CSE 5311 Homework 4 Solution

Problem 22.1-7

The **incidence matrix** of a directed graph $G = (V, E)$ with no self-loops is a $|V| \times |E|$ matrix $B = (b_{ij})$ such that

$$b_{ij} = \begin{cases} -1, & \text{if edge } j \text{ leaves vertex i,} \\ 1, & \text{if edge } j \text{ enter vertex i,} \\ 0, & \text{otherwise.} \end{cases}$$

Describe what the entries of the matrix product BB^T represent, where B^T is the transpose of B.

Answer

$$BB^T(i,j) = \sum_{e \in E} b_{ie} b^T_{ej} = \sum_{e \in E} b_{ie} b_{ej}$$

- If $i = j$, then $b_{ie}b_{je} = 1$ (it is $1 \cdot 1$ or $(-1) \cdot (-1)$ whenever e enters or leaves vertex i, and 0 otherwise.
- If $i \neq j$, then $b_{ie}b_{je} = 1$ when $e = (i, j)$ or $e = (j, i)$, and 0 otherwise.

Thus

$$BB^T_{ij} = \begin{cases} \text{degree of } i = \text{in-degree} + \text{out-degreee} \ , & \text{if } i = j, \\ -(\#\text{ of edges connecting } i \text{ and } j), & \text{if } i \neq j. \end{cases}$$

Problem 22.2-5

Argue that in a breadth-first search, the value $u.d$ assigned to a vertex u is independent of the order in which the vertices appear in each adjacency list. Using Figure 22.3 as an example, show that the breadth-first tree computed by BFS can depend on the ordering within adjacency lists.

Answer

The correctness proof for the BFS algorithm shows that $u.d = \delta(s, u)$, and the algorithm doesn't assume that the adjacency lists are in any particular order.

In Figure 22.3, if t precedes x in $Adj[w]$, we can get the breadth-first tree shown in the figure. But if x precedes t in $Adj[w]$ and u precedes y in $Adj[x]$, we can get edge (x, u) in the breadth-first tree.

Problem 22.2-7

There are two types of professional wrestlers: “babyfaces” (“good guys”) and “heels” (“bad guys”). Between any pair of professional wrestlers, there may or may not be a rivalry. Suppose we have n professional wrestlers and we have a list of r pairs of wrestlers for which there are rivalries. Give an $O(n+r)$-time algorithm that determines whether it is possible to designate some of the wrestlers as babyfaces and the remainder as heels such that each rivalry is between a babyface and a heel. If it is possible to perform such a designation, your algorithm should produce it.

Answer

Create a graph G where each vertex represents a wrestler and each edge represents a rivalry. The graph will contain n vertices and r edges.

Perform as many BFS’s as needed to visit all vertices. Assign all wrestlers whose distance is even to be babyfaces and all wrestlers whose distance is odd to be heels. Then check each edge to verify that it goes between a babyface and a heel. This solution would take $O(n+r)$ time for the BFS, $O(n)$ time to designate each wrestler as a babyface or heel, and $O(r)$ time to check edges, which is $O(n+r)$ time overall.

Problem 22.4-3

Give an algorithm that determines whether or not a given undirected graph $G = (V, E)$ contains a cycle. Your algorithm should run in $O(V)$ time, independent of $|E|$.

Answer

An undirected graph is acyclic (i.e., a forest) if and only if a DFS yields no back edges.

- If there’s a back edge, there’s a cycle.
- If there’s no back edge, then by Theorem 22.10, there are only tree edges. Hence, the graph is acyclic.

Thus, we can run DFS: if we find a back edge, there’s a cycle.

- Time: $O(V)$. (Not $O(V+E)$!) If we ever see $|V|$ distinct edges, we must have seen a back edge because (by Theorem B.2 on p. 1174) in an acyclic (undirected) forest, $|E| \leq |V| - 1$.

Problem 22-1 Classifying edges by breadth-first search

A depth-first forest classifies the edges of a graph into tree, back, forward, and cross edges. A breadth-first tree can also be used to classify the edges reachable from the source of the search into the same four categories.

a Prove that in a breadth-first search of an undirected graph, the following properties hold:

   1 There are no back edges and no forward edges.
   2 For each tree edge (u, v), we have $v.d = u.d + 1$.
   3 For each cross edge (u, v), we have $v.d = u.d$ or $v.d = u.d + 1$.

b Prove that in a breadth-first search of a directed graph, the following properties hold:

   1 There are no forward edges.
   2 For each tree edge (u, v), we have $v.d = u.d + 1$.
   3 For each tree edge (u, v), we have $v.d \leq u.d + 1$.
   4 For each tree edge (u, v), we have $0 \leq v.d \leq u.d$.

Answer

a
   1 Suppose (u, v) is a back edge or a forward edge in a BFS of an undirected graph. Then one of u and , say u, is a proper ancestor of the other (v) in the breadth-first tree. Since we explore all edges of u before exploring any edges of any of u's descendants, we must explore the edge (u, v) at the time we explore u. But then (u, v) must be a tree edge.
   2 In BFS, an edge (u, v) is a tree edge when we set $v.\pi = u$. But we only do so when we set $v.d = u.d + 1$. Since neither $u.d$ nor $v.d$ ever changes thereafter, we have $v.d = u.d + 1$ when BFS completes.
   3 Consider a cross edge (u, v) where, without loss of generality, u is visited before v. At the time we visit u, vertex must already be on the queue, for otherwise (u, v) would be a tree edge. Because v is on the queue, we have $v.d \leq u.d+1$ by Lemma 22.3. By Corollary 22.4, we have $v.d \geq u.d$. Thus, either $v.d = u.d$ or $v.d = u.d + 1$.

b
   1 Suppose (u, v) is a forward edge. Then we would have explored it while visiting u, and it would have been a tree edge.
   2 Same as for undirected graphs.
   3 For any edge $v.d \geq 0$, whether or not it's a cross edge, we cannot have $v.d > u.d + 1$, since we visit at the latest when we explore edge (u, v). Thus, $v.d \leq u.d + 1$.
   4 Clearly, $v.d \geq 0$ for all vertices v. For a back edge (u, v), v is an ancestor of u in the breadth-first tree, which means that $v.d \leq u.d$. (Note that since self-loops are considered to be back edges, we could have $u = v$.)

Problem 23.1-4

Give a simple example of a connected graph such that the set of edges $\{(u, v)$: there exists a cut $(S, V - S)$ such that (u, v) is a light edge crossing $(S, V - S)\}$ does not form a minimum spanning tree.

Answer

A triangle whose edge weights are all equal is a graph in which every edge is a light edge crossing some cut. But the triangle is cyclic, so it is not a minimum spanning tree.

Problem 23.1-10

Given a graph G and a minimum spanning tree T, suppose that we decrease the weight of one of the edges in T. Show that T is still a minimum spanning tree for G. More formally, let T be a minimum spanning tree for G with edge weights given by weight function w. Choose one edge $(x, y) \in T$ and a positive number k, and define the weight function w' by

$$w'(u,v) = \begin{cases} w(u,v), & \text{if } (u,v) \neq (x,y), \\ w(u,v) - k, & \text{if } (u,v) = (x,y). \end{cases}$$

Show that T is a minimum spanning tree for G with edge weights given by w'.

Answer

Let $w(T) = \sum_{(x,y)\in T} w(x,y)$. We have $w'(T) = w(T) - k$. Consider any other spanning tree T', so that $w(T) \leq w(T')$.

If $(x,y) \notin T'$, then $w'(T') = w(T') \geq w(T) > w'(T)$.

If $(x,y) \in T'$, then $w'(T) = w(T') - k \geq w(T) - k = w'(T)$.

Either way, $w'(T) \leq w'(T')$, and so T is a minimum spanning tree for weight function w'.

Problem 24.1-3

Given a weighted, directed graph $G = (V, E)$ with no negative-weight cycles, let m be the maximum over all vertices $v \in V$ of the minimum number of edges in a shortest path from the source s to v. (Here, the shortest path is by weight, not the number of edges.) Suggest a simple change to the Bellman-Ford algorithm that allows it to terminate in $m + 1$ passes, even if m is not known in advance.

Answer

If the greatest number of edges on any shortest path from the source is m, then the path-relaxation property tells us that after m iterations of BELLMAN-FORD, every vertex v has achieved its shortest-path weight in $v.d$. By the upper-bound property, after m iterations, no d values will ever change. Therefore, no d values will change in the $(m+1)$st iteration. Because we do not know m in advance, we cannot make the algorithm iterate exactly m times and then terminate. But if we just make the algorithm stop when nothing changes any more, it will stop after $m + 1$ iterations.

```
BELLMAN-FORD-(M+1)(G,w,s)
INITIALIZE-SINGLE-SOURCE(G,s)
changes = TRUE
```

```
while changes == TRUE
        changes = FALSE
        for each edge $(u,v) \in G.E$
                RELAX-M(u, v, w)

RELAX-M(u,v,w)
if v.d > u.d + w(u,v)
        $v.d = u.d + w(u,v)$
        $v.\pi = u$
        changes = TRUE
```

The test for a negative-weight cycle (based on there being a d value that would change if another relaxation step was done) has been removed above, because this version of the algorithm will never get out of the while loop unless all d values stop changing.

Problem 24.3-3

Suppose we change line 4 of Dijkstra's algorithm to the following.
4 **while** $|Q| > 1$
This change causes the **while** loop to execute $|V| - 1$ times instead of $|V|$ times. Is this proposed algorithm correct?

Answer

Yes, the algorithm still works. Let u be the leftover vertex that does not get extracted from the priority queue Q. If u is not reachable from s, then $u.d = \delta(s,u) = \infty$. If u is reachable from s, then there is a shortest path $p = s \to x \to u$. When the node x was extracted, $x.d = \delta(s,x)$ and then the edge (x,u) was relaxed; thus, $u.d = \delta(s,u)$.

Problem 24.3-4

Professor Gaedel has written a program that he claims implements Dijkstra's algorithm. The program produces $v.d$ and $v.\pi$ for each vertex $v \in V$. Give an $O(V + E)$-time algorithm to check the output of the professor's program. It should determine whether the d and attributes match those of some shortest-paths tree. You may assume that all edge weights are nonnegative.

Answer

1. Verify that $s.d = 0$ and $s.\pi = NIL$.
2. Verify that $v.d = v.\pi + w(v.\pi, v)$ for all $v \neq s$.
3. Verify that $v.d = \infty$ if and only if $v.\beta = NIL$ for $v \neq s$.
4. If any of the above verification tests fail, declare the output to be incorrect. Otherwise, run one pass of Bellman-Ford, i.e., relax each edge $(u,v) \in E$ one time. If any values of $v.d$ change, then declare the output to be incorrect; otherwise, declare the output to be correct.