Missouri University of Science & Technology	Department of Computer Science
Spring 2024	**CS 2500: Algorithms (Sec: 102)**

Solutions to HW 4: Greedy Algorithms

Instructor: *Sid Nadendla* **Due:** *April 12, 2024*

Problem 1 Greedy Scheduling *3 points*

Scheduling is a problem where the goal is to permute a set of n tasks, where each i^{th} task is prescribed with a length ℓ_i (which represents the time taken to complete this task) and a priority weight w_i (where higher weights represent higher priority).

(a) Implement your own class called `Tasks` which represents a data structure that stores all the jobs J, along with their respective lengths ℓ and weights w.

(b) Within the `Tasks` class, implement the `GreedyRatio`$(self)$ subroutine in this class to find the optimal schedule that minimizes the sum of weighted completion times.

(c) Perform the empirical runtime analysis of your `GreedyRatio`$(self)$ subroutine by simulating multiple `Tasks` objects with randomly generating n tasks (Let $n = 5 : 5 : 101$), each having a uniformly random length ℓ_i using the statement `randrange(5, 51, 5)`, and a uniformly random weight w_i using the statement `randrange(1, n+1, 1)`.

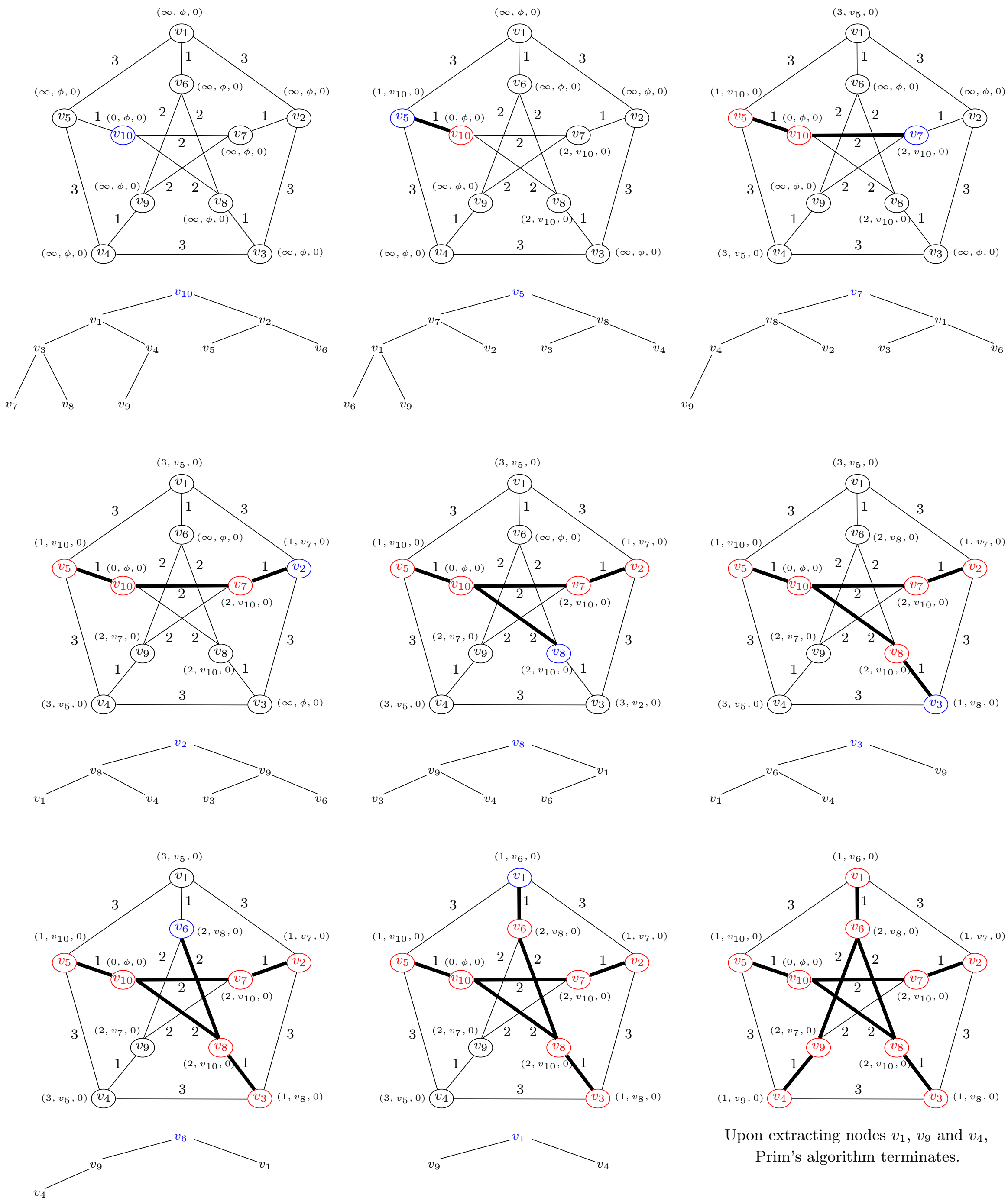
Solution: This is a programming exercise. Graders will provide feedback directly in the Gitlab repositories.

Problem 2 Prim's Min. Spanning Trees Algorithm *3 points*

(a) Demonstrate Prim's algorithm (with vertex v_{10} as the start node) for the Petersen graph shown in Figure 1. (*1.5 points*)

Solution:

Min. priority queue implemented using min. heaps.
Node Attributes: $(key, parent, adj\_count)$



(b) Prove the correctness of Prim's algorithm formally, i.e. show that Prim's algorithm always returns the minimum spanning tree of any given connected graph and some start node within the graph. (*2.5 points*)

Solution:

Since Prim's algorithm grows the tree by adding one node at a time, consider the following loop invariant:

Loop Invariant: At the end of k^{th} iteration, Prim's algorithm produces a minimum spanning tree (MST) on the subgraph G_k induced by the frontier X_k on G. Note that the frontier X_k exactly contains k vertices in G.

***Proof by induction*:**

Initialization (or Base Case)

During the initialization, the frontier X_0 contains exactly one node, which is the start node. Then, the subgraph G_0 induced by X_0 on the graph G contains just one node (i.e. the start node), the spanning tree is empty as there are no edges to add. This is consistent with the initialization of $T_0 = \emptyset$.

Maintainence (or Inductive Step)

Assume that the tree T_{K-1} is a MST on the subgraph G_{K-1} induced by the frontier X_{K-1} on the graph G at the end of $(K-1)^{th}$ iteration. During the K^{th} iteration, assume that an edge (a, b) is included in the tree. In other words, the node $a \in X_{K-1}$ and $b \notin X_{K-1}$.

Consider a violation of the loop invariant in the K^{th} iteration. In other words, T_K is not a MST on the subgraph G_K induced by the frontier X_K on the graph G at the end of K^{th} iteration. Since T_{K-1} is a MST on G_{K-1}, $X_K - X_{K-1} = \{b\}$ and $T_K - T_{K-1} = \{(a, b)\}$, T_K is a spanning tree on G_K. Therefore, T_K is not a MST.

In such a case, there exists another edge (c, d) with $c \in X_{K-1}$ and $d \notin X_{K-1}$ such that $w(c, d) < w(a, b)$. However, if such an edge (c, d) exists, the edge (a, b) cannot be the one with minimum weight amongst the set of edges that are incident to the frontier X_{K-1}. Therefore, we have a contradiction.

In other words, T_K is also a MST on the subgraph G_K induced by the frontier X_K on G.

Termination

When the Prim's algorithm terminates, if G is connected, Prim's algorithm returns a MST. On the contrary, if G is not connected, then the frontier X does not contain $|V|$ vertices since the algorithm terminates in fewer than $|V|$ number of iterations.

Therefore, by the principle of mathematical induction, Prim's algorithm always returns a minimum spanning tree of any connected graph G. □

Problem 3 Speeding up Dijkstra's Algorithm *4 points*

(a) Design Dijkstra's shortest path algorithm using `minheap` data structure and write its pseudocode. Evaluate its asymptotic runtime theoretically. (*1 point*)

Solution: Since Dijkstra's algorithm updates the distance estimates of all the nodes incident to the frontier S and updates S by including a node with minimum distance estimate in each iteration, it is natural to maintain all the vertices on a min. priority queue with distance estimates as keys. Min. priority queues can be implemented efficiently using `minheap`. Thus, we have the following implementation of Dijkstra's algorithm.

DIJKSTRA(G, s)

1 **for** each vertex $v \in G.V$
2     $v.d = \infty$ // Distance estimate from s to v is initialized to ∞.
3     $v.p = \emptyset$ // Parent of v is initialized to `None`.
4 $s.d = 0$ // Distance estimate from s to s is zero.
5 $S = \emptyset$ // S is the frontier.
6 $Q = G.V$ // Let Q be a minimum priority queue implemented using a `minheap`.
7 **while** $Q \neq \emptyset$
8     $u =$ EXTRACT-MIN(Q) // Find the node with min. distance estimate in $V - S$.
9     $S = S \cup \{u\}$
10   **for** each vertex $v \in G.Adj[u]$
11   **if** $v.d > u.d + w(u, v)$ // Update if the path from $s \rightsquigarrow u \rightarrow v$ is shorter.
12             $v.d = u.d + w(u, v)$
13             $v.parent = u$

***Asymptotic Runtime Analysis*:**

The runtime due to for-loop in Lines 1-3 is $O\big(|V|\big)$. Lines 4-6 is a constant effort, i.e. $O\big(1\big)$, which is dominated by Lines 1-3 and Line 6.

The while loop in Line 7 iterates at most $|V|$ times, in the case of any connected graph. Line 8 takes a $O\big(\log |V|\big)$ runtime, since the EXTRACT-MIN function uses MIN-HEAPIFY subroutine to maintain the min. heap property. This MIN-HEAPIFY subroutine runs for at most the height of `minheap` per call, which is given by $\log |V|$. Therefore, the runtime due to $|V|$ calls of EXTRACT-MIN is $O\big(|V| \log |V|\big)$.

Let D_v denote the out-degree of a vertex $v \in V$. Then, the for loop in Lines 10-13 will run for a total of $\sum\limits_{v \in V} D_v = |E|$ on a connected graph. However, each time the distance estimate is updated, the `minheap` automatically runs the MIN-HEAPIFY subroutine, which takes at most $O(\log |V|)$ time. Therefore, the total runtime of Lines 10-13 is $O\big(|E| \log |V|\big)$.

Combining all of the above, the asymptotic runtime for the above algorithm is given by

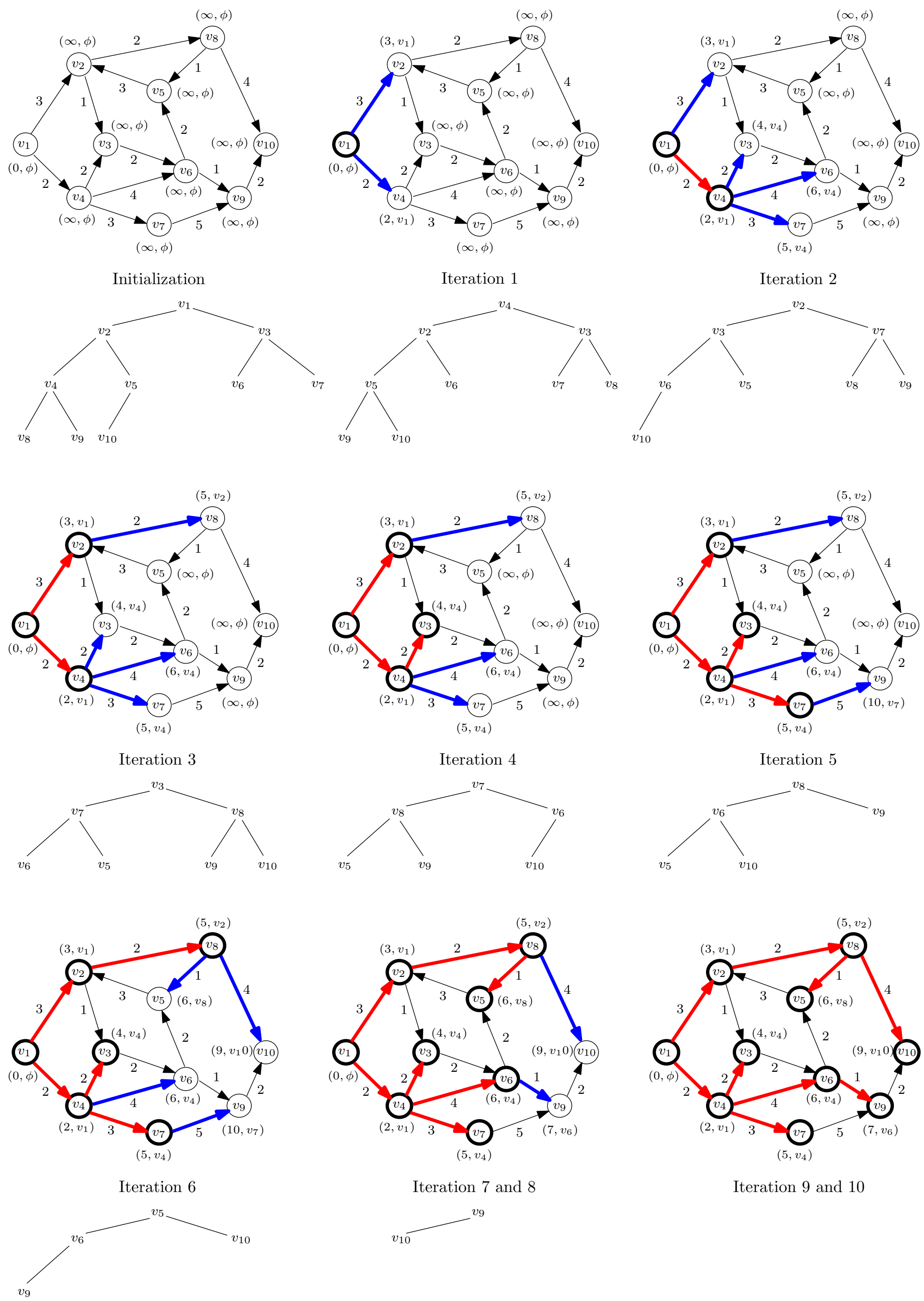
$$O\Big(|V| \log |V| + |E| \log |V|\Big) = O\Big(|E| \log |V|\Big).$$

(b) Demonstrate the above Dijsktra's shortest path algorithm (with "1" as the start node) on the unweighted, undirected graph shown in Figure 2. Clearly show how each node-attributes (i.e. distance estimate and parent) as well as the `minheap` data structure changes in each iteration in both the algorithms. (*1 point*)

Solution:

Data Structure: Min. Priority Queue implemented using min-heaps

Node Attributes: ($distance\_estimate, parent$)



(c) Implement your pseudocode in Python as a `Dijkstra`$(self, start\_vertex)$ subroutine in the `Graph` class built using adjacency list representation (similar to the one in HW3), and validate your implementation on the same example graph shown in Figure 2 by comparing its output against your answer in Problem 3(b). (*2 points*)

Solution: This is a programming exercise. Graders will provide feedback directly in the Gitlab repositories.