Tutorial #14. Graph algorithms

Theoretical part

Traversal: DFS (Depth first search)

DFS (even it is called SEARCH algorithms) is mostly a **traversal** algorithms with a beautiful recursive implementation that can lead to stack overflow for dense graphs (that's why better use iterative implementation). For search it is used in **trees**. Usually algorithm is used as a part of greater tasks: in topological search, Dinic's algorithms for maximum flow calculation (for transport networks), etc. Naturally, this approach is used to **find way out from a maze** without any heuristics.

Consider depth first search as **preorder tree traversal of spanning tree**. We are interested in visiting all **nodes**, but edges are not important for us. You visit the node (root node of spanning tree), and then repeat operation for the first adjacent node (child in terms of spanning tree). To avoid duplicate visiting we use either recursion or stack (as in lecture).

```
dfs(node)
    visit(node);              // this can mean: add to the tree, e.g.
    for (Node n: node.adjacent())
         if (!n.visited) dfs(n);
```

Traversal: BFS (Breadth first search)

BFS is an algorithm of (a) graph traversal (b) finding a path. For **unweighted** graph it always produces the **shortest way**. BFS is also a way to build a spanning tree on the graph. It looks like building a tree level-by-level: we put one vertex on the top level (level 0, root), visit it. All adjacent vertices become children, visit them. Then we repeat search from each node of the first level. To preserve visiting order we can use queue, as show on the lecture.

```
void BFS() {
  Queue<> q = new Queue<Node>();
  root.marked = true;
  q.push(root);
  while (!q.empty()) {
      int v = q.dequeue();
      visit(v);
      for (Node n: v.adjacent()) {
          if (!n.marked) {
            n.marked = true;
            q.push(n);
          }
      }
  }
}
```

Minimum spanning tree for weighted tree

This task is very important for logistics – we can calculate minimum fuel consumption for logistics, minimum cable length to cover the area with network, etc. As we already know, spanning tree for unweighted tree is always a minimum tree. But that's unfortunately false for weighted graphs. How shall we build this tree? That's also very easy – let's iteratively select the best known candidate edge for our tree.

1) Start from any proposed root (you can always "hang" undirected tree on any vertex and call is "root") and mark it as **visited**.
2) Put *adjacent to newly visited node* **edges** into a waiting list as "candidates".

3) Pick up the shortest edge from the list and add it to the tree (we know both vertices, so we can find where to attach this edge).
4) Mark second node of the edge **as visited ("create an office/warehouse/router there")**.
5) Clean up the graph: remove all candidate edges, that connect 2 already visited nodes.
6) Repeat 2-5 until candidate list is empty.

Practical part

1) Read the graph from the proposed file.
2) Build spanning tree for this graph starting from «Москва» using DFS, BFS and minimum spanning tree algorithms. Calculate edge weight sum for all these cases. Save resulting graphs to a file.
3) *** Visualize the graph and spanning trees using city coordinates.**