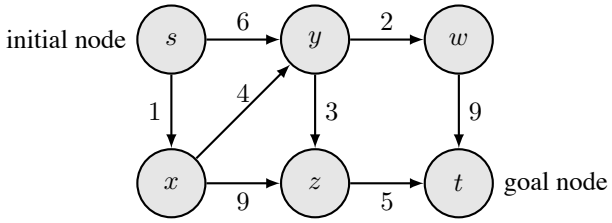
3010 Artificial Intelligence — Assignment 1
Due: Tuesday, May 21, 2019

Write a report answering Questions 1–4 (Note: questions continue to the back of the page). Submit the report in the drop-in box in front of Information Science Administration Office, by no later than May 21.

Question 1

Consider the state space graph shown below. This graph has six nodes $\{s, t, w, x, y, z\}$, among which s is the initial node and t is the goal node. The digit next to each arc represents the cost of the arc. For instance, the cost of arc (x, z) is $c(x, z) = 9$. Answer the following questions.



1. Find the cost of the cheapest path from ***each*** of the six nodes to the goal node t.
2. Suppose we run on this graph Dijkstra's shortest-path algorithm shown in Figure 1. In each iteration of lines 6–11 of function Dijkstra (in Figure 1),
   - show which nodes are in OPEN and what are their g-values when line 7 is executed, and
   - show which node is chosen as v on line 8.

```
1  function Dijkstra(s)
2      OPEN ← new PriorityQueue_g
3      g[s] ← 0
4      Insert_g(OPEN, s)
5      CLOSED ← ∅
6      loop do
7          if IsEmpty(OPEN) then return "failure"
8          v ← DeleteMin_g(OPEN)
9          CLOSED ← CLOSED ∪ {v}
10         if IsGoal(v) then return Solution(v, s)
11         Expand(v)
```

```
1  procedure Expand(v)
2      foreach u ∈ Succ(v) do
3          if u ∉ OPEN ∪ CLOSED then
4              Parent[u] ← v
5              g[u] ← g[v] + c(v, u)
6              Insert_g(OPEN, u)
7          else if u ∈ OPEN then
8              if g[v] + c(v, u) < g[u] then
9                  Parent[u] ← v
10                 g[u] ← g[v] + c(v, u)
```

Figure 1: Dijkstra's shortest path algorithm. OPEN, CLOSED, Parent, and g are global variables. See the lecture slides for other details.

Answer

1. See the table below:

node	s	x	y	z	w	t
cheapest path cost to t	13	12	8	5	9	0

2. See the table below—g-values for closed nodes are crossed out.

Iteration	OPEN nodes	$g[s]$	$g[x]$	$g[y]$	$g[z]$	$g[w]$	$g[t]$	node chosen as v
1	s	0						s
2	x, y	~~0~~	1	6				x
3	y, z	~~0~~	~~1~~	5	10			y
4	z, w	~~0~~	~~1~~	~~5~~	8	7		w
5	z, t	~~0~~	~~1~~	~~5~~	8	~~7~~	16	z
6	t	~~0~~	~~1~~	~~5~~	~~8~~	~~7~~	13	t

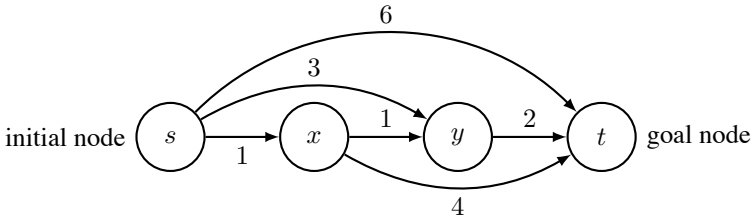
Question 2

Draw a state space graph G that satisfies all of the following conditions:

1. G has four nodes $\{s, t, x, y\}$, where s is the initial node and t is the (only) goal node;
2. G has six edges;
3. all edge costs are positive integers less than or equal to 6; and
4. when Dijkstra's shortest-path algorithm (of Figure 1) is run on G, the g value for one node is changed twice (i.e., line 10 of procedure Expand is executed twice).

Answer

Many graphs are possible, but the following reasoning applies regarless. For a node to have its g-value updated twice on line 10, it must have three incoming edges—relaxing one of them have the node generated, and relaxing the other two makes the node's g-value updated. And the expansion of that node must be preceded by the expansion of the three parent nodes of these edges. Since we have only four nodes, that node with three incoming edges must be t. The following is one such graph.



The g-value of t changes from 6 to 5, and then finally to 4.

Question 3

Let v_j be the jth node closed (i.e., placed in the CLOSED set) during a run of Dijkstra's algorithm, $j = 1, 2, \ldots$. (Because the first node closed by the algorithm is the initial node s, $v_1 = s$). For any node v, let $g^*(v)$ denote the cost of the chepeast path from the initial node s to v. Prove that $g^*(v_j)$ is nondecreasing over j; that is, $g^*(v_1) \leq g^*(v_2) \leq g^*(v_3) \leq \cdots$.

Answer

Proof (A) Assume $g^*(v_1) \leq \cdots \leq g^*(v_{k-1})$. We prove $g^*(v_{k-1}) \leq g^*(v_k)$ also holds. First, note that from the time when v_j ($j = 1, \ldots, k$) is closed, relation

$$g[v_j] = g^*(v_j) \quad \text{for } j = 1, 2, \ldots, k. \tag{1}$$

holds by the property of Dijkstra's algorithm. Now, two cases are possible, depending on the value of $\mathrm{Parent}[v_k]$ when v_k is closed:

- If $\mathrm{Parent}[v_k] = v_{k-1}$, it means that v_{k-1} is the parent of a cheapest path to v_k from s, and thus $g[v_k] = g^*(v_k) = g^*(v_{k-1}) + c(v_{k-1}, v_k)$. Since all edge costs are greater than 0, it follows that $g^*(v_k) > g^*(v_{k-1})$
- If $\mathrm{Parent}[v_k] \neq v_{k-1}$, it means that when v_{k-1} was closed, v_k must have been in OPEN with $g[v_k] = g^*(v_k)$ already. But since v_{k-1} is chosen as the node to close instead of v_k, it follows that $g[v_{k-1}] \leq g[(v_k]$. And since $g[v_{k-1}] = g^*(v_{k-1})$ and $g[(v_k] = g^*(v_k)$, we have $g^*(v_{k-1}) \leq g^*((v_k)$.

In either case, we have $g^*(v_{k-1}) \leq g^*(v_k)$.

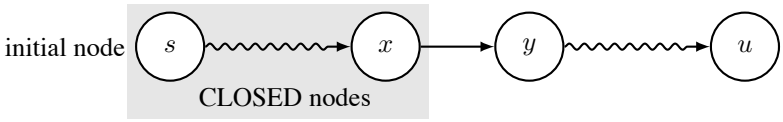
Proof (B) (Proof by contradiction) Suppose on the contrary that Dijkstra's algorithm closes a node v before u despite

$$g^*(u) < g^*(v). \tag{2}$$

Consider the time at which v is about to be closed. Let y be the first non-closed node along a cheapest path p from s to u. Since y precedes u along p,

$$g^*(y) < g^*(u). \tag{3}$$

Node y has a parent node along p, since y cannot be the initial node s (as it is closed in the first step of Dijkstra's algorithm). Let this parent node be x. The nodes along path p are depicted as follows:



Now, node y being the first non-closed node in p implies x is on CLOSED, which means

$$g[x] = g^*(x).$$

And when x was closed, edge (x, y) must have been relaxed. Hence

$$g[y] = g^*(x) + c(x, y) = g^*(y).$$

Now, node y must be on OPEN (because its parent x was expanded and y has ***not*** been closed). And the algorithm is about to choose v over y as the node to close. Thus,

$$g^*(v) = g[v] \leq g[y] = g^*(y).$$

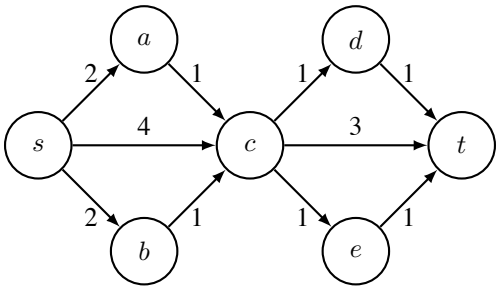
Combining this with Equation (3), we have

$$g^*(v) < g^*(u),$$

which contradicts the assumption $g^*(u) < g^*(v)$ given by Equation (2).

Question 4

Consider a finite state space graph with (only) one goal node. Modify Dijkstra's algorithm so that when run on such a graph, it returns the ***number*** of cheapest paths from the initial node to the goal node, instead of a cheapest path to the node.
For example, in the following state space graph with the initial node s and the goal node t, the modified algorithm must return 4, because there are four cheapest paths (all with cost 5), namely, (1) $s \to a \to c \to d \to t$, (2) $s \to a \to c \to e \to t$, (3) $s \to b \to c \to d \to t$, and (4) $s \to b \to c \to e \to t$. Note: cheapest paths need not be enumerated; only their number (i.e., "4" in this example) must be output.



Answer

Figure 2 shows a possible modification to Dijkstra's algorithm, with changes marked by red lines. The idea is to take advantage of the property of Dijkstra's algorithm: For any node v and any cheapest path to that node from s, all predecessors along the path are in CLOSED (with the correct g-value).

- Since we are not interested in a specific path in the graph, there is no need to maintain $\mathrm{Parent}[\cdot]$. (see Function Dijkstra in Figure 2, line 4)
- Instead, we maintain the number of cheapest paths to each node v in $p[v]$. (Function Dijkstra in Figure 2, line 11)
- The main function is changed so that it returns the $p[t]$ of the goal node t. (Function Dijkstra, line 11)
- At the initial node s, we initialize by setting $p[s] = 1$. This is because there is a single cheapest path from s to s, i.e., path with length 0, i.e., (s).
- When edge (v, u) is "relaxed" and a node u is generated, set $p[u] \leftarrow p[v]$. That is, the number of cheapest paths to node u is exactly the same as that of its parent v. Their numbers are identical, because for each cheapest path to v, it can be exteded by edge (v, u) to obtain the cheapest path to u. (Procedure Expand, line 4)
- When a cheaper path is later found (through a new parent v), we need to reset $p[u]$ to the number of cheapest paths through the new parent; $p[u] \leftarrow p[v]$. (Procedure Expand, line 9)
- When there is a path through a new parent giving the same cheapest path cost as we already have, the number of the path through the new parent must be added to $p[u]$. (Procedure Expand, lines 11–12)

```
1  function Dijkstra(s)
2      OPEN ← new PriorityQueue_g
3      g[s] ← 0
4      p[s] ← 1
5      Insert_g(OPEN, s)
6      CLOSED ← ∅
7      loop do
8          if IsEmpty(OPEN) then return "failure"
9          v ← DeleteMin_g(OPEN)
10         CLOSED ← CLOSED ∪ {v}
11         if IsGoal(v) then return p[v]
12         Expand(v)
```

```
1  procedure Expand(v)
2      foreach u ∈ Succ(v) do
3          if u ∉ OPEN ∪ CLOSED then
4              p[u] ← p[v]
5              g[u] ← g[v] + c(v, u)
6              Insert_g(OPEN, u)
7          else if u ∈ OPEN then
8              if g[v] + c(v, u) < g[u] then
9                  p[u] ← p[v]
10                 g[u] ← g[v] + c(v, u)
11             else if g[v] + c(v, u) = g[u] then
12                 p[u] ← p[u] + p[v]
```

Figure 2: Modified Dijkstra's algorithm to compute the number of shortest paths to the goal state.

Remark. Similar computation is utilized in Brandes's algorithm for computing the "betweenness"' centrality of nodes in a graph; see Fouss, Saerens, Shimbo: "Algorithms and Models for Network Data and Link Analysis", Cambridge University Press, 2016, Chapter 4.