Team Note of PetrSU QA

Compiled on November 26, 2018

Contents

1 Graph

1.1 Dinic

Usage: Almost linear in practice.
Time Complexity: $\mathcal{O}(n^2m)$

```
const int MAXE = 1e5, MAXV = 1e5;
const ll INF_FLOW = INF;

int edgeTo[MAXE], nextEdge[MAXE], E, edgeCap[MAXE];
int firstEdge[MAXV], firstEdgeTmp[MAXV], S, T;
int myQueue[MAXN], qHead, qTail, vertexLevel[MAXV];

void addEdge(int from, int to, ll cap) {
  edgeTo[E] = to,  nextEdge[E] = firstEdge[from];
  edgeCap[E] = cap, firstEdge[from] = E++;
  edgeTo[E] = from, nextEdge[E] = firstEdge[to];
  edgeCap[E] = 0, firstEdge[to] = E++;
}

void init() { E = 0; fill(firstEdge, firstEdge + MAXV, -1); }

bool buildLevelGraph() {
  qTail = qHead = 0;
  fill(vertexLevel, vertexLevel + MAXV, MAXV + 1);
  myQueue[qHead++] = S, vertexLevel[S] = 0;
  while(qTail != qHead) {
    int v = myQueue[qTail++];
    for(int id = firstEdge[v]; id != -1; id = nextEdge[id]) {
      int to = edgeTo[id];
      if(edgeCap[id] && vertexLevel[to] > vertexLevel[v] + 1) {
        vertexLevel[to] = vertexLevel[v] + 1, myQueue[qHead++] = to;
      }
    }
  }
  return vertexLevel[T] != MAXV + 1;
}

ll getBlockingFlow(int v, ll curFlow) {
  if(v == T || !curFlow) return curFlow;
  for(int &id = firstEdgeTmp[v]; id != -1; id = nextEdge[id]) {
    int to = edgeTo[id];
    if(vertexLevel[to] != vertexLevel[v] + 1 || !edgeCap[id])
    continue;
    ll newFlow = getBlockingFlow(to, min(edgeCap[id], curFlow));
    if(newFlow) {
      edgeCap[id] -= newFlow, edgeCap[id ^ 1] += newFlow;
      return newFlow;
    }
  }
  return 0;
}
```

```
ll maxFlow() {
  ll flow = 0, add = 0;
  while(buildLevelGraph()) {
    copy(firstEdge, firstEdge + MAXV, firstEdgeTmp);
    while((add = getBlockingFlow(S, INF_FLOW)))
      flow += add;
  }
  return flow;
}
```

2 Data Structure