Theory Assignment 2

CSE222: *Algorithm Design & Analysis*

Divyajeet Singh (2021529)   Agamdeep Singh (2021306)

April 8, 2023

Problem-1

The police department in the city of Computopia has made all the streets one-way.

(a) The mayor of the city claims that it is possible to legally drive from an intersection to any other intersection. Formulate this problem as a graph theoretic problem and explain why it can be solved in linear time.

(b) Suppose the mayor's claim was found to be wrong. She has now made a weaker claim: *"If you start driving from the town-hall, navigating one-way streets, then no matter where you reach, there is always a way to drive legally back to the town-hall."* Formulate this weaker property as a graph theoretic problem and explain how it can be solved in linear time.

Input

A directed graph $G = (V, E)^*$ representing the layout of the city. V is the set of all intersections and E is the set of all one-way streets in Computopia.

Note: *Formulation of the problem as a graph theoretic problem is given in the subsequent subsection.*

Output

(a) A boolean value *True* if the mayor's claim is true, else *False*.

(b) A boolean value *True* the mayor's weaker claim is true, else *False*.

Solution

(a) Check if G is strongly connected in linear time using Kosaraju's algorithm.

(b) Check the townhall is a part of a strongly connected component in G.

Problem-2

Assume that n is the number of vertices in a graph. Then, given an edge-weighted connected undirected graph $G = (V, E)$ with $n + 20$ edges, design an algorithm that runs in $O(n)$ time and outputs an edge with the smallest weight, connected in a cycle of G. You must give a justification why your algorithm works correctly.

Input

An edge-weighted connected undirected graph $G = (V, E)$ such that $|V| = n$ and $|E| = n + 20$.

Output

An edge (u, v) with the smallest weight, connected in a cycle of G.

Solution

Terminologies and Definitions

Definition 1 (Cut-Edge) *An edge $(u, v) \in E$ is a cut-edge iff removing it from the graph $G = (V, E)$ disconnects G.*

The complete Algorithm

1. Maintain a hash table $C[(u, v) \in E] \leftarrow 0$ to mark the cut-edges.
2. Find all the cut-edges in G, which takes $O(|V| + |E|)$ time.
3. While finding the cut-edges, mark the edges that are cut-edges as 1 in C.
4. Find the edge with the smallest weight in E such that $C[e] \neq 1$.

Problem3

Suppose that G is a directed acyclic graph with the following features:

- G has a single source s and several sinks $t_1, t_2, \ldots, t_k$.
- Each edge $(v \to w)$ (i.e. an edge directed from v to w) has an associated weight $P[v \to w]$ between 0 and 1.
- For each non-sink vertex v, the total weight of all the edges leaving v is 1, i.e.

$$\sum_{(v \to w) \in E} P[v \to w] = 1 \tag{1}$$

The weights $P[v \to w]$ define a random walk in G from s to some sink t_i. After reaching any non-sink vertex v, the walk follows the edge $(v \to w)$ with probability $P[v \to w]$. All the probabilities are mutually independent.
Describe and analyze an algorithm to compute the probability that this random walk reaches sink t_i $\forall$ $i \in \{1, 2, \ldots, k\}$.

Input

A weighted directed acyclic graph $G = (V, E)$ having the above properties.

Output

A sequence of probabilities $P[t_1], P[t_2], \ldots, P[t_k]$, where $P[t_i]$ is the probability that the random walk reaches sink t_i $\forall$ $i \in \{1, 2, \ldots, k\}$.

Solution

The given problem can be efficiently solved by applying dynamic programming on the topologically sorted graph G. It allows for the calculation of the probability of reaching every edge $v \in V$ starting from s.

Notations Used

Notation 1 **($P[v \to w]$)** *The weight of the edge $(v \to w) \in E$. This is also the probability of following the edge $(v \to w)$ for the random walk from v.*

Notation 2 **($P[v]$)** *The total probability of reaching vertex v from the source s. It also denotes the entry for vertex v in the dynamic programming table.*

Preprocessing

We first obtain a total ordering of the given graph G by topologically sorting it. This can be done using the $\textsc{Topological-sort}(G)$ routine covered in class.
The sorting step can be achieved in $\Theta(|V| + |E|)$ time.

Reason for Sorting:

When G is sorted in the aforementioned fashion, then the vertices are ordered such that v appears before w if $(v \to w) \in E$. This allows us to calculate $P[w]$ by using the probabilities $P[v]\ \forall\ (v \to w) \in E$. This is possible because of equation (1) and the fact that the probabilities are mutually independent.

Decomposition into Sub-Problems

The given problem of finding $P[t_i]\ \forall\ 1 \leq i \leq k$ is broken down into overlapping sub-problems of the form $P[v]$, where v is not a sink. For each problem $P[v]$, we intend to find the probability of reaching v from s.
Each probability $P[v]$ can be found by adding the probabilities of reaching v from all vertices having an edge directed to v. The existence of $P[u]\ \forall\ (u \to v) \in E$ is guaranteed by the topological ordering.
Hence, this decomposition of the given problem into smaller sub-problems guarantees substructure optimalilty. This is because at any step, we can find the optimal solution using the optimzal solutions obtained in the previous steps.

Recurrence Relation

Let $P[v]$ be the total probability of reaching any vertex v from the source s. Then, the following recurrence relation can be defined:

$$P[v] = \begin{cases} 1 & \text{if } v = s \\ \sum\limits_{(u \to v) \in E} P[u] \times P[u \to v] & \text{otherwise} \end{cases} \tag{2}$$

Using the above recurrence relation, we can find $P[v]\ \forall\ v \in V$ and maintain it in a $|V| \times 1$ dynamic programming table.

Base Case and Solvable Sub-Problems

The base case for the above recurrence relation is $P[s] = 1$. This is because the probability of reaching s from the source is 1, as the random walk starts from s.

The sub-problems that solve the original problem are $P[t_i]\ \forall\ 1 \leq i \leq k$, the probabilities of reaching the sinks t_i from s.

The complete Algorithm and Pseudocode

The complete algorithm consists of the following steps:

1. Topologically sort the graph G to get a total ordering of V.
2. Initialize the dynamic programming table P following the base case.
3. Iterate over the vertices in their topological order and tabulate P using the above recurrence relation.
4. Return $P[t_1], P[t_2], \ldots, P[t_k]$, the total probabilities of reaching the sinks from s.

The pseudocode for the complete algorithm is given in Algorithm (1). [1]

Algorithm 1 An algorithm to find the probabilities of reaching the sinks from s.

```
procedure SINK-PROBABILITY(G = (V, E)):
    V_T = TOPOLOGICAL-SORT(G)
    P[v ∈ V_T] ← 0
    P[s] ← 1
    for v ∈ V_T do
        for (u → v) ∈ E do
            P[v] ← P[v] + P[u] × P[u → v]
        end for
    end for
    return P[t_1], P[t_2], ..., P[t_k]
end procedure
```

Runtime Analysis of the Algorithm

The following operations are performed in the algorithm:

1. TOPOLOGICAL-SORT(G): This step takes $\Theta(|V| + |E|)$ time.
2. Initialization of the dynamic programming table P, which takes $\Theta(|V|)$ time.
3. The dynamic programming step takes $\Theta(|V| + |E|)$ time, as it iterates over all vertices, while accessing the edges incident on each vertex. Analogous to a BFS traversal, the total number of iterations of the outer loop is $|V|$ and the inner loop is $|E|$.

Dominated by the sorting step and the dynamic programming step, the time complexity of the algorithm is $\Theta(|V| + |E|)$.

[1] The pseudocode of the algorithm makes use of notations introduced in the section **Notations Used**. Here, $P[v \in V_T]$ denotes a $|V_T| \times 1$ table ordered in the same way as V_T.

Bibliography

1. **OpenGenius IQ:** How to find cut-edges in a Graph
2. **USCB Math173A: (Lemma 1**) Cut-edges are not contained in cycles