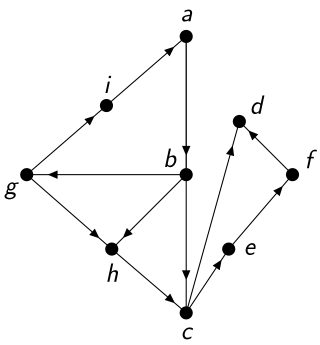
Homework 9

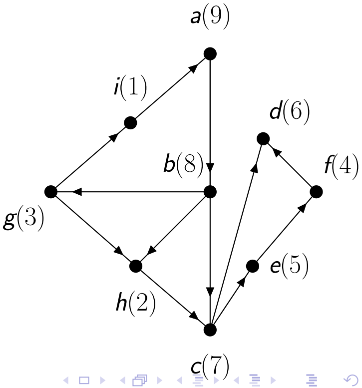
Problem1 (a)

Run the strongly connected components algorithm on the directed graph shown in Figure 1.
When traversing the graph, the algorithm should follow the given DFS numbers.
Show the *High* values as computed by the algorithm in each step.



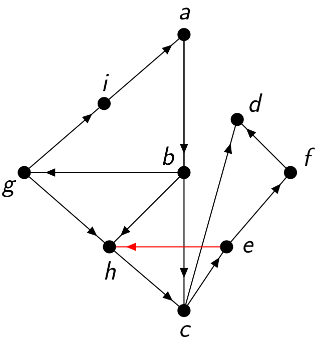
Problem1 (a)

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



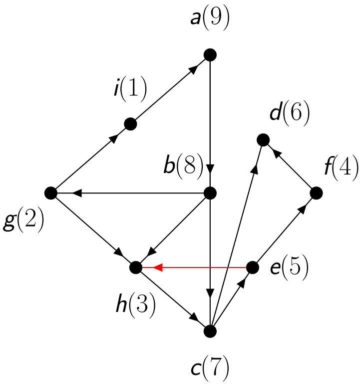
Problem1 (b)

Add the edge (5, 8) (=(e,h)) to the graph and discuss the changes this makes to the execution of the algorithm.



Problem1 (b)

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

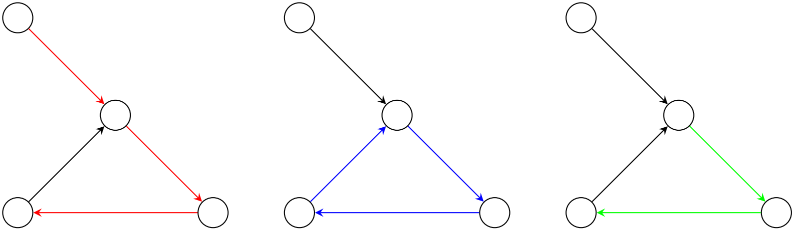


Problem2

Let $G = (V, E)$ be a directed graph, and let T be a DFS tree of G. Prove that the intersection of the edges of T with the edges of any strongly connected component of G form a subtree of T.

Problem2(cont'd)

Simple example



(a) DFS tree **(b)** SCC edges **(c)** intersection

Problem2(cont'd)

Proof by contradiction:
Suppose the intersection are two subtrees $T1$ and $T2$. Because $T1$ and $T2$ are in the same strongly connected component, according to the property of SCC, there must be an edge from $T1$ to $T2$ and an edge from $T2$ to $T1$.
No matter which subtree the DFS procedure reaches first, it will finally go through the edge which connects $T1$ and $T2$ and visit the other subtree. Then the DFS tree T must contain that edge but it clearly doesn't. Contradiction.

Problem3

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.

Problem3(cont'd)

Algorithm

function STRONGLY_CONNECTED_COMPONENTS*(G, n)*
    for *every vertex v of G* ***do***
        $v.DFS\_Number := 0;$
        $v.Component := 0;$
    $Current\_Component := 0;$ $DFS\_N := 0;$
    while $v.DFS\_Number = 0$ *for some v* ***do***
        $SCC(v)$

Problem3(cont'd)

Algorithm

```
procedure SCC(v)
    v.DFS_Number := DFS_N;
    DFS_N := DFS_N−1;
    insert v into Stack;
    v.High := v.DFS_Number;
    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.DFS_Number)
    if v.High = v.DFS_Number then
        CurrentComponent := CurrentComponent + 1;
        repeat
            remove x from the top of Stack;
            x.component := CurrentComponent
        until x = v
```

Problem3(cont'd)

Still correct.
Only when line 10 is True(We look at a vertex w that we have reached before and it does not belong to any SCC yet), we can reach line 11.
At this moment, if $w.DFS\_Number$ and $w.High$ are the same then this case has no impact. If they are different, the only case is $w.DFS\_Number < w.High$, indicating that v and w are in the same SCC. Since we will finally return to the vertex that is the leader of this SCC, its $High$ will set to $\max(v.High, w.High)$, which is exactly its DFS_Number(Because the propagation of the $High$ value of the SCC leader).
Hence when we reach line 11, we can argue that the algorithm is still correct if we replace line 11 by "$v.high := \max(v.high, w.high)$" .
If you have trouble understanding this, draw a figure and trace the code!

Problem4

給定兩個 sequence，求出這兩個 sequence 的最長共同子序列

Problem4

開一個二維陣列
每格代表兩個子字串的最長共同子序列長度

	-	a	b	c	a	b	c	a	b	c
-	0	0	0	0	0	0	0	0	0	0
a	0									
a	0									
a	0									
b	0									
b	0									
b	0									
c	0									
c	0									
c	0									

Problem4

首先先看 abcabcabc 與 a 能夠迸出什麼火花

	-	a	b	c	a	b	c	a	b	c
-	0	0	0	0	0	0	0	0	0	0
a	0									

從最左邊開始看

	-	a	b	c	a	b	c	a	b	c
-	0	0	0	0	0	0	0	0	0	0
a	0									

相當於找 a 與 a 的最長共同子序列

Problem4

LCS(A, B) = 0 if A or B are empty
LCS(A, B) = LCS(A-1, B-1) if A = (A-1) + x, B = (B-1) + x

Problem4

若有兩個字串 A 與 B，兩個字串的結尾相同，比如都是 x
那麼很明顯，A 與 B 的最長共同子序列應該是 A-1 與 B-1（去掉結尾字元）這兩個子字串的最長共同子序列，再加上原本被去掉的 x

在剛剛的例子，a 與 a 都有相同結尾
所以這格應該填上 (空字串) 與 (空字串) 的結果 +1
也就是 1

	-	a	b	c	a	b	c	a	b	c
-	0	0	0	0	0	0	0	0	0	0
a	0	1								

Problem4

LCS(A, B) = 0 if A or B are empty
LCS(A, B) = LCS(A-1, B-1) if A = (A-1) + x, B = (B-1) + x

otherwise...
LCS(A, B) = max(LCS(A-1, B), LCS(A, B-1), LCS(A-1, B-1))

Problem4

若有兩個字串 A 與 B，兩個字串的結尾不同，比如 x 與 y
最長共同子序列不可能同時包含這兩個，因為這兩個 x y 都在結尾
所以就去檢查 A-1 與 B 的，以及 A 與 B-1 的，以及 A-1 與 B-1

	-	a	b	c	a	b	c	a	b	c
-	0	0	0	0	0	0	0	0	0	0
a	0	1								

此時表格上方是 0 (ab 與空字串)、左方是 1 (a 與 a)、左上方是 0 (a 與空字串)。取最大的那個，也就是 1

	-	a	b	c	a	b	c	a	b	c
-	0	0	0	0	0	0	0	0	0	0
a	0	1	1							

其中檢查左上方的程序可以省略 (why?)
時間複雜度是 $O(mn)$

Problem4

	-	a	b	c	a	b	c	a	b	c
-	0	0	0	0	0	0	0	0	0	0
a	0	1	1	1	1	1	1	1	1	1
a	0	1	1	1	2	2	2	2	2	2
a	0	1	1	1	2	2	2	3	3	3
b	0	1	2	2	2	3	3	3	4	4
b	0	1	2	2	2	3	3	3	4	4
b	0	1	2	2	2	3	3	3	4	4
c	0	1	2	3	3	3	4	4	4	5
c	0	1	2	3	3	3	4	4	4	5
c	0	1	2	3	3	3	4	4	4	5

其中紅字代表這格是同字元結尾
也就是「左上 $+1$」步驟發生的地點

Problem4

```
function LCS(A, B)
    m := len(A), n := len(B);
    initialize Table with type int[m + 1][n + 1];
    for i from 0 to m do
        for j from 0 to n do
            if i = 0 or j = 0 then
                Table[i][j] := 0;
            else if A[i - 1] = B[j - 1] then
                Table[i][j] := Table[i - 1][j - 1] + 1;
            else
                Table[i][j] := max(Table[i][j - 1], Table[i - 1][j]);
    return Table[m][n];
```

Problem5

給定一個 sequence，求出最長遞增子序列

Problem5

length(n) = max(length(i) + 1 for i in 0..(n-1) if S[i] $<$ S[n])

Problem5

開一個陣列 *length*

1	3	11	5	12	14	7	9	15
1	1	1	1	1	1	1	1	1

一開始都是 1，因為只由自己構成的最長遞增子序列長度就是 1

Problem5

首先固定最左邊的元素，也就是確定 length(0) 的值
看後面哪些元素可以接在他後面

1	3	11	5	12	14	7	9	15
1	1	1	1	1	1	1	1	1

很明顯 3 可以接在 1 後面，且 (1, 3) 比 (3) 長，所以值更新為 2
從遞迴表達的角度來看，length(1) = max(length(0)+1, ...)

1	3	11	5	12	14	7	9	15
1	2	1	1	1	1	1	1	1

以此類推

1	3	11	5	12	14	7	9	15
1	2	2	2	2	2	2	2	2

Problem5

接下來固定第二項元素，並照這個作法做下去

1	3	11	5	12	14	7	9	15
1	2	3	3	3	3	3	3	3

當我們固定 11 時，出現了不能接在 11 後面的元素

1	3	11	5	12	14	7	9	15
1	2	3	3	3	3	3	3	3

那麼就不要更動，於是這輪的結果是

1	3	11	5	12	14	7	9	15
1	2	3	3	4	4	3	3	4

此時 5 的那格 3 代表 (1, 3, 5)，7 的那格 3 則是 (1, 3, 7)
length(n) 代表目前為止，以 S[n] 為終點的最長遞增子序列為何
把所有元素都固定一次，就能找到當中最長的

Problem5

接下來固定 5，遇到比較大的 12 了

1	3	11	5	12	14	7	9	15
1	2	3	3	4	4	3	3	4

要不要更動值？
前面說過 5 的那格 3 代表 (1, 3, 5)
而 12 那格 4 則是 (1, 3, 11, 12)
要讓 4 變成 5 的話，代表會有 (1, 3, 11, 5, 12)，不合題意
所以應該是拿 5 的那個序列與 12 產生的 (1, 3, 5, 12) 與 12 原本帶有的 (1, 3, 11, 12) 來比，也就是，max(3 + 1, 4)

1	3	11	5	12	14	7	9	15
1	2	3	3	4	4	3	3	4

length(4) = max(length(0)+1, length(1)+1, length(2)+1, length(3)+1)

Problem5

最終結果

1	3	11	5	12	14	7	9	15
1	2	3	3	4	5	4	5	6

答案取 5

Problem5

```
function LIS(S)
    n := len(S);
    initialize length with type int[n];
    for i from 0 to (n-1) do
        length[i] := 1;
    for i from 0 to (n-1) do
        for j from (i+1) to (n-1) do
            if S[i] < S[j] then
                length[j] := max(length[i]+1, length[j]);
    return max(length);
```

複雜度為 $O(n^2)$

Problem5

有另外一種演算法可以降低複雜度 ?!
依然是基於同樣的遞迴關係

Problem5

開一個 list

1	2	3	4	5	6	7
-	-	-	-	-	-	-

一開始先把開頭元素放進去

1	2	3	4	5	6	7
1	-	-	-	-	-	-

這象徵了 1 這個元素對應到的 length 值是 1

Problem5

接下來看 3 要放哪？
由關係式我們知道要往前看比 3 小的元素的 length 值再 +1
在 list 當中的操作是：找到比自己**略小**的元素，往後方加上去

1	2	3	4	5	6	7
1	3	-	-	-	-	-

加入 11 也同理。那麼加入 5 的時候呢？
由於 11 比 5 大，所以 11 對應到的 length 不納入考量

1	2	3	4	5	6	7
1	3	11	-	-	-	-

此時略小於 5 的元素是 3，於是在 3 的後面加上 5

1	2	3	4	5	6	7
1	3	11	-	-	-	-
		5				

Problem5

放進 12

在 12 前面，11 與 5 都有最大的 length

1	2	3	4	5	6	7
1	3	11	-	-	-	-
		5	12			

我們可以只看 5 就好，因為可能出現介於 5 與 11 的數字，他可以放在 5 後面變成最長

也就是，上頭的 11 是可以忽略的

1	2	3	4	5	6	7
1	3	5	12	-	-	-

這是演算法運行時，list 實際的內容

Problem5

又因為我們是挑正好略小的元素塞到右邊
所以被取代的格子右方的數字（如果有的話）必然比塞進去的數字大
所以這個 list 總會是有序的

二元搜尋！
利用二元搜尋就能在 log 時間級找到**略小**的元素在哪！
時間複雜度變成 $O(n \log n)$

Problem5

```
function LIS2(S)
    n := len(S), max_length := 1;
    initialize List with type int[n];
    List[0] := S[0];
    for i from 1 to (n-1) do
        find the largest element in List[0:max_length] that is smaller than S[i];
        put S[i] right after the element;
        if S[i] is put beyond List[0:max_length] then
            max_length++;
    return max_length;
```

C++ 的 std::lower_bond 函數就提供了「給出略小的元素的右邊」在哪的功能。

Problem5

範例題，概念上最終結果就是

1	2	3	4	5	6	7
1	3	11	-	-	-	-
		5	12	14		
			7	9	15	

實際內容則是

1	2	3	4	5	6	7
1	3	5	7	9	15	-