Standard Code Library

05 – Graph Theory

HeartFireY, eroengine, yezzz

Jiangxi Normal University

Ver.2021-EC-FINAL(2022.7.17)

Contents

图论 **2**
1.LCA 2
倍增 2
HLD 树剖 2
2.Kruskal 重构树 3
3.KM 二分图最大权匹配 4
4. 二分图匹配 4
5. 拓扑排序 5
6.EK+SPFA 5
7.Dinic 7

图论

1.LCA

倍增

```cpp
int n;
vector<int>to[N];
int rt;
int depth[N];
int fa[N][21];//fa[u][i] 表示 u 开始向上 2^i 的父节点
void dfs_lca(int u, int father) {
    depth[u] = depth[father] + 1;
    fa[u][0] = father;
    for (int i = 1; (1 << i) <= depth[u]; i++) {
        fa[u][i] = fa[fa[u][i - 1]][i - 1];
    }
    for (auto ed : to[u]) {
        int v = ed;
        if (v != father) {
            dfs_lca(v, u);
        }
    }
}
int lca(int u,int v) {
    if (depth[u] < depth[v])swap(u, v);
    for (int k = 20; k >= 0; k--) {
        if (depth[fa[u][k]] >= depth[v]) {
            u = fa[u][k];
        }
    }
    if (u == v) {
        return u;
    }
    for (int k = 20; k >= 0; k--) {
        if (fa[u][k] != fa[v][k]) {
            u = fa[u][k];
            v = fa[v][k];
        }
    }
    return fa[u][0];
}
void solve() {
    cin >> n;
    for (int i = 1; i <= n; i++) {
        int u, v;
        cin >> u >> v;
        if (v == -1) {
            rt = u;
        }
        to[u].pb(v);
        to[v].pb(u);
    }
    dfs_lca(rt, 0);
    int q;
    cin >> q;
    while (q--) {
        int u, v;
        cin >> u >> v;
        cout << lca(u, v) << "\n";
    }
}
```

HLD 树剖

```cpp
#include <bits/stdc++.h>

#define maxm 200010
namespace LCA{
    struct edge{ int to, len, next; } E[maxm];
    int cnt, last[maxm], fa[maxm], top[maxm], deep[maxm], siz[maxm], son[maxm], val[maxm];
```

```
    void addedge(int a, int b, int len = 0){
        E[++cnt] = (edge){b, len, last[a]}, last[a] = cnt;
    }
    void dfs1(int x){
        deep[x] = deep[fa[x]] + 1;
        siz[x] = 1;
        for (int i = last[x]; i; i = E[i].next){
            int to = E[i].to;
            if (fa[x] != to && !fa[to]){
                val[to] = E[i].len;
                fa[to] = x;
                dfs1(to);
                siz[x] += siz[to];
                if (siz[son[x]] < siz[to]) son[x] = to;
            }
        }
    }
    void dfs2(int x){
        if (x == son[fa[x]]) top[x] = top[fa[x]];
        else top[x] = x;
        for (int i = last[x]; i; i = E[i].next)
            if (fa[E[i].to] == x) dfs2(E[i].to);
    }
    void init(int root) { dfs1(root), dfs2(root); }
    int query(int x, int y){
        for (; top[x] != top[y]; deep[top[x]] > deep[top[y]] ? x = fa[top[x]] : y = fa[top[y]]);
        return deep[x] < deep[y] ? x : y;
    }
}
int n, m, x, y, v;
int main(){
    scanf("%d%d", &n, &m);
    for (int i = 1; i < n; i++){
        scanf("%d%d", &x, &y);
        LCA::addedge(x, y, v);
        LCA::addedge(y, x, v);
    }
    LCA::init(1);
    for (int i = 1; i <= m; i++){
        scanf("%d%d", &x, &y);
        printf("%d\n", LCA::query(x, y));
    }
    return 0;
}
```

2.Kruskal 重构树

```
const int N = 1e5 + 10;
int n = 0, m = 0;

namespace Graph
{
    struct edge { int to, nxt, val; } edges[N << 1];
    int cnt, head[N << 1], val[N << 1];
    void add(int u, int v, int val = 0){
        edges[++cnt] = (edge){v, head[u], val};
        head[u] = cnt;
    }
} // namespace Graph

namespace KR{
    using Graph::add;
    struct edge{
        int u, v, w;
        const bool operator< (const edge &x) const { return w < x.w; }
    } edges[N] ;
    int fa[N];
    void init(int n){ for(int i = 1; i <= n; i++) fa[i] = i; }
    int find(int x){ return fa[x] == x ? x : (fa[x] = find(fa[x])); }


```

```
    void kruskal(){
        int tot = 0, cnt = n;
        sort(edges + 1, edges + 1 + m);
        for(int i = 1; i <= m; i++){
            int fau = find(edges[i].u), fav = find(edges[i].v);
            if(fau != fav){
                cnt++, fa[fau] = fa[fav] = cnt;
                add(fau, cnt), add(cnt, fau);
                add(fav, cnt), add(cnt, fav);
                Graph::val[cnt] = edges[i].w;
                tot++;
            }
            if(tot == n - 1) break;
        }

    }
} // namespace KR
```

3.KM 二分图最大权匹配

```
namespace RMatch{
    #define LL long long
    const int M = 400 + 5, INF = 2E9;
    int n = 0; //! Attention-Outside !//
    int w[M][M], kx[M], ky[M], linky[M], vy[M], slk[M], pre[M];

    int KM(){
        for(int i = 1; i <= n; i++)
            for(int j = 1; j <= n; j++) kx[i] = max(kx[i], w[i][j]);
        for(int i = 1; i <= n; i++) {
            fill(vy, vy + n + 1, 0);
            fill(slk, slk + n + 1, INF);
            fill(pre, pre + n + 1, 0);
            int k = 0, p = -1;
            for(linky[k = 0] = i; linky[k]; k = p){
                vy[k] = 1;
                int d = INF, x = linky[k];
                for(int j = 1; j <= n; j++)
                    if (!vy[j]) {
                        int t = kx[x] + ky[j] - w[x][j];
                        if (t < slk[j]) { slk[j] = t; pre[j] = k; }
                        if (slk[j] < d) { d = slk[j]; p = j; }
                    }
                for(int j = 0; j <= n; j++)
                    if (vy[j]) { kx[linky[j]] -= d; ky[j] += d; }
                    else slk[j] -= d;
            }
            for(; k; k = pre[k]) linky[k] = linky[pre[k]];
        }
        int ans = 0;
        for(int i = 1; i <= n; i++) ans += kx[i] + ky[i];
        return ans;
    }

    inline void add(int x, int y, int val){ w[x][y] = val; }
}
```

4. 二分图匹配

```
/*
! + 最小覆盖数 = 最大匹配数
! + 最大独立集 = 顶点数 - 二分图匹配数
! + DAG 最小路径覆盖数 = 结点数 - 拆点后二分图最大匹配数
*/

const int N = 1e3 + 10;
struct MaxMatch {
    int n;
    vector<int> g[N];
    int vis[N], left[N], clk;
```

```

    void init(int n) {
        this->n = n;
        for(int i = 0; i <= n; i++) g[i].clear();
        memset(left, -1, sizeof left);
        memset(vis, -1, sizeof vis);
    }

    bool dfs(int u) {
        for (int v: g[u])
            if (vis[v] != clk) {
                vis[v] = clk;
                if (left[v] == -1 || dfs(left[v])) {
                    left[v] = u;
                    return true;
                }
            }
        return false;
    }

    int match() {
        int ret = 0;
        for (clk = 0; clk <= n; ++clk)
            if (dfs(clk)) ++ret;
        return ret;
    }
};
```

5. 拓扑排序

```
const int N = 1e5 + 10;
std::vector<int> g[N];
int in_cnt[N];

bool TopoSort(int n){
    std::vector<int> ans;
    std::queue<int> q;
    for(int i = 1; i <= n; i++) if(!in_cnt[i]) q.emplace(i);
    while(q.size()){
        int now = q.front(); q.pop();
        ans.emplace_back(now);
        for(auto nxt : g[now]) if(!--in_cnt[nxt]) q.emplace(nxt);
    }
    if(ans.size() == n){
        for(auto ansi : ans) cout << ansi << ' ';
        return true;
    }
    else return false;
}

inline void solve(){
    int n, m; std::cin >> n >> m;
    for(int i = 1; i <= m; i++){
        int u, v; std::cin >> u >> v;
        g[u].emplace_back(v);
        in_cnt[v]++;
    }
    bool status = TopoSort(n);
}
```

6.EK+SPFA

```
#include <bits/stdc++.h>
//#define int long long
using namespace std;

const int N = 5100, M = 20010, INF = 1e8;
int n, m, s, t;
template <int N,int M>struct MCMF{
    //EK+spfa
```

```
    const int INF=1e8;
    int S,T;
    MCMF(int s,int t):S(s),T(t){}
    int h[N], e[M], f[M], w[M], ne[M], idx;
    int q[N], d[N], pre[N], incf[N];
    bool st[N];

    void add(int a, int b, int c, int d)
    {
        e[idx] = b, f[idx] = c, w[idx] = d, ne[idx] = h[a], h[a] = idx ++ ;
        e[idx] = a, f[idx] = 0, w[idx] = -d, ne[idx] = h[b], h[b] = idx ++ ;
    }

    bool spfa()
    {
        int hh = 0, tt = 1;
        memset(d, 0x3f, sizeof d);
        memset(incf, 0, sizeof incf);
        q[0] = S, d[S] = 0, incf[S] = INF;
        while (hh != tt)
        {
            int t = q[hh ++ ];
            if (hh == N) hh = 0;
            st[t] = false;

            for (int i = h[t]; ~i; i = ne[i])
            {
                int ver = e[i];
                if (f[i] && d[ver] > d[t] + w[i])
                {
                    d[ver] = d[t] + w[i];
                    pre[ver] = i;
                    incf[ver] = min(f[i], incf[t]);
                    if (!st[ver])
                    {
                        q[tt ++ ] = ver;
                        if (tt == N) tt = 0;
                        st[ver] = true;
                    }
                }
            }
        }

        return incf[T] > 0;
    }

    void EK(int& flow, int& cost)
    {
        flow = cost = 0;
        while (spfa())
        {
            int t = incf[T];
            flow += t, cost += t * d[T];
            for (int i = T; i != S; i = e[pre[i] ^ 1])
            {
                f[pre[i]] -= t;
                f[pre[i] ^ 1] += t;
            }
        }
    }
};
signed main()
{
    int x;
    cin>>n;
    s=0,t=n+n+1;
    MCMF<N,M>mcmf1(s,t);
    MCMF<N,M>mcmf2(s,t);

    memset(mcmf1.h, -1, sizeof mcmf1.h);
    for(int i=1;i<=n;i++){
```

```
        for(int j=1;j<=n;j++){
            cin>>x;
            mcmf1.add(i,j+n,1,x);
            mcmf2.add(i,j+n,1,-x);
        }
        mcmf1.add(0,i,1,0);
        mcmf1.add(i+n,n+n+1,1,0);
        mcmf2.add(0,i,1,0);
        mcmf2.add(i+n,n+n+1,1,0);
    }

    int flow, cost,flow2,cost2;
    mcmf1.EK(flow, cost);
    //mcmf2.EK(flow2, cost2);
    printf("%d %d\n",cost,-cost2);

    return 0;
}
```

7.Dinic

```
#include <bits/stdc++.h>
#define int long long
using namespace std;
template <int N> struct Dinic {
    const int INF = 1e9;
    struct E {
        int to, cap, rev;
    };
    vector<E> G[N];
    int lev[N], cur[N];
    inline void add(int x, int y, int c) {
        G[x].push_back({ y, c, (int)G[y].size() });
        G[y].push_back({ x, 0, (int)G[x].size() - 1 });
    }
    void bfs(int s) {
        queue<int> q;
        memset(lev, -1, sizeof lev);

        for (lev[s] = 0, q.push(s); q.size();) {
            int x = q.front();
            q.pop();

            for (auto &e : G[x])
                if (e.cap > 0 && lev[e.to] < 0)
                    lev[e.to] = lev[x] + 1, q.push(e.to);
        }
    }
    int dfs(int x, int t, int f) {
        if (x == t)
            return f;

        for (int &i = cur[x], sz = G[x].size(), d; i < sz; i++) {
            auto &e = G[x][i];

            if (e.cap > 0 && lev[x] < lev[e.to]) {
                if ((d = dfs(e.to, t, min(f, e.cap))) > 0) {
                    e.cap -= d, G[e.to][e.rev].cap += d;
                    return d;
                }
            }
        }

        return 0;
    }
    int64_t maxflow(int s, int t) {
        for (int64_t flow = 0, f;;) {
            bfs(s);

            if (lev[t] < 0)
                return flow;
```

```
51
52              memset(cur, 0, sizeof cur);
53
54              while ((f = dfs(s, t, INF)) > 0)
55                  flow += f;
56          }
57      }
58  };
59  Dinic<1005> din;
60  signed main() {
61      cin.tie(0)->sync_with_stdio(0);
62      int n, m, s, t;
63      cin >> n ;
64      for(int i=1;i<=n;i++){
65          for(int j=1;j<=n;j++){
66              int x;
67              cin>>x;
68              din.add(0,i,x);
69              din.add(j+100,201,x);
70              din.add(i,j+100,1e9);
71          }
72      }
73      return cout << din.maxflow(0, 201), 0;
74  }
```