* LMCR2303
Competitive Programming

Dynamic Programming (DP)

(leccturer: Nazatul Aini Abd Majid)

Learning outcome

* Able to link knowledge of data structures and algorithms in programming.
* Able to select the appropriate problem-solving method for computer science problems based on several categories of problems.
* Able to write effective programming code in solving computer science problems competitively.

*Introduction

Intelligent

*Introduction

Because it is a technique to reduce repetition works

*Introduction

So that it will save time and resources

*Definition

In computer science, mathematics, management science, economics and bioinformatics, **dynamic programming** (also known as **dynamic optimization**) is a method for solving a complex problem by breaking it down into a collection of simpler subproblems, solving each of those subproblems just once, and storing their solutions.

The next time the same subproblem occurs, instead of recomputing its solution, one simply looks up the previously computed solution, thereby saving computation time at the expense of a (hopefully) modest expenditure in storage space.

(Each of the subproblem solutions is indexed in some way, typically based on the values of its input parameters, so as to facilitate its lookup.) The technique of storing solutions to subproblems instead of recomputing them is called "memoization".

Features of DP

- The problem has optimal sub-structures
  - The solution of the sub-problem is part of the solution of the original problem.
- The problem has overlapping sub-problems.

*Example

*0, 1, 1, 2, 3, 5, 8, 13, . . .

Recognize this sequence?

It's the Fibonacci sequence, described by the recursive formula:

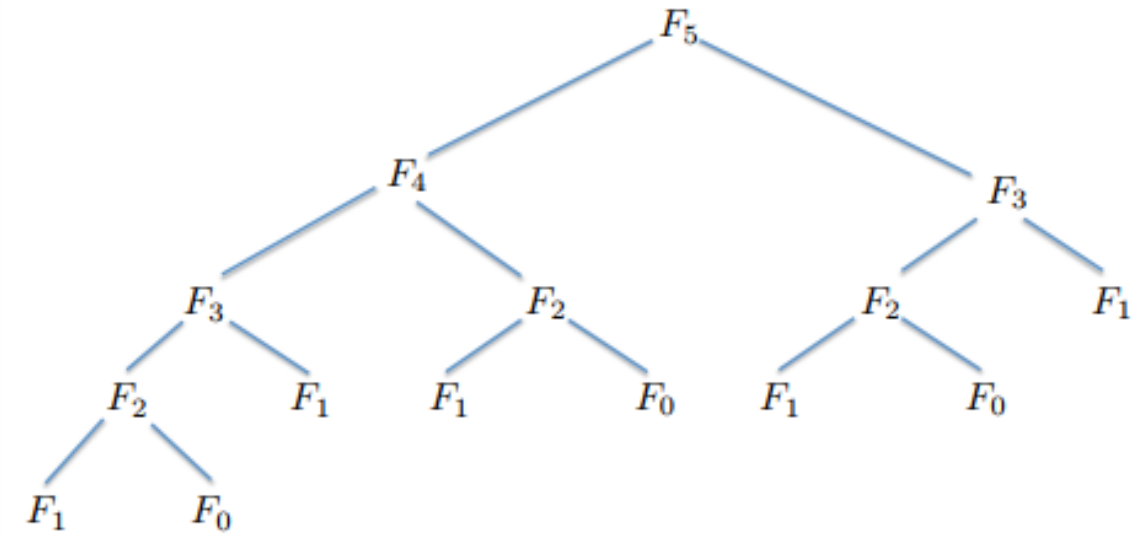
$F0 := 0; F1 := 1;$

$Fn = Fn-1 + Fn-2$, for all $n \geq 2$

Example

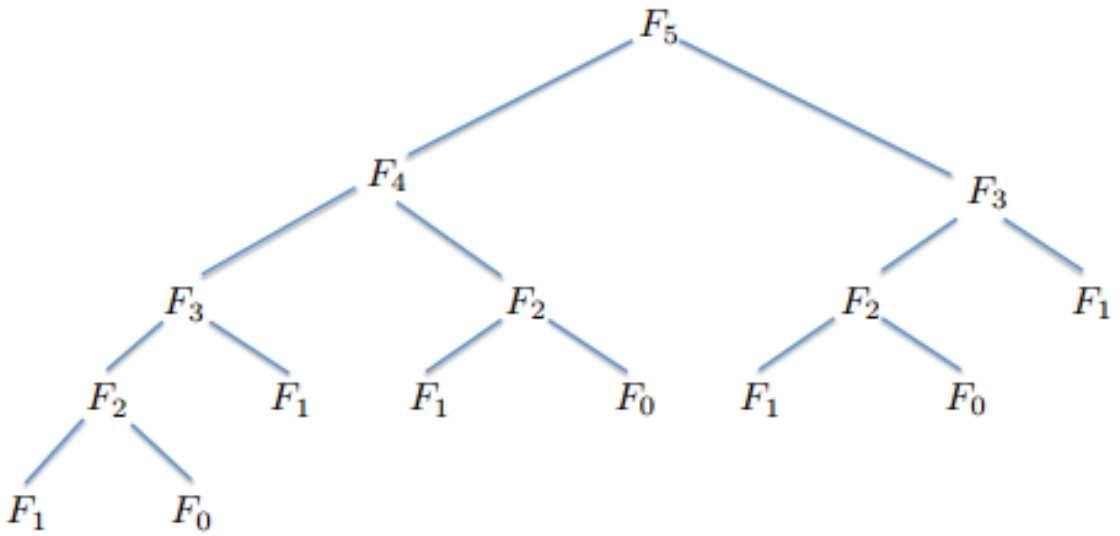
0, 1, 1, 2, 3, 5, 8, 13, . . .

Recognize this sequence?



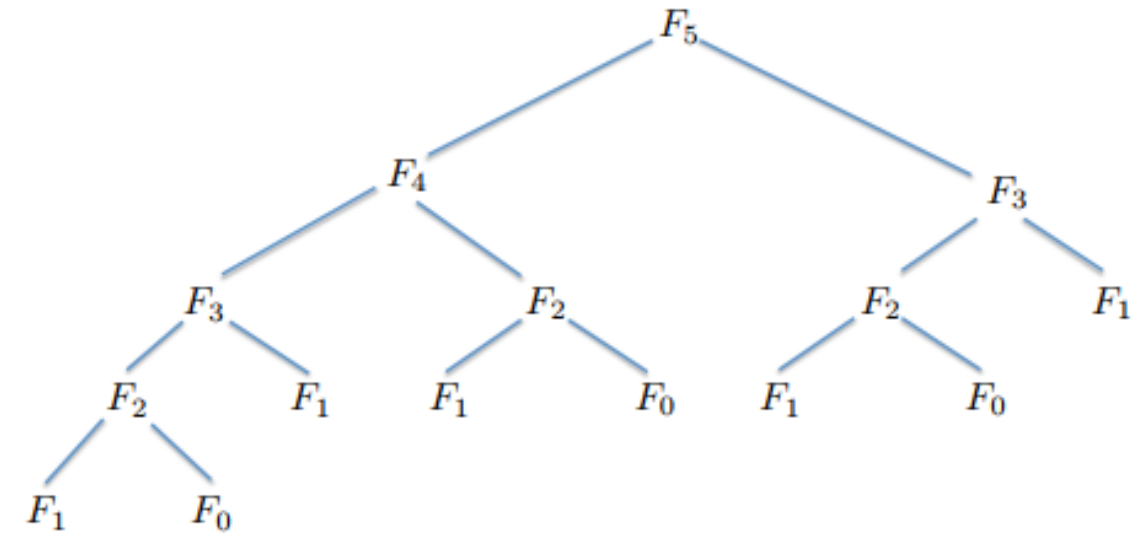
*Example

* Optimal substructure: Need to do sub calculation
* Overlapping: recurrence of calculation for F3 and F2.



Example

- Using recursive backtracking



Example

- Problem with recursive algorithm:

Computes F(n – 2) twice, F(n – 3) three times, etc., each time from scratch

*Code

*Exercise 1: What is the output of this code?

```
#include <algorithm>
int counter=0;
using namespace std;

int fibo(int n)
{
 printf("fibo(%d) and this function
recur:  %d \n", n, counter++);
if (n==0)
    return 0;
if (n==1)
    return 1;
return( fibo(n-2) + fibo(n-1) );
}
```

```
int main ()
{
 int number, n = 5;
 number= fibo(n);
 printf("Solution of the problem %d",
number);
 return 0;
}
```

Break 10 minutes

*Example

How to add intelligent in the code?

*Example

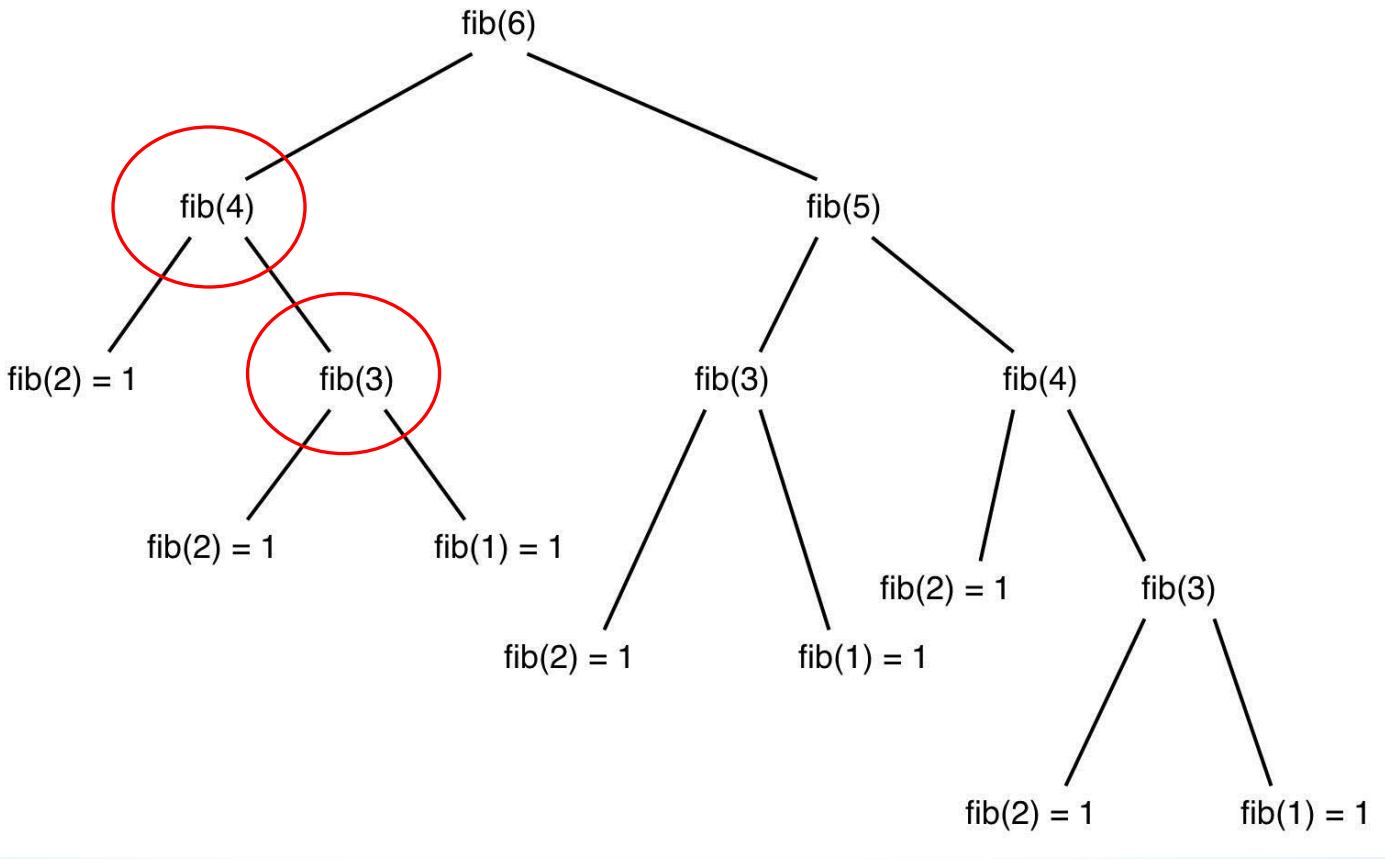
Add memoization…….

*Example

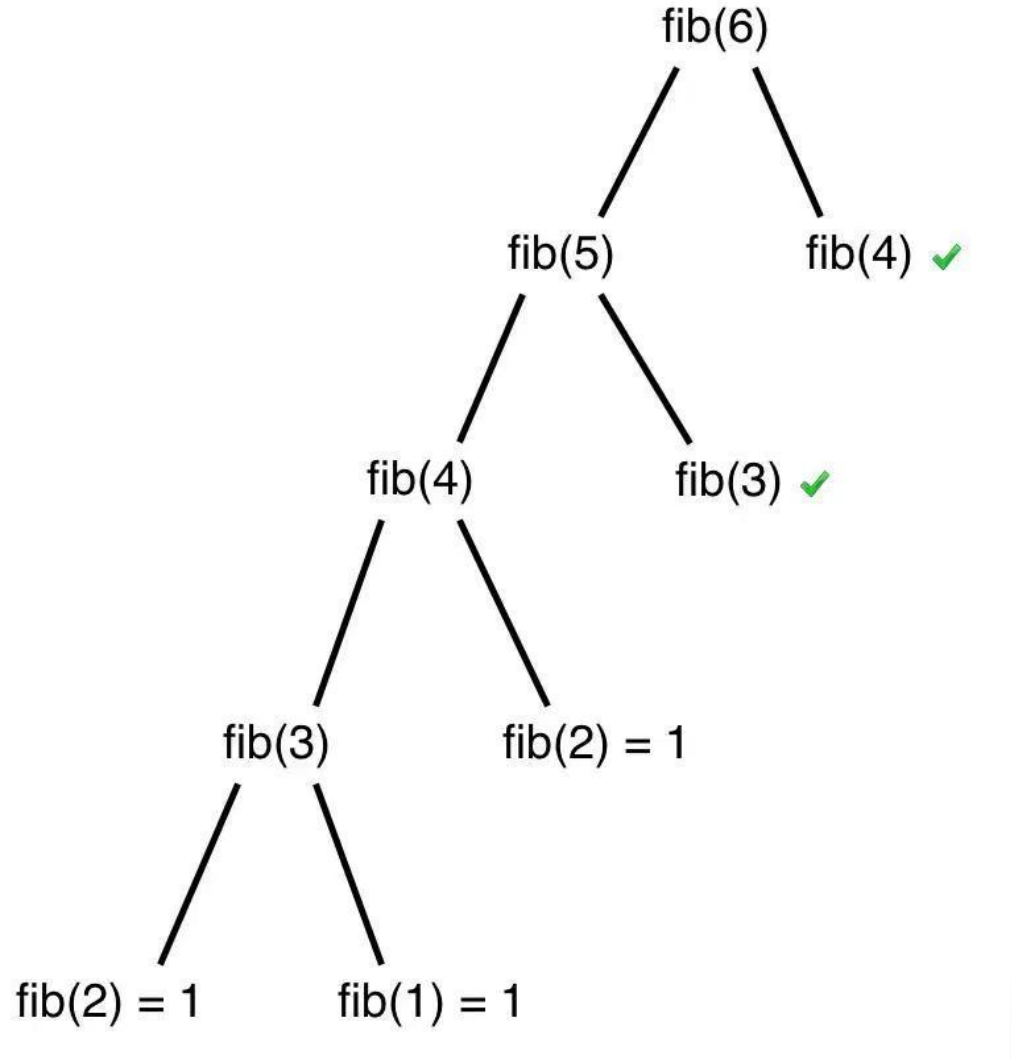
Never recompute a subproblem F(k), k ≤ n, if it has been computed before.

This technique
of remembering previously computed values is called memoization.

*Which one is DP



*Which one is DP



*Algorithm

memo = { }

fib(n): if n in memo: return memo[n]

else if n = 0: return 0

else if n = 1: return 1

else: if = fib(n – 1) + fib(n – 2)

memo[n] = f return f

*Code

*Exercise 2: What is the output of this code?

```
#include <algorithm>
int counter=0;
using namespace std;

int fibo(int n)
{
 int ans;
 printf("fibo(%d)and this function
recur: %d \n", n,counter++);
 if (memo[n] != -1) return memo[n];
 else if (n==0)return 0;
 else if (n==1)return 1;
 else {
  ans = fibo(n-2) + fibo(n-1);
return( memo[n]=ans );
 }}
```

```
int main ()
{
 int number, n = 5;
 number= fibo(n);
 printf("Solution of the problem %d",
number);
 return 0;
}
```

*DP

* DP ≈ recursion + memoization (i.e. re-use)

*DP

- Lab 5:
- Coin Change
- Wedding shopping: Add Memoization
- Knapsack (https://www.youtube.com/watch?v=dN_gQYo9Uf8)

*DP

- Break 20 minutes (Breakfast → Go to the Lab)