**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)), however if dist[v] = dist[u] + length(u,v), means we have another longest route to v, we need to spare another space to store this route.

As this algorithm explore each edge once, it takes O(E) times. I keep track of the intermediate route as well as intermediate people and days need for each node through this process, finally I just need to return the days, people and route of the final task. Thus the time complexity of this algorithm is 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 {le : e in E}; vertex s in V

Output: For all vertices u reachable from s, dist(u) is set to the days needs from start to task u. people(u) is set to the number of people needs from start to task u.

for all u in V:

**days(u) = 0**

route(u) = Null

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)

if dist(v) == dist(u) + l(u,v):

**add another route** to v ( (route(u).append(v) )

else if : dist(v) < dist(u) + l(u,v)

dist(v) = dist(u) + l(u,v)

**route(v) = route(u) .append (v)**

**Third**

Procedure getresults(final, route, dist)

Input: final task

route(updated during Second procedure)

dist(updated during Second procedure)

Output: days needed for longest route

the tasks in longest route

number of people needed for each longest route

for item in route:

return dist[final]

return item

people = 0

for each task in item:

people += task.people

return people

**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)