Contents

1 Articulation points

1.1 Articulation vertices

Find articulation vertices in a **connected graph** using Tarjan's algorithm.

```
const int NN = 101;

typedef vector< vector <int> > graph;

int low[NN], disc[NN], parent[NN], dtime = 0;
bool visit[NN], ap[NN];

void dfs(const graph &g, int x){
    low[x] = disc[x] = ++dtime;
    visit[x] = true;
    int children = 0;
    for(int i=0;i < g[x].size(); i++){
        const int &y = g[x][i];
        if(!visit[y]){
            parent[y] = x;
            children++;
            dfs(g,y);
            low[x] = min(low[x], low[y]);
            if(parent[x] < 0 && children > 1)
                ap[x] = true;
            if(parent[x] >= 0 && low[y] >= disc[x])
                ap[x] = true;
        }
        else if(y != parent[x]){
            low[x] = min(low[x], disc[y]);
        }
    }

}
```

1.1.1 Field test: Networks

Source: *UVA 315*
A connected network is given. Find the number of articulation vertices.

```
#include<iostream>
#include<sstream>
```

```
3  #include<string>
4  #include<vector>
5  #include<cstring>
6
7  #define min(a,b) (a) < (b)? (a) : (b)
8
9  using namespace std;
10
11 typedef vector< vector <int> > graph;
12
13 const int NN = 101;
14
15 int low[NN], disc[NN], parent[NN], dtime = 0;
16 bool visit[NN], ap[NN];
17 void dfs(const graph &g, int x){
18     low[x] = disc[x] = ++dtime;
19     visit[x] = true;
20     int children = 0;
21     for(int i=0;i < g[x].size(); i++){
22         const int &y = g[x][i];
23         if(!visit[y]){
24             parent[y] = x;
25             children++;
26             dfs(g,y);
27             low[x] = min(low[x], low[y]);
28             if(parent[x] < 0 && children > 1)
29                 ap[x] = true;
30             if(parent[x] >= 0 && low[y] >= disc[x])
31                 ap[x] = true;
32         }
33         else if(y != parent[x]){
34             low[x] = min(low[x], disc[y]);
35         }
36     }
37
38 }
39
40 int main(){
41     freopen("315.in", "r", stdin);
42     string line;
43     for(int n; cin >> n && n;){
44         memset(low, -1, sizeof(int)*n);
45         memset(disc, -1, sizeof(int)*n);
```

```
46          memset(visit, false, sizeof(bool)*n);
47          memset(ap, false, sizeof(bool)*n);
48          memset(parent, -1, sizeof(int)*n);
49          dtime = 0;
50          graph g(n , vector<int>());
51          while(getline(cin,line)){
52              if(line[0] == '0')
53                  break;
54              stringstream ss(line);
55              int v,x;
56              ss >> v;
57              while(ss >> x)
58                  g[v-1].push_back(x-1),g[x-1].push_back(v-1);
59          }
60          dfs(g,0);
61          int ans = 0;
62          for(int i=0;i<n;i++)
63              if(ap[i])
64                  ++ans;
65          cout << ans << endl;
66      }
67      return 0;
68  }
```

2 Classic algorithms

2.1 Prim's minimum spanning tree

```
1   /*
2    * Created on: Oct 30, 2011
3    * Author: sbaldrich
4    * Problem: Cable Network
5    * Source: South America Regional Contest 2000
6    */
7
8   #include<iostream>
9   #include<vector>
10  #include<limits>
11
12  #define min(a,b)    (a)<(b)?(a):(b)
13
14  using namespace std;
```

```
15
16  long long graph[25][25], t;
17  int n;
18
19  const long long oo = numeric_limits<long long>::max();
20
21  vector<int> prim(int start, int block) {
22      vector<int> parent(n + 1, -1);
23      vector<long long> distance(n + 1, oo);
24      vector<bool> intree(n + 1, false);
25
26      distance[start] = 0;
27      int v = start;
28      long long dist;
29      for (int i = 0; i <= n; i++) {
30          if (block & (1 << i))
31              intree[i] = true;
32      }
33
34      while (!intree[v]) {
35          intree[v] = true;
36          for (int i = 0; i <= n; i++) {
37              if ((distance[i] > graph[v][i]) && !intree[i])
38                  distance[i] = graph[v][i], parent[i] = v;
39          }
40
41          v = 1;
42          dist = oo;
43          for (int i = 0; i <= n; i++) {
44              if (!intree[i] && dist > distance[i])
45                  dist = distance[i], v = i;
46          }
47      }
48
49      return parent;
50  }
51
52  long long price(vector<int> v) {
53      long long l = 0LL, b = 0LL;
54      for (int i = 0; i <= n; i++)
55          if (v[i] != -1)
56              l += graph[i][v[i]];
57          else
```

```
            if(i>0)b++;
    return l + (t * b);
}

int main() {
    int c = 1;
    while (cin >> n && n) {
        cin >> t;
        for (int i = 0; i <= n; i++)
            graph[i][i] = oo;
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j <= n; j++)
                cin >> graph[i][j], graph[j][i] = graph[i][j];
        }
        int lim = 1 << (n+1);

        long long best = oo;
        for (int block = 0; block <= lim; block++) {
            long long x = price(prim(0, block));
            best = min(x,best);
        }
        cout << "Cable Net #" << c++ <<endl;
        cout << n*t<<endl<<best<<endl<<endl;


    }

    return 0;
}
```

2.2 Lowest common ancestor

```
#include<iostream>
#include<cstring>

using namespace std;

#define abs(a) (a) < 0? -1*(a) : (a)
const int NN = 131072;
int p[NN];
int parent[NN];
int cycle_size[NN];
int cycle_label[NN]; //Label of node in its cycle (If it belongs to any)
```

```cpp
bool visit[NN];
int L[NN]; //Level of node in a dfs for LCA
int P[NN][17]; //Sparse table for LCA

int root(int x){ return p[x] == x? x : p[x] = root(p[x]);}
int union_find(int a, int b){
    p[root(a)] = b = root(b);
    return a != b;
}

int level(int node){
    if(cycle_size[node])
        return L[node] = 0;
    if(L[node] != -1)
        return L[node];
    return L[node] = level(parent[node]) +1;
}

int lca(int p, int q){
    if(L[p] < L[q])
        p ^= q ^= p ^= q;
    int lg = 1;
    for(;(1 << lg) <= L[p];lg++);
    --lg;
    for(int i=lg;i>=0;i--)
        if(L[p] - (1 << i) >= L[q])
            p = P[p][i];
    if(p == q) // We were querying the same branch
        return p;
    for(;lg>=0;lg--)
        if(P[p][lg] != -1 && P[q][lg] != P[p][lg])
            p = P[p][lg], q = P[q][lg];
    return P[p][0];
}

int cycle_parent(int node){
    while(!cycle_size[node])
        node = parent[node];
    return node;
}

int main(){
    //freopen("j.in" , "r" , stdin);
```

```
55      //freopen("j.out", "w", stdout);
56      for(int n,q,caseno=0;cin>>n;caseno++){
57          int d,a,b;
58          for(int i=0;i<n;i++)
59              p[i] = i;
60          memset(parent, -1, sizeof(int) * n);
61          memset(cycle_size, 0, sizeof(int) * n);
62          memset(cycle_label, -1, sizeof(int) * n);
63          memset(visit, false, sizeof(bool) * n);
64          memset(L, -1, sizeof(int) * n);
65          memset(P, -1, sizeof(int) * n);
66          for(int i=0;i<n;i++){
67              cin>>d,--d;
68              if(!union_find(i,d)){
69                  int sz = 1,t,l=1;
70
71                  t = parent[d];
72                  do{
73                      ++sz;
74                      t = parent[t];
75
76                  }while(t!=-1);
77
78                  t = d;
79
80                  do{
81                      cycle_label[t] = l++;
82                      cycle_size[t] = sz;
83                      t = parent[t];
84                  }while(t!=-1);
85
86              }else{
87                  parent[i] = d;
88              }
89          }
90          for(int i=0;i<n;i++)
91              level(i);
92
93          for(int i=0; i<n; i++){
94              for(int j=0; 1 << j <n; j++)
95                  P[i][j] = -1;
96              if(!cycle_size[i])
97                  P[i][0] = parent[i];
```

```
98          }
99
100         for(int j=1; (1 << j) < n;j++)
101             for(int i=0;i<n;i++)
102                 if(P[i][j-1] != -1)
103                     P[i][j] = P[P[i][j-1]][j-1];
104
105         cin>>q;
106         for(int i=0;i<q;i++){
107             cin>>a>>b;
108             --a,--b;
109             if(root(a) != root(b))
110                 cout << -1 << endl;
111             else{
112                 int ca = lca(a,b);
113                 if(ca == -1){
114                     int pa = cycle_parent(a), pb = cycle_parent(b);
115                     d = abs(cycle_label[pa] - cycle_label[pb]);
116                     cout << L[a] + L[b] + min(d, cycle_size[pa] - d);
117                 }
118                 else{
119                     cout << L[a] + L[b] - 2*L[ca];
120                 }
121                 cout << endl;
122             }
123         }
124     }
125     return 0;
126 }
```