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: Sun Yiwen Student ID: 517370910213 Email: sunyw99@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.

Alg. 1: longestPath(G,s,d)

Input: graph $G = (V, E)$, directed; vertex $s \in V$; vertex $d \in V$
Output: a stack S of nodes that demonstrates the longest weighted simple path from node s to node d, with node s on the top and d in the bottom

```
for each u ∈ V do
    dist[u] = −∞;
dist[s] = 0;
topologicalSort(G);
for each u ∈ V in topological order do
    for each v ∈ Adj[u] do
        if dist[v] < dist[u] + w(u, v) then
            dist[v] ← dist[u] + w(u, v);
Create G^R = (V', E') as the reverse graph of G.
vertex temp ← d;
S.push(d);
while temp! = s do
    for each v ∈ Adj[temp] do
        if dist[temp] == dist[v] + w(temp, v) then
            temp ← v;
            S.push(v);
```

Alg. 2: topologicalSort(G)

Input: graph $G = (V, E)$, directed
Output: a stack S of nodes

```
Mark all the nodes as not visited;
for each u ∈ V do
    if u is not visited then
        topologicalSort_help(u,S);
```

Alg. 3: topologicalSort_help(u,S)

Input: vertex u,stack S

```
Set u as visited;
for each v ∈ Adj[u] do
    if v is not visited then
        topologicalSort_help(v,S);
    S.push(v);
```

The time complexity of my algorithm is $O(|V| + |E|)$.

□

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 $0 \leq r(u, v) \leq 1$

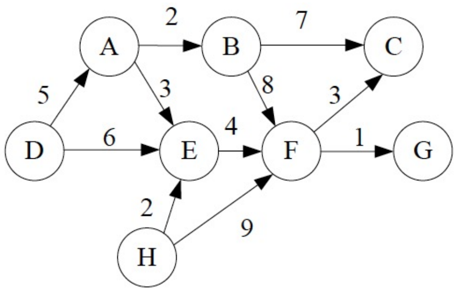

Figure 1: A weighted directed graph.

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.

□

Alg. 4: shortestPath(G,s,d)

Input: graph $G = (V, E)$, directed; vertex $s \in V$; vertex $d \in V$
Output: a stack S of vertexes that demonstrates the most reliable path from vertex s to vertex d, with s on the top and d in the bottom

for *each* $u \in V$ **do**
    $dist[u] = \infty$;
$dist[s] = 1$;
for *each* $u \in V$ **do**
    INSERT(Q, u);
while Q *is not empty* **do**
    $u \leftarrow$EXTRACT-MIN(Q);
    $S \leftarrow S \cup \{u\}$;
    for *each* $v \in Adj[u]$ **do**
        if $dist[v] > dist[u] * r(u, v)$ **then**
            $dist[v] \leftarrow dist[u] * r(u, v)$; DECREASE-KEY$(Q, v)$;
Create $G^R = (V', E')$ as the reverse graph of G.
vertex $temp \leftarrow d$;
S.push(d);
while $temp != s$ **do**
    for *each* $v \in Adj[temp]$ **do**
        if $dist[temp] == dist[v] * r(temp, v)$ **then**
            $temp \leftarrow v$;
            S.push(v);

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

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

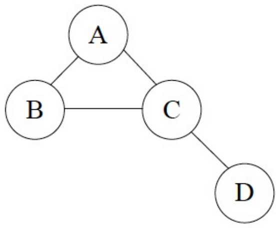

Figure 2: A undirected graph.

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

Solution.

Alg. 5: graphSearch(G)

Input: graph $G = (V, E)$

Output: a stack S of nodes that demonstrates the longest weighted simple path from node s to node d, with node s on the top and d in the bottom

for $v \in V$ **do**
  VISITED(v)=false;
for $e \in E$ **do**
  PASSED(e)=0;
for $v \in V$ **do**
  if *not VISITED(v)* **then**
    S.push(v);
    EXPLORE(G, v);

Alg. 6: EXPLORE(G, v, S)

Input: graph $G = (V, E)$; vertex $v \in V$; stack S that will be the final output of graphSearch(G)

VISITED(v)=true;
for *each* $u \in Adj[v]$ **do**
  if *PASSED(edge(u, v))==0* **then**
    PASSED(edge(u, v))++;
    S.push(u);
    EXPLORE(G, u, S);
for *each* $u \in Adj[v]$ **do**
  if *PASSED(edge(u, v))==1* **then**
    PASSED(edge(u, v))++;
    S.push(u);
    EXPLORE(G, u, S);

□