Homework #6 Solutions

CS 344: Design and Analysis of Computer Algorithms (Fall 2022)

Problem 1

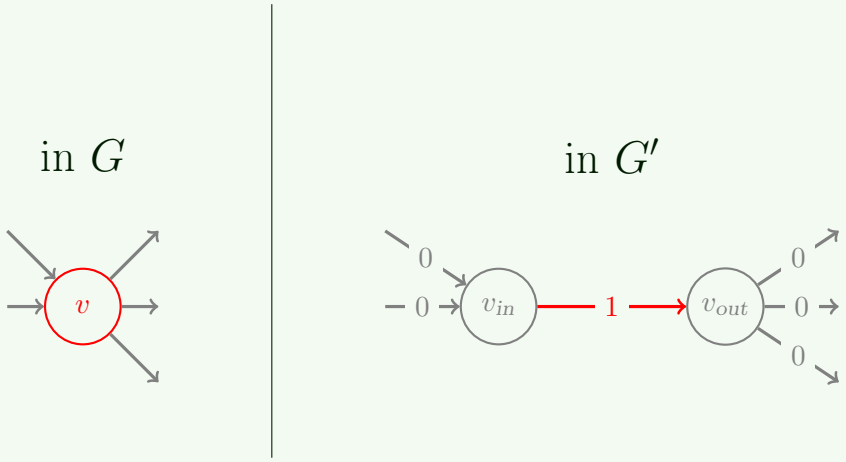
Consider the following problem:

- INPUT:
  - directed, *unweighted* graph $G = (V, E)$. Note that the graph is unweighted. Moreover, every vertex in the graph is either red or black. You can access the color of a vertex v in $O(1)$ time by looking at v.color. So for each vertex v, v.color is either "red" or "black"
  - Two vertices s, t.
- OUTPUT: output TRUE if there exists a path s to t that contains at most $\sqrt{|V|}$ red vertices. Output FALSE otherwise. Note that you don't need to output a path; you just need to determine if one exists.

Write pseudocode for an algorithm which solves the above problem in time $O(|E| \log(|V|))$. **For this problem you *must* use graph transformation.** That is, your algorithm should create a new graph G', and then run of an existing graph algorithm on G'; that is, the algorithm you run on G' should be one of the graph algorithms in our library, without any modifications to that algorithm.

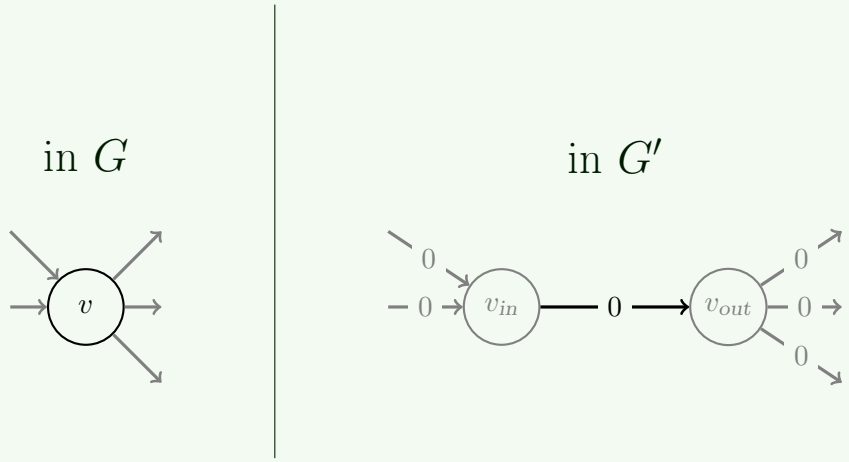
We take a graph transformation of G to G' (a weighted graph), and run DIJKSTRA on G'. The returned distance $d(s', t')$ will count the number of red vertices used on the path from s to t that minimizes this quantity. Hence, we output the truth-value of the logical expression $d(s', t') \leq \sqrt{|V|}$.

The details of the transformation are now explained. For every red vertex v, and edges incident to v, do the following where we split v into v_{in} and v_{out} and $w(v_{in}, v_{out}) = 1$:



Similarly, but with a different weighting scheme, for every black vertex v, and edges

incident to v, do the following where we split v into v_{in} and v_{out} and $w(v_{in}, v_{out}) = 0$:



Note that in both vertex transformations, the original graph edges are essentially the same (a directed edge $(u, v) \in E(G)$ will be sent to $(u_{out}, v_{in}) \in E(G')$, and the weight will be 0).

Pseudocode for the graph transformation follows:

```
TRANSFORM(G = (V, E)):
  V' ← {}
  E' ← {}
  for v ∈ V
      V' ← V' ∪ {v_in, v_out}
      E' ← E' ∪ {(v_in, v_out)}
      w(v_in, v_out) ← 1 if v.color = RED else 0
  for (u, v) ∈ E
      E' ← E' ∪ {(u_out, v_in)}
      w(u_out, v_in) ← 0
  return G' ← (V', E', w)
```

The algorithm is then:

```
PATHWITHFEWREDVERTICES(G = (V, E), s, t):
  G' ← TRANSFORM(G)
  d ← DIJKSTRA(G', s_in)
  return d(s_in, t_out) ≤ √|V|
```

Correctness. ($\Longrightarrow$) Suppose there is a path from s to t in G that contains at most $\sqrt{|V|}$ red vertices. Then expanding each vertex v in the path to v_{in}, v_{out} gives a path in G' from s_{in} to t_{out} that has length at most $\sqrt{|V|}$.

($\Longleftarrow$) Suppose conversely that there is a path from s_{in} to t_{out} in G' that has length at most $\sqrt{|V|}$. Observe that there is only one way to continue a path from an in-vertex: to go the the corresponding out-vertex. Moreover, every out-vertex only points to in-vertices. It then follows, by looking at in-out pairs in the path as a single vertex in G, that there is a path from s to t in G that contains at most $\sqrt{|V|}$ red vertices.

Time Complexity. Constructing G' takes $O(|E| + |V|)$ time. Note that $|V'| = O(|V|)$ and $|E'| = O(|E|+|V|)$. Running DIJKSTRA with a min-heap on G' then takes $O(|E'| \log(|V'|)) = O(|E| \log(|V|))$ since $|V| = O(|E|)$ (using the general assumption that $|E| \geq |V|/2$).

N.B. Transforming black vertices are not necessary. The arguments would have gone through had only red vertices been transformed, but we would have then had to then treat the vertices differently (e.g. if s was red, start from s_{in} and otherwise start from s).

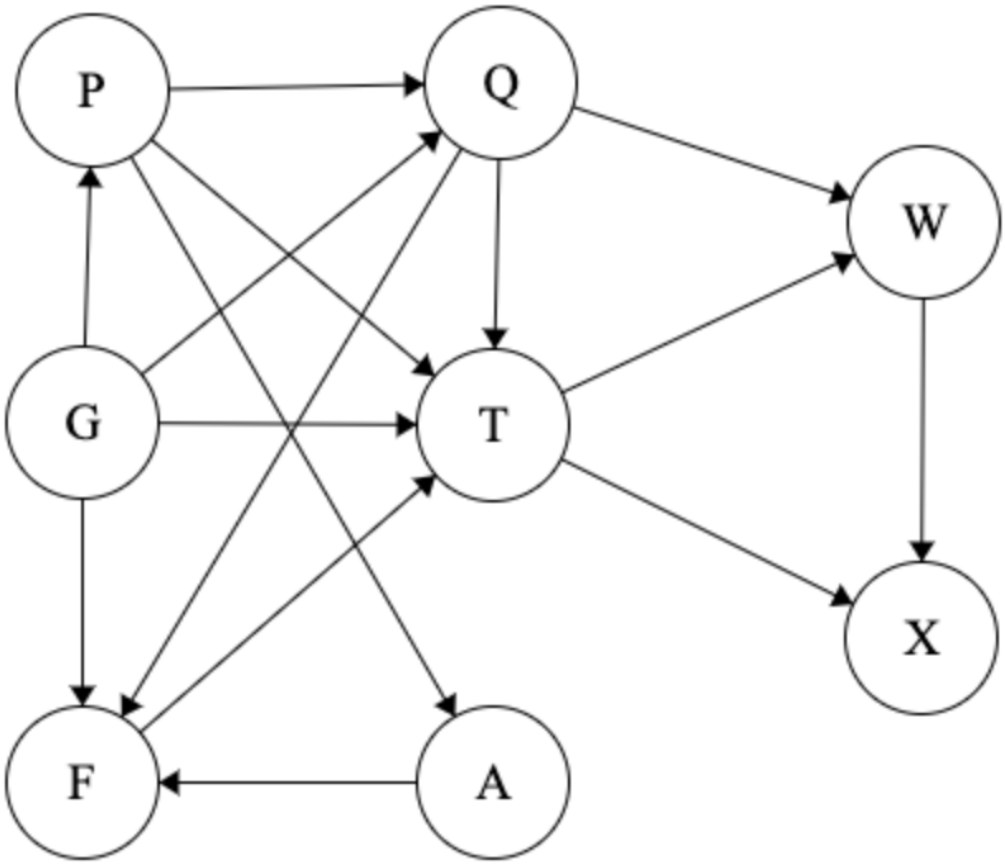


Figure 1: Graph for problem 2

Problem 2

Consider the DAG G in Figure 1. This graph has exactly two valid topological orderings. You should write down BOTH of them. No need to justify work. Your solution only needs to include the two orderings; no need for explanation or anything else.

(a) G, P, Q, A, F, T, W, X

(b) G, P, A, Q, F, T, W, X

Problem 3

Recall the topological ordering problem from class.

INPUT: a DAG $G = (V, E)$

OUTPUT: a topological ordering $v_1, ..., v_n$ of the vertices.

Write pseudocode for an algorithm that solves this problem in $O(|E|)$ time.

IMPORTANT NOTE: to keep notation consistent, please use the following notation for this problem. Assume that the original vertex set is $V = v'_1, ..., v'_n$, which are NOT in topological order. (I am using v'_i to avoid confusion with the v_i of a topological ordering.) Your goal is then to sort the vertices so they are in topological order. So for example, if the correct topological order is v'_2, v'_3, v'_1, then you should output the list v'_2, v'_3, v'_1. Alternatively, you could output

- $v_1 \leftarrow v'_2$
- $v_2 \leftarrow v'_3$
- $v_3 \leftarrow v'_1$,

You can use either of the two output methods described above.

```
TopologicalOrdering(G = (V, E)):
  //R stores processed vertices, in a topological ordering.
  R ← []
  //D maintains the current in-degree of unprocessed vertices.
  D[0..n − 1] ← in-degree(v'_i) in entry i, for all v'_i ∈ V
  Q ← BuildQueue()
  for i ← 0 to n − 1
      if D[i] = 0
          Q.push(i)
  while not Q.empty()
      i ← Q.pop()
      R.append(v'_i)
      for v'_j ∈ OUT(v'_i)
          D[j] ← D[j] − 1
          if D[j] = 0
              Q.push(j)
  return R
```

Correctness. This follows from noting that any time $v' \in V$ is added into R, everything pointing to v' has already been added into R and nothing v' points to has yet been added into R. A topological ordering is hence maintained in R.

Time Complexity. Initializing R, D, Q takes $O(|V|)$ time. The main loop takes $O(\sum_v \textit{out-degree}(v))$ time, which amounts to $O(|E|)$. The total running time is thus $O(|E|)$ using the general assumption for this problem set that $|E| \geq |V|/2$.

Problem 4

Consider the following problem:

- INPUT:
  - a *directed, weighted* graph $G = (V, E)$ where G is guaranteed to be a DAG. Moreover, every $v \in V$ has a color, which is either red or black. For any vertex v, you can access its color in $O(1)$ time by doing v.color. So for every v, v.color will be 'red' or 'black'.
  - two vertices $s, t \in V$.
- OUTPUT: find the length of shortest path from s to t such that the path contains at most 100 red vertices. That is, among all $s - t$ paths that contain at most 100 red vertices, you have to find the one with smallest total weight. Note that there is no restriction on the number of black vertices on the path.

Write pseudocode for an algorithm that solves the above problem in $O(|E|)$ time.

Once we order the vertices of G topologically, this problem becomes amenable to dynamic programming.

(1) Table values: $T[v][i]$ is the number of paths from s to v using exactly i red vertices.
T has dimensions $V \times [0..100]$.

(2) For $v \neq s$

$$T[v][i] = \begin{cases} \min\limits_{u:(u,v)\in E} (T[u][i] + w(u,v)) & \text{if } v.color = \textsc{Black}; 0 \leq i \leq 100 \\ \min\limits_{u:(u,v)\in E} (T[u][i-1] + w(u,v)) & \text{if } v.color = \textsc{Red}; 0 < i \leq 100 \end{cases}$$

(3) Initialize $T[s][0] = 0$ if $s.color = \textsc{Black}$ otherwise $T[s][1] = 0$.

(4) Return $\min\limits_{0\leq i\leq 100} (T[t][i])$.

(5) Finally, the pseudocode is as follows:

```
ShortestPathWithFewRedVertices(G = (V, E, w), s, t):
  Initialize T[V][0..100] ← ∞ in each entry
  Initialize T[s][0] ← 0 if s.color = Black otherwise T[s][1] = 0
  for v ∈ TopologicalSort(G)
      for u ∈ IN(v)
          if v.color = Black
              for i from 0 to 100
                  T[v][i] ← min(T[v][i], T[u][i] + w(u, v))
          else
              for i from 1 to 100
                  T[v][i] ← min(T[v][i], T[u][i − 1] + w(u, v))
  return min_{0≤i≤100} (T[t][i])
```

Time Complexity. Initializing T takes $O(|V|)$ time. The main loop takes $O(100 \cdot \sum_v \textit{in-degree}(v))$ time, which amounts to $O(|E|)$. Returning the answer takes $O(|V|)$ time. The total running time is thus $O(|E|)$ using the general assumption for this problem set that $|E| \geq |V|/2$.