Ph.D. Qualifying Exam: Data Structures and Algorithms

This is a closed book exam. The total score is 100 points. Please answer all questions.

1. In the standard merge-sort algorithm, an input array is split into two (about) equal halves; the algorithm is recursively called on each part; finally the two sorted halves are merged into one sorted array and returned. (DPV 2.3)

   In a three-way merge-sort, the input array is cut into three (about) equal portions instead. Please answer the following questions:

(10 points) (a) Write pseudocode to merge three sorted arrays. This can be a revision or reuse of the two-way merge algorithm given below.

---

function merge($x[1 \ldots k], y[1 \ldots l]$)
if $k == 0$: return $y[1 \ldots l]$
if $l == 0$: return $x[1 \ldots k]$
if $x[1] < y[1]$:
  return $x[1] \circ$ merge($x[2 \ldots k], y[1 \ldots l]$)
else:
  return $y[1] \circ$ merge($x[1 \ldots k], y[2 \ldots l]$)

Solution:

Option 1. By revision and reuse of the merge() function:

---

function three-part-merge($x[1 \ldots k], y[1 \ldots l], z[1 \ldots m]$)
if $m == 0$: return merge(x, y)
if $l == 0$: return merge(x, z)
if $k == 0$: return merge(y, z)
if $x[1] < y[1]$ and $x[1] < z[1]$:
  return $x[1] \circ$ three-part-merge($x[2 \ldots k], y, z$)
else if $y[1] < x[1]$ and $y[1] < z[1]$:
  return $y[1] \circ$ three-part-merge($x, y[2 \ldots l], z$)
else
  return $z[1] \circ$ three-part-merge($x, y, z[2 \ldots m]$)

Option 2: By reuse of the merge() function:

---

function three-part-merge($x[1 \ldots k], y[1 \ldots l], z[1 \ldots m]$)
u = merge(x, y)
return merge(u, z)

(10 points) (b) Integrating your three-part-merge function, give the pseudocode for an algorithm to do the three-part merge-sort by divide-and-conquer. Make sure to include the base case.

Solution:

---

function three-part-merge-sort($a[1 \dots n]$)
Input: An array of numbers $a[1 \dots n]$
Output: A sorted version of this array

if $n \leq 1$:
  return a
else:
  return three-part-merge(three-part-merge-sort($a[1 \dots \lfloor n/3 \rfloor]$),
    three-part-merge-sort($a[\lfloor n/3 \rfloor + 1 \dots \lfloor 2n/3 \rfloor]$),
    three-part-merge-sort($a[\lfloor 2n/3 \rfloor + 1 \dots n]$))

(10 points) (c) What is a tight asymptotic runtime of your three-part merge-sort algorithm above? Please include the recursive equation for runtime in your derivation.

Solution: The three-way-merge algorithm reduces the problem size by one in constant time, so we have

$$T_1(n) = T_1(n-1) + 1 = \Theta(n) \tag{1}$$

The three-way-merge-sort has the following recursive equation:

$$T(n) = 3T(n/3) + T_1(n) = 3T(n/3) + n = \Theta(n \log n) \tag{2}$$

based on the Master's theorem.

2. Describe a strategy to check whether there is a cycle in a directed graph $G = (V, E)$ in linear time in the number of edges and the number of nodes in the graph. (DPV 3.2, 4.2)

(10 points) (a) Give the pseudocode of your strategy.

Solution:

---

function contain-cycle($G = (V, E)$)
Input: A graph $G = (V, E)$
Output: Whether the graph contains a cycle

Perform DFS on the graph
During the graph traversal, if a back edge exists (pointing from a current node to
  an ancestor node of the DFS tree), return YES
return NO

(10 points) (b) Explain why the algorithm takes linear time.

Solution: The runtime is the same as DFS or BFS $O(|V| + |E|)$.

3. The following code solves the shortest path problem on a directed acyclic graph $G = (V, E)$. (DPV 6.1)

---

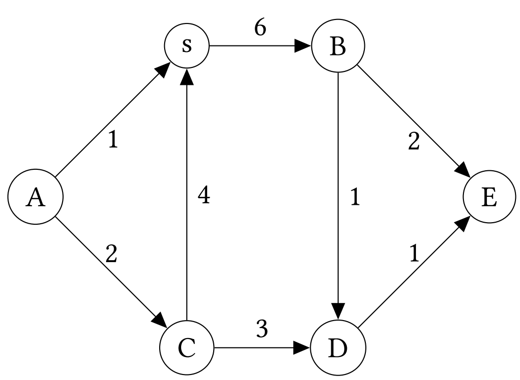
function shortest-path-1(G, s)
Input: A graph $G = (V, E)$, a start node s
initialize all dist() values to ∞
dist(s)=0
for each $v \in V - \{s\}$, in linearized order:
    $dist(v) = \min_{(u,v) \in E} dist(u) + l(u, v)$

A student changes the code to the following

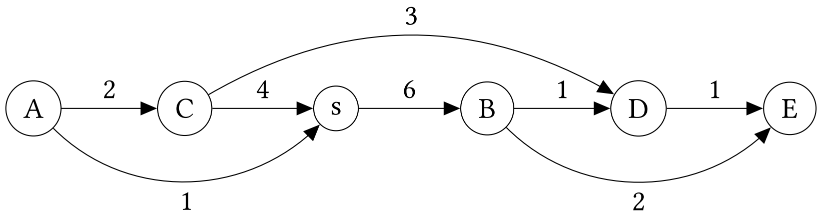
---

function shortest-path-2(G, s)
Input: A graph $G = (V, E)$, a start node s
initialize all dist() values to ∞
dist(s)=0
for each $u \in V$, in linearized order:
    for each edge (u, v):
        $dist(v) = \min\{dist(v), dist(u) + l(u, v)\}$

(10 points) (a) Show how algorithm shortest-path-1 works on the following graph. The source node is s. Please show the distances of all nodes after processing each v.



Solution:



	s	A	B	C	D	E
A	0	∞	∞	∞	∞	∞
C	0	∞	∞	∞	∞	∞
B	0	∞	6	∞	∞	∞
D	0	∞	6	∞	7	∞
E	0	∞	6	∞	7	8

(10 points) (b) Show how algorithm shortest-path-2 works on the above graph starting from s. Please show the distances of all nodes after processing each u.

Solution: We first obtain the topological ordering by decreasing order of DFS post-number (as the same shown in the solution above)

	s	A	B	C	D	E
A	0	∞	∞	∞	∞	∞
C	0	∞	∞	∞	∞	∞
s	0	∞	6	∞	∞	∞
B	0	∞	6	∞	7	8
D	0	∞	6	∞	7	8
E	0	∞	6	∞	7	8

(10 points) (c) Does shortest-path-2 always work? If yes, prove it is correct; otherwise, give a counter example.

Solution: Yes. It always works. If there is a shorter path to a node v via (u, v), then the algorithm would have updated $dist(v)$ when u was explored. This contradicts with how the code works. Thus there cannot be a lower distance to v from some path to from s to v.

4. Given a node s on a tree T, we would like to find the shortest paths from s to all other nodes on the tree. The tree could have negatively weighted edges. (DPV 4.4)

(5 points) (a) Define what is a tree in graph theory.

Solution: A tree is a connected, acyclic, and undirected graph.

(7 points) (b) Does Dijkstra's algorithm guarantee correct shortest paths on the tree? Please justify your answer.

Solution: Yes. Dijkstra's algorithm will guarantee to find correct shortest paths on the tree, as there is only one path from s to each node, which will also be the shortest.

(8 points) (c) Give a strategy to find the shortest paths from s in a runtime $o((|V|+|E|)\log|V|)$, asymptotically faster than the runtime of Dijkstra's algorithm using a binary heap.

Solution:

Yes. We can run DFS to find the paths from s to all other nodes and then calculate the distances of each path, which are also the shortest paths.

The runtime will be $O(|V|+|E|)$, faster than the best Dijkstra's algorithm.