Team Contest Reference

ChaosKITs
Karlsruhe Institute of Technology

26. Oktober 2014

Inhaltsverzeichnis

1 Max Flows **2**
1.1 Edmonds-Karp-Algorithmus 2

2 Geometry **2**
2.1 Closest Pair 2

3 Sonstiges **3**
3.1 2-SAT 3

1 Max Flows

1.1 Edmonds-Karp-Algorithmus

```
int s, t, f; //source, target, single flow
int res[MAX_V][MAX_V]; //adj-matrix
vector< vector<int> > adjList;
int p[MAX_V]; //bfs spanning tree

void augment(int v, int minEdge) {
  if (v == s) { f = minEdge; return; }
  else if (p[v] != -1) {
    augment(p[v], min(minEdge, res[p[v]][v]));
    res[p[v]][v] -= f; res[v][p[v]] += f;
}}

nt maxFlow() { //first inititalize res, adjList, s and t
  int mf = 0;
  while (true) {
    f = 0;
    bitset<MAX_V> vis; vis[s] = true;
    queue<int> q; q.push(s);
    memset(p, -1, sizeof(p));
    while (!q.empty()) { //BFS
      int u = q.front(); q.pop();
      if (u == t) break;
      for (int j = 0; j < (int)adjList[u].size(); j++) {
        int v = adjList[u][j];
        if (res[u][v] > 0 && !vis[v]) {
          vis[v] = true; q.push(v); p[v] = u;
    }}}

    augment(t, INF); //add found path to max flow
    if (f == 0) break;
    mf += f;
  }
  return maxFlow;
}
```

2 Geometry

2.1 Closest Pair

```
double squaredDist(point a, point b) {
  return (a.first-b.first) * (a.first-b.first) + (a.second-b.second) * (a.second-b.second);
}

bool compY(point a, point b) {
  if (a.second == b.second) return a.first < b.first;
  return a.second < b.second;
}

double shortestDist(vector<point> &points) {
  //check that points.size() > 1 and that ALL POINTS ARE DIFFERENT
  set<point, bool(*)(point, point)> status(compY);
  sort(points.begin(), points.end());
  double opt = 1e30, sqrtOpt = 1e15;
  auto left = points.begin(), right = points.begin();
  status.insert(*right); right++;

  while (right != points.end()) {
    if (fabs(left->first - right->first) >= sqrtOpt) {
      status.erase(*(left++));
    } else {
      auto lower = status.lower_bound(point(-1e20, right->second - sqrtOpt));
      auto upper = status.upper_bound(point(-1e20, right->second + sqrtOpt));
      while (lower != upper) {
        double cand = squaredDist(*right, *lower);
        if (cand < opt) {
```

```
27            opt = cand;
28            sqrtOpt = sqrt(opt);
29          }
30          ++lower;
31        }
32        status.insert(*(right++));
33      }
34    }
35    return sqrtOpt;
36  }
```

3 Sonstiges

3.1 2-SAT

```
1  vector< vector<int> > adjlist; //adjazenzliste
2  vector<int> sccs; //speichert die gefundenen SCCs
3  vector<bool> visited; //welcher Knoten ist schon besucht worden (in der DFS)
4  vector<bool> inStack; //ist Knoten gerade im Stack
5  vector<int> d; //discovery time
6  vector<int> low; //wie weit hoch geht's im Tiefensuchbaum
7  int counter; //Zaehler fuer discovery time
8  stack<int> st; //der Stack
9  int sccCounter; //Zaehler fuer SCCs
10
11 //Tiefensuche, die starke Zusammenhangskomponenten findet
12 void visit(int v) {
13   visited[v] = true;
14   d[v] = counter;
15   low[v] = counter;
16   counter++;
17   st.push(v);
18   inStack[v] = true;
19
20   for (int i = 0; i < (int)adjlist[v].size(); i++) {
21     int w = adjlist[v][i];
22
23     if (!visited[w]) {
24       visit(w);
25       low[v] = min(low[v], low[w]);
26     } else if (inStack[w]) {
27       low[v] = min(low[v], low[w]);
28     }
29   }
30
31   if (low[v] == d[v]) {
32     int next;
33     do {
34       next = st.top();
35       st.pop();
36       sccs[next] = sccCounter;
37       inStack[next] = false;
38     } while (next != v);
39
40     sccCounter++;
41   }
42 }
43
44 void solve() {
45   //adjlist initialisieren
46   //(a || b) wird zu (!a => b) und (!b => a)
47
48   visited.clear(); visited.assign(adjlist.size(), false);
49   inStack.clear(); inStack.assign(adjlist.size(), false);
50   d.clear(); d.assign(adjlist.size(), false);
51   low.clear(); low.assign(adjlist.size(), false);
52   sccs.clear(); sccs.resize(adjlist.size());
53
54   counter = 0;
55   sccCounter = 0;
```

```
56    for (i = 0; i < (int)adjlist.size(); i++) {
57      if (!visited[i]) {
58        visit(i);
59        sccCounter++;
60      }
61    }
62    // genau dann loesbar, wenn keine Variable in gleicher SCC wie Negation ist
63  }
```