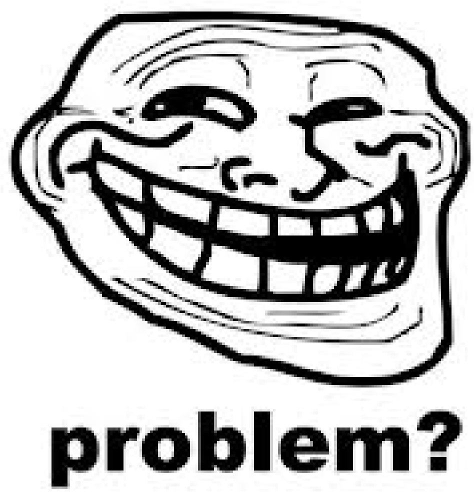
Tuxers Teambook

Sergio Guillen, Arnold Paye y Jhonatan Castro

August 6, 2013



Contents

Chapter 1

Sort Algorithms

1.1 Bubble sort

1.2 Insertion sort

1.3 Merge sort

1.4 Quick sort

1.5 Heap sort

Chapter 2

Number Theory

2.1 Prime sieves

2.1.1 Sieve of Eratosthenes

2.1.2 Sieve of Atkin

2.2 Primality Test

2.3 Greatest common divisor (GCD)

2.4 Lowest common multiple (LCM)

2.5 Combinatory

2.5.1 Fibonacci numbers

2.5.2 Catalan numbers

2.5.3 Binomial coefficients

2.6 Module aritmetics

2.7 Probability theory

2.8 Cycle-Finding

2.9 Game theory

2.9.1 Decission tree

2.9.2 Nim games

2.10 Square matrix

2.11 Matrix exponentiation

2.12 Karatsuba

2.13 Simpson's integration

2.14 Euler's phi

2.15 Factorial module ($n!$ mod p)

Chapter 3

Data Structures

import karma

jhtan

3.1 Fenwick tree (BIT)

3.2 Segment tree

```
int tree[400000];
int v[100000];

void init(int node, int a, int b) {
  if (a == b) {
    tree[node] = v[a];
    return;
  }
  init(2*node+1, a, (a + b)/2);
  init(2*node+2, (a+b)/2+1, b);
  tree[node] = tree[2*node+1] + tree[2*node+2];
}

int query(int node, int a, int b, int p, int q) {
  if (q < a || b < p) return 0; // return 0 for sum, 1 for product
  if (p <= a && b <= q) return tree[node];
  return query(2*node+1, a, (a+b)/2, p, q) + query(2*node+2, (a+b)/2+1, b, p, q);
}

void update(int node, int a, int b, int p, int val) {
  if (p < a || b < p) return;
  if (a == b) {
    tree[node] = val;
    return;
  }
  update(2*node+1, a, (a+b)/2, p, val);
  update(2*node+2, (a+b)/2+1, b, p, val);
  tree[node] = tree[2*node+1] + tree[2*node+2];
}
```

3.3 Trie

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

using namespace std;
```

```
struct trie{
  int words;
  int prefixes;
  struct trie *edges[26];
};

void init(trie *vertex) {
  vertex->words = 0;
  vertex->prefixes = 0;
  for(int i=0; i<26; i++) {
    vertex->edges[i] = NULL;
  }
}

void addWord(trie *vertex, string word) {
  if(word.length() == 0) {
    vertex->words++;
  } else {
    vertex->prefixes++;
    int k = word[0] - 'a';
    if(!vertex->edges[k]) {
      vertex->edges[k] = new trie();
      init(vertex->edges[k]);
    }
    addWord(vertex->edges[k], word.substr(1));
  }
}

int countWords(trie vertex, string word) {
  int k = word[0] - 'a';
  if(word.length() == 0) {
    return vertex.words;
  } else if(!vertex.edges[k]) {
    return 0;
  } else {
    return countWords(*vertex.edges[k], word.substr(1));
  }
}

int countPrefixes(trie vertex, string prefix) {
  int k = prefix[0] - 'a';
  if(prefix.length() == 0) {
    return vertex.prefixes;
  } else if(!vertex.edges[k]) {
    return 0;
  } else {
    return countPrefixes(*vertex.edges[k], prefix.substr(1));
  }
}

int main() {
  trie index;
  init(&index);

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

  string s;
  for(int i=0; i<n; i++) {
    cin >> s;
    addWord(&index, s);
  }

  cout << "There are " << countWords(index, "lol") << " lol words." << endl;
  cout << "There are " << countPrefixes(index, "lol") << " lol prefixes." << endl;

  return 0;
```

```
}
```

3.4 Union-Find-Disjoint set

```
int p[MAX];
void initSet(int n) {
  for (int i = 0; i < n ; i++) p[i] = i;
}
int findSet(int i) {
  return p[i] == i?i:p[i] = findSet(p[i]);
}
void unionSet(int i, int j) {
  p[findSet(i)] = findSet(j);
}
bool isSameSet(int i, int j) {
  return findSet(i) == findSet(j);
}
```

3.5 Binary search tree (BST)

3.6 Red black tree (BST)

3.7 Bit mask

Chapter 4

Dynamic Programming

Keep calm and code on

Churchil

4.1 Longest Increasing Subsequence (LIS)

```
int x[N];
int lis[N];
void lis() {
  lis[0] = 1;
  int maxim = 0;
  for (int i = 1; i < x.size(); i++) {
    int ma = 0;
    for (int j = 0; j < i; j++) if (x[j] < x[i]) ma = max(ma, lis[j]);
    lis[i] = 1 + ma;
    maxim = max(maxim, ma);
  }
}
```

4.2 Longest Common Subsequence (LCS)

4.3 Edit distance

4.4 Coin change

4.5 Max sum

4.6 1-0 Knapsack

4.7 Traveling Salesman Problem (TSP)

Chapter 5

Graphs

5.1 Depth First Search (DFS)

```
#define VISITED 1
#define NOT_VISITED 0
int n, e; // number of nodes and edges
vector<vi> graph; // adjacency list of the graph
int dfsm[MAX]; // max number of vertices in the graph

void dfs(int start) {
  dfsm[start] = VISITED;
  DBG(start);
  for (int i = 0; i < graph[start].size(); i++) {
    if(dfsm[graph[start][i]] == NOT_VISITED) {
      dfs(graph[start][i]);
    }
  }
}

int main() {
  scanf("%d %d", &n, &e);
  graph = vector<vi>(n);
  int ns, nt;
  while(e--) {
    scanf("%d %d", &ns, &nt);
    graph[ns].push_back(nt);
  }
  memset(dfsm, NOT_VISITED, sizeof dfsm);
  dfs(0);
  return 0;
}
```

5.1.1 Finding Connected Components in Undirect Graph

```
int n, e;
vector<vi> graph;
int dfsm[MAX];

void dfs(int start) {
  dfsm[start] = VISITED;
  cout << start << " ";
  for (int i = 0; i < graph[start].size(); i++) {
    if (dfsm[graph[start][i]] == NOT_VISITED) {
      dfs(graph[start][i]);
    }
  }
}

int main() {
```

```
  memset(dfsm, NOT_VISITED, sizeof dfsm);
  int numCC = 0;
  for (int i = 0; i < n; i++) {
    if (dfsm[i] == NOT_VISITED) {
      printf("Component %d:", ++numCC);
      dfs(i);
      printf("\n");
    }
  }
  return 0;
}
```

5.1.2 Flood Fill

```
#include <cstring>
#include <cstdio>

using namespace std;

int n;
int M[1001][1001];
bool B[1001][1001];

void ff(int i, int j) {
  B[i][j] = true;
  int X[8] = {0, 1, 1, 1, 0, -1, -1, -1};
  int Y[8] = {-1, -1, 0, 1, 1, 1, 0, -1};

  for(int k=0; k<8; k++) {
    int a = X[k] + i;
    int b = Y[k] + j;
    if(a >= 0 && b >= 0 && a < n && b < n) {
      if(!B[a][b] && M[i][j])
        ff(a, b);
    }
  }
}

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

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

  memset(B, false, sizeof(B));
  int c = 0;
  for(int i=0; i<n; i++) {
    for(int j=0; j<n; j++) {
      if(!B[i][j] && M[i][j]) {
        ff(i, j);
        c++;
      }
    }
  }

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

5.1.3 Flood FillsectionLabeling the Connected Components

5.1.4 Finding Articulation Points and Bridges [Hopcroft and Tarjan]

```
int dfsn[MAX];
int dfsl[MAX];
int dfsp[MAX];
```

```
int aVertex[MAX];
int dfsNumberCounter = 0;
int dfsRoot;
int rootChildren;

void articulationPointAndBridges(int u) {
  dfsl[u] = dfsn[u] = dfsNumberCounter++;
  for (int j = 0; j < graph[u].size(); j++) {
    if (dfsn[graph[u][j]] == NOT_VISITED) {
      dfsp[graph[u][j]] = u;
      if (u == dfsRoot) rootChildren++;
      articulationPointAndBridges(graph[u][j]);
      if (dfsl[graph[u][j]] >= dfsn[u])
        aVertex[u] = 1;
      if (dfsl[graph[u][j]] > dfsn[u])
        DBG(graph[u][j] << " " << u); // u and graph[u][j] are a bridge
      dfsl[u] = min(dfsl[u], dfsl[graph[u][j]]);
    } else if(graph[u][j] != dfsp[u]) {
      dfsl[u] = min(dfsl[u], dfsn[graph[u][j]]);
    }
  }
}

int main() {
  memset(dfsn, 0, sizeof dfsn);
  memset(dfsl, 0, sizeof dfsl);
  memset(dfsp, 0, sizeof dfsp);
  memset(aVertex, 0, sizeof aVertex);
  dfsNumberCounter = 0;
  for (int i = 0; i < n; i++) {
    if (dfsn[i] == NOT_VISITED) {
      dfsRoot = i;
      rootChildren = 0;
      articulationPointAndBridges(i);
      aVertex[dfsRoot] = (rootChildren > 1);
    }
  }
  for (int i = 0; i < n; i++) {
    if (aVertex[i])
      DBG(i); // i is a articulation point
  }
  return 0;
}
```

5.1.5 Finding Strongly Connected Components in Directed Graph [Kosaraju's][Tarjan]

5.1.6 Topological Sort (on a Directed Acyclic Graph)

5.1.7 Bipartite Graph Check (alway with BFS)

5.2 Breadth First Search (BFS)

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

typedef vector<int> vi;
#define pb push_back

vector<vi> G;
int dist[10010];
int parent[10010];

void bfs(int n) {
```

```
  queue<int> q;
  q.push(n);
  memset(dist, -1, sizeof(dist));
  memset(dist, -1, sizeof(parent));
  dist[n] = 0;

  while(!q.empty()) {
    int u = q.front();
    q.pop();

    for(int i=0; i<G[u].size(); i++) {
      if(dist[G[u][i]] == -1) {
        dist[G[u][i]] = dist[u] + 1;
        parent[G[u][i]] = u;
        q.push(G[u][i]);
      }
    }
  }
}

int main() {
  int v, e;
  scanf("%d %d", &v, &e);

  G.assign(v, vi());

  int a, b;
  for(int i=0; i<e; i++) {
    scanf("%d %d", &a, &b);
    G[a].pb(b);
  }

  bfs(0);

  printf("Distances\n");
  for(int i=0; i<v; i++)
    printf("%d ", dist[i]);
  printf("\n");

  printf("Parents\n");
  for(int i=0; i<v; i++)
    printf("%d ", parent[i]);
  printf("\n");

  return 0;
}
```

5.2.1 Graph Bicoloring

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

using namespace std;

typedef long long          ll;
typedef vector<int>        vi;
#define pb push_back

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

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

    vector<vi> G(n);
```

```
    int a, b;
    for(int i=0; i<m; i++) {
      scanf("%d %d", &a, &b);
      G[a].push_back(b);
      G[b].push_back(a);
    }

    bool sw = true;

    // BFS
    queue<int> Q;
    vi color(n, -1);
    Q.push(0);
    color[0] = 0;
    while(!Q.empty()) {
      int u = Q.front();
      Q.pop();

      for(int i=0; i<G[u].size(); i++) {
        if(color[G[u][i]] == -1) {
          color[G[u][i]] = (color[u]+1)%2;
          Q.push(G[u][i]);
        } else {
          if(color[G[u][i]] == color[u]) {
            sw = false;
            break;
          }
        }
      }

      if(!sw)
        break;
    }

    if(sw)
      printf("BICOLORABLE.\n");
    else
      printf("NOT BICOLORABLE.\n");

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

  return 0;
}
```

5.2.2 Finding Connected Components in Undirect Graph

```
int n, e;
vector<vi> graph;
int dfsm[MAX];

void dfs(int start) {
  dfsm[start] = VISITED;
  cout << start << " ";
  for (int i = 0; i < graph[start].size(); i++) {
    if (dfsm[graph[start][i]] == NOT_VISITED) {
      dfs(graph[start][i]);
    }
  }
}

int main() {
  memset(dfsm, NOT_VISITED, sizeof dfsm);
  int numCC = 0;
  for (int i = 0; i < n; i++) {
    if (dfsm[i] == NOT_VISITED) {
      printf("Component %d:", ++numCC);
```

```
      dfs(i);
      printf("\n");
    }
  }
  return 0;
}
```

5.2.3 Single-Source Shortest Paths (SSSP) on Unweighted Graph

5.3 Minimum Spanning Tree

5.3.1 [Kruskal's] Algorithm

5.3.2 [Prim's] Algorithm

5.3.3 'Maximum' Spanning Tree

5.3.4 Partial 'Minimum' Spanning Tree

5.3.5 Minimum Spanning 'Forest'

5.3.6 Second Best Spanning Tree

5.3.7 Minimax (and Maximim)

5.4 Single-Source Shortest Path (SSSP)

5.4.1 SSSP on Weighted Graph [Dijkstra's]

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

using namespace std;

typedef vector<int>        vi;
typedef pair<int, int>     ii;
typedef vector<ii>         vii;
#define pb push_back
#define INF 1000000000

vector<vii> G;
vi dist(10001, INF);
vi parent(10001, 0);

void dijkstra(int n) {
  dist[n] = 0;
  priority_queue<ii, vector<ii>, greater<ii> > pq;
  pq.push(ii(n, 0));

  while(!pq.empty()) {
    ii front = pq.top();
    pq.pop();
    int u = front.first, d = front.second;

    if(d > dist[u])
      continue;

    for(int i=0; i<G[u].size(); i++) {
      ii v = G[u][i];
      if(dist[u] + v.second < dist[v.first]) {
        dist[v.first] = dist[u] + v.second;
```

```
        parent[v.first] = u;
        pq.push(ii(v.first, dist[v.first]));
      }
    }
  }
}

int main() {
  int v, e;
  scanf("%d %d", &v, &e);

  G.assign(v, vii());
  int a, b, c;
  for(int i=0; i<e; i++) {
    scanf("%d %d %d", &a, &b, &c);
    G[a].pb(ii(b, c));
  }

  dijkstra(0);

  printf("Distances\n");
  for(int i=0; i<v; i++)
    printf("%d ", dist[i]);
  printf("\n");

  printf("Parents\n");
  for(int i=0; i<v; i++)
    printf("%d ", parent[i]);
  printf("\n");

  return 0;
}
```

5.4.2 Finding Connected Components in Undirect Graph

```
int n, e;
vector<vi> graph;
int dfsm[MAX];

void dfs(int start) {
  dfsm[start] = VISITED;
  cout << start << " ";
  for (int i = 0; i < graph[start].size(); i++) {
    if (dfsm[graph[start][i]] == NOT_VISITED) {
      dfs(graph[start][i]);
    }
  }
}

int main() {
  memset(dfsm, NOT_VISITED, sizeof dfsm);
  int numCC = 0;
  for (int i = 0; i < n; i++) {
    if (dfsm[i] == NOT_VISITED) {
      printf("Component %d:", ++numCC);
      dfs(i);
      printf("\n");
    }
  }
  return 0;
}
```

5.4.3 SSSP on Graph with Negative Weight Cycle [Bellman Ford's]

5.5 All-Pairs Shortest Paths

5.5.1 [Floyd Warshall's] Algorithm

5.5.2 Printing Shortest Paths

5.5.3 Transitive Closure [Warshall's]

5.5.4 Minimax and Maximim (Revisited)

5.5.5 Finding Negative Cicle

5.6 Maximum Flow

5.6.1 [Ford Fulkerson's] Algorithm

5.6.2 [Edmonds Karp's] Algorithm

5.6.3 Minimum Cut

5.6.4 Multi-source Multi-sink Max Flow

5.6.5 Max Flow with Vertex Capacities

5.6.6 Maximum Independent Paths

5.6.7 Maximum Edge-Disjoint Paths

5.6.8 Min Cost (Max) Flow

Chapter 6

String

6.1 KMP's Algorithm

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

using namespace std;

typedef vector<int> vi;
#define pb push_back

string s, t;
vi P;
vi M;

void KMPPreprocess() {
  P.assign(t.size() + 1, -1);
  for(int i=1; i<=t.size(); i++) {
    int pos = P[i-1];
    while(pos != -1 && t[pos] != t[i-1]) pos = P[pos];
    P[i] = pos + 1;
  }
}

void KMPSearch() {
  M.clear();
  for(int sp=0, kp=0; sp<s.size(); sp++) {
    while(kp != -1 && (kp == t.size() || t[kp] != s[sp]))
      kp = P[kp];
    kp++;
    if(kp == t.size()) M.pb(sp + 1 - t.size());
  }
}

int main() {
  cin >> s >> t;

  KMPPreprocess();
  KMPSearch();

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

  return 0;
}
```

6.2 Boyer-Moore's Algorithm

6.3 Rabin-Karp's Algorithm

6.4 Suffix tree

6.5 Suffix array

6.6 Aho-Corasick's Algorithm

6.7 Hashing

Chapter 7

Computational Geometry

I never program geometry problems, because there are better things to do with my life

Fidel Schaposnik

7.1 Trigonometric Functions

7.2 Geometry objects 2D

7.2.1 Point

7.2.2 Line

7.2.3 Circle

7.2.4 Polygon

7.3 Geometry objects 3D

7.3.1 Cube

7.3.2 Esphere

7.4 Distances

7.5 Intersection

7.5.1 Point of Intersection of Two Lines

7.6 Projection

7.7 Reflection

7.8 Polygon area

7.9 Transformations

7.9.1 Rotation

7.9.2 Line Reflection

7.9.3 Scale

7.9.4 Perpendicular Projection

7.9.5 Amood Monasef

7.10 Point in Triangle

7.11 Convex hull

7.11.1 Graham's Algorithms

Chapter 8

Others

8.1 Template

Default template for contests.

```
#include <iostream>
#include <cstdio>
#include <vector>
#include <algorithm>

#define TRvi(c, it) \
for (vi::iterator it = (c).begin(); it != (c).end(); it++)

using namespace std;

typedef vector<int> vi;

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

8.2 Probable Mistakes