The University of Newcastle
School of Information and Physical Sciences

COMP2230/6230 Algorithms

Tutorial Week 9 Solutions

13 – 17 September 2021

Tutorial

1. Write a dynamic programming algorithm for computing the n^{th} Fibonacci number *f(n)*. Trace your algorithm for $n = 7$. Compare the time complexity of your algorithm to the time complexity of a recursive algorithm for computing the n^{th} Fibonacci number *f(n)*. Refine your algorithm so that it does not use extra space. Trace the refined algorithm for $n = 7$.

Solution:
The dynamic programming algorithm for computing the n^{th} Fibonacci number *f(n)* fills elements of a one-dimensional array with *n* consecutive Fibonacci numbers, starting from *f(1)* to *f(n)*.

```
Input Parameters: n
Output Parameters: None
fibonacci1(n) {
    // f is a local array
    if (n = = 1 || n = = 2)
            return 1
    f[1] = 1
    f[2] = 1
    for i = 3 to n
            f[i] = f[i - 1] + f[i - 2]
    return f[n]
    }
```

Time complexity of this algorithm is $\Theta(n)$.
Tracing the algorithm for $n = 7$:

1						
1	**1**					
1	1	**2**				
1	1	2	**3**			
1	1	2	3	**5**		
1	1	2	3	5	**8**	

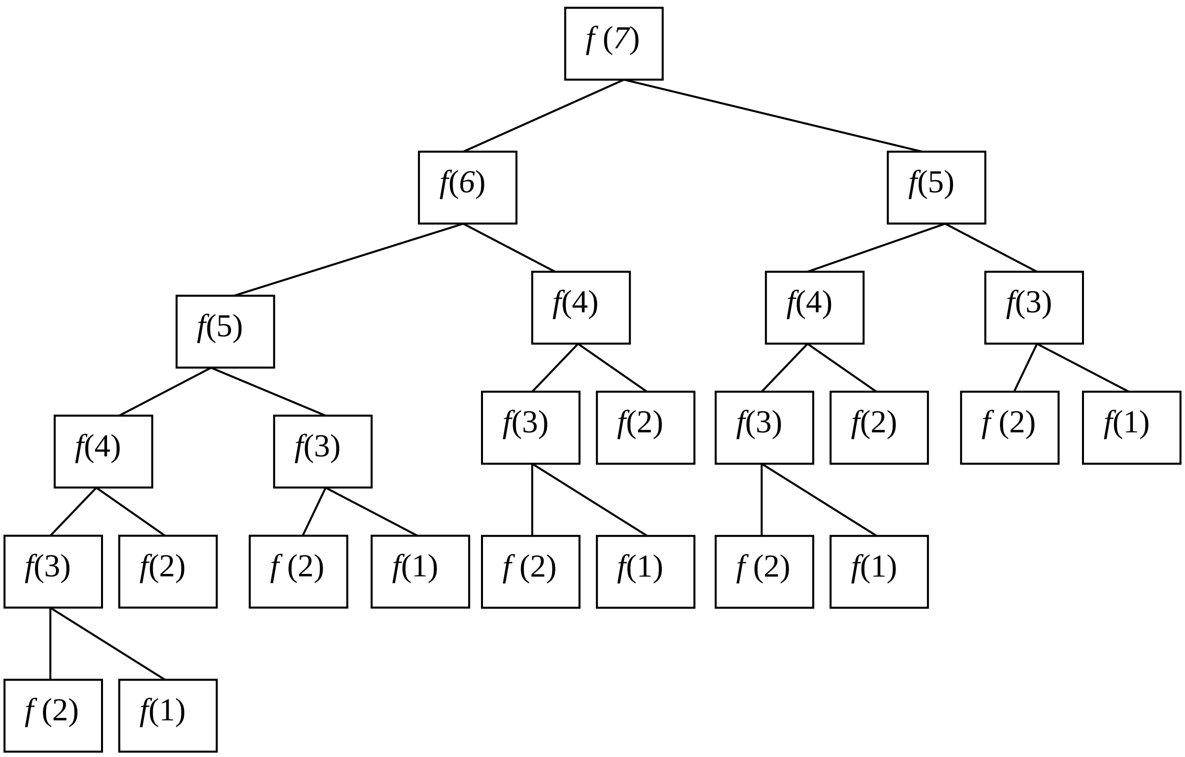
1	1	2	3	5	8	**13**

Recursive algorithm:

Input Parameters: *n*
Output Parameters: None

```
fibonacci_recurs(n) {
        if (n = = 1 || n = = 2)
                return 1
        return fibonacci_recurs(n - 2) + fibonacci_recurs(n - 1)
}
```

Time complexity of this algorithm is $\Theta(c^n)$.
Tracing the algorithm for $n = 7$:



Algorithm refined to save the space:

Input Parameters: *n*
Output Parameters: None

```
fibonacci2(n) {
        if (n == 1 || n == 2)
                return 1
        f_twoback = 1
```

```
        f_oneback = 1
        for i = 3 to n {
                f = f_twoback + f_oneback
                f_twoback = f_oneback
                f_oneback = f
        }
        return f
}
```

Time complexity of this algorithm is $\Theta(n)$, assuming that addition requires constant time (strictly speaking this is not the case).

Tracing the algorithm for $n = 7$:

f_twoback	*f_oneback*	*f*
1		
1	**1**	
1	1	**2**
1	**2**	2
1	2	**3**
2	**3**	3
2	3	**5**
3	**5**	5
3	5	**8**
5	**8**	8
5	8	**13**

2. Write a dynamic programming algorithm for computing binomial coefficient $C(n,k) = \binom{n}{k}$. What is the complexity of your algorithm? Trace the algorithm for *C(5,3).*

Solution:

Binomial coefficients are the coefficients in the binomial formula:
$(a + b)^n = C(n,0)a^n + \ldots + C(n,k)a^{n-k}b^k + \ldots + C(n,n)b^n$

We shall use the following recurrence relation:
$C(n,k) = C(n-1,k-1) + C(n-1,k)$, *for* $n>k>0$; $C(n,0) = C(n,n) = 1$

Algorithm:

//Input: A pair of nonnegative integers $n \geq k \geq 0$.
//Output: None

```
for i = 0 to n
    for j = 0 to min(i,k)
        if j=0 or j=i
            C[i,j] = 1
        else C[i,j] = C[i-1,j-1] + C[i-1,j]
 return C[n,k]
```

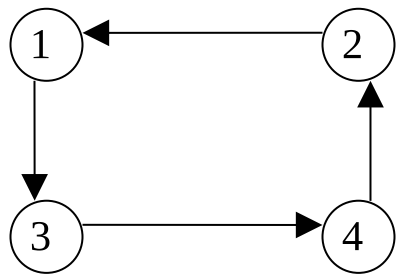
The time complexity is $(k+1)k/2 + k(n-k+1) = \Theta(nk)$ (again assuming that addition requires constant time).

Tracing the algorithm for $C(5,3)$:

	0	*1*	*2*	*3*
0	1			
1	1	1		
2	1	2	1	
3	1	3	3	1
4	1	4	6	4
5	1	5	10	10

3. The following is a pseudo code of Warshall's algorithm for computing the transitive closure of a digraph, where the transitive closure of a directed graph with n vertices is an $n \times n$ Boolean matrix TC such that TC(i,j) is 1 if there is a directed path from vertex i to vertex j, and 0 otherwise. Trace the algorithm for the digraph below.

//Input: The adjacency matrix A of a digraph D with n vertices.
//Output: None
$TC^{(0)} = A$
for $k=1$ to n
    for $i=1$ to n
        for $j=1$ to n
            $TC^{(k)}[i,j] = TC^{(k-1)}[i,j]$ or $(TC^{(k-1)}[i,k]$ and $TC^{(k-1)}[k,j])$
return $TC^{(n)}$



Solution:

Warshall's algorithm constructs a series of n×n matrices $TC^{(0)}, TC^{(1)}, \ldots, TC^{(n)}$. $TC^{(0)}$ is equal to the adjacency matrix A and $t_{ij}^{(0)}$ is 1 if and only if there is a directed edge from vertex i to vertex j. An element of $TC^{(1)}$, $t_{ij}^{(1)}$, is equal to 1 if and only if there is a directed path from i to vertex j, which can only contain vertex 1 as an intermediate vertex. In general, an element of $TC^{(k)}$, $t_{ij}^{(k)}$, is equal to 1 if and only if there is a directed path from i to vertex j, which can only contain vertices 1 to k as intermediate vertices. Finally, in $TC^{(n)}$ the element $t_{ij}^{(n)}$ is equal to 1 if and only if there is a directed path from i to vertex j, and that path can use all n vertices as intermediate vertices.

$$A = \begin{matrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{matrix}$$

$$TC^{(0)} = \begin{matrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{matrix} \qquad TC^{(1)} = \begin{matrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{matrix}$$

$$TC^{(2)} = \begin{matrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{matrix} \qquad TC^{(3)} = \begin{matrix} 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{matrix} \qquad TC^{(4)} = \begin{matrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{matrix}$$

4. Trace Floyd's algorithm for the following digraph.

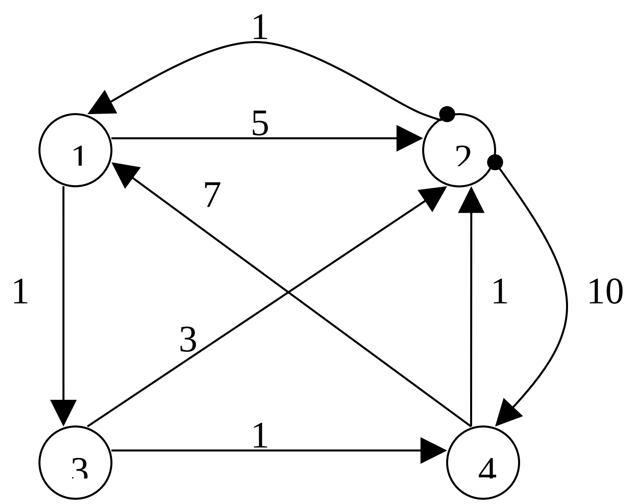
$$A = \begin{matrix} 0 & 5 & 1 & \infty \\ 1 & 0 & \infty & 10 \\ \infty & 3 & 0 & 1 \\ 7 & 1 & \infty & 0 \end{matrix}$$

Solution:

Floyd's algorithm finds the shortest paths between each pair of vertices in a graph. Floyd's algorithm can be applied to any directed or undirected graph that does not have a cycle of negative length. The output of the algorithm is the so-called distance matrix *D* where the element d_{ij} is the length of the shortest path from the vertex *i* to the vertex *j*.

Floyd's algorithm constructs D through a series of matrices $D^{(0)}$, $D^{(1)}$, . . . , $D^{(n)}$, where $D^{(0)}$ is equal to the matrix A. The element $d_{ij}^{(k)}$ of matrix $D^{(k)}$ is equal to the length of the shortest path from vertex i to vertex j, among all paths from vertex i to vertex j that can only contain vertices 1 to k as intermediate vertices.

//Input: The adjacency matrix A of a weighted digraph G with n vertices an no negative cycles.
//Output: None
$D^{(0)} = A$
for $k=1$ to n
    for $i=1$ to n
        for $j=1$ to n
            $D^{(k)}[i,j] = \min(D^{(k-1)}[i,j], D^{(k-1)}[i,k] + D^{(k-1)}[k,j])$
return $D^{(n)}$

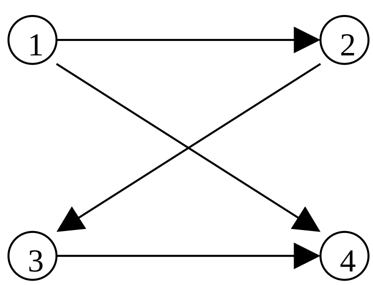


$$D^{(0)} = A = \begin{matrix} 0 & 5 & 1 & \infty \\ 1 & 0 & \infty & 10 \\ \infty & 3 & 0 & 1 \\ 7 & 1 & \infty & 0 \end{matrix} \qquad D^{(1)} = A = \begin{matrix} 0 & 5 & 1 & \infty \\ 1 & 0 & 2 & 10 \\ \infty & 3 & 0 & 1 \\ 7 & 1 & 8 & 0 \end{matrix} \qquad D^{(2)} = A = \begin{matrix} 0 & 5 & 1 & 15 \\ 1 & 0 & 2 & 10 \\ 4 & 3 & 0 & 1 \\ 2 & 1 & 3 & 0 \end{matrix}$$

$$D^{(3)} = A = \begin{matrix} 0 & 4 & 1 & 2 \\ 1 & 0 & 2 & 3 \\ 4 & 3 & 0 & 1 \\ 2 & 1 & 3 & 0 \end{matrix} \qquad D^{(4)} = A = \begin{matrix} 0 & 3 & 1 & 2 \\ 1 & 0 & 2 & 3 \\ 3 & 2 & 0 & 1 \\ 2 & 1 & 3 & 0 \end{matrix}$$

Workshop

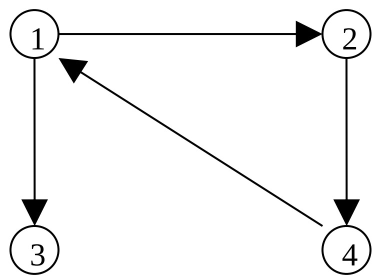
5. Trace Warshall's algorithm on the diagraphs below.



Solution:

$$TC^{(0)} = A = \begin{matrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{matrix} \qquad TC^{(1)} = \begin{matrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{matrix} \qquad TC^{(2)} = \begin{matrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{matrix}$$

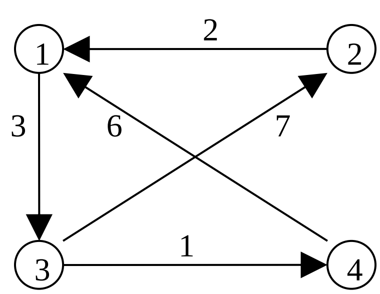
$$TC^{(3)} = \begin{matrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{matrix} \qquad TC^{(4)} = \begin{matrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{matrix}$$



$$TC^{(0)} = A = \begin{matrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{matrix} \qquad TC^{(1)} = \begin{matrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{matrix} \qquad TC^{(2)} = \begin{matrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{matrix}$$

$$TC^{(3)} = \begin{matrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{matrix} \qquad TC^{(4)} = \begin{matrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{matrix}$$

6. Trace Floyd's algorithm for the following digraph.



Solution:

$$D^{(0)} = A = \begin{matrix} 0 & \infty & 3 & \infty \\ 2 & 0 & \infty & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \infty & 0 \end{matrix} \qquad D^{(1)} = \begin{matrix} 0 & \infty & 3 & \infty \\ 2 & 0 & 5 & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & 9 & 0 \end{matrix} \qquad D^{(2)} = \begin{matrix} 0 & \infty & 3 & \infty \\ 2 & 0 & 5 & \infty \\ 9 & 7 & 0 & 1 \\ 6 & \infty & 9 & 0 \end{matrix}$$

$$D^{(3)} = \begin{matrix} 0 & 10 & 3 & 4 \\ 2 & 0 & 5 & 6 \\ 9 & 7 & 0 & 1 \\ 6 & 16 & 9 & 0 \end{matrix} \qquad D^{(4)} = \begin{matrix} 0 & 10 & 3 & 4 \\ 2 & 0 & 5 & 6 \\ 7 & 7 & 0 & 1 \\ 6 & 16 & 9 & 0 \end{matrix}$$

7. Write a pseudo code of a dynamic programming algorithm for Knapsack problem and trace it on the following instance: $a_1(w_1 = 2, v_1 = 5)$, $a_2(w_2 =3, v_2 = 8)$, $a_3(w_3 =1, v_3 = 7)$, $a_4(w_4 =2, v_4 = 15)$ and W=5.

Knapsack Problem: Given n objects and a knapsack of capacity W, where the object a_i has weight w_i and value v_i, find the maximum value that fit into knapsack. NOTE: There are different versions of the Knapsack problem.

Solution:

```
//Input: Weights w1, w2, . . . , wn , values v1, v2, . . , vn and a knapsack capacity W.
//Output: None
for j=1 to W
    V[0,j] = 0
for i=1 to n
    for j=0 to W
        if j >=  wi
            V[i,j] = max (V[i-1,j], V[i-1,j-wi] + vi)
        else
            V[i,j] = V[i-1,j]
return V[n,W]
```

	a_1	a_2	a_3	a_4
w	2	3	1	2
v	5	8	7	15

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	5	5	5	5
2	0	0	5	8	8	13
3	0	7	7	12	15	15
4	0	7	15	22	22	37

8. Prove that when *Fibonacci_recurs* computes f_n , $n>=3$, f_n computations are required for the base cases.

Solution *(from the text):*
We use induction.

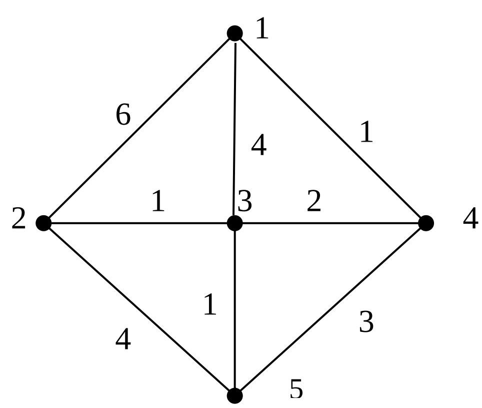
<u>The base cases:</u> n = 1,2; each of f_1 and f_2, compute base cases only once.

<u>Inductive hypothesis:</u> Suppose that when we execute *Fibonacci_recurs(k),* 0<k<n, the number of times that the base cases are computed is f_k .

<u>Inductive step:</u> When we execute *Fibonacci_recurs(n),*, we recursively invoke F*ibonacci_recurs(n-1) and Fibonacci_recurs(n-2).* By the inductive assumption, *Fibonacci_recurs(n-1)* invokes the base cases F_{n-1} times, and *Fibonacci_recurs(n-2)* invokes the base cases F_{n-2} times. Thus the total number of base cases computed $f_{n-1}+f_{n-2}=f_n$. The inductive step is complete.

More exercise

9. Trace Floyd's algorithm for the following graph.



Solution:

$$D^{(0)} = A = \begin{matrix} 0 & 6 & 4 & 1 & \infty \\ 6 & 0 & 1 & \infty & 4 \\ 4 & 1 & 0 & 2 & 1 \\ 1 & \infty & 2 & 0 & 3 \\ \infty & 4 & 1 & 3 & 0 \end{matrix} \qquad D^{(1)} = \begin{matrix} 0 & 6 & 4 & 1 & \infty \\ 6 & 0 & 1 & 7 & 4 \\ 4 & 1 & 0 & 2 & 1 \\ 1 & 7 & 2 & 0 & 3 \\ \infty & 4 & 1 & 3 & 0 \end{matrix}$$

$$D^{(2)} = \begin{matrix} 0 & 6 & 4 & 1 & 10 \\ 6 & 0 & 1 & 7 & 4 \\ 4 & 1 & 0 & 2 & 1 \\ 1 & 7 & 2 & 0 & 3 \\ 10 & 4 & 1 & 3 & 0 \end{matrix} \qquad D^{(3)} = \begin{matrix} 0 & 5 & 4 & 1 & 5 \\ 5 & 0 & 1 & 3 & 2 \\ 4 & 1 & 0 & 2 & 1 \\ 1 & 3 & 2 & 0 & 3 \\ 5 & 2 & 1 & 3 & 0 \end{matrix}$$

$$D^{(4)} = \begin{matrix} 0 & 4 & 3 & 1 & 4 \\ 4 & 0 & 1 & 3 & 2 \\ 3 & 1 & 0 & 2 & 1 \\ 1 & 3 & 2 & 0 & 3 \\ 4 & 2 & 1 & 3 & 0 \end{matrix} \qquad D^{(5)} = \begin{matrix} 0 & 4 & 3 & 1 & 4 \\ 4 & 0 & 1 & 3 & 2 \\ 3 & 1 & 0 & 2 & 1 \\ 1 & 3 & 2 & 0 & 3 \\ 4 & 2 & 1 & 3 & 0 \end{matrix}$$

10. There are six permutations of Floyds algorithm of the lines

```
for k = 1 to n
    for i = 1 to n
        for j = 1 to n
```

Which ones give a correct algorithm?

Solution:

Only lines *for i = 1 to n* and *for j = 1 to n* can be swapped; other 4 permutations do not produce a correct algorithm.

11. Suppose that G is directed, weighted graph in which some weights are negative. Write an algorithm that determines whether G contains a cycle of negative weight.

Solution idea: Use depth first search.

12. Explain why Warshall's algorithm can compute the matrices $A^{(k)}$ in place? What is the running time of Warshall's algorithm?

Solution idea:

$A^{(k)}(i,k) = A^{(k-1)}(i,k)$
$A^{(k)}(k,j) = A^{(k-1)}(k,j)$
Running time is $\Theta(n^3)$.