Discusión de los problemas

17 09 2014

Índice general

1. Introducción **1**

2. Problem A: Hardwood Species **3**

3. Problem B: Dominos **5**

Capítulo 1

Introducción

Presentamos estas soluciones para los futuros competidores, esperando que les sean útiles las breves discusiones que hacemos de los problemas y que puedan hacerse una idea a la forma en que deben abordarse los problemas propuestos.

Como competidores sabemos lo duro que resulta iniciarse en estas competencias, por eso queremos compartir no sólo nuestras soluciones, sino también explicaciones claras y detalladas sobre la forma de llegar a la solución.

A continuación listamos las personas que aportaron con sus soluciones para este documento:

Problema	**Autor de la solución**	**Autor de la discusión**
A	Juan Felipe Cañizares C.	Daniel Cañizares C.
B	Juan Felipe Cañizares C.	Juan Felipe Cañizares C.
C	Juan Felipe Cañizares C.	Juan Felipe Cañizares C.
D	Brian Giraldo E. Juan Felipe Cañizares C.	Daniel Cañizares C.
E	Brian Giraldo E.	Brian Giraldo E.

Capítulo 2

Problem A: Hardwood Species

En este problema hay que determinar, dadas unas especies de árboles, qué porcentaje representan del total. Este tipo de problemas de clasificación y agrupación de datos pueden resultar en *time-outs* para los competidores nóveles sino escogen una estructura correcta. De hecho, el competidor tendrá problemas si quiere definir los datos en un simple vector, porque los índices no son enteros $(0, 1, \ldots, n)$, por el contrario, los índices están dados por el nombre de la especie, de forma que podamos contar fácil y rápidamente las veces que aparece cada especie, por ejemplo:

1	3	2	$\cdots$	$\#apariciones$
Ash	Gum	Cherry	$\cdots$	Especie n

Dicho esto, necesitamos una estructura en la que podamos definir el índice como *string*, para lo cuál usaremos un *map<string, double >trees* , el *double* es para que al final realicemos realicemos la división que nos de el porcentaje sin necesidad de hacer un casting. Dicho porcentaje está dado por la siguiente fórmula:

$$\%especie = \frac{trees[especie]}{total} \tag{2.1}$$

En el ejemplo, trees[Ash] valdría 1, trees[Gum] valdría 3, y así sucesivamente. Para hallar el total basta con contar cuántos datos nos ingresan.

No sobra resaltar, que por el formato de la entrada se debe utilizar *getline*, de forma que podamos saber cuando termina un caso de prueba, que para este problema está especificado con una línea vacía (que el *cin* ignoraría).

Código fuente

```cpp
#include <bits/stdc++.h>

using namespace std;

// Casos de prueba
int c;
// Mapa de las especiea
map<string,double>trees;

int main()
{
    while(cin>>c){
        string l; getline(cin,l);
        for(int t = 0; t < c; ++t){
            double pob = 0;
            // Lee las especies del caso de prueba
            while(getline(cin,l)){
                // Termina con linea vacia
                if(l == "")break;
                // Cuenta la especie
                pob++;
                // Si no esta en el mapa lo agrega
                if(!trees.count(l))
                    trees[l]=0;
                // Anade +1 a la cuenta de la especie
                trees[l]++;
            }
            // Recorremos el mapa
            for(map<string,double>::iterator
                it = trees.begin();
                it != trees.end();
                ++it){
                // Imprimimos la especie
                cout<<it->first<<" ";
                // Imprimimos el %
                cout<<fixed<<setprecision(4)
                    <<(it->second/pob*100)<<"\n";
            }
            trees.clear();
        }
    }
    return 0;
}
```

Capítulo 3

Problem B: Dominos

Código fuente

```cpp
#include <bits/stdc++.h>

using namespace std;

const int MAX = 100100;
int c, n , m;

vector<int> graph[MAX];
vector<int> topo;
int seen_t[MAX], seen[MAX];

void ini()
{
    for(int i = 0; i <= n; ++i)graph[i].clear();
    memset(seen_t, 0, sizeof seen_t);
    memset(seen, 0, sizeof seen);
    topo.clear();
}

void topo_sort(int u)
{
    seen_t[u]=1;
    for(int i = 0; i < graph[u].size(); ++i){
        if(seen_t[graph[u][i]] != 0)continue;
        topo_sort(graph[u][i]);
    }
    topo.push_back(u);
}

void DotopoSort()
{
```

```
32      for(int i = 1; i <= n; ++i){
33          if(seen_t[i] == 0)topo_sort(i);
34      }
35      reverse(topo.begin(), topo.end());
36  }
37
38  void dfs(int u)
39  {
40      seen[u]=1;
41      for(int i = 0; i < graph[u].size(); ++i){
42          if(seen[graph[u][i]] != 0)continue;
43          dfs(graph[u][i]);
44      }
45  }
46
47  int main()
48  {
49      while(cin>>c){
50          for(int t = 0; t < c; ++t){
51              cin>>n>>m;
52              ini();
53              for(int i = 0; i < m; ++i){
54                  int x,y; cin>>x>>y;
55                  graph[x].push_back(y);
56              }
57              DotopoSort();
58              int ans = 0;
59              for(int i = 0; i < n; ++i){
60                  if(seen[topo[i]]==0){
61                      dfs(topo[i]);ans++;
62                  }
63              }
64              cout<<ans<<endl;
65          }
66      }
67      return 0;
68  }
```