

# 1 Max Flows

## 1.1 EDMONDS-KARP-Algorithmus

```

1 int s, t, f; //source, target, single flow
2 int res[MAX_V][MAX_V]; //adj-matrix
3 vector< vector<int> > adjList;
4 int p[MAX_V]; //bfs spanning tree
5
6 void augment(int v, int minEdge) {
7     if (v == s) { f = minEdge; return; }
8     else if (p[v] != -1) {
9         augment(p[v], min(minEdge, res[p[v]][v]));
10        res[p[v]][v] -= f; res[v][p[v]] += f;
11    }
12
13 int maxFlow() { //first initialize res, adjList, s and t
14     int mf = 0;
15     while (true) {
16         f = 0;
17         bitset<MAX_V> vis; vis[s] = true;
18         queue<int> q; q.push(s);
19         memset(p, -1, sizeof(p));
20         while (!q.empty()) { //BFS
21             int u = q.front(); q.pop();
22             if (u == t) break;
23             for (int j = 0; j < (int)adjList[u].size(); j++) {
24                 int v = adjList[u][j];
25                 if (res[u][v] > 0 && !vis[v]) {
26                     vis[v] = true; q.push(v); p[v] = u;
27                 }
28             }
29             augment(t, INF); //add found path to max flow
30             if (f == 0) break;
31             mf += f;
32         }
33     }
34     return mf;
35 }
```

# 2 Geometry

## 2.1 Closest Pair

```

1 double squaredDist(point a, point b) {
2     return (a.first-b.first) * (a.first-b.first) + (a.second-b.second) * (a.second-b.second);
3 }
4
5 bool compY(point a, point b) {
6     if (a.second == b.second) return a.first < b.first;
7     return a.second < b.second;
8 }
9
10 double shortestDist(vector<point> &points) {
11     //check that points.size() > 1 and that ALL POINTS ARE DIFFERENT
12     set<point, bool(*)>(point, point) status(compY);
13     sort(points.begin(), points.end());
14     double opt = 1e30, sqrtOpt = 1e15;
15     auto left = points.begin(), right = points.begin();
16     status.insert(*right); right++;
17
18     while (right != points.end()) {
19         if (fabs(left->first - right->first) >= sqrtOpt) {
20             status.erase(*left++);
21         } else {
22             auto lower = status.lower_bound(point(-1e20, right->second - sqrtOpt));
23             auto upper = status.upper_bound(point(-1e20, right->second + sqrtOpt));
24             while (lower != upper) {
25                 double cand = squaredDist(*right, *lower);
26                 if (cand < opt) {

```

```
27         opt = cand;
28         sqrt0pt = sqrt(opt);
29     }
30     ++lower;
31 }
32     status.insert(*(right++));
33 }
34 }
35 return sqrt0pt;
36 }
```