T->left && T->right)
            {
```

```
                if(T->left->priority > T->right->priority)
                    LeftRotate(T);
                else
                    RightRotate(T);

                Delete(T, _key);
            }
            else
            {
                Node* temp = T;

                if(T->left)
                    T = T->left;
                else
                    T = T->right;

                delete temp;
            }
        }


        UpdateCnt(T);
    }

    void Delete(int _key)
    {
        Delete(root, _key);
    }

    int Count(Node* T, int bound)
    {
        if(T == NULL)
            return 0;

        if(T->key < bound)
            return 1 + TreeSize(T->left) + Count(T->right, bound);

        return Count(T->left, bound);
    }

    int Count(int bound)
    {
        return Count(root,bound);
    }

    int FindKth(Node* T, int k)
    {
        if(TreeSize(T) < k)
            return -INF;

        int sz = 1 + TreeSize(T->left);

        if(sz == k)
            return T->key;

        if(sz < k)
            return FindKth(T->right, k-sz);
```

```
        return FindKth(T->left,k);
    }

    int FindKth(int k)
    {
        return FindKth(root, k);
    }
};

int main()
{
    // freopen("in.txt","r",stdin);

    int Q; scanf("%d",&Q);

    Treap oTreap;

    while(Q--)
    {
        char t[5];
        int p;
        scanf("%s%d",t,&p);

        if(t[0]=='I')
        {
            oTreap.Insert(p);
        }
        else if(t[0]=='D')
        {
            oTreap.Delete(p);
        }
        else if(t[0]=='K')
        {
            int v = oTreap.FindKth(p);

            if(v > -INF)
            {
                printf("%d\n",v);
            }
            else
                puts("invalid");
        }
        else
        {
            int v = oTreap.Count(p);

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

    }

    return 0;
}
```

6.10 KD-TREE

```
#include <stdio.h>
#include <string.h>
#include <algorithm>

using namespace std;

/*

    KD-TREE

    Aplicacoes:

        Dado um conjunto S de pontos e um ponto P de
        coordenadas (x,y), determina de forma eficiente
        A distancia do ponto mais proximo de P do conjunto
        S (e que seja diferente de P)

    Como chamar a funcao:

        1) Armazenar em n o numero de pontos do conjunto

        2) Armazenar os pontos no vetor P[] a partir da
           posicao 0.

        3) Chamar a funcao initialize() para inicializar a
           arvore

           Obs.: A funcao destroi o vetor P[], se precisar manter
                 os dados salve em um vetor auxiliar.

        4) Chamar a funcao query(P) para consultar qual o
           ponto X do conjunto S esta mais proximo do
           ponto P. A funcao retorna um long long
           que eh a distancia entre P e X ao quadrado.

    Resultado da funcao:

        A funcao query(P) retorna retorna a distancia
        euclidiana minima de um ponto X do conjunto S
        ao ponto P.

        Para descobrir a distancia real, faca:

            double d = sqrt(query(P));

    Complexidade do algoritmo:

        O(log n) -> Query

    Problemas resolvidos:

        MLE (SPOJ)

    Adicionado por:

        Jorge Gabriel Siqueira
```

```
*/

#define MAXN 101000

struct ponto
{
    long long axis [2];
};

ponto P[MAXN]; //conjunto de pontos
ponto PNode[MAXN];
long long minDist;
int esq[2*MAXN], dir[2*MAXN], at;
int root; //raiz da arvore
int n; //numero de pontos do conjunto

bool cmpX (ponto p1, ponto p2)
{
    if (p1.axis[0] != p2.axis[0])
        return p1.axis[0] < p2.axis[0];
    return p1.axis[1] < p2.axis[1];
}

bool cmpY (ponto p1, ponto p2)
{
    if (p1.axis[1] != p2.axis[1])
        return p1.axis[1] < p2.axis[1];
    return p1.axis[0] < p2.axis[0];
}

int make_tree (int ini , int fim , int axis)
{
    if (ini + 1 == fim)
    {
        int root = at++;
        PNode[root] = P[ini];
        esq[root] = dir[root] = -1;
        return root;
    }
    else if (ini + 1 > fim)
    {
        return -1;
    }
    int root = at++;
    int meio = (ini + fim)/2;
    if (!axis)
    {
        nth_element(P + ini, P + meio, P + fim, cmpX);
    }
    else
    {
        nth_element(P + ini, P + meio, P + fim, cmpY);
    }
    PNode[root] = P[meio];
    esq[root] = make_tree(ini, meio, 1 - axis);
    dir[root] = make_tree(meio + 1, fim, 1 - axis);
    return root;
```

```
}

void initialize ()
{
    at = 0;
    root = make_tree(0, n, 0);
}

long long dist (ponto p1, ponto p2, int axis = -1)
{
    if (axis == -1)
    {
        return (p1.axis[0] - p2.axis[0]) * (p1.axis[0] - p2.axis[0]) +
               (p1.axis[1] - p2.axis[1]) * (p1.axis[1] - p2.axis[1]);
    }
    else
    {
        return (p1.axis[axis] - p2.axis[axis])*(p1.axis[axis] -
p2.axis[axis]);
    }
}

void query (ponto p, int root, int axis = 0)
{
    if (root == -1)
        return;
    long long d = dist (p, PNode[root]);
    if (d != 0LL)
    {
        minDist = (minDist != -1LL) ? min(minDist, d) : d;
    }
    if (p.axis[axis] < PNode[root].axis[axis])
    {
        query(p, esq[root], 1 - axis);
        if (dist(p, PNode[root], axis) <= minDist)
        {
            query(p, dir[root], 1-axis);
        }
    }
    else
    {
        query(p, dir[root], 1 - axis);
        if (dist(p, PNode[root], axis) <= minDist)
        {
            query(p, esq[root], 1-axis);
        }
    }
}

long long query (ponto p)
{
    minDist = -1LL;
    query (p, root, 0);
    return minDist;
}
```

6.11 TRIE

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream>
#include <sstream>

using namespace std;

/*

    TRIE

    Aplicacoes:

        A arvore trie pode substituir uma tabela hash.
        Eh comumente usada para localizar uma palavra em
        um dicionario.

    Como chamar a funcao:

        1) Criar uma Trie:

            Trie t;

        2) Inserir palavras no dicionario da Trie:

            insert("word", &t);

        3) Para verificar se uma palavra encontra-se
           no dicionario chamar:

                find("word",&t);

    Resultado da funcao:

        A funcao find() retorna:

            true -> caso a palavra esteja no dicionario
            false -> caso contrario

    Complexidade do algoritmo:

        O(m) no pior caso, onde m eh o tamanho da maior
        palavra do dicionario.

    Problemas resolvidos:

        URI 1284

    Adicionado por:

        Jorge Gabriel Siqueira

*/
```

```
struct Trie
{
    bool end;

    Trie *next[0x100];

    Trie()
    {
        end = false;
        fill(next, next+0x100, (Trie*)0);
    }
};

void insert(char *t, Trie *r)
{
    for (int i = 0; t[i]; ++i)
    {
        char c = t[i];

        if (!r->next[c])
        {
            r -> next[c] = new Trie;
        }

        r = r->next[c];
    }

    r -> end = true;
}

bool find(char *t, Trie *r)
{
    for (int i = 0; t[i]; ++i)
    {
        char c = t[i];

        if (!r->next[c]) return false;

        r = r->next[c];
    }

    return r -> end;
}

int main()
{
    Trie t;
    insert("jorge1",&t);
    cout << find("jorge",&t);
    return 0;
}
```

6.12 SEGMENT TREE CLASSIC

```
#include <cstdio>
#include <string>
#include <cstring>
#include <vector>
#include <algorithm>
#include <map>
#include <utility>
#include <cmath>
#include <queue>
#include <stack>
#include <set>
#include <deque>
#include <iostream>
#include <math.h>
#include <sstream>
#include <assert.h>
#include <numeric>
#include <fstream>
#include <limits>
#include <bitset>
#define INF 0x3f3f3f3f
#define MAX 1000000

using namespace std;

/*

    CLASSIC SEGMENT TREE

    Aplicacoes:

        Queries de minimo valor de elemento em sub-intervalo de um
conjunto.

    Como chamar a funcao:

        1) Armazenar em n o numero de elementos do conjunto

        2) Armazenar no vetor int array[] os elementos do conjunto

        3) Montar a Segment Tree:

            buildSegTree();

        4) Para updates:

            update(pos,val);

            para a posicao "pos" passar a ter o valor "val"

        5) Para queries:

            query(i,j);

    Resultado da funcao:
```

```
        query(i,j) -> retorna o valor do minimo elemento entre os
                      elementos de i a j do conjunto.

    Complexidade do algoritmo:

        update() , query() -> O( log(n) )
        buildSegTree() -> O( n * log(n) )

    Problemas resolvidos:

        URI 1301

    Adicionado por:

        Jorge Gabriel Siqueira

*/

typedef struct node
{
    node *pred;
    node *left, *right;
    int value;
} node;

node *segTree;
node *cacheLeaf[MAX];
int array[MAX], n;

int buildSegTreeRec(node *segment, int l, int r)
{
    if(l == r)
    {
        cacheLeaf[l] = segment;
        return segment -> value = array[l];
    }
    else
    {
        segment -> left = (node *)malloc(sizeof(node));
        segment -> right = (node *)malloc(sizeof(node));

        segment ->  left -> pred = segment;
        segment ->  right -> pred = segment;

        segment ->  left -> right = NULL;
        segment ->  left -> left = NULL;
        segment ->  right -> right = NULL;
        segment ->  right -> left = NULL;

        int left = buildSegTreeRec(segment ->  left, l, (l+r)/2);
        int right = buildSegTreeRec(segment ->  right, (l+r)/2+1, r);

        return segment -> value = min(left, right);
    }
}

void buildSegTree()
{
```

```
    segTree = (node *)malloc(sizeof(node));
    segTree ->  pred = NULL;
    segTree ->  left = NULL;
    segTree ->  right = NULL;
    buildSegTreeRec(segTree, 0, n-1);
}

int queryRec(node *segment, int &l, int &r, int i, int j)
{
    if((i >= l && i <= r && j >= l && j <= r))
    {
        return segment -> value;
    }
    else
    {
        int a = (i+j)/2;
        int b = a+1;
        int p1, p2;
        p1 = p2 = INF;

        if(i >= l && i <= r || a >= l && a <= r || (i < l && a > r)) p1 =
queryRec(segment -> left, l, r, i, a);
        if(b >= l && b <= r || j >= l && j <= r || (b < l && j > r)) p2 =
queryRec(segment -> right, l, r, b, j);

        return min(p1,p2);
    }
}

inline int query(int l, int r)
{
    return queryRec(segTree, l, r, 0, n-1);
}

void update(int p, int val)
{
    int left, right;
    node *segment = cacheLeaf[p];
    segment -> value = val;
    segment = segment -> pred;
    while(segment)
    {
        left = right = INF;
        if(segment -> left) left = segment -> left -> value;
        if(segment -> right) right = segment -> right -> value;
        segment -> value = min(left, right);
        segment = segment -> pred;
    }
    array[p] = val;
}
```

6.13 SEGMENT TREE TO INTERVAL PRODUCT

```
#include <cstdio>
#include <string>
#include <cstring>
#include <vector>
#include <algorithm>
#include <map>
#include <utility>
#include <cmath>
#include <queue>
#include <stack>
#include <set>
#include <deque>
#include <iostream>
#include <math.h>
#include <sstream>
#include <assert.h>
#include <numeric>
#include <fstream>
#include <limits>
#include <bitset>
#define INF 0x3f3f3f3f
#define MAX 1000000

using namespace std;

/*

    SEGMENT TREE ADAPTED TO INTERVAL PRODUCT

    Aplicacoes:

        Queries de produto de sub-intervalo de um conjunto.

    Como chamar a funcao:

        1) Armazenar em n o numero de elementos do conjunto

        2) Armazenar no vetor int array[] os elementos do conjunto

        3) Montar a Segment Tree:

            buildSegTree();

        4) Para updates:

            update(pos,val);

            para a posicao "pos" passar a ter o valor "val"

        5) Para queries:

            query(i,j);

    Resultado da funcao:

        query(i,j) -> retorna o valor do produto dos elementos
```

```
                        de i a j do conjunto.

    Complexidade do algoritmo:

        update() , query() -> O( log(n) )
        buildSegTree() -> O( n * log(n) )

    Problemas resolvidos:

        URI 1301

    Adicionado por:

        Jorge Gabriel Siqueira

*/

typedef struct node
{
    node *pred;
    node *left, *right;
    int value;
} node;

node *segTree;
node *cacheLeaf[MAX];
int array[MAX], n;

int buildSegTreeRec(node *segment, int l, int r)
{
    if(l == r)
    {
        cacheLeaf[l] = segment;
        return segment -> value = array[l];
    }
    else
    {
        segment -> left = (node *)malloc(sizeof(node));
        segment -> right = (node *)malloc(sizeof(node));

        segment ->  left -> pred = segment;
        segment ->  right -> pred = segment;

        segment ->  left -> right = NULL;
        segment ->  left -> left = NULL;
        segment ->  right -> right = NULL;
        segment ->  right -> left = NULL;

        int left = buildSegTreeRec(segment ->  left, l, (l+r)/2);
        int right = buildSegTreeRec(segment ->  right, (l+r)/2+1, r);

        return segment -> value = left*right;
    }
}

void buildSegTree()
{
    segTree = (node *)malloc(sizeof(node));
```

```
    segTree ->  pred = NULL;
    segTree ->  left = NULL;
    segTree ->  right = NULL;
    buildSegTreeRec(segTree, 0, n-1);
}

int queryRec(node *segment, int &l, int &r, int i, int j)
{
    if((i == j) || (i >= l && i <= r && j >= l && j <= r))
    {
        return segment -> value;
    }
    else
    {
        int a = (i+j)/2;
        int b = a+1;
        int p1, p2;
        p1 = p2 = INF;

        if(i >= l && i <= r || a >= l && a <= r || (i < l && a > r)) p1 =
queryRec(segment -> left, l, r, i, a);
        if(b >= l && b <= r || j >= l && j <= r || (b < l && j > r)) p2 =
queryRec(segment -> right, l, r, b, j);

        if(p1 == INF && p2 == INF) return 1;
        else if(p1 == INF) return p2;
        else if(p2 == INF) return p1;
        else return p1*p2;
    }
}

inline int query(int l, int r)
{
    return queryRec(segTree, l, r, 0, n-1);
}

void update(int p, int val)
{
    int left, right;
    node *segment = cacheLeaf[p];
    int den = segment -> value;
    segment -> value = val;
    segment = segment -> pred;
    while(segment)
    {
        left = right = 1;
        if(segment -> left) left = segment -> left -> value;
        if(segment -> right) right = segment -> right -> value;
        segment -> value = left*right;
        segment = segment -> pred;
    }
    array[p] = val;
}
```

6.14 SEGMENT TREE WITH LAZY PROPAGATION

```
#include <cstdio>
#include <string>
#include <cstring>
#include <vector>
#include <algorithm>
#include <map>
#include <utility>
#include <cmath>
#include <queue>
#include <stack>
#include <set>
#include <deque>
#include <iostream>
#include <math.h>
#include <sstream>
#include <assert.h>
#include <numeric>
#include <fstream>
#include <limits>
#include <time.h>
#include <bitset>
#define INF 0x3f3f3f3f
#define MAX 1000000
#define NOLAZY INF

using namespace std;

/*

    SEGMENT TREE WITH LAZY PROPAGATION

    Aplicacoes:

        Queries de minimo valor de elemento em sub-intervalo de um
conjunto.

    Como chamar a funcao:

        1) Armazenar em n o numero de elementos do cnojunto

        2) Armazenar no vetor int array[] os elementos do conjunto

        3) Montar a Segment Tree:

            buildSegTree();

        4) Para updates unitarios:

            update(pos,val);

            para a posicao "pos" passar a ter o valor "val"

        5) Para updates de intervalos:

            update(ini,fim,val);
```

```
            para todos os elementos do intervalo [ini,fim] passarem
            a possuir o valor "val"

        5) Para queries:

            query(i,j);

    Resultado da funcao:

        query(i,j) -> retorna o valor do minimo elemento entre os
                      elementos de i a j do conjunto.

    Complexidade do algoritmo:

        update() , query() -> O( log(n) )
        queryInterval() -> Lazy propagation
        buildSegTree() -> O( n * log(n) )

    Problemas resolvidos:

    Adicionado por:

        Jorge Gabriel Siqueira

*/

typedef struct node
{
    node *pred;
    node *left, *right;
    int value;
    int lazy;
} node;

node *segTree;
node *cacheLeaf[MAX];
int array[MAX], n;

int buildSegTreeRec(node *segment, int l, int r)
{
    if(l == r)
    {
        cacheLeaf[l] = segment;
        segment -> lazy = NOLAZY;
        return segment -> value = array[l];
    }
    else
    {
        segment -> left = (node *)malloc(sizeof(node));
        segment -> right = (node *)malloc(sizeof(node));

        segment ->  left -> pred = segment;
        segment ->  right -> pred = segment;

        segment -> left -> lazy = NOLAZY;
        segment -> right -> lazy = NOLAZY;

        segment ->  left -> right = NULL;
```

```
        segment ->  left -> left = NULL;
        segment ->  right -> right = NULL;
        segment ->  right -> left = NULL;

        int left = buildSegTreeRec(segment ->  left, l, (l+r)/2);
        int right = buildSegTreeRec(segment ->  right, (l+r)/2+1, r);
        segment -> lazy = NOLAZY;

        return segment -> value = min(left, right);
    }
}

void buildSegTree()
{
    segTree = (node *)malloc(sizeof(node));
    segTree ->  pred = NULL;
    segTree ->  left = NULL;
    segTree ->  right = NULL;
    segTree -> lazy = NOLAZY;
    buildSegTreeRec(segTree, 0, n-1);
}

int queryRec(node *segment, int &l, int &r, int lazy, int i, int j)
{
    if(i >= l && i <= r && j >= l && j <= r) //[l,r] is totaly inside de
interval [i,j]
    {
        return lazy != NOLAZY ? (segment -> lazy = segment -> value =
lazy) : (segment -> value);
    }
    else if((i < l && j < l) || (i > r && j > r)) //[l,r] is totaly
outside de interval [i,j]
    {
        int newLazy = segment -> pred -> lazy;
        if(newLazy != NOLAZY) segment -> lazy = segment -> value =
newLazy;
        return INF;
    }
    else
    {
        int a = (i+j)/2;
        int b = a+1;
        int newLazy = (lazy != NOLAZY) ? lazy : segment -> lazy;
        segment -> lazy = newLazy;
        if(newLazy != NOLAZY) segment -> value = newLazy;
        int left = queryRec(segment -> left, l, r, newLazy, i, a);
        int right = queryRec(segment -> right, l, r, newLazy, b, j);
        segment -> lazy = NOLAZY;
        return min(left, right);
    }
}

inline int query(int l, int r)
{
    return queryRec(segTree, l, r, segTree -> lazy, 0, n-1);
}

void update(int p, int val)
```

```
{
    int left, right;
    node *segment = cacheLeaf[p];
    segment -> value = val;
    segment = segment -> pred;
    while(segment)
    {
        left = right = INF;
        if(segment -> left) left = segment -> left -> value;
        if(segment -> right) right = segment -> right -> value;
        segment -> value = min(left, right);
        segment = segment -> pred;
    }
    array[p] = val;
}

int updateIntervalRec(node *segment, int &l, int &r, int lazy, int i, int
j)
{
    if(i >= l && i <= r && j >= l && j <= r) //[l,r] is totaly inside de
interval [i,j]
    {
        return segment -> lazy = segment -> value = lazy;
    }
    else if((i < l && j < l) || (i > r && j > r)) //[l,r] is totaly
outside de interval [i,j]
    {
        int newLazy = segment -> pred -> lazy;
        if(newLazy != NOLAZY) segment -> lazy = segment -> value =
newLazy;
        return segment -> value;
    }
    else
    {
        int a = (i+j)/2;
        int b = a+1;

        if(!(i==0 && j==n-1))
        {
            int newLazy = segment -> pred -> lazy;
            if(newLazy != NOLAZY) segment -> lazy = newLazy;
        }

        int left = updateIntervalRec(segment -> left, l, r, lazy, i, a);
        int right = updateIntervalRec(segment -> right, l, r, lazy, b, j);

        segment -> lazy = NOLAZY;
        return segment -> value = min(left, right);
    }
}

inline void updateInterval(int l, int r, int val)
{
    updateIntervalRec(segTree, l, r, val, 0, n-1);
    for(int i=l; i<=r; i++) array[i] = val;
}
```

6.15 ÁRVORE BINÁRIADE BUSCA

```
#include <cstdio>
#include <string>
#include <cstring>
#include <vector>
#include <algorithm>
#include <map>
#include <utility>
#include <cmath>
#include <queue>
#include <stack>
#include <set>
#include <deque>
#include <iostream>
#include <math.h>
#include <sstream>
#include <assert.h>
#define INF 0x3f3f3f3f

using namespace std;

/*

    ABB - Arvore Binaria de Busca

    Aplicacoes:

        Busca em O(log(n)) em um conjunto de numeros
        Percurso In,Pre e Pos ordem
        Contar os nos de uma arvore
        Contar as folhas de uma arvore
        Calcular a altura de uma arvore

    Como chamar a funcao:

        1) Criar a arvore:

            No *arvore;
            criarArvore(&arvore);

        2) Inserir um elemento:

            int numero;
            scanf("%d",&numero);
            inserir(&arvore,numero);

        3) Remover um elemento:

            int numero;
            scanf("%d",&numero);
            remover(&arvore,numero);

        4) Exibir os percursos:

            exibirPreOrdem(arvore);
            exibirEmOrdem(arvore);
            exibirPosOrdem(arvore);
```

```
        5) Contar nos, folhas, verificar a altura:

            contarNos(arvore);
            contarFolhas(arvore);
            altura(arvore);

    Resultado da funcao:

        1,2) void

        3) void, caso nao exista o elemento a ser removido eh
           possivel tratar esta situacao (primeira linha da funcao)

        4) Imprime o percurso escolhido

        5) Retorna um inteiro representando o valor da
           funcao selecionada

    Complexidade do algoritmo:

        1) O(1)

        2,3) O(log(n))

        4,5) O(n)

    Problemas resolvidos:

        URI 1200
        URI 1201

    Adicionado por:

        Jorge Gabriel Siqueira

*/

typedef struct No
{
    int numero;
    struct No *esquerda;
    struct No *direita;
} No;

void criarArvore(No **pRaiz)
{
    *pRaiz = NULL;
}

void inserir(No **pRaiz, int numero)
{
    if(*pRaiz == NULL)
    {
        *pRaiz = (No *) malloc(sizeof(No));
        (*pRaiz)->esquerda = NULL;
        (*pRaiz)->direita = NULL;
        (*pRaiz)->numero = numero;
```

```
    }
    else
    {
        if(numero < (*pRaiz)->numero)
            inserir(&(*pRaiz)->esquerda, numero);
        if(numero > (*pRaiz)->numero)
            inserir(&(*pRaiz)->direita, numero);
    }
}

No *MaiorDireita(No **no)
{
    if((*no)->direita != NULL)
        return MaiorDireita(&(*no)->direita);
    else
    {
        No *aux = *no;
        if((*no)->esquerda != NULL) // se nao houver essa verificacao,
esse no vai perder todos os seus filhos da esquerda!
            *no = (*no)->esquerda;
        else
            *no = NULL;
        return aux;
    }
}

No *MenorEsquerda(No **no)
{
    if((*no)->esquerda != NULL)
        return MenorEsquerda(&(*no)->esquerda);
    else
    {
        No *aux = *no;
        if((*no)->direita != NULL) // se nao houver essa verificacao, esse
no vai perder todos os seus filhos da direita!
            *no = (*no)->direita;
        else
            *no = NULL;
        return aux;
    }
}

void remover(No **pRaiz, int numero)
{
    if(*pRaiz == NULL)    // esta verificacao serve para caso o numero nao
exista na arvore.
    {
        printf("Numero nao existe na arvore!\n");
        return;
    }
    if(numero < (*pRaiz)->numero)
        remover(&(*pRaiz)->esquerda, numero);
    else if(numero > (*pRaiz)->numero)
        remover(&(*pRaiz)->direita, numero);
    else     // se nao eh menor nem maior, logo, eh o numero que estou
procurando! :)
    {
        No *pAux = *pRaiz;     // quem programar no Embarcadero vai ter
```

```
que declarar o pAux no inicio do void! :[
        if (((*pRaiz)->esquerda == NULL) && ((*pRaiz)->direita == NULL))
// se nao houver filhos...
        {
            free(pAux);
            (*pRaiz) = NULL;
        }
        else      // so tem o filho da direita
        {
            if ((*pRaiz)->esquerda == NULL)
            {
                (*pRaiz) = (*pRaiz)->direita;
                pAux->direita = NULL;
                free(pAux);
                pAux = NULL;
            }
            else             //so tem filho da esquerda
            {
                if ((*pRaiz)->direita == NULL)
                {
                    (*pRaiz) = (*pRaiz)->esquerda;
                    pAux->esquerda = NULL;
                    free(pAux);
                    pAux = NULL;
                }
                else        //Escolhi fazer o maior filho direito da
subarvore esquerda.
                {
                    pAux = MaiorDireita(&(*pRaiz)->esquerda); //se vc
quiser usar o Menor da esquerda, so o que mudaria seria isso:
                    pAux->esquerda = (*pRaiz)->esquerda;          //
pAux = MenorEsquerda(&(*pRaiz)->direita);
                    pAux->direita = (*pRaiz)->direita;
                    (*pRaiz)->esquerda = (*pRaiz)->direita = NULL;
                    free((*pRaiz));
                    *pRaiz = pAux;
                    pAux = NULL;
                }
            }
        }
    }
}

void exibirEmOrdem(No *pRaiz)
{
    if(pRaiz != NULL)
    {
        exibirEmOrdem(pRaiz->esquerda);
        printf("\n%i", pRaiz->numero);
        exibirEmOrdem(pRaiz->direita);
    }
}

void exibirPreOrdem(No *pRaiz)
{
    if(pRaiz != NULL)
    {
        printf("\n%i", pRaiz->numero);
```

```
        exibirPreOrdem(pRaiz->esquerda);
        exibirPreOrdem(pRaiz->direita);
    }
}

void exibirPosOrdem(No *pRaiz)
{
    if(pRaiz != NULL)
    {
        exibirPosOrdem(pRaiz->esquerda);
        exibirPosOrdem(pRaiz->direita);
        printf("\n%i", pRaiz->numero);
    }
}

int contarNos(No *pRaiz)
{
    if(pRaiz == NULL)
        return 0;
    else
        return 1 + contarNos(pRaiz->esquerda) + contarNos(pRaiz->direita);
}

int contarFolhas(No *pRaiz)
{
    if(pRaiz == NULL)
        return 0;
    if(pRaiz->esquerda == NULL && pRaiz->direita == NULL)
        return 1;
    return contarFolhas(pRaiz->esquerda) + contarFolhas(pRaiz->direita);
}

int maior(int a, int b)
{
    if(a > b)
        return a;
    else
        return b;
}

int altura(No *pRaiz)
{
    if((pRaiz == NULL) || (pRaiz->esquerda == NULL && pRaiz->direita ==
NULL))
        return 0;
    else
        return 1 + maior(altura(pRaiz->esquerda), altura(pRaiz->direita));
}

int busca (No *inicio, int valor)
{
    while (inicio != NULL)
    {
        if (valor == inicio->numero)
            return inicio->numero;
        else if (valor < inicio->numero)
            inicio = inicio->esquerda;
        else
```

```
            inicio = inicio->direita;
    }
    return 0;
}
```

7. GEOMETRIA 2D

7.1 TEMPLATE

```
#include <cstdio>
#include <string>
#include <cstring>
#include <vector>
#include <algorithm>
#include <map>
#include <utility>
#include <cmath>
#include <queue>
#include <stack>
#include <set>
#include <deque>
#include <iostream>
#include <sstream>
#define REP(i,n) for(int i=0;i<(int)n;++i)
#define EACH(i,c) for(__typeof((c).begin()) i=(c).begin(); i!=(c).end(); ++i)
#define ALL(c) (c).begin(), (c).end()
#define SIZE(x) (int((x).size()))
#define MAXSZ 1000
using namespace std;
const int INF = 0x3F3F3F3F;
const double PI = 2*acos(0);
const double EPS = 1e-10;

/*
    Função de Comparação de 2 valores

    Parametros:
        double x;
        double y;

    Retorna:
        -1 se x <= y
        0 se x == y
        1 se x >= y

*/

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

```
/*
    Estrutura de dados para pontos
*/
struct point
{
    double x, y;
    point(double x = 0, double y = 0): x(x), y(y) {}
    point operator +(point q) { return point(x + q.x, y + q.y); }
    point operator -(point q) { return point(x - q.x, y - q.y); }
    point operator *(double t) { return point(x * t, y * t); }
    point operator /(double t) { return point(x / t, y / t); }
    double operator *(point q) { return x * q.x + y * q.y; }
    double operator %(point q) { return x * q.y - y * q.x; }
    int cmp(point q) const {
    if (int t = ::cmp(x, q.x)) return t;
    return ::cmp(y, q.y);
    }
    bool operator ==(point q) const { return cmp(q) == 0; }
    bool operator !=(point q) const { return cmp(q) != 0; }
    bool operator < (point q) const { return cmp(q) < 0; }
    bool operator <=(point q) const { return cmp(q)<=0; }
    friend ostream& operator <<(ostream& o, point p) {
    return o << "(" << p.x << ", " << p.y << ")";
    }
    static point pivot;
};

point point::pivot;

double abs(point p) { return hypot(p.x, p.y); }
double arg(point p) { return atan2(p.y, p.x); }

typedef vector<point> polygon;
typedef pair<point, double> circle;

int ccw(point p, point q, point r)
{
    return cmp((p - r) % (q - r));
}
```

7.2 ÂNGULO ENTRE DOIS SEGMENTO DE RETAS

```
#include "head.h"

/*

    Aplicações:

        Calcula o angulo em Radianos entre os segmentos qp e qr

*/

double angle(point p, point q, point r)
{
    point u = p - q, v = r - q;
    return atan2(u % v, u * v);
}

double dist(point a, point b)
{
    point c = a-b;
    return sqrt(c.x*c.x+c.y*c.y);
}
```

7.3 CLASSIFICAÇÃO DE PONTO EMRELAÇÃO A RETA

```
#include "head.h"

/*
    Aplicações:

        Decide se um ponto p esta sobre um segmento de reta pr

    Parametros

        point p -> ponto inicial da reta pq
        point q -> ponto a ser analisado
        point r -> ponto final da reta pq

    Devolve

        true -> se q esta sobrea reta pq
        false -> se q nao esta sobre a reta pq

*/
bool between(point p, point q, point r)
{
    return ccw(p, q, r) == 0 && cmp((p - q) * (r - q)) <= 0;
}
```

7.4 INTERSECÇÃO DE SEGMENTOS

```
#include "head.h"

/*
    Aplicações:

        Decide se os segmentos fechados pq e rs têm pontos em comum.

    Parametros

        point p -> ponto inicial da reta pq
        point q -> ponto final da reta pq
        point r -> ponto inicial da reta rs
        point s -> ponto final da reta rs

    Devolve

        true -> se os segmentos de retas pq e rs se cruzam
        false -> se os segmentos de retas pq e rs nao se cruzam

*/
bool seg_intersect(point p, point q, point r, point s)
{
    point A = q - p, B = s - r, C = r - p, D = s - q;
    int a = cmp(A % C) + 2 * cmp(A % D);
    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;
}
```

7.5 DISTÂNCIA DE PONTO A SEGMENTO DE RETA

```c
#include "head.h"

/*
    Aplicações:

        Calcula a distância do ponto r ao segmento pq.

    Parametros

        point p -> ponto inicial da reta pq
        point q -> ponto final da reta pq
        point r -> ponto a ser analisado

    Devolve

        true -> se os segmentos de retas pq e rs se cruzam
        false -> se os segmentos de retas pq e rs nao se cruzam

*/

double seg_distance(point p, point q, point r)
{
    point A = r - q, B = r - p, C = q - p;
    double a = A * A, b = B * B, c = C * C;
    if (cmp(b, a + c) >= 0) return sqrt(a);
    else if (cmp(a, b + c) >= 0) return sqrt(b);
    else return fabs(A % B) / sqrt(c);
}
```

7.6 CLASSIFICAÇÃODE PONTO EM RELAÇÃO A POLIGONO

```
#include "head.h"

/*
    Aplicações:

        Classifica o ponto p em relação ao polígono T.

    Parametros

        point p -> ponto a ser analisado
        polygon T -> poligono a ser analisado

    Devolve

        0 -> p esta no exterior do poligono T
        1 -> p esta na fronteira de T
        -1 -> p esta no interior de T

*/
bool between(point p, point q, point r)
{
    return ccw(p, q, r) == 0 && cmp((p - q) * (r - q)) <= 0;
}

double angle(point p, point q, point r)
{
    point u = p - q, v = r - q;
    return atan2(u % v, u * v);
}

int in_poly(point p, polygon& T)
{
    double a = 0; int N = T.size();

    for (int i = 0; i < N; i++)
    {
        if (between(T[i], p, T[(i+1) % N])) return -1;
        a += angle(T[i], p, T[(i+1) % N]);
    }

    return cmp(a) != 0;
}
```

7.7 FECHO CONVEXO DE UM CONJUNTO DE PONTOS

```
#include "head.h"

/*

    Aplicações:

        Determina o fecho convexo de um conjunto de pontos

    Parametros:

        vector<point> pts -> pontos a serem analisados

    Devolve

        vector<point> pts -> devolve o fecho convexo no proprio container pts

    ATENCAO: A funcao destroi o conjunto de pontos pts
             deixando somente o fecho convexo

*/

#define REMOVE_REDUNDANT

double cross(point p, point q)
{
    return p.x*q.y-p.y*q.x;
}
double area2(point a, point b, point c)
{
    return cross(a,b) + cross(b,c) + cross(c,a);
}

#ifdef REMOVE_REDUNDANT
bool bet(const point &a, const point &b, const point &c)
{
    //Para considerar pontos colineares troque o <= por <
    return (fabs(area2(a,b,c)) < EPS && (a.x-b.x)*(c.x-b.x) <= 0 && (a.y-
b.y)*(c.y-b.y) <= 0);
}
#endif

void ConvexHull(vector<point> &pts)
{
    sort(pts.begin(), pts.end());
```

```
    pts.erase(unique(pts.begin(), pts.end()), pts.end());

    vector<point> up, dn;

    for (int i = 0; i < pts.size(); i++)
    {
        //Para considerar pontos colineares troque o >= por >
        while (up.size() > 1 && area2(up[up.size()-2], up.back(), pts[i])
>= 0) up.pop_back();

        //Para considerar pontos colineares troque o <= por <
        while (dn.size() > 1 && area2(dn[dn.size()-2], dn.back(), pts[i])
<= 0) dn.pop_back();

        up.push_back(pts[i]);

        dn.push_back(pts[i]);
    }

    pts = dn;

    for (int i = (int) up.size() - 2; i >= 1; i--) pts.push_back(up[i]);

#ifdef REMOVE_REDUNDANT

    if (pts.size() <= 2) return;

    dn.clear();

    dn.push_back(pts[0]);
    dn.push_back(pts[1]);

    for (int i = 2; i < pts.size(); i++)
    {
        if (bet(dn[dn.size()-2], dn[dn.size()-1], pts[i])) dn.pop_back();
        dn.push_back(pts[i]);
    }

    if (dn.size() >= 3 && bet(dn.back(), dn[0], dn[1]))
    {
        dn[0] = dn.back();
        dn.pop_back();
    }

    pts = dn;
```

```
#endif
}

//Função alternativa - PUC-RIO

bool radial_lt(point p, point q)
{
    point P = p - point::pivot, Q = q - point::pivot;
    double R = P % Q;
    if (cmp(R)) return R > 0;
    return cmp(P * P, Q * Q) < 0;
}

polygon convex_hull(vector<point>& T)
{
    int j = 0, k, n = T.size();
    polygon U(n);
    point::pivot = *min_element(ALL(T));
    sort(ALL(T), radial_lt);
    for (k = n-2; k >= 0 && ccw(T[0], T[n-1], T[k]) == 0; k--);
    reverse((k+1) + ALL(T));
    for (int i = 0; i < n; i++)
    {
        // troque o >= por > para manter pontos colineares
        while (j > 1 && ccw(U[j-1], U[j-2], T[i]) >= 0) j--;
        U[j++] = T[i];
    }
    U.erase(j + ALL(U));
    return U;
}

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

7.8 ÁREA DE POLIGONO

```
#include "head.h"

/*
    Aplicações:

        Retorna a area orientada de um poligono T

    Parametros

        polygon T -> Poligono a ser analisado

    Devolve

        double area -> Area orientada do poligono T

*/
double poly_area(polygon& T)
{
    double s = 0; int n = T.size();
    for (int i = 0; i < n; i++) s += T[i] % T[(i+1) % n];
    return s / 2;
}


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

7.9 PONTO DE INTERSECÇÃO ENTRE SEGMENTOS

```
#include "head.h"

/*

    Aplicações:

        Encontra o ponto de interseção das retas pq e rs.

    Parametros

        point p -> ponto inicial da reta pq
        point q -> ponto final da reta pq
        point r -> ponto inicial da reta rs
        point s -> ponto final da reta rs

    Devolve

        point p -> ponto de intersecao das retas pq e rs

*/
point line_intersect(point p, point q, point r, point s)
{
    point a = q - p, b = s - r, c = point(p % q, r % s);
    return point(point(a.x, b.x) % c, point(a.y, b.y) % c) / (a % b);
}

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

7.10 SPANNING CIRCLE

```
#include "head.h"

/*
    Aplicações:

        Encontra o menor círculo que contém todos os pontos dados.

    Parametros

        point p -> ponto inicial da reta pq
        point q -> ponto final da reta pq
        point r -> ponto inicial da reta rs
        point s -> ponto final da reta rs

    Devolve

        circle c -> menor circulo que contém todos os pontos dados.

*/


bool in_circle(circle C, point p)
{
    return cmp(abs(p - C.first), C.second) <= 0;
}

point circumcenter(point p, point q, point r)
{
    point a = p - r, b = q - r, c = point(a * (p + r) / 2, b * (q + r) / 2);
    return point(c % point(a.y, b.y), point(a.x, b.x) % c) / (a % b);
}

circle spanning_circle(vector<point>& T)
{
    int n = T.size();

    random_shuffle(ALL(T));

    circle C(point(), -INFINITY);

    for (int i = 0; i < n; i++) if (!in_circle(C, T[i]))
    {
        C = circle(T[i], 0);

        for (int j = 0; j < i; j++) if (!in_circle(C, T[j]))
```

```
        {
            C = circle((T[i] + T[j]) / 2, abs(T[i] - T[j]) / 2);

            for (int k = 0; k < j; k++) if (!in_circle(C, T[k]))
            {
                point o = circumcenter(T[i], T[j], T[k]);
                C = circle(o, abs(o - T[k]));
            }
        }
    }

    return C;
}
```

7.11 POLIGONO DE INTERSECÇÃO ENTRE DOIS POLIGONOS

```
#include "head.h"

bool between(point p, point q, point r)
{
    return ccw(p, q, r) == 0 && cmp((p - q) * (r - q)) <= 0;
}

bool seg_intersect(point p, point q, point r, point s)
{
    point A = q - p, B = s - r, C = r - p, D = s - q;

    int a = cmp(A % C) + 2 * cmp(A % D);

    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;
}

double angle(point p, point q, point r)
{
    point u = p - q, v = r - q;
    return atan2(u % v, u * v);
}

int in_poly(point p, polygon& T)
{
    double a = 0; int N = T.size();

    for (int i = 0; i < N; i++)
    {
        if (between(T[i], p, T[(i+1) % N])) return -1;
        a += angle(T[i], p, T[(i+1) % N]);
    }

    return cmp(a) != 0;
}

point line_intersect(point p, point q, point r, point s)
```

```
{
    point a = q - p, b = s - r, c = point(p % q, r % s);
    return point(point(a.x, b.x) % c, point(a.y, b.y) % c) / (a % b);
}

/*
    Aplicações:

        Determina o polígono interseção dos dois polígonos convexos P e Q.

    Parametros

        polygon P;
        polygon Q;

    Devolve

        polygon R -> Poligono formado pela interseção dos dois polígonos convexos P e Q

    ATENCAO: Tanto P quanto Q devem estar orientados positivamente.

*/

polygon poly_intersect(polygon& P, polygon& Q)
{
    int m = Q.size(), n = P.size();

    int a = 0, b = 0, aa = 0, ba = 0, inflag = 0;

    polygon R;

    while ((aa < n || ba < m) && aa < 2*n && ba < 2*m)
    {
        point p1 = P[a], p2 = P[(a+1) % n], q1 = Q[b], q2 = Q[(b+1) % m];

        point 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 (between(p1, q1, p2)) R.push_back(q1);
            if (between(p1, q2, p2)) R.push_back(q2);
            if (between(q1, p1, q2)) R.push_back(p1);
            if (between(q1, p2, q2)) R.push_back(p2);
            if (R.size() < 2) return polygon();
```

```
            inflag = 1; break;
        }
        else if (cross != 0 && seg_intersect(p1, p2, q1, q2))
        {
            if (inflag == 0) aa = ba = 0;
            R.push_back(line_intersect(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 (in_poly(P[0], Q)) return P;
        if (in_poly(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;
}
```

7.12 CLOSEST POINTS

```
#include "head.h"

double dist(point a, point b)
{
    point c = a-b;
    return sqrt(c.x*c.x+c.y*c.y);
}

/*

    Aplicações:

        Determina os dois pontos mais proximos de pts

    Parametros

        vector<point> pts -> Pontos a serem analisados

    Devolve

        pair<point,point> -> Os dois pontos mais proximos pertencentes a pts

*/

bool compy(point a, point b)
{
    return cmp(a.y,b.y) ? cmp(a.y,b.y) < 0 : cmp(a.x,b.x) < 0;
}

pair<point, point> closest_points_rec(vector<point>& px, vector<point>& py)
{
    pair<point, point> ret;

    double d;

    if(px.size() <= 3)
    {
        double best = 1e10;

        for(int i = 0; i < px.size(); ++i)
        {
            for(int j = i + 1; j < px.size(); ++j)
            {
                if(dist(px[i], px[j]) < best)
                {
```

```
                    ret = make_pair(px[i], px[j]);
                    best = dist(px[i], px[j]);
                }
            }
        }

        return ret;
    }

    point split = px[(px.size() - 1)/2];

    vector<point> qx, qy, rx, ry;

    for(int i = 0; i < px.size(); ++i)
    {
        if(px[i] <= split) qx.push_back(px[i]);
        else rx.push_back(px[i]);
    }

    for(int i = 0; i < py.size(); ++i)
    {
        if(py[i] <= split) qy.push_back(py[i]);
        else ry.push_back(py[i]);
    }

    ret = closest_points_rec(qx, qy);

    pair<point, point> rans = closest_points_rec(rx, ry);

    double delta = dist(ret.first, ret.second);

    if((d = dist(rans.first, rans.second)) < delta)
    {
        delta = d;
        ret = rans;
    }

    vector<point> s;

    for(int i = 0; i < py.size(); ++i)
    {
        if(cmp(abs(py[i].x - split.x), delta) <= 0) s.push_back(py[i]);
    }

    for(int i = 0; i < s.size(); ++i)
    {
```

```
        for(int j = 1; j <= 7 && i + j < s.size(); ++j)
        {
            if((d = dist(s[i], s[i+j])) < delta)
            {
                delta = d;
                ret = make_pair(s[i], s[i+j]);
            }
        }
    }

    return ret;
}

pair<point, point> closest_points(vector<point> pts)
{
    if(pts.size() == 1) return make_pair(point(-INF, -INF), point(INF, INF));

    sort(pts.begin(), pts.end());

    for(int i = 0; i + 1 < pts.size(); ++i)
        if(pts[i] == pts[i+1])
            return make_pair(pts[i], pts[i+1]);

    vector<point> py = pts;

    sort(py.begin(), py.end(), compy);

    return closest_points_rec(pts, py);
}
```

7.13 CIRCULO DEFINIDO POR 3 PONTOS

```
#include "head.h"

/*

    Aplicações:

        Acha o circulo definido pelos pontos p1, p2 e p3

    Parametros

        point p1;
        point p2;
        point p3;

    Devolve

        double x,y -> Pontos do centro do circulo
        double r -> Raio do circulo

*/

double x, y, r;

void find_circle(point *p1, point *p2, point *p3)
{
    double x1, y1, x2, y2, x3, y3, temp;

    x1 = p1->x, y1 = p1->y;
    x2 = p2->x, y2 = p2->y;
    x3 = p3->x, y3 = p3->y;

    if (x1 == x2) swap(y3,y2), swap(x3,x2);

    y    =    ((x1-x2)*(x1*x1+y1*y1-x3*x3-y3*y3)-(x1-x3)*(x1*x1+y1*y1-x2*x2-
y2*y2)) / (2*((y2-y1)*(x1-x3)-(y3-y1)*(x1-x2)));

    x = (x1*x1+y1*y1-x2*x2-y2*y2+2*y*(y2-y1)) / (2*(x1-x2));

    r = sqrt((x-x1)*(x-x1)+(y-y1)*(y-y1));
}
```

7.14 CENTRÓIDE

```
#include "head.h"

/*

    Aplicações:

        Calcula o centroide (centro de massa/gravidade) de um poligono

    Parametros

        polygon pts -> poligono a ser analisado

    Devolve

        point p -> centroide do poligono pts

*/
point centroide(polygon &pts)
{
    pts.push_back(pts[0]);

    int n = pts.size() - 1;
    double area = 0;

    for(int i=0; i<n; i++)
    {
        area += pts[i].x*pts[i+1].y-pts[i].y*pts[i+1].x;
    }

    double cx = 0, cy = 0;

    for(int i=0; i<n; i++)
    {
        cx += (pts[i].x+pts[i+1].x) * (pts[i].x*pts[i+1].y - pts[i+1].x*pts[i].y);
        cy += (pts[i].y+pts[i+1].y) * (pts[i].x*pts[i+1].y - pts[i+1].x*pts[i].y);
    }

    cx /= area*3.0;
    cy /= area*3.0;

    return point(cx,cy);
}
```

7.15 PROJEÇÃO DE PONTO SOBRE RETA

```
#include "head.h"

double escalar(point a, point b)
{
    return a.x*b.x+a.y*b.y;
}

point getProjecaoPontoNaReta(point a, point b, point c)
{
    point u(b.x-a.x, b.y-a.y);
    point v(c.x-a.x, c.y-a.y);
    double k = escalar(u, v)/escalar(u, u);
    point w((int)(k*u.x), (int)(k*u.y));
    point d(a.x+w.x, a.y+w.y);
    return d;
}

point getPontoDeslocadoNaReta(point a, point b, double dist)
{
    int dx = a.x-b.x;
    int dy = a.y-b.y;
    double normaAB = distancia(a,b);
    double razao = dist/normaAB;
    return point((int)(a.x-dx*razao), (int)(a.y-dy*razao));
}

point getPontoDeslocadoEmDirecaoAReta(point a, point b, point c, double
dist)
{
    point d = getProjecaoPontoNaReta(a, b, c);
    point e = getPontoDeslocadoNaReta(c, d, dist);
    return e;
}

int main()
{
    point a(-1,2);
    point b(2,1);
    point c(1,3);

    point d = getProjecaoPontoNaReta(a,b,c);

    cout << d << endl;
}
```

7.16 CORTE DE POLIGONOS

```
#include <cmath>
#include <vector>

using namespace std;

const double eps = 1e-9;
inline int diff(double lhs, double rhs)
{
    if (lhs - eps < rhs && rhs < lhs + eps) return 0;
    return (lhs < rhs) ? -1 : 1;
}
inline bool is_between(double check, double a, double b)
{
    if (a < b)
        return (a - eps < check && check < b + eps);
    else
        return (b - eps < check && check < a + eps);
}

struct Point
{
    double x, y;
    Point() {}
    Point(double x_, double y_): x(x_), y(y_) {}
    bool operator==(const Point& rhs) const
    {
        return diff(x, rhs.x) == 0 && diff(y, rhs.y) == 0;
    }
    const Point operator+(const Point& rhs) const
    {
        return Point(x + rhs.x, y + rhs.y);
    }
    const Point operator-(const Point& rhs) const
    {
        return Point(x - rhs.x, y - rhs.y);
    }
    const Point operator*(double t) const
    {
        return Point(x * t, y * t);
    }
};

struct Line
{
    Point pos, dir;
    Line() {}
    Line(const Point& pos_, const Point& dir_): pos(pos_), dir(dir_) {}
};

inline double inner(const Point& a, const Point& b)
{
    return a.x * b.x + a.y * b.y;
}

inline double outer(const Point& a, const Point& b)
{
```

```
    return a.x * b.y - a.y * b.x;
}

inline int ccw_line(const Line& line, const Point& point)
{
    return diff(outer(line.dir, point - line.pos), 0);
}


bool get_cross(const Line& a, const Line& b, Point& ret)
{
    double mdet = outer(b.dir, a.dir);
    if (diff(mdet, 0) == 0) return false;
    double t2 = outer(a.dir, b.pos - a.pos) / mdet;
    ret = b.pos + b.dir * t2;
    return true;
}

inline double dist2(const Point &a, const Point &b)
{
    return inner(a - b, a - b);
}

// left side of a->b
vector<Point> cut_polygon(const vector<Point>& polygon, Line line)
{
    if (!polygon.size()) return polygon;
    typedef vector<Point>::const_iterator piter;
    piter la, lan, fi, fip, i, j;
    la = lan = fi = fip = polygon.end();
    i = polygon.end() - 1;
    bool lastin = diff(ccw_line(line, polygon[polygon.size() - 1]), 0) > 0;
    for (j = polygon.begin() ; j != polygon.end() ; j++)
    {
        bool thisin = diff(ccw_line(line, *j), 0) > 0;
        if (lastin && !thisin)
        {
            la = i;
            lan = j;
        }
        if (!lastin && thisin)
        {
            fi = j;
            fip = i;
        }
        i = j;
        lastin = thisin;
    }
    if (fi == polygon.end())
    {
        if (!lastin) return vector<Point>();
        return polygon;
    }
    vector<Point> result;
    for (i = fi ; i != lan ; i++)
    {
        if (i == polygon.end())
        {
```

```
            i = polygon.begin();
            if (i == lan) break;
        }
        result.push_back(*i);
    }
    Point lc, fc;
    get_cross(Line(*la, *lan - *la), line, lc);
    get_cross(Line(*fip, *fi - *fip), line, fc);
    result.push_back(lc);
    if (diff(dist2(lc, fc), 0) != 0) result.push_back(fc);
    return result;
}
```

8. GEOMETRIA 3D

8.1 TEMPLATE

```
#include <cstdio>
#include <string>
#include <cstring>
#include <vector>
#include <algorithm>
#include <map>
#include <utility>
#include <cmath>
#include <queue>
#include <stack>
#include <set>
#include <deque>
#include <iostream>
#include <sstream>

#define REP(i,n) for(int i=0;i<(int)n;++i)
#define EACH(i,c) for(__typeof((c).begin()) i=(c).begin(); i!=(c).end(); ++i)
#define ALL(c) (c).begin(), (c).end()
#define SIZE(x) (int((x).size()))

using namespace std;

const int INF = 0x3F3F3F3F;
const double PI = 2*acos(0);
const double EPS = 1e-10;

int cmpD(double a, double b = 0.0) { return a+EPS < b ? -1 : a-EPS > b; }


/*
    Estrutura de dados que define um ponto 3D
*/

struct Point
{
    double x, y, z;
    Point(double a=0.0,double b=0.0,double c=0.0){x=a,y=b,z=c;}
    Point operator+(const Point &P) const {return Point(x+P.x,y+P.y,z+P.z);}
    Point operator-(const Point &P) const {return Point(x-P.x,y-P.y,z-P.z);}
    Point operator*(double c) const {return Point(x*c,y*c,z*c);}
    Point operator/(double c) const {return Point(x/c,y/c,z/c);}
    double operator!() const {return sqrt(x*x+y*y+z*z);}
};
```

```
typedef struct
{
        Point ver1;
        Point ver2;
        Point ver3;
        Point ver4;

} tetrahedron;

typedef struct
{
        Point v1;
        Point v2;
        Point v3;

} triangle;

Point cross(Point A, Point B)
{
    return Point(A.y*B.z-A.z*B.y, A.z*B.x-A.x*B.z, A.x*B.y-A.y*B.x);
}

Point project(Point W, Point V) { return V * dot(W,V) / dot(V,V); }

Point diff(Point a, Point b)
{
        Point c;

        c.x = a.x - b.x;
        c.y = a.y - b.y;
        c.z = a.z - b.z;

        return(c);
}
```

8.2 PRODUTO ESCALAR

```
double dot(Point A, Point B) { return A.x*B.x + A.y*B.y + A.z*B.z; }
```

8.3 INTERSECÇÃO DE SEGMENTOS DE RETAS

```
// do the segments A-B and C-D intersect? (assumes coplanar)
bool seg_intersect(Point A, Point B, Point C, Point D)
{
    return cmpD(dot(cross(A-B,C-B),cross(A-B,D-B))) <= 0 &&
    cmpD(dot(cross(C-D,A-D),cross(C-D,B-D))) <= 0;
}
```

8.4 DISTÂNCIA DE PONTO A SEGMENTO DE RETA

```
double dist_point_seg(Point P, Point A, Point B)
{
    Point PP = A + project(P-A,B-A);
    if (cmpD(!(A-PP)+!(PP-B),!(A-B)) == 0) return !(P-PP);//distance point-line!
    return min(!(P-A),!(P-B));
}
```

8.5 DISTÂNCIA DE SEGMENTO DE RETA A SEGMENTO DE RETA

```
// segment-segment distance (lines too!)
double dist_seg_seg(Point A, Point B, Point C, Point D)
{
    Point E = project(A-D,cross(B-A,D-C)); // distance between lines!
    if (seg_intersect(A,B,C+E,D+E)) return !E;
    return min( min( dist_point_seg(A,C,D), dist_point_seg(B,C,D) ),
    min( dist_point_seg(C,A,B), dist_point_seg(D,A,B) ) );
}
```

8.6 DISTÂNCIA DE PONTO A TRIANGULO

```
// point-triangle distance. dps = dist_point_seg
double dist_point_tri(Point P, Point A, Point B, Point C)
{
    Point N = cross(A-C,B-C);
    Point PP = P + project(C-P,N);
    Point V1 = cross(PP-A,B-A);
    Point V2 = cross(PP-B,C-B);
    Point V3 = cross(PP-C,A-C);

    if (cmpD(dot(V1,V2)) >= 0 && cmpD(dot(V1,V3)) >= 0 && cmpD(dot(V2,V3)) >= 0)
    return !(PP-P); // distance point-plane!

    return min(dist_point_seg(P,A,B),
           min(dist_point_seg(P,A,C),
           dist_point_seg(P,B,C)));
}
```

8.7 DISTÂNCIA DE TETRAEDRO A TETRAEDRO

```
double dist_tet_tet(Point T1[4], Point T2[4])
{
    double ans = INF;
    for (int i=0; i < 4; i++) // arestas -> arestas
        for (int j=i+1; j < 4; j++)
            for (int ii=0; ii < 4; ii++)
                for (int jj=ii+1; jj < 4; jj++)
                    ans = min( ans, dist_seg_seg(T1[i],T1[j],T2[ii],T2[jj]) );

    // pontos -> planos
    for (int i=0; i < 4; i++)
        for (int j=i+1; j < 4; j++)
            for (int k=j+1; k < 4; k++)
                for (int x=0; x < 4; x++)
                    ans = min( ans, dist_point_tri(T1[x],T2[i],T2[j],T2[k]) ),
                    ans = min( ans, dist_point_tri(T2[x],T1[i],T1[j],T1[k]) );

    return ans;
}
```

8.8 VOLUME DE TETRAEDRO

```
double volume(Point T[4])
{
    return fabs(dot(T[3],cross(T[1]-T[0],T[2]-T[0]))/6.);
}
```

8.9 DISTANCIA DE PONTO A ORIGEM

```
double Length(Point a)
{
    return(sqrt(a.x*a.x + a.y*a.y + a.z*a.z));
}
```

8.10 ÁREA DE TRIÂNGULO

```
double Area(triangle t)
{
        Point a, b, c;

        a = diff(t.v3,t.v1);
        b = diff(t.v2,t.v1);
        c = cross(a,b);

        return(Length(c)/2);
}
```

8.11 ÁREA DE TETRAEDRO

```
double Area(triangle t)
{
        Point a, b, c;

        a = diff(t.v3,t.v1);
        b = diff(t.v2,t.v1);
        c = cross(a,b);

        return(Length(c)/2);
}

triangle Side(tetrahedron T,int i)
{
        triangle t;

        switch (i)
        {
        case 1: t.v1 = T.ver2;
                t.v2 = T.ver3;
                t.v3 = T.ver4;
                break;

        case 2: t.v1 = T.ver1;
                t.v2 = T.ver3;
                t.v3 = T.ver4;
                break;

        case 3: t.v1 = T.ver1;
                t.v2 = T.ver2;
                t.v3 = T.ver4;
                break;

        case 4: t.v1 = T.ver1;
                t.v2 = T.ver2;
                t.v3 = T.ver3;
                break;
        }

        return(t);
}
```

```
double area_tet(tetrahedron T)
{
    double area = 0;
    for(int i=0; i<4; i++) area = area + Area(Side(T,i));
    return area;
}
```

9. PROGRAMAÇÃO DINÂMICA

9.1 MAX 1D RANGE SUM (KADANE)

```
#include <stdio.h>
#include <vector>

using namespace std;

/*
    Aplicações:

        Determinar a subarray de maior soma de um
        conjunto de numeros

    Complexidade:

        O(n)

*/

int max_so_far; //Max 1D range Sum
int begin; //Inicio da subsequencia
int end; //Fim da subsequencia

//Max 1D range Sum
void kadane(std::vector<int>& numbers)
{
    // Initialize variables here
    int max_ending_here = numbers[0];
    int begin_temp = 0;

    max_so_far  = numbers[0];
    begin = 0;
    end = 0;

    for(int i = 1; i < numbers.size(); i++)
    {
        // calculate max_ending_here
        if(max_ending_here < 0)
        {
            max_ending_here = numbers[i];
            begin_temp = i;
        }
        else
        {
```

```
            max_ending_here += numbers[i];
        }

        // calculate max_so_far
        if(max_ending_here > max_so_far )
        {
            max_so_far  = max_ending_here;
            begin = begin_temp;
            end = i;
        }
    }
}
```

9.2 MAX 2D RANGE SUM

```
#include <stdio.h>
#include <vector>
#include <string.h>
#include <limits.h>

using namespace std;

/*
    Aplicações:

        Determina a sub-matriz de soma maxima em uma matriz

    Retorna:

        int finalTop ->
        int finalLeft ->
        int finalBottom ->
        int finalRight ->
        int maxSum ->
*/

#define ROW 100
#define COL 100

int M[ROW][COL];
int finalTop;
int finalLeft;
int finalBottom;
int finalRight;
int maxSum;

int kadane(int* arr, int* start, int* finish, int n)
{
    int sum = 0, maxSum = INT_MIN, i;
    // needed if sum NEVER becomes less than 0
    *start = 0;
    for (i = 0; i < n; ++i)
    {
        sum += arr[i];
        if (sum < 0)
        {
            sum = 0;
            *start = i+1;
        }
```

```
        else if (sum > maxSum)
        {
            maxSum = sum;
            *finish = i;
        }
    }
    return maxSum;
}
void findMaxSum()
{
    // Variables to store the final output
    maxSum = 0;
    int left, right, i;
    int temp[ROW], sum, start, finish;
    // Set the left column
    for (left = 0; left < COL; ++left)
    {
        // Initialize all elements of temp as 0
        memset(temp, 0, sizeof(temp));
        // Set the right column for the left column set by outer loop
        for (right = left; right < COL; ++right)
        {
            // Calculate sum between current left and right for every row 'i'
            for (i = 0; i < ROW; ++i)
                temp[i] += M[i][right];

            // Find the maximum sum subarray in temp[]. The kadane() function
            // also sets values of start and finish.  So 'sum' is sum of
            // rectangle between (start, left) and (finish, right) which is the
            //  maximum sum with boundary columns strictly as left and right.
            sum = kadane(temp, &start, &finish, ROW);
            // Compare sum with maximum sum so far. If sum is more, then update
            // maxSum and other output values
            if (sum > maxSum)
            {
                maxSum = sum;
                finalLeft = left;
                finalRight = right;
                finalTop = start;
                finalBottom = finish;
            }
        }
    }
}
```

9.3 LONGEST INCREASING SUBSEQUENCE (LIS)

```
#include <stdio.h>
#include <iostream>
#define MAX 500

/*
    LIS -> Maior Subsequencia Crescente
    Tempo: ( n^2 + n ) / 2

    dist[i] -> vetor auxiliar que computa a LIS do i-ésimo vertice
    seq[] -> vetor que guarda a sequencia inteira
    pred[] -> vetor de predecessores para se recuperar a LIS

    Parametros;
        n = tamanho da sequencia seq[]

    Retorno:
        pair<int,int>:
            first -> Tamanho da LIS
            second -> vertice final da LIS
*/

using namespace std;

int dist[MAX], seq[MAX], pred[MAX];

pair<int,int> LIS(int n)
{
    int i, j;
    pair<int,int> lis;

    lis.first = 0;

    for(i=0; i<n; i++)
    {
        dist[i] = 1;
        pred[i] = i;

        for(j=0; j<i; j++)
        {
            if(seq[j] < seq[i] && dist[j]+1 > dist[i]) //mudar para menor
ou igual para ter elementos repetidos
            {
                dist[i] = dist[j]+1;
                pred[i] = j;
```

```
                if(lis.first < dist[i])
                {
                    lis.first = dist[i];
                    lis.second = i;
                }
            }
        }
    }

    return lis;
}

int main()
{
    int n, i, j,vet[MAX];
    pair<int,int> lis;

    scanf("%d",&n);

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

    lis = LIS(n);

    i = lis.second;
    j=0;

    while(1)
    {
        vet[j++] = i;
        if(i == pred[i]) break;
        else i = pred[i];
    }

    for(i=j-1; i>=0; i--) printf("%d ",seq[vet[i]]);

    return 0;
}
```

9.4 LONGEST COMOM SUBSEQUENCE (LCS)

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

using namespace std;

/*
    LCS -> Maior Subsequencia Comum
    Tempo: m*n

    Parametros:
        x = string 1 a ser comparada
        y = string 2 a ser comparada

    Retorno:
        pair<int,string>:
            first -> tamanho da maior subsequencia
            second -> maior subsequencia comum;
*/

pair<int, string> LCS(string x, string y)
{
    int m,n,i,j;

    m = x.length();
    n = y.length();

    pair<int, string> PD[m+1][n+1];

    for(i=0; i<=m; i++) //Zera a coluna 0
    {
        PD[i][0].first = 0;
        PD[i][0].second = "";
    }

    for(i=0; i<=n; i++) //Zera a linha 0
    {
        PD[0][i].first = 0;
        PD[0][i].second = "";
    }


    for(i=1; i<=m; i++)
    {
```

```
        for(j=1; j<=n; j++)
        {
            if(x[i-1] == y[j-1]) //Elemento igual, logo PD[i][j] = PD[i-1][j-1]+1
            {
                PD[i][j].first = PD[i-1][j-1].first + 1;
                PD[i][j].second = PD[i-1][j-1].second + x[i-1];
            }
            else //Elementos diferentes, PD[i][j] = MAX(PD[i-1][j], PD[i][j-1])
            {
                PD[i][j] = PD[i-1][j].first > PD[i][j-1].first ? PD[i-1][j] : PD[i][j-1];
            }
        }
    }

    return PD[m][n];
}
```

9.5 0-1 KNAPSACK (SUBSET-SUM)

```
#include <stdio.h>
#include <iostream>
#define MAX 100
using namespace std;
/*
    Aplicações:

        Dados nÃºmeros naturais  (p1, p2, â€¦ , pn), (v1, v2, â€¦ , vn)
        e  c,  encontrar um subconjunto X de {1,2,â€¦,n} que
        maximize v(X)   sob a restriÃ§Ã£o  p(X) â‰¤ c. Em outras palavras
        Determinar o subconjunto de elementos que maximize a soma
        dos valores e que possua a soma dos pesos menores ou iguais a c
*/
int p[MAX]; //Peso dos elementos
int v[MAX]; //Valor dos elementos
int n; //Numero de elementos do conjunto
int c; //Peso maximo suportado

int knapsack_0_1()
{
      register int mat[n+1][c+1], i, j;
      for(i=0; i<n; i++) mat[i][0] = 0;
      mat[i][0] = 0;
      for(i=0; i<=c; i++) mat[0][i] = 0;
      for(i=1; i<=n; i++)
      {
            for(j=1; j<=c; j++)
            {
                  if(p[i-1] > j) mat[i][j] = mat[i-1][j];
                  else mat[i][j] = max(mat[i-1][j], mat[i-1][j-p[i-1]]+v[i-1]);
            }
      }

      /*for(i=0; i<=n; i++)
      {
            for(j=0; j<=c; j++) printf("%d ",mat[i][j]);
            printf("\n");
      }*/

      return mat[n][c];
}
```

9.6 EDITION DISTANCE

```cpp
#include <iostream>
#include <stdio.h>
#include <vector>

using namespace std;

/*
    Aplicações:

        Encontrar o numero minimo de operacoes
        necessarias para transformar uma string em outra

    Parametros

        s1 -> string 1
        s2 -> string 2

    Retorna

        A distancia de edicao

*/

int edit_distance(string s1, string s2)
{
    const int len1 = s1.size(), len2 = s2.size();
    vector<vector<int> > d(len1 + 1, vector<int>(len2 + 1));
    d[0][0] = 0;
    for(int i = 1; i <= len1; ++i) d[i][0] = i;
    for(int i = 1; i <= len2; ++i) d[0][i] = i;
    for(int i = 1; i <= len1; ++i)
    {
        for(int j = 1; j <= len2; ++j)
        {
            d[i][j] = min( min(d[i-1][j]+1,d[i][j-1]+1),
                           d[i-1][j-1]+(s1[i-1]==s2[j-1] ? 0 : 1) );
        }
    }

    return d[len1][len2];
}
```

9.7 COIN CHANGE (CC)

```c
#include <stdio.h>
#define MAX 100


/*
    Aplicações:

        Descobrir o numero de maneiras de obter um valor m
        somando os elementos do conjunto S.

        Obs. Os elementos de S podem ser usados ilimitadamente

*/

int S[MAX];

int count(int n, int m)
{
    if(n == 0) return 1;
    if(n < 0) return 0;
    if(m <= 0 && n >= 1) return 0;
    return count(n,m-1) + count(n-S[m], m);
}
```

9.8 TODAS AS SUBSEQUENCIAS DE SOMA K

```
#include <stdio.h>
#include <algorithm>
#include <map>
#include <iostream>
#include <set>
#include <vector>
#include <stdlib.h>

using namespace std;

/*
    Aplicações:

        Dado um conjunto de elementos, determinar
        todas as subsequencias deste conjunto que possuam
        soma v

    Parametros:

        int vet[] -> conjunto de elementos
        int n -> numero de elementos do conjunto
        int t -> valor de soma a ser buscado

    Retorna:

        intervals -> intervalos (inicio, fim) de vet que possuem soma v
*/

vector< pair<int,int> > interval_sum(int vet[], int n, int v)
{
    vector< pair<int,int> > intervals;
    int sum=vet[0];
    int ini=0;
    int fim=0;

    while(ini < n)
    {
        if(sum == v)
        {
            intervals.push_back(pair<int,int>(ini,fim));
            sum -= vet[ini];
            ini++;
        }
        else if(sum < v)
```

```
        {
            if(fim+1 < n)
            {
                fim++;
                sum += vet[fim];
            }
            else break;
        }
        else
        {
            sum -= vet[ini];
            ini++;
        }
    }

    return intervals;
}
```

9.9 TRAVELING SALESMAN PROBLEM (TSP)

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

using namespace std;

/*
    Apliclações:

        O problema do caixeiro viajante é um problema
        que tenta determinar a menor rota para percorrer
        uma série de cidades (visitando cada uma pelo
        menos uma vez), retornando à cidade de origem.
        Ele é um problema de otimização NP-Completo
        inspirado na necessidade dos vendedores em realizar
        entregas em diversos locais (as cidades) percorrendo
        o menor caminho possível, reduzindo o tempo necessário
        para a viagem e os possíveis custos com transporte
        e combustível.


*/

int n, grafo[11][11], memo[11][1 << 11];

int tsp(int pos, int bitmask)    // bitmask stores the visited coordinates
{
    if (bitmask == (1 << (n + 1)) - 1) return grafo[pos][0]; // return
trip to close the loop

    if (memo[pos][bitmask] != -1) return memo[pos][bitmask];

    int ans = 2000000000;

    for (int nxt = 0; nxt <= n; nxt++) // O(n) here
        if (nxt != pos && !(bitmask & (1 << nxt))) // if coordinate nxt is not visited yet
            ans = min(ans, grafo[pos][nxt] + tsp(nxt, bitmask | (1 << nxt)));

    return memo[pos][bitmask] = ans;
}
```

9.10 INTERVAL SUM

```
#include <stdio.h>
#include <algorithm>
#include <map>
#include <iostream>
#include <set>
#include <vector>
#include <stdlib.h>

using namespace std;

/*
    Aplicacoes:

        Dado um conjunto de elementos, determinar
        todas as subsequencias deste conjunto que possuam
        soma v

    Parametros:

        int vet[] -> conjunto de elementos
        int n -> numero de elementos do conjunto
        int t -> valor de soma a ser buscado

    Retorna:

        intervals -> intervalos (inicio, fim) de vet que possuem soma v
*/

vector< pair<int,int> > interval_sum(int vet[], int n, int v)
{
    vector< pair<int,int> > intervals;
    int sum=vet[0];
    int ini=0;
    int fim=0;

    while(ini < n)
    {
        if(sum == v)
        {
            intervals.push_back(pair<int,int>(ini,fim));
            sum -= vet[ini];
            ini++;
        }
        else if(sum < v)
        {
            if(fim+1 < n)
            {
                fim++;
                sum += vet[fim];
            }
            else break;
        }
        else
        {
            sum -= vet[ini];
            ini++;
```

```
            }
        }

        return intervals;
}

int main()
{
        for(int i=0; i<5000; i++)
        for(int j=0; j<5000; j++);
}
```

10. MISCELLANEOUS

10.1 SUBCOLEÇÃO DISJUNTA MÁXIMA

```
#include <stdio.h>
#include <algorithm>
#include <map>
#include <iostream>
#include <set>
#include <vector>

using namespace std;

/*

    Aplica��es:

        Dados um conjunto de tarefas determinandas
        por seus momentos de inicio e fim, determinar a
        subcole��o m�xima destas tarefas de modo a
        maximizar o numero de tarefas realizadas

        Obs. Armazenar o conjunto de tarefas na variavel
        tarefas de forma inversa, ou seja, o momento de
        fim da i-�sima tarefa deve ser armazendo em
        tarefas[i].first e o momento de inicio da i-�sima
        tarefa deve ser armazenado em tarefas[i].second

*/
vector< pair<int,int> > tarefas; //Armazenar em first o termino e em
second o inicio

int sdm()
{
    if(tarefas.empty()) return 0;

    sort(tarefas.begin(), tarefas.end());

    int cont=1;
    int i = 0;

    for(int k=1; k<tarefas.size(); k++)
    {
        if(tarefas[k].second > tarefas[i].first)
        {
            cont++;
            i = k;
        }
    }

    return cont;
}
```

10.2 CREDIT CARD CHECK

```
#include <iostream>

using namespace std;

int sumDigits(int n)
{
    return n>=10 ? (n%10+n/10) : n;
}

bool creditCard(string number)
{
    int i, sum1=0, sum2=0;

    for(i=0; i<number.size(); i+=2)
    {
        sum1 += sumDigits((number[i]-'0')*2);
    }

    for(i=1; i<number.size(); i+=2)
    {
        sum2 += (number[i]-'0');
    }

    return ((sum1+sum2)%10 == 0) ? true : false;
}

int main()
{

}
```

10.3 HORA TO INT

```
#include <iostream>
#include <string.h>
#include <stdlib.h>

using namespace std;

int convertTime(char time[])
{
    int h,m,s,ms;

    char *t = strtok(time, ":");
    h = atoi(t);
    t = strtok(NULL, ":");
    m = atoi(t);
    t = strtok(NULL, ":");
    s = atoi(t);
    t = strtok(NULL, ":");
    ms = atoi(t);

    return ms + s*1000 + m*60000 + h*3600000;
}

int main()
{

}
```

10.4 BUSCA BINARIA

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

using namespace std;

int buscabinaria( pair<int,int> array[], int chave , int N)
{
     int inf = 0;
     int sup = N-1;
     int meio;
     while (inf <= sup)
     {
          meio = inf + (sup-inf)/2;
          if (chave == array[meio].first)
               return array[meio].second;
          else if (chave < array[meio].first)
               sup = meio-1;
          else
               inf = meio+1;
     }
     return -1;
}

int main()
{

}
```

10.5 SUDOKU SOLVER

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

/*

    Sudoku Solver

    Aplica��es:

        Resolver um sudoku pre preenchido

    Como chamar a funcao:

        1) Criar uma variavel do tipo SudokuSolver e inicializa-la
           com uma string que representa a uniao das linhas do sudoku
           em uma unica linha. Exemplo:

                SudokuSolver ss((string) "850002400" +
                                (string) "720000009" +
                                (string) "004000000" +
                                (string) "000107002" +
                                (string) "305000900" +
                                (string) "040000000" +
                                (string) "000080070" +
                                (string) "017000000" +
                                (string) "000036040"
                               );

        2) Chamar a funcao solve da estrutura. Exemplo:

                ss.solve();

    Resultado da funcao:

        A funcao solve() imprime a solucao do sudo caso ela exista.
        Se o sudoku n�o possuir solucao uma mensagem informando isso
        � impressa na tela.

    Complexidade do algoritmo:

        O(?)

    Problemas resolvidos:

        EASUDOKU (SPOJ)

    Adicionado por:

        Jorge Gabriel Siqueira

*/

class SudokuSolver
{
private:
```

```
    int grid[81];

public:

    SudokuSolver(string s)
    {
        for (unsigned int i = 0; i < s.length(); i++)
        {
            grid[i] = (int) (s[i] - '0');
        }
    }

    void solve()
    {
        try
        {
            placeNumber(0);
            cout << "Unsolvable!" << endl;
        }
        catch (char* ex)
        {
            cout << ex << endl;
            cout << this->toString() << endl;
        }
    }

    void placeNumber(int pos)
    {
        if (pos == 81)
        {
            throw (char*) "Finished!";
        }
        if (grid[pos] > 0)
        {
            placeNumber(pos + 1);
            return;
        }
        for (int n = 1; n <= 9; n++)
        {
            if (checkValidity(n, pos % 9, pos / 9))
            {
                grid[pos] = n;
                placeNumber(pos + 1);
                grid[pos] = 0;
            }
        }
    }

    bool checkValidity(int val, int x, int y)
    {
        for (int i = 0; i < 9; i++)
        {
            if (grid[y * 9 + i] == val || grid[i * 9 + x] == val)
                return false;
        }
        int startX = (x / 3) * 3;
        int startY = (y / 3) * 3;
        for (int i = startY; i < startY + 3; i++)
```

```
        {
            for (int j = startX; j < startX + 3; j++)
            {
                if (grid[i * 9 + j] == val)
                    return false;
            }
        }
        return true;
    }

    string toString()
    {
        string sb;
        for (int i = 0; i < 9; i++)
        {
            for (int j = 0; j < 9; j++)
            {
                char c[2];
                c[0] = grid[i * 9 + j] + '0';
                c[1] = '\0';
                sb.append(c);
                sb.append(" ");
                if (j == 2 || j == 5)
                    sb.append("| ");
            }
            sb.append("\n");
            if (i == 2 || i == 5)
                sb.append("------+-------+------\n");
        }
        return sb;
    }

};

int main()
{
    SudokuSolver ss((string) "850002400" +
                    (string) "720000009" +
                    (string) "004000000" +
                    (string) "000107002" +
                    (string) "305000900" +
                    (string) "040000000" +
                    (string) "000080070" +
                    (string) "017000000" +
                    (string) "000036040"
                   );
    ss.solve();
}
```

10.6 JOSEPHUS

```
#include <cstdio>
#include <string>
#include <cstring>
#include <vector>
#include <algorithm>
#include <map>
#include <utility>
#include <cmath>
#include <queue>
#include <stack>
#include <set>
#include <deque>
#include <iostream>
#include <math.h>
#include <sstream>
#define INF 0x3f3f3f3f

using namespace std;

/*

    Josephus Problem Solver

    Aplicações:

        n pessoas dispostas em um circulo, as pessoas sao numeradas de [0
.. n-1]
        e a cada intervalo k de pessoas é morta a pessoa correspondente ao
final do intervalo,
        no final sobra apenas um sobrevivete(Josephus).

    Como chamar a função:

        1) Chamar a função int Josephus(int n, int k)

            int n -> quantidade de pessoas
            int k -> intervalo no qual cada pessoa morre

    Resultado da função:

        Função Josephus retorna a pessoa sobrevivente

    Complexidade do algoritmo:

       O(n) se K>2 e O(1) se K<=2

    Problemas resolvidos:



    Adicionado por:
        Fúlvio Abrahão
*/

int Josephus(int n, int k){
        if(k == 1)return n-1;
```

```
        if(k == 2){
            int j = floor(log10(n)/log10(2.0));
            n &= ~(1<<j);
            n = n<<1;
            n|=1;
            return n-1;
        }

        if(n==1)return 0;
        return (Josephus(n-1,k)+k)%n;
}

int main()
{

    return 0;
}
```

10.7 FAST SUDOKU SOLVER

```
#include <stdio.h>

#define INF 500

using namespace std;

/*

    Fast Sudoku Solver

    Aplicacoes:

        Resolver um sudoku pre preenchido

    Como chamar a funcao:

        1) Chamar a funcao initialize()
            Obs. S? eh necess?ria a chamada desta funcao uma vez

        2) Preencher o sudoku com os valores ja preenchidos usando
           a funcao insert() de V na variavel s. Exemplo:

                s.insert(i*9+j,x);

                Insere o valor x na linha i e coluna j do sudoku
                a representacao eh a representacao usual de matriz
                em vetor.

                O valor 0 em x representa uma cehlula vazia do sudoku

        3.1) Chamar a funcao solve() para resolver o sudoku passando
             o sudoku (s) como parametro. Exemplo de chamada:

                solve(s);

        3.2) Para verificar se o sudoku possui erro chamar a funcao
             find_error(). Esta funcao verifica se o sudoku inicial
             possui algum erro de preenchimento que invalida sua
             solucao. Exemplo de chamada:

                find_error(s);

    Resultado da funcao:

        3.1) A funcao solve imprime o resultado (caso haja). A funcao
             retorna:

                true -> caso o sudoku foi resolvido com sucesso
                false -> caso o sudoku nao possua solucao

        3.2) A funcao find_error() retorna:

                true -> caso o sudoku nao possua erro de preencimento
                false -> caso contrario
```

```
    Complexidade do algoritmo:

        O(?)

    Problemas resolvidos:

        BSUDO (SPOJBR)
        EASUDOKU (SPOJ)

    Adicionado por:

        Jorge Gabriel Siqueira

*/

/*vetor de palavras com tamanho menor que 32 bits**/
typedef long long lint;
typedef unsigned long long ulint;

struct V
{
    //tam_word=9  NMAX=20;
    //n_words=64/9 numero de palavras por chunck
    int n_words, nbits, MASK;

    ulint v[6];
    ulint mask;

    V(int n_words=5,int nbits=10, int MASK=1023)
        :n_words(n_words),nbits(nbits),mask(MASK)
    {
        for(int i=0; i<6; i++) v[i]=0;
    }

    V(const V& b)
    {
        n_words=b.n_words;
        nbits=b.nbits;
        MASK=b.MASK;
        mask=b.mask;
        for(int i=0; i<6; i++) v[i]=b.v[i];
    }

    inline
    void insert(int p, ulint V)
    {
        int i;

        i=p/n_words;
        p=p%n_words;

        p*=nbits;

        v[i]=(v[i]^(v[i] & (mask<<p))) | (V<<p);
    }

    inline
    int get(int p)
```

```
    {
        int i;

        i=p/n_words;
        p=p%n_words;
        p*=nbits;

        return (int)((v[i]>>p)&mask);
    }

    V& operator=(const V& b)
    {
        n_words=b.n_words;
        nbits=b.nbits;
        MASK=b.MASK;
        mask=b.mask;
        for(int i=0; i<6; i++) v[i]=b.v[i];
        return *this;
    }

};

struct st
{
    V s, pl, pc, ps;//estado anterior
    int le, x, y;//ultima escolha e posicao onde ocorreu

    st():s(16,4,15),pl(),pc(),ps(),le(1),x(0),y(0) {}

    st(const V& s, const V& pl, const V& pc, const V& ps, int le=1, int
x=0, int y=0)
        :s(s),pl(pl),pc(pc),ps(ps),le(le),x(x),y(y) {}

    st& operator=(const st& a)
    {
        s=a.s;
        pl=a.pl;
        pc=a.pc;
        ps=a.ps;
        le=a.le;
        x=a.x;
        y=a.y;
        return *this;
    }
};


static st Q[81];
int ip;
static int hel[10];
static int lg[1<<10];
static int ones[1<<10];
static const ulint inf=~(0LL);

inline void print(V& sol)
{
    int i, j;
    for(i=0; i<9; i++)
```

```
    {
        for(j=0; j<9; j++) printf("%d",sol.get(i*9+j));
        printf("\n");
    }
}

bool solve(V& s)
{
    V podeLin, podeCol, podeSet;
    int i, j, x, y, a, b, set;
    int px, py, mi, l, ns;
    bool ok;

    for(i=0; i<2; i++) podeCol.v[i]=podeSet.v[i]=podeLin.v[i]=inf;

    ok=true;
    ip=0;
    int np=0;
    for(;;)
    {
init:
        np++;
        if(!ok)
            for(;;)
            {
                ip--;

                if(ip==-1) return false;
                else
                {
                    s=Q[ip].s;
                    podeLin=Q[ip].pl;
                    podeCol=Q[ip].pc;
                    podeSet=Q[ip].ps;
                    px=i=Q[ip].x;
                    py=j=Q[ip].y;

                    a=(i/3);
                    b=(j/3);
                    set=a*3+b;
                    y=podeLin.get(i);
                    y&=podeCol.get(j);
                    y&=podeSet.get(set);

                    for(i=Q[ip].le+1; i<10; i++)
                        if((y>>i)&0x1)
                        {
                            Q[ip]=st(s,podeLin,podeCol,podeSet,i,px,py);
                            ip++;
                            s.insert(px*9+py,i);
                            goto cont;
                        }
                }
            }

cont:
        ns=0;
        for(i=0; i<9; i++)
```

```
            for(j=0; j<9; j++)
            {
                x=s.get(i*9+j);
                a=(i/3);
                b=(j/3);
                set=a*3+b;

                if(x>0)
                {
                    ns++;
                    //Atualiza linha
                    y=podeLin.get(i);
                    y&=hel[x];
                    podeLin.insert(i,y);
                    //Atualiza coluna
                    y=podeCol.get(j);
                    y&=hel[x];
                    podeCol.insert(j,y);
                    //Atualiza setor
                    y=podeSet.get(set);
                    y&=hel[x];
                    podeSet.insert(set,y);
                }
            }


        if(ns==81)
        {
            print(s);
            return true;
        }


        mi=INF;
        ok=false;
        for(i=0; i<9; i++)
            for(j=0; j<9; j++)
            {
                x=s.get(i*9+j);
                a=(i/3);
                b=(j/3);
                set=a*3+b;

                if(x==0)
                {
                    y=podeLin.get(i);
                    y&=podeCol.get(j);
                    y&=podeSet.get(set);
                    x=lg[y];
                    if(x>0)
                    {
                        s.insert(i*9+j,x);
                        y=podeLin.get(i);
                        y&=hel[x];
                        podeLin.insert(i,y);
                        y=podeCol.get(j);
                        y&=hel[x];
                        podeCol.insert(j,y);
```

```
                        y=podeSet.get(set);
                        y&=hel[x];
                        podeSet.insert(set,y);
                        ok=true;
                    }
                    else if(y==0)  goto init;
                    else
                    {
                        x=ones[y];
                        if(x<mi)
                        {
                            mi=x;
                            l=y;
                            px=i;
                            py=j;
                        }
                    }
                }
            }


        if(!ok)
        {
            i=px;
            j=py;
            a=(i/3);
            b=(j/3);
            set=a*3+b;
            y=podeLin.get(i);
            y&=podeCol.get(j);
            y&=podeSet.get(set);

            for(i=1; i<10; i++)
                if((y>>i)&0x1)
                {
                    Q[ip]=st(s,podeLin,podeCol,podeSet,i,px,py);
                    ip++;
                    s.insert(px*9+py,i);
                    ok=true;
                    break;
                }
        }
    }
}

bool find_error(V& s)
{
    for(int n=1; n<10; n++)
    {
        int noc;
        for(int i=0; i<9; i++)
        {
            noc=0;
            for(int j=0; j<9; j++)
                if(s.get(i*9+j)==n) noc++;
            if(noc>1) return false;
        }
```

```
        for(int i=0; i<9; i++)
        {
            noc=0;
            for(int j=0; j<9; j++)
                if(s.get(j*9+i)==n) noc++;
            if(noc>1) return false;
        }

        for(int i=0; i<3; i++)
        {
            int lm=3*(i+1);
            int auxI=3*i;

            for(int j=0; j<3; j++)
            {
                int lp=3*(j+1);
                noc=0;

                for(int m=auxI; m<lm; m++)
                    for(int p=3*j; p<lp; p++)
                    {
                        int x=s.get(m*9+p);
                        if(x==n) noc++;
                    }
                if(noc>1) return false;
            }
        }

    }
    return true;
}

V s;
int len, m,c;

void initialize()
{
    int i, j, x;
    s = V(16,4,15);
    len=1<<10;
    m=len-1;
    for(i=0; i<len; i++)
    {
        lg[i]=-1;
        x=0;
        for(j=0; j<10; j++)
            if(i>>j & 0x1)x++;
        ones[i]=x;
    }

    for(i=0; i<10; i++)
    {
        lg[1<<i]=i;
        hel[i]=m-(1<<i);
        if(i>0) hel[i]--;
    }
}
```

```
int main()
{
    initialize();
    int N,i,j,x;

    scanf("%d",&N);
    while(N--)
    {
        for(i=0; i<9; i++)
        {
            for(j=0; j<9; j++)
            {
                scanf("%1d",&x);
                s.insert(i*9+j,x);
            }
        }
        solve(s);
    }
    return 0;
}
```