Uva 10243

Fire! Fire!! Fire!!!

Time: 3 seconds

Problem Descriptions (1/2)

- The ACM (Asian Cultural Museum) authority is planning to install fire exits in its galleries in order to handle the emergency situation arising in case of a sudden fire.
- The museum is a collection of numerous interconnected galleries.
- The galleries are connected by corridors in such a way that from any gallery there is exactly one path to reach any other gallery without visiting any intermediate gallery (a gallery that is on that path) more than once.

Problem Descriptions (2/2)

- However, in order to reduce installation cost, it has been decided that not every gallery will have a fire exit.
- Fire exits will be installed in such a way that if any gallery does not have a fire exit then at least one of its adjacent galleries must have one and for each corridor at least one of the two galleries it connects must have a fire exit. You are hired to determine where to put the fire exits under this constraint.
- However, as a first step, you are expected to determine the minimum number of fire exits required.

Input (1/3)

- The input file may contain multiple test cases.
- The first line of each test case contains an integer N (1 ≤ N ≤ 1, 000) indicating the number of galleries in this test case.
- Then follow N lines where the i-th (1 ≤ i ≤ N) line is the adjacency list of the i-th gallery.
- (Each gallery is given a unique identification number from 1 to N for convenience.)

Input (1/3)

- The adjacency list for *gallery i* starts with an integer n_i (1 ≤ n_i ≤ N − 1) indicating the number of galleries adjacent to this gallery, followed by n_i integers giving the identification numbers of those galleries.
- A test case containing a zero for N terminates the input.

Output

- **For each test case in the input file print a line containing the minimum number of fire exits required to meet the given constraint.**

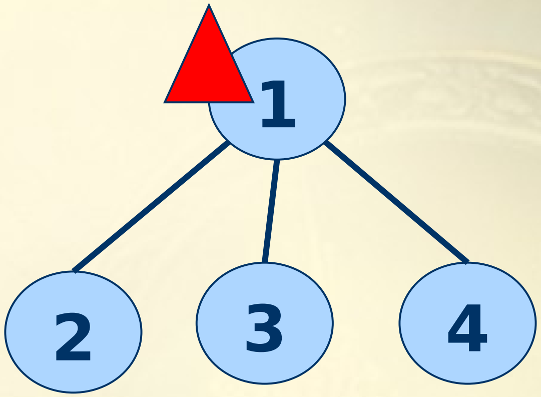
Sample I/O

```
4
3 2 3 4
1 1
1 1
1 1
16
4 6 12 15 16
3 3 8 10
4 2 4 6 9
1 3
1 6
3 1 3 5
1 15
1 2
1 3
1 2
1 16
1 1
1 15
1 15
4 1 7 13 14
2 1 11
0
```

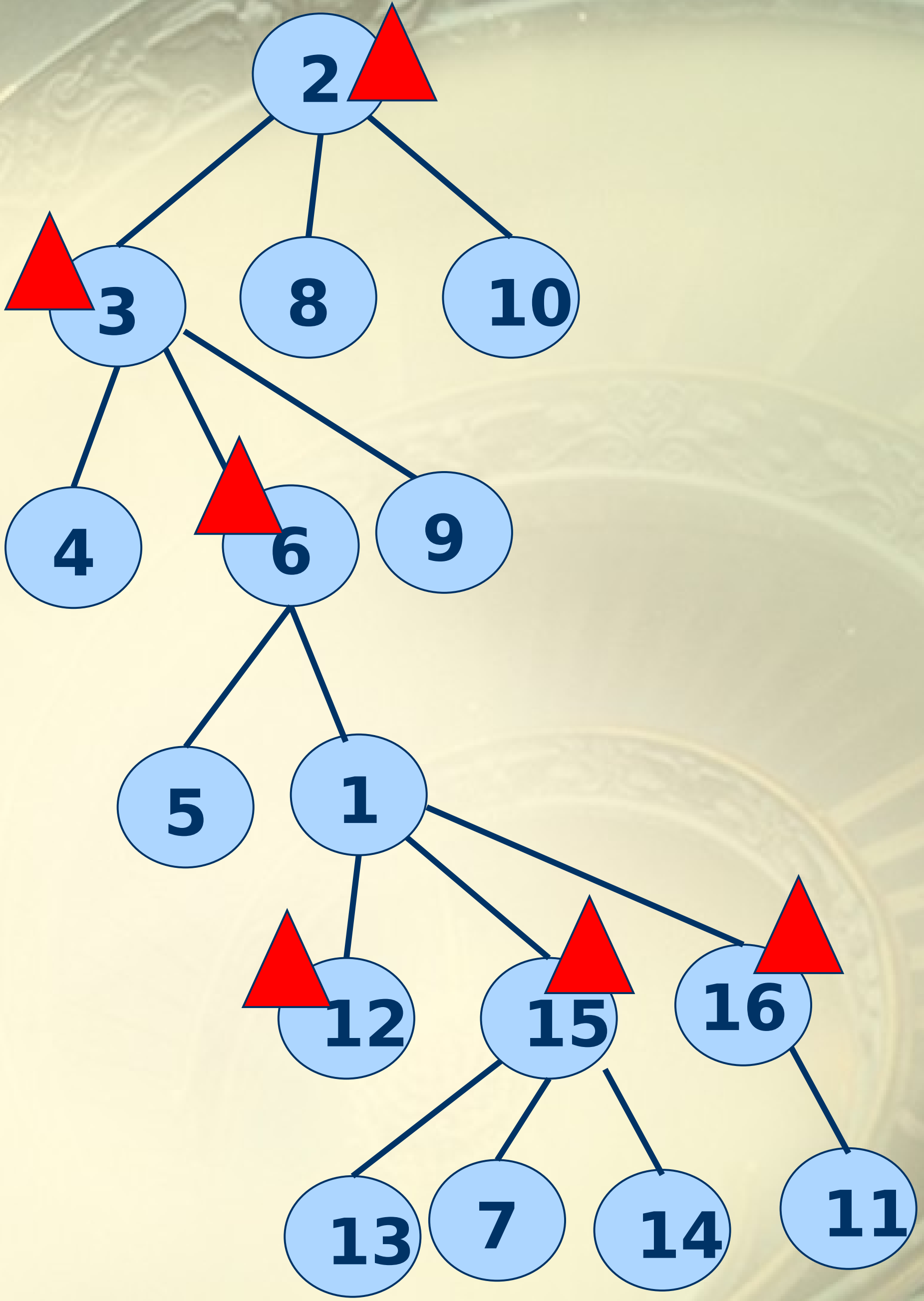
num of galleries

galleries that 1st gallery connected

```
1
6
```



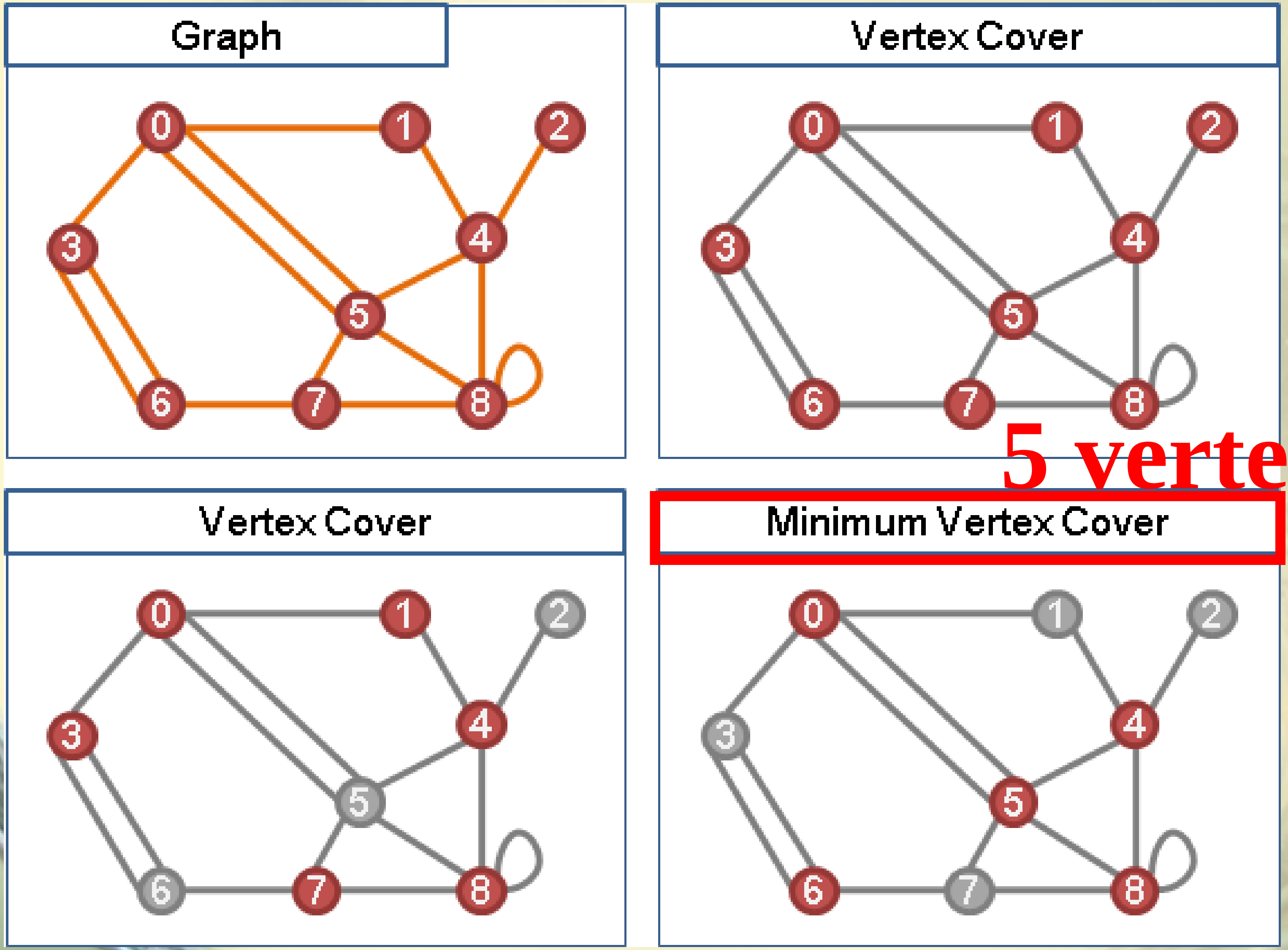
```
4
3 2 3 4
1 1
1 1
1 1
16
4 6 12 15 16
3 3 8 10
4 2 4 6 9
1 3
1 6
3 1 3 5
1 15
1 2
1 3
1 2
1 16
1 1
1 15
1 15
4 1 7 13 14
2 1 11
0
```



Minimum Vertex Cover (1/5)

- A vertex cover of an undirected graph is a subset of its vertices such that for every edge (u, v) of the graph, either ‘u’ or ‘v’ is in vertex cover.
- Although the name is Vertex Cover, the set covers all edges of the given graph. Given an undirected graph, the vertex cover problem is to find minimum size vertex cover.

Minimum Vertex Cover (2/5)



5 vertexes

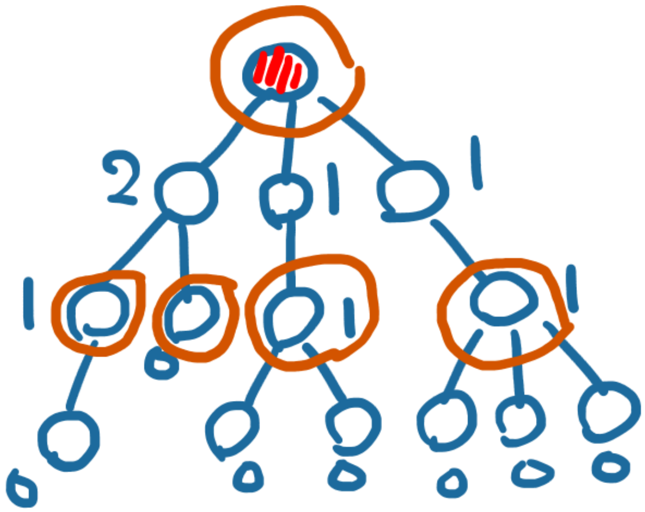
Minimum Vertex Cover (3/5)

- **Minimum Vertex Cover [NP-complete]**

- **In Tree [P- 問題]**
  - **Dynamic Programming**

- **In Bipartite Graph [P- 問題]**
  - **轉 Maximum Cardinality Bipartite Matching**

Minimum Vertex Cover in a tree(4/5)



d[u] : minimun vertex cover of u

$$= \min \begin{cases} ① \text{ u is in} \\ \quad 1 + \sum_{v \text{ is children of } u} d[v] \\ ② \text{ u is out} \\ \quad \text{num of u's children} \\ \quad + \sum_{x \text{ is grandchildren of } u} d[x] \end{cases}$$

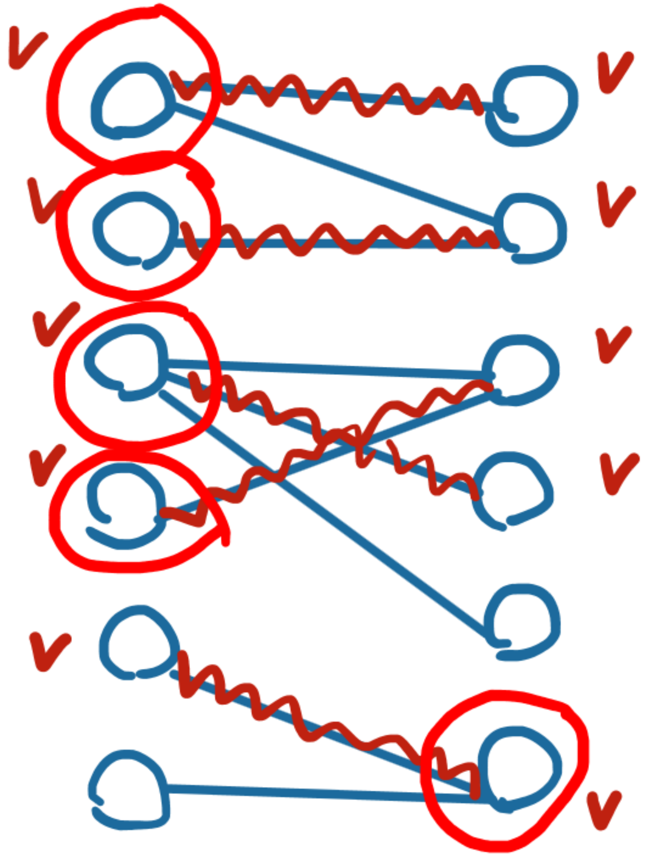
① $1+(2+1+1)=5$

② $3+(1+0+1+1)=6$

→ 5 #

copy to clipboard

Minimum Vertex Cover in Bipartile (5/5)



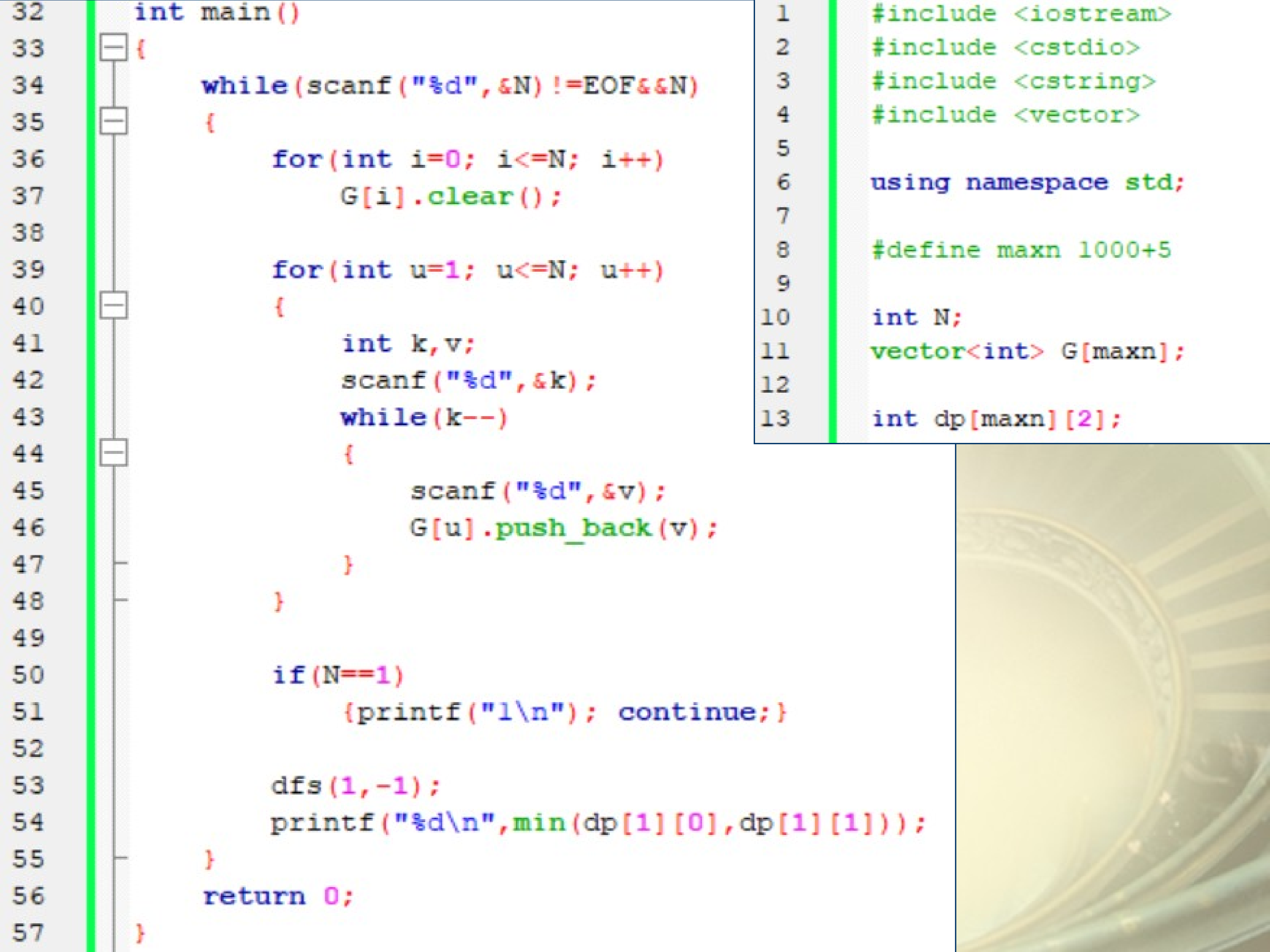
minimun vertex cover in Bipartile

≡ num of Maximun Match

↓

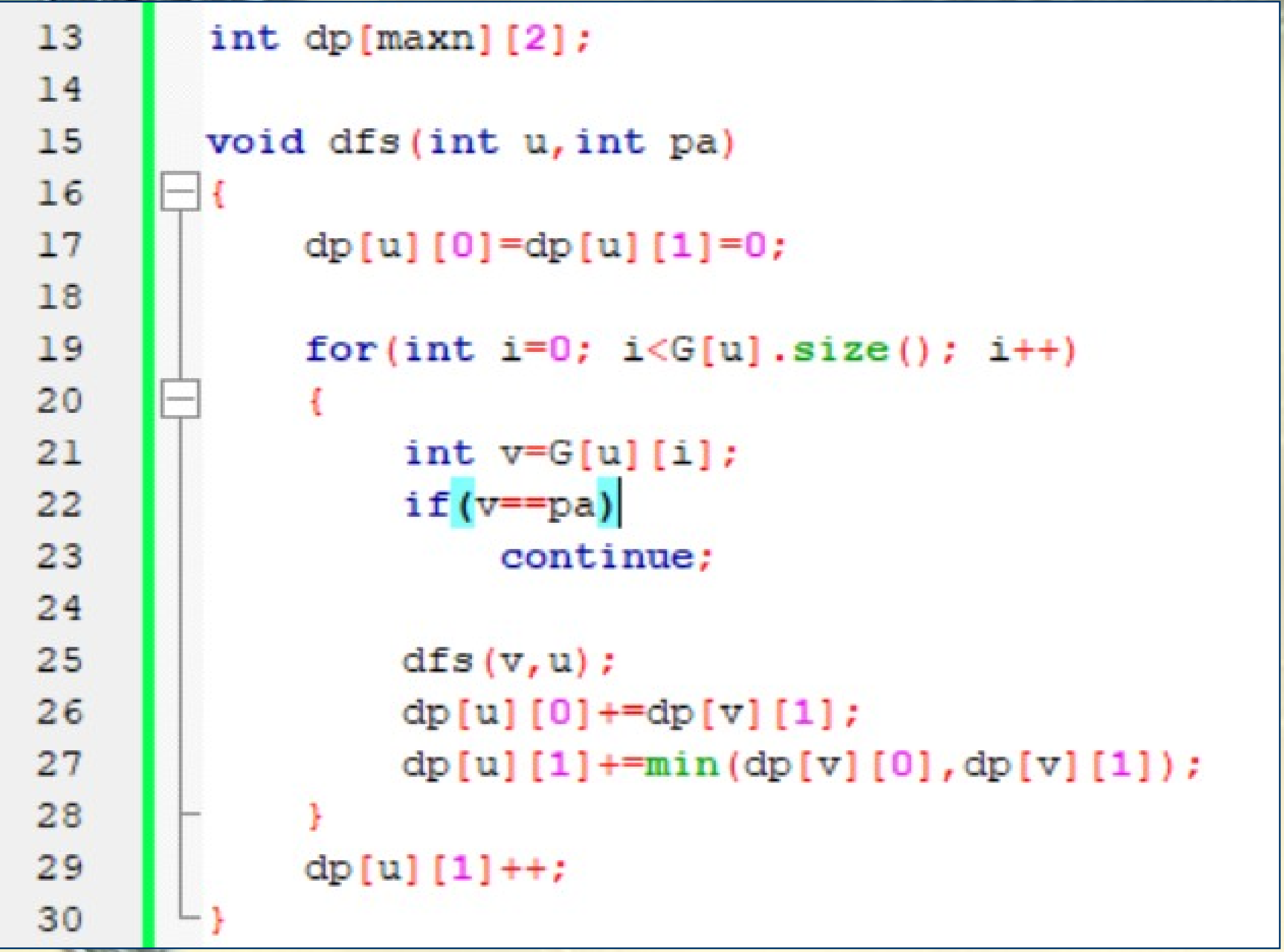
KM algorithm #

5



```cpp
#include <iostream>
#include <cstdio>
#include <cstring>
#include <vector>

using namespace std;

#define maxn 1000+5

int N;
vector<int> G[maxn];

int dp[maxn][2];
```

```cpp
int main()
{
    while(scanf("%d",&N)!=EOF&&N)
    {
        for(int i=0; i<=N; i++)
            G[i].clear();

        for(int u=1; u<=N; u++)
        {
            int k,v;
            scanf("%d",&k);
            while(k--)
            {
                scanf("%d",&v);
                G[u].push_back(v);
            }
        }

        if(N==1)
            {printf("1\n"); continue;}

        dfs(1,-1);
        printf("%d\n",min(dp[1][0],dp[1][1]));
    }
    return 0;
}
```



```
int dp[maxn][2];

void dfs(int u,int pa)
{
    dp[u][0]=dp[u][1]=0;

    for(int i=0; i<G[u].size(); i++)
    {
        int v=G[u][i];
        if(v==pa)
            continue;

        dfs(v,u);
        dp[u][0]+=dp[v][1];
        dp[u][1]+=min(dp[v][0],dp[v][1]);
    }
    dp[u][1]++;
}
```