Tutorial 6

Analysis and Design of Algorithms

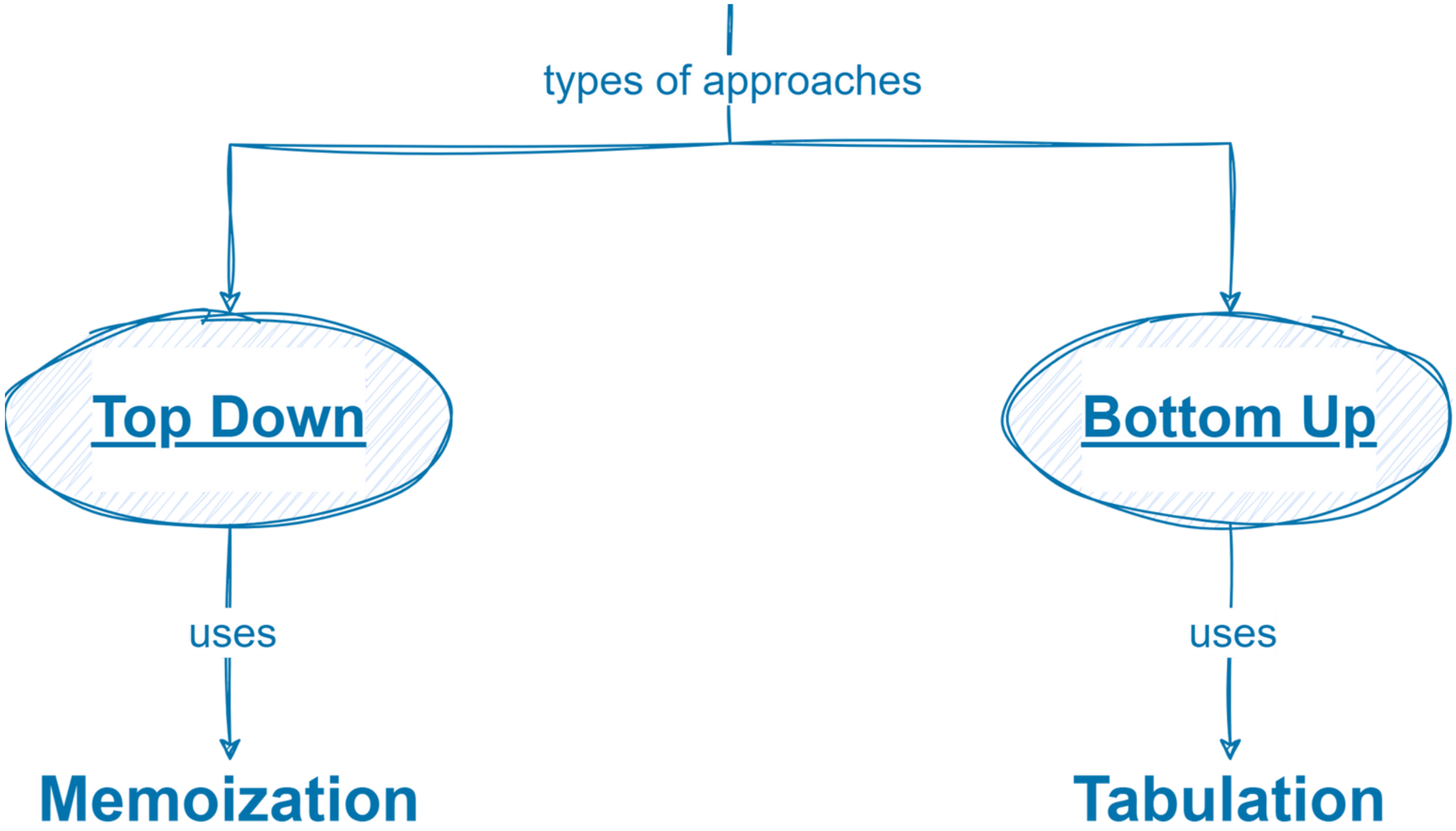
Dynamic Programming I

Dynamic Programming

- It is a technique for solving a complex problem by first **dividing it into sub-problems,** solving each of them **just once,** and then **storing their solutions** to avoid repetitive computations.

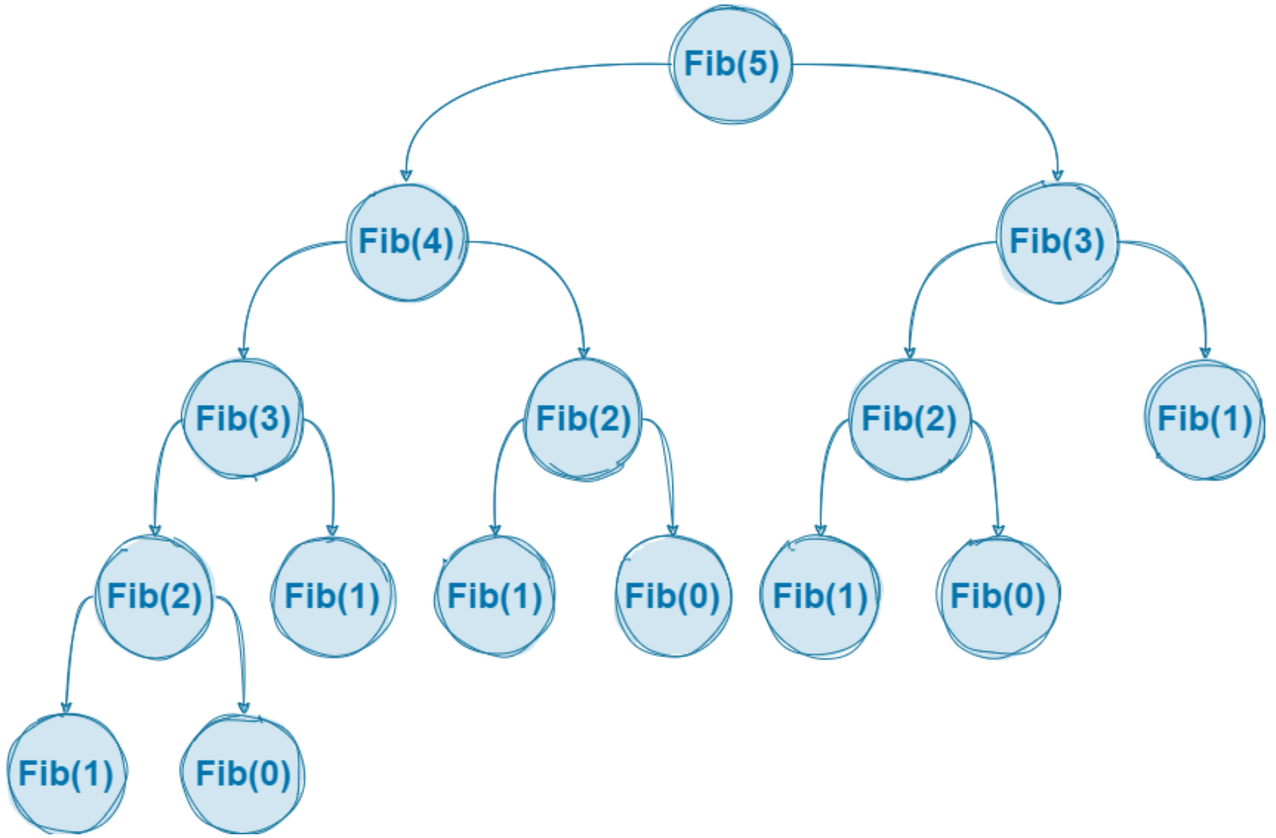
- This means that we need to have **overlapping sub-problems** in order for this design paradigm to be useful.

Dynamic Programming



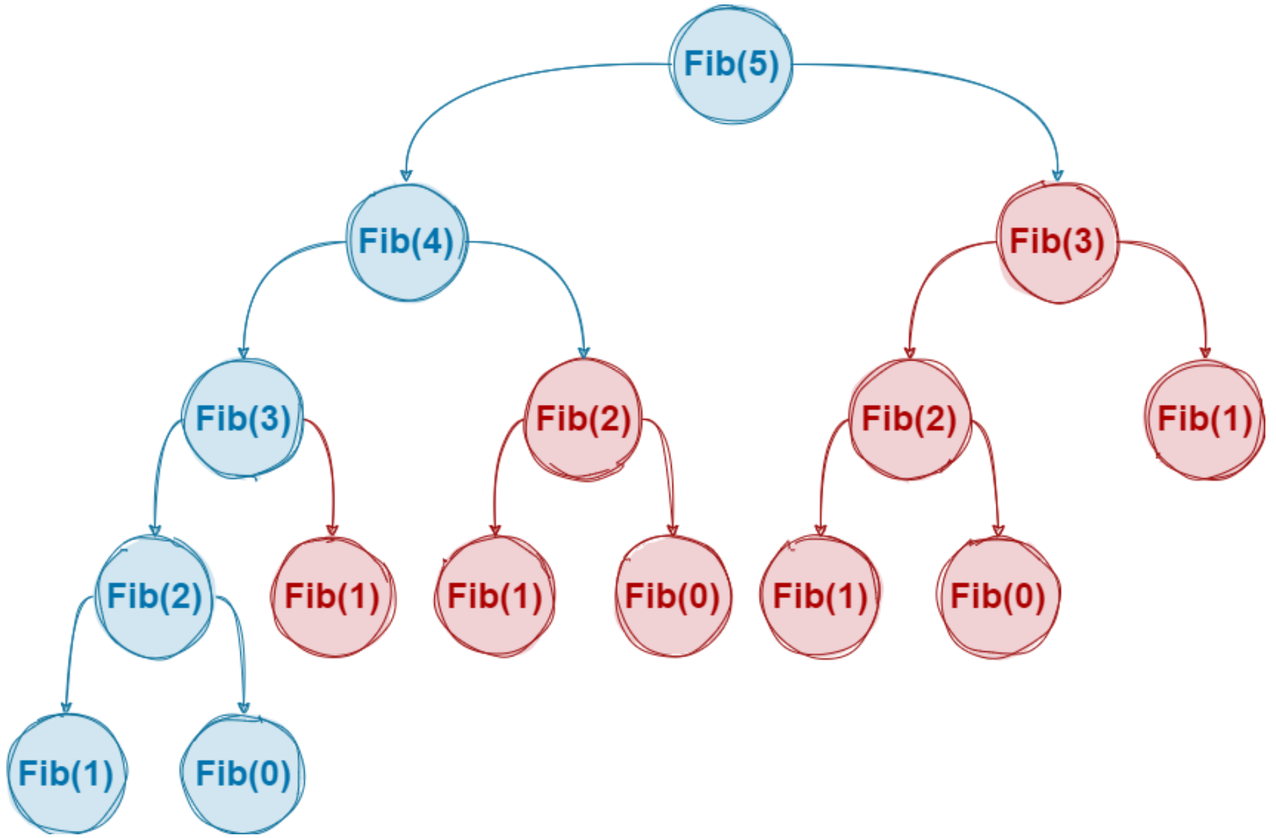
Fibonacci D&C

```
1  Fib(n)
2  if n == 0 then
3  |   return 0;
4  else
5  |   if n ≤ 2 then
6  |   |   return 1;
7  |   else
8  |   |   return Fib(n − 1) + Fib(n − 2);
9  |   end
10 end
```



Fibonacci D&C → DP

```
1  Fib(n)
2  if n == 0 then
3  |   return 0;
4  else
5  |   if n ≤ 2 then
6  |   |   return 1;
7  |   else
8  |   |   return Fib(n − 1) + Fib(n − 2);
9  |   end
10 end
```

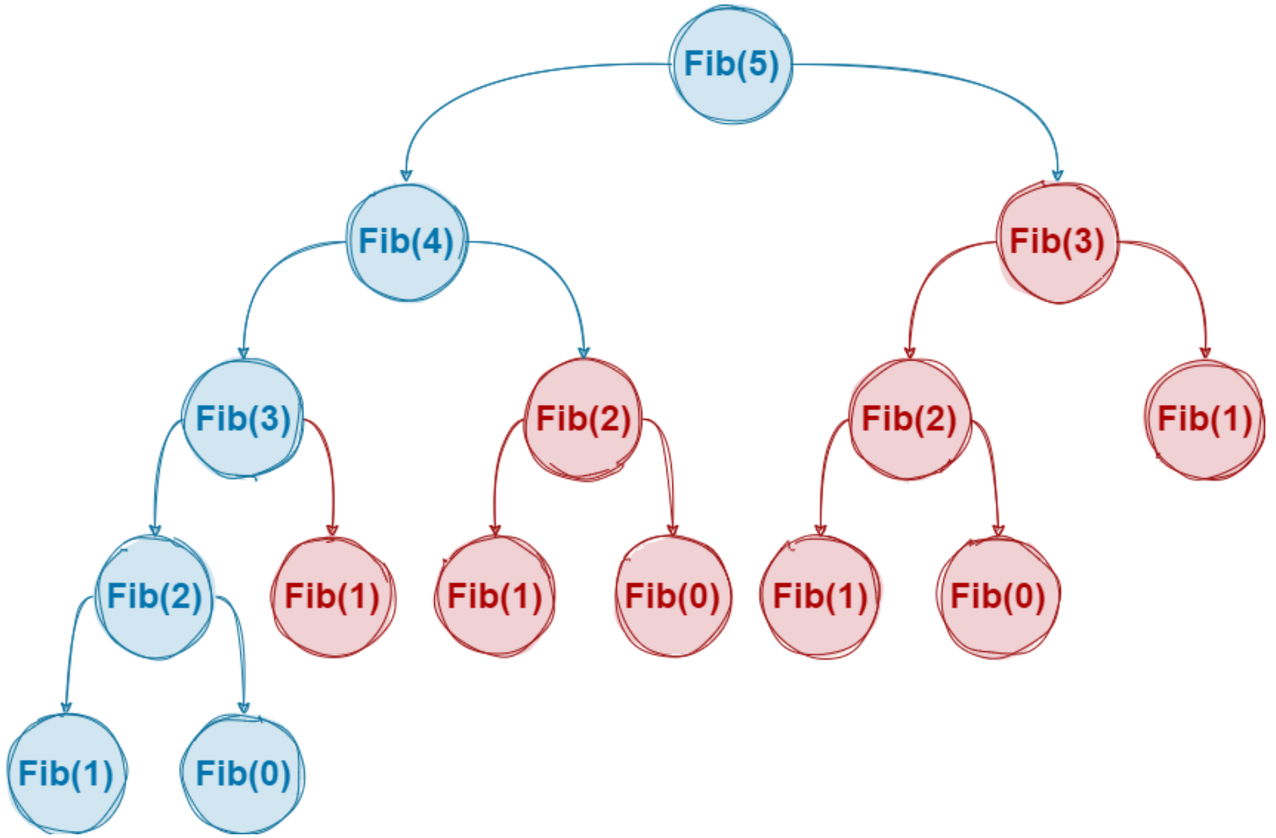


Fibonacci D&C → DP

```
1  Fib(n)
2  if n == 0 then
3  |   return 0;
4  else
5  |   if n ≤ 2 then
6  |   |   return 1;
7  |   else
8  |   |   return Fib(n − 1) + Fib(n − 2);
9  |   end
10 end
```

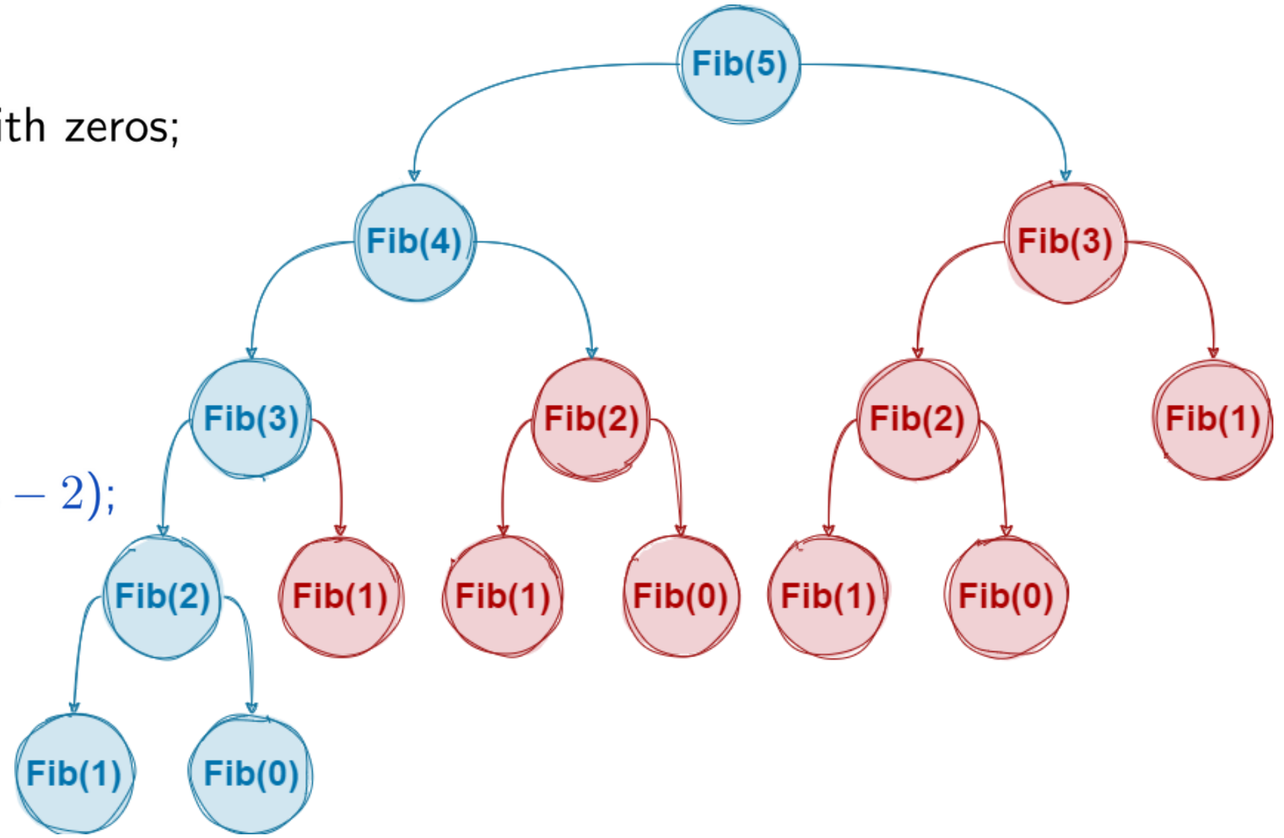
1) Place in storage instead of returning

2) Check if available in storage before doing the recursive call



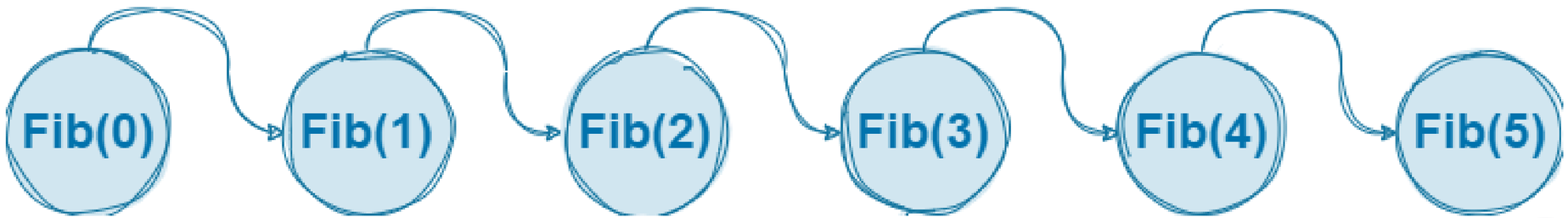
Fibonacci Top-Down (Memoization)

```
1  a = empty array of integers of size n + 1 initialized with zeros;
2  Fib-DP-Mem(n)
3  if n ≤ 2 and n! = 0 then
4  |   a[n] = 1;
5  else
6  |   if a[n] == 0 and n! = 0 then
7  |   |   a[n] = Fib-DP-Mem(n − 1) + Fib-DP-Mem(n − 2);
8  |   end
9  end
10 return a[n];
```



Fibonacci Bottom Up (Tabulation)

```
1 Fib-DP-Tab(n)
2 a=empty array of integers of size n + 1 initialized with zeros
3 a[1] = 1;
4 a[2] = 1;
5 for i = 3 to n do
6 |   a[i] = a[i − 1] + a[i − 2] ;
7 end
8 return a[n];
```



Exercise 6-3

Consider the following recursive definition for the problem of calculating a given binomial coefficient $\binom{n}{k}$:

$$\binom{n}{k} = \begin{cases} \binom{n-1}{k-1} + \binom{n-1}{k} & \text{if } 0 < k < n; \\ 1 & \text{if } k = 0 \, or k = n. \end{cases}$$

i. Write a divide-and-conquer algorithm to compute the binomial coefficients.

ii. Provide the recurrence relation for your answer in part (i).

ii. Write an iterative algorithm that employs the dynamic programming technique using the bottom-up approach.

iv. Write an algorithm for the same problem using the memoization technique.

v. Modify your answer in (iii) to use a single one-dimensional array indexed from 0 to k

Exercise 6-3

Consider the following recursive definition for the problem of calculating a given binomial coefficient $\binom{n}{k}$:

$$\binom{n}{k} = \begin{cases} \binom{n-1}{k-1} + \binom{n-1}{k} & \text{if } 0 < k < n; \\ 1 & \text{if } k = 0 \, or k = n. \end{cases}$$

i. Write a divide-and-conquer algorithm to compute the binomial coefficients.

Exercise 6-3

Consider the following recursive definition for the problem of calculating a given binomial coefficient $\binom{n}{k}$:

$$\binom{n}{k} = \begin{cases} \binom{n-1}{k-1} + \binom{n-1}{k} & \text{if } 0 < k < n; \\ 1 & \text{if } k = 0 \, or k = n. \end{cases}$$

ii. Provide the recurrence relation for your answer in part (i).

Exercise 6-3

Consider the following recursive definition for the problem of calculating a given binomial coefficient $\binom{n}{k}$:

$$\binom{n}{k} = \begin{cases} \binom{n-1}{k-1} + \binom{n-1}{k} & \text{if } 0 < k < n; \\ 1 & \text{if } k = 0 \, or k = n. \end{cases}$$

iv. Write an algorithm for the same problem using the memoization technique.

Exercise 6-3

Consider the following recursive definition for the problem of calculating a given binomial coefficient $\binom{n}{k}$:

$$\binom{n}{k} = \begin{cases} \binom{n-1}{k-1} + \binom{n-1}{k} & \text{if } 0 < k < n; \\ 1 & \text{if } k = 0 \, or k = n. \end{cases}$$

ii. Write an iterative algorithm that employs the dynamic programming technique using the bottom-up approach.

Exercise 6-3

Consider the following recursive definition for the problem of calculating a given binomial coefficient $\binom{n}{k}$:

$$\binom{n}{k} = \begin{cases} \binom{n-1}{k-1} + \binom{n-1}{k} & \text{if } 0 < k < n; \\ 1 & \text{if } k = 0 \, or k = n. \end{cases}$$

v. Modify your answer in (iii) to use a single one-dimensional array indexed from 0 to k

Exercise 6-4

Suppose that you are given an $n \times n$ checkerboard and a checker. You must move the checker from the bottom edge of the board to the top edge of the board according to the following rule. At each step you may move the checker to one of three squares:

a) the square immediately above,

b) the square that is one up and one to the left (but only if the checker is not already in the leftmost column),

c) the square that is one up and one to the right (but only if the checker is not already in the rightmost column).

Each time you move from square x to square y, you receive $p(x, y)$ dollars. You are given $p(x, y)$ for all pairs (x, y) for which a move from x to y is legal. Do not assume that $p(x, y)$ is positive. Give an algorithm that figures out the set of moves that will move the checker from somewhere along the bottom edge to somewhere along the top edge while gathering as many dollars as possible. Your algorithm is free to pick any square along the bottom edge as a starting point and any square along the top edge as a destination in order to maximize the number of dollars gathered along the way. What is the recursive definition for calculating the score?

a) the square immediately above,

b) the square that is one up and one to the left (but only if the checker is not already in the leftmost column),

c) the square that is one up and one to the right (but only if the checker is not already in the rightmost column).

All done!