COMP4500 Assignment 1

Student name: Bosheng Zhang

Student no: 45004830

Part A (30 marks total)

Question 1: Constructing SNI and directed graph

1. My origin SNI: 9845 0048 3052

1 for i = 2 to 12

图片包含 游戏机, 钟表

描述已自动生成2 if d[i] == d[i - 1]

3 d[i] = (d[i] + 3) mod 10

My new SNI: 9845 0348 3052

b) 0->3 0->5

3->4 3->0

4->5 4->8

5->0 5->2

8->4 8->3

9->8

Question 2: Strongly connected components

(a) Perform step 1 of the SCC algorithm using S as input.

Note: the purple node means on visit, gray node means finish visited, white node means still not visited.

The bold edge is the path of performing depth-first search

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| x | Color[x] | Pi[x] |  | x | Color[x] | Pi[x] |  | x | Color[x] | Pi[x] |
| 0 | purple | undef | 0 | purple | undef | 0 | purple | undef |
| 1 | white |  | 1 | white |  | 1 | white |  |
| 2 | white |  | 2 | white |  | 2 | white |  |
| 3 | white | 0 | 3 | white | 0 | 3 | purple | 0 |
| 4 | white |  | 4 | white |  | 4 | purple | 3 |
| 5 | white |  | 5 | white |  | 5 | white |  |
| 6 | white |  | 6 | white |  | 6 | white |  |
| 7 | white |  | 7 | white |  | 7 | white |  |
| 8 | white |  | 8 | white |  | 8 | white |  |
| 9 | white |  | 9 | white |  | 9 | white |  |

图片包含 游戏机, 项链

描述已自动生成图片包含 游戏机

描述已自动生成图片包含 游戏机

描述已自动生成

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| x | Color[x] | Pi[x] |  | x | Color[x] | Pi[x] |  | x | Color[x] | Pi[x] |
| 0 | purple | undef | 0 | purple | undef | 0 | purple | undef |
| 1 | white |  | 1 | white |  | 1 | white |  |
| 2 | white |  | 2 | purple | 5 | 2 | grey | 5 |
| 3 | purple | 0 | 3 | purple | 0 | 3 | purple | 0 |
| 4 | purple | 3 | 4 | purple | 3 | 4 | purple | 3 |
| 5 | purple | 4 | 5 | purple | 4 | 5 | purple | 4 |
| 6 | white |  | 6 | white |  | 6 | white |  |
| 7 | white |  | 7 | white |  | 7 | white |  |
| 8 | white |  | 8 | white |  | 8 | white |  |
| 9 | white |  | 9 | white |  | 9 | white |  |

图片包含 游戏机

描述已自动生成图片包含 游戏机, 画

描述已自动生成图片包含 游戏机

描述已自动生成

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| x | Color[x] | Pi[x] |  | x | Color[x] | Pi[x] |  | x | Color[x] | Pi[x] |
| 0 | purple | undef | 0 | purple | undef | 0 | purple | undef |
| 1 | white |  | 1 | white |  | 1 | white |  |
| 2 | grey | 5 | 2 | grey | 5 | 2 | grey | 5 |
| 3 | purple | 0 | 3 | purple | 0 | 3 | purple | 0 |
| 4 | purple | 3 | 4 | purple | 3 | 4 | purple | 3 |
| 5 | grey | 4 | 5 | grey | 4 | 5 | grey | 4 |
| 6 | white |  | 6 | white |  | 6 | white |  |
| 7 | white |  | 7 | white |  | 7 | white |  |
| 8 | white |  | 8 | purple | 4 | 8 | grey | 4 |
| 9 | white |  | 9 | white |  | 9 | white |  |

图片包含 游戏机, 灯光

描述已自动生成图片包含 游戏机, 灯光

描述已自动生成图片包含 游戏机, 灯光

描述已自动生成

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| x | Color[x] | Pi[x] |  | x | Color[x] | Pi[x] |  | x | Color[x] | Pi[x] |
| 0 | purple | undef | 0 | purple | undef | 0 | grey | undef |
| 1 | white |  | 1 | white |  | 1 | white |  |
| 2 | grey | 5 | 2 | grey | 5 | 2 | grey | 5 |
| 3 | purple | 0 | 3 | grey | 0 | 3 | grey | 0 |
| 4 | grey | 3 | 4 | grey | 3 | 4 | grey | 3 |
| 5 | grey | 4 | 5 | grey | 4 | 5 | grey | 4 |
| 6 | white |  | 6 | white |  | 6 | white |  |
| 7 | white |  | 7 | white |  | 7 | white |  |
| 8 | grey | 4 | 8 | grey | 4 | 8 | grey | 4 |
| 9 | white |  | 9 | white |  | 9 | white |  |

图片包含 游戏机, 灯光

描述已自动生成图片包含 游戏机, 灯光, 电脑

描述已自动生成图片包含 游戏机, 灯光, 空气, 电脑

描述已自动生成

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| x | Color[x] | Pi[x] |  | x | Color[x] | Pi[x] |  | x | Color[x] | Pi[x] |
| 0 | grey | undef | 0 | grey | undef | 0 | grey | undef |
| 1 | purple | undef | 1 | grey | undef | 1 | grey | undef |
| 2 | grey | 5 | 2 | grey | 5 | 2 | grey | 5 |
| 3 | grey | 0 | 3 | grey | 0 | 3 | grey | 0 |
| 4 | grey | 3 | 4 | grey | 3 | 4 | grey | 3 |
| 5 | grey | 4 | 5 | grey | 4 | 5 | grey | 4 |
| 6 | white |  | 6 | white |  | 6 | purple | undef |
| 7 | white |  | 7 | white |  | 7 | white |  |
| 8 | grey | 4 | 8 | grey | 4 | 8 | grey | 4 |
| 9 | white |  | 9 | white |  | 9 | white |  |

图片包含 游戏机, 电脑, 灯光

描述已自动生成图片包含 游戏机, 电脑, 灯光

描述已自动生成图片包含 游戏机, 电脑, 灯光

描述已自动生成

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| x | Color[x] | Pi[x] |  | x | Color[x] | Pi[x] |  | x | Color[x] | Pi[x] |
| 0 | grey | undef | 0 | grey | undef | 0 | grey | undef |
| 1 | grey | undef | 1 | grey | undef | 1 | grey | undef |
| 2 | grey | 5 | 2 | grey | 5 | 2 | grey | 5 |
| 3 | grey | 0 | 3 | grey | 0 | 3 | grey | 0 |
| 4 | grey | 3 | 4 | grey | 3 | 4 | grey | 3 |
| 5 | grey | 4 | 5 | grey | 4 | 5 | grey | 4 |
| 6 | grey | undef | 6 | grey | undef | 6 | grey | undef |
| 7 | white |  | 7 | purple | undef | 7 | grey | undef |
| 8 | grey | 4 | 8 | grey | 4 | 8 | grey | 4 |
| 9 | white |  | 9 | white |  | 9 | white |  |

图片包含 游戏机, 电脑, 灯光

描述已自动生成图片包含 游戏机, 电脑, 灯光

描述已自动生成图片包含 游戏机, 电脑

描述已自动生成

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| x | Color[x] | Pi[x] |  | x | Color[x] | Pi[x] |
| 0 | grey | undef | 0 | grey | undef |
| 1 | grey | undef | 1 | grey | undef |
| 2 | grey | 5 | 2 | grey | 5 |
| 3 | grey | 0 | 3 | grey | 0 |
| 4 | grey | 3 | 4 | grey | 3 |
| 5 | grey | 4 | 5 | grey | 4 |
| 6 | grey | undef | 6 | grey | undef |
| 7 | grey | undef | 7 | grey | undef |
| 8 | grey | 4 | 8 | grey | 4 |
| 9 | purple | undef | 9 | grey | undef |

图片包含 游戏机, 电脑

描述已自动生成图片包含 游戏机, 电子, 电脑

描述已自动生成

Second last graph: Finishing times for the original graph G

Last graph: Strongly Connected Components

图片包含 游戏机, 钟表

描述已自动生成(b) 0 -> 5 0 -> 3

2 -> 5

3 -> 0 3 -> 8

4 -> 8 4 -> 3

5 -> 4 5 -> 0

8 -> 9 8 -> 4

(c) Perform steps 3, 4 of the SCC algorithms.图片包含 游戏机

描述已自动生成图片包含 游戏机

描述已自动生成图片包含 游戏机

描述已自动生成图片包含 游戏机

描述已自动生成图片包含 游戏机

描述已自动生成手机屏幕的截图

描述已自动生成图片包含 游戏机

描述已自动生成手机屏幕的截图

描述已自动生成手机屏幕的截图

描述已自动生成图片包含 游戏机

描述已自动生成图片包含 游戏机, 画

描述已自动生成图片包含 游戏机, 画

描述已自动生成手机屏幕的截图

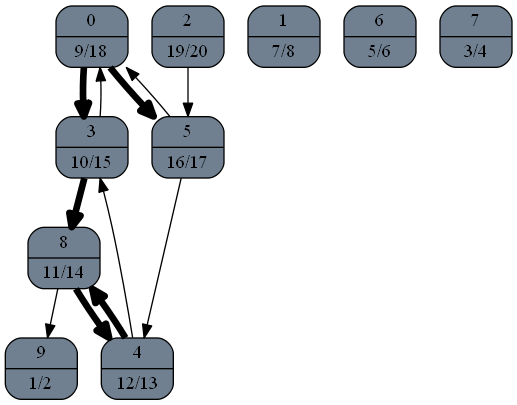
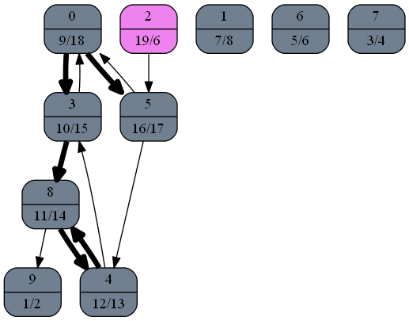
描述已自动生成手机屏幕的截图

描述已自动生成手机屏幕的截图

描述已自动生成手机屏幕的截图

描述已自动生成图片包含 游戏机, 钟表

描述已自动生成图片包含 游戏机

描述已自动生成Finishing times for GT

图片包含 游戏机

描述已自动生成Strongly Connected Component

1. 9
2. 7
3. 6
4. 1
5. 0 3 8 4 5
6. 2

Part B (70 marks total)

Question 4: Worst-case time complexity analysis

a) In my method, I treat each interaction as a node in the graph, and the edges represent the two element sub paths of possible route of transmission from personFrom to personTo.

List<Interaction> FindTransmissionPath(int start, int end, List<Interaction> interactions) {  
1 personToInteractions: HashMap<Integer, HashSet<Interaction>> = new HashMap().onLookupFail(new HashSet())  
2 endInteractions: HashSet<Interaction> = new HashSet()  
3 // I (number of interactions) iterations with O(1) loop body  
4 // with O(P) (number of person) Hashset construction cost   
5 // Overall: O(I + P)  
6 // Worst-case: |P| = 2 \* |I|, if each interaction transmit between two unique person  
7 // 3 \* |I| give us Overall O(I)  
8 for interaction in interactions:  
9 startInteractions: HashSet<Interaction> = personToInteractions[interaction.PersonFrom]   
10 startInteractions.add(interaction) // O(1)  
11 if interaction.personTo == end -> endInteractions.add(interaction)  
12 // O(I) due to implementation limitations  
13 sources: HashSet<Interaction> = personToInteractions[start]  
  
14 adjacency: HashMap<Interaction, HashSet<Interaction>> = new HashMap().onLookupFail(new HashSet())  
15 // I iterations with worst-case: O(I) loop body, if successors' time >= predecessors' time  
16 // and successors' personFrom don't equal to predecessors' personTo  
17 // Overall: O(I^2)  
18 for interaction in interactions:  
19 neighbors: HashSet<Interaction> = personToInteractions[interaction.personTo]  
20 .filter(i -> interaction.time <= i.time && interaction.PersonFrom != i.PersonTo)//O(I)  
21 adjacency[interaction] = neighbors // O(1)  
  
22 // Running time for Dijkstra's algorithm using a Java Heap as a priority queue is  
23 // O((|E| + |V|) \* log|V|).  
24 // Worst-case: |E| = |V^2|, we get O((|V^2| + |V|) \* log|V|).  
25 // as we use Interaction I as our Vertex  
26 // so O((|V^2| + |V|) \* log|V|) give us Overall O(I^2 \* logI)  
27 Dijkstra(adjacency, sources);  
28 maximumProb: Double = personToInteractions[end].  
29 .filter(v -> v.prob != Double.*MAX\_VALUE*).maxBy(v -> v.prob).prob  
30 finalInteractionList: List<Interaction> = new ArrayList<>();  
31 finalInteractionList.add(lowestDDestinationVertex.get().element);  
32 while True // O(I)  
33 if head is not null  
34 finalInteractionList.add(0, head.element);  
35 head = head.predecessor;  
36 else // head is null (finish)  
37 return finalInteractionList;

b) The first part of my algorithm to prepare the parameters of Dijkstra's algorithm, which are a HashMap of one interaction to its valid neighbors and a HashSet of source interactions.

We assume that when HashSet and HashMap execute put, add, and get, its worst-case time complexity will be O(1) instead of O(p).

For doing a graph search, I first convert all interaction to a vertex and build an endInteraction HashSet where the path finish, which is Overall O(i);

For creating the HashSet of sources, it is O(i) in worst-case;

For creating the HashMap, I run through all interactions and filter its time elapsed and check if there is a loop in interactions, which is overall O(i^2).

Then the Running time for Dijkstra's algorithm using a Java Heap as a priority queue is Overall O(i^2 \* log i) in worst-case. We use a Binary Heap in the implementation of Dijskstra’s, with O(lgV) Extract-Mins and Decrease-Keys.

In the last part, my algorithm is to find the path with the highest probability through all endInteraction HashSets, and then add it to an ArrayList, both of which are O(i).

Thus, the time complexity is O(i^2 \* log i) which describes an asymptotic upper bound on the worst-case time complexity of this algorithm.