LUT University

BM40A1500 DATA STRUCTURES AND ALGORITHMS

ALGORITHM DESIGN PRINCIPLES 2

2024

ALGORITHM DESIGN PRINCIPLES

- Greedy approach
- Backtracking
  - Branch and bound
- Divide and conquer
- Dynamic programming
- Probabilistic algorithms
  - Las Vegas algorithms
  - Monte Carlo algorithms

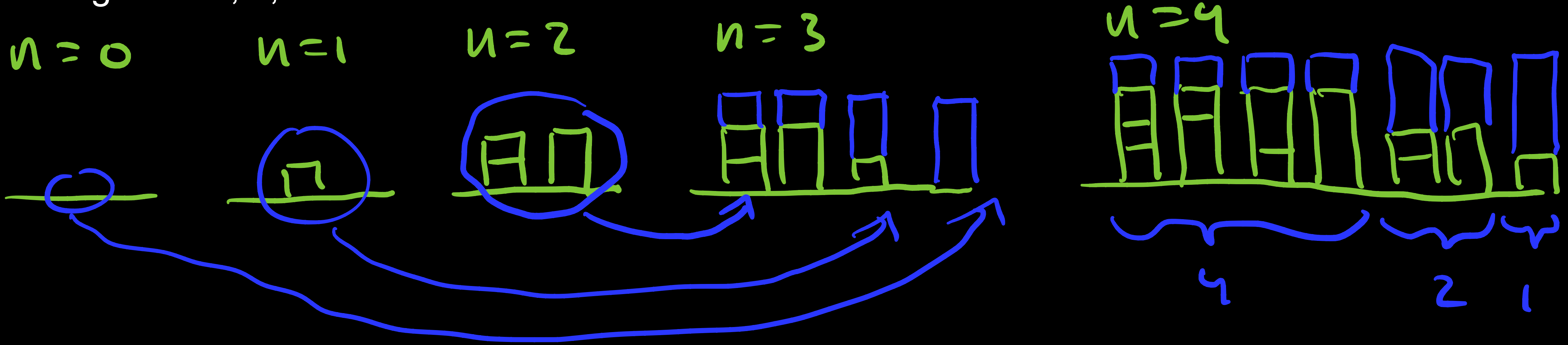
DYNAMIC PROGRAMMING

- A way to improve the efficiency of any inherently recursive algorithm that repeatedly re-solves the same subproblems.
- Steps of utilizing dynamic programming:
    1. Find a recursive solution to your problem
    2. Identify the subproblems that are redundantly solved many times.
    3. Optimize the algorithm by eliminating re-solving of subproblems
        - **Storing subproblem results in a table**
- The final algorithm can be either recursive or iterative.
    - The iterative form is commonly referred to by the term dynamic programming.

LUT University

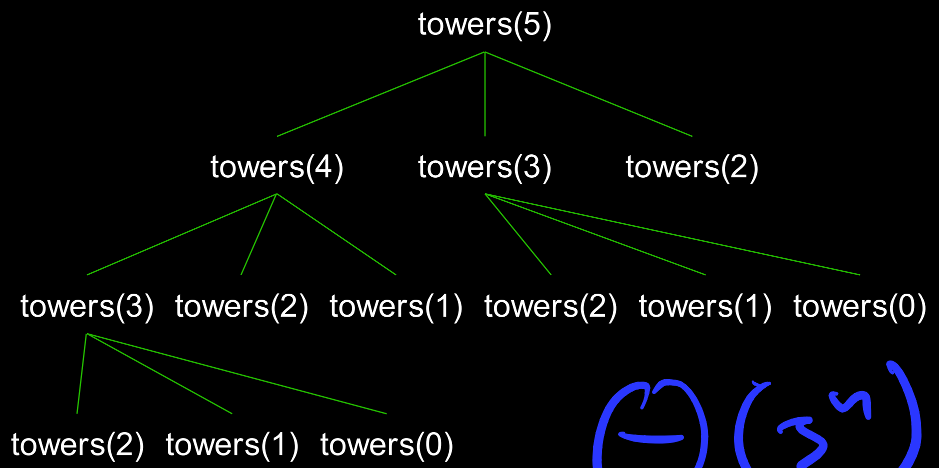
DYNAMIC PROGRAMMING

- **Example:** In how many ways we can build a tower with the heigh *n* by using blocks with heights of 1, 2, and 3?



DYNAMIC PROGRAMMING

- Example: In how many ways we can build a tower with the heigh *n* by using blocks with heights of 1, 2, and 3?
  - towers(n) = towers(n-1) + towers(n-2) + towers(n-3)
  - Can be solved easily using recursion.
  - Very slow due to the repetitive solving of the subproblems with small value of n.



$\Theta(3^n)$

  - Solution using dynamic programming (values from the subproblems stored in a table):

```
towers[0] = 1
towers[1] = 1
towers[2] = 2
for i = 3 to n
        towers[i] = towers[i-1] + towers[i-2] + towers[i-3]
```

$\Theta(n)$

DYNAMIC PROGRAMMING

- Knapsack problem (subset sum problem):
  - find a subset of the *n* items whose sizes exactly sum to the size of the knapsack, if one exist.

$$\sum_{i \in S} k_i = K$$

S: subset of items

  - E.g If we have 4 items of sizes 3, 8, 7, and 5, and *K* = 10 there exists a solution (3 + 7 = 10), but if *K* = 14, there is no solution.
  - Let's denote an instance of the problem as P(*n*,*K*)
    - *n* is the index of the last item (the number of items - 1).
    - *K* is the size of the knapsack

DYNAMIC PROGRAMMING

P(10, 3)
$k_i = \{5, 3, 7, 2\}$

$$\sum_{i \in S} k_i = 10\ ?$$

P(10, 2)
$k_i = \{5, 3, 7\}$

$$\sum_{i \in S} k_i = 10\ ?$$

$k_3 = 2$ is omitted

P(8, 2)
$k_i = \{5, 3, 7\}$

$$\sum_{i \in S} k_i = 8\ ?$$

$k_3 = 2$ is included

- The problem P(n,K) can be divided into simpler subproblems:
  - P($n-1$, K) – nth item is omitted
  - P($n-1$, $K-k_n$) -- nth item is included
- Can be solved recursively
  - Base cases are those, where there are only one item, or the knapsack has the size of 0.
- To avoid solving the same subproblems multiple times, the solutions can be stored in a table.

DYNAMIC PROGRAMMING

P(10, 3)
k_i = {5, 3, 7, 2}

$$\sum_{i \in S} k_i = 10\ ?$$

P(10, 2)
k_i = {5, 3, 7}

$$\sum_{i \in S} k_i = 10\ ?$$

k_3 = 2 is omitted

P(8, 2)
k_i = {5, 3, 7}

$$\sum_{i \in S} k_i = 8\ ?$$

k_3 = 2 is included

P(8, 1)

P(8, 0) P(5, 0)

	0	1	2	3	4	5	6	7	8	9	10
k_0=5	0	-	-	-	-	1	-	-	-	-	-
k_1=3	0	-	-	1	-	0	-	-	1	-	-
k_2=7	0	-	-	0	-	0	-	1	0	-	1
k_3=2	0	-	1	0	-	1/0	-	1/0	0	1	1/0

S = {2, 3, 5}

S = {7, 3}

PROBABILISTIC ALGORITHMS

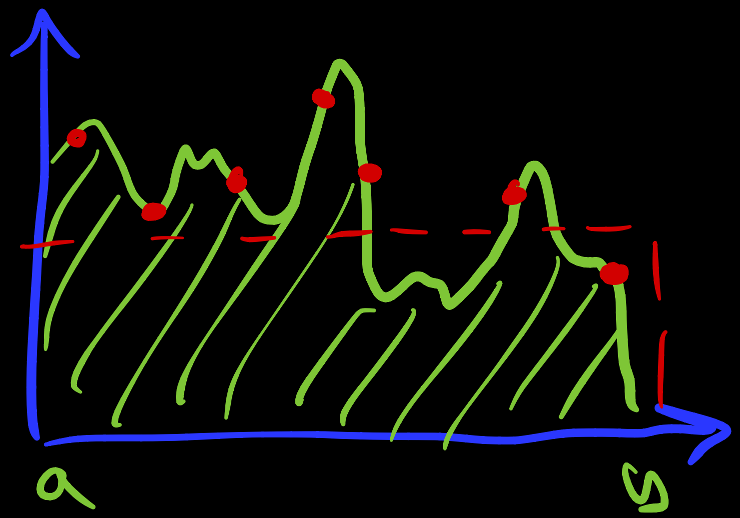
- Typically, algorithms are defined as set of instructions that are executed *deterministically.*
- If we relax the definition a bit, we can introduce randomness to our algorithms to:
  - reduce the execution time,
  - increase the probability of finding a good solution/result within time limits, and
  - reduce the probability of a bad case with long running time.
- Especially useful for very difficult problems for which efficient algorithm is not known.

MONTE CARLO ALGORITHMS

- Probabilistic algorithms that do not necessary produce exact (or optimal) result
- But produce some result fast.
- Accuracy or goodness of the result can be typically improved by increasing the computation time.
- Example: numerical integration:

$$\int_a^b f(x)dx$$

```
s=0
for i = 1 to N
    x = random([a,b])
    s = s + f(x)
return (b - a)(s / N)
```



LAS VEGAS ALGORITHMS

- Produce only correct/optimal results (or informs that result was not found),
- But the running time is not guaranteed.
- Typically, the running time is restricted (e.g., maximum number of iterations)
  - The result may not be found at all.
- Probability of finding the (correct) result can be increased by increasing the maximum running time

LAS VEGAS ALGORITHMS

- Example: prime factorization of large numbers

```
prime_factors = []
for i = 1 to N
    x = random_prime_number()
    if mod(number, x) == 0
        prime_factors = [prime_factors, x]
        number = number / x
         if isprime(number)
            prime_factors = [prime_factors, number]
            return prime_factors
```


LUT
University