



Dynamic Programming and Applications

Yıldırım TAM

Instructor: Asst.Prof.Dr.İbrahim YAKUT

Dynamic Programming

Dynamic Programming is a general algorithm design technique for solving problems defined by recurrences with overlapping subproblems

- Invented by American mathematician Richard Bellman in the 1950s to solve optimization problems and later assimilated by CS
- “Programming” here means “planning”
- Main idea:
 - set up a recurrence relating a solution to a larger instance to solutions of some smaller instances
 - solve smaller instances once
 - record solutions in a table
 - extract solution to the initial instance from that table

Divide-and-conquer

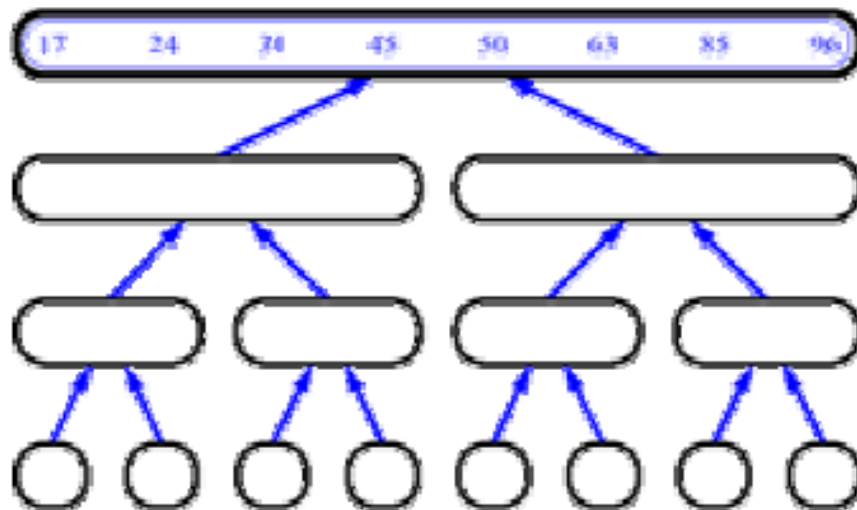
- **Divide-and-conquer** method for algorithm design:
- **Divide:** If the input size is too large to deal with in a straightforward manner, divide the problem into two or more disjoint subproblems
- **Conquer:** conquer recursively to solve the subproblems
- **Combine:** Take the solutions to the subproblems and “merge” these solutions into a solution for the original problem

Divide-and-conquer - Example

- For example,
MergeSort

```
Merge-Sort(A, p, r)
  if p < r then
    q ← (p+r)/2
    Merge-Sort(A, p, q)
    Merge-Sort(A, q+1, r)
    Merge(A, p, q, r)
```

- The subproblems are
independent, all
different.



Dynamic programming

- **Dynamic programming** is a way of improving on inefficient divide-and-conquer algorithms.
- By “inefficient”, we mean that the same recursive call is made over and over.
- If same subproblem is solved several times, we can use table to store result of a subproblem the first time it is computed and thus never have to recompute it again.
- Dynamic programming is **applicable** when the subproblems are dependent, that is, when subproblems share subsubproblems.
- “**Programming**” refers to a tabular method

Difference between DP and Divide-and-Conquer

- Using Divide-and-Conquer to solve these problems is **inefficient** because the **same** common **subproblems** have to be solved **many times**.
- DP will solve each of them **once** and **their answers are stored in a **table**** for future use.

Dynamic Programming vs. Recursion and Divide & Conquer

Divide & Conquer	Dynamic Programming
1. Partitions a problem into independent smaller sub-problems	1. Partitions a problem into overlapping sub-problems
2. Doesn't store solutions of sub-problems. (Identical sub-problems may arise - results in the same computations are performed repeatedly.)	2. Stores solutions of sub-problems: thus avoids calculations of same quantity twice
3. Top down algorithms: which logically progresses from the initial instance down to the smallest sub-instances via intermediate sub-instances.	3. Bottom up algorithms: in which the smallest sub-problems are explicitly solved first and the results of these used to construct solutions to progressively larger sub-instances

Elements of Dynamic Programming (DP)

DP is used to solve problems with the following characteristics:

- Simple subproblems
 - We should be able to break the original problem to **smaller subproblems** that have the same structure
- Optimal substructure of the problems
 - The optimal solution to the problem contains within **optimal solutions to its subproblems**.
- Overlapping sub-problems
 - there exist some places where we solve the same subproblem more than **once**.

Steps to Designing a Dynamic Programming Algorithm

1. Characterize **optimal substructure**
2. **Recursively** define the value of an optimal solution
3. Compute the value **bottom up**
4. (if needed) **Construct** an optimal solution

Principle of Optimality

- The dynamic Programming works on a principle of optimality.
- Principle of optimality states that in an optimal sequence of decisions or choices, each sub sequences must also be optimal.

Example 1: Fibonacci numbers

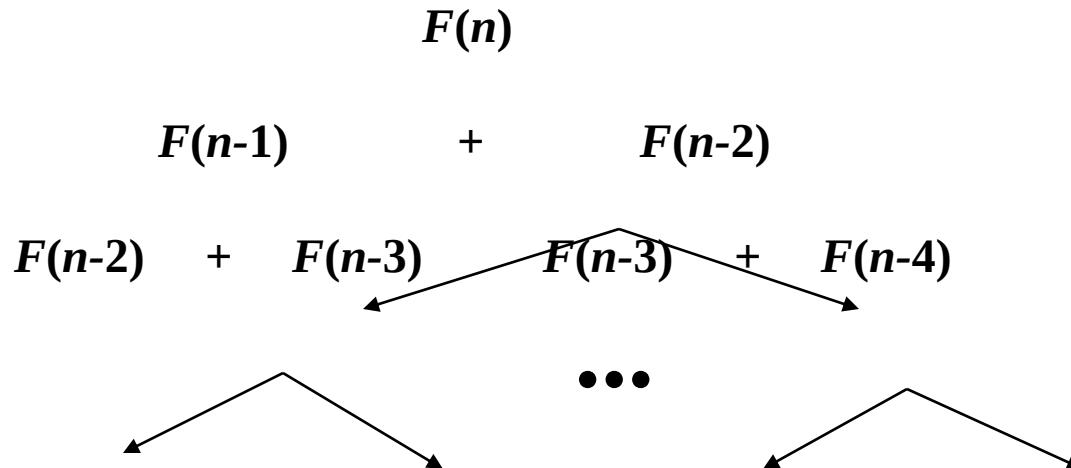
- Recall definition of Fibonacci numbers:

$$F(n) = F(n-1) + F(n-2)$$

$$F(0) = 0$$

$$F(1) = 1$$

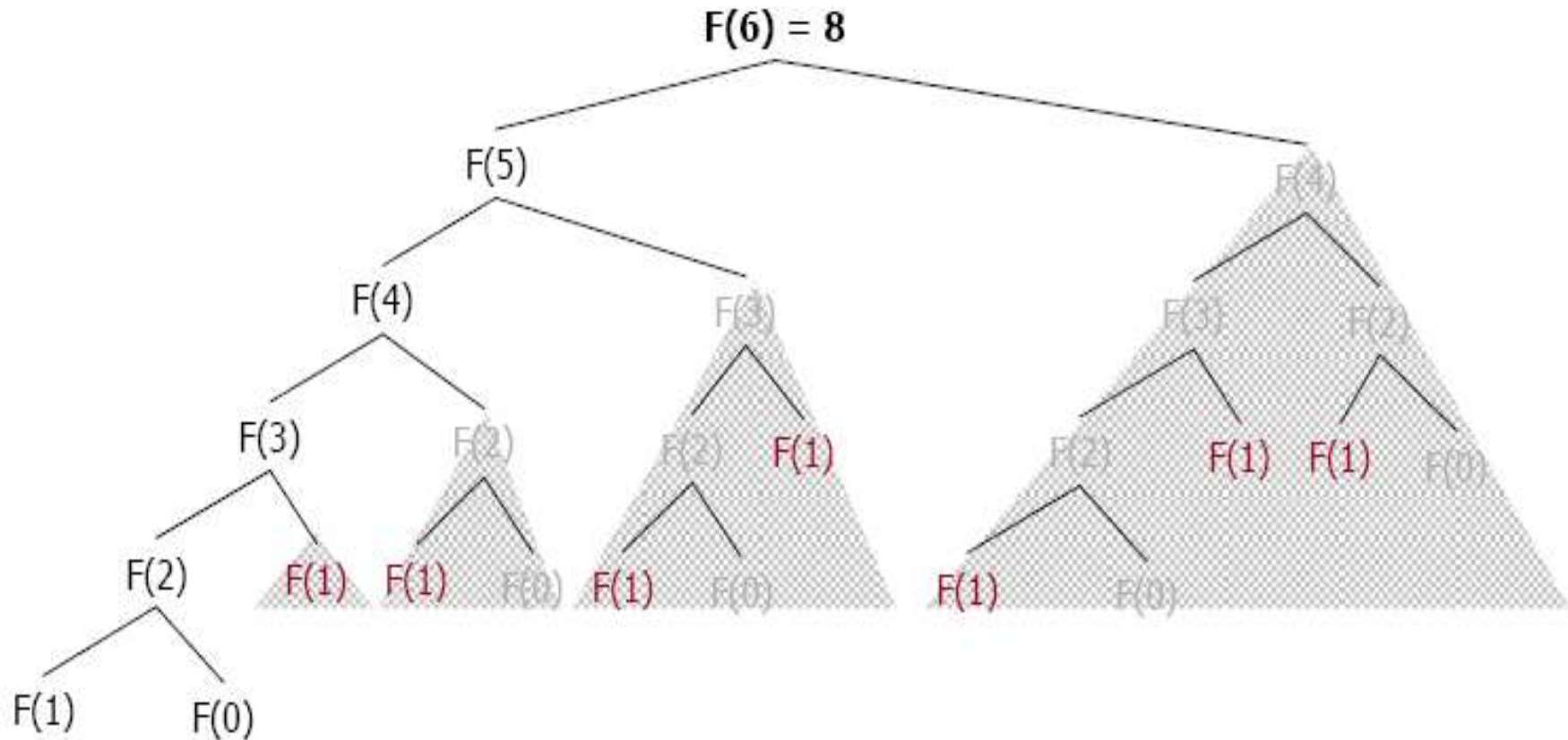
- Computing the n^{th} Fibonacci number recursively (top-down):



Fibonacci Numbers

- $F_n = F_{n-1} + F_{n-2} \quad n \geq 2$
- $F_0 = 0, F_1 = 1$
- 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...
- Straightforward recursive procedure is slow!
- Let's draw the recursion tree

Fibonacci Numbers



- We keep calculating the same value over and over!

Fibonacci Numbers

- We can calculate F_n in **linear time** by remembering solutions to the solved subproblems – *dynamic programming*
- Compute solution in a bottom-up fashion
- In this case, **only two values** need to be remembered **at any time**

```
Fibonacci (n)
   $F_0 \leftarrow 0$ 
   $F_1 \leftarrow 1$ 
  for  $i \leftarrow 2$  to  $n$  do
     $F_i \leftarrow F_{i-1} + F_{i-2}$ 
```

Example Applications of Dynamic Programming

Knapsack Problem

Knapsack problem.

- Given n objects and a "knapsack."
- Item i weighs $w_i > 0$ kilograms and has value $v_i > 0$.
- Knapsack has capacity of W kilograms.
- Goal: fill knapsack so as to maximize total value.

Ex: { 3, 4 } has value 40.

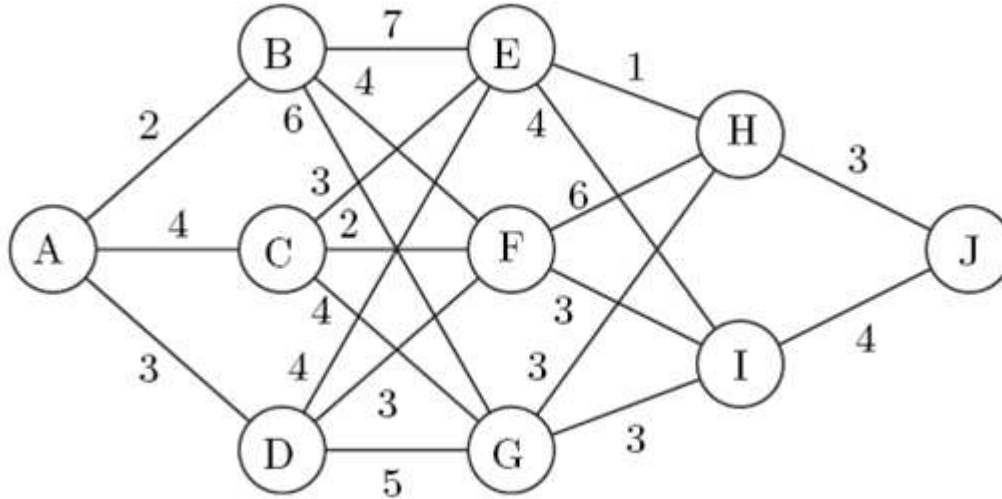
$$W = 11$$

Item	Value	Weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

Greedy: repeatedly add item with maximum ratio v_i / w_i .

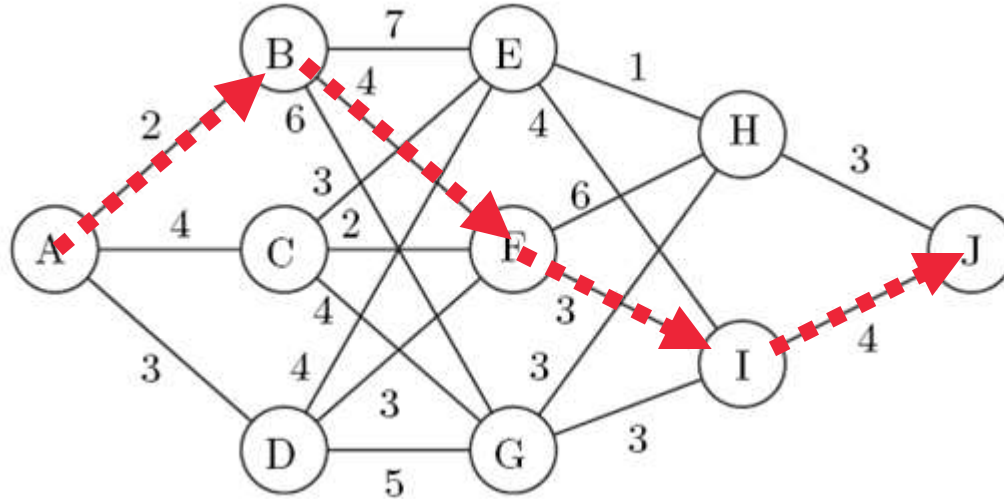
Ex: { 5, 2, 1 } achieves only value = 35 $\Rightarrow$ greedy not optimal.

Shortest path problems



A traveler wishes to minimize the length of a journey from town **A** to **J**.

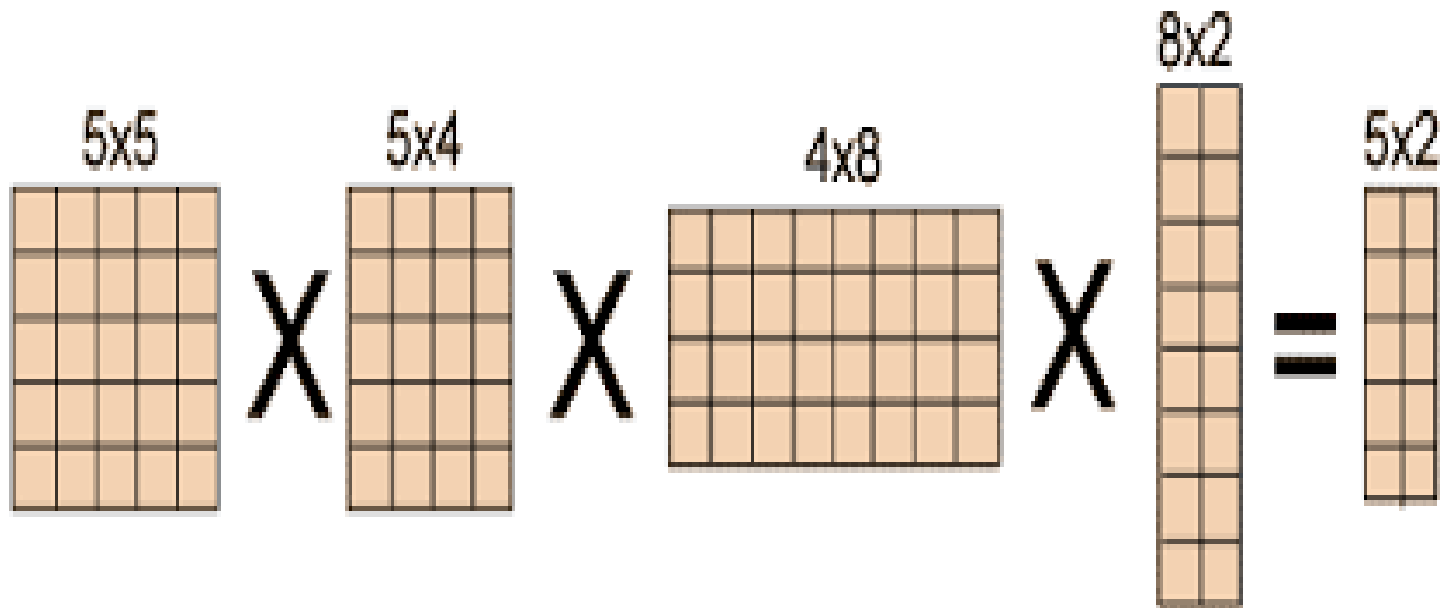
Shortest path problems(Cont..)



The length of the route A-B-F-I-J: $2+4+3+4=13$.

Can we find shorter route?

Matrix chain multiplication



Matrix Chain Multiplication

- Given : a chain of matrices $\{A_1, A_2, \dots, A_n\}$.
- Once all pairs of matrices are *parenthesized*, they can be multiplied by using the standard algorithm as a sub-routine.
- A product of matrices is ***fully parenthesized*** if it is either a single matrix or the product of two fully parenthesized matrix products, surrounded by parentheses. [Note: since matrix multiplication is associative, all parenthesizations yield the same product.]

Matrix Chain Multiplication cont.

- For example, if the chain of matrices is $\{A, B, C, D\}$, the product A, B, C, D can be fully parenthesized in 5 distinct ways:

$(A (B (C D)))$,

$(A ((B C) D))$,

$((A B) (C D))$,

$((A (B C)) D)$,

$((((A B) C) D))$.

- The way the chain is parenthesized can have a dramatic impact on the cost of evaluating the product.

Matrix Chain Multiplication Optimal Parenthesization

- Example: A[30][35], B[35][15], C[15][5]

minimum of $A*B*C$

$$A*(B*C) = 30*35*5 + 35*15*5 = 7,585$$

$$(A*B)*C = 30*35*15 + 30*15*5 = 18,000$$

- How to optimize:
 - Brute force – look at every possible way to parenthesize : $\Omega(4^n/n^{3/2})$
 - Dynamic programming – time complexity of $\Omega(n^3)$ and space complexity of $\Theta(n^2)$.

Matrix Chain Multiplication Structure of Optimal Parenthesization

- For n matrices, let $A_{i..j}$ be the result of $A_i A_{i+1} \dots A_j$
- An optimal parenthesization of $A_i A_{i+1} \dots A_n$ splits the product between A_k and A_{k+1} where $1 \leq k < n$.
- Example, $k = 4$ $(A_1 A_2 A_3 A_4)(A_5 A_6)$

Total cost of $A_{1..6}$ = cost of $A_{1..4}$ plus total cost of multiplying these two matrices together.

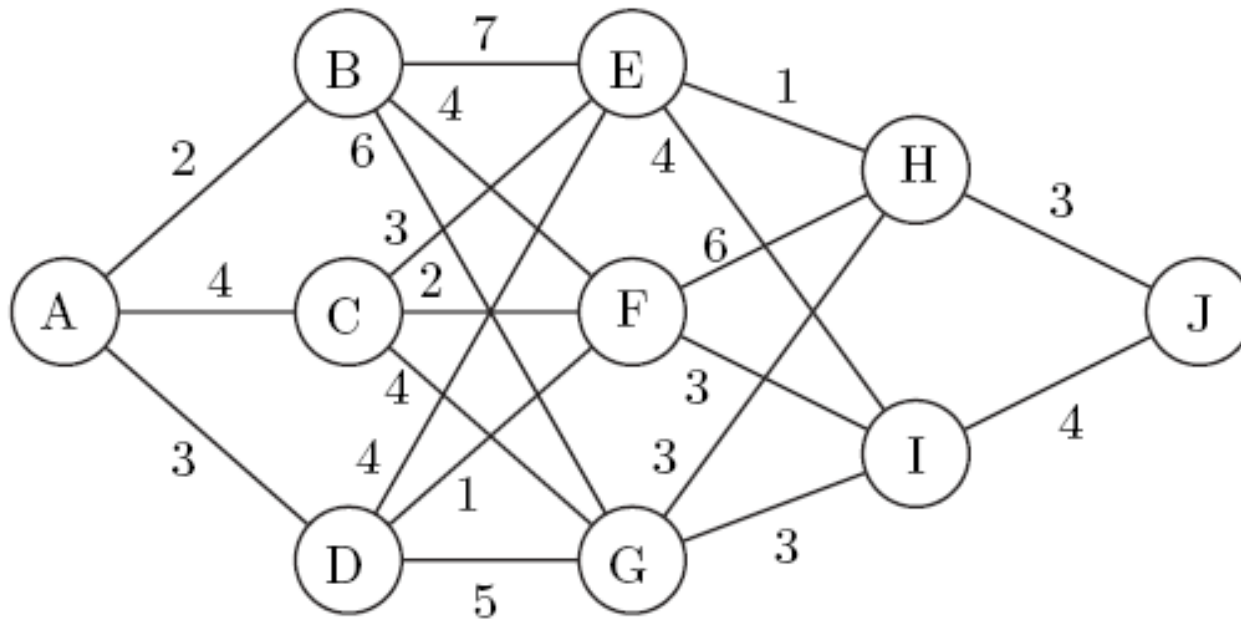
Matrix Chain Multiplication

Overlapping Sub-Problems

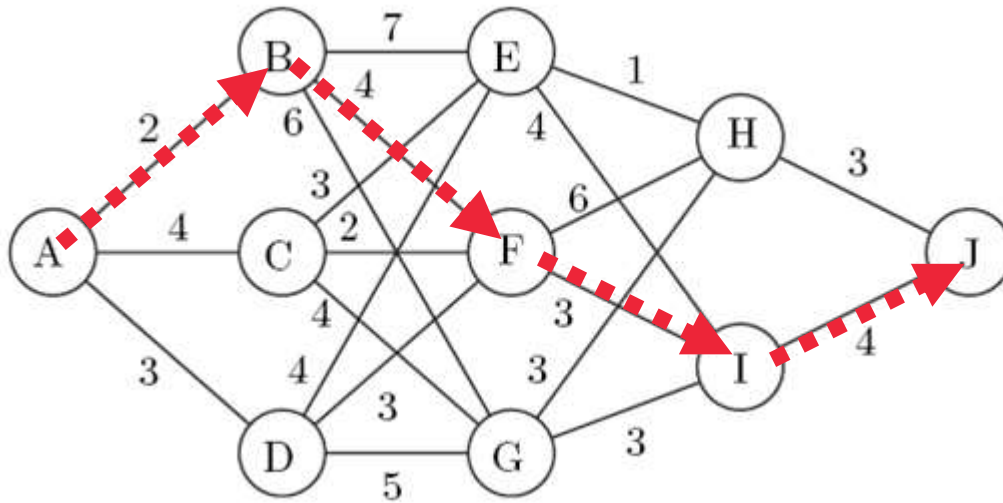
- Overlapping sub-problems helps in reducing the running time considerably.
 - Create a table M of minimum Costs
 - Create a table S that records index k for each optimal sub-problem
 - Fill table M in a manner that corresponds to solving the parenthesization problem on matrix chains of increasing length.
 - Compute cost for chains of length 1 (this is 0)
 - Compute costs for chains of length 2
$$A_{1..2}, A_{2..3}, A_{3..4}, \dots, A_{n-1..n}$$
 - Compute cost for chain of length n

Each level relies on smaller sub-strings

Q1. What is the shortest path?



Answer Q1



The length of the route A-B-F-I-J:
 $2+4+3+4=13$.

Q2. Who invited dynamic programming?

- A. Richard Ernest Bellman
- B. Edsger Dijkstra
- C. Joseph Kruskal
- D. David A. Huffman

Answer Q2

- A. Richard Ernest Bellman
- B. Edsger Dijkstra
- C. Joseph Kruskal
- D. David A. Huffman

Q3. What is meaning of programming in DP?

- A. Plannig
- B. Computer Programming
- C. Programming languages
- D. Curriculum

Q3. What is meaning of programming in DP?

- ☒ A. Plannig
- ☐ B. Computer Programming
- ☐ C. Programming languages
- ☐ D. Curriculum