ICPC Template Manual

作者: 贺梦杰

August 6, 2019

Contents

Chapter 1

图论

1.1 最短路

1.1.1 单源最短路径

1.1.1.1 Dijkstra

```
void Dijkstra()
{
    memset(dist, 0x3f, sizeof(dist));
    memset(vis, 0, sizeof(vis));
    priority_queue<pii, vector<pii>, greater<pii>> q;
    dist[1] = 0;
    q.push({dist[1], 1});
    while (!q.empty())
    {
        int x = q.top().second;
        q.pop();
        if (!vis[x])
        {
            vis[x] = 1;
            for (auto it : v[x])
            {
                int y = it.first;
                if (dist[y] > dist[x] + it.second)
                {
                    dist[y] = dist[x] + it.second;
                    q.push({dist[y], y});
                }
            }
        }
    }
}
```

1.1.1.2 Bellman-Ford 和 SPFA

```
void SPFA()
{
    memset(dis, 0x3f, sizeof(dis));
    memset(vis, 0, sizeof(vis));
    queue<int> q;
    dis[1] = 0;
    vis[1] = 1;
    q.push(1);
    while (!q.empty())
    {
        int x = q.front();
        q.pop();
        vis[x] = 0;
        for (int i = 0; i < v[x].size(); i++)
        {
            int y = v[x][i].first;
            int z = v[x][i].second;
            if (dis[y] > dis[x] + z)
```

```
            {
                dis[y] = dis[x] + z;
                if (!vis[y])
                    q.push(y), vis[y] = 1;
            }
        }
    }
}
```

例题分析

POJ3662 Telephone Lines（分层图最短路/二分答案，双端队列 BFS）

P1073 最优贸易（原图与反图，枚举节点）

P3008 [USACO11JAN] 道路和飞机 Roads and Planes（DAG，拓扑序，连通块）

1.1.2 任意两点间最短路径

1.1.2.1 Floyd

```
void get_path(int i, int j)
{
    if (!path[i][j])
        return;
    get_path(i, path[i][j]);
    p.push_back(path[i][j]);
    get_path(path[i][j], j);
}
void Floyd()
{
    memcpy(d, a, sizeof(d));
    for (int k = 1; k <= n; k++)
    {
        for (int i = 1; i < k; i++)
        {
            for (int j = i + 1; j < k; j++)
            {
                //注意溢出
                ll temp = d[i][j] + a[i][k] + a[k][j];
                if (ans > temp)
                {
                    ans = temp;
                    p.clear();
                    p.push_back(i);
                    get_path(i, j);
                    p.push_back(j);
                    p.push_back(k);
                }
            }
        }
        for (int i = 1; i <= n; i++)
        {
            for (int j = 1; j <= n; j++)
            {
                ll temp = d[i][k] + d[k][j];
                if (d[i][j] > temp)
```

```
                {
                    d[i][j] = temp;
                    path[i][j] = k;
                }
            }
        }
    }
}
```

例题分析

POJ1094 Sorting It All Out（传递闭包）

POJ1734 Sightseeing trip（无向图最小环）

POJ3613 Cow Relays（离散化，广义矩阵乘法，快速幂）

1.2 最小生成树

1.2.1 Kruskal

基于并查集

```
void Init()
{
    for (int i = 1; i <= n; i++)
        fa[i] = i;
}
int Find(int x)
{
    if (x == fa[x])
        return x;
    return fa[x] = Find(fa[x]);
}
void Kruskal()
{
    Init();
    sort(e.begin(), e.end());
    int ans=0;
    for (int i = 0; i < e.size(); i++)
    {
        int u = e[i].u, v = e[i].v;
        int fu = Find(u), fv = Find(v);
        if (fu != fv)
        {
            fa[fu] = fv;
            ans += e[i].w;
        }
    }
}
```

1.2.2 Prim

```
void Prim()
{
    memset(vis, 0, sizeof(vis));
    memset(d, 0x3f, sizeof(d));
    d[1] = 0;
    int temp = n;
    int ret = 0;
    while (temp--)
    {
        int min_pos = 0;
        for (int i = 1; i <= n; i++)
            if (!vis[i] && (!min_pos || d[i] < d[min_pos]))
                min_pos = i;
        if (min_pos)
        {
            vis[min_pos] = 1;
            ret += d[min_pos];
```

```cpp
            for (int i = 1; i <= n; i++)
                if (!vis[i]) d[i] = min(d[i], weight[min_pos][i]);
        }
    }
}
```

例题分析

走廊泼水节（Kruskal，最小生成树扩充为完全图）

POJ1639 Picnic Planning（度限制最小生成树，连通块，树形 DP）

```cpp
#include <algorithm>
#include <cstring>
#include <iostream>
#include <map>
#include <string>
#include <vector>
using namespace std;
#define inf 0x3f3f3f3f
#define N 25
#define M 500
map<string, int> name;
struct edge
{
    int u, v, w;
    bool operator<(const edge &e) const
    {
        return w < e.w;
    }
};
int n, s, ptot = 0, a[N][N], ans, fa[N], d[N], ver[N];
vector<edge> e;
bool vis[N][N];
edge dp[N]; //dp[i]  1...i路径上的最大边
void Init()
{
    for (int i = 1; i <= ptot; i++)
        fa[i] = i;
}
int Find(int x)
{
    if (x == fa[x])
        return x;
    return fa[x] = Find(fa[x]);
}
void Kruskal()
{
    Init();
    sort(e.begin(), e.end());
    for (int i = 0; i < e.size(); i++)
    {
        int u = e[i].u, v = e[i].v;
        if (u != 1 && v != 1)
        {
            int fu = Find(u), fv = Find(v);
            if (fu != fv)
```

```
            {
                fa[fu] = fv;
                vis[u][v] = vis[v][u] = 1;
                ans += e[i].w;
            }
        }
    }
}
void DFS(int cur, int pre)
{
    for (int i = 2; i <= ptot; i++)
    {
        if (i != pre && vis[cur][i])
        {
            if (dp[i].w == -1)
            {
                if (dp[cur].w < a[cur][i])
                {
                    dp[i].u = cur;
                    dp[i].v = i;
                    dp[i].w = a[cur][i];
                }
                else
                    dp[i] = dp[cur];
            }
            DFS(i, cur);
        }
    }
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    cin >> n;
    string s1, s2;
    int len;
    name["Park"] = ++ptot;
    memset(a, 0x3f, sizeof(a));
    memset(d, 0x3f, sizeof(d));
    //Park: 1
    for (int i = 0; i < n; i++)
    {
        cin >> s1 >> s2 >> len;
        if (!name[s1])
            name[s1] = ++ptot;
        if (!name[s2])
            name[s2] = ++ptot;
        int u = name[s1], v = name[s2];
        a[u][v] = a[v][u] = min(a[u][v], len); //无向图邻接矩阵
        e.push_back({u, v, len});
    }
    cin >> s; //度数限制
    ans = 0;
```

```
    Kruskal();
    for (int i = 2; i <= ptot; i++)
    {
        if (a[1][i] != inf)
        {
            int rt = Find(i);
            if (d[rt] > a[1][i])
                d[rt] = a[1][i], ver[rt] = i;
        }
    }
    for (int i = 2; i <= ptot; i++)
    {
        if (d[i] != inf)
        {
            s--;
            ans += d[i];
            vis[1][ver[i]] = vis[ver[i]][1] = 1;
        }
    }
    while (s-- > 0)
    {
        memset(dp, -1, sizeof(dp));
        dp[1].w = -inf;
        for (int i = 2; i <= ptot; i++)
        {
            if (vis[1][i])
                dp[i].w = -inf;
        }
        DFS(1, -1);
        int w = -inf;
        int v;
        for (int i = 2; i <= ptot; i++)
        {
            if (w < dp[i].w - a[1][i])
            {
                w = dp[i].w - a[1][i];
                v = i;
            }
        }
        if (w <= 0)
            break;
        ans -= w;
        vis[1][v] = vis[v][1] = 1;
        vis[dp[v].u][dp[v].v] = vis[dp[v].v][dp[v].u] = 0;
    }
    cout << "Total miles driven: " << ans << endl;
    system("pause");
    return 0;
}
```

POJ2728 Desert King （最优比率生成树，0/1 分数规划，二分）
黑暗城堡（最短路径生成树计数，最短路，排序）

1.3 树的直径

1.3.1 树形 DP 求树的直径

仅能求出直径长度，无法得知路径信息，可处理负权边。

```
int dp[N];
//dp[rt] 以rt为根的子树 从rt出发最远可达距离
/*
    对于每个结点x f[x]:经过节点x的最长链长度
*/
void DP(int rt)
{
    dp[rt]=0;//单点
    vis[rt]=1;
    for(int i=head[rt];i;i=nxt[i])
    {
        int s=ver[i];
        if(!vis[s])
        {
            DP(s);
            diameter=max(diameter,dp[rt]+dp[s]+edge[i]);
            dp[rt]=max(dp[rt],dp[s]+edge[i]);
        }
    }
}
```

1.3.2 两次 BFS/DFS 求树的直径

无法处理负权边，容易记录路径

```
void DFS(int start,bool record_path)
{
    vis[start]=1;
    for(int i=head[start];i;i=nxt[i])
    {
        int s=ver[i];
        if(!vis[s])
        {
            dis[s]=dis[start]+edge[i];
            if(record_path) path[s]=i;
            DFS(s,record_path);
        }
    }
    vis[start]=0;//清理
}
```

例题分析

P3629 [APIO2010] 巡逻（两种求树直径方法的综合应用）

P1099 树网的核（枚举）

1.4 最近公共祖先（LCA）

1.4.1 树上倍增

```
void BFS()
{
    queue<int> q;
    q.push(1);
    d[1] = 1;
    while (!q.empty())
    {
        int x = q.front();
        q.pop();
        for (int i = head[x]; i; i = nxt[i])
        {
            int y = ver[i];
            if (!d[y])
            {
                d[y] = d[x] + 1;
                fa[y][0] = x;
                for (int j = 1; j <= k; j++)
                {
                    fa[y][j] = fa[fa[y][j - 1]][j - 1];
                }
                q.push(y);
            }
        }
    }
}
int LCA(int x, int y)
{
    if (d[x] < d[y])
        swap(x, y);
    for (int i = k; i >= 0; i--)
        if (d[fa[x][i]] >= d[y])
            x = fa[x][i];
    if (x == y)
        return y;
    for (int i = k; i >= 0; i--)
        if (fa[x][i] != fa[y][i])
            x = fa[x][i], y = fa[y][i];
    return fa[x][0];
}
```

1.4.2 Tarjan 算法

```
int Find(int x)
{
    if (x == fa[x])
        return x;
    return fa[x] = Find(fa[x]);
}
```

```
void Tarjan(int x)
{
    vis[x] = 1;
    for (int i = head[x]; i; i = nxt[i])
    {
        int y = ver[i];
        if (!vis[y])
        {
            Tarjan(y);
            fa[y] = x;
        }
    }
    for (int i = 0; i < q[x].size(); i++)
    {
        int y = q[x][i].first, id = q[x][i].second;
        if (vis[y] == 2)
            lca[id] = Find(y);
    }
    vis[x] = 2;
}
```

1.5 树上差分与 LCA 的综合应用

1.6 负环与差分约束

1.6.1 负环

例题分析
POJ3621 Sightseeing Cows（0/1 分数规划，SPFA 判定负环）

1.6.2 差分约束系统

例题分析
POJ1201 Intervals（单源最长路）

1.7 Tarjan 算法与无向图连通性

1.7.1 无向图的割点与桥

1.7.1.1 割边判定法则

```
void Tarjan(int x, int in_edge)
{
    dfn[x] = low[x] = ++num;
    for (int i = head[x]; i; i = nxt[i])
    {
        int y = ver[i];
        if (!dfn[y])
        {
            Tarjan(y, i);
            low[x] = min(low[x], low[y]);
            if (low[y] > dfn[x])
            {
                bridge[i] = bridge[i ^ 1] = true;
            }
        }
        else if (i != (in_edge ^ 1))
            low[x] = min(low[x], dfn[y]);
    }
}
```

1.7.1.2 割点判定法则

```
void Tarjan(int x)
{
    dfn[x] = low[x] = ++num;
    int flag = 0;
    for (int i = head[x]; i; i = nxt[i])
    {
        int y = ver[i];
        if (!dfn[y])
        {
            Tarjan(y);
            low[x] = min(low[x], low[y]);
            if (low[y] >= dfn[x])
            {
                flag++;
                if (x != root || flag >= 2)
                    cut[x] = true;
            }
        }
        else
            low[x] = min(low[x], dfn[y]);
    }
}
```

例题分析
P3469 [POI2008]BLO-Blockade（割点，连通块计数）

1.7.2 无向图的双连通分量

1.7.2.1 边双连通分量 e-DCC 与其缩点

```
void DFS(int x)
{
    color[x] = dcc;
    for (int i = head[x]; i; i = nxt[i])
    {
        int y = ver[i];
        if (!color[y] && !bridge[i])
            DFS(y);
    }
}
void e_DCC()
{
    dcc = 0;
    for (int i = 1; i <= n; i++)
        if (!color[i])
            ++dcc, DFS(i);
    totc = 1;
    for (int i = 2; i <= tot; i++)
    {
        int u = ver[i ^ 1], v = ver[i];
        if (color[u] != color[v])
            add_c(color[u], color[v]);
    }
    origin_bridges = (totc - 1) / 2;
    k = log2(dcc) + 1;
}
```

1.7.2.2 点双连通分量 v-DCC 与其缩点

```
void Tarjan(int x)
{
    dfn[x] = low[x] = ++num;
    int flag = 0;
    stack[++top] = x;
    if (x == root && !head[x])
    {
        dcc[++cnt].push_back(x);
        return;
    }
    for (int i = head[x]; i; i = nxt[i])
    {
        int y = ver[i];
        if (!dfn[y])
        {
            Tarjan(y);
            low[x] = min(low[x], low[y]);
            if (low[y] >= dfn[x])
            {
                flag++;
```

```
                if (x != root || flag >= 2)
                    cut[x] = true;
                cnt++;
                int z;
                do
                {
                    z = stack[top--];
                    dcc[cnt].push_back(z);
                } while (z != y);
                dcc[cnt].push_back(x);
            }
        }
        else
            low[x] = min(low[x], dfn[y]);
    }
}
void v_DCC()
{
    cnt = 0;
    top = 0;
    for (int i = 1; i <= n; i++)
    {
        if (!dfn[i])
            root = i, Tarjan(i);
    }
    // 给每个割点一个新的编号(编号从cnt+1开始)
    num = cnt;
    for (int i = 1; i <= n; i++)
        if (cut[i]) new_id[i] = ++num;
    // 建新图，从每个v-DCC到它包含的所有割点连边
    tc = 1;
    for (int i = 1; i <= cnt; i++)
        for (int j = 0; j < dcc[i].size(); j++)
        {
            int x = dcc[i][j];
            if (cut[x]) {
                add_c(i, new_id[x]);
                add_c(new_id[x], i);
            }
            else c[x] = i; // 除割点外，其它点仅属于1个v-DCC
        }
}
```

例题分析

POJ3694 Network（e-DCC 缩点，LCA，并查集）

POJ2942 Knights of the Round Table（补图，v-DCC，染色法奇环判定）

1.7.3 欧拉路问题

欧拉图的判定

无向图连通，所有点度数为偶数。

欧拉路的存在性判定

无向图连通，恰有两个节点度数为奇数，其他节点度数均为偶数

```
// 模拟系统栈，答案栈
void Euler() {
    stack[++top] = 1;
    while (top > 0) {
        int x = stack[top], i = head[x];
        // 找到一条尚未访问的边
        while (i && vis[i]) i = Next[i];
        // 沿着这条边模拟递归过程，标记该边，并更新表头
        if (i) {
            stack[++top] = ver[i];
            head[x] = Next[i];
            vis[i] = vis[i ^ 1] = true;
        }
        // 与x相连的所有边均已访问，模拟回溯过程，并记录于答案栈中
        else {
            top--;
            ans[++t] = x;
        }
    }
}
```

例题分析

POJ2230 Watchcow（欧拉回路）

1.8 Tarjan 算法与有向图连通性

1.8.1 强连通分量（SCC）判定法则

```
void Tarjan(int x)
{
    dfn[x]=low[x]=++num;
    stack[++top]=x,in_stack[x]=true;
    for(int i=head[x];i;i=nxt[i])
    {
        int y=ver[i];
        if(!dfn[y])
        {
            Tarjan(y);
            low[x]=min(low[x],low[y]);
        }
        else if(in_stack[y])
            low[x]=min(low[x],dfn[y]);
    }
    if(dfn[x]==low[x])
    {
        cnt++;
        int y;
        do
        {
            y=stack[top--],in_stack[y]=false;
            color[y]=cnt, scc[cnt].push_back(y);
        } while (x!=y);
    }
}
```

1.8.2 SCC -> DAG

```
void SCC()
{
    for (int i = 0; i <= n; i++)
        if (!dfn[i])
            Tarjan(i);
    //缩点
    for (int x = 1; x <= n; x++)
    {
        for (int i = head[x]; i; i = nxt[i])
        {
            int y = ver1[i];
            if (color[x] != color[y])
                add_c(color[x], color[y]);
        }
    }
}
```

例题分析

POJ1236 Network of Schools（SCC->DAG，入度出度）

P3275 [SCOI2011] 糖果（SPFA TLE，SCC->DAG，Topo，DP）

1.8.3 有向图的必经点与必经边