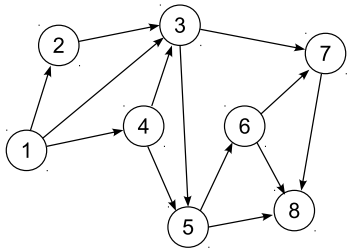
ECE-250 – Algorithms and Data Structures (Winter 2012)

Assignment 5 – Suggested solutions

1 – (30 pts) Given the following Directed Acyclic Graph (DAG), run a topological sort, showing the steps and the contents at each step of any arrays used.

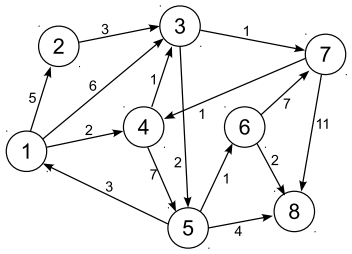


Solution:

The in-degrees are: $0, 1, 3, 1, 2, 1, 2, 3$. Thus, we can only start by vertex 1 (the only one with in-degree 0). We enqueue 1 and start the iterations: dequeue (and output) 1, decreasing the in-degrees for 2, 3, and 4. The in-degrees are now $0, 0, 2, 0, 2, 1, 2, 3$; thus, we enqueue 2 and 4 in any order (since they just reached in-degree 0). The next iteration dequeues 2 (assuming that we enqueued 2, then 4), decreasing the in-degree of 3; then, dequeue 4, decreasing the in-degrees of 3 and 5; in-degree of 3 reaches 0, so we enqueue it.

Etc.

2 – (30 pts) Run Dijkstra's shortest path algorithm to find the shortest path from vertex 1 to vertex 5 in the following graph. You must show the steps, and the contents at each step of any arrays used (including the pointers to "previous" node).



Solution:

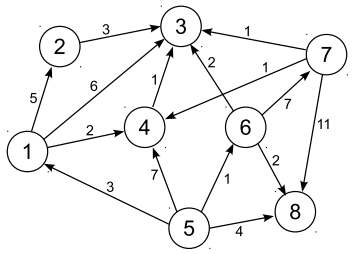
We set all vertices to unvisited, set all distances to ∞ except for vertex 1, with distance 0, and set all pointers to NULL. At the first iteration, we visit vertex 1 (the only one with non-∞ distance), so we mark vertices 2 (distance 5, pointing back to 1), 3 (distance 6, pointing back to 1), and 4 (distance 2, pointing back to 1). At the next iteration, we visit 4 (the smallest distance among the unvisited

vertices), so we mark its neighbors; 3 already has a distance, but the distance through vertex 4 is smaller (dist(4) is 2, added with the weight of the edge from 4 to 3, we get $2+1 > 6$), so we update vertex 3, with distance 3, pointing back to 4. Vertex 5 is marked with distance 9 ($2+7$). At the next iteration, we visit 3 (smallest distance among unvisited vertices), so we mark vertex 7 with distance $3+1=4$, pointing back to 3, and we update vertex 5 (it has distance 9, but the distance going through vertex 3 is smaller; $3+2=5<9$), with distance 5 and pointing back to 3.

Etc. (after a couple more iterations, we visit 5 and the algorithm ends, finding the shortest path to be $(1,4,3,5)$, with length 5)

Notice that at one iteration, we have the the choice between vertex 2 (distance 5) and vertex 5 (distance 5). The choice here is arbitrary — we can choose either one, and things will work either way.

3 – (30 pts) Run Dijkstra's shortest path algorithm to find the shortest path from vertex 1 to vertex 5 in the following graph, to show that there is no such path. Explain the stop condition for the algorithm and how it reveals that no such path exists.



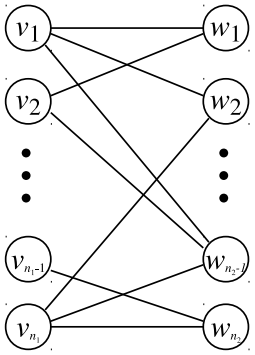
Solution:

The key detail here is that if at some iteration, we have not yet reached the target vertex and all the unvisited vertices have ∞ distance, then we know that no path exists to the target vertex (or any of the unvisited vertices, for that matter). This is clearly the case, since the "marked" vertices (with a non-∞ distance) indicate that a path to them has been found (not necessarily the shortest); if at some point no path has been found to any of the possible candidates to continue expanding the set of paths, then the condition indicates that we have exhausted all the paths available from the source vertex.

4 – (10 pts) We recall that a Hamiltonian cycle in a graph is a simple path that is a cycle (that is, a cycle that visits each vertex exactly once, except for the first and last vertex, which are the same vertex since it is a cycle).

We have an undirected graph where the vertices can be grouped into two sets $\mathcal{V} = \{v_1, v_2, \cdots, v_{n_1}\}$ and $\mathcal{W} = \{w_1, w_2, \cdots, w_{n_2}\}$ and every edge has the form (v_i, w_j) or (w_i, v_j); that is, every edge in the graph can only associate a vertex from $\mathcal{V}$ with a vertex from $\mathcal{W}$.

The figure below shows an example of such a graph (you can think of a graph where we have vertices at the left and vertices at the right, and edges always go between a vertex in the left group and a vertex in the right group).



Prove that a graph with the above characteristic and an odd number of vertices can not have a Hamiltonian cycle.

Solution:

Proof by induction:

Base case: graph with three vertices. Two possible cases: either the three vertices are on the same set (either all in $\mathcal{V}$ or all in $\mathcal{W}$), in which case clearly there can not be a Hamiltonian cycle, since there can not be edges at all; or we have two vertices in one set and one in the other set, in which case clearly there can not be a Hamiltonian cycle (there can only be two edges in this case; not enough to complete a simple cycle).

Induction step: Here, we need to prove that the statement being true for n vertices implies that the statement holds for $n + 2$ vertices. We could show the contrapositive, which is a strictly equivalent statement: if a bipartite graph with $n + 2$ (n odd) can have a Hamiltonian cycle, then a bipartite graph with n vertices can also have one.

Notice that this is essentially the same as arguing by contradiction: the induction hypothesis is that a graph with n vertices (n odd) can not have a Hamiltonian cycle, and we have to show that this implies that one with $n + 2$ vertices can not have a Hamiltonian cycle either. So, we assume (for a contradiction) that one with $n + 2$ can, and try to reach a contradiction (the contradiction being that we find a graph of n vertices with a Hamiltonian cycle, which contradicts the induction hypothesis).

The construction is as follows: Let G be a bipartite graph with $n + 2$ vertices (n odd), and let $H = \{v_1, v_2, \ \cdots \ v_{n+1}\}$ be a Hamiltonian cycle. Consider the sequence $v_{k-1}, v_k, v_{k+1}, v_{k+2}$ of consecutive vertices in the Hamiltonian cycle. Clearly, v_{k-1} is in one side, v_{k+2} is on the other side (since these four vertices necessarily alternate sides if they're part of a path). Thus, if we remove v_k and v_{k+1}, we can connect v_{k-1} and v_{k+2}, and because all of the remaining vertices were part of a Hamiltonian cycle in the original graph, then we must necessarily have a Hamiltonian cycle after we remove v_k and v_{k+1} and connect v_{k-1} to v_{k+2}, resulting in a bipartite graph of n vertices with a Hamiltonian cycle.

10% Bonus Marks – The subset sum problem is known to be NP-complete. However, in this problem we work with an array of arbitrary numbers (in particular, unsorted values). Suppose that someone attempted to solve the problem efficiently if the input values are in order.

Prove that this problem (subset sum with the input values in ascending order) is also NP-complete.

Solution:

Proving that a problem $\mathcal{P}$ is NP-complete only requires finding a polynomial time reduction from some other NP-complete problem to $\mathcal{P}$.

In this case, the reduction is straightforward — let $\mathcal{A}$ be an algorithm that solves the problem of subset sum with input values in ascending order. To solve the subset sum problem, we sort the values in $\Theta(n \log n)$ (or in $\Theta(n^2)$, for that matter), and feed the sorted values to algorithm $\mathcal{A}$.

This reduction is clearly polynomial time, proving that the problem is NP-complete.

The intuition behind this "trivial" exercise is that if a problem is NP-complete, we can not possibly expect that a relatively simple transformation of the input will change the difficult nature of the problem — if having sorted input values helped in finding a subset sum solution, then the subset sum problem *would not* be a difficult problem at all!