Notebook Maratona de Programação

IFNMG Montes Claros

Vinícius Rodrigues Souza

24 de maio de 2019

Sumário

Capítulo 1

Teoria dos Grafos

1.1 Detecção de pontes

```
bool visitado[MAX];
int ss[MAX][MAX];
int tempo[MAX];
int timer = 0;
int cont;
int dfs(grafo& G, int u, int pai) {
    visitado[u] = true;
    tempo[u] = timer++;
    int aresta_tras = INF;
    for(int i=0; i < G[u].size(); ++i) {
        int v = G[u][i];
        if(visitado[v] == false) {
            int y = dfs(G, v, u);
            aresta_tras = min(aresta_tras, y);
            if(y > tempo[u]) cont+= ss[u][v];
        }
        else if(visitado[v] == true && v != pai)
            aresta_tras = min(aresta_tras, tempo[v]);
    }
    return aresta_tras;
}
```

1.2 Detecção de vértices de articulação

```
const int MN = 500;
set<int> vertice_corte;
int vis[MN];
vector<vector<int> > G(MN);
int dfs(int u, int ant, int timer) {
    vis[u] = timer++;
    int menor_ancestral = vis[u];
    int qtd_filhos = 0;

    for(int i = 0; i < G[u].size(); ++i) {
        int v = G[u][i];
        if(vis[v] == 0) {
            qtd_filhos++;
            int m = dfs(v, u, timer);
            menor_ancestral = min(menor_ancestral, m);
            if(vis[u] <= m && (u!=1 || qtd_filhos >=2 )){
               vertice_corte.insert(u);
            }
        }
        else if(v != ant) {
            menor_ancestral = min(menor_ancestral, vis[v]);
        }
    }
    return menor_ancestral;
}
```

1.3 Grafo bipartido

Flag é setada para True caso o grafo não for bipartido.

```
void bipartido(int u) {
    for(int i = 0; i < G[u].size(); ++i) {
        int v = G[u][i];
        if(vis[v] == -1) {
            vis[v] = vis[u] ^ 1;
```

```
            bipartido(v);
        }
        else {
            if(vis[v] == vis[u]) flag = true;
        }
    }
}
```

1.4 Componentes fortemente conexas

1.4.1 Algoritmo de Kosajaru

```
const int MN = 10010;
vector<int> G[MN], R[MN];
stack<int> pilha;
bool vis[MN];
int n;
int dfsStack(int u) {
    vis[u] = true;
    for(int i = 0; i < G[u].size(); ++i) {
        int v = G[u][i];
        if(!vis[v]) dfsStack(v);
    }
    pilha.push(u);
}
int inverteGrafo() {
    for(int i = 0; i < n; ++i)
        for(int j = 0; j < G[i].size(); ++j)
            R[G[i][j]].push_back(i);
}
int dfs(int u) {
    vis[u] = true;
    for(int i = 0;i < R[u].size(); ++i) {
        int v = R[u][i];
        if(!vis[v]) dfs(v);
    }
```

```
}
int Kosajaru() {
    memset(vis, false, sizeof vis);
    for(int i = 0; i < n; ++i)
        if(!vis[i]) dfsStack(i);
    inverteGrafo();
    memset(vis, false, sizeof vis);
    int cnt_c = 0;
    while(!pilha.empty()) {
        int v = pilha.top();
        if(!vis[v]) {
            dfs(v);
            cnt_c++;
        }
        pilha.pop();
    }
    return cnt_c;
}
```

1.5 Algoritmo de Tarjan

```
const int MN = 10010;
vector<int> G[MN];
int vis[MN], low[MN], num[MN], ans = 0, counter = 0;
stack<int> p;
void dfs(int u) {
    low[u] = num[u] = counter++;
    vis[u] = true;
    p.push(u);
    for(int i = 0; i < G[u].size(); ++i) {
        int v = G[u][i];
        if(num[v] == -1) {
            dfs(v);
            low[u] = min(low[u], low[v]);
        } else if(vis[v] == true) low[u] = min(low[u], low[v]);
    }
```

```
    if(low[u] == num[u]) {
        while(true) {
            int v = p.top(); p.pop();
            vis[v] = false;
            if(u == v) break;
        }

        ans++;
    }
}
void solve(int n) {
    memset(num, -1, sizeof num);
    for(int i = 1; i <= n; ++i) {
        if(num[i] == -1) dfs(i);
    }
}
```

1.6 Árvore Geradora Mínima: Algoritmo de Kruskal

```
const int MN = 10010;
typedef pair<int, pair<int, int> > ii;
ii arestas[MN];
int children[MN], pi[MN];
int find_parent(int u) {
    while(u != pi[u]) {
        u = pi[u];
    }
    return u;
}
void UnionSet(int ku, int kv) {
    if(children[kv] > children[ku]) {
        pi[kv] = ku;
        children[ku] += children[kv];
    }
    else {
        pi[ku] = kv;
```

```
            children[kv] += children[ku];
        }
}
int Kruskal(int n) {
    int custo_total = 0;
    sort(arestas, arestas + n);
    for(int i = 0; i < n; ++i) pi[i] = i, children[i] = 1;
    for(int i = 0; i < n; ++i) {
        int u = arestas[i].second.first;
        int v = arestas[i].second.second;
        int ku = find_parent(u);
        int kv = find_parent(v);

        if(ku != kv){
            UnionSet(ku, kv);
            custo_total += arestas[i].first;
        }
    }
    return custo_total;
}
```

1.7 Caminho mínimo

1.7.1 Algoritmo de Dijkstra

```
typedef pair<int, int> ii;
const int INF = 1e9 + 10;
struct compare
{
    bool operator() (const ii& x, const ii& y) const
    {
        return x.second > y.second;
    }
};
vector<vector<ii> > adj;
int aretas, vertices;
```

```
void Dijkstra(int src) {
    vector<int> dist(vertices, INF);
    priority_queue<ii, vector<ii>, compare> pq;
    pq.push(make_pair(src, 0));
    dist[src] = 0;

    while(!pq.empty()) {
        ii v = pq.top(); pq.pop();
        for(unsigned int i=0; i < adj[v.first].size(); ++i) {
            ii u = adj[v.first][i];
            if(dist[u.first] > dist[v.first] + u.second)
                pq.push(make_pair(u.first, dist[u.first] = dist[v.first] +
                    u.second));
        }
    }
}
```

1.7.2 Algoritmo de Floyd-Warshall

```
void Floyd(vv &dist) {
    for(int i=0; i < n; ++i)
        for(int j=0; j < n; ++j)
            for(int k=0; k < n; ++k)
                dist[j][k] = min(dist[j][k], dist[j][i] + dist[i][k]);
}
```

1.8 Fluxo máximo: Algoritmo de Edmonds-Karp

```
const int MN = 110;
const int INF = 100100;

vector<int> G[MN];
int rGraph[MN][MN], graph[MN][MN];
int N;

bool bfs(int s, int t, int parent[])  {
    bool visited[N+1];
    memset(visited, 0, sizeof(visited));
    queue <int> q;
    q.push(s);
    visited[s] = true;
    parent[s] = -1;
    while (!q.empty()) {
        int u = q.front(); q.pop();
        for (int i=0; i < G[u].size(); i++) {
            int v = G[u][i];
            if (visited[v] == false && rGraph[u][v] > 0)  {
                q.push(v);
                parent[v] = u;
                visited[v] = true;
            }
        }
    }

    return (visited[t] == true);
}

int solve(int s, int t) {
    int u, v;
    for (u = 1; u <= N; u++)
        for (v = 1; v <= N; v++)
             rGraph[u][v] = graph[u][v];
```

```

    int parent[N+1];
    int max_flow = 0;
    while (bfs(s, t, parent)) {
        int path_flow = INF;
        for (v=t; v!=s; v=parent[v]) {
            u = parent[v];
            path_flow = min(path_flow, rGraph[u][v]);
        }

        for (v=t; v != s; v=parent[v]) {
            u = parent[v];
            rGraph[u][v] -= path_flow;
            rGraph[v][u] += path_flow;
        }

        max_flow += path_flow;
    }
    return max_flow; // == sum_demanda;
}
```

Capítulo 2

Estruturas

2.1 Segment Tree

Consultas do tipo RMQ e RSQ em Log(n) Update O(n).

```
#define left(x) (x << 1)
#define right(x) ((x << 1) + 1)
typedef vector<int> vv;
vv st, arr;
void buildUtil(int si, int l, int r) {
    if(l == r) st[si] = arr[r];
    else {
        buildUtil(left(si), l, (l + r) / 2);
        buildUtil(right(si), (l + r) / 2 + 1, r);
        st[si] = st[left(si)] + st[right(si)];
    }
}
void build(int n) {
    st.resize(4 * n);
    buildUtil(1, 0, n-1);
}
void update(int si, int l, int r, int a, int b, int value) {
    if(a > r || b < l) return;
    if(l == r) {
        printf("%d\n", r);
        st[si] += value;
```

```
    }
    else {
        update(left(si), l, (l + r) / 2, a, b, value);
        update(right(si), (l + r) / 2 + 1, r, a, b, value);
        st[si] = st[left(si)] + st[right(si)];
    }
}
int getSum(int si, int l, int r, int a, int b) {
    if(l >= a && r <= b) return st[si];
    if(a > r || l > b) return 0;
    return getSum(left(si), l, (l + r) / 2, a, b) +
           getSum(right(si), (l + r) / 2 + 1, r, a, b);
}
```

2.2 Segment Tree with Lazy Propagation

```
#define left(x) (x << 1)
#define right(x) ((x << 1) + 1)
typedef vector<int> vv;
vv st, lazy, arr;
void buildUtil(int si, int l, int r) {
    if(l == r) st[si] = arr[r];
    else {
        buildUtil(left(si), l, (l + r) / 2);
        buildUtil(right(si), (l + r) / 2 + 1, r);
        // st[si] = st[left(si)] + st[right(si)];
        st[si] = max(st[left(si)] , st[right(si)]);
    }
}
void build(int n) {
    st.assign(4 * n, 0);
    lazy.assign(4 * n, 0);
    buildUtil(1, 0, n-1);
}
void update(int si, int l, int r, int a, int b, int value){
```

```
    if(lazy[si] != 0) {
        // st[si] += (r - l + 1) * lazy[si];
        st[si] +=  lazy[si];
        if(l != r) {
            lazy[left(si)] += lazy[si];
            lazy[right(si)] += lazy[si];
        }
        lazy[si] = 0;
    }
    if(a > r || b < l) return;
    if(l >= a && r <= b) {
        // st[si] += (r -l + 1) * value;
        st[si] += value;
        if(l != r) {
            lazy[left(si)] += value;
            lazy[right(si)] += value;
        }
    }
    else {
        update(left(si), l, (l + r) / 2, a, b, value);
        update(right(si), (l + r) / 2 + 1, r, a, b, value);
        // st[si] = st[left(si)] + st[right(si)];
        st[si] = max(st[left(si)], st[right(si)]);
    }
}
int getSum(int si, int l, int r, int a, int b) {
    if(lazy[si] != 0) {
        // st[si] += (r - l + 1) * lazy[si];
        st[si] += lazy[si];
        if(l != r) {
            lazy[left(si)] += lazy[si];
            lazy[right(si)] += lazy[si];
        }
        lazy[si] = 0;
    }
    if(l >= a && r <= b) return st[si];
```

```
    if(a > r || l > b) return 0;
    return max(getSum(left(si), l, (l + r) / 2, a, b),
           getSum(right(si), (l + r) / 2 + 1, r, a, b));
}
```

2.3 Prefix Sum 2D

Pré processamento O(nm). Realiza consultas do tipo RSQ em O(1). Indexado a partir de 1.

```
int v[100][100];
int p[101][101];

int getSum(int x, int y, int a, int b) {
    return p[a][b] - p[a][y-1] - p[x-1][b] + p[x-1][y-1];
}
void prefixSum(int n, int m) {
    for(int i=1; i <= n; ++i)
        for(int j=1; j <=m; ++j)
            p[i][j] = p[i-1][j] + v[i-1][j-1];

    for(int i=1; i <= n; ++i)
        for(int j=1; j <=m; ++j)
            p[i][j] += p[i][j-1];
}
```

2.4 Binary Indexed Tree: Fenwick

Consultas RSQ e update em Log(n)

```
const long long MN = 100010;
long long BIT[MN], v[MN];
long long getSum(long long index)
{
        long long sum = 0;
        index = index + 1;
        while (index>0) {
                sum += BIT[index];
```

```
                index -= index & (-index);
        }
        return sum;
}

void updateBIT(long long n, long long index, long long val) {
        index = index + 1;
        while (index <= n)  {
                BIT[index] += val;
                index += index & (-index);
        }
}
```

2.5 Binary Indexed Tree 2D: Fenwick

Coordenadas indexadas a partir de 1.

Consultas do somatório de uma região retangular em Log(n).

```
#define MAXN 1010
#define swap(x, y)((x)^=(y)^=(x)^=(y))
int bit[MAXN][MAXN];
int n, m;
int rsq(int i, int j) { // returns RSQ((1,1), (i,j))
        int sum = 0, k = j;
        for(; i > 0; i -= (i & -i)) {
                j = k;
                for(; j > 0; j -= (j & -j))
                        sum += bit[i][j];
        }
        return sum;
}
void update(int i, int j, int v) {
        int k = j;
        for(; i <= n; i += (i&-i)) {
                for(j = k; j <= m; j += (j&-j))
                        bit[i][j] += v;
```

```
        }
}
int getSum(int xa, int ya, int xb, int yb) {
    if(xa > xb) swap(xa, xb);
    if(ya > yb) swap(ya, yb);
        return rsq(xb, yb) - rsq(xb, ya-1) - rsq(xa-1, yb) + rsq(xa-1, yb
            -1);
}
```

2.6 SQRT Decomposition

Consultas em sqrt(n). Update O(1). Pré processamento O(n).

```
int arr[MAXN];                  // original array
int block[SQRSIZE];             // decomposed array
int blk_sz;                                     // block size
void update(int idx, int val) {
        int blockNumber = idx / blk_sz;
        block[blockNumber] += val - arr[idx];
        arr[idx] = val;
}
int query(int l, int r) {
        int sum = 0;
        while (l<r and l%blk_sz!=0 and l!=0) {
                sum += arr[l];
                l++;
        }
        while (l+blk_sz <= r) {
                sum += block[l/blk_sz];
                l += blk_sz;
        }
        while (l<=r) {
                sum += arr[l];
                l++;
        }
        return sum;
```

```
}
```

2.7 Union Find

```
vector<int> rank;
vector<int> parent;
int find(int i) {
    while(i != parent[i]) i = parent[i];
    return i;
}
void unionSet(int i, int j) {
    int x = find(i);
    int y = find(j);
    if(x == y) return;
    if(rank[x] > rank[y]) parent[y] = x;
    else {
        parent[x] = y;
        if(rank[x] == rank[y]) rank[y]++;
    }
}
```

2.8 Ancestral Comum mais Próximo (LCA)

Consultas de LCA em $Log(n)$

```
#define left(x) (x << 1)
#define right(x) (x << 1) + 1
#define parent(x) (x >> 1)

const int MN = 1010;
int first[MN], vis[MN], height[MN];
vector<int> G[MN], euler, st;
void init() {
    st.resize(euler.size() * 4);
}
void build(int s, int l, int r) {
```

```
    if(l == r) st[s] = euler[l];
    else {
        build(left(s), l, (l+r)/2);
        build(right(s), (l+r)/2 + 1, r);
        int L = st[left(s)];
        int R = st[right(s)];
        st[s] = height[L] < height[R] ? L : R;
    }
}
int query(int s, int l, int r, int a, int b) {
    if(a > r || b < l) return -1;
    if(l >= a && r <= b) return st[s];
    int L = query(left(s), l, (l + r)/ 2, a, b);
    int R = query(right(s), (l + r)/ 2 + 1, r, a, b);
    if(L == -1) return R;
    if(R == -1) return L;
    return height[L] < height[R] ? L : R;
}
int LCA(int n, int a, int b) {
    a = first[a]; b = first[b];
    if(b < a) swap(a, b);
    return query(1, 0, euler.size() - 1, a, b);
}
void dfs(int u, int h) {
    vis[u] = true;
    height[u] = h;
    first[u] = euler.size();
    euler.push_back(u);
    for(auto to : G[u]) {
        if(!vis[to]) {
            dfs(to, h+1);
            euler.push_back(u);
        }
    }
}
int main() {
```

```
    dfs(0, 0);
    init();
    build(1, 0, euler.size() -1);
}
```

2.9 Heavy Light Decomposition (HLD)

Atualiza um intervalo na arvore em Log(n). Consulta valor do vértice em Log(n).

```
const int MAXN = 5e3 + 10;
const int INF = 1e9 + 10;
#define left(x) (x << 1)
#define right(x) ((x << 1) + 1)
int ncha;
int parent[MAXN], fson[MAXN], size[MAXN];
int nchain[MAXN], id[MAXN], depth[MAXN], up[MAXN];
vector<int> G[MAXN], chain[MAXN];
class SegmentTree {
private:
    vector<int> st, lazy;
    int size;
    void update(int si, int l, int r, int a, int b, int value) {
        if(lazy[si] != 0) {
            st[si] += (r - l + 1) * lazy[si];
            if(l != r) {
                lazy[left(si)] += lazy[si];
                lazy[right(si)] += lazy[si];
            }
            lazy[si] = 0;
        }
        if(a > r || b < l) return;
        if(l >= a && r <= b) {
            st[si] += (r -l + 1) * value;
            if(l != r) {
                lazy[left(si)] += value;
                lazy[right(si)] += value;
```

```
            }
        }
        else {
            update(left(si), l, (l + r) / 2, a, b, value);
            update(right(si), (l + r) / 2 + 1, r, a, b, value);
            st[si] = st[left(si)] + st[right(si)];
        }
    }
    int query(int si, int l, int r, int a, int b) {
        if(lazy[si] != 0) {
            st[si] += (r - l + 1) * lazy[si];
            if(l != r) {
                lazy[left(si)] += lazy[si];
                lazy[right(si)] += lazy[si];
            }
            lazy[si] = 0;
        }
        if(l >= a && r <= b) return st[si];
        if(a > r || l > b) return 0;
        return query(left(si), l, (l + r) / 2, a, b) +
            query(right(si), (l + r) / 2 + 1, r, a, b);
    }
public:
    SegmentTree(int sz) {
        size = sz;
        st.assign(size * 4, 0);
        lazy.assign(size * 4, 0);
    }
    int query(int a, int b) {
        return query(1, 0, size - 1, a, b);
    }
    void update(int a, int b, int value) {
        update(1, 0, size - 1, a, b, value);
    }
};
int chainsz(int u, int p) {
```

```
    size[u] = 1; fson[u] = -1; parent[u] = p;
    int heavy = 0;
    for(int i=0; i < (int)G[u].size(); ++i) {
        int v = G[u][i];
        if(v == p) continue;
        size[u] += chainsz(v, u);
        if(size[v] > heavy) {
            fson[u] = v; heavy = size[v];
        }
    }
    return size[u];
}
void build(int u, int ch, int h) {
    nchain[u] = ch; id[u] = chain[ch].size();
    chain[ch].push_back(u);
    for(int i=0; i < (int)G[u].size(); ++i) {
        int v = G[u][i];
        if(v == parent[u]) continue;
        if(v == fson[u]) build(v, ch, h + 1);
        else {
            up[ncha] = u; depth[ncha] = h;
            chain[ncha].clear();
            build(v, ncha++, h + 1);
        }
    }
}
vector<SegmentTree> hld;
void HLD(int root) {
    chainsz(root, -1);
    ncha = 0;
    chain[ncha].clear();
    up[ncha] = -1; depth[ncha] = 0;
    build(root, ncha++, 1);

    for(int i=0; i < ncha; ++i) {
        hld.push_back(SegmentTree(chain[i].size()));
```

```
    }
}
void update(int u, int v, int value) {
    int cu = nchain[u], cv = nchain[v];
    while(cu != cv) {
        if(depth[cu] > depth[cv]) {
            hld[cu].update(0, id[u], value);
            u = up[cu];
        }
        else {
            hld[cv].update(0, id[v], value);
            v = up[cv];
        }
        cu = nchain[u]; cv = nchain[v];
    }
    if (id[u] < id[v]) {
        hld[cu].update(id[u], id[v], value);
    }
    else {
        hld[cu].update(id[v], id[u], value);
    }
}

int custo[MAXN];
int agua[MAXN];

int main() {
    int n, d, u, v, w, m, q;
    scanf("%d %d", &n , &d);
    for(int i=0; i < n -1; ++i) {
        scanf("%d %d", &u, &v);
        G[u].push_back(v);
        G[v].push_back(u);
    }
    HLD(1);
    memset(custo, INF, sizeof custo);
```

```
    scanf("%d", &m);
    for(int i=0; i < m; ++i) {
        scanf("%d %d", &u, &v);
        custo[u] = v;
    }
    scanf("%d", &q);
    for(int i=0; i < q; ++i) {
        scanf("%d %d %d", &u, &v, &w);
        update(u, v, w);
    }
    for(int i=1; i <= n; ++i) {
        agua[i] = hld[nchain[i]].query(id[i], id[i]);
    }
```

Capítulo 3

Paradigmas

3.1 Problema da mochila

```
for(int i = 0; i <= n+1; ++i) {
    for(int j = 0; j <= s; ++j) {
        if(i == 0) dp[i][j] = 0;
        else {
            dp[i][j] = dp[i-1][j];
            if(a[i] <= j) {
                dp[i][j] = max(dp[i][j], dp[i-1][j - a[i]] + b[i]);
            }
        }
    }
}
cout << dp[n][s] << '\n';
```

3.2 Kadane

Dado um array de inteiros, essa função retorna a soma da maior subsequência contigua de maior soma.

```
int kadane(int n){
    int soma, ans;
    soma = ans = 0;
    for(int i=0; i < n; ++i) {
        soma = soma + v[i];
        ans = std::max(ans, soma);
        if(soma < 0) soma = 0;
```

```
    }
    return ans;
}
```

3.3 LIS

```
int lis(vector<int> const& a) {
    int n = a.size();
    vector<int> d(n, 1);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < i; j++) {
            if (a[j] < a[i])
                d[i] = max(d[i], d[j] + 1);
        }
    }
    int ans = d[0];
    for (int i = 1; i < n; i++) {
        ans = max(ans, d[i]);
    }
    return ans;
}
```

3.4 LCS

Usar getline para lê as strings. O(mn).

```
for(int i = 0; i <= S.size(); ++i) {
    for(int j = 0; j <= P.size(); ++j) {
        if(i == 0 || j == 0) dp[i][j] = 0;
        else if(S[i-1] == P[j-1]) dp[i][j] = dp[i-1][j-1] + 1;
        else dp[i][j] = max(dp[i][j-1], dp[i-1][j]);
    }
}
//ans = [S.size()][P.size()];
```

3.5 Contagem de inversões do Merge Sort

```
#define INF 1000000000
long long merge_sort(vector<long long> &v){
        long long inv=0;
        if(v.size()==1) return 0;
        vector<long long> u1, u2;
        for(long long i=0;i<v.size()/2;i++)
                u1.push_back(v[i]);
        for(long long i=v.size()/2;i<v.size();i++)
                u2.push_back(v[i]);
        inv+=merge_sort(u1);
        inv+=merge_sort(u2);
        u1.push_back(INF);
        u2.push_back(INF);
        long long ini1=0, ini2=0;
        for(long long i=0;i<v.size();i++){
        // Comparacao da ordenacao
        if(u1[ini1]<=u2[ini2]){
            v[i]=u1[ini1];
            ini1++;
                }
        else{
            v[i]=u2[ini2];
            ini2++;
            inv+=u1.size()-ini1-1;
        }
    }
    return inv;
}
```

3.6 Problema do troco

Capítulo 4

Teoria dos números

4.1 Recorrência Linear

É preciso ficar atento na hora de definir o valor de K. Exemplo: $f(i) = 2f(i-1) + f(i-4)$, O K deve ser 4, ja que essa mesma recorrência escrita explicitamente é da forma $f(i) = 0f(i-1) + 2f(i-2) + 0f(i-3) + f(i-4)$. matriz de transformação é dada por:

$$T = \begin{bmatrix} 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ C_K & C_{K-1} & C_{K-2} & C_{K-3} & \dots & C_1 \end{bmatrix}_{K \times K}$$

O código abaixo encontra o n-ésimo termo da sequência de Fibonacci.

```
#include <vector>
#define REP(i,n) for (int i = 1; i <= n; i++)
using namespace std;

typedef long long ll;
typedef vector<vector<ll> > matrix;
const ll MOD = 1000000007;
const int K = 2; //Numero de termos das quais f(n) depende
// computes A * B
matrix mul(matrix A, matrix B)
{
    matrix C(K+1, vector<ll>(K+1));
```

```
    REP(i, K) REP(j, K) REP(k, K)
        C[i][j] = (C[i][j] + A[i][k] * B[k][j]) % MOD;
    return C;
}
// computes A ^ p
matrix pow(matrix A, int p)
{
    if (p == 1)
        return A;
    if (p % 2)
        return mul(A, pow(A, p-1));
    matrix X = pow(A, p/2);
    return mul(X, X);
}
// returns the N-th term of Fibonacci sequence
int fib(int N) {
    // create vector F1
    vector<ll> F1(K+1);
    F1[1] = 1;
    F1[2] = 1;

    // create matrix T
    matrix T(K+1, vector<ll>(K+1));
    T[1][1] = 0, T[1][2] = 1;
    T[2][1] = 1, T[2][2] = 1;

    // raise T to the (N-1)th power
    if (N == 1)
        return 1;
    T = pow(T, N-1);

    // the answer is the first row of T . F1
    ll res = 0;
    REP(i, K)
        res = (res + T[1][i] * F1[i]) % MOD;
    return res;
```

```
}
```

Exemplo de uma variante: $f(i) = Mf(i-2) + Nf(i-3)$. Essa recorrência de pode ser reescrita da forma $f(i) = 0f(i-1) + f(i-2) + f(i-3)$, sendo assim o K = 3, a matriz de transformação T é dado por:

$$T = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ N & M & 0 \end{bmatrix}_{K \times K}$$

Capítulo 5

String

5.1 KMP

```
const int MN = 1000010;
int b[MN];
vector<int> ans;
void pre(string pattern) {
    b[0] = -1;
    int n = pattern.size();
    for(int i = 0, j = -1; i < n;) {
        while(j >= 0 && pattern[i] != pattern[j]) j = b[j];
        b[++i] = ++j;
    }
}
void KMP(string pattern, string text) {
    pre(pattern);
    for(int i = 0, j = 0; i < text.size();) {
        while(j >= 0 && pattern[j] != text[i]) j = b[j];
        ++i; ++j;
        if(j == pattern.size()) ans.push_back(i);
    }
}
```

Capítulo 6

Geométrico

6.1 Convex Hull

Solução do problema URI 1982.

```
#include <iostream>
#include <stack>
#include <math.h>
#include <stdlib.h>
using namespace std;

struct Point {
        int x, y;
};

Point p0;
const int MN = 2010;
Point nextToTop(stack<Point> &S) {
        Point p = S.top();
        S.pop();
        Point res = S.top();
        S.push(p);
        return res;
}
int swap(Point &p1, Point &p2) {
        Point temp = p1;
```

```
        p1 = p2;
        p2 = temp;
}
int distSq(Point p1, Point p2) {
        return (p1.x - p2.x)*(p1.x - p2.x) +
                (p1.y - p2.y)*(p1.y - p2.y);
}
// To find orientation of ordered triplet (p, q, r).
// The function returns following values
// 0 --> p, q and r are colinear
// 1 --> Clockwise
// 2 --> Counterclockwise
int orientation(Point p, Point q, Point r)
{
        int val = (q.y - p.y) * (r.x - q.x) -
                        (q.x - p.x) * (r.y - q.y);

        if (val == 0) return 0; // colinear
        return (val > 0)? 1: 2; // clock or counterclock wise
}

int compare(const void *vp1, const void *vp2) {
Point *p1 = (Point *)vp1;
Point *p2 = (Point *)vp2;
int o = orientation(p0, *p1, *p2);
if (o == 0)
        return (distSq(p0, *p2) >= distSq(p0, *p1))? -1 : 1;
return (o == 2)? -1: 1;
}

double convexHull(Point points[], int n) {
int ymin = points[0].y, min = 0;
for (int i = 1; i < n; i++)
{
        int y = points[i].y;
        if ((y < ymin) || (ymin == y &&
```

```
                points[i].x < points[min].x))
                ymin = points[i].y, min = i;
}
swap(points[0], points[min]);
p0 = points[0];
qsort(&points[1], n-1, sizeof(Point), compare);
int m = 1;
for (int i=1; i<n; i++)  {
    while (i < n-1 && orientation(p0, points[i],points[i+1]) == 0) i++;
        points[m] = points[i];
        m++;
}
stack<Point> S;
S.push(points[0]);
S.push(points[1]);
S.push(points[2]);

for (int i = 3; i < m; i++)
{
        while (orientation(nextToTop(S), S.top(), points[i]) != 2) S.pop();
        S.push(points[i]);
}
double custo = 0;
Point pivo = S.top(), q, p;
p = pivo;
S.pop();

while (!S.empty())  {
        q = S.top(); S.pop();
        custo += sqrt(pow(p.x - q.x, 2) + pow(p.y - q.y, 2));
        p = q;
}
custo += sqrt(pow(pivo.x - q.x, 2) + pow(pivo.y - q.y, 2));
        return custo;
}
```

```

int main()
{
	Point Pontos[MN];
	int n, u, v;
	Point p;

	while(true) {
		cin >> n;
		if(n == 0) break;

		for(int i = 0; i < n; ++i) {
			scanf("%d %d", &p.x, &p.y);
			Pontos[i] = p;
		}

		printf("Tera que comprar uma fita de tamanho %.2lf.\n",
		   convexHull(Pontos, n));
	}
	return 0;
}
```