



南京工業大學
NANJING TECH
UNIVERSITY

ICPC Template Manual



作者: 贺梦杰

August 6, 2019

Contents

| | | |
|----------|-----------------------|----------|
| 1 | 图论 | 2 |
| 1.1 | 最短路 | 3 |
| 1.1.1 | 单源最短路径 | 3 |
| 1.1.1.1 | Dijkstra | 3 |
| 1.1.1.2 | Bellman-Ford 和 SPFA | 3 |
| 1.1.2 | 任意两点间最短路径 | 4 |
| 1.1.2.1 | Floyd | 4 |
| 1.2 | 最小生成树 | 6 |
| 1.2.1 | Kruskal | 6 |
| 1.2.2 | Prim | 6 |
| 1.3 | 树的直径 | 10 |
| 1.3.1 | 树形 DP 求树的直径 | 10 |
| 1.3.2 | 两次 BFS/DFS 求树的直径 | 10 |
| 1.4 | 最近公共祖先 (LCA) | 11 |
| 1.4.1 | 树上倍增 | 11 |
| 1.4.2 | Tarjan 算法 | 11 |
| 1.5 | 树上差分与 LCA 的综合应用 | 12 |
| 1.6 | 负环与差分约束 | 13 |
| 1.6.1 | 负环 | 13 |
| 1.6.2 | 差分约束系统 | 13 |
| 1.7 | Tarjan 算法与无向图连通性 | 14 |
| 1.7.1 | 无向图的割点与桥 | 14 |
| 1.7.1.1 | 割边判定法则 | 14 |
| 1.7.1.2 | 割点判定法则 | 14 |
| 1.7.2 | 无向图的双连通分量 | 15 |
| 1.7.2.1 | 边双连通分量 e-DCC 与其缩点 | 15 |
| 1.7.2.2 | 点双连通分量 v-DCC 与其缩点 | 15 |
| 1.7.3 | 欧拉路问题 | 16 |
| 1.8 | Tarjan 算法与有向图连通性 | 18 |
| 1.8.1 | 强连通分量 (SCC) 判定法则 | 18 |
| 1.8.2 | SCC \rightarrow DAG | 18 |
| 1.8.3 | 有向图的必经点与必经边 | 19 |

Chapter 1

图论

1.1 最短路

1.1.1 单源最短路径

1.1.1.1 Dijkstra

```
1 void Dijkstra()
2 {
3     memset(dist, 0x3f, sizeof(dist));
4     memset(vis, 0, sizeof(vis));
5     priority_queue<pii, vector<pii>, greater<pii>> q;
6     dist[1] = 0;
7     q.push({dist[1], 1});
8     while (!q.empty())
9     {
10         int x = q.top().second;
11         q.pop();
12         if (!vis[x])
13         {
14             vis[x] = 1;
15             for (auto it : v[x])
16             {
17                 int y = it.first;
18                 if (dist[y] > dist[x] + it.second)
19                 {
20                     dist[y] = dist[x] + it.second;
21                     q.push({dist[y], y});
22                 }
23             }
24         }
25     }
26 }
```

1.1.1.2 Bellman-Ford 和 SPFA

```
1 void SPFA()
2 {
3     memset(dis, 0x3f, sizeof(dis));
4     memset(vis, 0, sizeof(vis));
5     queue<int> q;
6     dis[1] = 0;
7     vis[1] = 1;
8     q.push(1);
9     while (!q.empty())
10    {
11        int x = q.front();
12        q.pop();
13        vis[x] = 0;
14        for (int i = 0; i < v[x].size(); i++)
15        {
16            int y = v[x][i].first;
17            int z = v[x][i].second;
18            if (dis[y] > dis[x] + z)
```

```

19         {
20             dis[y] = dis[x] + z;
21             if (!vis[y])
22                 q.push(y), vis[y] = 1;
23         }
24     }
25 }
26 }

```

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

```

1 void get_path(int i, int j)
2 {
3     if (!path[i][j])
4         return;
5     get_path(i, path[i][j]);
6     p.push_back(path[i][j]);
7     get_path(path[i][j], j);
8 }
9 void Floyd()
10 {
11     memcpy(d, a, sizeof(d));
12     for (int k = 1; k <= n; k++)
13     {
14         for (int i = 1; i < k; i++)
15         {
16             for (int j = i + 1; j < k; j++)
17             {
18                 //注意溢出
19                 ll temp = d[i][j] + a[i][k] + a[k][j];
20                 if (ans > temp)
21                 {
22                     ans = temp;
23                     p.clear();
24                     p.push_back(i);
25                     get_path(i, j);
26                     p.push_back(j);
27                     p.push_back(k);
28                 }
29             }
30         }
31         for (int i = 1; i <= n; i++)
32         {
33             for (int j = 1; j <= n; j++)
34             {
35                 ll temp = d[i][k] + d[k][j];
36                 if (d[i][j] > temp)

```

```
37         {  
38             d[i][j] = temp;  
39             path[i][j] = k;  
40         }  
41     }  
42 }  
43 }  
44 }
```

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

```
1 void Init()
2 {
3     for (int i = 1; i <= n; i++)
4         fa[i] = i;
5 }
6 int Find(int x)
7 {
8     if (x == fa[x])
9         return x;
10    return fa[x] = Find(fa[x]);
11 }
12 void Kruskal()
13 {
14     Init();
15     sort(e.begin(), e.end());
16     int ans=0;
17     for (int i = 0; i < e.size(); i++)
18     {
19         int u = e[i].u, v = e[i].v;
20         int fu = Find(u), fv = Find(v);
21         if (fu != fv)
22         {
23             fa[fu] = fv;
24             ans += e[i].w;
25         }
26     }
27 }
```

1.2.2 Prim

```
1 void Prim()
2 {
3     memset(vis, 0, sizeof(vis));
4     memset(d, 0x3f, sizeof(d));
5     d[1] = 0;
6     int temp = n;
7     int ret = 0;
8     while (temp--)
9     {
10        int min_pos = 0;
11        for (int i = 1; i <= n; i++)
12            if (!vis[i] && (!min_pos || d[i] < d[min_pos]))
13                min_pos = i;
14        if (min_pos)
15        {
16            vis[min_pos] = 1;
17            ret += d[min_pos];
18        }
19    }
```

```

18         for (int i = 1; i <= n; i++)
19             if (!vis[i]) d[i] = min(d[i], weight[min_pos][i]);
20     }
21 }
22 }

```

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)

```

1  #include <algorithm>
2  #include <cstring>
3  #include <iostream>
4  #include <map>
5  #include <string>
6  #include <vector>
7  using namespace std;
8  #define inf 0x3f3f3f3f
9  #define N 25
10 #define M 500
11 map<string, int> name;
12 struct edge
13 {
14     int u, v, w;
15     bool operator<(const edge &e) const
16     {
17         return w < e.w;
18     }
19 };
20 int n, s, ptot = 0, a[N][N], ans, fa[N], d[N], ver[N];
21 vector<edge> e;
22 bool vis[N][N];
23 edge dp[N]; //dp[i] 1...i路径上的最大边
24 void Init()
25 {
26     for (int i = 1; i <= ptot; i++)
27         fa[i] = i;
28 }
29 int Find(int x)
30 {
31     if (x == fa[x])
32         return x;
33     return fa[x] = Find(fa[x]);
34 }
35 void Kruskal()
36 {
37     Init();
38     sort(e.begin(), e.end());
39     for (int i = 0; i < e.size(); i++)
40     {
41         int u = e[i].u, v = e[i].v;
42         if (u != 1 && v != 1)
43         {
44             int fu = Find(u), fv = Find(v);
45             if (fu != fv)

```



```

46         {
47             fa[fu] = fv;
48             vis[u][v] = vis[v][u] = 1;
49             ans += e[i].w;
50         }
51     }
52 }
53 }
54 void DFS(int cur, int pre)
55 {
56     for (int i = 2; i <= ptot; i++)
57     {
58         if (i != pre && vis[cur][i])
59         {
60             if (dp[i].w == -1)
61             {
62                 if (dp[cur].w < a[cur][i])
63                 {
64                     dp[i].u = cur;
65                     dp[i].v = i;
66                     dp[i].w = a[cur][i];
67                 }
68                 else
69                     dp[i] = dp[cur];
70             }
71             DFS(i, cur);
72         }
73     }
74 }
75 int main()
76 {
77     ios::sync_with_stdio(false);
78     cin.tie(0);
79     cin >> n;
80     string s1, s2;
81     int len;
82     name["Park"] = ++ptot;
83     memset(a, 0x3f, sizeof(a));
84     memset(d, 0x3f, sizeof(d));
85     //Park: 1
86     for (int i = 0; i < n; i++)
87     {
88         cin >> s1 >> s2 >> len;
89         if (!name[s1])
90             name[s1] = ++ptot;
91         if (!name[s2])
92             name[s2] = ++ptot;
93         int u = name[s1], v = name[s2];
94         a[u][v] = a[v][u] = min(a[u][v], len); //无向图邻接矩阵
95         e.push_back({u, v, len});
96     }
97     cin >> s; //度数限制
98     ans = 0;

```

```

99     Kruskal();
100    for (int i = 2; i <= ptot; i++)
101    {
102        if (a[1][i] != inf)
103        {
104            int rt = Find(i);
105            if (d[rt] > a[1][i])
106                d[rt] = a[1][i], ver[rt] = i;
107        }
108    }
109    for (int i = 2; i <= ptot; i++)
110    {
111        if (d[i] != inf)
112        {
113            s--;
114            ans += d[i];
115            vis[1][ver[i]] = vis[ver[i]][1] = 1;
116        }
117    }
118    while (s-- > 0)
119    {
120        memset(dp, -1, sizeof(dp));
121        dp[1].w = -inf;
122        for (int i = 2; i <= ptot; i++)
123        {
124            if (vis[1][i])
125                dp[i].w = -inf;
126        }
127        DFS(1, -1);
128        int w = -inf;
129        int v;
130        for (int i = 2; i <= ptot; i++)
131        {
132            if (w < dp[i].w - a[1][i])
133            {
134                w = dp[i].w - a[1][i];
135                v = i;
136            }
137        }
138        if (w <= 0)
139            break;
140        ans -= w;
141        vis[1][v] = vis[v][1] = 1;
142        vis[dp[v].u][dp[v].v] = vis[dp[v].v][dp[v].u] = 0;
143    }
144    cout << "Total miles driven: " << ans << endl;
145    system("pause");
146    return 0;
147 }

```

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。

```

1  int dp[N];
2  //dp[rt] 以rt为根的子树 从rt出发最远可达距离
3  /*
4   对于每个结点x f[x]:经过节点x的最长链长度
5  */
6  void DP(int rt)
7  {
8      dp[rt]=0;//单点
9      vis[rt]=1;
10     for(int i=head[rt];i;i=nxt[i])
11     {
12         int s=ver[i];
13         if(!vis[s])
14         {
15             DP(s);
16             diameter=max(diameter,dp[rt]+dp[s]+edge[i]);
17             dp[rt]=max(dp[rt],dp[s]+edge[i]);
18         }
19     }
20 }
```

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

```

1  void DFS(int start,bool record_path)
2  {
3      vis[start]=1;
4      for(int i=head[start];i;i=nxt[i])
5      {
6          int s=ver[i];
7          if(!vis[s])
8          {
9              dis[s]=dis[start]+edge[i];
10             if(record_path) path[s]=i;
11             DFS(s,record_path);
12         }
13     }
14     vis[start]=0;//清理
15 }
```

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

```

1 void BFS()
2 {
3     queue<int> q;
4     q.push(1);
5     d[1] = 1;
6     while (!q.empty())
7     {
8         int x = q.front();
9         q.pop();
10        for (int i = head[x]; i; i = nxt[i])
11        {
12            int y = ver[i];
13            if (!d[y])
14            {
15                d[y] = d[x] + 1;
16                fa[y][0] = x;
17                for (int j = 1; j <= k; j++)
18                {
19                    fa[y][j] = fa[fa[y][j - 1]][j - 1];
20                }
21                q.push(y);
22            }
23        }
24    }
25 }
26 int LCA(int x, int y)
27 {
28     if (d[x] < d[y])
29         swap(x, y);
30     for (int i = k; i >= 0; i--)
31         if (d[fa[x][i]] >= d[y])
32             x = fa[x][i];
33     if (x == y)
34         return y;
35     for (int i = k; i >= 0; i--)
36         if (fa[x][i] != fa[y][i])
37             x = fa[x][i], y = fa[y][i];
38     return fa[x][0];
39 }

```

1.4.2 Tarjan 算法

```

1 int Find(int x)
2 {
3     if (x == fa[x])
4         return x;
5     return fa[x] = Find(fa[x]);
6 }

```

```
7 void Tarjan(int x)
8 {
9     vis[x] = 1;
10    for (int i = head[x]; i; i = nxt[i])
11    {
12        int y = ver[i];
13        if (!vis[y])
14        {
15            Tarjan(y);
16            fa[y] = x;
17        }
18    }
19    for (int i = 0; i < q[x].size(); i++)
20    {
21        int y = q[x][i].first, id = q[x][i].second;
22        if (vis[y] == 2)
23            lca[id] = Find(y);
24    }
25    vis[x] = 2;
26 }
```

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

```

1 void Tarjan(int x, int in_edge)
2 {
3     dfn[x] = low[x] = ++num;
4     for (int i = head[x]; i; i = nxt[i])
5     {
6         int y = ver[i];
7         if (!dfn[y])
8         {
9             Tarjan(y, i);
10            low[x] = min(low[x], low[y]);
11            if (low[y] > dfn[x])
12            {
13                bridge[i] = bridge[i ^ 1] = true;
14            }
15        }
16        else if (i != (in_edge ^ 1))
17            low[x] = min(low[x], dfn[y]);
18    }
19 }

```

1.7.1.2 割点判定法则

```

1 void Tarjan(int x)
2 {
3     dfn[x] = low[x] = ++num;
4     int flag = 0;
5     for (int i = head[x]; i; i = nxt[i])
6     {
7         int y = ver[i];
8         if (!dfn[y])
9         {
10            Tarjan(y);
11            low[x] = min(low[x], low[y]);
12            if (low[y] >= dfn[x])
13            {
14                flag++;
15                if (x != root || flag >= 2)
16                    cut[x] = true;
17            }
18        }
19        else
20            low[x] = min(low[x], dfn[y]);
21    }
22 }

```

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

```

1 void DFS(int x)
2 {
3     color[x] = dcc;
4     for (int i = head[x]; i; i = nxt[i])
5     {
6         int y = ver[i];
7         if (!color[y] && !bridge[i])
8             DFS(y);
9     }
10 }
11 void e_DCC()
12 {
13     dcc = 0;
14     for (int i = 1; i <= n; i++)
15         if (!color[i])
16             ++dcc, DFS(i);
17     totc = 1;
18     for (int i = 2; i <= tot; i++)
19     {
20         int u = ver[i ^ 1], v = ver[i];
21         if (color[u] != color[v])
22             add_c(color[u], color[v]);
23     }
24     origin_bridges = (totc - 1) / 2;
25     k = log2(dcc) + 1;
26 }

```

1.7.2.2 点双连通分量 v-DCC 与其缩点

```

1 void Tarjan(int x)
2 {
3     dfn[x] = low[x] = ++num;
4     int flag = 0;
5     stack[++top] = x;
6     if (x == root && !head[x])
7     {
8         dcc[++cnt].push_back(x);
9         return;
10    }
11    for (int i = head[x]; i; i = nxt[i])
12    {
13        int y = ver[i];
14        if (!dfn[y])
15        {
16            Tarjan(y);
17            low[x] = min(low[x], low[y]);
18            if (low[y] >= dfn[x])
19            {
20                flag++;

```



```

21         if (x != root || flag >= 2)
22             cut[x] = true;
23         cnt++;
24         int z;
25         do
26         {
27             z = stack[top--];
28             dcc[cnt].push_back(z);
29         } while (z != y);
30         dcc[cnt].push_back(x);
31     }
32 }
33 else
34     low[x] = min(low[x], dfn[y]);
35 }
36 }
37 void v_DCC()
38 {
39     cnt = 0;
40     top = 0;
41     for (int i = 1; i <= n; i++)
42     {
43         if (!dfn[i])
44             root = i, Tarjan(i);
45     }
46     // 给每个割点一个新的编号(编号从cnt+1开始)
47     num = cnt;
48     for (int i = 1; i <= n; i++)
49         if (cut[i]) new_id[i] = ++num;
50     // 建新图, 从每个v-DCC到它包含的所有割点连边
51     tc = 1;
52     for (int i = 1; i <= cnt; i++)
53         for (int j = 0; j < dcc[i].size(); j++)
54         {
55             int x = dcc[i][j];
56             if (cut[x]) {
57                 add_c(i, new_id[x]);
58                 add_c(new_id[x], i);
59             }
60             else c[x] = i; // 除割点外, 其它点仅属于1个v-DCC
61         }
62 }

```

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

```
1 // 模拟系统栈，答案栈
2 void Euler() {
3     stack[++top] = 1;
4     while (top > 0) {
5         int x = stack[top], i = head[x];
6         // 找到一条尚未访问的边
7         while (i && vis[i]) i = Next[i];
8         // 沿着这条边模拟递归过程，标记该边，并更新表头
9         if (i) {
10             stack[++top] = ver[i];
11             head[x] = Next[i];
12             vis[i] = vis[i ^ 1] = true;
13         }
14         // 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
15         else {
16             top--;
17             ans[++t] = x;
18         }
19     }
20 }
```

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

```

1 void Tarjan(int x)
2 {
3     dfn[x]=low[x]=++num;
4     stack[++top]=x,in_stack[x]=true;
5     for(int i=head[x];i;i=nxt[i])
6     {
7         int y=ver[i];
8         if(!dfn[y])
9         {
10             Tarjan(y);
11             low[x]=min(low[x],low[y]);
12         }
13         else if(in_stack[y])
14             low[x]=min(low[x],dfn[y]);
15     }
16     if(dfn[x]==low[x])
17     {
18         cnt++;
19         int y;
20         do
21         {
22             y=stack[top--],in_stack[y]=false;
23             color[y]=cnt, scc[cnt].push_back(y);
24         } while (x!=y);
25     }
26 }

```

1.8.2 SCC -> DAG

```

1 void SCC()
2 {
3     for (int i = 0; i <= n; i++)
4         if (!dfn[i])
5             Tarjan(i);
6     //缩点
7     for (int x = 1; x <= n; x++)
8     {
9         for (int i = head[x]; i; i = nxt[i])
10        {
11            int y = ver1[i];
12            if (color[x] != color[y])
13                add_c(color[x], color[y]);
14        }
15    }
16 }

```

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