DYNAMIC PROGRAMMING

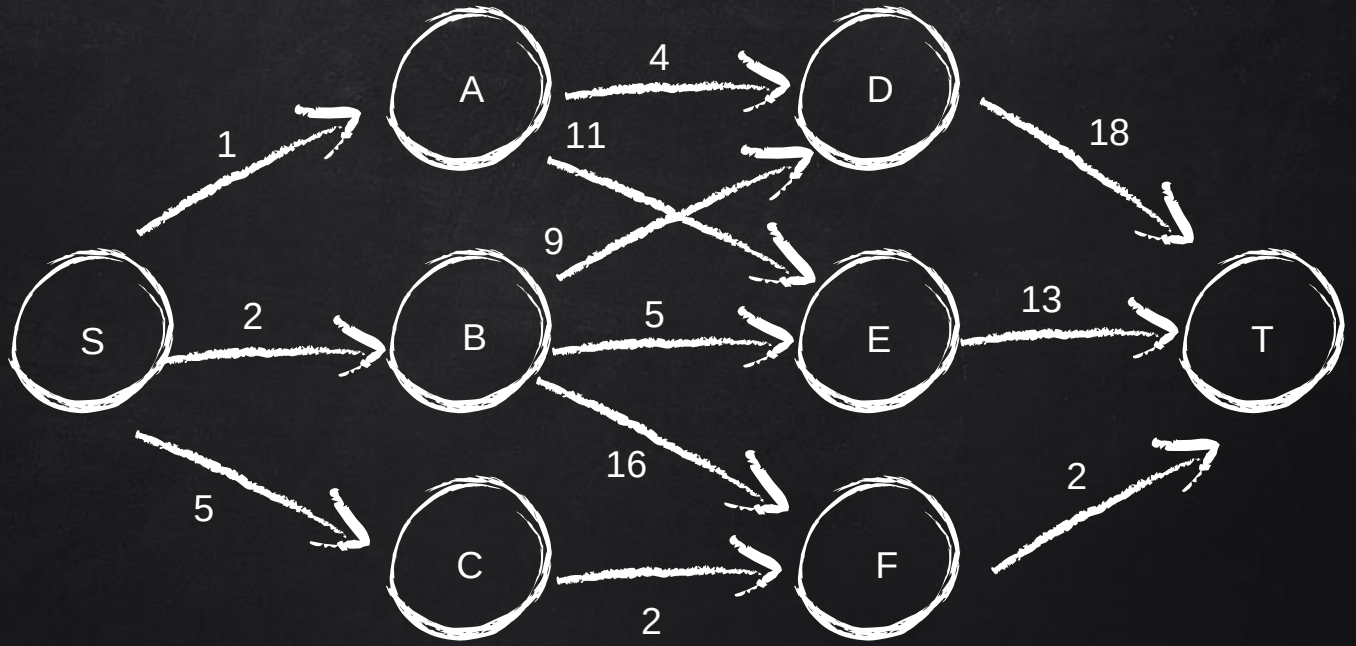
DP

DESIGN AND ANALYSIS OF ALGORITHMS

LECTURER: Nguyễn Thanh Sơn

CS112.L23.KHCL.N12

The shortest path



Apply the Greedy approach, the shortest path from S to T is?

The shortest path

The Greedy approach can not be applied to this case:
(S, A, D, T) 1+4+18=23

The real shortest path is:
(S, C, F, T) 5+2+2=9

So today we will learn about a new algorithm

CONTENT

1. What is Dynamic Programming?
2. Characteristics of Dynamic Programming
3. Dynamic Programming Methods
4. Compare with other algorithms
5. Steps in Dynamic Programming
6. List of Dynamic Programming Problems

1. What is Dynamic programMing?

- Dynamic programming (DP) approach is similar to Divide and Conquer in breaking down the problem in smaller and yet smaller possible sub-problems.
- But unlike Divide and Conquer, results of these smaller sub-problems are remembered and used for similar or overlapping sub-problems.

Characteristics of Dynamic Programming

- Overlapping Subproblems
- Optimal Substructure

- **Overlapping Subproblems**

There exist some places where we solve the same subproblem more than once

EX:

```python
def fib(n):
    if n <= 1:
        return n
    return fib(n -1) + fib(n - 2)
```

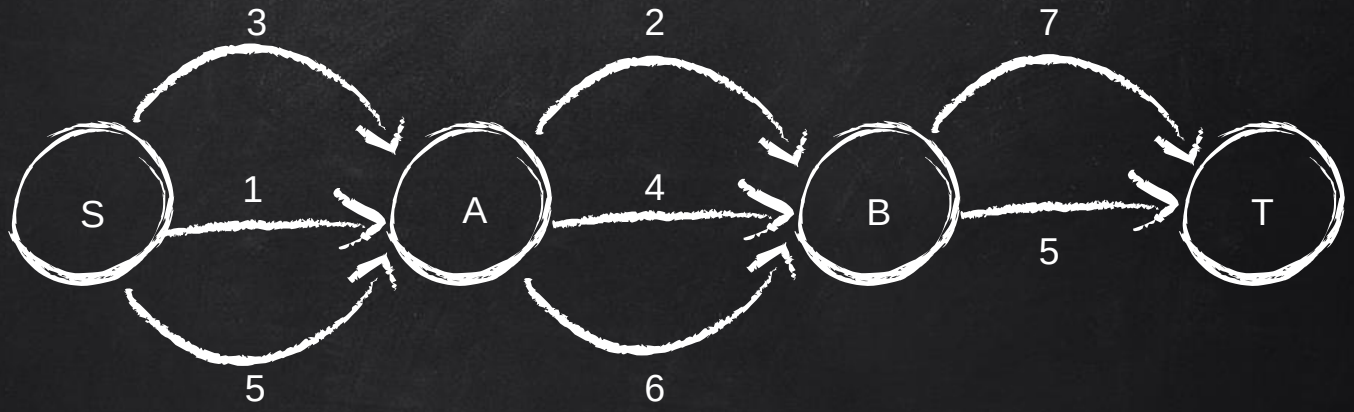
DP:

```python
def fib(n):
    dp = [0] * (n + 1)
    dp[1] = 1
    for i in range(2, n + 1):
        dp[i] = dp[i - 1] + dp[i - 2]
    return dp[n]
```

- **Optimal Substructure**

The optimal solution to the problem contains within optimal solutions to its subproblems.

EX:



$$d_{min}(S, T) = d_{min}(S, A) + d_{min}(A, B) + d_{min}(B, T)$$

$$= 1 + 2 + 5$$

$$= 8$$

Dijkstra

3. Dynamic Programming Methods

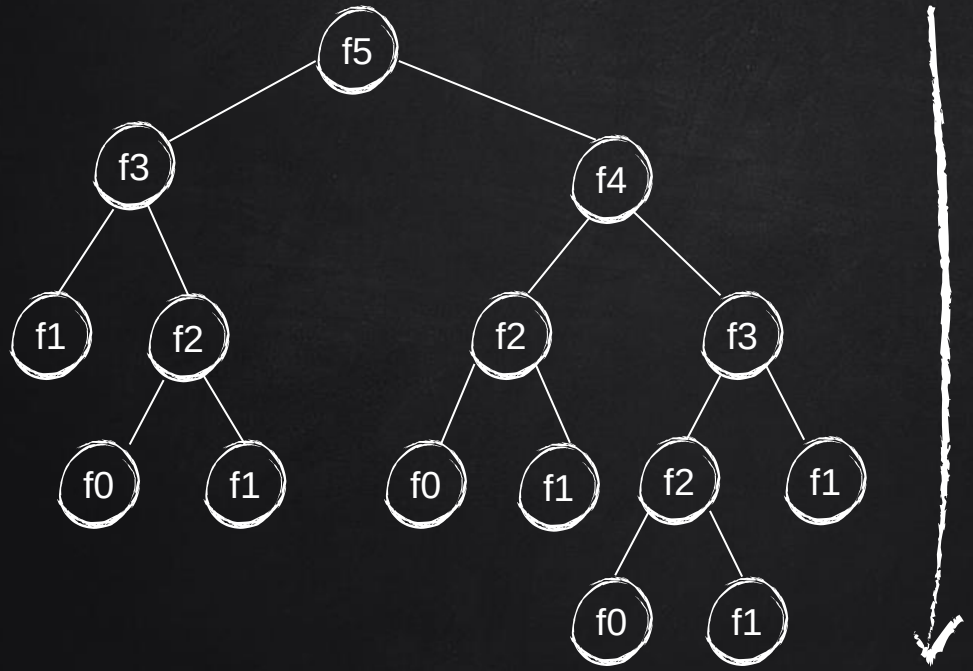
DP offers two methods to solve a problem:

- Top-down with Memoization
- Bottom-up with Tabulation

3. Dynamic Programming Methods

- Top-down with Memoization

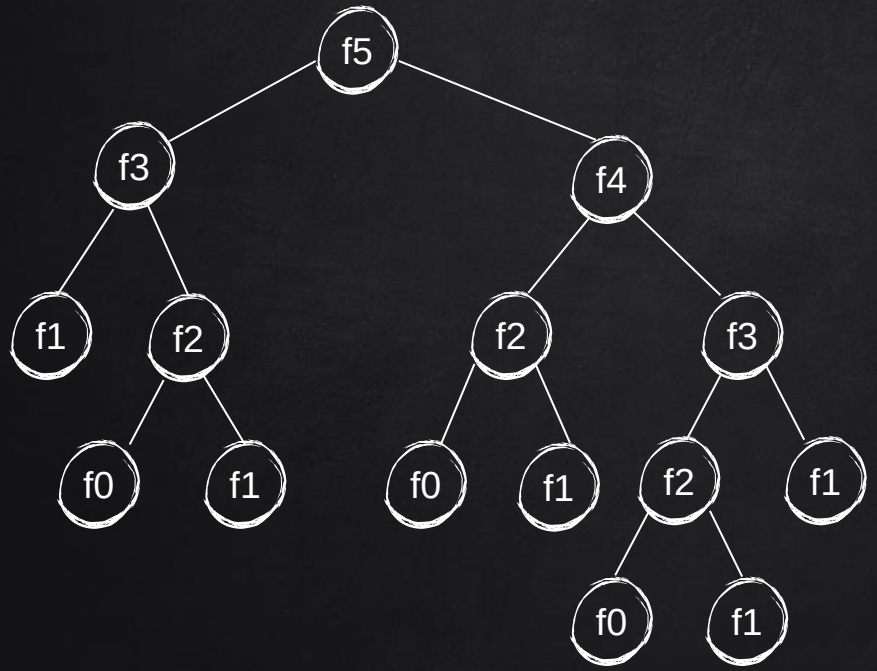
MemoRization



In this approach, we try to solve the bigger problem by recursively finding the solution to smaller sub-problems. Whenever we solve a sub-problem, we cache its result so that we don't end up solving it repeatedly if it's called multiple times. Instead, we can just return the saved result. This technique of storing the results of already solved subproblems is called **Memoization.**

3. Dynamic Programming Methods

- Top-down with Memoization



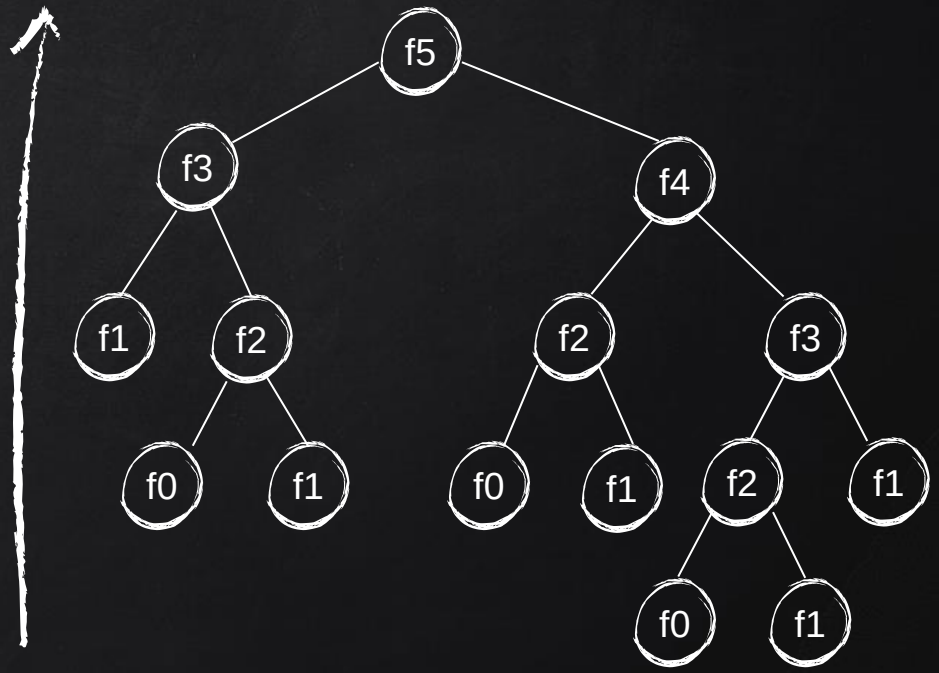

```
F=[]
def Fib(n):
    if n in F:
        return F[n]
    if (n < 2):
        return 1
    result = Fib(n-2) + Fib(n-1)
    F[n] = result
    return result
```

O(n)

3. Dynamic Programming Methods

- Bottom-up with Tabulation

Tabulation is the opposite of the top-down approach and avoids recursion. In this approach, we solve the problem "bottom-up" (i.e. by solving all the related sub-problems first). This is typically done by filling up an n-dimensional table. Based on the results in the table, the solution to the top/original problem is then computed.

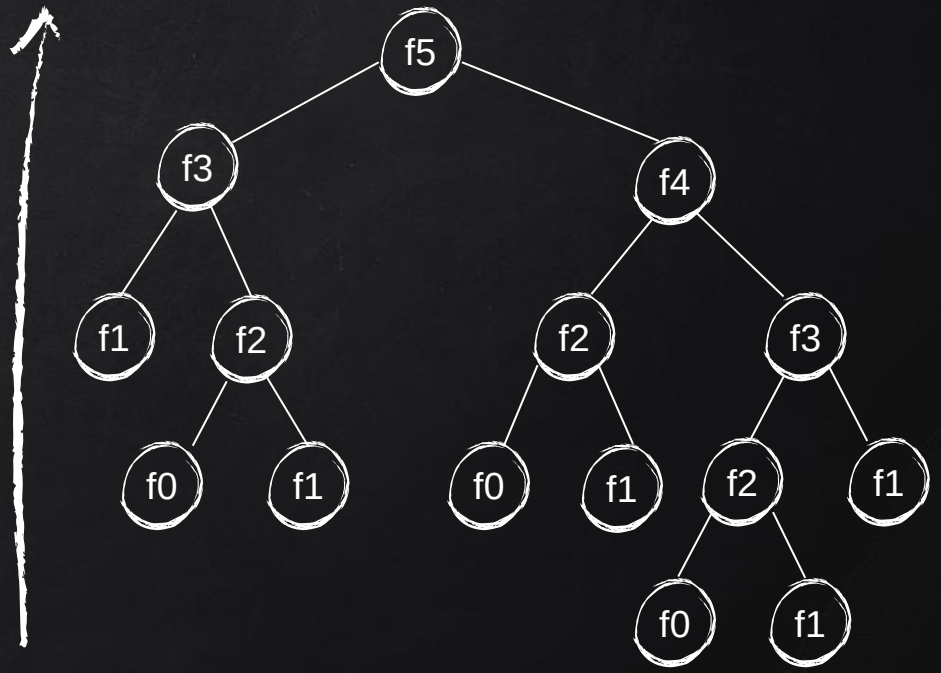


3. Dynamic Programming Methods

- Bottom-up with Tabulation

O(n)

```
def Fib(n):
    F[0] = 0
    F[1] = 1
    for i in 2...n
        F[i] = F[i-2] + F[i-1]
    return F[n]
```



Comparison

Top-down with Memoization	Bottom-up with Tabulation
Easy to set up	Gets complicated if there are multiple conditions
Must be solved from the top down	Must be solved from the bottom up
Slower due to recursion	Faster, due to direct access to the results stored in the table
Just solve the necessary problems	Must solve all subproblems

4. Compare with other algorithms

- Dynamic programming vs Greedy method

Dynamic programming	Greedy method
DP are motivated for an overall optimization of the problem	Local optimization is addressed

Dynamic programming	Greedy method
1. Dynamic Programming is used to obtain the optimal solution.	1. Greedy Method is also used to get the optimal solution.
2. In Dynamic Programming, we choose at each step, but the choice may depend on the solution to sub-problems.	2. In a greedy Algorithm, we make whatever choice seems best at the moment and then solve the sub-problems arising after the choice is made.
3. Less efficient as compared to a greedy approach	3. More efficient as compared to a greedy approach
4. Example: 0/1 Knapsack	4. Example: Fractional Knapsack
5. It is guaranteed that Dynamic Programming will generate an optimal solution using Principle of Optimality.	5. In Greedy Method, there is no such guarantee of getting Optimal Solution.

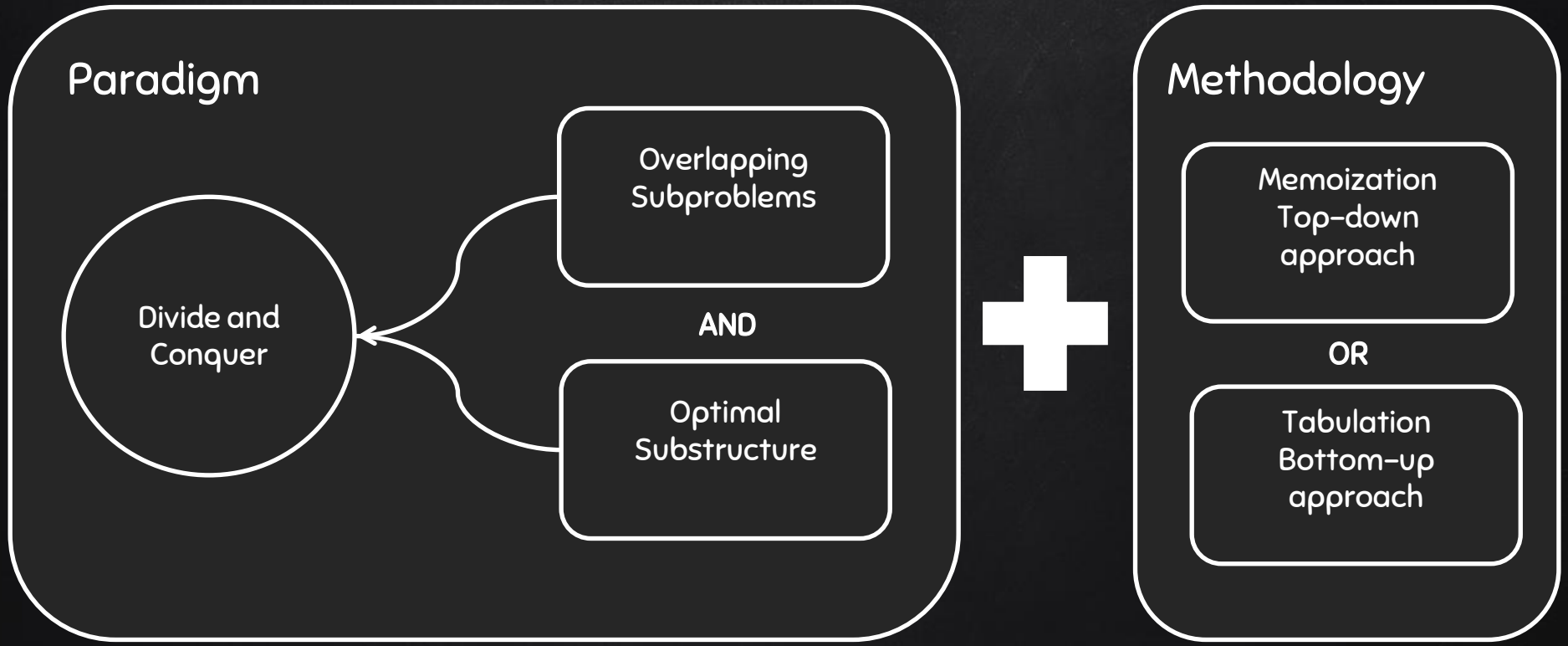
4. Compare with other algorithms

- Dynamic programming vs Divide and Conquer

Dynamic Programming	Divide and Conquer
DP use the output of a smaller sub-problem and then try to optimize a bigger sub-problem and use Memoization to remember the output of already solved sub-problems.	Solutions are combined to achieve an overall solution

Dynamic programming	Divide and Conquer
1. It involves the sequence of four steps: ▪ Characterize the structure of optimal solutions. ▪ Recursively defines the values of optimal solutions. ▪ Compute the value of optimal solutions in a Bottom-up minimum. ▪ Construct an Optimal Solution from computed information.	1. It deals (involves) three steps at each level of recursion: ▪ Divide the problem into a number of subproblems. ▪ Conquer the subproblems by solving them recursively. ▪ Combine the solution to the subproblems into the solution for original subproblems.
2. It is non Recursive.	2. It is Recursive.
3. It solves subproblems only once and then stores in the table.	3. It does more work on subproblems and hence has more time consumption.
4. It is a Bottom-up approach.	4. It is a top-down approach.
5. In this subproblems are interdependent.	5. In this subproblems are independent of each other.
6. For example: Matrix Multiplication.	6. For example: Merge Sort & Binary Search etc.

Dynamic Programming



Steps in Dynamic Programming

1. Characterize structure of an optimal solution.
2. Define value of optimal solution recursively.
3. Compute optimal solution values either top-down with caching or bottom-up in a table.
4. Construct an optimal solution from computed values.

6. List of Dynamic Programming Problems

1. Kadane'sao Algorithm
2. 0 1 Knapsack Problem
3. Longest Increasing Subsequence Problem
4. Edit Distance Problem
5. Integer Knapsack Problem
6. Fibonacci Numbers Problem
7. Rod Cutting Problem
8. Subset Sum Problem
9. Coin change Problem
10. Treats for the Cows

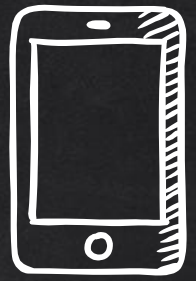
You can find more Dynamic Programming problems here.

0-1 Knapsack Problem

```python
def knapSack(W, wt, val, n):
    # Base Case
    if n == 0 or W == 0:
        return 0
    # If weight of the nth item is
    # more than Knapsack of capacity W,
    # then this item cannot be included
    # in the optimal solution
    if (wt[n-1] > W):
        return knapSack(W, wt, val, n-1)
    # return the maximum of two cases:
    # (1) nth item included
    # (2) not included
    else:
        return max(
            val[n-1] + knapSack(
                W-wt[n-1], wt, val, n-1),
            knapSack(W, wt, val, n-1))
# end of function knapSack
#Driver Code
val = [60, 100, 120]
wt = [10, 20, 30]
W = 50
n = len(val)
print(knapSack(W, wt, val, n))
# This code is contributed by Nikhil Kumar Singh
```

Matrix Chain Multiplication

```python
import sys
# Matrix A[i] has dimension p[i-1] x p[i]
# for i = 1..n
def MatrixChainOrder(p, i, j):
    if i == j:
        return 0
    _min = sys.maxsize
    # place parenthesis at different places
    # between first and last matrix,
    # recursively calculate count of
    # multiplications for each parenthesis
    # placement and return the minimum count
    for k in range(i, j):
        count = (MatrixChainOrder(p, i, k)
                 + MatrixChainOrder(p, k + 1, j)
                 + p[i-1] * p[k] * p[j])
        if count < _min:
            _min = count
    # Return minimum count
    return _min
# Driver code
arr = [1, 2, 3, 4, 3]
n = len(arr)
print("Minimum number of multiplications is ",
    MatrixChainOrder(arr, 1, n-1))
# This code is contributed by Aryan Garg
```



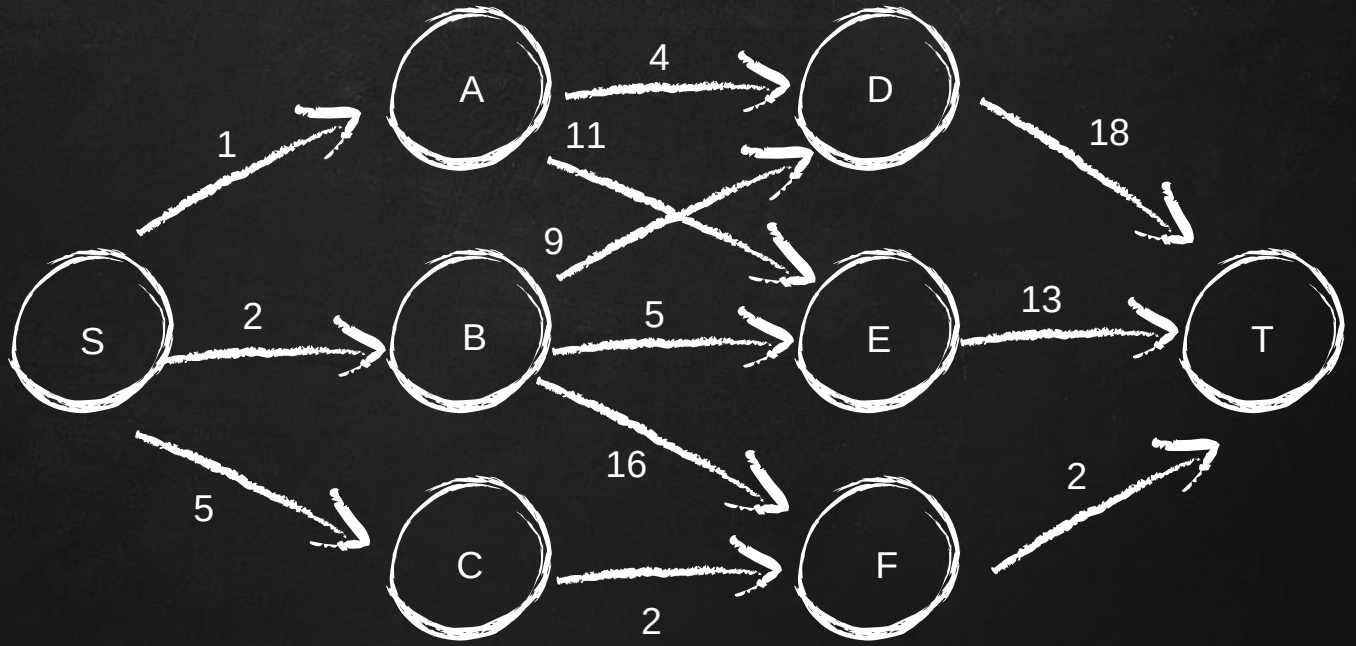
KAHOOT!

HOMEWORK

Gmail: 19521482@gm.uit.edu.vn

Deadline: 11h59 PM, 30/05/2021

The shortest path

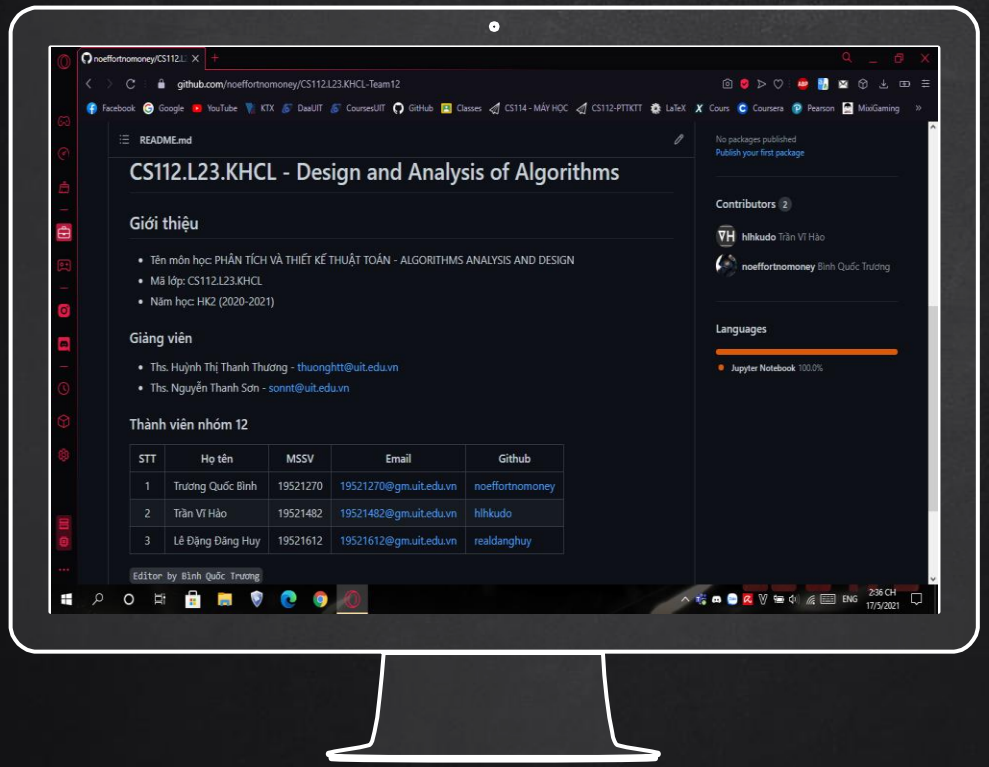


Apply the Dynamic programming, the shortest path from S to T is?

THANKS!

Any questions?

You can review these slides on our team's GitHub



Our team's github link:

https://github.com/noeffortnomoney/CS112.L23.KHCL-Team12

Credits

Special thanks to all the people who made and released these awesome resources for free:

- ✘ Presentation template by SlidesCarnival
- ✘ TOPDev.vn
- ✘ educative.io
- ✘ tutorialspoint.com
- ✘ geeksforgeeks.org
- ✘ gacsach.vn
- ✘ Some PowerPoint files in the Reference section (GitHub)