

Notebook Maratona de Programação

IFNMG Montes Claros

Vinícius Rodrigues Souza

24 de maio de 2019

Sumário

1	Teoria dos Grafos	2
1.1	Detecção de pontes	2
1.2	Detecção de vértices de articulação	3
1.3	Grafo bipartido	3
1.4	Componentes fortemente conexas	4
1.4.1	Algoritmo de Kosajaru	4
1.5	Algoritmo de Tarjan	5
1.6	Árvore Geradora Mínima: Algoritmo de Kruskal	6
1.7	Caminho mínimo	7
1.7.1	Algoritmo de Dijkstra	7
1.7.2	Algoritmo de Floyd-Warshall	8
1.8	Fluxo máximo: Algoritmo de Edmonds-Karp	9
2	Estruturas	11
2.1	Segment Tree	11
2.2	Segment Tree with Lazy Propagation	12
2.3	Prefix Sum 2D	14
2.4	Binary Indexed Tree: Fenwick	14
2.5	Binary Indexed Tree 2D: Fenwick	15
2.6	SQRT Decomposition	16
2.7	Union Find	17
2.8	Ancestral Comum mais Próximo (LCA)	17
2.9	Heavy Light Decomposition (HLD)	19
3	Paradigmas	24
3.1	Problema da mochila	24
3.2	Kadane	24

3.3	LIS	25
3.4	LCS	25
3.5	Contagem de inversões do Merge Sort	26
3.6	Problema do troco	26
4	Teoria dos números	27
4.1	Recorrência Linear	27
5	String	30
5.1	KMP	30
6	Geométrico	31
6.1	Convex Hull	31

Capítulo 1

Teoria dos Grafos

1.1 Detecção de pontes

```
1  bool visitado[MAX];
2  int ss[MAX][MAX];
3  int tempo[MAX];
4  int timer = 0;
5  int cont;
6  int dfs(grafo& G, int u, int pai) {
7      visitado[u] = true;
8      tempo[u] = timer++;
9      int aresta_tras = INF;
10     for(int i=0; i < G[u].size(); ++i) {
11         int v = G[u][i];
12         if(visitado[v] == false) {
13             int y = dfs(G, v, u);
14             aresta_tras = min(aresta_tras, y);
15             if(y > tempo[u]) cont+= ss[u][v];
16         }
17         else if(visitado[v] == true && v != pai)
18             aresta_tras = min(aresta_tras, tempo[v]);
19     }
20     return aresta_tras;
21 }
```

1.2 Detecção de vértices de articulação

```
1  const int MN = 500;
2  set<int> vertice_corte;
3  int vis[MN];
4  vector<vector<int> > G(MN);
5  int dfs(int u, int ant, int timer) {
6      vis[u] = timer++;
7      int menor_ancestral = vis[u];
8      int qtd_filhos = 0;
9
10     for(int i = 0; i < G[u].size(); ++i) {
11         int v = G[u][i];
12         if(vis[v] == 0) {
13             qtd_filhos++;
14             int m = dfs(v, u, timer);
15             menor_ancestral = min(menor_ancestral, m);
16             if(vis[u] <= m && (u!=1 || qtd_filhos >=2 )){
17                 vertice_corte.insert(u);
18             }
19         }
20         else if(v != ant) {
21             menor_ancestral = min(menor_ancestral, vis[v]);
22         }
23     }
24     return menor_ancestral;
25 }
```

1.3 Grafo bipartido

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

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

```

6         bipartido(v);
7     }
8     else {
9         if(vis[v] == vis[u]) flag = true;
10    }
11 }
12 }

```

1.4 Componentes fortemente conexas

1.4.1 Algoritmo de Kosajaru

```

1  const int MN = 10010;
2  vector<int> G[MN], R[MN];
3  stack<int> pilha;
4  bool vis[MN];
5  int n;
6  int dfsStack(int u) {
7      vis[u] = true;
8      for(int i = 0; i < G[u].size(); ++i) {
9          int v = G[u][i];
10         if(!vis[v]) dfsStack(v);
11     }
12     pilha.push(u);
13 }
14 int invertGrafo() {
15     for(int i = 0; i < n; ++i)
16         for(int j = 0; j < G[i].size(); ++j)
17             R[G[i][j]].push_back(i);
18 }
19 int dfs(int u) {
20     vis[u] = true;
21     for(int i = 0; i < R[u].size(); ++i) {
22         int v = R[u][i];
23         if(!vis[v]) dfs(v);
24     }

```

```

25 }
26 int Kosajaru() {
27     memset(vis, false, sizeof vis);
28     for(int i = 0; i < n; ++i)
29         if(!vis[i]) dfsStack(i);
30     inverteGrafo();
31     memset(vis, false, sizeof vis);
32     int cnt_c = 0;
33     while(!pilha.empty()) {
34         int v = pilha.top();
35         if(!vis[v]) {
36             dfs(v);
37             cnt_c++;
38         }
39         pilha.pop();
40     }
41     return cnt_c;
42 }

```

1.5 Algoritmo de Tarjan

```

1  const int MN = 10010;
2  vector<int> G[MN];
3  int vis[MN], low[MN], num[MN], ans = 0, counter = 0;
4  stack<int> p;
5  void dfs(int u) {
6      low[u] = num[u] = counter++;
7      vis[u] = true;
8      p.push(u);
9      for(int i = 0; i < G[u].size(); ++i) {
10         int v = G[u][i];
11         if(num[v] == -1) {
12             dfs(v);
13             low[u] = min(low[u], low[v]);
14         } else if(vis[v] == true) low[u] = min(low[u], low[v]);
15     }

```

```

16     if(low[u] == num[u]) {
17         while(true) {
18             int v = p.top(); p.pop();
19             vis[v] = false;
20             if(u == v) break;
21         }
22
23         ans++;
24     }
25 }
26 void solve(int n) {
27     memset(num, -1, sizeof num);
28     for(int i = 1; i <= n; ++i) {
29         if(num[i] == -1) dfs(i);
30     }
31 }

```

1.6 Árvore Geradora Mínima: Algoritmo de Kruskal

```

1  const int MN = 10010;
2  typedef pair<int, pair<int, int> > ii;
3  ii arestas[MN];
4  int children[MN], pi[MN];
5  int find_parent(int u) {
6      while(u != pi[u]) {
7          u = pi[u];
8      }
9      return u;
10 }
11 void UnionSet(int ku, int kv) {
12     if(children[kv] > children[ku]) {
13         pi[kv] = ku;
14         children[ku] += children[kv];
15     }
16     else {
17         pi[ku] = kv;

```



```

18         children[kv] += children[ku];
19     }
20 }
21 int Kruskal(int n) {
22     int custo_total = 0;
23     sort(arestas, arestas + n);
24     for(int i = 0; i < n; ++i) pi[i] = i, children[i] = 1;
25     for(int i = 0; i < n; ++i) {
26         int u = arestas[i].second.first;
27         int v = arestas[i].second.second;
28         int ku = find_parent(u);
29         int kv = find_parent(v);
30
31         if(ku != kv){
32             UnionSet(ku, kv);
33             custo_total += arestas[i].first;
34         }
35     }
36     return custo_total;
37 }

```

1.7 Caminho mínimo

1.7.1 Algoritmo de Dijkstra

```

1  typedef pair<int, int> ii;
2  const int INF = 1e9 + 10;
3  struct compare
4  {
5      bool operator() (const ii& x, const ii& y) const
6      {
7          return x.second > y.second;
8      }
9  };
10 vector<vector<ii> > adj;
11 int aretas, vertices;

```

```

12 void Dijkstra(int src) {
13     vector<int> dist(vertices, INF);
14     priority_queue<ii, vector<ii>, compare> pq;
15     pq.push(make_pair(src, 0));
16     dist[src] = 0;
17
18     while(!pq.empty()) {
19         ii v = pq.top(); pq.pop();
20         for(unsigned int i=0; i < adj[v.first].size(); ++i) {
21             ii u = adj[v.first][i];
22             if(dist[u.first] > dist[v.first] + u.second)
23                 pq.push(make_pair(u.first, dist[u.first] = dist[v.first] +
24                                     u.second));
25         }
26     }

```

1.7.2 Algoritmo de Floyd-Warshall

```

1 void Floyd(vv &dist) {
2     for(int i=0; i < n; ++i)
3         for(int j=0; j < n; ++j)
4             for(int k=0; k < n; ++k)
5                 dist[j][k] = min(dist[j][k], dist[j][i] + dist[i][k]);
6 }

```

1.8 Fluxo máximo: Algoritmo de Edmonds-Karp

```
1  const int MN = 110;
2  const int INF = 100100;
3
4  vector<int> G[MN];
5  int rGraph[MN][MN], graph[MN][MN];
6  int N;
7
8  bool bfs(int s, int t, int parent[]) {
9      bool visited[N+1];
10     memset(visited, 0, sizeof(visited));
11     queue<int> q;
12     q.push(s);
13     visited[s] = true;
14     parent[s] = -1;
15     while (!q.empty()) {
16         int u = q.front(); q.pop();
17         for (int i=0; i < G[u].size(); i++) {
18             int v = G[u][i];
19             if (visited[v] == false && rGraph[u][v] > 0) {
20                 q.push(v);
21                 parent[v] = u;
22                 visited[v] = true;
23             }
24         }
25     }
26
27     return (visited[t] == true);
28 }
29
30 int solve(int s, int t) {
31     int u, v;
32     for (u = 1; u <= N; u++)
33         for (v = 1; v <= N; v++)
34             rGraph[u][v] = graph[u][v];
```

```

35
36     int parent[N+1];
37     int max_flow = 0;
38     while (bfs(s, t, parent)) {
39         int path_flow = INF;
40         for (v=t; v!=s; v=parent[v]) {
41             u = parent[v];
42             path_flow = min(path_flow, rGraph[u][v]);
43         }
44
45         for (v=t; v != s; v=parent[v]) {
46             u = parent[v];
47             rGraph[u][v] -= path_flow;
48             rGraph[v][u] += path_flow;
49         }
50
51         max_flow += path_flow;
52     }
53     return max_flow; // == sum_demanda;
54 }

```

Capítulo 2

Estruturas

2.1 Segment Tree

Consultas do tipo RMQ e RSQ em $\text{Log}(n)$ Update $O(n)$.

```
1  #define left(x) (x << 1)
2  #define right(x) ((x << 1) + 1)
3  typedef vector<int> vv;
4  vv st, arr;
5  void buildUtil(int si, int l, int r) {
6      if(l == r) st[si] = arr[r];
7      else {
8          buildUtil(left(si), l, (l + r) / 2);
9          buildUtil(right(si), (l + r) / 2 + 1, r);
10         st[si] = st[left(si)] + st[right(si)];
11     }
12 }
13 void build(int n) {
14     st.resize(4 * n);
15     buildUtil(1, 0, n-1);
16 }
17 void update(int si, int l, int r, int a, int b, int value) {
18     if(a > r || b < l) return;
19     if(l == r) {
20         printf("%d\n", r);
21         st[si] += value;
```

```

22     }
23     else {
24         update(left(si), l, (l + r) / 2, a, b, value);
25         update(right(si), (l + r) / 2 + 1, r, a, b, value);
26         st[si] = st[left(si)] + st[right(si)];
27     }
28 }
29 int getSum(int si, int l, int r, int a, int b) {
30     if(l >= a && r <= b) return st[si];
31     if(a > r || l > b) return 0;
32     return getSum(left(si), l, (l + r) / 2, a, b) +
33            getSum(right(si), (l + r) / 2 + 1, r, a, b);
34 }

```

2.2 Segment Tree with Lazy Propagation

```

1  #define left(x) (x << 1)
2  #define right(x) ((x << 1) + 1)
3  typedef vector<int> vv;
4  vv st, lazy, arr;
5  void buildUtil(int si, int l, int r) {
6      if(l == r) st[si] = arr[r];
7      else {
8          buildUtil(left(si), l, (l + r) / 2);
9          buildUtil(right(si), (l + r) / 2 + 1, r);
10         // st[si] = st[left(si)] + st[right(si)];
11         st[si] = max(st[left(si)], st[right(si)]);
12     }
13 }
14 void build(int n) {
15     st.assign(4 * n, 0);
16     lazy.assign(4 * n, 0);
17     buildUtil(1, 0, n-1);
18 }
19 void update(int si, int l, int r, int a, int b, int value){

```

```

20     if(lazy[si] != 0) {
21         // st[si] += (r - l + 1) * lazy[si];
22         st[si] += lazy[si];
23         if(l != r) {
24             lazy[left(si)] += lazy[si];
25             lazy[right(si)] += lazy[si];
26         }
27         lazy[si] = 0;
28     }
29     if(a > r || b < l) return;
30     if(l >= a && r <= b) {
31         // st[si] += (r - l + 1) * value;
32         st[si] += value;
33         if(l != r) {
34             lazy[left(si)] += value;
35             lazy[right(si)] += value;
36         }
37     }
38     else {
39         update(left(si), l, (l + r) / 2, a, b, value);
40         update(right(si), (l + r) / 2 + 1, r, a, b, value);
41         // st[si] = st[left(si)] + st[right(si)];
42         st[si] = max(st[left(si)], st[right(si)]);
43     }
44 }
45 int getSum(int si, int l, int r, int a, int b) {
46     if(lazy[si] != 0) {
47         // st[si] += (r - l + 1) * lazy[si];
48         st[si] += lazy[si];
49         if(l != r) {
50             lazy[left(si)] += lazy[si];
51             lazy[right(si)] += lazy[si];
52         }
53         lazy[si] = 0;
54     }
55     if(l >= a && r <= b) return st[si];

```

```

56     if(a > r || l > b) return 0;
57     return max(getSum(left(si), l, (l + r) / 2, a, b),
58               getSum(right(si), (l + r) / 2 + 1, r, a, b));
59 }

```

2.3 Prefix Sum 2D

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

```

1  int v[100][100];
2  int p[101][101];
3
4  int getSum(int x, int y, int a, int b) {
5      return p[a][b] - p[a][y-1] - p[x-1][b] + p[x-1][y-1];
6  }
7  void prefixSum(int n, int m) {
8      for(int i=1; i <= n; ++i)
9          for(int j=1; j <=m; ++j)
10             p[i][j] = p[i-1][j] + v[i-1][j-1];
11
12     for(int i=1; i <= n; ++i)
13         for(int j=1; j <=m; ++j)
14             p[i][j] += p[i][j-1];
15 }

```

2.4 Binary Indexed Tree: Fenwick

Consultas RSQ e update em $\text{Log}(n)$

```

1  const long long MN = 100010;
2  long long BIT[MN], v[MN];
3  long long getSum(long long index)
4  {
5      long long sum = 0;
6      index = index + 1;
7      while (index>0) {
8          sum += BIT[index];

```



```

9             index -= index & (-index);
10        }
11        return sum;
12    }
13
14    void updateBIT(long long n, long long index, long long val) {
15        index = index + 1;
16        while (index <= n) {
17            BIT[index] += val;
18            index += index & (-index);
19        }
20    }

```

2.5 Binary Indexed Tree 2D: Fenwick

Coordenadas indexadas a partir de 1.

Consultas do somatório de uma região retangular em $\text{Log}(n)$.

```

1  #define MAXN 1010
2  #define swap(x, y)((x)^(y)^(x)^(y))
3  int bit[MAXN][MAXN];
4  int n, m;
5  int rsq(int i, int j) { // returns RSQ((1,1), (i,j))
6      int sum = 0, k = j;
7      for(; i > 0; i -= (i & -i)) {
8          j = k;
9          for(; j > 0; j -= (j & -j))
10             sum += bit[i][j];
11      }
12      return sum;
13  }
14  void update(int i, int j, int v) {
15      int k = j;
16      for(; i <= n; i += (i&-i)) {
17          for(j = k; j <= m; j += (j&-j))
18             bit[i][j] += v;

```

```

19         }
20     }
21     int getSum(int xa, int ya, int xb, int yb) {
22         if(xa > xb) swap(xa, xb);
23         if(ya > yb) swap(ya, yb);
24         return rsq(xb, yb) - rsq(xb, ya-1) - rsq(xa-1, yb) + rsq(xa-1, yb
                -1);
25     }

```

2.6 Sqrt Decomposition

Consultas em \sqrt{n} . Update $O(1)$. Pré processamento $O(n)$.

```

1  int arr[MAXN];                      // original array
2  int block[SQRSIZE];                 // decomposed array
3  int blk_sz;                         // block size
4  void update(int idx, int val) {
5      int blockNumber = idx / blk_sz;
6      block[blockNumber] += val - arr[idx];
7      arr[idx] = val;
8  }
9  int query(int l, int r) {
10     int sum = 0;
11     while (l < r and l%blk_sz != 0 and l != 0) {
12         sum += arr[l];
13         l++;
14     }
15     while (l+blk_sz <= r) {
16         sum += block[l/blk_sz];
17         l += blk_sz;
18     }
19     while (l <= r) {
20         sum += arr[l];
21         l++;
22     }
23     return sum;

```

24 }

2.7 Union Find

```
1  vector<int> rank;
2  vector<int> parent;
3  int find(int i) {
4      while(i != parent[i]) i = parent[i];
5      return i;
6  }
7  void unionSet(int i, int j) {
8      int x = find(i);
9      int y = find(j);
10     if(x == y) return;
11     if(rank[x] > rank[y]) parent[y] = x;
12     else {
13         parent[x] = y;
14         if(rank[x] == rank[y]) rank[y]++;
15     }
16 }
```

2.8 Ancestral Comum mais Próximo (LCA)

Consultas de LCA em $\log(n)$

```
1  #define left(x) (x << 1)
2  #define right(x) (x << 1) + 1
3  #define parent(x) (x >> 1)
4
5  const int MN = 1010;
6  int first[MN], vis[MN], height[MN];
7  vector<int> G[MN], euler, st;
8  void init() {
9      st.resize(euler.size() * 4);
10 }
11 void build(int s, int l, int r) {
```

```

12     if(l == r) st[s] = euler[l];
13     else {
14         build(left(s), l, (l+r)/2);
15         build(right(s), (l+r)/2 + 1, r);
16         int L = st[left(s)];
17         int R = st[right(s)];
18         st[s] = height[L] < height[R] ? L : R;
19     }
20 }
21 int query(int s, int l, int r, int a, int b) {
22     if(a > r || b < l) return -1;
23     if(l >= a && r <= b) return st[s];
24     int L = query(left(s), l, (l + r)/ 2, a, b);
25     int R = query(right(s), (l + r)/ 2 + 1, r, a, b);
26     if(L == -1) return R;
27     if(R == -1) return L;
28     return height[L] < height[R] ? L : R;
29 }
30 int LCA(int n, int a, int b) {
31     a = first[a]; b = first[b];
32     if(b < a) swap(a, b);
33     return query(1, 0, euler.size() - 1, a, b);
34 }
35 void dfs(int u, int h) {
36     vis[u] = true;
37     height[u] = h;
38     first[u] = euler.size();
39     euler.push_back(u);
40     for(auto to : G[u]) {
41         if(!vis[to]) {
42             dfs(to, h+1);
43             euler.push_back(u);
44         }
45     }
46 }
47 int main() {

```

```

48     dfs(0, 0);
49     init();
50     build(1, 0, euler.size() - 1);
51 }

```

2.9 Heavy Light Decomposition (HLD)

Atualiza um intervalo na arvore em $\text{Log}(n)$. Consulta valor do vértice em $\text{Log}(n)$.

```

1  const int MAXN = 5e3 + 10;
2  const int INF = 1e9 + 10;
3  #define left(x) (x << 1)
4  #define right(x) ((x << 1) + 1)
5  int ncha;
6  int parent[MAXN], fson[MAXN], size[MAXN];
7  int nchain[MAXN], id[MAXN], depth[MAXN], up[MAXN];
8  vector<int> G[MAXN], chain[MAXN];
9  class SegmentTree {
10 private:
11     vector<int> st, lazy;
12     int size;
13     void update(int si, int l, int r, int a, int b, int value) {
14         if(lazy[si] != 0) {
15             st[si] += (r - l + 1) * lazy[si];
16             if(l != r) {
17                 lazy[left(si)] += lazy[si];
18                 lazy[right(si)] += lazy[si];
19             }
20             lazy[si] = 0;
21         }
22         if(a > r || b < l) return;
23         if(l >= a && r <= b) {
24             st[si] += (r - l + 1) * value;
25             if(l != r) {
26                 lazy[left(si)] += value;
27                 lazy[right(si)] += value;

```

```

28         }
29     }
30     else {
31         update(left(si), l, (l + r) / 2, a, b, value);
32         update(right(si), (l + r) / 2 + 1, r, a, b, value);
33         st[si] = st[left(si)] + st[right(si)];
34     }
35 }
36 int query(int si, int l, int r, int a, int b) {
37     if(lazy[si] != 0) {
38         st[si] += (r - l + 1) * lazy[si];
39         if(l != r) {
40             lazy[left(si)] += lazy[si];
41             lazy[right(si)] += lazy[si];
42         }
43         lazy[si] = 0;
44     }
45     if(l >= a && r <= b) return st[si];
46     if(a > r || l > b) return 0;
47     return query(left(si), l, (l + r) / 2, a, b) +
48         query(right(si), (l + r) / 2 + 1, r, a, b);
49 }
50 public:
51     SegmentTree(int sz) {
52         size = sz;
53         st.assign(size * 4, 0);
54         lazy.assign(size * 4, 0);
55     }
56     int query(int a, int b) {
57         return query(1, 0, size - 1, a, b);
58     }
59     void update(int a, int b, int value) {
60         update(1, 0, size - 1, a, b, value);
61     }
62 };
63 int chainsz(int u, int p) {

```

```

64     size[u] = 1; fson[u] = -1; parent[u] = p;
65     int heavy = 0;
66     for(int i=0; i < (int)G[u].size(); ++i) {
67         int v = G[u][i];
68         if(v == p) continue;
69         size[u] += chainsz(v, u);
70         if(size[v] > heavy) {
71             fson[u] = v; heavy = size[v];
72         }
73     }
74     return size[u];
75 }
76 void build(int u, int ch, int h) {
77     nchain[u] = ch; id[u] = chain[ch].size();
78     chain[ch].push_back(u);
79     for(int i=0; i < (int)G[u].size(); ++i) {
80         int v = G[u][i];
81         if(v == parent[u]) continue;
82         if(v == fson[u]) build(v, ch, h + 1);
83         else {
84             up[ncha] = u; depth[ncha] = h;
85             chain[ncha].clear();
86             build(v, ncha++, h + 1);
87         }
88     }
89 }
90 vector<SegmentTree> hld;
91 void HLD(int root) {
92     chainsz(root, -1);
93     ncha = 0;
94     chain[ncha].clear();
95     up[ncha] = -1; depth[ncha] = 0;
96     build(root, ncha++, 1);
97
98     for(int i=0; i < ncha; ++i) {
99         hld.push_back(SegmentTree(chain[i].size()));

```

```

100     }
101 }
102 void update(int u, int v, int value) {
103     int cu = nchain[u], cv = nchain[v];
104     while(cu != cv) {
105         if(depth[cu] > depth[cv]) {
106             hld[cu].update(0, id[u], value);
107             u = up[cu];
108         }
109         else {
110             hld[cv].update(0, id[v], value);
111             v = up[cv];
112         }
113         cu = nchain[u]; cv = nchain[v];
114     }
115     if (id[u] < id[v]) {
116         hld[cu].update(id[u], id[v], value);
117     }
118     else {
119         hld[cu].update(id[v], id[u], value);
120     }
121 }
122
123 int custo[MAXN];
124 int agua[MAXN];
125
126 int main() {
127     int n, d, u, v, w, m, q;
128     scanf("%d %d", &n, &d);
129     for(int i=0; i < n -1; ++i) {
130         scanf("%d %d", &u, &v);
131         G[u].push_back(v);
132         G[v].push_back(u);
133     }
134     HLD(1);
135     memset(custo, INF, sizeof custo);

```



```

136     scanf("%d", &m);
137     for(int i=0; i < m; ++i) {
138         scanf("%d %d", &u, &v);
139         custo[u] = v;
140     }
141     scanf("%d", &q);
142     for(int i=0; i < q; ++i) {
143         scanf("%d %d %d", &u, &v, &w);
144         update(u, v, w);
145     }
146     for(int i=1; i <= n; ++i) {
147         agua[i] = hld[nchain[i]].query(id[i], id[i]);
148     }

```

Capítulo 3

Paradigmas

3.1 Problema da mochila

```
1  for(int i = 0; i <= n+1; ++i) {
2      for(int j = 0; j <= s; ++j) {
3          if(i == 0) dp[i][j] = 0;
4          else {
5              dp[i][j] = dp[i-1][j];
6              if(a[i] <= j) {
7                  dp[i][j] = max(dp[i][j], dp[i-1][j - a[i]] + b[i]);
8              }
9          }
10     }
11 }
12 cout << dp[n][s] << '\n';
```

3.2 Kadane

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

```
1  int kadane(int n){
2      int soma, ans;
3      soma = ans = 0;
4      for(int i=0; i < n; ++i) {
5          soma = soma + v[i];
6          ans = std::max(ans, soma);
7          if(soma < 0) soma = 0;
```

```

8     }
9     return ans;
10 }

```

3.3 LIS

```

1  int lis(vector<int> const& a) {
2      int n = a.size();
3      vector<int> d(n, 1);
4      for (int i = 0; i < n; i++) {
5          for (int j = 0; j < i; j++) {
6              if (a[j] < a[i])
7                  d[i] = max(d[i], d[j] + 1);
8          }
9      }
10     int ans = d[0];
11     for (int i = 1; i < n; i++) {
12         ans = max(ans, d[i]);
13     }
14     return ans;
15 }

```

3.4 LCS

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

```

1  for(int i = 0; i <= S.size(); ++i) {
2      for(int j = 0; j <= P.size(); ++j) {
3          if(i == 0 || j == 0) dp[i][j] = 0;
4          else if(S[i-1] == P[j-1]) dp[i][j] = dp[i-1][j-1] + 1;
5          else dp[i][j] = max(dp[i][j-1], dp[i-1][j]);
6      }
7  }
8  //ans = [S.size()][P.size()];

```

3.5 Contagem de inversões do Merge Sort

```
1  #define INF 1000000000
2  long long merge_sort(vector<long long> &v){
3      long long inv=0;
4      if(v.size()==1) return 0;
5      vector<long long> u1, u2;
6      for(long long i=0;i<v.size()/2;i++)
7          u1.push_back(v[i]);
8      for(long long i=v.size()/2;i<v.size();i++)
9          u2.push_back(v[i]);
10     inv+=merge_sort(u1);
11     inv+=merge_sort(u2);
12     u1.push_back(INF);
13     u2.push_back(INF);
14     long long ini1=0, ini2=0;
15     for(long long i=0;i<v.size();i++){
16         // Comparacao da ordenacao
17         if(u1[ini1]<=u2[ini2]){
18             v[i]=u1[ini1];
19             ini1++;
20         }
21         else{
22             v[i]=u2[ini2];
23             ini2++;
24             inv+=u1.size()-ini1-1;
25         }
26     }
27     return inv;
28 }
```

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, já 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.

```
1  #include <vector>
2  #define REP(i,n) for (int i = 1; i <= n; i++)
3  using namespace std;
4
5  typedef long long ll;
6  typedef vector<vector<ll> > matrix;
7  const ll MOD = 1000000007;
8  const int K = 2; //Numero de termos das quais f(n) depende
9  // computes A * B
10 matrix mul(matrix A, matrix B)
11 {
12     matrix C(K+1, vector<ll>(K+1));
```

```

13     REP(i, K) REP(j, K) REP(k, K)
14         C[i][j] = (C[i][j] + A[i][k] * B[k][j]) % MOD;
15     return C;
16 }
17 // computes A ^ p
18 matrix pow(matrix A, int p)
19 {
20     if (p == 1)
21         return A;
22     if (p % 2)
23         return mul(A, pow(A, p-1));
24     matrix X = pow(A, p/2);
25     return mul(X, X);
26 }
27 // returns the N-th term of Fibonacci sequence
28 int fib(int N) {
29     // create vector F1
30     vector<ll> F1(K+1);
31     F1[1] = 1;
32     F1[2] = 1;
33
34     // create matrix T
35     matrix T(K+1, vector<ll>(K+1));
36     T[1][1] = 0, T[1][2] = 1;
37     T[2][1] = 1, T[2][2] = 1;
38
39     // raise T to the (N-1)th power
40     if (N == 1)
41         return 1;
42     T = pow(T, N-1);
43
44     // the answer is the first row of T . F1
45     ll res = 0;
46     REP(i, K)
47         res = (res + T[1][i] * F1[i]) % MOD;
48     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

```
1  const int MN = 1000010;
2  int b[MN];
3  vector<int> ans;
4  void pre(string pattern) {
5      b[0] = -1;
6      int n = pattern.size();
7      for(int i = 0, j = -1; i < n;) {
8          while(j >= 0 && pattern[i] != pattern[j]) j = b[j];
9          b[++i] = ++j;
10     }
11 }
12 void KMP(string pattern, string text) {
13     pre(pattern);
14     for(int i = 0, j = 0; i < text.size(); ) {
15         while(j >= 0 && pattern[j] != text[i]) j = b[j];
16         ++i; ++j;
17         if(j == pattern.size()) ans.push_back(i);
18     }
19 }
```


Capítulo 6

Geométrico

6.1 Convex Hull

Solução do problema URI 1982.

```
1  #include <iostream>
2  #include <stack>
3  #include <math.h>
4  #include <stdlib.h>
5  using namespace std;
6
7  struct Point {
8      int x, y;
9  };
10
11  Point p0;
12  const int MN = 2010;
13  Point nextToTop(stack<Point> &S) {
14      Point p = S.top();
15      S.pop();
16      Point res = S.top();
17      S.push(p);
18      return res;
19  }
20  int swap(Point &p1, Point &p2) {
21      Point temp = p1;
```

```

22         p1 = p2;
23         p2 = temp;
24     }
25     int distSq(Point p1, Point p2) {
26         return (p1.x - p2.x)*(p1.x - p2.x) +
27             (p1.y - p2.y)*(p1.y - p2.y);
28     }
29     // To find orientation of ordered triplet (p, q, r).
30     // The function returns following values
31     // 0 --> p, q and r are colinear
32     // 1 --> Clockwise
33     // 2 --> Counterclockwise
34     int orientation(Point p, Point q, Point r)
35     {
36         int val = (q.y - p.y) * (r.x - q.x) -
37             (q.x - p.x) * (r.y - q.y);
38
39         if (val == 0) return 0; // colinear
40         return (val > 0)? 1: 2; // clock or counterclock wise
41     }
42
43     int compare(const void *vp1, const void *vp2) {
44     Point *p1 = (Point *)vp1;
45     Point *p2 = (Point *)vp2;
46     int o = orientation(p0, *p1, *p2);
47     if (o == 0)
48         return (distSq(p0, *p2) >= distSq(p0, *p1))? -1 : 1;
49     return (o == 2)? -1: 1;
50 }
51
52 double convexHull(Point points[], int n) {
53     int ymin = points[0].y, min = 0;
54     for (int i = 1; i < n; i++)
55     {
56         int y = points[i].y;
57         if ((y < ymin) || (ymin == y &&

```

```

58         points[i].x < points[min].x))
59         ymin = points[i].y, min = i;
60     }
61     swap(points[0], points[min]);
62     p0 = points[0];
63     qsort(&points[1], n-1, sizeof(Point), compare);
64     int m = 1;
65     for (int i=1; i<n; i++) {
66         while (i < n-1 && orientation(p0, points[i],points[i+1]) == 0) i++;
67         points[m] = points[i];
68         m++;
69     }
70     stack<Point> S;
71     S.push(points[0]);
72     S.push(points[1]);
73     S.push(points[2]);
74
75     for (int i = 3; i < m; i++)
76     {
77         while (orientation(nextToTop(S), S.top(), points[i]) != 2) S.pop()
78             ;
79         S.push(points[i]);
80     }
81     double custo = 0;
82     Point pivo = S.top(), q, p;
83     p = pivo;
84     S.pop();
85
86     while (!S.empty()) {
87         q = S.top(); S.pop();
88         custo += sqrt(pow(p.x - q.x, 2) + pow(p.y - q.y, 2));
89         p = q;
90     }
91     custo += sqrt(pow(pivo.x - q.x, 2) + pow(pivo.y - q.y, 2));
92     return custo;
93 }

```

```

93
94  int main()
95  {
96      Point Pontos[MN];
97      int n, u, v;
98      Point p;
99
100     while(true) {
101         cin >> n;
102         if(n == 0) break;
103
104         for(int i = 0; i < n; ++i) {
105             scanf("%d %d", &p.x, &p.y);
106             Pontos[i] = p;
107         }
108
109         printf("Tera que comprar uma fita de tamanho %.2lf.\n",
                convexHull(Pontos, n));
110     }
111     return 0;
112 }

```