



04/18/24

- Set: a built-in data type that stores an unordered collection of unique elements.
- ↳ commonly used when the presence of an element in a collection is more important than the order or frequency of elements.
- ↳ Being implemented as hash tables, sets provide average time complexity of $O(1)$ for lookups, which makes them highly efficient for certain operations like checking for memberships, removing duplicates from a sequence, and performing mathematical operations like intersection, union, difference, and symmetric difference.
- ↳ Create set using: {} or set() constructor.
- ↳ Add elements : • add() method
- ↳ Remove " " : • remove() (raises an error if the element is not present).
• discard() (doesn't raise error)
- ↳ Membership test : 'in' keyword to check if an element exists in a set

④ Recursion: fundamental programming concept and technique in which a function calls itself directly or indirectly to solve a problem.

↳ It breaks down a large problem into smaller, more manageable parts, each of which is solved by a recursive call.

Key Concepts:

① **Recursive Case:** part of recursion that splits the problem into sub-problems and calls the same func to solve these sub-problems

② **Base Case:** Also known as terminating condition, this stops the recursion.

③ **Stack:** Each recursive call adds a layer to the stack. When a recursive call reaches the base case, the stack begins to unwind as each call completes and returns its result.

Used in Tree traversal and sorting algorithms.

Optimize by:

↳ Tail recursion: → special kind
→ recursive call is the last operation in the func.

↳ Memoization.

→ Dynamic Programming ↗

↳ The basic idea of DP is :

- Identifying and solving subproblems.
- using subproblems together to solve larger problem.

5 simple steps for solving DP problems:

↳ Using 2 problems:

1. longest Increasing Subsequence (LIS):

For a sequence a_1, a_2, \dots, a_n , find the length of the longest increasing subsequence
 $a_{i_1}, a_{i_2}, \dots, a_{i_k}$

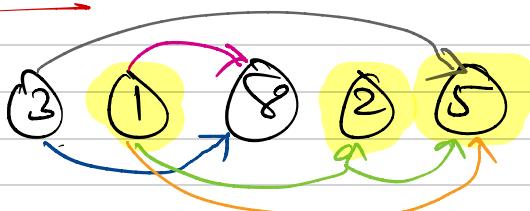
Constraints: $i_1 < i_2 < \dots < i_k$;
 $a_{i_1} < \dots < a_{i_k}$

[Focus on length]

Ex: LIS([3 1 8 2 5]) → 3 ^{of LIS}

LIS([5 2 8 6 3 6 9 5]) → 4

Step 1. Visualize Ex:-



LIS = Longest Path in DAG + 1

2. Find an appropriate subproblem

↳ simpler version of our ^{main} problem.

→ Here, All increasing subsequences are subsets of original sequence.

→ All increasing subsequences have a start and end.

→ Let's focus on the end index of an increasing subsequence

So, here : subproblem : LIS [k] = LIS ending at index k

Ex: LIS[3] = 2

3. Find relationship among subproblems

→ Dynamic Programming: Abdul Basit

Diffr b/w Greedy Method & DP:

↳ Both are used for solving optimization problems

↳ both are different strategies but the purpose is same.

Optimization problems are those which requires minimum result or maximum result.

→ In Greedy Method, we try to follow predefined procedure to get optimal result.

↳ The procedure is known to be optimal, we follow the procedure to get best result like Kruskal's method to find minimum cost spanning tree → always select a minimum cost edge and that gives us best result.

*→ Mostly DP problems are solved by using recursive formulas.

↳ Though we won't use recursion of programming but the formulas are recursive.

⇒ DP → follows Principle of Optimality

it says that a problem can be solved by taking sequence of decisions to get the optimal solution.

In Greedy → decision taken only one time.

Difference b/w Tabulation and Memoization:

A recursive ex: Memoization

Fibonacci Series:

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

fib. seq = 0, 1, 1, 2, 3, 5, 8, 13... -

Put fib (int n)

{

if (n <= 1)

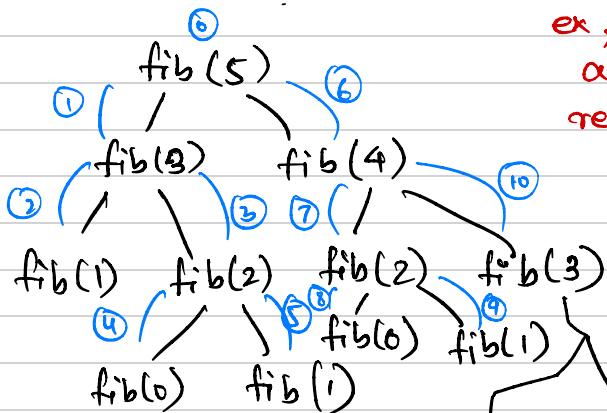
return n;

return fib(n-2) + fib(n-1);

}

This code follows recursion

for ex: fib(5) = ?



In memoization, in this ex, we create an array to store the results of each fib(n) we need to get the ultimate result.

If we do not use memoization, it will take 15 calls to add each stage and move up to finally get the o/p.

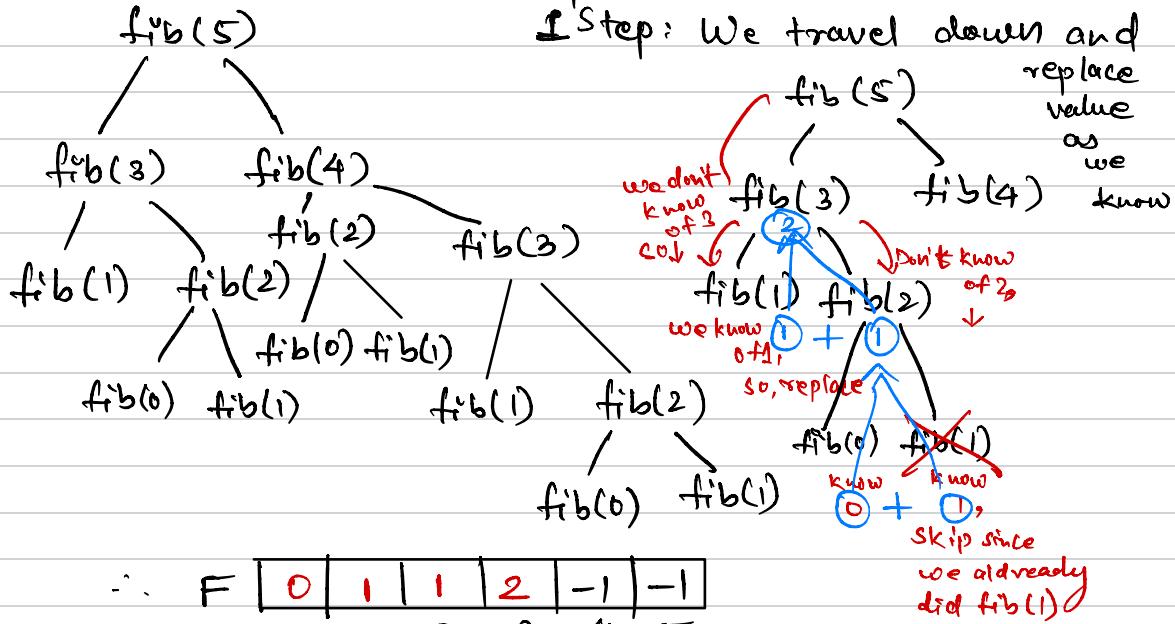
Time Complexity without memoization : $O(2^n)$

Now: memoization :

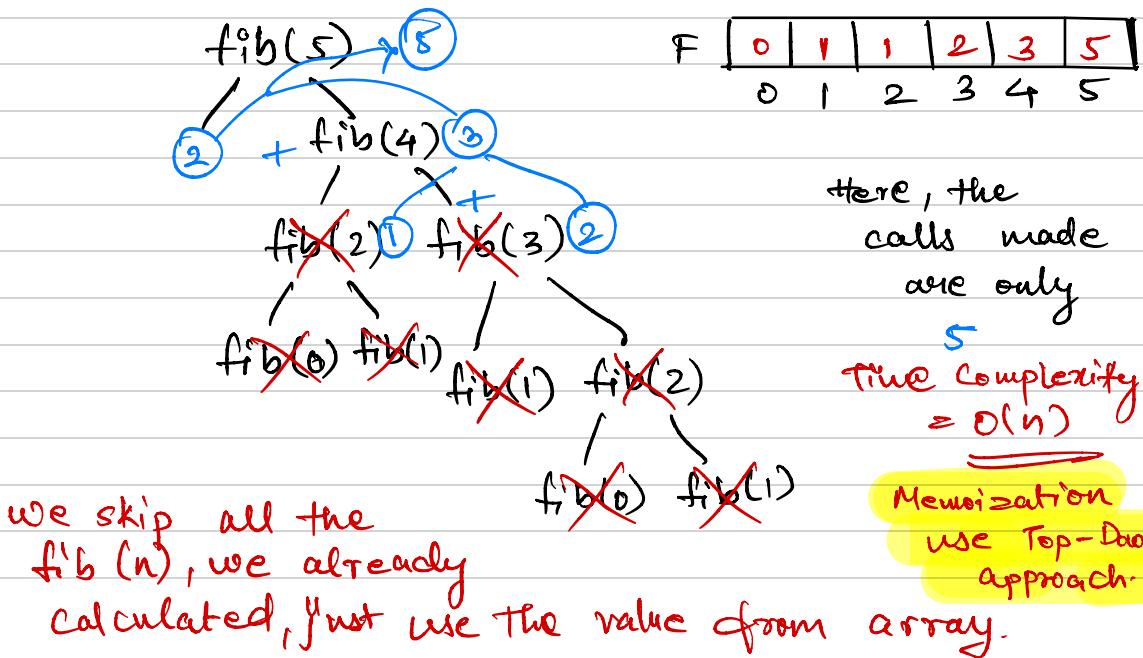
F	-1	-1	-1	-1	-1	-1
	0	1	2	3	4	5

OK, so we'll initially fill the array with -1, since we do not know any value right now and we'll replace the values as we know.

F	1	-1	-1	-1	-1	-1
	0	1	2	3	4	5



2nd Step: We travel remaining side.



Tabulation method: we iterative approach

Bottom-Top approach

We create a table and fill it as we get the values to get the ultimate o/p.

int fib(int n)

{ if ($n \leq 1$)

 return n;

 F[0] = 0; F[1] = 1;

 for (int i = 2; i <= n; i++)

{

 F[i] = F[i-2] + F[i-1];

}

 return F[n];

}

F	0	1	1	2	3	5
	0	1	2	3	4	5

↑
1st
step
2nd
3rd
4th
5th
6th

Multistage Graph: Problem

↳ is a directed weighted graph where vertices are divided into stages such as the edges are connecting vertices from one stage to next stage.

First stage and last stage have only one single vertex to represent the starting point (aka source) or ending point (aka sink of a graph).

↳ usually useful for representing resource allocation.

↳ Objective of the problem:

↳ Objective is to find the shortest path
b/w starting point and sink giving the minimum cost.

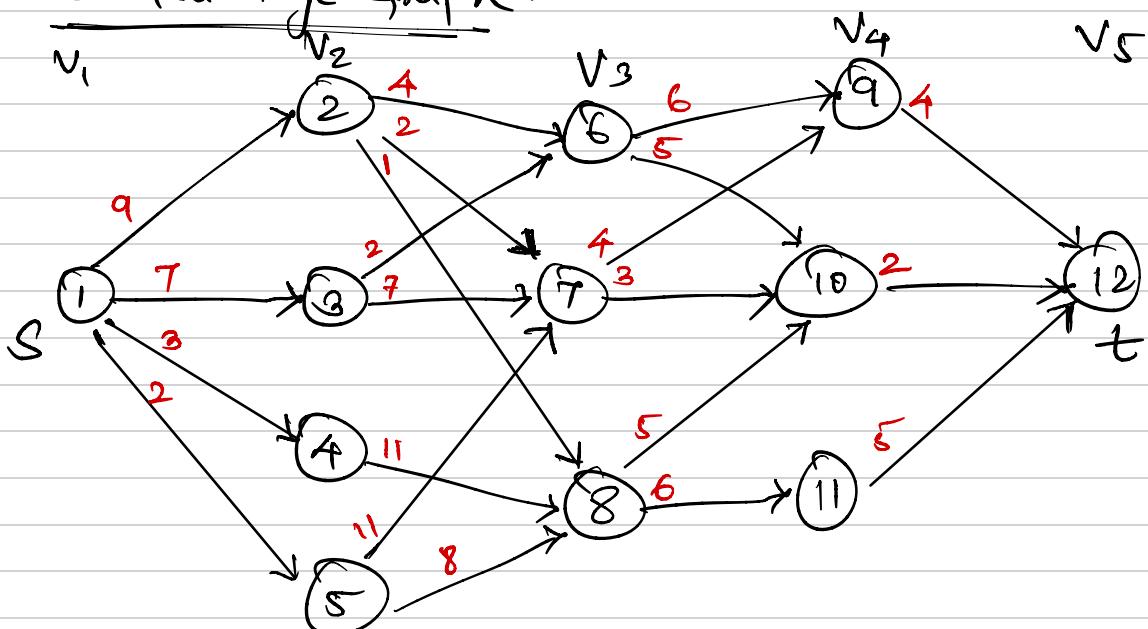
↳ A minimization problem \rightarrow an optimization problem

↳ can be solved with DP.

1st

↳ Check if the problem can be solved by DP. DP works on principle of optimality starting a problem must be solved in seq of decisions.

The Multistage Graph:



Since, we travel stage by stage, we have to choose the most optimal route \rightarrow stating principle of optimality \therefore DP can be used.

↳ we'll use tabulation method to get/fill data in the table.

for each vertex, calculate the cost and to do that start with sink because if we start from earlier stages, we have to calculate all the possible cost (by travelling through all possible edges to the sink).

V	1	2	3	4	5	6	7	8	9	10	11	12
cost						7 10	5 10		4 12	2 12	5 12	0 12
of												

↳ write down the vertex that gave shortest path (min. cost)

To calculate cost:

stage vertex
↓
cost (5, 12) = 0

$$\text{cost}(4, 9) = 4$$

$$\text{cost}(4, 10) = 2$$

$$\text{cost}(4, 11) = 5$$

$$\text{cost}(3, 6) = \min \left\{ \begin{matrix} \text{vertex} & \text{stage vertex} \\ \text{c}(6, 9) + \text{c}(4, 9), & \\ \text{c}(6, 10) + \text{c}(4, 10) \end{matrix} \right\} + \min \left\{ \begin{matrix} 6+4, \\ 5+2 \end{matrix} \right\}$$

$$= \min(10, 7)$$

$$= 7$$

$$\text{cost}(3, 7) = \min \left\{ \begin{matrix} \text{c}(7, 9) + \text{c}(4, 9), \\ \text{c}(7, 10) + \text{c}(4, 10) \end{matrix} \right\} = \min(4+4, (3+2))$$

$$= \min(8, 5)$$

$$= \underline{\underline{5}}$$

V	1	2	3	4	5	6	7	8	9	10	11	12
Cost	16	7	9	18	15	7	5	7	4	2	5	0
Q	2/3	7	6	8	8	10	10	10	12	12	12	

write down the vertex that gave you shortest path

up continuing

$$C(3,8) = \min \left\{ C(8,10) + C(4,10), C(8,11) + C(4,11) \right\} = \min \left\{ (5+2), (6+5) \right\} = \min (7, 11) = 7$$

$$C(2,2) = \min \left\{ C(2,6) + C(3,6), C(2,7) + C(3,7), C(2,8) + C(3,8) \right\} = \min \left\{ (4+7), (2+5), (1+7) \right\} = \min (11, 7, 8) = 7$$

$$C(2,3) = \min \left\{ C(3,6) + C(3,6), C(3,7) + C(3,7) \right\} = \min \left\{ (2+7), (7+5) \right\} = \min (9, 12) = 9$$

$$C(2,4) = C(4,8) + C(3,8) = 11 + 7 = 18$$

$$C(2,5) = \min \left\{ C(5,7) + C(3,7), C(5,8) + C(3,8) \right\} = \min \left\{ (11+5), (8+7) \right\} = \min (16, 15) = 15$$

$$\text{cost}(1,1) = \min \left\{ C(1,2) + C(2,2), C(1,3) + C(2,3), C(1,4) + C(2,4), C(1,5) + C(2,5) \right\} = \min \left\{ (9+7), (7+9), (3+18), (2+15) \right\} = \min (16, 16, 21, 17)$$

This is called forward method.

formula we used:

$$\text{cost}(i, j) = \min_{\substack{\text{vertex no. } i, j \\ \langle j, l \rangle \in E, \text{ level } l}} \{ c(j, l) + \text{cost}(i+1, l) \}$$

$$\text{Ex: cost}(2, 3) = \min_{i, j} \left\{ \begin{array}{l} c(3, 6) + \text{cost}(3, 6), \\ c(j, l) + \text{cost}(i+1, l) \\ c(3, 7) + \text{cost}(3, 7) \end{array} \right\}$$

Now, DP:

↳ based on data available:

↳ we take decisions of where to go from each stage.

We use this:

V	1	2	3	4	5	6	7	8	9	10	11	12
Cost	16	7	9	18	15	7	5	7	4	2	5	0
d	2/3	7	6	8	8	10	10	10	12	12	12	0

↳ write down the vertex that gave you shortest path

↳ We start from vertex 1. (in forward direction)

$$d(1, 1) = 2$$

$$d(4, 10) = 12 \rightarrow \text{shortest path}$$

$$d(2, 2) = \begin{cases} 7 & \text{if we choose 2 out of 2/3} \end{cases}$$

$$d(3, 7) = 10$$

if we choose 3

$$\begin{aligned}d(1,1) &= 3 \\d(2,3) &= 6 \\d(3,6) &= 10 \\d(4,10) &= 12\end{aligned}$$

→ Shortest path

Two paths with same cost

Now, Same with Program:

To solve this, we use two formulas :

• Calculate the cost of vertex :

$\xrightarrow{\text{this symbolizes cost from i to k}}$

$$\text{cost}[i] = \{ c[i][k] + \text{cost}[k] \}$$

$\uparrow \quad \uparrow \quad \uparrow$

vertex ($i+1 \rightarrow n$) cost of k

\uparrow
i.e. cost of
 $i+1 \rightarrow n$

• $p[i] = d[p[i-1]]$ To find the
shortest path

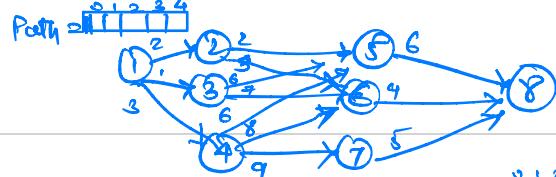
• We will use graph, cost adjacency matrix representing the cost of the edges, arrays as data structures.

shortest
path vertex

• Three arrays: cost, d, path

cost	0	1	2	3	4	5	6	7	8
d	1	1	1	1	1	1	1	1	1
	0	1	2	3	4	5	6	7	8

Ex code:



c	0	1	2	3	4	5	6	7	8
0	1	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0
4	0	0	0	0	1	0	0	0	0
5	0	0	0	0	0	1	0	0	0
6	0	0	0	0	0	0	1	0	0
7	0	0	0	0	0	0	0	1	0
8	0	0	0	0	0	0	0	0	1

main()

{ int stages = 4, min;

int n=8;

int cost[9], d[9], Path[9];

int C[9][9] = { { 0, 0, 0, 0, 0, 0, 0, 0, 0 },

{ fill the values of the row in C },

 { .. };

:

} ;

cost[n]=0;

for (int i=n-1; i>=1; i++)

{ min=32767;

 for (k=i+1; k<n; k++)

{

 if (C[i][k]==0 && c[i][k]+c[k]<min)

{ min=c[i][k]+c[k];

 d[i]=k;

}

}

 cost[i]=min;

}

P[1]=1; P[stages]=n;

for (i=2; i<stages; i++) P[i]=P[d[i-1]];

Time Complexity = O(n²)