Lab08-Graphs

VE281 - Data Structures and Algorithms, Xiaofeng Gao, TA: Li Ma, Autumn 2019

∗ Please upload your assignment to website. Contact webmaster for any questions.

∗ Name:Wu Jiayao Student ID:517370910257 Email: jiayaowu1999@sjtu.edu.cn

1. **DAG.** Suppose that you are given a directed acyclic graph $G = (V, E)$ with real-valued edge weights and two distinct nodes s and d. Describe an algorithm for finding a longest weighted simple path from s to d. For example, for the graph shown in Figure 1, the longest path from node A to node C should be $A \to B \to F \to C$. If there is no path exists between the two nodes, your algorithm just tells so. What is the efficiency of your algorithm? (Hint: consider topological sorting on the DAG.)

Solution.

Denote $< u, v >$ the edge goes from u to v.

```
Input: A DAG G, points s, d
Output: The longest path from s to d
Made a copy of G to do topological sort.
visitNum ← 0
H ← {}
while E ≠ ∅ do
    v ← a vertex in V that no edge points to it
    visitNum++
    H[visitNum] ← v
    V ← V\v
    for vertex u in V do
        if edge <v,u> exists then
            E ← E\<v,u>
foreach v in G.V do
    v.pathLength = ∞
foreach u in H do  // Visit in topological order
    foreach edge <u,v> do
        if v.pathLength < u.pathLength + edge <v,i>.weight then
            u.pathLength = v.pathLength + edge <v,i>.weight
            v.prev = u
if d.pathLength = ∞ then
    return Not found
itr = d
path = []
while True do
    push itr into the front of path
    if itr == s then
        break
    itr = itr.prev
return path
```

The time complexity is $\mathcal{O}(|V|+|E|)$, where $\mathcal{O}(|V|)$ for topological sort, $\mathcal{O}(|E|)$ for path length determination.

$\square$

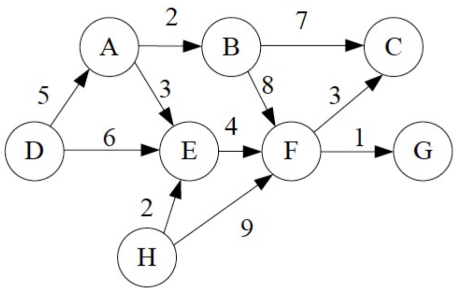

Figure 1: A weighted directed graph.

2. **ShortestPath.** Suppose that you are given a directed graph $G = (V, E)$ on which each edge $(u, v) \in E$ has an associated value $r(u, v)$, which is a real number in the range $0 \le r(u, v) \le 1$ that represents the reliability of a communication channel from vertex u to vertex v. We interpret $r(u, v)$ as the probability that the channel from u to v will not fail, and we assume that these probabilities are independent. Give an efficient algorithm to find the most reliable path between two given vertices.

Solution.

```
Input: A Graph G, points s, d
Output: The most reliable path from s to d
Set a priotity queue pq, where the value is the edges, the key is the current probability of v,
  v.p, in the edge, edge with the largest probability v is on the top.
visit[s] = 1
s.pathLength = 1
foreach edge < s, i > do foreach-comment
    i.p = s.p × r(s, i)
    i.prev = s
    push (s, i) into pq
while pq is not empty do
    (u, v) = pq.dequeue
    if visit[v] == 1 then
        continue
    foreach (v, i) do
        if visit[i] == 0 and i.p < v.p × r(v, i) then
            i.p = v.p × r(v, i)
            visit[i] = 1
            i.prev = v
            push (v, i) into pq
itr = d
path = []
while True do
    push itr into the front of path
    if itr == s then
        break
    itr = itr.prev
return path
```

□

3. **GraphSearch.** Let $G = (V, E)$ be a connected, undirected graph. Give an $O(|V| + |E|)$-time algorithm to compute a path in G that traverses each edge in E **exactly once in each direction**. For example, for the graph shown in Figure 2, one path satisfying the requirement is

$$A \to B \to C \to D \to C \to A \to C \to B \to A$$

Note that in the above path, each edge is visited exactly once in each direction.

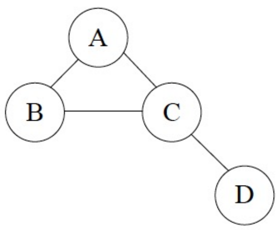

Figure 2: A undirected graph.

Solution.

```
Input: An undirected graph G < V, E >
Output: A path
Reconstruct the graph into a directed graph, as one undirected edge(u,v) means to directed
  edges (u,v),(v,u)
path=[]
Create a stack q
visit[v] = 0 for all vertices
prev[v] = v for all vertices
Push a vertice U that has an edge in G.E
while q is not empty do
    u = q.dequeue
    Append u to path
    visit[u] = 1
    foreach edge < u, v > do
        if visit[v] == 0 then
            push v into q
            prev[v] = u
            G.E ← G.E\ < u, v >
        else
            if prev[u]! = v then
                Append [v, u] into the path
                G.E ← G.E\ < u, v >
                G.E ← G.E\ < v, u >
return path
```

□