MÁXIMA COMPONENTE CONEXA

UNIVERSIDAD COMPLUTENSE MADRID

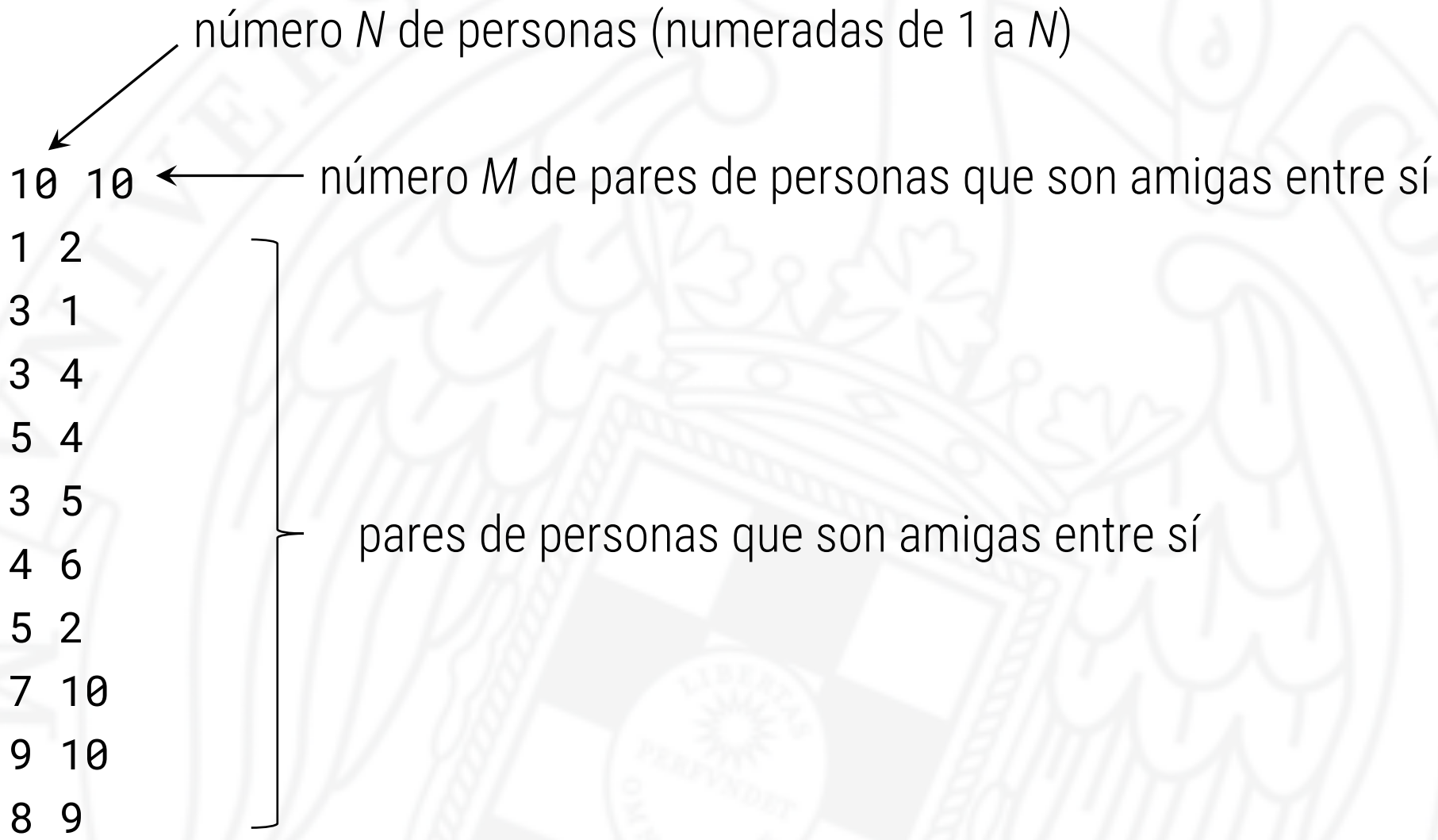
ALBERTO VERDEJO

Los amigos de mis amigos son mis amigos

En esta ciudad vive una serie de personas, y sabemos que algunas de ellas son amigas entre sí. De acuerdo con el refrán que dice "*Los amigos de mis amigos son mis amigos*", sabemos que si *A* y *B* son amigos y *B* y *C* son amigos, entonces también son amigos *A* y *C*.

Nuestra misión consiste en contar las personas en el grupo de amigos más grande.

Entrada


número N de personas (numeradas de 1 a N)
10 10
número M de pares de personas que son amigas entre sí
1 2
3 1
3 4
5 4
3 5
4 6
5 2
7 10
9 10
8 9
pares de personas que son amigas entre sí

Solución

- Representamos los datos del problema mediante un grafo donde los vértices representan a las personas y dos vértices están conectados mediante una arista si sabemos que son amigos.
- Cada componente conexa representa un grupo de amigos. Podemos recorrerla con un recorrido en profundidad que vaya contando su número de vértices. Necesitamos recorrer todas las componentes recordando cuál es el tamaño mayor.
- La complejidad de la solución está en $O(N + M)$, donde N es el número de personas y M el número de relaciones de amistad dadas.

Implementación

```
class MaximaCompConexa {
public:
   MaximaCompConexa(Grafo const& g) : visit(g.V(), false), maxim(0) {
      for (int v = 0; v < g.V(); ++v) {
         if (!visit[v]) { // se recorre una nueva componente conexa
            int tam = dfs(g, v);
            maxim = max(maxim, tam);
         }
      }
   }
   // tamaño máximo de una componente conexa
   int maximo() const {
      return maxim;
   }
```

Implementación

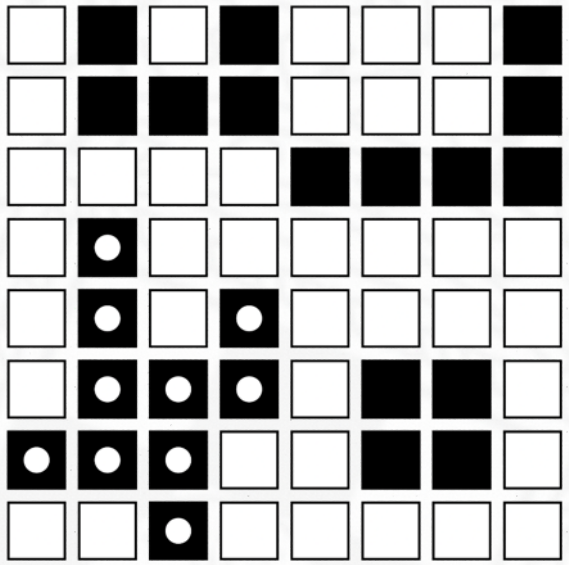
```
private:
   vector<bool> visit;  // visit[v] = se ha visitado el vértice v?
   int maxim;           // tamaño de la componente mayor

   int dfs(Grafo const& g, int v) {
      visit[v] = true;
      int tam = 1;
      for (int w : g.ady(v)) {
         if (!visit[w])
            tam += dfs(g, w);
      }
      return tam;
   }
};
```

Implementación

```
void resuelveCaso() {  // O(N + M)
   int N, M;
   cin >> N >> M;

   Grafo amigos(N);

   int v,w;
   for (int i = 0; i < M; ++i) {
      cin >> v >> w;
      amigos.ponArista(v-1, w-1);
   }

   MaximaCompConexa mcc(amigos);
   cout << mcc.maximo() << '\n';
}
```

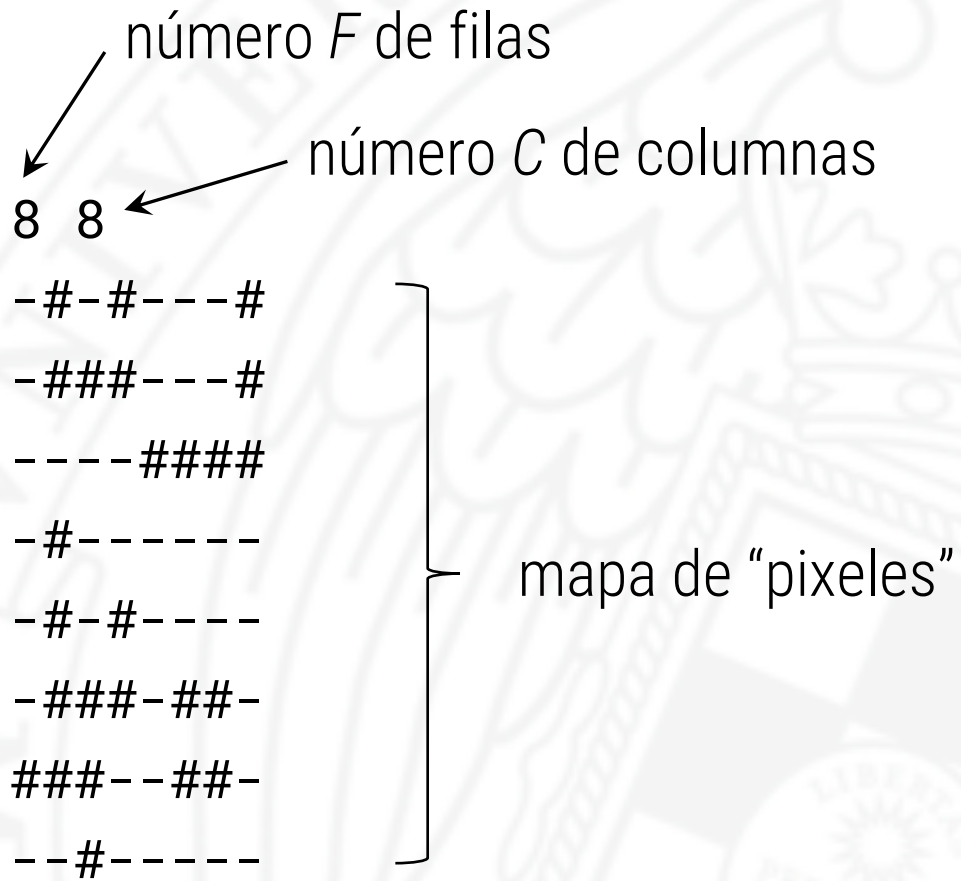
Detección de manchas negras

Dado un *bitmap* de píxeles blancos y negros, queremos saber el número de manchas negras que contiene y el tamaño (número de píxeles) de la mayor.



En esta imagen aparecen 4 manchas y la mancha más grande (marcada con puntos blancos) tiene 10 píxeles.

Entrada



Implementación

```
using Mapa = vector<string>;  // grafo implícito en el mapa

class Manchas {
public:
   Manchas(Mapa const& M) : F(M.size()), C(M[0].size()),
                            visit(F, vector<bool>(C, false)), num(0), maxim(0) {
      for (int i = 0; i < F; ++i) {
         for (int j = 0; j < C; ++j) {
            if (!visit[i][j] && M[i][j] == '#') { // se recorre una nueva mancha
               ++num;
               int nuevotam = dfs(M, i, j);
               maxim = max(maxim, nuevotam);
            }
         }
      }
   }
```

Implementación

```
    int numero() const { return num; }

    int maximo() const { return maxim; }


private:
    int F, C; // tamaño del mapa
    vector<vector<bool>> visit;  // visit[i][j] = se ha visitado el píxel <i,j>?
    int num;     // número de manchas
    int maxim;   // tamaño de la mancha más grande

    bool correcta(int i, int j) const {
        return 0 <= i && i < F && 0 <= j && j < C;
    }
```

Implementación

```
    const vector<pair<int,int>> dirs = {{1,0},{0,1},{-1,0},{0,-1}};

    int dfs(Mapa const& M, int i, int j) {
        visit[i][j] = true;
        int tam = 1;
        for (auto d : dirs) {
            int ni = i + d.first, nj = j + d.second;
            if (correcta(ni,nj) && M[ni][nj] == '#' && !visit[ni][nj]) {
                tam += dfs(M, ni, nj);
            }
        }
        return tam;
    }
};
```

Implementación

```
bool resuelveCaso() {  // O(F*C)
    int F, C;
    cin >> F >> C; // número de filas y columnas
    if (!cin) return false;

    Mapa mapa(F);

    // leemos la imagen
    for (string & linea : mapa)
        cin >> linea;

    // la analizamos
    Manchas manchas(mapa);
    cout << manchas.numero() << ' ' << manchas.maximo() << '\n';
    return true;
}
```