Graphs
(Programming Club 3)

Tomasz Kosciuszko

05.02.2018

1 Introduction

What is a graph? It is a set of verteces (like places) together with connections between some of them (like roads). There can be different kinds of graphs, directed, weighted, full, trees... Most of the time we are concerned with questions about optimal paths between verteces. We use DFS and BFS algorithms, with their many variations, for unweighted graphs. Dijkstra, Bellman-Ford among many others are used for weighted graphs. There are also other interesting problems, like maximum flow problem, graph isomorphism or colouring of verteces. See the wikipedia pages for more information:

- Graph `en.wikipedia.org/wiki/Graph_(discrete_mathematics)`
- DFS `en.wikipedia.org/wiki/Depth-first_search`
- BFS `en.wikipedia.org/wiki/Breadth-first_search`
- Dijkstra `en.wikipedia.org/wiki/Dijkstra's_algorithm`
- Bellman-Ford `en.wikipedia.org/wiki/Bellman\OT1\textendashFord_algorithm`
- Isomorphism `en.wikipedia.org/wiki/Graph_isomorphism`
- Coloring `en.wikipedia.org/wiki/Graph_coloring`
- Maximum flow `en.wikipedia.org/wiki/Maximum_flow_problem`

2 Problem example - "The Two Routes"

Let's have a look at this problem's statement:
`http://codeforces.com/problemset/problem/601/A`
We are presented with a graph and have to find two routes from the town 1 to the town n. One has to go only through the given edges, the other one must

not go through any. The longer route has to be shortest possible and they must not intersect, except for the first and the last city.

First, let's decide on a good representation of the graph. I will stick to my favourite C++, but in all languages we have 2 ways of approaching the representation. Either we keep an $n \times n$ matrix with all distances, or we have an array of lists, each representing neighbours of a vertex. Both options have advantages, but the second one is more common.

Nevertheless, in this problem I have decided to use a matrix. Let's read the input and fill the matrix with 1s wherever a railway connection occurs.

```
int n, m;
cin >> n >> m;
for (int i = 0; i < m; i++) {
        int a, b;
        cin >> a >> b;
        graph[a][b] = 1;
        graph[b][a] = 1;
}
```

Now, how about this observation: we can either get from 1 to n directly by road or by rail. Which means our shorter path will have length 1, let's take it and make the vehicle wait in the town n. We just need to find the other route, such that it is shortest possible. That should be easily done later. To choose the right vehicle let's check the edge at the position $graph[1][n]$ and if it is 1 then flip all 0s to 1s and 1s to 0s. This way we can next deal with both cases in the same way:

```
if (graph[1][n] == 1) {
        for (int i = 1; i <= n; i++) {
                for (int j = 1; j <= n; j++) {
                        graph[i][j] = !graph[i][j];
                }
        }
}
```

The only thing left is a standard problem: given a graph find the shortest path between two verteces. We will use the BFS algorithm for that. In each 'move' we try to expand all unexpanded verteces from the queue. This way we ensure that after k 'moves' we reach all verteces at distance k. Please read on the internet if you need a detailed explanation, here is the code of my BFS implementation:

```
dist[1] = 0;
visited[1] = 1;
queue.push_back(1);
for (int i = 0; i < queue.size(); i++) {
        for (int j = 1; j <= n; j++) {
                if (graph[queue[i]][j] && visited[j] == 0) {
                        dist[j] = dist[queue[i]] + 1;
                        visited[j] = 1;
                        queue.push_back(j);
                        if (j == n) {
                                cout << dist[j] << "\n";
                                return 0;
                        }
                }
        }
}
cout << "-1\n";
```

Now try to put the example code together and compile it. Or, write your own if you prefer. All popular programming languages are supported on codeforces! Once you are convinced it works submit it via codeforces to check if it passes all the tests.

3 Important remarks

- In many problems it might be not obvious to represent data as a graph. Always check whether something useful comes from it.
- Try your algorithm on many different examples, once you have come up with one. There might be some nasty cases you have not considered! Like separated verteces, full graphs...
- Notice that the topics from the last two weeks are applicable when dealing with graphs. We know both greedy and DP examples of graph problems solutions.
- If you have an account on codeforces, consider setting your Univeristy as 'University of Edinburgh' to plug yourself into the graph of Edinburgh coders!
- If you are not sure how to deal with **standard input and output** in your favourite language, have a look at some accepted solutions by other people: `http://codeforces.com/problemset/status`.

4 Practice problems

- Party (easy):
  `http://codeforces.com/problemset/problem/115/A`
- Dijkstra? (easy):
  `http://codeforces.com/problemset/problem/20/C`
- Bipartiteness (medium):
  `http://codeforces.com/problemset/problem/862/B`
- Colorful Graph (medium):
  `http://codeforces.com/problemset/problem/505/B`
- Checkpoints (hard):
  `http://codeforces.com/problemset/problem/427/C`

5 Hints

- Party: Try to think in terms of the 'depth' of the graph.
- Dijkstra?: Dijkstra?
- Bipartiteness: How many ways are there to bipartite a tree?
- Colorful Graph: Just use DFS many times.
- Checkpoints: Try googling 'Strongly connected component'.