Dynamic Programming

Fibonacci Numbers

- The series: 0 1 1 2 3 5 8 13 21... or 1 1 2 3 5 8 13 21 ...
- Fib(n) = Fib(n-1) + Fib(n-2)
- A simple recursive implementation

```
int fib(int n)
{
   if (n <= 1)
     return n;
   return fib(n - 1) + fib(n - 2);
}
</pre>
int fib(int n)
{
   if (n <= 1)
     return 1;
   return fib(n - 1) + fib(n - 2);
}

return fib(n - 1) + fib(n - 2);
}
```

Fibonacci Numbers

- Any problem with the previous solution approach?
- Something to do with complexity!

Fibonacci Numbers

```
fib(5)

/
fib(4) fib(3)

/ \ / \
fib(3) fib(2) fib(2) fib(1)

/ \ / \ / \
fib(2) fib(1) fib(0) fib(1) fib(0)

/ (
fib(1) fib(0)
```

Memoization

- Store the results of the subproblems (creating a table or memo)
- Instead of computing the subproblems every time, if it is calculated before, lookup in the memo and return

Memoization (Top Down Approach)

```
-1
                    -1
                              -1
                                         -1
                                                   -1
                                                              -1
                                                                        -1
                                                                                   -1
memo
                    0
                                                                                   0
          0
                              0
                                         0
                                                   0
                                                              0
                                                                        0
 set
```

```
int fib(int n)
{
    if(set[n])
       return memo[n]
    if (n <= 1){
       res = 1;
    else res = fib(n - 1) + fib(n - 2);
    memo[n] = res;
    set[n] = 1;
    return res;
}</pre>
```

Bottom up approach (Tabulation)

Bottom up approach (Tabulation)

Bottom up approach (Tabulation)

memo

-1 -1 0 1 -1 -1 -1 -1 memo[0] = 0;memo[1] = 1; int fib(int n) if (n <= 1){ return = memo[n]; for(int i=2; i<=n; i++) memo[n] = memo[n-1] + memo[n-2];return memo[n];

Friends Pairing Problem

- •Input : n = 3
- Output: 4
- Explanation:
 - {1}, {2}, {3} : all single
 - {1}, {2, 3}: 2 and 3 paired but 1 is single.
 - {1, 2}, {3}: 1 and 2 are paired but 3 is single.
 - {1, 3}, {2}: 1 and 3 are paired but 2 is single.
- Similar complexity as Finding Fibonacci series

Friends Pairing Problem

• Find smaller sub problem

Friends Pairing Problem

- Find smaller sub problem
 - f(n) = f(n-1) + (n-1) * f(n-2)

• Given an integer array of coins[] of size N representing different types of currency and an integer sum, The task is to find the number of ways to make sum by using different combinations from coins[].

Note: Assume that you have an infinite supply of each type of coin.

•Input: sum = 4, coins[] = {1,2,3}, n =3 Output: 4 Explanation: there are four solutions:

• {1, 1, 1, 1}, {1, 1, 2}, {2, 2}, {1, 3}.

•Input: sum = 10, coins[] = {2, 5, 3, 6}, n =4 Output: 5

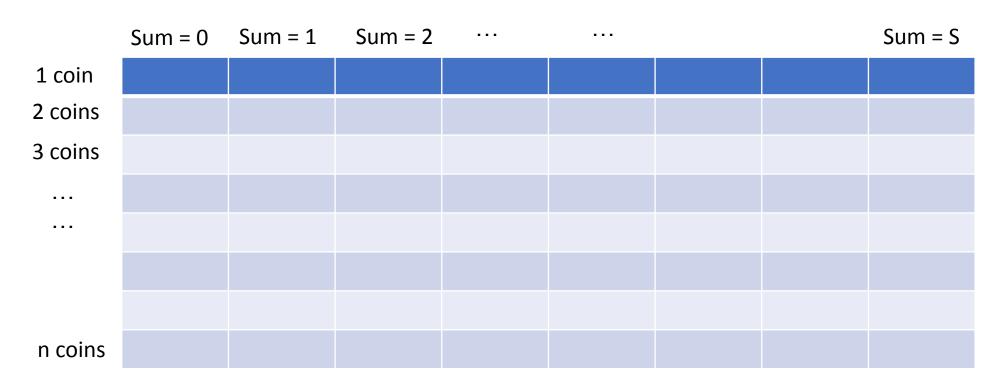
Explanation: There are five solutions:

• {2,2,2,2,2}, {2,2,3,3}, {2,2,6}, {2,3,5} and {5,5}.

- Smaller sub problems:
 - change(n, sum) = change(n-1, sum) + change(n, sum coins[n])

```
C(\{1,2,3\},5)
     C(\{1,2,3\},2) C(\{1,2\},5)
C(\{1,2,3\},-1) C(\{1,2\},2) C(\{1,2\},3) C(\{1\},5)
 C(\{1,2\},0) C(\{1\},2) C(\{1,2\},1) C(\{1\},3) C(\{1\},4) C(\{\},5)
       . . . . . . C(\{1\},3) C(\{\},4)
```

Coin Change Memoization



Coin Change Memoization

	Sum = 0	Sum = 1	Sum = 2	•••	•••		Sum = S
1 coin	1						
2 coins	1						
3 coins	1						
	1						
•••	1						
	1						
	1						
n coins	1						

Bottom up implementation 1

```
for (i = 0; i < n; i++)
     dp[i][0] = 1;
for (i = 0; i < n; i++) {
     for (j = 1; j \le sum; j++) {
    if(j < coins[i])</pre>
         if(i > 0) dp[i][j] = dp[i-1][j];
    else
         dp[i][j] = dp[i-1][j] + dp[i] [j - coins[i]];
  return table[n-1][sum];
```

Coin Change Memoization

	Sum = 0	Sum = 1	Sum = 2	•••	•••			Sum = S
0 coin	1	0	0	0	0	0	0	0
1 coins	1							
2 coins	1							
	1							
•••	1							
	1							
	1							
n coins	1							

Bottom up implementation 1

```
for (i = 0; i \le n; i++)
dp[i][0] = 1;
for (j = 1; i <= sum; i++)
     dp[0][j] = 0;
for (i = 1; i <= n; i++) {
     for (j = 1; j \le sum; j++) {
     if(j < coins[i - 1])
           dp[i][j] = dp[i-1][j];
     else
           dp[i][j] = dp[i-1][j] + dp[i][j - coins[i - 1]];
   return dp[n][sum];
```

Top down solution?

- Initialize the dp array as -1 //to check whether the value is updated or not
- f(n, sum)
- Base Cases:
 - if(sum == 0) return dp[sum][n] = 1;
 - else if(n == 0) return dp[sum][n] = 0;
 - else if(dp[n][sum] != -1) return dp[n][sum];
 - else recursive call

- Your goal is to find the minimum number of coins required, to give an exchange of a target (sum) by using n coins having values:
 - coins[] = {c1, c2, c3....cn}

Smaller sub problems and the relation between them ?

< Excluding nth coins and including nth coins>

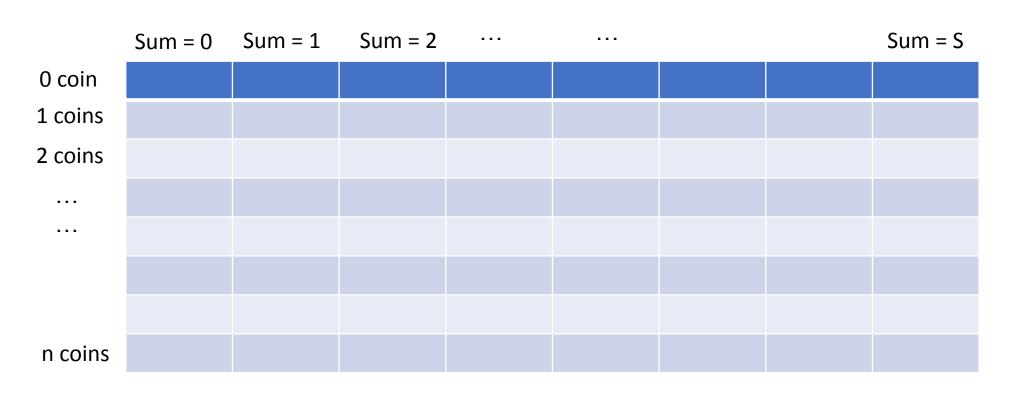
- Eexcluding:
 - dp[n-1][sum]
- Including:
 - 1 + dp[n] [sum coins[n]]

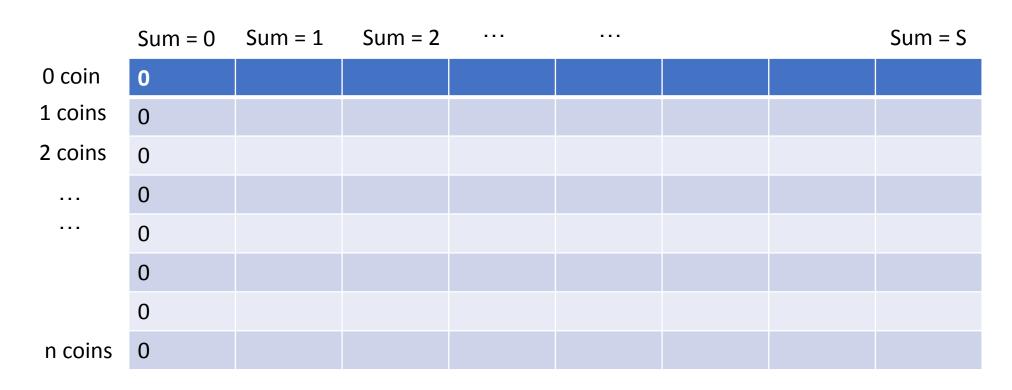
Relation between them?

< Excluding nth coins and including nth coins>

- Eexcluding:
 - dp[n-1][sum]
- Including:
 - 1 + dp[n] [sum coins[n]]

min (excluding, including)





	Sum = 0	Sum = 1	Sum = 2		•••			Sum = S
0 coin	0	∞						
1 coins	0							
2 coins	0							
	0							
•••	0							
	0							
	0							
n coins	0							

Bottom up implementation 1

```
for (i = 0; i \le n; i++)
     for (j = 0; j \le sum; j++) {
    if(j==0) dp[i][j] = 0;
    else if(i==0) dp[i][j] = INF;
    else if(j < coins[i - 1])
        dp[i][j] = dp[i-1][j];
    else
        dp[i][j] = min(dp[i-1][j], 1 + dp[i][j - coins[i - 1]]);
```

• Given weights and values of N items, put these items in a knapsack of capacity W to get the maximum total value in the knapsack.

- Input: N = 3, W = 4
- values[] = {1,2,3}
- weight[] = {4,5,1}
- Output: 3
- Input: N = 3, W = 3
- $values[] = \{1,2,3\}$
- weight[] = {4,5,6}
- Output: 0

Smaller sub problems and the relation between them ?

Smaller sub problems and the relation between them ?

