
ACM ICPC Template



GuessEver

Last Modify at January 29, 2015

Contents

1	Graph	3
1.1	Shortest path	3
1.1.1	Dijkstra	3
1.1.2	Spfa	3
1.1.3	Floyd	3
1.2	Minimum Spanning Tree	4
1.2.1	Prime	4
1.2.2	Kruskal	5
1.3	Tarjan - Strong Union	5
1.4	LCA	6
1.4.1	Tarjan	6
1.4.2	Doubling Algorithm	8
1.5	Bipartite Graph	8
1.5.1	Maximal Matching - The Hungarian algorithm	8
1.5.2	Optimal Matching - KM	8
1.6	Network flow	8
1.6.1	Maximum Flow - isap	8
1.6.2	Minimum Cost Maximum Flow - spfa	9
2	Tools	10
2.1	C char*	10
2.2	C++ std::string	10
2.3	Batch test	11
2.3.1	@Linux	11
2.3.2	@Windows	11

1 Graph

1.1 Shortest path

1.1.1 Dijkstra

```

1 void dijkstra()
2 {
3     memset(dist, 0, sizeof(dist));
4     while(!Q.empty())
5     {
6         int x = Q.top().second; Q.pop();
7         if(done[x]) continue;
8         done[x] = 1;
9         for(Link p = head[x]; p; p = p->next)
10            if(dist[p->y] > dist[x] + p->z)
11            {
12                dist[p->y] = dist[x] + p->z;
13                Q.push(make_pair(dist[p->y], p->y));
14            }
15     }
16 }
```

1.1.2 Spfa

```

1 void spfa()
2 {
3     memset(dist, 0x3f, sizeof(dist));
4     Q.push(S); // S为源点
5     while(!Q.empty())
6     {
7         int x = Q.front();
8         Q.pop(); inQ[x] = 0;
9         for(Link p = head[x]; p; p = p->next)
10            if(dist[p->y] > dist[x] + p->z)
11            {
12                dist[p->y] = dist[x] + p->z;
13                if(!inQ[p->y])
14                {
15                    Q.push(p->y);
16                    inQ[p->y] = 1;
17                }
18            }
19     }
20 }
```

1.1.3 Floyd

```

1 void floyd()
2 {
3     for(int k = 1; k <= n; k++) // 这里可以看作是一个加边的过程
```

```

4         for(int i = 1; i <= n; i++)
5             for(int j = 1; j <= n; j++)
6                 map[i][j] = min(map[i][j], map[i][k] + map[k][j]);
7     }
8
9     // 最小环
10    void MinCircle()
11    {
12        cap[] = map[];
13        int circle = 0x3f3f3f3f;
14        for(int k = 1; k <= n; k++)
15        {
16            for(int i = 1; i < k; i++)
17                for(int j = i+1; j < k; j++)
18                    circle = min(circle, map[i][j] + cap[j][k] + cap[k][i])
19                    ;
20            for(int i = 1; i <= n; i++)
21                for(int j = 1; j <= n; j++)
22                    map[i][j] = min(map[i][j], map[i][k] + map[k][j]);
23        }
24        return circle == 0x3f3f3f3f ? -1 : circle;
25    }
26    // floyd判圈法 (大白书 p44)
27    void Circle()
28    {
29        int ans = k;
30        int k1 = k, k2 = k;
31        do{
32            k1 = next(k1);
33            k2 = next(k2); ans = max(ans, k2);
34            k2 = next(k2); ans = max(ans, k2);
35        }while(k1 != k2);
36        return ans;
37    }

```

1.2 Minimum Spanning Tree

1.2.1 Prime

```

1    void prime()
2    {
3        memset(dist, 0, sizeof(dist));
4        int res = 0;
5        while(!Q.empty())
6        {
7            int x = Q.top().second;
8            if(done[x]) {Q.pop(); continue;}
9            res += Q.top().first;
10           Q.pop();
11           for(Link p = head[x]; p; p = p->next)
12               if(dist[p->y] > p->z)

```

```

13         {
14             dist[p->y] = p->z;
15             Q.push(make_pair(dist[p->y], p->y));
16         }
17     }
18 }

```

1.2.2 Kruskal

```

1 void prime()
2 {
3     sort(edge, edge+Cnt, cmp);
4     int res = 0;
5     for(int i = 0; i < Cnt; i++)
6     {
7         if(getroot(edge[i].x) == getroot(edge[i].y)) continue;
8         merge(edge[i].x, edge[i].y);
9         res += edge[i].z;
10    }
11 }

```

1.3 Tarjan - Strong Union

```

1 void dfs(int x)
2 {
3     now[x] = low[x] = ++dfstime;
4     hash[x] = 1;
5     st.push(x); inst[x] = 1;
6     for(int i = 1; i <= n; i++)
7         if(map[x][i])
8         {
9             if(!hash[i])
10            {
11                dfs(i);
12                low[x] = min(low[x], low[i]);
13            }
14            else if(inst[i]) low[x] = min(low[x], now[i]);
15        }
16     if(low[x] == now[x])
17     {
18         while(!st.empty())
19         {
20             int u = st.top();
21             st.pop(); inst[u] = 0;
22             belong[u] = number;
23             if(u == x) break;
24         }
25         number++;
26     }
27 }
28 void tarjan()

```

```

29 {
30     for(int i = 1; i <= n; i++)
31         if(!hash[i]) dfs(i);
32     if(!st.empty()) // 这是一个未知 bug    栈中还会剩下一个强连通分量
33     {
34         while!st.empty()
35         {
36             int u = st.top();
37             st.pop();
38             belong[u] = number;
39         }
40         number++;
41     }
42 }

```

1.4 LCA

1.4.1 Tarjan

```

1 // poj 1330 (changed something)
2 // LCA tarjan
3 #include <cstdio>
4 #include <cstring>
5
6 const int N = 10000 + 10;
7
8 int n;
9 struct Link{int y, idx; Link *next;}*head[N], *ask[N];
10 int tx, ty;
11 bool in[N], vis[N];
12 int f[N];
13 int ans[N]; // Query Answer
14
15 void inLink(int x, int y)
16 {
17     Link *p = new Link;
18     p -> y = y;
19     p -> next = head[x];
20     head[x] = p;
21 }
22 void inAsk(int x, int y, int idx)
23 {
24     Link *p = new Link;
25     p -> y = y;
26     p -> idx = idx;
27     p -> next = ask[x];
28     ask[x] = p;
29 }
30
31 int getroot(int x)
32 {
33     return f[x] == x ? x : f[x] = getroot(f[x]);

```

```

34 }
35
36 void LCA(int x)
37 {
38     vis[x] = 1;
39     f[x] = x;
40     for(Link *p = ask[x]; p; p = p -> next)
41         if(vis[p->y]) ans[p->idx] = getroot(p->y);
42     for(Link *p = head[x]; p; p = p -> next)
43         if(!vis[p->y])
44         {
45             LCA(p->y);
46             f[p->y] = x;
47         }
48 }
49
50 int main()
51 {
52     int T; scanf("%d", &T);
53     while(T--)
54     {
55         memset(head, 0, sizeof(head));
56         memset(ask, 0, sizeof(ask));
57         memset(in, 0, sizeof(in));
58         memset(vis, 0, sizeof(vis));
59         scanf("%d", &n);
60         for(int i = 1; i <= n; i++) f[i] = i;
61         for(int i = 1; i < n; i++)
62         {
63             int x, y;
64             scanf("%d%d", &x, &y);
65             inLink(x, y);
66             in[y] = 1;
67         }
68         int q = 1; // the number of query
69         for(int i = 1; i <= q; i++)
70         {
71             int x, y; scanf("%d%d", &x, &y);
72             inAsk(x, y, i); inAsk(y, x, i);
73         }
74         int root = -1;
75         for(int i = 1; i <= n; i++)
76             if(!in[i]) {root = i; break;}
77         LCA(root);
78         for(int i = 1; i <= q; i++)
79             printf("%d\n", ans[i]);
80     }
81     return 0;
82 }

```

1.4.2 Doubling Algorithm

还不会...

1.5 Bipartite Graph

1.5.1 Maximal Matching - The Hungarian algorithm

```

1  int ttt = 0; // 全局时间戳变量
2
3  bool search(int x)
4  {
5      for(int i = 1; i <= m; i++)
6          if(map[x][i] && vis[i] != ttt)
7              {
8                  vis[i] = ttt;
9                  if(pre[i] == -1 || search(pre[i]))
10                     {
11                         pre[i] = x;
12                         return 1;
13                     }
14             }
15     return 0;
16 }
17
18 int match()
19 {
20     int res = 0;
21     for(int i = 1; i <= n; i++)
22     {
23         ++ttt; // 这里不用 memset 节省时间
24         res += search(i);
25     }
26     return res;
27 }
```

1.5.2 Optimal Matching - KM

不会... 用费用流解决

1.6 Network flow

1.6.1 Maximum Flow - isap

```

1  // h[x] : 点 x 在第 h[x] 层
2  // v[k] : 第 k 层有 v[k] 个点
3  int sap(int x, int flow)
4  {
5      if(x == n) return flow;
6      int res = 0;
7      for(int i = S; i <= T; i++)
8          if(g[x][i] && h[x] == h[i] + 1)
```



```

9      {
10         int t = sap(i, min(g[x][i], flow - res));
11         res += t; g[x][i] -= t; g[i][x] += t;
12         if(res == flow) return res;
13         if(h[S] >= T) return res;
14     }
15     //if(h[S] >= T) return res;
16     if((--v[h[x]]) == 0) h[S] = T;
17     ++v[++h[x]];
18     return res;
19 }
20 int main()
21 {
22     v[0] = T;
23     int maxflow = 0;
24     while(h[S] < T) maxflow += sap(1, inf);
25     return 0;
26 }

```

1.6.2 Minimum Cost Maximum Flow - spfa

```

1 struct EG{int from,to,flow,cost,next;}edge[M];
2
3 void add_edge(int a,int b,int c,int d)
4 {
5     edge[L]=(EG){a,b,c,+d,head[a]};
6     head[a]=L++;
7     edge[L]=(EG){b,a,0,-d,head[b]};
8     head[b]=L++;
9 }
10
11 bool spfa()
12 {
13     memset(inQ, 0, sizeof(inQ));
14     memset(dist, 0x3f, sizeof(dist));
15     dist[S] = 0;
16     q.push(S);
17     while(!q.empty())
18     {
19         int x = q.front();
20         q.pop();
21         inQ[x] = 0;
22         for(int i = head[x]; i != -1; i = edge[i].next)
23             if(edge[i].flow && dist[edge[i].to] > dist[x] + edge[i].
                cost)
24             {
25                 pre[edge[i].to] = i;
26                 dist[edge[i].to] = dist[x] + edge[i].cost;
27                 if(!inQ[edge[i].to])
28                 {
29                     inQ[edge[i].to] = 1;
30                     q.push(edge[i].to);

```

```

31         }
32     }
33 }
34 return dist[T] != inf;
35 }
36 void MFMC()
37 {
38     memset(head, -1, sizeof(head));
39     建图调用 add_edge();
40
41     int mincost = 0, maxflow = 0;
42     while(spfa())
43     {
44         int res = inf;
45         for(int i = T; i != S; i = edge[pre[i]].from)
46         {
47             res = min(res, edge[pre[i]].flow);
48         }
49         for(int i = T; i != S; i = edge[pre[i]].from)
50         {
51             edge[pre[i]].flow -= res;
52             edge[pre[i] ^ 1].flow += res;
53         }
54         maxflow += res;
55         mincost += res * dist[T];
56     }
57 }

```

2 Tools

2.1 C char*

```

1 头文件 cstring
2 strlen(s); // 获取长度  $O(N)$ 
3 strcpy(a+2, b+1) // 从  $b+1$  开始全部赋值给  $a+2$  开始的字符串
4 strncpy(a+2, b+1, 2) // 从  $b+1$  开始赋值 2 个给  $a+2$  开始的字符串
5 strcmp(a, b) // 比较  $a$  和  $b$  的大小, 相等返回 0,  $a > b$  返回正整数
6 strcat(a, b) // 相当于 string 类的  $a += b$ ;
7 strstr(a, b) - a; // 返回  $b$  在  $a$  中第一次出现的位置, 不存在返回 NULL (即 0), 由于  $-a$ 
    , 所以最后应该是  $-a$ 

```

2.2 C++ std::string

```

1 //==== 初始化 ====
2 头文件 string 并加上 std::
3 string s(str); // 相当于 string s = str;
4 string s(cstr); // 把 char 数组类型的字符串 cstr 作为  $s$  的初值
5 s.clear(); // 清空, 相当于  $s = ""$ ;
6
7 //==== 长度 ====

```

```

8 s.length();//获取s的长度, 0(1)
9 s.size();//一样
10
11 //====插入删除====
12 s.insert(2, "a"); //在s的位置2插入string类字符串"a"
13 s.erase(2, 3); //从s的位置2开始删除3个字符
14
15 //====查找====
16 s.find("abc");//查找字符串"abc"在s中第一次出现的位置 ( 据说是KMP实现的 )
17 //s="aabcc"; printf("%d %d\n", (int)s.find("abc"), (int)s.find("aabb"));
18 //上一行程序应输出 1 -1 ( 若没找到必须强行转换为int才为 -1 )

```

2.3 Batch test

2.3.1 @Linux

```

1 mkdata=mk
2 filea=a
3 fileb=b
4
5 g++ $mkdata.cpp -o $mkdata
6 g++ $filea.cpp -o $filea
7 g++ $fileb.cpp -o $fileb
8 cas=0
9 while true; do
10     ./mkdata > $filea.in
11     ./filea < $filea.in > $filea.out
12     ./fileb < $filea.in > $fileb.out
13     if ! diff $filea.out $fileb.out
14     then
15         echo "Wrong Answer"
16         break
17     fi
18     echo $((cas=cas+1)) "Accepted"
19 done

```

2.3.2 @Windows

```

1 :loop
2     mk > A.in
3     A < A.in > A.out
4     p < A.in > p.out
5     fc A.out p.out
6     if errorlevel 1 goto end
7     goto loop
8 :end
9     pause

```