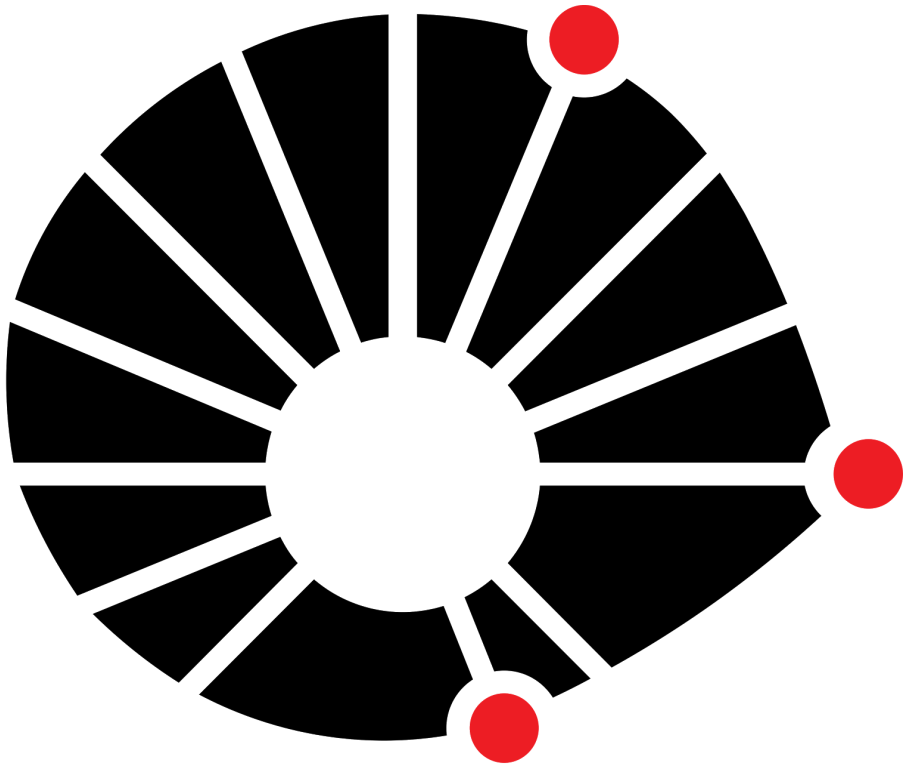
TEAM REFERENCE DOCUMENT
STATE UNIVERSITY OF CAMPINAS



UNICAMP

Contents

1 Dadalto

1.1 2 SAT

```
Complexidade: O(n + m)
Descricao:

list<int> sorted;
vector<int> graph[4123];
vector<int> tgraph[4123];
int scc[4123];

char s1[51], s2[51];
int a[1123], b[1123], neg[4123];
bool been[4123];
int id2;

void dfs(int a)
{
  been[a] = true;
  for(int i = 0; i < graph[a].size(); i++)
    if(!been[graph[a][i]])
      dfs(graph[a][i]);
  sorted.push_front(a);
}

void dfs2(int a)
{
  been[a] = true;
  scc[a] = id2;
  for(int i = 0; i < tgraph[a].size(); i++)
    if(!been[tgraph[a][i]])
      dfs2(tgraph[a][i]);
}

int
main(void)
{
  int n, t = 1;
  while(scanf("%d", &n) != EOF)
  {
    map<string,int> hash;
    sorted.clear();
    for(int i = 0; i < 4123; i++)
      graph[i].clear(), tgraph[i].clear();

    memset(been, 0, sizeof(been));
    memset(neg, 0, sizeof(neg));
    memset(scc, 0, sizeof(scc));
    int id = 1;

    for(int i = 0; i < n; i++)
    {
      scanf(" %s %s", s1, s2);
      if(hash[s1] == 0)
        hash[s1] = id++;
      if(hash[s2] == 0)
        hash[s2] = id++;
      a[i] = hash[s1], b[i] = hash[s2];
    }
```

```
    string no = "!";
    map<string,int> hash2 = hash;
    for(auto it = hash.begin(); it != hash.end(); it++)
      if((it->first)[0] != '!')
        neg[neg[it->second] = hash2[no + it->first]] = it->second;

    for(int i = 0; i < n; i++)
    {
      if(neg[a[i]] != 0)
        graph[neg[a[i]]].pb(b[i]), tgraph[b[i]].pb(neg[a[i]]);
      if(neg[b[i]] != 0)
        graph[neg[b[i]]].pb(a[i]), tgraph[a[i]].pb(neg[b[i]]);
    }

    for(int i = 1; i < id; i++)
      if(!been[i])
        dfs(i);

    memset(been, 0, sizeof(been));
    id2 = 1;
    for(auto it = sorted.begin(); it != sorted.end(); it++)
    {
      if(!been[*it])
      {
        dfs2(*it);
        id2++;
      }
    }
    bool ans = true;
    for(int i = 1; i < id; i++)
//          printf("%d\n", scc[i]);
      if(scc[neg[i]] == scc[i])
         ans = false;

    printf("Instancia %d\n%s\n\n", t++, ans ? "sim" : "nao");

  }

}
```

1.2 Bellman-Ford

```
Complexidade: O(n*m)
Descricao: Calcula distâncias a partir
da origem e encontra ciclos negativos

//http://www.spoj.com/problems/CHAMPS/
int a[112345], b[112345], c[112345], d[512];

int main(void)
{
  int n, m;
  while(scanf("%d %d", &n, &m) != EOF)
  {
    memset(d, 0x3f, sizeof(d));
    d[0] = 0;
    bool ans = true;
    for(int i = 0; i < m; c[i] *= -1, i++)
      scanf("%d %d %d", &a[i], &b[i], &c[i]);
    for(int i = 0; i + 1 < n; i++)
      for(int j = 0; j < m; j++)
        if(d[b[j]] > d[a[j]] + c[j])
          d[b[j]] = d[a[j]] + c[j];
    for(int j = 0; j < m; j++)
      if(d[b[j]] > d[a[j]] + c[j])
        ans = false;
    printf("%c\n", ans ? 'y' : 'n');
  }
}
```

1.3 Convex Hull e Point Query

```
Complexidade: O(n log n)
Descricao: Encontra o convex hull com line sweep
e responde várias queries se um ponto está dentro do convex hull

struct point{
  long long x, y;
  point(long long _x = 0, long long _y = 0) : x(_x), y(_y) {}
  long long cross(point p1, point p2) {
    return (p1.x - x)*(p2.y - y) - (p2.x - x)*(p1.y - y);
  }
  bool operator < (point b) const {
    if(x != b.x)
      return x < b.x;
    return y > b.y;
  }
} p[112345];

point ch[112345];

bool comp(point a, point b){ return a.x < b.x; }

int main(void)
{
  int n, q;
  scanf("%d %d", &n, &q);
  for(int i = 0; i < n; i++)
    scanf("%lld %lld", &p[i].x, &p[i].y);
  sort(p, p+n);
  vector<point> upper, lower;
  upper.push_back(p[0]), lower.push_back(p[0]);
  upper.push_back(p[1]), lower.push_back(p[1]);
  for(int i = 2; i < n; i++)
  {
    while(upper.size() >= 2
      && upper[upper.size() - 2].cross(upper.back(), p[i]) >= 0)
      upper.pop_back();
    upper.push_back(p[i]);
    while(lower.size() >= 2
      && lower[lower.size() - 2].cross(lower.back(), p[i]) <= 0)
      lower.pop_back();
    lower.push_back(p[i]);
  }
  int ans = 0;
  while(q--)
  {
    point o;
    scanf("%lld %lld", &o.x, &o.y);
    if(o.x < upper[0].x || o.x > upper.back().x)
      continue;
    int i = lower_bound(upper.begin(), upper.end(), o, comp)
    - upper.begin();
    int j = lower_bound(lower.begin(), lower.end(), o, comp)
    - lower.begin();

    if((i != 0 && upper[i-1].cross(upper[i], o) > 0)
      || (i + 1 < upper.size() && upper[i].cross(upper[i+1], o) > 0)
      || (i + 2 < upper.size() && upper[i+1].cross(upper[i+2], o) > 0))
      continue;
    if((j != 0 && lower[j-1].cross(lower[j], o) < 0)
      || (j + 1 < lower.size() && lower[j].cross(lower[j+1], o) < 0)
      || (j + 2 < lower.size() && lower[j+1].cross(lower[j+2], o) < 0))
      continue;
    ans++;
  }
  printf("%d\n", ans);
}
```

1.4 Decomposição por centroide

```
Complexidade: O((n + m)*log n)
Descricao: Resolve o seguinte problema, dado uma árvore encontra
o maior conjunto de vértices tal que a maior distância entre dois deles

vector<int> graph[MAXN];
int n, k, maxh, num;
int ans[MAXN], sz[MAXN], hsum[MAXN], subtree_hsum[MAXN];
bool block[MAXN];

int dfs1(int a, int p)
{
  sz[a] = 1;
  for(int i = 0; i < graph[a].size(); i++)
    if(graph[a][i] != p && !block[graph[a][i]])
      sz[a] += dfs1(graph[a][i], a);
  return sz[a];
}

void dfs2(int a, int p, pii &res)
{
  int mx = num - sz[a];
  for(int i = 0; i < graph[a].size(); i++)
    if(graph[a][i] != p && !block[graph[a][i]])
    {
      mx = max(mx, sz[graph[a][i]]);
      dfs2(graph[a][i], a, res);
    }
  if(mx < res.first)
    res = pii(mx, a);
}

int find_centroid(int a)
{
  num = dfs1(a, a);
  pair<int,int> res(inf, 0);
  dfs2(a, a, res);
  return res.second;
}
```

```
void dfs3(int a, int p, int h, int vec[])
{
  maxh = max(h, maxh);
  vec[h]++;
  for(int i = 0; i < graph[a].size(); i++)
    if(graph[a][i] != p && !block[graph[a][i]])
      dfs3(graph[a][i], a, h+1, vec);
}

void dfs4(int a, int p, int h)
{
  if(h > k)
    return;
  ans[a] += hsum[min(num, k - h)] - subtree_hsum[min(maxh, k - h)];
  for(int i = 0; i < graph[a].size(); i++)
    if(graph[a][i] != p && !block[graph[a][i]])
      dfs4(graph[a][i], a, h+1);
}

void decomp(int a)
{
  a = find_centroid(a);
  maxh = 0;
  num = dfs1(a, a);
  memset(hsum, 0, sizeof(int) * (num + 1));
  dfs3(a, a, 0, hsum);
  for(int i = 1; i <= num; i++)
    hsum[i] += hsum[i-1];
  ans[a] += hsum[min(num, k)];

  for(int i = 0; i < graph[a].size(); i++)
    if(!block[graph[a][i]])
    {
      memset(subtree_hsum, 0, sizeof(int) * (sz[graph[a][i]] + 1));
      maxh = 0;
      dfs3(graph[a][i], a, 1, subtree_hsum);
      for(int j = 1; j <= maxh; j++)
        subtree_hsum[j] += subtree_hsum[j-1];
      dfs4(graph[a][i], a, 1);
    }

  block[a] = true;
  for(int i = 0; i < graph[a].size(); i++)
    if(!block[graph[a][i]])
      decomp(graph[a][i]);
}

int main(void)
{
  int a, b;
  scanf("%d %d", &n, &k);
  for(int i = 1; i < n; i++)
  {
    scanf("%d %d", &a, &b);
    graph[a].push_back(n+i);
    graph[n+i].push_back(b);
    graph[b].push_back(n+i);
    graph[n+i].push_back(a);
  }
  decomp(1);
  int retv = 0;
  if(k%2 == 0)
    for(int i = 1; i <= n; i++)
      retv = max(retv, ans[i]);
  else
    for(int i = n+1; i < 2*n; i++)
      retv = max(retv, ans[i]);
  printf("%d\n", (retv + 1) / 2);
}
```

1.5 Grundy number

```
Complexidade: O(n^2)
Descricao: Encontra grundy number por PD

#include <stdio.h>
#include <set>
using namespace std;

int a[112345];
int tab[112345];
bool been[112345];

int pd(int n)
{
  if(n == 0)
    return 0;
  if(been[n])
    return tab[n];
  set<int> rdm;
  for(int i = 0; i < n; i++)
    rdm.insert(pd(i) ^ pd(n-i-1));
  for(int i = 0; i + 1 < n; i++)
    rdm.insert(pd(i) ^ pd(n-i-2));
  int count = 0;
  for(set<int>::iterator it = rdm.begin(); it != rdm.end(); it++)
    if(*it != count)
      break;
    else
      count++;
  been[n] = true;
  return tab[n] = count;
}

int main(void)
{
  int T;
  scanf("%d", &T);
  for(int t = 1; t <= T; t++)
  {
    int n, m, ans = 0;
    scanf("%d %d", &n, &m);

    for(int i = 0; i < m; i++)
      scanf("%d", &a[i]);
    for(int i = 1; i < m; i++)
      ans ^= pd(a[i] - a[i-1] - 1);
    ans ^= pd(a[0] - 1 + n - a[m-1]);

    printf("Case %d: %s\n", t, ans ? "yes" : "no");
  }
}
```

1.6 KMP

```
Complexidade: O(n + m)
Descricao: Procura palavra em uma string

//http://www.spoj.com/problems/NHAY/
string needle;
int next[11234567];

int foo(int i, char c)
{
  if(i == 0)
    return (c == needle[0]) ? 1 : 0;

  if(needle[next[i]] == c)
    return next[i] + 1;

  else
    return foo(next[i], c);
}

void process()
{
  next[0] = 0;
  next[1] = 0;
  for(int i = 2; i <= needle.size(); i++)
    next[i] = foo(i-1, needle[i-1]);
}

int
main(void)
{
  int n;
  while(scanf("%d", &n) != EOF)
  {
    cin >> needle;
    scanf(" ");
    char c;
    process();
    int match = 0, i = 0;
    while((c = getchar()) != '\n')
    {
      while(c != needle[match] && match != 0)
        match = next[match];
      if(c == needle[match])
        match++;

      if(match == n)
        printf("%d\n", i - n + 1);
      i++;
    }
    printf("\n");
  }
}
```

1.7 LCA

```
Complexidade: O(n log n)
Descricao: Calcula LCA entre dois vértices de uma árvore
com pre-processamento

//http://www.spoj.com/problems/QTREE2/

int n;
vector<int> graph[11234];
vector<int> cost[11234];
long long dist[11234];
int l[11234];
int p[20][11234];
int log2 = 0;

void dfs(int a, int parent, long long c, int h)
{
  p[0][a] = parent;
  dist[a] = c;
  l[a] = h;
  for(int i = 0; i < graph[a].size(); i++)
    if(graph[a][i] != parent)
      dfs(graph[a][i], a, c + cost[a][i], h+1);
}

void process()
{
  for(int i = 1; (1<<i) <= n; log2 = i, i++)
    for(int j = 1; j <= n; j++)
      p[i][j] = p[i-1][p[i-1][j]];
}

int nthp(int a, int x)
{
  for(int i = 0; x; i++, x >>= 1)
    if(x&1)
      a = p[i][a];
  return a;
}

int lca(int a, int b)
{
  if(l[a] < l[b])
    swap(a,b);
  int x = l[a]-l[b];
  a = nthp(a, x);

  if(a == b) return a;

  for(int i = log2; i >= 0; i--)
    if(p[i][a] != p[i][b])
    {
      a = p[i][a];
      b = p[i][b];
    }
  return p[0][a];
}

int
main(void)
{
  char type[20];
  int t, a, b, c, k;
  scanf("%d", &t);
  while(t--)
  {
    scanf("%d", &n);
    log2 = 0;
    for(int i = 1; i <= n; i++)
    {
      graph[i].clear();
      cost[i].clear();
    }
    for(int i = 1; i < n; i++)
    {
      scanf("%d %d %d", &a, &b, &c);
      graph[a].push_back(b);
      cost[a].push_back(c);
      graph[b].push_back(a);
      cost[b].push_back(c);
    }
    dfs(1,1,0,0);
    process();
    while(scanf(" %s", type) && type[1] != 'O')
    {
      if(type[0] == 'D')
      {
        scanf("%d %d", &a, &b);
        int x = lca(a,b);
        printf("%lld\n", (dist[a]-dist[x])+(dist[b]-dist[x]));
      }
      else
      {
        scanf("%d %d %d", &a, &b, &k);
        int x = lca(a,b);
        k--;
        if(l[a]-l[x] >= k)
          printf("%d\n", nthp(a, k));
        else
          printf("%d\n", nthp(b, (l[a]-l[x])+(l[b]-l[x]) - k));
      }
    }
    printf("\n");
  }
  return 0;
}
```

1.8 Max flow

```
Complexidade: O(VE^2)
Descricao: Max flow

int a[1123], b[1123];

struct edge{
  int dest, cap, re;
  edge(int x, int y, int z) : dest(x), cap(y), re(z){}
};

vector<edge> graph[1123];

void put_edge(int u, int v, int cap)
{
  graph[u].push_back(edge(v, cap, graph[v].size()));
  graph[v].push_back(edge(u, 0, graph[u].size() - 1));
}

bool in[1123];
int p[1123];
int e_used[1123];

int bfs(int s, int t)
{
  queue<int> q;
  q.push(s);
  memset(in, 0, sizeof(in));
  in[s] = true;
  while(!q.empty())
  {
    int a = q.front();
    q.pop();
    if(a == t)
      return true;
    for(int i = 0; i < graph[a].size(); i++)
      if(graph[a][i].cap > 0 && !in[graph[a][i].dest])
      {
        q.push(graph[a][i].dest);
        in[graph[a][i].dest] = true;
        p[graph[a][i].dest] = a;
        e_used[graph[a][i].dest] = i;
      }
  }
  return false;
}

int flow(int s, int t)
{
  int retv = 0;
  while(bfs(s, t))
  {
    int x = 0x3f3f3f3f;
    for(int i = t; i != s; i = p[i])
      x = min(x, graph[p[i]][e_used[i]].cap);
    for(int i = t; i != s; i = p[i]) {
      graph[p[i]][e_used[i]].cap -= x;
      graph[i][graph[p[i]][e_used[i]].re].cap += x;
    }
    retv += x;
  }
  return retv;
}

int ans[1123][1123];

int main(void)
{
  int n, m, u, v, expectedB = 0, expectedA = 0;
  scanf("%d %d", &n, &m);
  for(int i = 1; i <= n; i++)
    scanf("%d", &a[i]);
```

```
  for(int i = 1; i <= n; expectedB += b[i], expectedA += a[i], i++)
    scanf("%d", &b[i]);
  for(int i = 1; i <= n; i++)
  {
    put_edge(0, 2*i, a[i]);
    put_edge(2*i, 1, b[i]);
    put_edge(2*i, 2*i + 1, a[i]);
  }

  for(int i = 0; i < m; i++)
  {
    scanf("%d %d", &u, &v);
    put_edge(2*u + 1, 2*v, a[u]);
    put_edge(2*v + 1, 2*u, a[v]);
  }

  if(flow(0, 1) == expectedA && expectedA == expectedB)
  {
    printf("YES\n");
    for(int i = 1; i <= n; i++)
      for(int j = 0; j < graph[2*i + 1].size(); j++)
        ans[i][graph[2*i + 1][j].dest / 2] += a[i] - graph[2*i + 1][j].cap;
    for(int i = 1; i <= n; printf("\n"), i++)
    {
      printf("%d", ans[i][1]);
      for(int j = 2; j <= n; j++)
        printf(" %d", ans[i][j]);
    }
  }
  else
  {
    printf("NO\n");
  }

}
```

1.9 Merge Sort Tree

```
Complexidade: O(n log n)
Descricao: Constroi a mergesort tree para responder
queries sobre um intervalo se ele estivesse ordenado.

#define MAXN 112345
#define left(i) ((i)<<1)
#define right(i) (((i) << 1) + 1)
typedef pair<int,int> pii;

vector<int> val[4 * MAXN];
pii a[MAXN];
int ans[MAXN];

void build(int id, int l, int r)
{
  if(l == r)
    val[id].push_back(a[l].second);
  else
  {
    int mid = (l+r)/2;
    build(left(id), l, mid), build(right(id), mid+1, r);
    val[id] = vector<int>(r-l+1);
    merge(val[left(id)].begin(), val[left(id)].end(),
      val[right(id)].begin(), val[right(id)].end(), val[id].begin());
  }
}

int count_interval(int id, int a, int b) {
  return (upper_bound(val[id].begin(), val[id].end(), b)
    - lower_bound(val[id].begin(), val[id].end(), a));
}

int get(int id, int l, int r, int a, int b, int x)
{
  if(l == r)
    return ans[val[id][0]];
  int mid = (l+r)/2;
  if(count_interval(left(id), a, b) >= x)
    return get(left(id), l, mid, a, b, x);
  else
    return get(right(id), mid+1, r, a, b, x
      - count_interval(left(id), a, b));
}
int main(void)
{
  int n, q, x, y, z;
  scanf("%d %d", &n, &q);
  for(int i = 0; i < n; ans[i] = a[i].first, a[i].second = i, i++)
    scanf("%d", &a[i].first);
  sort(a, a+n);
  build(1, 0, n-1);
  for(int i = 0; i < q; i++)
  {
    scanf("%d %d %d", &x, &y, &z);
    printf("%d\n", get(1, 0, n-1, x-1, y-1, z));
  }
}
```

1.10 Min cost max flow

```
Complexidade: O(VE^2)
Descricao: Encontra o fluxo máximo de custo mínimo

//http://www.spoj.com/problems/SPHIWAY/
struct edge{
  int dest, cap, re, cost;
  edge(int x, int y, int z, int w) : dest(x),
  cap(y), re(z), cost(w){}
};

vector<edge> graph[112345];

bool been[11234];
int p[112345];
int e_used[112345];
long long dist[11234];
typedef pair<long long, int> pli;
#define inf 0x3f3f3f3f3f3f3f3fLL

bool dijkstra(int s, int t) // not dijkstra
{
  memset(been, 0, sizeof(been));
  memset(dist, 0x3f, sizeof(dist));
  queue<int> q;
  dist[s] = 0;
  p[s] = s;
  q.push(s);
  bool retv = false;
  while(!q.empty())
  {
    int a = q.front();
    q.pop();
    been[a] = false;
    if(a == t)
      retv = true;
    for(int i = 0; i < graph[a].size(); i++)
      if(graph[a][i].cap
        && dist[graph[a][i].dest] > dist[a] + graph[a][i].cost)
      {
        dist[graph[a][i].dest] = dist[a] + graph[a][i].cost;
        p[graph[a][i].dest] = a;
        e_used[graph[a][i].dest] = i;
        if(!been[graph[a][i].dest])
          q.push(graph[a][i].dest);
        been[graph[a][i].dest] = true;
      }
  }
  return retv;
}

pli mincost_maxflow(int s, int t)
{
  pli retv = pli(0,0);
  while(dijkstra(s, t))
  {
    retv.first += dist[t];
    int x = 0x3f3f3f3f;
    for(int i = t; p[i] != i; i = p[i])
      x = min(x, graph[p[i]][e_used[i]].cap);
    for(int i = t; p[i] != i; i = p[i]) {
      graph[p[i]][e_used[i]].cap -= x;
      graph[i][graph[p[i]][e_used[i]].re].cap += x;
    }
    retv.second += x;
  }
  return retv;
}

int main(void)
{
  int n, m, s, t, a, b, c;
  scanf("%d %d %d %d", &n, &m, &s, &t);
  for(int i = 0; i < m; i++)
  {
    scanf("%d %d %d", &a, &b, &c);
    graph[a].push_back(edge(b, 1, graph[b].size(), c));
    graph[b].push_back(edge(a, 0, graph[a].size() - 1, -1*c));

    graph[b].push_back(edge(a, 1, graph[a].size(), c));
    graph[a].push_back(edge(b, 0, graph[b].size() - 1, -1*c));
  }
```

```
  graph[0].push_back(edge(s, 2, graph[s].size(), 0));
  graph[s].push_back(edge(0, 0, graph[0].size() - 1, 0));

  pli ans = mincost_maxflow(0, t);
  if(ans.second == 2)
    printf("%lld\n", ans.first);
  else
    printf("-1\n");
}
```

1.11 Otimização de pd - busca binária

```
Complexidade: O(n^2 log n)
Descricao: Otimização de pd busca binaria

int a[1123];
int cost[1123][1123];
int tab[1123][1123];

int pd(int n, int k)
{
  if(n == 0)
    return 0;
  if(k == 0)
    return cost[1][n];
  int retv = inf;
  for(int i = 0; i < n; i++)
    retv = min(retv, pd(i, k-1) + cost[i+1][n]);
  return retv;
}

void compute(int k, int l, int r, int a, int b)
{
  if(r < l)
    return;
  int mid = (l+r) / 2;
  int opt = -1;
  for(int i = a; i <= b && i < mid; i++)
    if(tab[i][k-1] + cost[i+1][mid] < tab[mid][k])
      tab[mid][k] = tab[i][k-1] + cost[i+1][mid], opt = i;

  if(l != r)
    compute(k, l, mid - 1, a, opt), compute(k, mid + 1, r, opt, b);
}

int main(void)
{
  int n, m;
  while(scanf("%d %d", &n, &m) && n)
  {
    for(int i = 1; i <= n; a[i] += a[i-1], i++)
      scanf("%d", &a[i]);

    for(int i = 1; i <= n; i++)
      for(int j = i + 1; j <= n; j++)
        cost[i][j] = cost[i][j-1] + ((a[j-1] - a[i-1]) * (a[j] - a[j-1]));

    memset(tab, 0x3f, sizeof(tab));
    for(int i = 0; i <= m; i++)
      tab[0][i] = 0;

    for(int i = 1; i <= n; i++)
      tab[i][0] = cost[1][i];

    for(int i = 1; i <= m; i++)
      compute(i, 1, n, 0, n-1);
    printf("%d\n", tab[n][m]);
  }
}
```

1.12 Par de pontos mais próximos

```
Complexidade: O(n log n)
Descricao: Encontra o par de pontos mais próximos em um plano.

struct point{
  int x, y, id;

  bool operator < (point a) const
  {
    return x < a.x;
  }
};

point p[51234];
point tmp[51234];

double ans = 1e8;
int idx[2];

void update(point i, point j)
{
  double dist = sqrt((i.x-j.x)*1.0*(i.x - j.x)
    + (i.y-j.y)*1.0*(i.y - j.y));
  if(dist < ans)
  {
    ans = dist;
    idx[0] = min(i.id,j.id);
    idx[1] = max(i.id,j.id);
  }
}

bool comp(point a, point b)
{
  return a.y < b.y;
}

int abs(int x) { return x < 0 ? -x : x; }

void solve(int l, int r)
{
  if(r - l <= 3)
  {
    for(int i = l; i <= r; i++)
      for(int j = i + 1; j <= r; j++)
        update(p[i], p[j]);
    sort(p + l, p + r + 1, comp);
    return;
  }
  int mid = (l+r)/2;
  int xmid = p[mid].x, sz = 0;

  solve(l, mid), solve(mid+1, r);
  merge(p + l, p + mid + 1, p + mid + 1, p + r + 1, tmp, comp);
  copy(tmp, tmp + r - l + 1, p + l);
  for(int i = l; i <= r; i++)
    if(abs(p[i].x - xmid) < ans)
    {
      for(int j = sz - 1; j >= 0 && p[i].y - tmp[j].y < ans; j--)
        update(p[i], tmp[j]);
      tmp[sz++] = p[i];
    }
}

int main(void)
{
  int n;
  scanf("%d", &n);
  for(int i = 0; i < n; p[i].id = i, i++)
    scanf("%d %d", &p[i].x, &p[i].y);
  sort(p, p+n);
  solve(0, n-1);
  printf("%d %d %.6lf\n", idx[0], idx[1], ans);
}
```

1.13 Pontos de articulação e pontes

```
Complexidade: O(n + m)
Descricao: Encontra pontos de articulações e pontes, faz algumas coisas

set<pii > bridges;
vector<int> graph[112345];
bool art[112345], been[112345], check[112345];
int low[112345], num[112345], id = 1, comp[112345], out[112345];
vector<int> val[112345];
vector<int> rdm[112345];

void dfs(int a, int p)
{
  low[a] = num[a] = id++;
  been[a] = true;
  val[a].push_back(num[a]);
  rdm[a].push_back(0);
  for(int i = 0; i < graph[a].size(); i++)
  {
    if(!been[graph[a][i]])
    {
      comp[a]++;
      dfs(graph[a][i], a);
      val[a].push_back(out[graph[a][i]]);
      low[a] = min(low[a], low[graph[a][i]]);
      if(a != 1 && low[graph[a][i]] >= num[a])
      {
        art[a] = true;
        rdm[a].push_back(comp[a]);
      }
      else if(a == 1)
        rdm[a].push_back(comp[a]);
      else
        rdm[a].push_back(0);
      if(low[graph[a][i]] > num[a])
      {
```

```
        check[a] = check[graph[a][i]] = true;
        bridges.insert(pii(min(a,graph[a][i]), max(a,graph[a][i])) );
      }
    }
    else if(graph[a][i] != p && num[graph[a][i]] < low[a])
      low[a] = num[graph[a][i]];
  }
  if(a == 1 && comp[a] > 1)
    art[a] = true;
  out[a] = id-1;
}

int find(int a, int b)
{
  int c = (lower_bound(val[a].begin(), val[a].end(), num[b]) - val[a].begin());
  if(c == val[a].size() || c == 0)
    return 0;
  return rdm[a][c];
}

char ans[1523456];

int
main(void)
{
  int n, m, a, b, c, d, tp, q;
  scanf("%d %d", &n, &m);
  for(int i = 0; i < m; i++)
  {
    scanf("%d %d", &a, &b);
    graph[a].push_back(b);
    graph[b].push_back(a);
  }
  dfs(1, 1);
  scanf("%d", &q);
  int l = 0;
  for(int i = 0; i < q; i++)
  {
    scanf("%d", &tp);
    if(tp == 1)
    {
      scanf("%d %d %d %d", &a, &b, &c, &d);
      if(check[c] == false || check[d] == false || bridges.find(pii(min(c,d), max(c,d))) == bridges.end())
      {
        strcat(ans+(l),"yes\n");
        l+= 4;
        continue;
      }
      if(num[c] > num[d])
        swap(c,d);
      int x = 0, y = 0;
      if(num[a] >= num[d] && num[a] <= out[d])
        x = 1;
      if(num[b] >= num[d] && num[b] <= out[d])
        y = 1;
      if(y == x)
      {
        strcat(ans+l,"yes\n");
        l+= 4;
      }
      else
      {
        strcat(ans+l,"no\n");
        l+= 3;
      }
    }
    else
    {
      scanf("%d %d %d", &a, &b, &c);
      if(!art[c])
      {
        strcat(ans+l,"yes\n");
        l+= 4;
        continue;
      }
      if(find(c, a) != find(c, b))
      {
        strcat(ans+l,"no\n");
        l+= 3;
      }
      else
      {
        strcat(ans+l,"yes\n");
        l+= 4;
      }
    }
  }
  printf("%s",ans);
}
```

1.14 Rotating Calipers

```
Complexidade: O(n log n)
Descricao: Encontrar pontos antipodais em um polígono convexo.
Nesse caso usado para par de pontos mais distantes.

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

typedef long long ll;
typedef pair<ll,ll> pll;
#define x first
#define y second

vector<pll> v;
vector<pll> h;

ll cross(pll o, pll a, pll b) {
  return (a.x - o.x) * (b.y - o.y) - (a.y - o.y) * (b.x - o.x);
}

ll sqr(ll x) { return x*x; }

// máximo da distância ao quadrado de a<->opo ou b<->opo
ll dist(pll a, pll b, pll opo) {
  return max(sqr(a.x - opo.x) + sqr(a.y - opo.y),
      sqr(b.x - opo.x) + sqr(b.y - opo.y));
}

int main()
{
  int t;
  scanf("%d", &t);
  while(t--) {
    v.clear();
    int n;
    scanf("%d", &n);

    ll a,b;
    for (int i = 0; i < n; i++)
      scanf("%lld %lld", &a, &b), v.push_back({a,b});

    if (n == 1) { printf("%d\n", 0); continue;}

    // ordena os pontos pelo x e depois pelo y
    sort(v.begin(), v.end());
    h.clear();

    // acha o convex hull (upper hull + lower hull) (inclue pontos coli
    int cnt = 0;
    for (int i = 0; i < n; i++) {
      while (cnt > 1 && cross(h[cnt-2], h[cnt-1], v[i]) >= 0)
        h.pop_back(), cnt--;
      h.push_back(v[i]); cnt++;
    }

    int cnt0 = cnt;

    for (int i = n-2; i >= 0; i--) {
      while (cnt - cnt0 > 0 && cross(h[cnt-2], h[cnt-1], v[i]) >= 0)
        h.pop_back(), cnt--;
      h.push_back(v[i]); cnt++;
    }

    // faz rotating calipers e atualiza resposta
    ll ans = 0;
    int opo = 1;
    // itera pelos pontos:
    for (int i = 1; i < h.size(); i++) {
      // atualiza a reta oposta (par de pontos que definem a reta);
      while (1) {
        int next = opo+1 < h.size() ? opo+1 : 1;
        if (dist(h[i-1], h[i], h[opo]) <= dist(h[i-1], h[i], h[next]))
          opo = next;
        else
          break;
      }

      ans = max(ans, dist(h[i], h[i-1], h[opo]));
    }

    printf("%lld\n", ans);
  }
}
```

1.15 Segment Tree 2D

```
Complexidade: O(n log^2(MAX_COORDENADA))
Descricao: Implementa uma seg 2D com update em uma célula e
```

```
query em um retângulo

int n, m;

long long join(long long x, long long y) {
  return x+y;
}

struct nodes {
  long long val;
  nodes *left, *right;
  nodes() : left(NULL), right(NULL), val (0) {}

  void update(int l, int r, int a, long long x)
  {
    if(l == r)
      val = x;
    else
    {
      int mid = (l+r)/2;
      if(a <= mid)
      {
        if(left == NULL) left = new nodes;
        left->update(l, mid, a, x);
      }
      else
      {
        if(right == NULL) right = new nodes;
        right->update(mid+1, r, a, x);
      }
      val = join(left ? left->val : 0, right ? right->val : 0);
    }
  }

  void updateb(int l, int r, int a, long long x, nodes *o, nodes *p)
  {
    if(l == r)
      val = join(o ? o->val : 0, p ? p->val : 0);
    else
    {
      int mid = (l+r)/2;
      if(a <= mid)
      {
        if(left == NULL) left = new nodes;
        left->updateb(l, mid, a, x, o ? o->left : NULL, p ? p->left : NULL);
      }
      else
      {
        if(right == NULL) right = new nodes;
        right->updateb(mid+1, r, a, x, o ? o->right : NULL, p ? p->right : NULL);
      }
      val = join(o ? o->val : 0, p ? p->val : 0);
    }
  }

  long long get(int l, int r, int a, int b)
  {
    if(l == a && r == b)
      return val;
    else
    {
      int mid = (l+r)/2;
      if(b <= mid)
        return left ? left->get(l, mid, a, b) : 0;
      else if(a > mid)
        return right ? right->get(mid+1, r, a, b) : 0;
      else
        return join(left ? left->get(l, mid, a, mid) : 0, right ? right->get(mid+1, r, mid+1, b) : 0);
    }
  }
};

struct nodef {
  nodes *val;
  nodef *left, *right;
  nodef() : left(NULL), right(NULL) { val = new nodes; }

  void update(int l, int r, int a, int b, long long x)
  {
    if(l == r)
      val->update(0, m, b, x);
    else
    {
      int mid = (l+r)/2;
      if(a <= mid)
      {
        if(left == NULL) left = new nodef;
        left->update(l, mid, a, b, x);
      }
      else
      {
        if(right == NULL) right = new nodef;
        right->update(mid+1, r, a, b, x);
      }
      val->updateb(0, m, b, x, left ? left->val : NULL, right ? right->val : NULL);
    }
  }

  long long get(int l, int r, int a, int b, int c, int d)
  {
    if(l == a && r == b)
      return val->get(0, m, c, d);
    else
    {
      int mid = (l+r)/2;
      if(b <= mid)
        return left ? left->get(l, mid, a, b, c, d) : 0;
      else if(a > mid)
        return right ? right->get(mid+1, r, a, b, c, d) : 0;
      else
        return join(left ? left->get(l, mid, a, mid, c, d) : 0, right ? right->get(mid+1, r, mid+1, b, c, d) : 0);
    }
  }
};

int main(void)
{
  long long a;
  nodef *root = new nodef;
  int q, tp, x, y, z, w;
  scanf("%d %d %d", &n, &m, &q);
  while(q--)
  {
    scanf("%d", &tp);
    if(tp == 1)
    {
      scanf("%d %d %lld", &x, &y, &a);
      root->update(0, n, x, y, a);
    }
    else
    {
      scanf("%d %d %d %d", &x, &y, &z, &w);
      printf("%lld\n", root->get(0, n, x, z, y, w));
    }
  }
}
```

1.16 Treap

```
Complexidade: O(n log n)
Descricao: Treap com inserção, remoção e encontrar k-ésimo.

struct node{
  int val, p, num;
  node *left, *right;

  node(int _val) : val(_val), p(rand()),
  num(1), left(NULL), right(NULL) {}
};

int get_num(node *root) {
  return (root == NULL) ? 0 : root->num;
}

void update_num(node *root) {
  root->num = get_num(root->left) + get_num(root->right) + 1;
}

node *rotate_left(node *root) {
  node *a = root;
  node *b = root->right;

  a->right = b->left;
  b->left = a;
  update_num(a); update_num(b);
  return b;
}

node *rotate_right(node *root) {
  node *a = root;
  node *b = root->left;

  a->left = b->right;
  b->right = a;
  update_num(a); update_num(b);
  return b;
}

node *insert(node *root, int x) {
  if(root == NULL)
```

```
    return new node(x);
  if(x > root->val)
    root->right = insert(root->right, x);
  else if(x < root->val)
    root->left = insert(root->left, x);

  update_num(root);

  if(root->right && root->right->p > root->p)
    root = rotate_left(root);
  if(root->left && root->left->p > root->p)
    root = rotate_right(root);
  return root;
}

node *remove(node *root, int x) {
  if(root == NULL)
    return NULL;
  if(x > root->val)
    root->right = remove(root->right, x);
  else if(x < root->val)
    root->left = remove(root->left, x);
  else if(root->left == NULL)
    root = root->right;
  else if(root->right == NULL)
    root = root->left;
  else if(root->left->p > root->right->p)
  {
    root = rotate_right(root);
    root->right = remove(root->right, x);
  }
  else
  {
    root = rotate_left(root);
    root->left = remove(root->left, x);
  }
  if(root)
    update_num(root);
  return root;
}

int kth(node *root, int k) {
  if(get_num(root->left) >= k)
    return kth(root->left, k);
  else if(get_num(root->left) + 1 == k)
    return root->val;
  else
    return kth(root->right, k - get_num(root->left) - 1);
}

int count(node *root, int x) {
  if(root == NULL)
    return 0;
  if(x < root->val)
    return count(root->left, x);
  else if(x == root->val)
    return get_num(root->left);
  else
    return get_num(root->left) + 1 + count(root->right, x);
}
```

```
int main(void) {
  int q, x; char tp;
  node *root = NULL;
  scanf("%d", &q);
  while(q-- && scanf(" %c %d", &tp, &x))
  {
    if(tp == 'I')
      root = insert(root, x);
    else if(tp == 'D')
      root = remove(root, x);
    else if(tp == 'C')
      printf("%d\n", count(root, x));
    else
    {
      if(get_num(root) < x)
        printf("invalid\n");
      else
        printf("%d\n", kth(root, x));
    }
//      printf("size: %d\n", get_num(root));
  }
}
```

1.17 Trie

```
Complexidade: O(n * alfabeto)
Descricao: Implementação Trie.

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

int n, count = 0;

struct node{
node *next[27];
bool end;

void garante(char c)
{
if(next[c-'a'] == NULL)
{
count++;
next[c-'a'] = new node;
}
}

void insert(char s[], int i)
{
if(s[i] == '\0')
end = true;
else
{
garante(s[i]);
next[s[i]-'a']->insert(s, i+1);
}
}

void dfs(char s[], int i)
{
if(end == true)
printf("P\n");
for(char c = 'a'; c <= 'z'; c++)
if(next[c-'a'] && c != s[i])
{
printf("%c\n", c);
next[c-'a']->dfs(s, 30);
printf("-\n");
}
if(s[i])
{
printf("%c\n", s[i]);
next[s[i]-'a']->dfs(s,i+1);
}
}

};

char avoid[33];

int
main(void)
{
node *root = new node;
char s[30];
int tmp = 0;
scanf("%d", &n);
for(int i = 0; i < n; i++)
{
scanf(" %s", s);
root->insert(s, 0);
if(strlen(s) > tmp)
{
tmp = strlen(s);
strcpy(avoid, s);
}
}
printf("%d\n", n + 2*count - tmp);
root->dfs(avoid, 0);
}
```

2 Drazy

2.1 Articulation Points and Bridges

```
Complexidade: O(n+m)

const int MAXN = 100010;
vi adj[MAXN];
int d[MAXN], low[MAXN], visi[MAXN], t;
bool art[MAXN];
set<pii> bs;

//call with FOR0(i,n) if (!visi[i]) dfs(i);
void dfs(int u, int p=-1) {
  visi[u] = true;
  d[u] = low[u] = t++;
  bool found = false;
  int ct=0;
  for(auto v : adj[u]) {
```

```
    if(!visi[v]) {
      ++ct;
      dfs(v,u);
      low[u] = min(low[u],low[v]);
      if(low[v]>=d[u]) found = true;
      if(low[v]>d[u]) bs.insert(mp(min(u,v),max(u,v)));
    }
    else if(v!=p) {
      low[u] = min(low[u], d[v]);
    }
  }
  art[u] = (u ? found : ct>1);
}
```

2.2 BIT with O(log n) Binary Search

```
Complexidade: O(n+m)

int tree[MAXN];
void update(int idx, int val) {
  while(idx <= n) {
    tree[idx] += val;
    idx += (idx & -idx);
  }
}
int read(int idx) {
  int sum = 0;
  while(idx > 0) {
    sum += tree[idx];
    idx -= (idx & -idx);
  }
  return sum;
}

int findG(int cumFre){
  int idx = 0, bitMask = 1;
  for (; (bitMask<<1) <=n; bitMask <<= 1);
  while ((bitMask != 0) && (idx < n)){
    int tIdx = idx + bitMask;
    if (cumFre >= tree[tIdx]){
      idx = tIdx;
      cumFre -= tree[tIdx];
    }
    bitMask >>= 1;
  }
  return (cumFre != 0 ? -1 : idx);
}
```

2.3 Centroid Decomposition

```
Complexidade: O(nlogn)

vi adj[MAXN];
int rnd[MAXN], qtd[MAXN];

int dfs(int src, int par) {
  int ans = 1;
  for (int x : adj[src]) if (x!=par && !rnd[x])
    ans += dfs(x, src);
  return qtd[src] = ans;
}

void solve(int src, int crnd) {
  int ctr = src;
  int par = -1;
  int tgt = dfs(src, -1)/2;
  while (true) {
    bool found = false;
    for (int x : adj[ctr]) if (x!=par && !rnd[x] && qtd[x]>tgt) {
      par = ctr;
      ctr = x;
      found = true;
      break;
    }
    if (!found) break;
  }
  rnd[ctr] = crnd;
  for (int x : adj[ctr]) if (!rnd[x]) solve(x,crnd+1);
}
```

2.4 Divide and Conquer DP

```
Complexidade: O(nlogn)

const ll INFINITE = 1e17;

ll dp[8010][810], v[8010];
int n;

ll cost(int x, int y) {
  if(x>y) return 0;
  ll sum = v[y] - (x ? v[x-1] : 0LL);
  return sum*(y-x+1);
}

void fill(int g, int lmin, int lmax, int pmin, int pmax) {
  if(lmin>lmax) return;
  int lmid = (lmin+lmax)/2;
  dp[lmid][g] = INFINITE;
  ll minp;
  FOR(i,pmin, pmax+1) {
    ll ncost = dp[i][g-1] + cost(i+1,lmid);
    if(ncost < dp[lmid][g]) {
      dp[lmid][g] = ncost;
      minp = i;
    }
  }
  fill(g,lmin,lmid-1,pmin,minp);
  fill(g,lmid+1,lmax,minp,pmax);
}
```

2.5 Dual String Hash (with iseq and isless)

```
Complexidade: O(n + logn)

const int MAXN = 5000;
const int QTD = 2;
int B[] = {31,97};
int MODH[] = {1000000007,1000000009};
int pwr[MAXN][QTD];
int h[MAXN][QTD];
string s;
int n;

void precalc() {
  FOR0(j,QTD) {
    h[0][j] = s[0]%MODH[j];
    pwr[0][j] = 1%MODH[j];
    FOR(i,1,n) {
      h[i][j] = (h[i-1][j]*1LL*B[j] + s[i])%MODH[j];
      pwr[i][j] = pwr[i-1][j]*1LL*B[j]%MODH[j];
    }
  }
}

void calc(int x, int y, int ans[QTD]) {
  if (x>y) return;
  FOR0(j,QTD) {
    ans[j] =(h[y][j]-(x?h[x-1][j]:0)*1LL*pwr[y-x+1][j])%MODH[j];
    if (ans[j] < 0) ans[j] += MODH[j];
  }
}

bool is_eq(int x1, int y1, int x2, int y2) {
  int aux1[QTD], aux2[QTD];
  calc(x1,y1,aux1);
  calc(x2,y2,aux2);
  FOR0(i,QTD) if (aux1[i] != aux2[i]) return false;
  return true;
}

bool is_less(int x1, int y1, int x2, int y2) {
  int sz1 = y1-x1+1, sz2 = y2-x2+1;
  int beg = 0, end = min(sz1,sz2);
  while (beg < end) {
    int mid = (beg+end)/2;
    if (!is_eq(x1,x1+mid, x2, x2+mid)) end = mid;
    else beg = mid + 1;
  }
  if (end == min(sz1,sz2)) return (sz1 < sz2);
  return s[x1+end] < s[x2+end];
}
```

2.6 FFT

```
Complexidade: O(nlogn)

const double PI = acos(-1.0);
typedef complex<double> base;

void fft(vector<base> & a, bool invert) {
  int n = (int)a.size();
  for (int i = 1, j = 0; i<n; ++i) {
    int bit = n >> 1;
    for (; j >= bit; bit >>= 1)
      j -= bit;
    j += bit;
    if (i < j)
      swap(a[i], a[j]);
  }
```

```
  for (int len = 2; len <= n; len <<= 1) {
    double ang = 2 * PI / len * (invert ? -1 : 1);
    base wlen(cos(ang), sin(ang));
    for (int i = 0; i<n; i += len) {
      base w(1);
      for (int j = 0; j<len / 2; ++j) {
        base u = a[i + j], v = a[i + j + len / 2] * w;
        a[i + j] = u + v;
        a[i + j + len / 2] = u - v;
        w *= wlen;
      }
    }
  }
  if (invert)
  for (int i = 0; i<n; ++i)
    a[i] /= n;
}

vi mult(vi a, vi b) {
  vector<base> fa(a.begin(), a.end()), fb(b.begin(), b.end());
  size_t n = 1;
  while (n < max(a.size(), b.size()))  n <<= 1;
  n <<= 1;
  fa.resize(n), fb.resize(n);
  fft(fa, false), fft(fb, false);
  for (size_t i = 0; i<n; ++i)
    fa[i] *= fb[i];
  fft(fa, true);
  vi res;
  res.resize(n);
  for (size_t i = 0; i<n; ++i)
    res[i] = int(fa[i].real() + 0.5);
  return res;
}
```

2.7 Integer Ternary Search

```
Complexidade: O(logn)

//max
int lo = -1, hi = n;
while (hi - lo > 1) {
  int mid = (hi + lo)>>1;
  if (f(mid) > f(mid + 1)) hi = mid;
  else lo = mid;
}
//ans = lo + 1
```

2.8 KMP

```
Complexidade: O(n)

int pref[MAXN];
void calc_pref(string s) {
  int n = sz(s);
  for (int i=1, j=0; i<n; ++i) {
    while (j && s[j] != s[i]) j = pref[j-1];
    if (s[i]==s[j]) ++j;
    pref[i] = j;
  }
}
```

2.9 LCA

```
Complexidade: O(nlogn)

const int MAXN = 100010;
const int LOGMAXN = 18;
vi adj[MAXN];
int par[MAXN], l[MAXN];
int p[MAXN][LOGMAXN];
int n;

void dfs(int src) {
  if(src) l[src] = l[par[src]] + 1;
  TRAV(it,adj[src]) if(par[*it]==-1) {
    par[*it] = src;
    dfs(*it);
  }
}

void preproc() {
  SET(par);
  l[0] = 0;
  par[0]=0;
  dfs(0);
  SET(p);
  for(int j=0; (1<<j)<n; ++j) FOR0(i,n) {
    if(!j) p[i][j] = par[i];
    else if(p[i][j-1]!=-1) {
      p[i][j] = p[p[i][j-1]][j-1];
    }
  }
}

int lca(int x, int y) {
  if(x==y) return x;
  if(l[x]>l[y]) swap(x,y);
  int log = 0;
  while((1<<(log+1)) <= l[y]) ++log;
  int dl = l[y] - l[x];
  if(dl) for(int i=log; i>=0; --i) if(dl&(1<<i)) y = p[y][i];
  if(x==y) return x;
  for(int i=log; i>=0; --i) if(p[x][i]!=p[y][i]) {
    x = p[x][i];
    y = p[y][i];
  }
  return p[x][0];
}
```

2.10 Maxflow

```
Complexidade: O(n^3)

const int NN = 250;
int cap[NN][NN], deg[NN], adj[NN][NN];
int q[NN], prv[NN];

int dinic(int n, int s, int t) {
  int flow = 0;
  CLR(deg);
  FOR0(u,n) FOR0(v,n) {
    if(cap[u][v] || cap[v][u]) adj[u][deg[u]++] = v;
  }
  while(true) {
    SET(prv);
    int qf = 0, qb = 0;
    prv[q[qb++] = s] = -2;
    while(qb > qf && prv[t] == -1)
      for(int u = q[qf++], i = 0, v; i < deg[u]; i++)
        if( prv[v = adj[u][i]] == -1 && cap[u][v] )
          prv[q[qb++] = v] = u;
    if( prv[t] == -1 ) break;
    FOR0(z,n) if( cap[z][t] && prv[z] != -1 ) {
      int bot = cap[z][t];
      for(int v = z, u = prv[v]; u >= 0; v = u, u = prv[v])
        bot = min(bot, cap[u][v]);
      if(!bot) continue;
      cap[z][t] -= bot;
      cap[t][z] += bot;
      for(int v = z, u = prv[v]; u >= 0; v = u, u = prv[v]) {
        cap[u][v] -= bot;
        cap[v][u] += bot;
      }
      flow += bot;
    }
  }
    return flow;
}
```

2.11 Mincost Maxflow

```
const ll LINF = 0x3f3f3f3f3f3f3f3fLL;
const int MAXN = 3050;

struct edge {
  // flow in the edge = orcap - cap
  int viz, cap, orcap, cost, dual;
  edge(int viz, int cap, int cost, int dual) : viz(viz),
  cap(cap), orcap(cap), cost(cost), dual(dual) {}
};

vector<edge> g[MAXN];
ll d[MAXN];
int p[MAXN], p_edge[MAXN], color[MAXN];
ll flow, fcost;

void add_edge(int x, int y, int cap, int cost) {
  g[x].pb(edge(y, cap, cost, (int)g[y].size()));
  g[y].pb(edge(x, 0, -cost, (int)g[x].size() - 1));
}

int SPFA(int s, int t, int n) {
  const int WHITE = 0, GRAY = 1;
  int next, viz, cap, cost;
  queue<int> fila;
  FOR0(i,n) {
    d[i] = LINF;
    color[i] = WHITE;
```

```
  }
  d[s] = 0;
  fila.push(s);
  while (!fila.empty()) {
    next = fila.front();
    fila.pop();
    color[next] = WHITE;
    FOR0(i,sz(g[next])) {
      viz = g[next][i].viz;
      cost = g[next][i].cost;
      cap = g[next][i].cap;
      if (cap && d[viz] > d[next] + cost) {
        d[viz] = d[next] + cost;
        p[viz] = next;
        p_edge[viz] = i;
        if (color[viz] == WHITE) {
          color[viz] = GRAY;
          fila.push(viz);
        }
      }
    }
  }
  return d[t] != LINF;
}

void mcmf(int s, int t, int n) {
  ll augment;
  int idx, dual;
  flow = 0;
  fcost = 0;
  while (SPFA(s, t, n)) {
    augment = LINF;
    for (int v = t; v != s; v = p[v]) {
      idx = p_edge[v];
      augment = min(augment, 1LL*g[p[v]][idx].cap);
    }
    for (int v = t; v != s; v = p[v]) {
      idx = p_edge[v];
      dual = g[p[v]][idx].dual;
      g[p[v]][idx].cap -= augment;
      g[v][dual].cap += augment;
    }
    flow += augment;
    fcost += augment*d[t];
  }
}
```

2.12 MOS

```
Complexidade: O(n*sqrtn)

const int MAXN = 100010;
const int BSZ = 350; // ~sqrt(MAXN)
int ans[MAXN], cans;

void add(int pos) {}

void rem(int pos) {}

struct node {
  int l, r, idx;
} qrs[MAXN];

void mos(int q) {
  if (!q) return;
  sort(qrs, qrs+q, [](node x, node y) {
    if (x.l/BSZ != y.l/BSZ) return x.l/BSZ < y.l/BSZ;
    return x.r < y.r;
  });
  int cl = qrs[0].l, cr = qrs[0].l;
  FOR0(i,q) {
    int l = qrs[i].l, r = qrs[i].r;
    while(cl > l) add((cl--)-1);
    while(cr <= r) add(cr++);
    while(cl < l) rem(cl++);
    while(cr > r+1) rem((cr--)-1);
    ans[qrs[i].idx] = cans;
  }
}
```

2.13 Regular Convex Hull Trick

```
Complexidade: O(n + logn)

typedef long double ld;
struct node {
  ld left, p, q;
};
vector<node> lines;

ld get_inter(ld p, ld q, ld r, ld s) { return (q-s)/(r-p); }

// lines must be inserted in strictly decreasing order of p
void insert(ld p, ld q) {
  while (sz(lines) >= 2) {
    ld r = lines[sz(lines)-2].p;
    ld s = lines[sz(lines)-2].q;
    if (lines.back().left > get_inter(p,q,r,s)) lines.pop_back();
    else break;
  }
  node nd;
  nd.p = p; nd.q = q;
  nd.left = (lines.empty() ?
            -INFINITY :
            get_inter(lines.back().p, lines.back().q, p, q));
  lines.pb(nd);
}

ld get_min(ld x) {
  node aux;
  aux.left = x;
  auto nd = *prev(upper_bound(all(lines),aux,[](node a, node b){
    return a.left < b.left;
  }));
  return nd.p*x + nd.q;
}
```

2.14 Segment Tree with Lazy Propagation

```
Complexidade: O(nlogn)

const int MAXN = 100010;
int tree[4*MAXN];
int lazy[4*MAXN];
int v[MAXN];

void build(int idx, int l, int r) {
  if(l==r) {
    tree[idx] = v[l];
  }
  else {
    int m = (l+r)/2;
    build(2*idx,l,m);
    build(2*idx+1,m+1,r);
    tree[idx] = tree[2*idx] + tree[2*idx+1];
  }
}

void go(int idx, int l, int r) {
  if(lazy[idx] != 0) {  //needs to be updated
    tree[idx] += (r-l+1)*lazy[idx];
    if(l!=r) { //not leaf
      lazy[2*idx] += lazy[idx];
      lazy[2*idx+1] += lazy[idx];
    }
    lazy[idx] = 0; //reset it
  }
}

void update(int px, int py, int val, int idx, int l, int r) {
  //it needs to come first because of the lines 64 to 66
  go(idx,l,r);
  if(py<l || px>r) return;
  if(l>=px && r<=py) {  //fully within range
    tree[idx] += (r-l+1)*val;
    if(l!=r) { //not leaf
      lazy[2*idx] += val;
      lazy[2*idx+1] += val;
    }
    return;
  }
  int m = (l+r)/2;
  update(px,py,val,2*idx,l,m);
  update(px,py,val,2*idx+1,m+1,r);
  tree[idx] = tree[2*idx] + tree[2*idx+1];
}

int query(int px, int py, int idx, int l, int r) {
  if(py<l || px>r) {
    return 0;
  }
  go(idx,l,r);
  if(l>=px && r<=py) {
    return tree[idx];
  }
  int m = (l+r)/2;
  int p1 = query(px,py,2*idx,l,m);
  int p2 = query(px,py,2*idx+1,m+1,r);
  return p1+p2;
}
```

2.15 Sieve / Phi / Integer Factorization

```
Complexidade: O(nloglogn)

const int MAXN = 10000010;
int pr[MAXN];
int divisor[MAXN];
int phi[MAXN];

void sieve(int n) {
  FOR0(i,n+1) pr[i] = true;
  pr[0] = pr[1] = false;
  for(int i=2; i*i<=n; ++i) {
    if(!pr[i]) continue;
    int k = i*i;
    while(k<=n) {
      divisor[k] = i;
      pr[k] = false;
      k += i;
    }
  }
}

void calc_phi(int n) {
  FOR(i,1,n+1) phi[i] = i;
  FOR(i,2,n+1) if(pr[i]) {
    for(int j=i; j<=n; j+=i) {
      phi[j] -= phi[j]/i;
    }
  }
}

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

int factorize(int v[], int n) {
  if(n<=1) return 0;
  int sz=0;
  while(n>1) {
    int p = get_div(n);
    v[sz++] = p;
    n/=p;
  }
  return sz;
}
```

2.16 Sparse table

```
Complexidade: O(nlogn + 1)

const int MAXN = 100010;
const int MAX_LOG = 18;
int v[MAXN];
int st[MAXN][MAX_LOG];
int logt[MAXN];
int n;

void build() {
  logt[1] = 0;
  for (int i=2, val=1; i<=n; i*=2, ++val) {
    int lim = min(2*i, n+1);
    FOR(j,i,lim) {
      logt[j] = val;
    }
  }
  FOR0(j,MAX_LOG) FOR0(i,n) if(i + (1 << j) - 1 < n) {
    st[i][j] = (j ?
      min(st[i][j-1], st[i + (1 << (j-1))][j-1]): v[i]);
  }
}

int query(int l, int r) {
  int x = logt[r-l+1];
  return min(st[l][x],st[r-(1<<x)+1][x]);
}
```

2.17 Sqrt decomposition

```
Complexidade: O(nsqrtn)
Descricao: <Pra que serve o algoritmo>

const int MAXN = 1010;
const int SQRTN = 35;
int v[MAXN];
int blocks[SQRTN + 10];
int n;

void build_sqrt() {
  SET(blocks);
  FOR0(i,n) {
    int b = i/SQRTN;
    if(blocks[b]==-1 || v[i]<v[blocks[b]]) {
      blocks[b] = i;
    }
  }
}

void update_sqrt(int x, int vx) {
  v[x] = vx;
  int b = x/SQRTN;
  if(vx < v[blocks[b]]) blocks[b] = x;
}

int query_sqrt(int x, int y) {
  int ans;
  int minv = INF;
  int i = x;
  while(i<=y && i%SQRTN) {
    if(v[i]<minv) {
      minv = v[i];
      ans = i;
    }
    ++i;
  }
  while(i+SQRTN-1 <= y) {
    int b = i/SQRTN;
    if(v[blocks[b]] < minv) {
      minv = v[blocks[b]];
      ans = blocks[b];
    }
    i += SQRTN;
  }
  while(i<=y) {
    if(v[i]<minv) {
      minv = v[i];
      ans = i;
    }
    ++i;
  }
  return ans;
}
```

2.18 Suffix Array

```
Complexidade: O(nlogn)

const int MAXN = 200010;
const int LOGMAXN = 20; // > floor(log(MAXN))
int p[LOGMAXN][MAXN];
int sa[MAXN];
int step;
pair<pii,int> vaux[MAXN];
string s;

void calcsa() {
  int n = s.size();
  FOR0(i,n) p[0][i] = s[i];
  if(n==1) p[0][0] = 0;
  int pot;
  for(step=1,pot=2; pot<2*n; ++step, pot*=2) {
    FOR0(i,n) {
      vaux[i].first.first = p[step-1][i];
      vaux[i].first.second = i+pot/2<n ? p[step-1][i+pot/2]:-1;
      vaux[i].second = i;
    }
    sort(vaux,vaux+n);
    int id = 0;
    FOR0(i,n) {
      if(i && vaux[i].first != vaux[i-1].first) ++id;
      p[step][vaux[i].second] = id;
    }
  }
  --step;
  FOR0(i,n) {
    sa[p[step][i]] = i;
  }
}
//sa[i] = index of the i-th suffix

int lcp(int x, int y) { //x,y are idx of suffix
  int n = s.size();
  if(x==y) return n-x;
  int ans = 0;
  for(int i=step; i>=0; --i) {
    if(p[i][x]==p[i][y]) {
      ans |= (1<<i);
      x += (1<<i);
      y += (1<<i);
      if(x>=n || y>=n) break;
```

```
    }
  }
  return ans;
}
```

2.19 Suffix Automaton

```
Complexidade: O(n)

// edges[i]  : the labeled edges from node i
vector<map<char,int>> edges;
// link[i]   : the parent of i
vi link;
// length[i] : the length of the longest string in the ith class
vi length;
// the index of the equivalence class of the whole string
int last;
unordered_set<int> terminals;

void build(string s) {
  // add the initial node
  edges.push_back(map<char,int>());
  link.push_back(-1);
  length.push_back(0);
  last = 0;
  FOR0(i, sz(s)){
    // construct r
    edges.push_back(map<char,int>());
    length.push_back(i+1);
    link.push_back(0);
    int r = edges.size() - 1;
    // add edges to r and find p with link to q
    int p = last;
    while(p >= 0 && edges[p].find(s[i]) == edges[p].end()) {
      edges[p][s[i]] = r;
      p = link[p];
    }
    if(p != -1) {
      int q = edges[p][s[i]];
      if(length[p] + 1 == length[q]) {
      // we do not have to split q,
      // just set the correct suffix link
        link[r] = q;
      }
      else {
        // we have to split, add q'
        edges.push_back(edges[q]); // copy edges of q
        length.push_back(length[p] + 1);
        link.push_back(link[q]); // copy parent of q
        int qq = edges.size()-1;
        // add qq as the new parent of q and r
        link[q] = qq;
        link[r] = qq;
        // move short classes pointing to q to point to q'
        while(p >= 0 && edges[p][s[i]] == q) {
          edges[p][s[i]] = qq;
          p = link[p];
        }
      }
    }
    last = r;
  }
  // finding terminals
  int p = last;
  while(p > 0) {
    terminals.insert(p);
    p = link[p];
  }
}
```

2.20 Template

```

#include <bits/stdc++.h>

using namespace std;

#define INF 0x3f3f3f3f
#define NSYNC ios::sync_with_stdio(false)
#define FOR(i,a,b) for(int i=a; i<(b); ++i)
#define FOR0(i,b) for(int i=0; i<(b); ++i)
#define DBG(x) cout << #x << " == " << x << endl
#define DBGV(v) for(int x : v) cout << x << " "; cout << endl
#define DBGP(x,y) cout << "(" << x << ", " << y << ")" << endl
#define pb(x) push_back(x)
#define mp(x,y) make_pair(x,y)
#define sz(a) (int)((a).size())
#define all(c) (c).begin(),(c).end()
#define R(x) scanf(" %d",&(x))
#define RR(x,y) scanf(" %d %d",&(x), &(y))
#define CLR(v) memset(v, 0, sizeof(v))
#define SET(v) memset(v, -1, sizeof(v))

typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;
```

2.21 Z algorithm

```
Complexidade: O(n)

const int MAXN = 1000000;
int z[MAXN];
char s[MAXN+1];
int n;
void run_z() {
  int L = 0, R = 0;
  for (int i = 1; i < n; i++) {
    if (i > R) {
      L = R = i;
      while (R < n && s[R-L] == s[R]) R++;
      z[i] = R-L; R--;
    }
    else {
      int k = i-L;
      if (z[k] < R-i+1) z[i] = z[k];
      else {
        L = i;
        while (R < n && s[R-L] == s[R]) R++;
        z[i] = R-L; R--;
      }
    }
  }
}
```

3 Grafos

3.1 Circuito e Passeio de Euler

```
Complexidade: O(n+m)
Descricao: Verifica se há e encontra um circuito/passeio
de Euler em grafo não-direcionado contendo todas arestas
podendo ser paralelas ou laços e o grafo conexo ou não

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

using namespace std;
#define MAXM 100100
#define MAXN 100100

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

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

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

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

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

  if (qi > 2) return 0;

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

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

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

    if (g[v].empty()) {
      vtour.pb(v);
      st.pop();
    }
    else {
      st.push(g[v].back());
      g[v].pop_back();
    }
  }

  if ((int)vtour.size() != m+1) return 0;
  return 1;
}
```

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

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

3.2 Cliques Maximais

```
Complexidade: O(3^(n/3))
Descricao: Acha todas as cliques maximais de um grafo.

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

typedef long long int int64;

#define MAXN 55
#define INF 0x3f3f3f3f

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

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

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

3.3 Componentes Fortemente Conexas

```
Complexidade: O(n+m)
Descricao: Encontra as componentes fortemente conexas de um
grafo orientado. Componentes nomeadas de 1 à ncomp.

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

#define MAXN 1024

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

/* Output */
int ncomp;      // numero de componentes fortemente conexas
int comp[MAXN]; // comp[i] = componente do vertice i

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

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

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

3.4 Corte Mínimo Geral (Stoer-Wagner)

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

#include <algorithm>
using namespace std;

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

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

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

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

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

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

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

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

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

      // update the weights of its neighbours
      for(int j = 1; j < n; j++)
        if(!a[v[j]])
          w[j] += adj[v[zj]][v[j]];
    }
  }
  return best;
}
```

3.5 Emparalhamento Bipartido de Custo Máximo

```
Complexidade: O(n^3)
Descricao: Encontra o emparelhamento maximo
de custo maximo, para custo minimo insira as
arestas com peso negativo. Se uma aresta nao
existe o valor na matriz deve ser -1*INF.
Cuidado: NAO UTILIZAR MEMSET PARA O -1*INF
necessariamente n <= m

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

#define INF 0x3f3f3f3f
#define MAXN 351

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

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

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

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

3.6 Emparelhamento Máximo e Cobertura Mínima

```
Complexidade: O(n*m)
Descricao: Encontra um emparelhamento máximo e uma cobertura
de vértice mínima em grafo bipartido.
Para um grafo bipartido:
- |cobertura mínima de vertices| = |emparelhamento máximo|
- |conj. independente máximo| = n - |emparelhamento máximo|
- |cobertura mínima de arestas| = n - |emparelhamento máximo|

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

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

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

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

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

/* Código apenas para cobertura mínima de vertice */

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

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

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

3.7 Emparelhamento Máximo Geral (Edmonds)

```
Complexidade: O(n^3)
Uso: CUIDADO - Nao utilizar o vertice 0
```

```
- Para cada aresta 'i' (sem direcao) u-v,
faca from[i] = u, to[i] = v e coloque i
na lista de adjacencia de ambos u e v.
- n e m devem ser utilizados obrigatoriamente.
- E() retorna o tamanho do emparalhamento (# de casais).
- mate[v] quando diferente de 0 indica que o vertice v esta
casado com mate[v]

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

#define MAXN 110
#define MAXM MAXN*MAXN

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

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

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

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

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

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

3.8 Fluxo Mínimo

```
Complexidade: O(m*Flow)
Descricao: Calcula o fluxo mínimo viável entre s e t,
onde cada aresta e do grafo tem um capacidade mínima lb[e] e
máxima ub[e], de forma que o fluxo: lb[e] <= f[e] <= ub[e].

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

/* Fluxo Máximo SPFA AQUI */

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

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

  for (int i = 0; i < n; i++) {
    int b = 0;
    for (int j = 0; j < n; j++) {
      b += lb[j][i];
      b -= lb[i][j];
    }
    if (b >= 0)
      add(n, i, b);
    else
      add(i, n+1, -b);
  }
  int fcost;
  mfmc(n, n+1, n+2, fcost);

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

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

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

4 Geométricos

4.1 Algoritmos Básicos para Circunferência

```
Descricao: Calcula interseção entre duas circunferência,
e a aréa dessa intersecção.

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

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

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

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

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

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

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

/* Calcula a area da intersecção entre dois círculos */
double area_inter(circ c1, circ c2) {
  if (c1.S > c2.S) swap(c1, c2);
  c2.F.x = norma(c1.F - c2.F); c2.F.y = 0;
  c1.F.x = 0; c1.F.y = 0;
  pt ia, ib;
  int it = inter_circ(ia, ib, c1, c2);
  if (it == 1) return 0.0;
  if (it == 0) {
    if (cmp(c2.F.x, c1.S + c2.S) > 0) return 0.0;
    else return  pi * c1.S * c1.S;
  }

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

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

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

  return 0;
}
```

4.2 Algoritmos Básicos para Geométricos

```
Descricao: Contem algoritmos simples para geometricos

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

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

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

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

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

  return 0;
}

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

4.3 Algoritmos de Intersecções

```
- UVa 11068 [intersect] [acha] t=0.010s
- UVa 866 [intersect_seg] [intersect_seg_2] [acha] t=0.000s
- UVa 378 [intersect] [acha] t=0.010s
- UVa 191 [intersect_seg] [intersect_seg_2] t=0.000s
- POJ 3819 [inter_reta_circ]
- comparacoes na estrutura de ponto - soh intersect_seg()
- norma() - distPR(), inter_reta_circ()
- projecao() - distPR(), inter_reta_circ()
- between3() - distPR(), intersect_seg_2(), inter_reta_circ()
- ccw() - soh intersect_seg_2()
Descricao: Determina se há intersecção ou o ponto de
intersecção entre segmentos de reta ou retas. Acha inter-
secções entre segmento de reta ou reta e circunferência.
Também contém função que devolve a distância de um ponto
a uma reta.

/**
Estruturas aqui
**/

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

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

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

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

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

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

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

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

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

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

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

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

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

  return norma(v - proj);
}

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

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

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

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

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

4.4 Diâmetro de Pontos e Polígono

```
Complexidade: O(n) polígono convexo, O(n lg n) pontos
Descrição: Calcula a maior distância entre um par de pontos
de um polígono ou de um conjunto de pontos em posição geral

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

typedef pair<int,int> pii;

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

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

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

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

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

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

  return res;
}

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

4.5 Distância Esférica

```
Complexidade: O(1)
Descricao: Calcula a distância entre 2 pontos em uma esfera

#include <cmath>

double torad;
double r = 6378;

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

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

4.6 Estrutura e Base para Geométricos

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

#include <cmath>
#include <vector>

using namespace std;

const double pi = acos(-1);

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

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

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

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

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

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

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

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

pt normal(pt v){ return pt(-v.y,v.x); }
double norma(pt v){ return hypot(v.x, v.y); }
pt versor(pt v){ return v/norma(v); }
double anglex(pt v){ return atan2(v.y, v.x); }
double angle(pt v1, pt v2){ /* angulo orientado ccw */
  return atan2(v1%v2 , v1*v2);
}
double triarea(pt a, pt b, pt c){ /* area c/ sinal */
  return ((b-a)%(c-a))/2.0; /* area>0 = ccw ; area<0 = cw */
}
int ccw(pt a, pt b, pt c){ /* b-a em relacao a c-a */
  return cmp((b-a)%(c-a)); /* ccw=1 ; cw=-1 ; colinear=0 */
  /* equivalente a cmp(triarea(a,b,c)), mas evita divisao */
}
pt projecao(pt v, pt w){ /* proj de v em w */
  double alfa = (v*w)/(w*w);
  return w*alfa;
}
```

4.7 Intersecção de Polígonos Convexos

```
Complexidade: O(n+m)
- ccw()
- between3()
- intersect_seg()
- acha()
- inpoly()
Descricao: O algoritmo devolve a interseccao de dois poligonos
convexos, orientados em sentido anti-horario.
Pode ser utilizado inpoly_convex(), sem verificacao de ponto
na borda do poligono, ja que os poligonos sao convexos.

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

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

4.8 Verificações de Ponto em Polígono

```
Complexidade: O(n), O(lg(n))
- inpoly(): uva.634
- inpoly_convex(): testes gerados na mão
- ccw()
- between3()
- intri() - soh inpoly_convex()

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

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

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

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

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

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

    if( between3(k,q,r) ) return 2;
    if( q.y>r.y ) swap(q,r);
    if( q.y<k.y && r.y>=k.y  && ccw(k,q,r)>0 ) cross++;
  }

  return cross%2;
}

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

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

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

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

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

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

5 Numéricos

5.1 Eliminação de Gauss

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

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

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

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

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

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

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

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

5.2 Euclides Extendido

```
Complexidade: O(lg x)
Descricao: Calcula um par x,y tal que a*x+b*y=mdc(a,b)

#include <algorithm>
using namespace std;

typedef pair<int,int> pii;

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

5.3 Inverso Modular

```
Complexidade: O(lg x)
Descricao: Calcula um x tal que a*x === 1 (mod M)
Para a e M coprimos, eh garantido que x eh unico
Nesse caso, pode ser usado para determinar
a divisao modular como exemplificado.

#include <algorithm>
using namespace std;

typedef pair<int,int> pii;

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

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

#include <cstdio>
```

5.4 Log Discreto

```
Complexidade: O(sqrt(m)*lg m)
Descricao: Dados a, b e m (a e m devem ser coprimos),
calcula o menor x tal que a^x = b (mod m)

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

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

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

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

5.5 Teorema Chinês do Resto

```
Complexidade: O(n * lg X)
Descricao: Resolve o conj de eqs: a[i]*x === b[i] (mod m[i])
para 0<=i<n com a restricao m[i]>1
Se a[i] == 1 para todo i, existe solucao sse
b[i]===b[j] (mod gcd(m[i],m[j])) para todo i e j

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

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

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

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

6 Miscelânea

6.1 Convex Hull Trick

```
Complexidade: O(lg n) - insert e query
Descrição: Estrutura de dados que permite inserir retas
não-verticais na forma y(x)=A*x+B e consultar, dado x*,
qual é o maior y(x*) entre as retas existentes. Para se
obter o menor no lugar do maior, troque os sinais de A,
B e do resultado da query.
Funciona para int e double.

#include <set>
#include <algorithm>
#include <cmath>
using namespace std;

// Caso queira double, troque as 2 linhas abaixo
typedef long long int T;
#define INF 0x3f3f3f3f3f3f3f3fLL

struct line {
  T a, b, xmax;
  line(T a, T b) : a(a), b(b) {};
};

bool compa(line a, line b) {
  return a.a < b.a;
}

bool compx(line a, line b) {
  return a.xmax < b.xmax;
}

bool(*fcompa)(line,line) = compa;
bool(*fcompx)(line,line) = compx;

set<line, bool(*)(line, line)> sa(fcompa);
set<line, bool(*)(line, line)> sx(fcompx);

set<line, bool(*)(line, line)>::iterator it;

/* Funcao auxiliar */
void add(line r) {
  sa.insert(r);
  sx.insert(r);
}

/* Funcao auxiliar */
void remove(line r) {
```

```
  sa.erase(r);
  sx.erase(r);
}

/* Funcao auxiliar */
T getxMax(line r, line s) {
   //return (s.b - r.b) / (r.a - s.a); // para double
  return floor((double) (s.b - r.b) / (double) (r.a - s.a));
}

/* Funcao auxiliar */
T gety(line r, T x) {
  return r.a * x + r.b;
}

void init() {
  sa.clear();
  sx.clear();
}

T query(T x) {
  if (sx.empty()) return -INF;
  line r(0, 0);
  r.xmax = x;
  it = sx.lower_bound(r);
  return gety(*it, x);
}

bool insert(T a, T b) {
  line r(a, b);
  it = sa.lower_bound(r);
  if (it != sa.end() && it->a == r.a) {
    if (it->b >= r.b) return false;
    remove(*it);
  }
  it = sa.lower_bound(r);
  if (it != sa.end() && it != sa.begin()) {
    line s = *it;
    it--;
    if (getxMax(r, s) <= getxMax(r, *it)) return false;
  }
  while (1) {
    it = sa.lower_bound(r);
    if (it == sa.end()) break;
    if (getxMax(r, *it) >= it->xmax) {
      remove(*it);
    }
    else {
      break;
    }
  }
  while (1) {
    it = sa.lower_bound(r);
    if (it == sa.begin()) break;
    it--;
    line s = *it;
    if (it == sa.begin()) {
      remove(s);
      s.xmax = getxMax(s, r);
      add(s);
      break;
    }
    it--;
    line t = *it;
    remove(s);
    if (getxMax(s, r) > t.xmax) {
      s.xmax = getxMax(s, r);
      add(s);
      break;
    }
  }
  it = sa.lower_bound(r);
  if (it == sa.end()) r.xmax = INF;
  else r.xmax = getxMax(r, *it);
  add(r);
  return true;
}
```

6.2 Decomposição Heavy Light

```
Complexidade: O(lg n) LCA / O(lg^2 n) queries
Descrição: Particiona os vértices de uma árvore em chains
(sequência de vértices ancestrais) de modo que qualquer
caminho usa um número logarítmico de chains, que podem ser
incrementadas para responder queries em caminhos (ver ex.)
```

```
#include <vector>
using namespace std;
#define MAXN 100100

vector<int> g[MAXN];
/* Vértice do topo da chain i, tam da chain i e qtd delas */
int head[MAXN], chsz[MAXN], nch;
/* Chain do vértice i e seu índice nela (cresce pra raiz) */
int chain[MAXN], chidx[MAXN];
/* Altura do vértice i, seu antecessor e tam da subárvore */
int depth[MAXN], pai[MAXN], size[MAXN];

/* Adiciona um vértice v no topo da chain c */
void chadd(int v, int c) {
  chidx[v] = chsz[c]++;
  chain[v] = c;
  head[c] = v;
}

/* Gera as chains e vetores associados */
void dfshl(int x) {
  size[x]=1;

  for (int i = 0; i < g[x].size(); i++) {
    int v = g[x][i];
    if (pai[x] != v) {
      depth[v] = depth[x]+1;
      pai[v] = x;

      dfshl(v);
      size[x] += size[v];
    }
  }

  chain[x] = -1;
  for (int i = 0; i < g[x].size(); i++)
    if (g[x][i] != pai[x] && size[g[x][i]] > size[x]/2)
      chadd(x, chain[g[x][i]]);

  if (chain[x] == -1) chadd(x, nch++);
}

/* Exemplo de LCA. Percorre as chains no caminho entre a e b
 Pode ser alterado para responder query usando uma estrutura
 de dados de intervalos por chain (por ex. BIT, segtree) */
int lca(int a, int b) {
  while (chain[a] != chain[b]) {
    if (depth[head[chain[a]]] > depth[head[chain[b]]])
      // query chain[a] em [chidx[a], chsz[chain[a]]-1]
      a = pai[head[chain[a]]];
    else
      // query chain[b] em [chidx[b], chsz[chain[b]]-1]
      b = pai[head[chain[b]]];
  }

  if (depth[a] < depth[b]) {
    // query chain[a] em [chidx[b], chidx[a]]
    return a;
  }
  // query chain[a] em [chidx[a], chidx[b]]
  return b;
}
```

6.3 Floyd Cycle Finding

```
pair<int, int> floyd(int x0) {
    int t = f(x0), h = f(f(x0)), start = 0, length = 1;
    while(t != h)
        t = f(t), h = f(f(h));

    h = t; t = x0;
    while(t != h)
        t = f(t), h = f(h), ++start;

    h = f(t);
    while(t != h)
        h = f(h), ++length;

    return make_pair(start, length);
}
```

6.4 Funções para Datas

```
Complexidade: O(1)

Zeller: Eh capaz de calcular o dia da semana para o calendario
gregoriano (atual) - chame zeller() -, ou calendario juliano
(antigo, considerava bissexto todo ano multiplo de 4, sem as
regras de multiplo de 100 e 400) - chame zeller_julian().
Getdate: Retorna o numero de dias a partir do ano 0 ate a data
```

```
bool bissex(int y) { return (y%4==0 && (y%100 || y%400==0)); }
```

```
int zeller(int d, int m, int y) {
  if(m<3) --y, m+=12;
  return (d + ((m+1)*13)/5 + y + y/4 +
          6*(y/100) + y/400 + 6) % 7;
}

int zeller_julian(int d, int m, int y) {
  if(m<3) --y, m+=12;
  return (d + ((m+1)*13)/5 + y + y/4 + 4) % 7;
}

int getdate(int d, int m, int y) { //mes e dia a partir de 1
  int qm[]={0,31,28,31,30,31,30,31,31,30,31,30,31};
  int s=0;
  for (int i=1; i<m; i++) s+=qm[i];

  int res=365*y+s+d+(y/4-y/100+y/400);
  if (m<3 && bissex(y)) res--;
  return res;
}
```

6.5 Knight Distance

```
Complexidade: O(1)
Descricao: Determina em O(1) a distância (em movimentos de
cavalo) entre 2 pontos de um tabuleiro (infinito ou finito).
Se o tabuleiro for finito, deve ter tamanho n x m com
n >= 4 e m >= 4.

#include <algorithm>
using namespace std;

int knightdist_inf(int x1, int y1, int x2, int y2) {
  int dx=abs(x2-x1);
  int dy=abs(y2-y1);
  if (abs(dx)==1 && dy==0) return 3;
  if (abs(dy)==1 && dx==0) return 3;
  if (abs(dx)==2 && abs(dy)==2) return 4;

  int lb=max((dx+1)/2, (dy+1)/2);
  lb = max(lb, (dx+dy+2)/3);
  if ((lb%2)!=(dx+dy)%2) lb++;
  return lb;
}

int n,m; //tamanho do tabuleiro

int knightdist(int x1, int y1, int x2, int y2) {
  if(x1==n || x2==n) {
    x1 = n+1 - x1;
    x2 = n+1 - x2;
  }
  if(y1==m || y2==m) {
    y1 = m+1 - y1;
    y2 = m+1 - y2;
  }
  if((x1==1 && y1==1) || (x2==1 && y2==1)) {
    int a=abs(x1-x2), b=abs(y1-y2);
    if(a==0 && b==3 && m==4) return 5;
    if(b==0 && a==3 && n==4) return 5;
    if(a==1 && b==1) return 4;
  }
  return knightdist_inf(x1,y1,x2,y2);
}
```

6.6 Maior Retângulo em um Histograma

```
Complexidade: O(n)
Descricao: Dado um vetor que contem alturas (>=0) das
barras de um histograma de largura fixa = 1, calcula
a área do maior retangulo contido no histograma

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

int sh[MAX], sp[MAX];

long long histogram(int *v, int n) {
  int qs=1, curh=0;
  long long res=0;

  sh[0]=-1; sp[0]=0;
  v[n]=-1;

  for (int i=0; i<n+1; i++) {
    if (i<n && v[i]>=curh) {
      sh[qs]=v[i];
      sp[qs++]=i;
    }
    else {
      while (sh[qs-1]>v[i]) {
        qs--;
        res=max(res, (long long) sh[qs]*(i-sp[qs]));
      }
      sh[qs++]=v[i];
    }
    curh=v[i];
  }

  return res;
}
```

6.7 Polynomial Roots

```
typedef complex<double> cdouble;
int cmp(cdouble x, cdouble y = 0) {
  return cmp(abs(x), abs(y));
}

const int TAM = 200;

struct poly {
  cdouble poly[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;
  }
};
```

6.8 Romberg - Integral

```
long double romberg(long double a, long double b,
                    long double(*func)(long double)) {
    long double R[16][16], div = (b-a)/2;

    R[0][0] = div * (func(a) + func(b));
    for(int n = 1; n <= 15; n++, div /= 2) {
        R[n][0] = R[n-1][0]/2;
        for(long double sample = a + div; sample < b;
         sample += 2 * div)
            R[n][0] += div * func(a + sample);
    }

    for(int m = 1; m <= 15; m++)
        for(int n = m; n <= 15; n++)
            R[n][m] = R[n][m-1] + 1/(pow(4, m)-1) *
             (R[n][m-1] - R[n-1][m-1]);
```

```
    return R[15][15];
}
```

7 Strings

8 Matemática

8.1 Geometria

Matriz de rotação

$$\begin{bmatrix} x_\theta \\ y_\theta \end{bmatrix} = \begin{bmatrix} cos\ \theta & -sen\ \theta \\ sen\ \theta & cos\ \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Fórmula de Brahmagupta Sendo a, b, c, d os lados do quadrilátero, $s = \frac{1}{2}(a+b+c+d)$:

$$A = \sqrt{(s-a)(s-b)(s-c)(s-d) - k}$$

E sendo θ a soma do ângulo de dois lados opostos, ou p e q os comprimentos das diagonais do quadrilátero, temos:

$$\begin{aligned} k &= abcd\cos^2\tfrac{\theta}{2} \\ &= \tfrac{1}{4}(ac+bd+pq)(ac+bd-pq) \end{aligned}$$

Calota Esférica Sendo R o raio da esfera, r o raio da base, e h a altura da calota:

$$A_{calota} = 2\pi Rh$$

$$V_{calota} = \frac{\pi h}{6}\left(3r^2+h^2\right) = \frac{\pi h^2}{3}(3R-h)$$

Área de Segmento Circular Sendo α o ângulo formado pelo segmento circular, temos:

$$A_{segmento} = \frac{r^2}{2}(\alpha - sen\ \alpha)$$

Se tivermos h, a altura do segmento circular, ao invés de α:

$$\alpha = 2acos\left(\frac{h}{r}\right)$$

Centróide de um Polígono

$$c_x = \frac{1}{6A}\sum_{i=0}^{n-1}(x_i + x_{i+1})(x_i y_{i+1} - x_{i+1}y_i)$$

$$c_y = \frac{1}{6A}\sum_{i=0}^{n-1}(y_i + y_{i+1})(x_i y_{i+1} - x_{i+1}y_i)$$

Área de triângulo Sendo R o raio da circunferência circunscrita, e r da inscrita, temos:

$$A_\triangle = \frac{abc}{4R} = \frac{(a+b+c)r}{2}$$

Fórmula de Euler para Poliedros Convexos V vértices, A arestas, F faces: $V - A + F = 2$

Teorema de Pick Sendo A a área de um polígono e i e b a quantidade de pontos de coordenadas inteiras no interior e na borda no polígono,respectivamente, temos:

$$A = i + b/2 - 1$$

Quantidade de pontos de coordenas inteiras num segmento Sendo (x_1, y_1) e (x_2, y_2) pontos de coordenadas inteiras nos extremos de um segmento:

$$q = mdc(|x_1 - x_2|, |y_1 - y_2|) + 1$$

8.2 Relações Binomiais

Relação de Stifel:

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$$

Absorções:

$$\binom{n}{k} = \frac{n-k+1}{k}\binom{n}{k-1} = \frac{n}{k}\binom{n-1}{k-1} = \frac{n}{n-k}\binom{n-1}{k}$$

Soma de quadrados de binomiais:

$$\sum_{k=0}^{n}\binom{n}{k}^2 = \binom{2n}{n}$$

8.3 Equações Diofantinas

Dados inteiros $a, b > 0$ e c, a equação $ax + by = c$ tem soluções sse $g = gcd(a,b)$ é divisor de c.

Sejam x_g e y_g a solução de $a \cdot x_g + b \cdot y_g = g$ obtida por Euclides. Então:

$$\begin{cases} x = x_g(c/g) + k \cdot b/g \\ y = y_g(c/g) - k \cdot a/g \end{cases} \quad k \in Z$$

8.4 Fibonacci

Fórmula em $lg(n)$:
$f(0) = 1$ e $f(1) = 1$

$$\begin{aligned} f(n) &= f(x)f(n-x) + f(x-1)f(n-x-1) \\ &= f(\lfloor \tfrac{n}{2} \rfloor)f(n - \lfloor \tfrac{n}{2} \rfloor) + f(\lfloor \tfrac{n}{2} \rfloor + 1)f(n - \lfloor \tfrac{n}{2} \rfloor - 1) \\ &= f(\lfloor \tfrac{n}{2} \rfloor)f(\lceil \tfrac{n}{2} \rceil) + f(\lfloor \tfrac{n}{2} \rfloor + 1)f(\lceil \tfrac{n}{2} \rceil - 1) \end{aligned}$$

Fórmula com potência de matrizes:

$$\begin{bmatrix} f(n+1) \\ f(n) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n \begin{bmatrix} f(1) \\ f(0) \end{bmatrix}$$

Propriedades:

- $f(n+1)f(n-1) - f(n)^2 = (-1)^n$
- $f(m)$ múltiplo de $f(n)$ sse m múltiplo de n
- $mdc(f(m), f(n)) = f(mdc(m,n))$

8.5 Problemas clássicos

Fila do cinema: Sendo n pessoas com \$5 e m com \$10, temos:
$K_{0,m} = 0$ e $K_{n,0} = 1$
$K_{n,m} = K_{n-1,m} + K_{n,m-1}$

$$K_{n,m} = \binom{n+m}{n} - \binom{n+m}{n+1} = \frac{n-m+1}{n+1}\binom{n+m}{n}$$

Números de Catalan: É um caso do problema da *Fila de cinema*, com $n = m$.

$$C_n = \binom{2n}{n} - \binom{2n}{n+1} = \frac{1}{n+1}\binom{2n}{n}$$

Aplicações: 1) Número de expressões com n pares de parênteses, todos abrindo e fechando corretamente. Exemplo: (()) ()(); 2) Número de maneiras de parentizar completamente $n+1$ fatores. Exemplo: $(ab)c$ $a(bc)$; 3) Número de árvores binárias completas com $n+1$ folhas; 4) Número de maneiras de triangularizar um polígono convexo de $n+2$ lados;

Número de somas $x_1 + x_2 + \cdots + x_n = p$
- Soluções não negativas: $CR_n^p = \binom{n+p-1}{p}$
- Soluções positivas $CP_n^p = \binom{p-1}{n-1}$

Variáveis com restrições: Quando alguns x_i têm restrições do tipo $x_i \geq 3$, adotamos um y_i tal que $x_i = 3 + y_i$.

Assim, seguindo a restrição de que $y_i \geq 0$, teremos $x_i \geq 3$. A soma fica, então:

$$\begin{aligned} x_1 + x_2 + \cdots + x_i + \cdots + x_n &= p \\ x_1 + x_2 + \cdots + y_i + \cdots + x_n &= p - 3 \end{aligned}$$

De forma geral, teremos:

$$CR_n^p = \binom{n + p - (b_1 + b_2 + \cdots + b_n) - 1}{p}$$

Sendo b_i o decremento (pode ser negativo) na variável x_i.

$x_1 + x_2 + \cdots + x_n \leq p$

Definimos uma variável de $folga$, $f = p - (x_1 + x_2 + \cdots + x_n)$, e obtemos:

$$\begin{aligned} f &\geq 0 \\ x_1 + x_2 + \cdots + x_n + f &= p \end{aligned}$$

Permutações Caóticas: O número de permutações caóticas para n elementos é dado por: $D_0 = 1; D_n = (-1)^n + nD_{n-1} = (n-1)(D_{n-1} + D_{n-2})$

Triângulos de Lados em $\{1, 2, \cdots, n\}$

$$f_{n+1} = f_n + \begin{cases} \frac{(n-2)^2}{2} & \text{, n par} \\ \left\lceil \frac{(n-2)(n-4)}{4} \right\rceil & \text{, n ímpar} \end{cases}$$

Problema de Josephus: Sendo n pessoas em circulo, eliminando-se de k em k, temos a recorrência:

$$\begin{aligned} f(1,k) &= 0 \\ f(n,k) &= (f(n-1,k) + k)(mod\ n) \end{aligned}$$

Formas de Conectar um Grafo: Seja um grafo com k componentes com tamanhos $s_1, \cdots, s_k$. O número de maneiras de adicionar $k-1$ arestas de modo a conectá-lo é: $s_1 \cdots s_k n^{k-2}$

Código de Gray: $gray(i) = i \texttt{ xor } \frac{i}{2}$

Código de Gray Invertido (n-bits):

$$\overline{gray_n}(i) = (\frac{i}{2} \texttt{ or } (i \texttt{ and } ((n\%2)2^{n-1}))) \texttt{ xor } \begin{cases} i & \text{, i par} \\ \bar{i} & \text{, i ímpar} \end{cases}$$

8.6 Séries Numéricas

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{i=1}^{n} i^3 = \frac{n^2(n+1)^2}{4}$$

PA de 2ª ordem:

$$a_n = a_1 + b_1(n-1) + \frac{r}{2}(n-2)(n-1)$$

$$S_n = a_1 n + \frac{b_1 n(n-1)}{2} + \frac{r}{6}n(n-2)(n-1)$$

PA de nª ordem:

$$S_k = a_1\binom{k}{1} + \sum_{i=1}^{n} \Delta_i \binom{k}{i+1}$$

Δ_i: Primeiro elemento considerando a i-ésima PA.
Exemplo: n = 5 ; seq = (1,32,243,1024,3125,7776,...)

$$\Delta_1 = 32 - 1 = 31$$

$$\Delta_2 = 211 - 31 = 180$$

$$\Delta_3 = 570 - 180 = 390$$

$$\Delta_4 = 750 - 390 = 360$$

$$\Delta_5 = 480 - 360 = 120$$

Para a PA de 2ª ordem ficaria:

$$\Delta_1 = b_1$$

$$\Delta_2 = b_2 - b_1 = r$$

8.7 Matrizes e Determinantes

Determinante de Vandermonde:

$$V_n = \begin{vmatrix} 1 & 1 & \cdots & 1 \\ a_1 & a_2 & \cdots & a_n \\ a_1^2 & a_2^2 & \cdots & a_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ a_1^{n-1} & a_2^{n-1} & \cdots & a_n^{n-1} \end{vmatrix} = \prod_{i>j}(a_i - a_j)$$

8.8 Probabilidades

Probabilidade Condicional:

$$P(B|A) = \frac{n(A \cap B)}{n(A)} = \frac{P(A \cap B)}{P(A)}$$

Experimentos Repetidos: Seja um experimento que se repete n vezes, e em qualquer um deles temos $P(A) = p$ e, portanto, $P(\bar{A}) = 1 - p$. A probabilidade do evento A ocorrer k das n vezes é:

$$P_k = \binom{n}{k} p^k (1-p)^{n-k}$$

8.9 Teoria dos Números

Teorema de Fermat-Euler: Se p é primo, temos, para todo inteiro a: $a^{p-1} \equiv 1 (mod\ p)$. Se temos a e n coprimos: $a^{\phi(n)} \equiv 1 (mod\ n)$ onde $\phi(n) = n\prod_{p|n}\left(1 - \frac{1}{p}\right)$, p é fator primo de n, é a quantidade de números entre 1 e n que são coprimos com n.

Teorema de Wilson: n é primo sse $(n-1)! \equiv -1 (mod\ n)$

Soma dos Divisores: A soma dos divisores de n elevados à x-ésima potência, sendo p_i os fatores primos e a_i os expoentes correspondentes:

$$\sigma_x(n) = \prod_{i=1}^{r} \frac{p_i^{(a_i+1)x} - 1}{p_i^x - 1}$$

Divisibilidade
Considere o numéro como: $a_n a_{n-1} a_{n-2} ... a_2 a_1 a_0$

Por 3: A soma dos dígitos deve ser divisível por 3
Por 4: O número formado por $a_1 a_0$ deve ser divisível por 4
Por 7: A soma $a_2 a_1 a_0 - a_5 a_4 a_3 + a_8 a_7 a_6 - ...$ deve ser divisível por 7
Por 8: O número formado por $a_2 a_1 a_0$ deve ser divisível por 8
Por 9: A soma dos dígitos deve ser divisível por 9
Por 11: A soma $a_0 - a_1 + a_2 - a_3 + a_4 -$ deve ser divisível por 11
Por 13: A soma $a_2 a_1 a_0 - a_5 a_4 a_3 + a_8 a_7 a_6 - ...$ deve ser divisível por 13

Equação Modular Linear: Dada equação $ax \equiv b (mod\ m)$, se $b \equiv 0 (mod\ g)$ onde $g = \gcd(a, m)$, então as soluções são:

$$x = \frac{b}{g} * \text{invmod}(\frac{a}{g}, \frac{m}{g}) + k\frac{m}{g} \quad , k \in Z$$

8.10 Prime counting function ($\pi(x)$)

The prime counting function is asymptotic to $\frac{x}{\log x}$, by the prime number theorem.

x	10	10^2	10^3	10^4	10^5	10^6	10^7	10^8
$\pi(x)$	4	25	168	1.229	9.592	78.498	664.579	5.761.455

8.11 Partition function

The partition function $p(x)$ counts show many ways there are to write the integer x as a sum of integers.

x	36	37	38	39	40	41	42
p(x)	17.977	21.637	26.015	31.185	37.338	44.583	53.174

x	43	44	45	46	47	100	
p(x)	63.261	75.175	89.134	105.558	125.754	190.569.292	

8.12 Catalan numbers

Catalan numbers are defined by the recurrence:

$$C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i}$$

A closed formula for Catalan numbers is:

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1}$$

8.13 Stirling numbers of the first kind

These are the number of permutations of I_n with exactly k disjoint cycles. They obey the recurrence:

$$\begin{bmatrix} n \\ k \end{bmatrix} = (n-1)\begin{bmatrix} n-1 \\ k \end{bmatrix} + \begin{bmatrix} n-1 \\ k-1 \end{bmatrix}$$