Dynamic Programming

Peter Parker
1705XXX
Tony Stark
1705XXX

August 13, 2022

Table of Contents

We are going to see

Dynamic Programming

- Algorithm technique that systematically **records** the answers to sub-problems and **reuses** them those recorded result.

Dynamic Programming

- Algorithm technique that systematically **records** the answers to sub-problems and **reuses** them those recorded result.
- A simple example:
  Calculating the n-th Fibonacci number
  $Fib(n) = Fib(n-1) + Fib(n-2)$

continued.

- The method was developed by Richard Bellman in the 1950s
- It breaks down a complicated problem into simpler sub-problems in a recursive manner.
- If optimal solutions can be found recursively for the sub-problems, then it is said to have optimal substructure.

continued.

- The method was developed by Richard Bellman in the 1950s
- It breaks down a complicated problem into simpler sub-problems in a recursive manner.
- If optimal solutions can be found recursively for the sub-problems, then it is said to have optimal substructure.

continued.

- The method was developed by Richard Bellman in the 1950s
- It breaks down a complicated problem into simpler sub-problems in a recursive manner.
- If optimal solutions can be found recursively for the sub-problems, then it is said to have optimal substructure.

We are going to see

Properties of Dynamic Programming

Such problems exhibits following two properties:

- **Optimal Substructure**
- **Overlapping sub-problems**

Optimal Substructure

A problem is said to have optimal substructure if an optimal solution can be constructed from optimal solutions of its sub-problems.
e.g. in Floyd-Warshall algorithm, travelling from node i to j using node k, dist[i][j]=dist[i][k]+dist[k][j]

Overlapping sub-problems

A problem has overlapping sub-problems if finding its solution involves solving the same sub-problem multiple times.
Example: Calculating n-th Fibonacci number $F(n)$

Example of Overlapping sub-problems

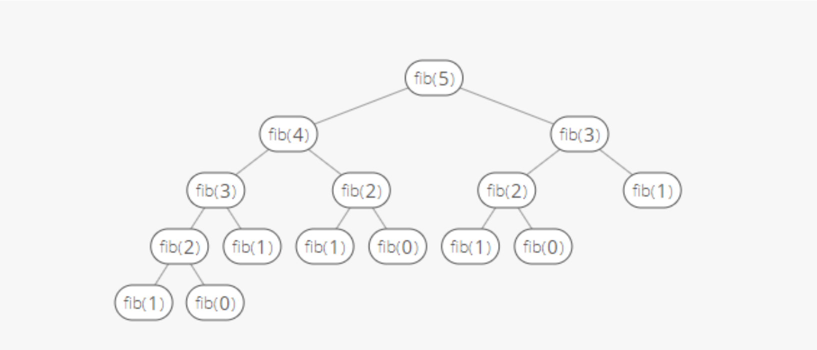


Figure: Overlapping Sub-problems in determination of Fibonacci series

We are going to see

Binomial Coefficient a.k.a C(n,r)

problem statement: ways to select r objects from n objects regardless of the ordering

Binomial Coefficient a.k.a C(n,r)

Naive approach : calculating $\frac{n!}{r!(n-r)!}$

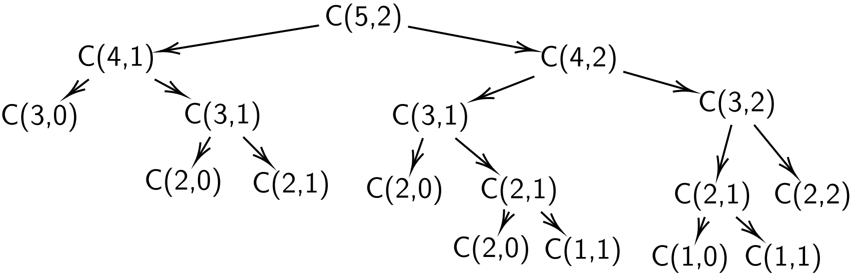
- Problem : overflow will be caused calculating factorials, unsigned long long wouldn't be enough. May be BigInteger would do but not efficient.
- Solution : using dynamic programming.

C(n,r) having dynamic programming properties

Optimal Substructure: C(n,r) can be recursively calculated using the formula,
$C(n, r) = C(n-1, r-1) + C(n-1, r)$
with base cases, $C(n, 0) = C(n, n) = 1$ and $C(n, 1) = n$

C(n,r) having dynamic programming properties

Overlapping Sub-problems: let n=5, r=2



Algorithm List		
Algorithm Name	Time Complexity	Space Complexity
BFS	$O(\|V\| + \|E\|)$	$O(\|V\|)$
DFS	$O(\|V\| + \|E\|)$	$O(\|V\|)$
Dijkstra	$O(\|V\|+\|E\|log\|V\|)$	$O(\|V\| + \|E\|)$
Bellman Ford	$O(\|V\| * \|E\|)$	$O(\|V\|)$
Floyd–Warshall	$O(\|V\|^3)$	$O(\|V\|^2)$
Edmonds–Karp	$O(\|V\| * \|E\|^2)$	$O(\|V\| + \|E\|)$