Advanced Algorithms & Data Structures

Department of CSE

ADVANCED ALGORITHMS AND DATA STRUCTURES
23CS03HF

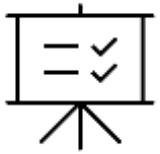
Topic:

Dynamic Programming

AIM OF THE SESSION

To familiarize students with the concept of Dynamic Programming.

INSTRUCTIONAL OBJECTIVES



This Session is designed to:

1. Demonstrate :- Dynamic programming and its elements.
2. Describe :- Steps that the dynamic programming follows.

LEARNING OUTCOMES

At the end of this session, you should be able to:

1. Define :- Dynamic programming and its elements.
2. Describe :- Principles of optimality
3. Summarize:- top-down and bottom-up approaches of dynamic programming.

Dynamic Programming

- Dynamic programming is a technique that breaks the problems into sub-problems, and saves the result for future purposes so that we do not need to compute the result again.
- The main use of dynamic programming is to solve optimization problems. Here, optimization problems mean that when we are trying to find out the minimum or the maximum solution of a problem.
- The dynamic programming guarantees to find the optimal solution of a problem if the solution exists.

Elements of Dynamic Programming

DP is used to solve problems with the following characteristics:

- **Simple subproblems**
  - We should be able to break the original problem to smaller subproblems that have the same structure

- **Optimal substructure of the problems**
  - The optimal solution to the problem contains within optimal solutions to its subproblems.

- **Overlapping sub-problems**
  - It is mainly used where the solution of one sub-problem is needed repeatedly. The computed solutions are stored in a table, so that these don't have to be re-computed. Hence, this technique is needed where overlapping sub-problem exists.

Principle of optimality

- **The dynamic Programming works on a principle of optimality.**
- **The principle of optimality states that each sub-sequence must also be optimal in an optimal sequence of decisions or choices.**

Example

Consider an example of the Fibonacci series. The following series is the Fibonacci series:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ,…

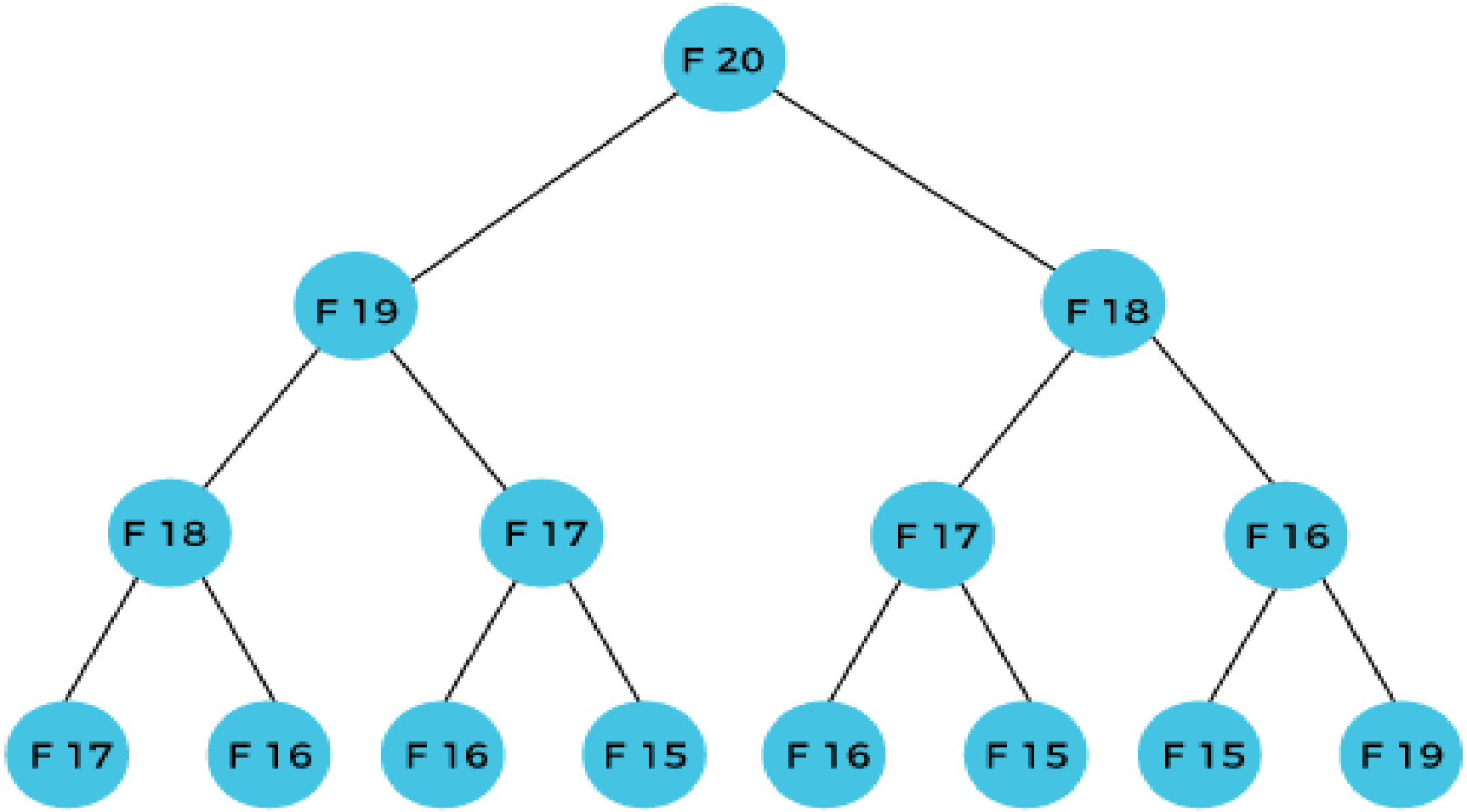
The numbers in the above series are not randomly calculated.

Mathematically, we could write each of the terms using the below formula:

F(n) = F(n-1) + F(n-2),

How can we calculate F(20)?

The F(20) term will be calculated using the nth formula of the Fibonacci series. The below figure shows how F(20) is calculated.



Approach of work

The following are the steps that the dynamic programming follows:

- It breaks down the complex problem into simpler subproblems.
- It finds the optimal solution to these sub-problems.
- It stores the results of subproblems (memorization). The process of storing the results of subproblems is known as memorization.
- It reuses them so that the same sub-problem is calculated more than once.
- Finally, calculate the result of the complex problem.

Approach of DP

There are two approaches to dynamic programming:

1. Top-down approach
2. Bottom-up approach

Top-Down approach

The top-down approach follows the memorization technique, while the bottom-up approach follows the tabulation method. Here memorization is equal to the sum of recursion and caching. Recursion means calling the function itself, while caching means storing the intermediate results.

Advantages

- It is very easy to understand and implement.
- It solves the subproblems only when it is required.
- It is easy to debug.

Disadvantages

- It uses the recursion technique that occupies more memory in the call stack. Sometimes when the recursion is too deep, the stack overflow condition will occur.
- It occupies more memory degrades the overall performance

```
int fib(int n)
{
   if(n<0)
   error;
 if(n==0)
 return 0;
 if(n==1)
return 1;
sum = fib(n-1) + fib(n-2);
}
```

In the above code, we have used the recursive approach to find out the Fibonacci series. When the value of 'n' increases, the function calls will also increase, and computations will also increase. In this case, the time complexity increases exponentially, and it becomes 2^n.

One solution to this problem is to use the dynamic programming approach. Rather than generating the recursive tree again and again, we can reuse the previously calculated value. If we use the dynamic programming approach, then the time complexity would be O(n).

Bottom-Up approach

Bottom-Up approach

- The bottom-up approach is also one of the techniques which can be used to implement dynamic programming.
- It uses the tabulation technique to implement the dynamic programming approach. It solves the same kind of problems but it removes the recursion.
- In this tabulation technique, we solve the problems and store the results in a matrix.
- The bottom-up is the approach used to avoid the recursion, thus saving the memory space.
- In the bottom-up approach, we start from the base case to find the answer for the end. As we know, the base cases in the Fibonacci series are 0 and 1. Since the bottom approach starts from the base cases, so we will start from 0 and 1.

Key points

- **We solve all the smaller sub-problems that will be needed to solve the larger sub-problems then move to the larger problems using smaller sub-problems.**
- **We use a for loop to iterate over the sub-problems.**
- **The bottom-up approach is also known as the tabulation or table-filling method.**

Let's understand through an example.
Suppose we have an array that has 0 and 1 values at a[0] and a[1] positions, respectively shown as below:

0	1
a[0]	a[1]

Since the bottom-up approach starts from the lower values, so the values at a[0] and a[1] are added to find the value of a[2] shown as below:

0	1	1
a[0]	a[1]	a[2]

The value of a[3] will be calculated by adding a[1] and a[2], and it becomes 2 shown as below:

0	1	1	2
a[0]	a[1]	a[2]	a[3]

The value of a[4] will be calculated by adding a[2] and a[3], and it becomes 3 shown as below:

0	1	1	2	3
A[0]	A[1]	A[2]	A[3]	A[4]

The value of a[5] will be calculated by adding the values of a[4] and a[3], and it becomes 5 shown as below:

0	1	1	2	3	5
A[0]	A[1]	A[2]	A[3]	A[4]	A[5]

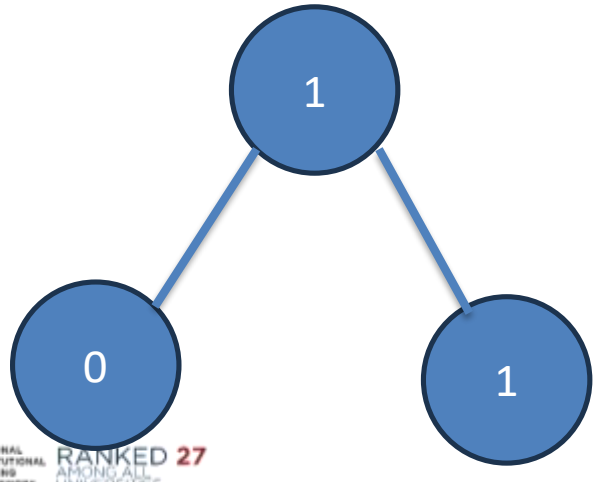
The code for implementing the Fibonacci series using the bottom-up approach is given below:

```
int fib(int n)
{
   int A[];
   A[0] = 0, A[1] = 1;
   for( i=2; i<=n; i++)
   {
       A[i] = A[i-1] + A[i-2]
   }
   return A[n];
}
```

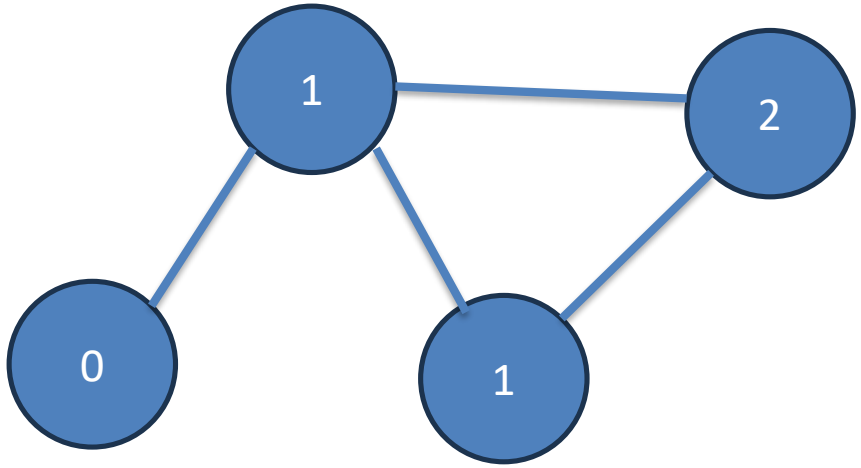
In the above code, base cases are 0 and 1 and then we have used for loop to find other values of Fibonacci series.

Let's understand through the diagrammatic representation.
Initially, the first two values, i.e., O and 1 can be represented as:

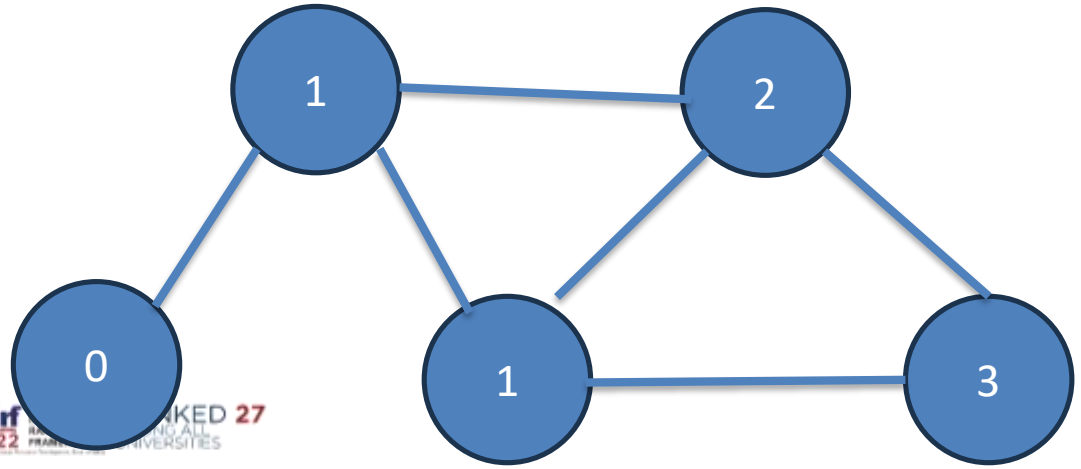
When i=2 then the values 0 and 1 are added shown as below:



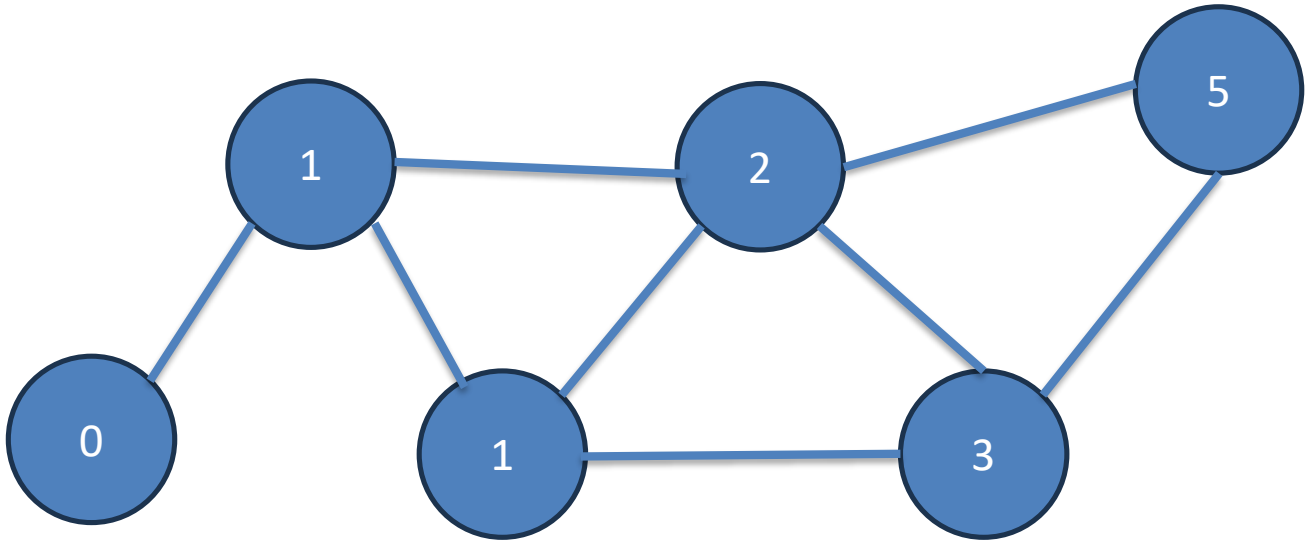
When i=3 then the values of 1 and 1 are added shown as below:



When i=4 then the values 2 and 1 are added shown as below:



When i=5 then the values of 3 and 2 are added shown as below:



In the above case, we are starting from the bottom and reaching to the top.

DP vs Divide & Conquer

Divide & Conquer

1. Partitions a problem into independent smaller sub-problems
2. Doesn't store solutions of sub-problems. (Identical sub-problems may arise results in the same computations being performed repeatedly)
3. Top-down algorithms: which logically progresses from the initial instance down to the smallest sub-instances via intermediate sub-instances.

Dynamic Programming

1. Partitions a problem into overlapping sub-problems
2. Stores solutions of sub-problems: thus avoids calculations of same quantity twice
3. Bottom-up algorithms: in which the smallest sub-problems are explicitly solved first and the results of these are used to construct solutions to progressively larger sub-instances

DP vs Greedy method

Dynamic Programming

1. Dynamic Programming is used to obtain the optimal solution.
2. In Dynamic Programming, we choose at each step, but the choice may depend on the solution to sub-problems.
3. It is guaranteed that Dynamic Programming will generate an optimal solution using the Principle of Optimality.
4. Example: 0/1 Knapsack

Greedy Method

1. Greedy Method is also used to get the optimal solution.
2. In a greedy Algorithm, we make whatever choice seems best at the moment and then solve the sub-problems arising after the choice is made.
3. In the Greedy Method, there is no such guarantee of getting Optimal Solution
4. Example: Fractional Knapsack

Examples

- 0/1 Knapsack Method
- Traveling sales person Problem
- Optimal Binary search tree
- Matrix chain multiplication
- Longest Common Sequence

SUMMARY

- Dynamic programming is a technique that breaks the problems into sub-problems, and saves the result for future purposes so that we do not need to compute the result again.
- The main use of dynamic programming is to solve optimization problems. Here, optimization problems mean that when we are trying to find out the minimum or the maximum solution of a problem.
- The dynamic programming guarantees to find the optimal solution of a problem if the solution exists.

SELF-ASSESSMENT QUESTIONS

1. Which of the following is/are property/properties of a dynamic programming problem?

(a) Optimal substructure
(b) Overlapping subproblems
(c) Greedy approach
(d) Both optimal substructure and overlapping subproblems

2. If a problem can be solved by combining optimal solutions to non-overlapping problems, the strategy is called

(a) Dynamic programming
(b) Greedy
(c) Divide and conquer
(d) Recursion

TERMINAL QUESTIONS

1. What is the dynamic programming question?
2. Which problem is solved by dynamic programming?
3. What are the two methods of dynamic programming?
4. What are the applications of dynamic programming?
5. What are the characteristics of dynamic programming?

REFERENCES FOR FURTHER LEARNING OF THE SESSION

Reference Books :

1. Introduction to Algorithms, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein., 3rd, 2009, The MIT Press.
2 Algorithm Design Manual, Steven S. Skiena., 2nd, 2008, Springer.
3 Data Structures and Algorithms in Python, Michael T. Goodrich, Roberto Tamassia, and Michael H. Goldwasser., 2nd, 2013, Wiley.
4 The Art of Computer Programming, Donald E. Knuth, 3rd, 1997, Addison-Wesley Professiona.

MOOCS :

1. https://www.coursera.org/specializations/algorithms?=
2.https://www.coursera.org/learn/dynamic-programming-greedy-algorithms#modules

THANK YOU