Team Notebook

greedyLazy

September 27, 2024

Contents

1 Articular

```
void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    int children=0;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] >= tin[v] && p!=-1)
                IS_CUTPOINT(v);
            ++children;
        }
    }
    if(p == -1 && children > 1)
        IS_CUTPOINT(v);
}
```

2 Bellman-Ford

```
struct Edge {
    int a, b, cost;
};

int n, m;
vector<Edge> edges;
const int INF = 1000000000;

void solve()
{
    vector<int> d(n);
    vector<int> p(n, -1);
    int x;
    for (int i = 0; i < n; ++i) {
        x = -1;
        for (Edge e : edges) {
            if (d[e.a] + e.cost < d[e.b]) {
                d[e.b] = d[e.a] + e.cost;
                p[e.b] = e.a;
                x = e.b;
            }
        }
    }

    if (x == -1) {
        cout << "No negative cycle found.";
    } else {
        for (int i = 0; i < n; ++i)
            x = p[x];

        vector<int> cycle;
        for (int v = x;; v = p[v]) {
            cycle.push_back(v);
            if (v == x && cycle.size() > 1)
                break;
        }
        reverse(cycle.begin(), cycle.end());

        cout << "Negative cycle: ";
        for (int v : cycle)
            cout << v << ' ';
        cout << endl;
    }
```

3 Bridge

```
void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] > tin[v])
                IS_BRIDGE(v, to);
        }
    }
}
```

4 Kosaraju

```
vector<vector<int>> adj, adj_rev;
vector<bool> used;
vector<int> order, component;

void dfs1(int v) {
    used[v] = true;

    for (auto u : adj[v])
        if (!used[u])
            dfs1(u);

    order.push_back(v);
}

void dfs2(int v) {
    used[v] = true;
    component.push_back(v);

    for (auto u : adj_rev[v])
        if (!used[u])
            dfs2(u);
}

int main() {
    int n;
    // ... read n ...

    for (;;) {
        int a, b;
        // ... read next directed edge (a,b) ...
        adj[a].push_back(b);
        adj_rev[b].push_back(a);
    }

    used.assign(n, false);

    for (int i = 0; i < n; i++)
        if (!used[i])
            dfs1(i);

    used.assign(n, false);
    reverse(order.begin(), order.end());

    for (auto v : order)
        if (!used[v]) {
            dfs2 (v);

            // ... processing next component ...

            component.clear();
        }
}
```

5 Max Flow

```
// https://pastebin.com/exQM152L

template <typename T>
class flow_graph {
 public:
  static constexpr T eps = (T) 1e-9;

  struct edge {
    int to;
    T c;
    T f;
    int rev;
  };

  vector<vector<edge>> g;
  vector<int> ptr;
  vector<int> d;
  vector<int> q;
  vector<int> cnt_on_layer;
  vector<int> prev_edge;
  bool can_reach_sink;

  int n;
  int st, fin;
  T flow;

  flow_graph(int _n, int _st, int _fin) : n(_n), st(_st),
      fin(_fin) {
    assert(0 <= st && st < n && 0 <= fin && fin < n && st !=
        fin);
    g.resize(n);
    ptr.resize(n);
    d.resize(n);
    q.resize(n);
    cnt_on_layer.resize(n + 1);
    prev_edge.resize(n);
    flow = 0;
  }

  void clear_flow() {
    for (int i = 0; i < n; i++) {
      for (edge &e : g[i]) {
        e.f = 0;
      }
    }
    flow = 0;
  }

  void add(int from, int to, T forward_cap, T backward_cap)
      {
    assert(0 <= from && from < n && 0 <= to && to < n);
    int from_size = g[from].size();
    int to_size = g[to].size();
    g[from].push_back({to, forward_cap, 0, to_size});
    g[to].push_back({from, backward_cap, 0, from_size});
  }

  bool expath() {
    fill(d.begin(), d.end(), n);
    q[0] = fin;
    d[fin] = 0;
    fill(cnt_on_layer.begin(), cnt_on_layer.end(), 0);
    cnt_on_layer[n] = n - 1;
    cnt_on_layer[0] = 1;
    int beg = 0, end = 1;
    while (beg < end) {
      int i = q[beg++];
      for (const edge &e : g[i]) {
        const edge &back = g[e.to][e.rev];
        if (back.c - back.f > eps && d[e.to] == n) {
          cnt_on_layer[d[e.to]]--;
          d[e.to] = d[i] + 1;
          cnt_on_layer[d[e.to]]++;
          q[end++] = e.to;
        }
      }
    }
    return (d[st] != n);
  }

  T augment(int &v) {
    T cur = numeric_limits<T>::max();
    int i = fin;
    while (i != st) {
      const edge &e = g[i][prev_edge[i]];
      const edge &back = g[e.to][e.rev];
      cur = min(cur, back.c - back.f);
      i = e.to;
    }
    i = fin;
    while (i != st) {
      edge &e = g[i][prev_edge[i]];
      edge &back = g[e.to][e.rev];
      back.f += cur;
      e.f -= cur;
      i = e.to;
      if (back.c - back.f <= eps) {
        v = i;
      }
    }
    return cur;
  }

  int retreat(int v) {
    int new_dist = n - 1;
    for (const edge &e : g[v]) {
      if (e.c - e.f > eps && d[e.to] < new_dist) {
        new_dist = d[e.to];
      }
    }
    cnt_on_layer[d[v]]--;
    if (cnt_on_layer[d[v]] == 0) {
      if (new_dist + 1 > d[v]) {
        can_reach_sink = false;
      }
    }
    d[v] = new_dist + 1;
    cnt_on_layer[d[v]]++;
    if (v != st) {
      v = g[v][prev_edge[v]].to;
    }
    return v;
  }

  T max_flow() {
    can_reach_sink = true;
    for (int i = 0; i < n; i++) {
      ptr[i] = (int) g[i].size() - 1;
    }
    if (expath()) {
      int v = st;
      while (d[st] < n) {
        while (ptr[v] >= 0) {
          const edge &e = g[v][ptr[v]];
          if (e.c - e.f > eps && d[e.to] == d[v] - 1) {
            prev_edge[e.to] = e.rev;
            v = e.to;
            if (v == fin) {
              flow += augment(v);
            }
            break;
          }
          ptr[v]--;
        }
        if (ptr[v] < 0) {
          ptr[v] = (int) g[v].size() - 1;
          v = retreat(v);
          if (!can_reach_sink) {
```

```
          break;
        }
      }
    }
   }
   return flow;
  }

  vector<bool> min_cut() {
   max_flow();
   assert(!expath());
   vector<bool> ret(n);
   for (int i = 0; i < n; i++) {
    ret[i] = (d[i] != n);
   }
   return ret;
  }
};
```

6 TwoSatSolver

```
struct TSS {
 int nvar;
 int nvrt;
 vector<vector<int>> adj;
 vector<int> res, scc, topo, vis, in, low, del, idx;

 TSS(int nvar):
  nvar(nvar),
  nvrt(nvar * 2),
  adj(nvrt + 1),
  res(nvar + 1),
  scc(nvrt + 1),
  in(nvrt + 1),
  low(nvrt + 1),
  del(nvrt + 1),
  idx(nvrt + 1) {}

 int conj(int u) {
  if (u > nvar) return u - nvar;
  return u + nvar;
 }
 void edge(int u, bool nu, int v, bool nv) {
  if (nu) u = conj(u);
  if (nv) v = conj(v);
  adj[u].push_back(v);
 }
 int curdfs = 0;
 stack<int> st;
 int curidx = nvrt;
 void tarjan(int u) {
  in[u] = low[u] = ++curdfs;
  st.push(u);
  for (int v : adj[u]) {
   if (del[v]) continue;
   if (!in[v]) {
    tarjan(v);
    low[u] = min(low[u], low[v]);
   }
   else {
    low[u] = min(low[u], in[v]);
   }
  }
  if (low[u] == in[u]) {
   idx[u] = curidx--; ;
   while (st.top() != u) {
    int v = st.top();
    st.pop();
    del[v] = true;
    scc[v] = u;
   }
   scc[u] = u;
   del[u] = true;
   st.pop();
  }
 }
 bool solve() {
  for (int i = 1; i <= nvrt; i++) {
   if (!in[i]) tarjan(i);
  }
  for (int i = 1; i <= nvar; i++) {
   if (scc[i] == scc[conj(i)]) return false;
   res[i] = idx[scc[i]] > idx[scc[conj(i)]];
  }
  return true;
 }
};
```