```cpp
struct Edge {
  int from, to, cap, flow;
  Edge(int u, int v, int c, int f) : from(u), to(v), cap(c), flow(f) {}
};

bool operator<(const Edge& a, const Edge& b) {
  return a.from < b.from || (a.from == b.from && a.to < b.to);
}

struct ISAP {
  int n, m, s, t;
  vector<Edge> edges;
  vector<int> G[maxn];
  bool vis[maxn];
  int d[maxn];
  int cur[maxn];
  int p[maxn];
  int num[maxn];

  void AddEdge(int from, int to, int cap) {
    edges.push_back(Edge(from, to, cap, 0));
    edges.push_back(Edge(to, from, 0, 0));
    m = edges.size();
    G[from].push_back(m - 2);
    G[to].push_back(m - 1);
  }

  bool BFS() {
    memset(vis, 0, sizeof(vis));
    queue<int> Q;
    Q.push(t);
    vis[t] = 1;
    d[t] = 0;
    while (!Q.empty()) {
      int x = Q.front();
      Q.pop();
      for (int i = 0; i < G[x].size(); i++) {
        Edge& e = edges[G[x][i] ^ 1];
        if (!vis[e.from] && e.cap > e.flow) {
          vis[e.from] = 1;
          d[e.from] = d[x] + 1;
          Q.push(e.from);
        }
      }
    }
    return vis[s];
  }
```

```
void init(int n) {
  this->n = n;
  for (int i = 0; i < n; i++) G[i].clear();
  edges.clear();
}

int Augment() {
  int x = t, a = INF;
  while (x != s) {
    Edge& e = edges[p[x]];
    a = min(a, e.cap - e.flow);
    x = edges[p[x]].from;
  }
  x = t;
  while (x != s) {
    edges[p[x]].flow += a;
    edges[p[x] ^ 1].flow -= a;
    x = edges[p[x]].from;
  }
  return a;
}

int Maxflow(int s, int t) {
  this->s = s;
  this->t = t;
  int flow = 0;
  BFS();
  memset(num, 0, sizeof(num));
  for (int i = 0; i < n; i++) num[d[i]]++;
  int x = s;
  memset(cur, 0, sizeof(cur));
  while (d[s] < n) {
    if (x == t) {
      flow += Augment();
      x = s;
    }
    int ok = 0;
    for (int i = cur[x]; i < G[x].size(); i++) {
      Edge& e = edges[G[x][i]];
      if (e.cap > e.flow && d[x] == d[e.to] + 1) {
        ok = 1;
        p[e.to] = G[x][i];
        cur[x] = i;
        x = e.to;
        break;
      }
    }
    if (!ok) {
```

```
        int m = n - 1;
        for (int i = 0; i < G[x].size(); i++) {
          Edge& e = edges[G[x][i]];
          if (e.cap > e.flow) m = min(m, d[e.to]);
        }
        if (--num[d[x]] == 0) break;
        num[d[x] = m + 1]++;
        cur[x] = 0;
        if (x != s) x = edges[p[x]].from;
      }
    }
    return flow;
  }
};
```