Homework 9

Yu Hsiao   Yu-Hsuan Wu

Question 1

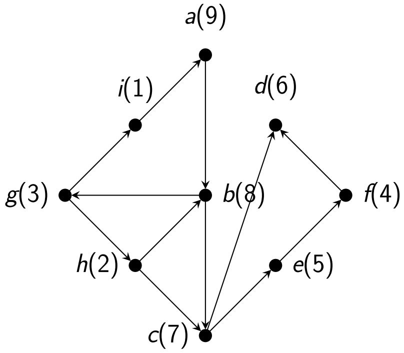
1. (7.16 modified)

   (a) Run the strongly connected components algorithm (the original version) on the directed graph shown in Figure 1. When traversing the graph, the algorithm should follow the given DFS numbers (from 9 down to 1). Show the *High* values as computed by the algorithm in each step.

   (b) Add the edge $(4, 7)$ to the graph and discuss the changes this makes to the execution of the algorithm.

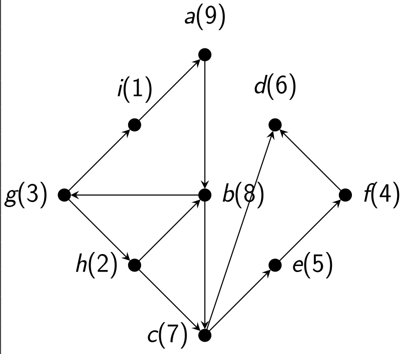
Question 1 (a)

Change the DFS numbers to the corresponding alphabetical order to avoid confusion.



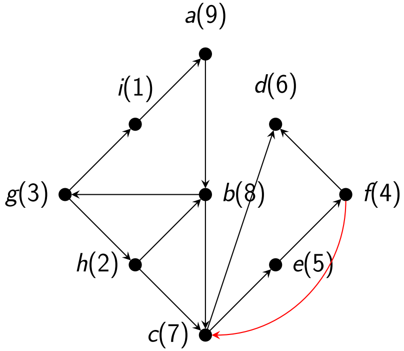
Question 1 (a)

Vertex	a	b	c	d	e	f	g	h	i
DFS_Number	9	8	7	6	5	4	3	2	1
a	9	-	-	-	-	-	-	-	-
b	9	8	-	-	-	-	-	-	-
c	9	8	7	-	-	-	-	-	-
ⓓ	9	8	7	6	-	-	-	-	-
c	9	8	7	6	-	-	-	-	-
e	9	8	7	6	5	-	-	-	-
ⓕ	9	8	7	6	5	4	-	-	-
ⓔ	9	8	7	6	5	4	-	-	-
ⓒ	9	8	7	6	5	4	-	-	-
b	9	8	7	6	5	4	-	-	-
g	9	8	7	6	5	4	3	-	-
h	9	8	7	6	5	4	3	8	-
g	9	8	7	6	5	4	8	8	-
i	9	8	7	6	5	4	8	8	9
g	9	8	7	6	5	4	9	8	9
b	9	9	7	6	5	4	9	8	9
ⓐ	9	9	7	6	5	4	9	8	9
Component_Num	5	5	4	1	3	2	5	5	5



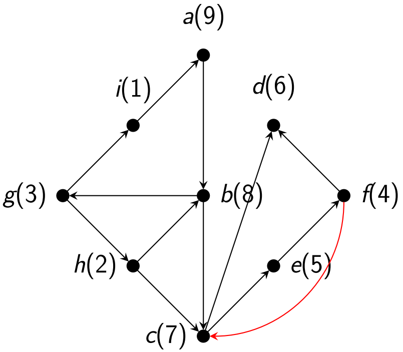
Question 1 (b)

Add the edge (4, 7) (=(f,c)) to the graph and discuss the changes.



Question 1 (b)

Vertex	a	b	c	d	e	f	g	h	i
DFS_Number	9	8	7	6	5	4	3	2	1
a	9	-	-	-	-	-	-	-	-
b	9	8	-	-	-	-	-	-	-
c	9	8	7	-	-	-	-	-	-
ⓓ	9	8	7	6	-	-	-	-	-
e	9	8	7	6	5	-	-	-	-
f	9	8	7	6	5	7	-	-	-
e	9	8	7	6	7	7	-	-	-
ⓒ	9	8	7	6	7	7	-	-	-
b	9	8	7	6	7	7	-	-	-
g	9	8	7	6	7	7	3	-	-
h	9	8	7	6	7	7	3	8	-
g	9	8	7	6	7	7	8	8	-
i	9	8	7	6	7	7	8	8	9
g	9	8	7	6	7	7	9	8	9
b	9	9	7	6	7	7	9	8	9
ⓐ	9	9	7	6	7	7	9	8	9
Component_Num	3	3	2	1	2	2	3	3	3



Question 2

2. Consider the algorithm discussed in class for determining the strongly connected components of a directed graph. Is the algorithm still correct if we replace the line "$v.high := \max(v.high, w.DFS\_Number)$" by "$v.high := \max(v.high, w.high)$"? Why? Please explain.

Question 2

```
procedure SCC($v$)
    ...
    for all edges $(v, w)$ do
        if $w.DFS\_Number = 0$ then
            $SCC(w)$;
            $v.High := \max(v.High, w.High)$;
        else if $w.DFS\_Number > v.DFS\_Number$ and $w.Component = 0$ then
            $v.High := \max(v.High,\ w.high)$;
        end if
    end for
    ...
end procedure
```

Question 2

The algorithm is still correct.

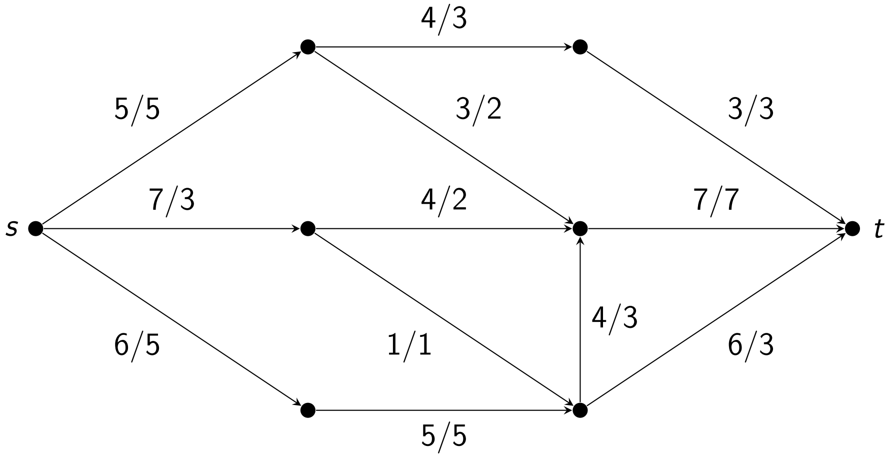
Since $w.high \geq w.DFS\_Number$ by the design of the algorithm naturally (i.e., case $w.high < w.DFS\_Number$ won't occur), we can now consider two cases:

1. If $w.high = w.DFS\_Number$, then the algorithm is the same as the original version.
2. The **else if** statement being true indicates v and w must belong to the same SCC. Thus, when $w.high > w.DFS\_Number$, we know that w belongs to SCC of a higher leader. Therefore, assigning $w.high$ to $v.high$ will also be correct.

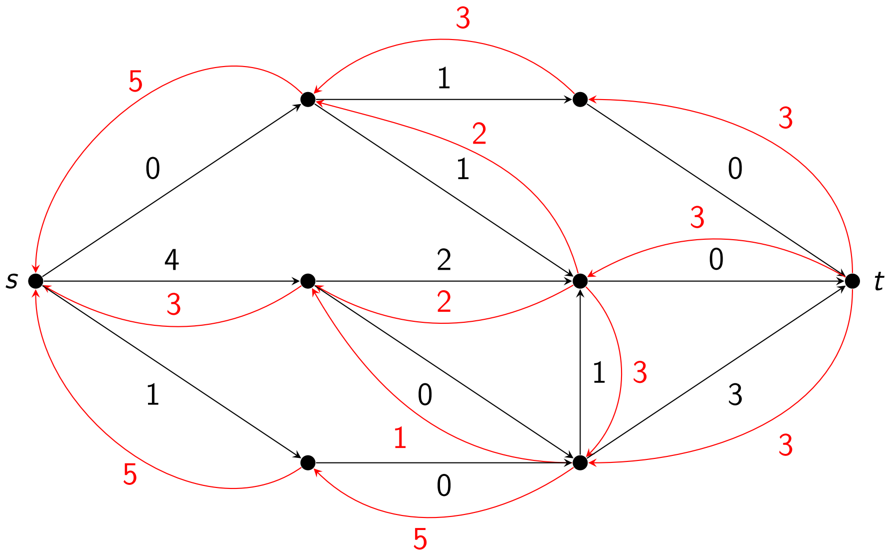
Question 3

3. We have discussed in class the idea of using DFS to find an augmenting path (if one exists) in a network with some given flow. Please present the algorithm in suitable pseudocode.

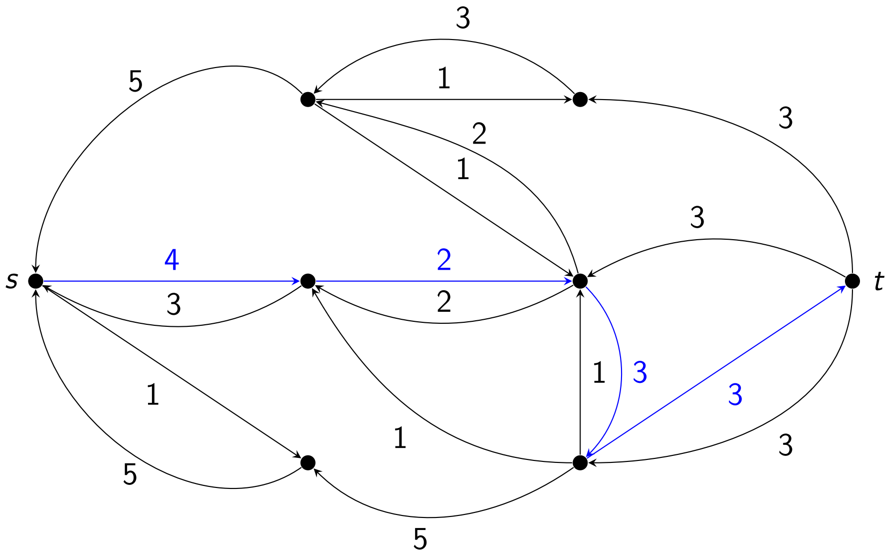
Question 3



Question 3



Question 3



Question 3

```
Algorithm AugmentingPathDFS($G(V, E), v$)
    begin
        mark $v$;
        $Temp\_S$.push($v$);
        if $v = t$ then
            $ResultS := TempS$;
            $found := true$;
        end if
        for each edge $(v, w) \in E$ do
            if $found$ then
                break;
            end if
            if $(f(v, w) < c(v, w)$ or $f(w, v) > 0)$ and $w$ is unmarked then
                AugmentingPathDFS($G, w$);
            end if
        end for
        $Temp\_S$.pop();
    end
end Algorithm
```

Question 3

```
Algorithm AugmentingPath($G(V, E), s, t$)
    begin
        $Temp\_S$, $Result\_S$ := empty stack;
        $found$ := false;
        AugmentingPathDFS($G, s$);
        if $found$ then
            $ResultPath\_S$ := empty stack;
            $post\_v$ := $Result\_S$.pop();
            while $Result\_S \neq$ empty do
                $current\_v$ := $Result\_S$.pop();
                $ResultPath\_S$.push(edge($current\_v$, $post\_v$));
                $post\_v$ := $current\_v$;
            end while
        else
            print "Augmenting path does not exist.";
        end if
    end
end Algorithm
```

Question 4

4. Consider designing by dynamic programming an algorithm that, given as input a sequence of distinct numbers, determines the length of a longest increasing subsequence in the input sequence. For instance, if the input sequence is $1, 3, 11, 5, 12, 14, 7, 9, 15$, then a longest increasing subsequence is $1, 3, 5, 7, 9, 15$ whose length is 6 (another longest increasing subsequence is $1, 3, 11, 12, 14, 15$).

   (a) Formulate the solution using recurrence relations.

   (b) Present the algorithm in adequate pseudocode, based on the previous recursive formulation. What is the time complexity of your algorithm?

Question 4 (a)

Suppose we have a sequence S with length n. We first consider the length of the longest increasing subsequence of S ending with $S[n]$. We have three cases:

- $n = 1$: The longest increasing subsequence length is obviously 1.
- $n > 1$:
  - all the previous elements are larger than $S[n]$: the longest increasing subsequence length is 1.
  - otherwise: the length would be 1 plus the maximum value of all the possible prefixes with its last element less than s_n.

Thus, for any i such that $1 \leq i \leq n$, we have

$$L(i) = \begin{cases} 1 & \text{if } i = 1, \\ 1 & \text{if } S[i] < S[j],\ \forall j.\ 1 \leq j < i, \\ 1 + \max_{j \in J}(L(j)) & \text{otherwise, where} \\ & \quad J = \{j \mid 1 \leq j < i \land S[i] > S[j]\}. \end{cases}$$

Question 4 (a)

The length of the longest increasing subsequence in S, denoted $LIS(S)$, is the **maximum** value in $L(S')$ for all prefix sequences S' of S.

That is, $LIS(S) = \max_{1 \leq i \leq n}(L(i))$.

Question 4 (b)

```
Algorithm LIS($S$)
    begin
        $n := len(S)$, $max := 1$;
        initialize $L[1..n]$ with 1;
        for $i := 2$ to $n$ do
            for $j := 1$ to $i - 1$ do
                if $S[i] > S[j]$ and $L[j] + 1 > L[i]$ then
                    $L[i] := L[j] + 1$;
                end if
            end for
            if $L[i] > max$ then
                $max := L[i]$;
            end if
        end for
        return $max$;
    end
end Algorithm
```

Time Complexity: $O(n^2)$.

Question 5

5. The cost of finding a key value in a binary search tree is linearly proportional to the depth/level of the node where the key value is stored, with the root considered to be at level 0. Obviously, for a key value that is known to be looked up more frequently, it is better stored in a node at a smaller level.

   Consider designing by dynamic programming an algorithm that, given the look-up frequencies of n key values, constructs an optimal binary search tree that will incur the least total cost for performing all the look-ups.

   (a) Formulate the solution using recurrence relations; let $F[1..n]$ be the look-up frequencies of the n key values $K[1..n]$, which are in sorted order.

   (b) Present the algorithm in suitable pseudocode, based on the previous recursive formulation. What is the time complexity of your algorithm?

Question 5 (a)

Among all possible subtrees, find the one with minimal cost.

For each subtree, its cost is the sum of (1) left child, (2) right child, and (3) the frequency of all nodes in the subtree since every level of each point is increasing by 1 due to the new added root.

Let $OPTcost(i, j)$ be the minimal cost of the subtree containing items from i to j.

Making each node as root r, try to find the one which can provide the minimal cost.

$OPTcost(i, j) =$

$$\begin{cases} 0 & \text{if } i > j \\ F[i] & \text{if } i = j \\ \min_{i \le r \le j} \{optcost(i, r-1) + optcost(r+1, j) + \sum_{k=i}^{j} F[k]\} & \text{otherwise} \end{cases}$$

Question 5 (b)

```
function ConstructOptimalBST($K, F, n$)
    // initialization
    initialize $cost[1..n][1..n]$ with $\infty$;
    initialize $root[1..n][1..n]$ with 0;
    for $i = 1$ to $n$ do
        $cost[i][i] := F[i]$
        $root[i][i] := i$
    end for
    for $L = 2$ to $n$ do
        for $i = 1$ to $n - (L - 1)$ do
            $j := i + L - 1$
            $freqSum := FreqSum(F, i, j)$ // calculate $\sum_{k=i}^{j} F[k]$
            $\vdots$
```

Question 5 (b)

```
        for $r = i$ to $j$ do
            $c := freqSum$
            if $r > i$ then
                $c := c + cost[i][r-1]$
            end if
            if $r < j$ then
                $c := c + cost[r+1][j]$
            end if
            if $c < cost[i][j]$ then
                $cost[i][j] := c$
                $root[i][j] := r$
            end if
        end for
    end for
end for
return $cost[1][n]$ , $BuildTree(K, root, 1, n)$
```

Question 5 (b)

```
function FreqSum($F, i, j$)
    $freq\_sum := 0$
    for $k = itoj$ do
        $freq\_sum := freq\_sum + F[k]$
    end for
    return $freq\_sum$
end function=0
function BuildTree($K, R, i, j$)
    if $i > j$ then
        return null;
    end if
    $root := K[R[i][j]]$
    $root.left := BuildTree(K, R, i, R[i][j] - 1)$
    $root.right := BuildTree(K, R, R[i][j] + 1, j)$
    return $root$
end function
```

Time complexity: $O(n^3)$.