CEOI'2012 Day2, Task: network

Consider the directed graph $G = (V, E)$ where V is the set of the nodes of the network and E is the direct communication channels of the network. The following three properties hold for the graph G.

1. Each node is reachable from the central node r.
2. Every edge of the graph belongs to at most one simple cycle.
3. Each edge in any depth-first search (DFS) is either tree-edge or back-edge.

We are going to compute for each node x of a DFS tree T the following data:

- $TRn[x]$: the number of nodes in T that are reachable from node x in T.
- B[x] : the highest node in T which is reachable from x by tree edges and at most one back-edge.
- $H[x]$: the highest node in T which is reachable from x in G.

It is clear that the answer for the first subtask is $TRn[H[x]]$ for each node x. We compute $TRn[x]$ and $B[x]$ by the depth-first search DFS1. $H[x]$ is computed by the second depth-first search DFS2.
In order to solve the second subtask observe that the requirement of the task is equivalent to the following: add minimum number of new edges to the graph such that every edge will be contained in exactly one cycle.
Consider the graph $\overline{G}$ which is obtained from G by removing each edge that belongs to a cycle. It is clear that $\overline{G}$ consists of one or more trees. Let K be the number of the leaves of the trees of $\overline{G}$. It is obvious that the minimum number of the necessary new edges is at least K. On the other hand, consider a (directed, rooted) tree that has L leaves. Adding a new edge (u, r) where r is the root, and u is a leaf, and then for all other leaves of this tree v add the new edge (v, w), where w is the least ancestor of v that is already contained in a cycle. By this procedure we obtain a graph that satisfies the required property of the task. Therefore K is sufficient.
If the number of nodes of input graph G is n then G has at most $2 \cdot (n-1)$ edges, therefore the running time of the algorithm is $\mathbf{O}(n)$.

Implementation

```
#include <stdio.h>
#include <stdlib.h>
#define maxN 100001

using namespace std;

typedef struct LList{
    int to;
    LList *next;
}LList;
typedef enum Paletta {White, Gray, Black, Blue} Paletta;
    LList* G[maxN];
    int m,n,r;
    int inDeg[maxN];        //node in-degree
    int outDeg[maxN];       //node out-degree
    Paletta Color[maxN];    //DFS coloring
    int Parent[maxN];       //DFS spanning tree representation
    int B[maxN];            //B[p] is the highest node in the DFS tree
                            //that is reachable from p by 1 or 0 back edge
    int H[maxN];            //H[p] is the highest node in the DFS tree
                            //that is reachable from p
    int TRn[maxN];          //# reachable nodes in the DFS tree
    int Sol[maxN];          //solution edges: (p,Sol[p]), Sol[p]!=0
```

```
void ReadIn(){
    int p,q;
    LList* pq;
    scanf("%d %d %d", &n,&m,&r);
    for (p=1; p<=n;p++){
        Color[p]=White; G[p]=NULL;
        inDeg[p]=0; outDeg[p]=0;
        Sol[p]=0;
   }
   for (int i=1; i<=m; i++){
      scanf("%d %d", &p,&q);
      pq = new LList;
      pq->to=q; pq->next=G[p];
     G[p]=pq;
      outDeg[p]++; inDeg[q]++;
   }
}
void DFS1(int p){
    Color[p]=Gray;
    LList* pq=G[p];
    int q, rep=0;
    B[p]=p;
    while (pq!=NULL){
        q=pq->to;
        if (Color[q]==White ){
            Parent[q]=p;
            DFS1(q);
            if (B[q]!=q && B[q]!=p) B[p]=B[q];
            rep+=TRn[q];
        }else if (Color[q]==Gray){ //p->q back edge
            B[p]=q;
            int x=p;
            while (x!=q){       //"removing" edges that are in the
                inDeg[x]--;     //cycle p~>q->p
                outDeg[x]--;
                x=Parent[x];
            }
            inDeg[x]--; outDeg[x]--;
        }
        pq=pq->next;
    }
    TRn[p]=1+rep;
}//DFS1
void DFS2(int p){
    Color[p]=Black;
    LList* pq=G[p];
    int q;
    if (B[p]==p)
        H[p]=p;
    else
        H[p]=H[B[p]];
    while (pq!=NULL){
        q=pq->to;
        if (Color[q]==Gray)
            DFS2(q);
        pq=pq->next;
    }
}//DFS2
```

```
int main() {
    ReadIn();
    DFS1(r);
    DFS2(r);

    for (int p=1;p<n;p++)
        printf("%d ",TRn[H[p]]);
    printf("%d\n",TRn[H[n]]);

    int Soln=0;
    for (int p=1;p<=n;p++) if (inDeg[p]==1 && outDeg[p]==0){
        int x=p;
        do{
            Color[x]=Blue;
            x=Parent[x];
        }while(Color[x]==Black && inDeg[x]!=0);
        //x is either root of a tree or contained in a cycle
        Color[x]=Blue;
        Sol[p]=x;
        Soln++;
    }
    printf("%d\n", Soln);
    for (int i=1;i<=n;i++) if (Sol[i]>0)
        printf("%d %d\n", i, Sol[i]);

    return 0;
}
```