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;
}}

int main() { //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;
}}
//max flow in mf}
```

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;
}

void 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) {
```

```
27          double cand = squaredDist(*right, *lower);
28          if (cand < opt) {
29            opt = cand;
30            sqrtOpt = sqrt(opt);
31          }
32          ++lower;
33        }
34        status.insert(*(right++));
35      }
36    } // closest distance in sqrtOpt
37 }
```