T-301-REIR, REIKNIRIT
HAUST 2019
D5 – GRAPHS

Problem 1. You have managed to learn all friendship connections in the social network Basehook(Bh).You want to forward a message from one person to another, but have observed that the impact of a message depends a lot on along how many connections it has to travel. You therefore want to compute the following: Given a sequence of friendship connections and specific persons A and B, determine the smallest k such that there is a sequence of persons $X_0,X_1,X_2,...,X_k$ where $A = X_0$, $B = X_k$, and X_i and X_{i+1} are Bh-friends, for $i = 0,1,...,k-1$. Describe briefly an efficient solution in words to this problem, using the concepts of this course, and give its time complexity.

- Preferable way to implement this would be to use a BFS graph (Breadth-First-Search), where a given source vertex (A) support queries of this particular form, if there is a connection between A and B and what the shortest path between these two vertices is (single-source shortest path).

  This gives the time complexity of E + V in the worst case scenario, where E is the number of edges (connections between two people in the graph, from source to end result) and V is the number of vertices (people you have to travel between to connect A and B).

Problem 2. (Continued) You have studied this further and discovered that not all friendships are equal. Strong friendship connections are special, and using such friendship connections doesn't cost anything so to speak: the message can travel along arbitrarily many strong friendships connections without its impact becoming weaker. We call other connections weak. You want to compute the fewest number of weak friendship connections needed to route a message from given person A to person B (possibly using many strong connections).

- Two vertices A and B are strongly connected if they are mutually reachable; that is, if there is a directed path from A to B and a directed path from B to A. Thereby, cycles do play an important role in these strong connectivity's – two vertices are strongly connected if and only if there exists a general directed cycle that contains them both.

  If two persons are already friends, they would have a strong connection. So, if we store all friendship connections with the source (A) in an array, an all other friends connection that are connected to these vertices in another array, we can first look up if connections lies within the first array (close friends of A), and if they do not exist, check if there is connection in the second array (friend of friends of A).

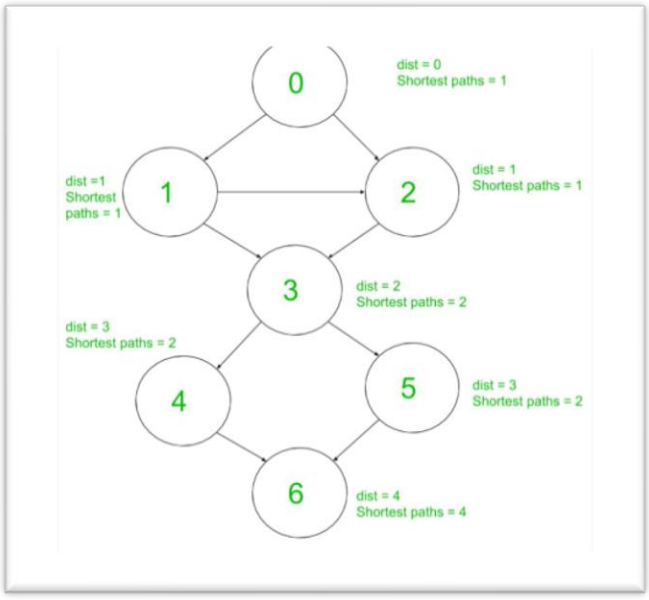
Problem 3. Modify BreadthFirstPaths.java to compute the number of shortest paths between two given vertices v and w in a given digraph. This replaces the method boolean hasPathTo(inv v) with int nrOfPathsTo(inv v). Hint: Note that if vertex w is of distance k from v, and it is adjacent to vertices a, b and c of distance k−1 from v, then each shortest v−w path runs through one of a, b, and c.

- Starting from the source vertex, which we will call 'v', we make a check to see if any path goes from v to the ending vertex, which we call 'w' – thereby given a path from v to w, and the count of edges between the two vertices gives us the shortest path.

  If we then check for another path, selecting different edges between the two vertices (v-w), and the count of edges are the same as before, we have two shortest paths between the two vertices.

  As each vertex has a path to itself, we can initialize the path count from v to v as being 1.
  Then, by checking each edge connected to v, we count the path to this edge as being 1 as well.

  But then it gets more complicated, as we have to check if there has already been found a connection between the two vertices (the vertex w is marked as 'true', as it has already been checked). And if the vertex has been marked as 'true', what is the path distance between the two vertices previously found (number of edges)?
  - If the path distance is less than in the new path found, then we already have the shortest path (and the count of edges between them), so we do nothing.
  - If the new path distance is less than the current path between v-w, then we have found a shorter path and have to update the distance count, the number of edges between the two vertices (and initialize again the number of paths to one).
  - If the new path distance is equal to the current path distance, we have found another shortest path from v to w, and therefore we increase the number of shortest paths by one.



Example for number of shortest path(s) and the distance from source vertex [1]

[1] https://www.geeksforgeeks.org/number-shortest-paths-unweighted-directed-graph/

Code:

```
private void bfs(Graph G, int s) {
	Queue<Integer> q = new Queue<Integer>();
	for (int v = 0; v < G.V(); v++)
		distTo[v] = INFINITY;
	distTo[s] = 0;		// Count to itself is 0
	marked[s] = true;
	pathCount[s] = 1;	// Single shortest path to itself
	q.enqueue(s);

	while (!q.isEmpty()) {
		int v = q.dequeue();
		for (int w : G.adj(v)) {
			if (!marked[w]) {
				//Edge count increased:
				distTo[w] = distTo[v] + 1;
				marked[w] = true;
				// Path v-w becomes one (1):
				pathCount [w] = 1;
				q.enqueue(w);
			}

			// And if path distance is the same:
			else if (distTo[w] == distTo[v] + 1)
				// Number of shortest path increased to v+w path counts:
				pathCount[w] += pathCount[v];
		}
	}
}
```

Problem4. Suppose the weight of every edge in a weighted graph is decreased by one. Explain (in 20 words or less) why the MST computed by Kruskal does not change.

- Because the MST computed by Kruskal is sorted by the weight of edges, therefore it does not change if every edge-weight is decreased by one (still sorted order).

Problem 5. Consider a graph with distinct edge weights such that Prim and Kruskal select the edges of the spanning tree T in opposite order. How must T look like?

- It does not have any circles within the spanning tree, having the tree connected to every vertex (with a single edge) in the tree, with a path from the furthest away vertex from the source till the original source of the tree.