Algorithm Unibo

Grafi

Giovanni Spadaccini

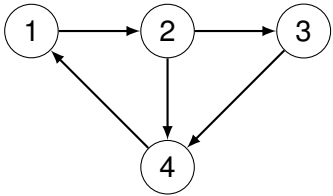
Università di Bologna

t.me/algo_unibo

Outline

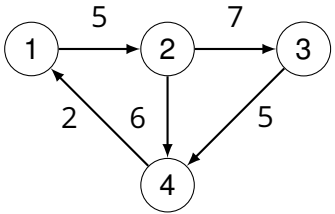
Cos’è un grafo

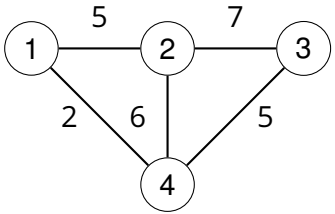
Un grafo è un insieme di nodi e di archi.



Cos’è un grafo

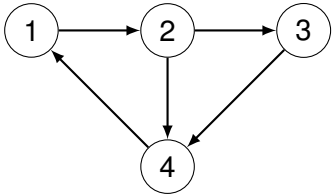
Un grafo è un insieme di nodi e di archi. Gli archi possono anche contenere informazioni e può essere diretto o non diretto.





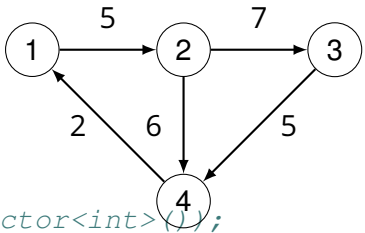
Come li memorizziamo

```
vector<int> adj[5];
adj[1].push_back(2);
adj[2].push_back(3);
adj[2].push_back(4);
adj[3].push_back(4);
adj[4].push_back(1);
///vector<vector<int>> adj(5,vector<int>());
```

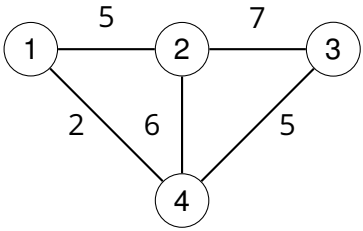


Come li memorizziamo

```
vector<array<int,2>> adj[5];
adj[1].push_back({2,5});
adj[2].push_back({3,7});
adj[2].push_back({4,6});
adj[3].push_back({4,5});
adj[4].push_back({1,2});
///vector<vector<array<int,2>>> adj(5,vector<int>());
```



Come li memorizziamo




```
vector<array<int,2>> adj[5];
adj[2].push_back({1,5});
adj[3].push_back({2,7});
adj[4].push_back({2,6});
adj[4].push_back({3,5});
adj[1].push_back({4,2});
adj[1].push_back({2,5});
adj[2].push_back({3,7});
adj[2].push_back({4,6});
adj[3].push_back({4,5});
adj[4].push_back({1,2});
```

BFS

BFS, visita in ampiezza.
Serve per esplorare un grafo e possiamo rispondere a domande:

- come se un nodo è raggiungibilie da un nodo di partenza
- a che distanza è ciascun nodo dal nodo di partenza.

Costo algoritmico $O(n + m)$ dove n è il numero di nodi e m è il numero di archi

BFS

```
queue<int> q;
bool visited[N];
int distance[N];
visited[x] = true;
distance[x] = 0;
q.push(x);
while (!q.empty()) {
    int s = q.front(); q.pop();
    //for (auto u : adj[s]) {
    for(int i=0;i<adj[s].size();i++) {
        int u=adj[s][i];
        if (visited[u]) continue;
        visited[u] = true;
        distance[u] = distance[s]+1;
        q.push(u);
    }
```

DFS

Depth-first search, è un algoritmo sempre per visitare un grafo, può essere più intuitivo ma viene implementato ricorsivamente

```
vector<int> adj[N];
bool visited[N];
void dfs(int s) {
    if (visited[s]) return;
    visited[s] = true;
    for (auto u: adj[s]) {
        dfs(u);
    }
}
```

Problema Assieme

https://cses.fi/problemset/task/1666/

Byteland has n cities, and m roads between them. The goal is to construct new roads so that there is a route between any two cities.

Your task is to find out the minimum number of roads required, and also determine which roads should be built.

Input

The first input line has two integers n and m: the number of cities and roads. The cities are numbered $1, 2, \ldots, n$.

After that, there are m lines describing the roads. Each line has two integers a and b: there is a road between those cities.

A road always connects two different cities, and there is at most one road between any two cities.

Output

First print an integer k: the number of required roads.

Then, print k lines that describe the new roads. You can print any valid solution.

Constraints

- $1 \leq n \leq 10^5$
- $1 \leq m \leq 2 \cdot 10^5$
- $1 \leq a, b \leq n$

Example

Input:
```
4 2
1 2
3 4
```

Output:

Mia soluzione

```
using namespace std;
const int MN = 100002;
int n, k,a,b;
vector<int> adj[MN];
vector<bool> visited;
vector<array<int, 2>> sol;

void dfs(int c) {
  visited[c] = true;
  for (int i = 0; i < adj[c].size(); i++) {
    int current = adj[c][i];
    if (!visited[current]) {
      dfs(current);
    }
  }
}
```

```
cin >> n >> k;
visited = vector<bool>(n, false);
for (int i = 0; i < k; i++) {
  cin >> a >> b; a--;b--;
  adj[a].push_back(b);adj[b].push_back(a
}
for (int i = 0; i < n; i++) {
  if (!visited[i]) {
    dfs(i);
    if (i != 0) {sol.push_back({0, i});}
  }
}
cout << sol.size() << endl;
for (int i = 0; i < sol.size(); i++) {
  cout << sol[i][0] + 1 << " "
  << sol[i][1] + 1 << endl;
}
```

Dijkstra

Dijkstra, è un algorimo per trovare la path meno costosa tra due nodi di un grafo pesato . (notare che il caso non pesato è equivalente a una bfs)
L'algoritmo ha un costo di $O(n + m \log m)$

Dijkstra

```
for (int i = 1; i <= n; i++) distance[i] = INF;
distance[x] = 0;
q.push({0,x});
while (!q.empty()) {
    int a = q.top().second; q.pop();
    if (processed[a]) continue;
    processed[a] = true;
    for (auto u : adj[a]) {
        int b = u.first, w = u.second;
        if (distance[a]+w < distance[b]) {
            distance[b] = distance[a]+w;
            q.push({-distance[b],b});
        }
    }
}
```

Problemi

- https://cses.fi/problemset/task/1682
- https://cses.fi/problemset/task/1683
- https://cses.fi/problemset/task/1193
- https://cses.fi/problemset/task/1671
- https://codeforces.com/contest/17/problem/B
- https://codeforces.com/contest/35/problem/C