CS/ECE 374 P24

Jiawei Tang, Pengxu Zheng

TOTAL POINTS

85 / 100

QUESTION 1

1 24.A. **25 / 30**

- **- 0 pts** Correct
- ✓ **- 5 pts Did not make any statement of the running time, or the analysis of the running time is wrong.**
- **- 10 pts** Running time is slower than linear.
- **- 15 pts** No explanation of the algorithm, i.e. English description.
- **- 25 pts** Wrong solution.
- **- 5 pts** Error in base case.
- **- 5 pts** Other minor error (-5 pts per minor error).
- **- 22.5 pts** IDK policy.

QUESTION 2

2 24.B. **35 / 40**

- **- 0 pts** Correct
- ✓ **- 5 pts Did not make any statement of the running time, or the analysis of the running time is wrong.**
- **- 10 pts** Running time is slower than linear.
- **- 15 pts** No explanation of the algorithm, i.e. English description.
- **- 25 pts** Wrong solution.
- **- 10 pts** Algorithm works for only one vertex and not all.
- **- 5 pts** Other minor error (-5 pts per minor error).
- **- 30 pts** IDK policy.

QUESTION 3

3 24.C. **25 / 30**

- ✓ **- 0 pts Correct**
- ✓ **- 5 pts Did not make any statement of the running time, or the analysis of the running time is wrong.**
- **- 10 pts** Running time is slower than linear.
- **- 15 pts** No explanation of the algorithm, i.e. English description.
- **- 25 pts** Wrong solution.
- **- 10 pts** Algorithm works for only one vertex and not all.
- **- 5 pts** Other minor error (-5 pts per minor error).
- **- 22.5 pts** IDK policy.

HW Solution

CS/ECE 374: Algorithms & Models of Computation, Spring 2019 Version: **1.0**

Submitted by:

- **≪Jiawei Tang≫**: **≪jiaweit2≫**
- **≪Pengxu Zheng≫**: **≪pzheng5≫**

24

Solution:

24.A.

Since we want to find the shortest interval in the DAG G, we want to find the longest path in G. We initialize an array d so that $|d|$ =number of vertices and set all of the elements to -1. Our algorithm is to keep updating the longest path we can get.

$len = 0$
for each $v \in V$ **do**
  if v not visited **then**
    $len = max(len, DFS(v))$
  end if
end for
return len

DFS(v):
$dist = 0$
for each out-edges from v to w **do**
  if $d[w] == -1$ **then**
    $dist = max(dist, DFS(w))$
  end if
end for
return $(d[v] = dist + T(v))$

The time complexity to this algorithm is $O(m)$ where m is the number of edges in G. So this algorithm has linear time complexity.

24.B.

To find the earliest time that job v can begin, we need to find the longest path that ends at v. To find the longest path from a starting vertex i to any $v \in V$, we compute the longest path from i to the predecessors u of v. The distance is maintained in a search table such that dist[v] = max(dist(u) + time(u)). After that, we negate the graph G to make its reverse graph G'. Combined with dist[v] calculation, we apply the same algorithm described in part A to calculate the ealierest time for v to begin.

24.C.

Using t_1 to denote result of part A that all jobs in G can be executed, and using t_v to denote the latest time that job v can begin, we obtain a relationship for the base case where v is the sink node:

$t_v = t_1 - time(v)$.

For recursive steps, we assume that v is in the middle of G such that for all outgoing edges of v, we compute the latest time to start for all of its successors v'. As such, we have the latest time for v to end since v' starts right after v ends. The latest time for v to start is then calculated as $t_v = t_{v'} - time(v)$.

1 24.A. **25 / 30**

- **- 0 pts** Correct
- ✓ **- 5 pts Did not make any statement of the running time, or the analysis of the running time is wrong.**
- **- 10 pts** Running time is slower than linear.
- **- 15 pts** No explanation of the algorithm, i.e. English description.
- **- 25 pts** Wrong solution.
- **- 5 pts** Error in base case.
- **- 5 pts** Other minor error (-5 pts per minor error).
- **- 22.5 pts** IDK policy.

HW Solution

CS/ECE 374: Algorithms & Models of Computation, Spring 2019 Version: **1.0**

Submitted by:

- **≪Jiawei Tang≫**: **≪jiaweit2≫**
- **≪Pengxu Zheng≫**: **≪pzheng5≫**

24

Solution:

24.A.

Since we want to find the shortest interval in the DAG G, we want to find the longest path in G. We initialize an array d so that $|d|$ =number of vertices and set all of the elements to -1. Our algorithm is to keep updating the longest path we can get.

$len = 0$
for each $v \in V$ **do**
  if v not visited **then**
    $len = max(len, DFS(v))$
  end if
end for
return len

DFS(v):
$dist = 0$
for each out-edges from v to w **do**
  if $d[w] == -1$ **then**
    $dist = max(dist, DFS(w))$
  end if
end for
return $(d[v] = dist + T(v))$

The time complexity to this algorithm is $O(m)$ where m is the number of edges in G. So this algorithm has linear time complexity.

24.B.

To find the earliest time that job v can begin, we need to find the longest path that ends at v. To find the longest path from a starting vertex i to any $v \in V$, we compute the longest path from i to the predecessors u of v. The distance is maintained in a search table such that dist[v] = max(dist(u) + time(u)). After that, we negate the graph G to make its reverse graph G'. Combined with dist[v] calculation, we apply the same algorithm described in part A to calculate the ealierest time for v to begin.

24.C.

Using t_1 to denote result of part A that all jobs in G can be executed, and using t_v to denote the latest time that job v can begin, we obtain a relationship for the base case where v is the sink node:

$t_v = t_1 - time(v)$.

For recursive steps, we assume that v is in the middle of G such that for all outgoing edges of v, we compute the latest time to start for all of its successors v'. As such, we have the latest time for v to end since v' starts right after v ends. The latest time for v to start is then calculated as $t_v = t_{v'} - time(v)$.

2 24.B. **35 / 40**

- **- 0 pts** Correct
- ✓ **- 5 pts Did not make any statement of the running time, or the analysis of the running time is wrong.**
- **- 10 pts** Running time is slower than linear.
- **- 15 pts** No explanation of the algorithm, i.e. English description.
- **- 25 pts** Wrong solution.
- **- 10 pts** Algorithm works for only one vertex and not all.
- **- 5 pts** Other minor error (-5 pts per minor error).
- **- 30 pts** IDK policy.

HW Solution

CS/ECE 374: Algorithms & Models of Computation, Spring 2019 Version: **1.0**

Submitted by:

- **≪Jiawei Tang≫**: **≪jiaweit2≫**
- **≪Pengxu Zheng≫**: **≪pzheng5≫**

24

Solution:

24.A.

Since we want to find the shortest interval in the DAG G, we want to find the longest path in G. We initialize an array d so that $|d|$ =number of vertices and set all of the elements to -1. Our algorithm is to keep updating the longest path we can get.

$len = 0$
for each $v \in V$ **do**
  if v not visited **then**
    $len = max(len, DFS(v))$
  end if
end for
return len

DFS(v):
$dist = 0$
for each out-edges from v to w **do**
  if $d[w] == -1$ **then**
    $dist = max(dist, DFS(w))$
  end if
end for
return $(d[v] = dist + T(v))$

The time complexity to this algorithm is $O(m)$ where m is the number of edges in G. So this algorithm has linear time complexity.

24.B.

To find the earliest time that job v can begin, we need to find the longest path that ends at v. To find the longest path from a starting vertex i to any $v \in V$, we compute the longest path from i to the predecessors u of v. The distance is maintained in a search table such that dist[v] = max(dist(u) + time(u)). After that, we negate the graph G to make its reverse graph G'. Combined with dist[v] calculation, we apply the same algorithm described in part A to calculate the ealierest time for v to begin.

24.C.

Using t_1 to denote result of part A that all jobs in G can be executed, and using t_v to denote the latest time that job v can begin, we obtain a relationship for the base case where v is the sink node:

$t_v = t_1 - time(v)$.

For recursive steps, we assume that v is in the middle of G such that for all outgoing edges of v, we compute the latest time to start for all of its successors v'. As such, we have the latest time for v to end since v' starts right after v ends. The latest time for v to start is then calculated as $t_v = t_{v'} - time(v)$.

3 24.C. **25 / 30**

✓ **- 0 pts** Correct

✓ **- 5 pts** **Did not make any statement of the running time, or the analysis of the running time is wrong.**

- 10 pts Running time is slower than linear.

- 15 pts No explanation of the algorithm, i.e. English description.

- 25 pts Wrong solution.

- 10 pts Algorithm works for only one vertex and not all.

- 5 pts Other minor error (-5 pts per minor error).

- 22.5 pts IDK policy.