Programación Competitiva

Alex Josué Flórez Farfán

Magister en Ciencias de la Computación

Ciencia de la Computación – UNSA
Segundo semestre 2021

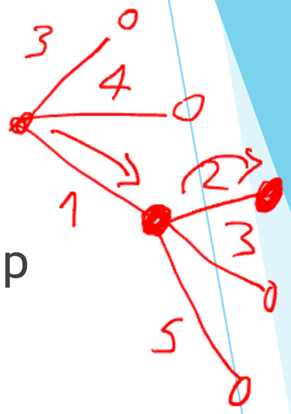
Dynamic Programming

Dynamic Programming

- Greedy approach
- Dynamic Programming

- Strategies to solve optimization problems
  - Minimum result
  - Maximum result

Dynamic Programming



- Greedy approach: a solution is determined based on making the locally optimal choice at any given moment.
- Dynamic Programming. try to find all the possible solutions and then pick up the best solution
- **Principle of Optimality**
- A problem can be solved by taking sequence of decisions to get the optimal solution

Fibonacci

- Fibonacci sequence

  `0 1 1 2 3 5 8`

- Recursion
- Memoization
- Bottom-up approach

Fibonacci. Recursive

- 0 1 1 2 3 5 8

0 1 1 2 3 5 8

$fib(0) = 0 \qquad n = 0$

$fib(1) = 1 \qquad n = 1$

$fib(n) = fib(n-2) + fib(n-1) \qquad n > 1$

$T(n)$ fib(n)
  if $n <= 1$
    1   return n
  return fib(n-2)
  $T(n-2)$ + fib(n-1)
  $+ T(n-1)$

$$T(n) = \underline{T(n-2)} + \underline{T(n-1)} + 1$$

$$\approx$$

$$= \underline{2}\, T(n-1) + 1$$

$$2^n$$

Fibonacci. Recursive

$$fib(n) = \begin{cases} 0, & n = 0 \\ 1, & n = 1 \\ fib(n-2) + fib(n-1), & n > 1 \end{cases}$$

0 1 1 2 3 5 8

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

Fibonacci. Memoization

- Storing the results of the function to avoid repeated callings

- 0 1 1 2 3 5 8

```
long fib_memoization(int n) {
    vector<long> memo(n+1);
    return fib(n, memo);
}
```

```
long fib(int n, vector<long> &memo) {
    if (memo[n])
        return memo[n];

    long result{};
    if (n <= 1)
        result = n;
    else
        result = fib(n-2, memo) + fib(n-1, memo);

    memo[n] = result;
    return result;
}
```

Fibonacci. Bottom-up

- 0 1 1 2 3 5 8

```
long fib_bottomup(int n) {
    if (n <= 1)
        return n;

    vector<long> F(n+1);
    F[0] = 0;
    F[1] = 1;
    for (int i{2}; i <= n; ++i) {
        F[i] = F[i-2] + F[i-1];
    }
    return F[n];
}
```

0	1	1	2	3

0-1 Knapsack Problem

We have n items, each has a weight and value

The problem is, we try to decide which items to put in the knapsack which can only carry a certain capacity.

We want to maximize the total amount of value that we carry with those items

- n = 5
- weight(kg) 1 2 4 2 5
- value($)   5 3 5 3 2

C

- Yes 1
- No   0

0-1 Knapsack Problem. Recursive

$W = 12$

$n = 0 \quad C = 0$

return 0

$C = 10$

$w > C$

$KS(n-1, C)$

$$\max \begin{cases} KS(n-1, C) \\ V[n] + KS(n-1, C - W[n]) \end{cases}$$

$C = 10$

$n = 5$

weight(kg)	1	2	4	2	$\boxed{5}$
value($)	5	3	5	3	$\boxed{2}$

Yes

No

$n = 4$

$C = 5$

$V = 2$

yes

$n = 5$

$C = 10$

$v = 0$

No

Yes

No

$n = 4$

$C = 10$

$V = 0$

0-1 Knapsack Problem. Recursive

$w[n] > C$

Knapsack$(n-1, C)$

$$\max\begin{cases} \text{Knapsack}(n-1, C) \\ \text{value}[n] + \text{Knapsack}(n-1, C-w[n]) \end{cases}$$

C = 10

n = 5

weight(kg)	1	2	4	2	5
value($)	5	3	5	3	2

weight = [$\emptyset$, 1, 2, 4, 2, 5]

value = [$\emptyset$, 5, 3, 5, 3, 2]

value[3]

Yes

No

$n=4$
$C=5$
$v=2$

Yes

No

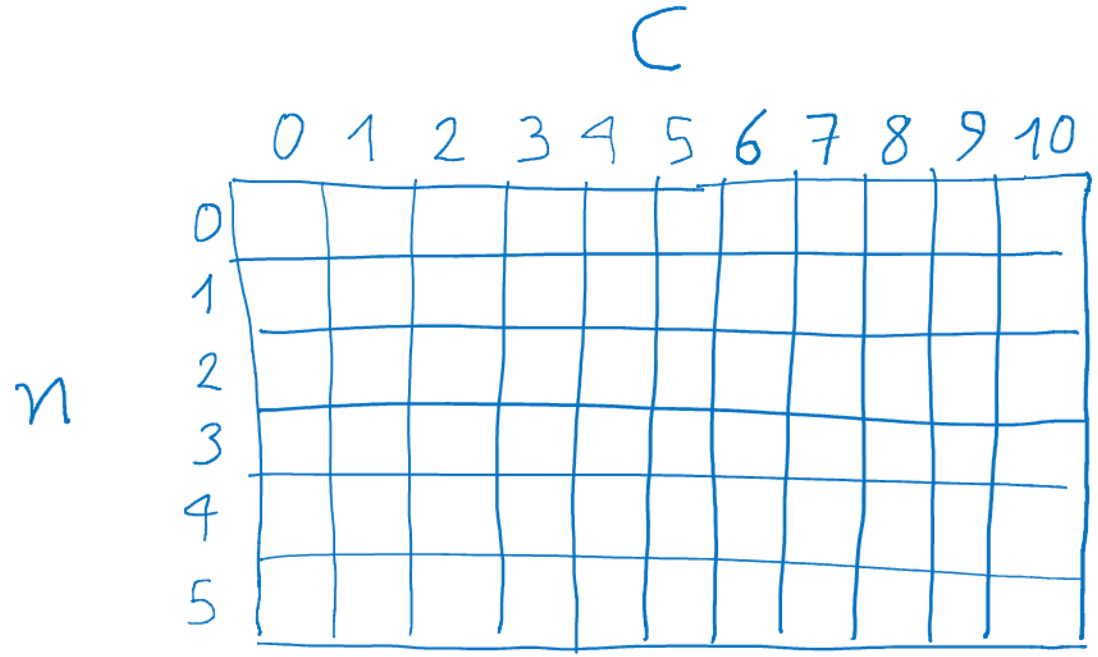
$n=4$
$C=10$
$v=0$

Yes

No

$n=5$
$C=10$
$v=0$

0-1 Knapsack Problem. Memoization



C = 10

n = 5

weight(kg)	1	2	4	2	5
value($)	5	3	5	3	2

0-1 Knapsack Problem. Bottom-up

C = 10

n = 5

weight(kg) 1 2 4 2 5

value($) 5 3 5 3 2

C

n

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0										
2	0										
3	0										
4	0										
5	0										