Assignment 4

Jian Li

Problem 1

---step 1---

Dist is

(1: INF) (2: INF) (3 : 0) (4: INF) (5: INF) (6: INF) (7: INF) (8: INF) (9: INF) (10: INF)

Heap is

Node3.dist = 0

Node2.dist = INF

Node1.dist = INF

Node4.dist = INF

Node5.dist = INF

Node6.dist = INF

Node7.dist = INF

Node8.dist = INF

Node9.dist = INF

Node10.dist = INF

---step 2---

Dist is

(1: INF) (2: INF) (3 : 0) (4: INF) (5: INF) (6: INF) (7: INF) (8: INF) (9: INF) (10 : 1)

Heap is

Node10.dist = 1

Node2.dist = INF

Node1.dist = INF

Node4.dist = INF

Node5.dist = INF

Node6.dist = INF

Node7.dist = INF

Node8.dist = INF

Node9.dist = INF

---step 3---

Dist is

(1: INF) (2: INF) (3 : 0) (4 : 10) (5: INF) (6: INF) (7: INF) (8 : 5) (9: INF) (10 : 1)

Heap is

Node8.dist = 5

Node4.dist = 10

Node6.dist = INF

Node1.dist = INF

Node5.dist = INF

Node9.dist = INF

Node7.dist = INF

Node2.dist = INF

---step 4---

Dist is

(1: INF) (2: INF) (3 : 0) (4 : 10) (5: INF) (6 : 8) (7: INF) (8 : 5) (9 : 8) (10 : 1)

Heap is

Node6.dist = 8

Node1.dist = INF

Node9.dist = 8

Node2.dist = INF

Node5.dist = INF

Node4.dist = 10

Node7.dist = INF

---step 5---

Dist is

(1: INF) (2: INF) (3 : 0) (4 : 10) (5: INF) (6 : 8) (7 : 10) (8 : 5) (9 : 8) (10 : 1)

Heap is

Node9.dist = 8

Node1.dist = INF

Node4.dist = 10

Node2.dist = INF

Node5.dist = INF

Node7.dist = 10

---step 6---

Dist is

(1: INF) (2: INF) (3 : 0) (4 : 10) (5: INF) (6 : 8) (7 : 10) (8 : 5) (9 : 8) (10 : 1)

Heap is

Node4.dist = 10

Node1.dist = INF

Node7.dist = 10

Node2.dist = INF

Node5.dist = INF

---step 7---

Dist is

(1: INF) (2 : 17) (3 : 0) (4 : 10) (5: INF) (6 : 8) (7 : 10) (8 : 5) (9 : 8) (10 : 1)

Heap is

Node7.dist = 10

Node2.dist = 17

Node5.dist = INF

Node1.dist = INF

---step 8---

Dist is

(1: INF) (2 : 17) (3 : 0) (4 : 10) (5: INF) (6 : 8) (7 : 10) (8 : 5) (9 : 8) (10 : 1)

Heap is

Node2.dist = 17

Node1.dist = INF

Node5.dist = INF

---step 9---

Dist is

(1: INF) (2 : 17) (3 : 0) (4 : 10) (5: INF) (6 : 8) (7 : 10) (8 : 5) (9 : 8) (10 : 1)

Heap is

Node1.dist = INF

Node5.dist = INF

---step 10---

Dist is

(1: INF) (2 : 17) (3 : 0) (4 : 10) (5: INF) (6 : 8) (7 : 10) (8 : 5) (9 : 8) (10 : 1)

Heap is

Node5.dist = INF

Problem 2

---iteration 1:---

Node 1:INF

Node 2:INF

Node 3:0

Node 4:INF

Node 5:INF

Node 6:INF

Node 7:INF

Node 8:INF

Node 9:INF

Node 10:INF

---iteration 2:---

Node 1:INF

Node 2:INF

Node 3:0

Node 4:10

Node 5:11

Node 6:8

Node 7:5

Node 8:8

Node 9:13

Node 10:12

---iteration 3:---

Node 1:INF

Node 2:INF

Node 3:0

Node 4:10

Node 5:11

Node 6:8

Node 7:5

Node 8:8

Node 9:7

Node 10:12

---iteration 4:---

Node 1:INF

Node 2:INF

Node 3:0

Node 4:10

Node 5:11

Node 6:8

Node 7:5

Node 8:8

Node 9:7

Node 10:12

Problem 3

General Idea:

Take each task as vertex, the dependencies as edges, and the time needs to finish each task as weights. We can abstract the problem as finding the longest distance in a directed acylic graph. (The problem states that there are no pairs of tasks that depend on each other, so it is sure that the graph will be acylic).

In my algorithm, similar to the single-source shortest-path algorithm for directed acylic graphs in the textbook: First, I topologically sort the graph by running DFS on the graph, with some additional operation in the post-visist procedure, when post-visit a node, add it to a stack, after DFS, I have the topological sort results of nodes in this stack, and it takes O(n) time; Second, run the dag-longest-paths, the difference with the dag-shortest-paths in the text book is that initially set dist[u] = 0, and in update procedure, dist[v] = max(dist[v] , dist[u] + length(u,v)), as this algorithm explore each edge once, it takes O(E) times. I keep track of the parent of each node through this process, finally I track back from the finial task to start task and add the person needed for each task in a collection, this procedure take O(n) time, since the longest route have n nodes in it at most. Thus, the total time complexity will be O(n).

First

Procedure dfs(G)

mystack = empty

for all v in V:

    visited[v] = false

for all v in V:

    if not visited[v]: explore(v)

Procedure explore(G,v)

Input: G=(V,E) is a graph, v in V

Output: visited(u) is set to true for all nodes u reachable from v

visited[v] = true

previsit(v)

for each edge(v,u) in E:

    if not visited[u]: explore(G,u)

postvisit(v)

Procedure previsit(v)

pre[v] = clock

clock = clock +1

Procedure postvisit(v)

post[v] = clock

clock = clock +1

mystack.add(v)

Second

procedure dag-longest-paths(G,l,s)

Input: Dag G = (V,E)

    edge lengths $\{l_e : e \text{ in } E\}$; vertex s in V

Output: For all vertices u reachable from s, dist(u) is set to the distance from s to u

for all u in V:

```
    dist(u) = 0
    prev(u) = nil
dist(s) = s.days
while mystack not emplty:
    u= mystack.pop()
    for all edges (u,v) in E:
        update(u,v)

Procedure update((u,v) in E)
dist(v) = max {dist(v), dist(u) + l(u,v)}
```

Third

```
Procedure track-back(start,final,prev)
Input: start task,final task
       prev: the parent collection task of each task
Output: days and people need for the longest route.

people += u.people
days += u.days
if u == start:
    return people and days
else:
    Procedure track-back(start,prev[final],prev)
```

Problem 4

After topologically sort:

[3, 9, 8, 5, 4, 6, 7, 2, 1, 10]

---initial distance---

Node 1:inf

Node 2:inf

Node 3:0.0

Node 4:inf

Node 5:inf

Node 6:inf

Node 7:inf

Node 8:inf

Node 9:inf

Node 10:inf

---update edges from vertex3:---

Node 1:-4.2

Node 2:3.0

Node 3:0.0

Node 4:9.3

Node 5:inf

Node 6:inf

Node 7:14.4

Node 8:inf

Node 9:18.7

Node 10:5.2

(1.prev is 3)

(2.prev is 3)

(3.prev is None)

(4.prev is 3)

(5.prev is None)

(6.prev is None)

(7.prev is 3)

(8.prev is None)

(9.prev is 3)

(10.prev is 3)

---update edges from vertex9:---

Node 1:-4.2

Node 2:3.0

Node 3:0.0

Node 4:9.3

Node 5:36.2

Node 6:inf

Node 7:14.4

Node 8:38.5

Node 9:18.7

Node 10:5.2

(1.prev is 3)

(2.prev is 3)

(3.prev is None)

(4.prev is 3)

(5.prev is 9)

(6.prev is None)

(7.prev is 3)

(8.prev is 9)

(9.prev is 3)

(10.prev is 3)

---update edges from vertex8:---

Node 1:-4.2

Node 2:3.0

Node 3:0.0

Node 4:9.3

Node 5:36.2

Node 6:inf

Node 7:14.4

Node 8:38.5

Node 9:18.7

Node 10:5.2

(1.prev is 3)

(2.prev is 3)

(3.prev is None)

(4.prev is 3)

(5.prev is 9)

(6.prev is None)

(7.prev is 3)

(8.prev is 9)

(9.prev is 3)

(10.prev is 3)

---update edges from vertex5:---

Node 1:-4.2

Node 2:3.0

Node 3:0.0

Node 4:9.3

Node 5:36.2

Node 6:inf

Node 7:14.4

Node 8:38.5

Node 9:18.7

Node 10:5.2

(1.prev is 3)

(2.prev is 3)

(3.prev is None)

(4.prev is 3)

(5.prev is 9)

(6.prev is None)

(7.prev is 3)

(8.prev is 9)

(9.prev is 3)

(10.prev is 3)

---update edges from vertex4:---

Node 1:-4.2

Node 2:3.0

Node 3:0.0

Node 4:9.3

Node 5:36.2

Node 6:12.8

Node 7:14.4

Node 8:38.5

Node 9:18.7

Node 10:5.2

(1.prev is 3)

(2.prev is 3)

(3.prev is None)

(4.prev is 3)

(5.prev is 9)

(6.prev is 4)

(7.prev is 3)

(8.prev is 9)

(9.prev is 3)

(10.prev is 3)

---update edges from vertex6:---

Node 1:-4.2

Node 2:3.0

Node 3:0.0

Node 4:9.3

Node 5:36.2

Node 6:12.8

Node 7:14.0

Node 8:38.5

Node 9:18.7

Node 10:5.2

(1.prev is 3)

(2.prev is 3)

(3.prev is None)

(4.prev is 3)

(5.prev is 9)

(6.prev is 4)

(7.prev is 6)

(8.prev is 9)

(9.prev is 3)

(10.prev is 3)

---update edges from vertex7:---

Node 1:-4.2

Node 2:3.0

Node 3:0.0

Node 4:9.3

Node 5:36.2

Node 6:12.8

Node 7:14.0

Node 8:38.5

Node 9:18.7

Node 10:5.2

(1.prev is 3)

(2.prev is 3)

(3.prev is None)

(4.prev is 3)

(5.prev is 9)

(6.prev is 4)

(7.prev is 6)

(8.prev is 9)

(9.prev is 3)

(10.prev is 3)

---update edges from vertex2:---

Node 1:-4.2

Node 2:3.0

Node 3:0.0

Node 4:9.3

Node 5:36.2

Node 6:12.8

Node 7:14.0

Node 8:38.5

Node 9:18.7

Node 10:1.0

(1.prev is 3)

(2.prev is 3)

(3.prev is None)

(4.prev is 3)

(5.prev is 9)

(6.prev is 4)

(7.prev is 6)

(8.prev is 9)

(9.prev is 3)

(10.prev is 2)

---update edges from vertex1:---

Node 1:-4.2

Node 2:3.0

Node 3:0.0

Node 4:9.3

Node 5:36.2

Node 6:12.8

Node 7:14.0

Node 8:38.5

Node 9:18.7

Node 10:-13.4

(1.prev is 3)

(2.prev is 3)

(3.prev is None)

(4.prev is 3)

(5.prev is 9)

(6.prev is 4)

(7.prev is 6)

(8.prev is 9)

(9.prev is 3)

(10.prev is 1)

---update edges from vertex10:---

Node 1:-4.2

Node 2:3.0

Node 3:0.0

Node 4:9.3

Node 5:36.2

Node 6:12.8

Node 7:14.0

Node 8:38.5

Node 9:18.7

Node 10:-13.4

(1.prev is 3)

(2.prev is 3)

(3.prev is None)

(4.prev is 3)

(5.prev is 9)

(6.prev is 4)

(7.prev is 6)

(8.prev is 9)

(9.prev is 3)

(10.prev is 1)