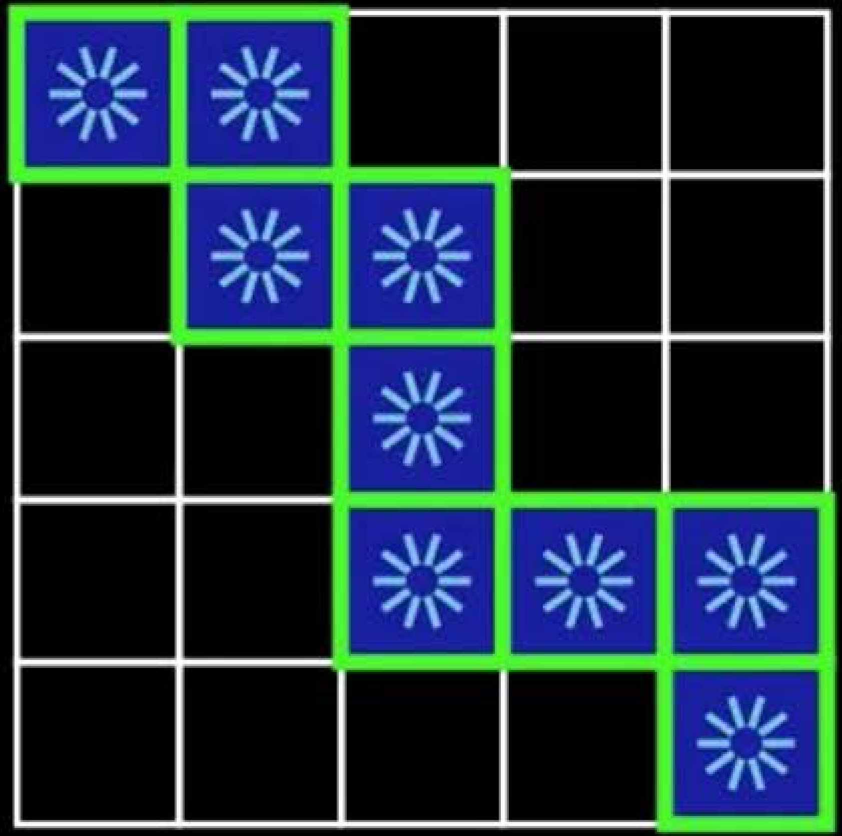
Lecture -1

Introduction



Dynamic Programming

- Dividing problem in to sub problem then combining the Result

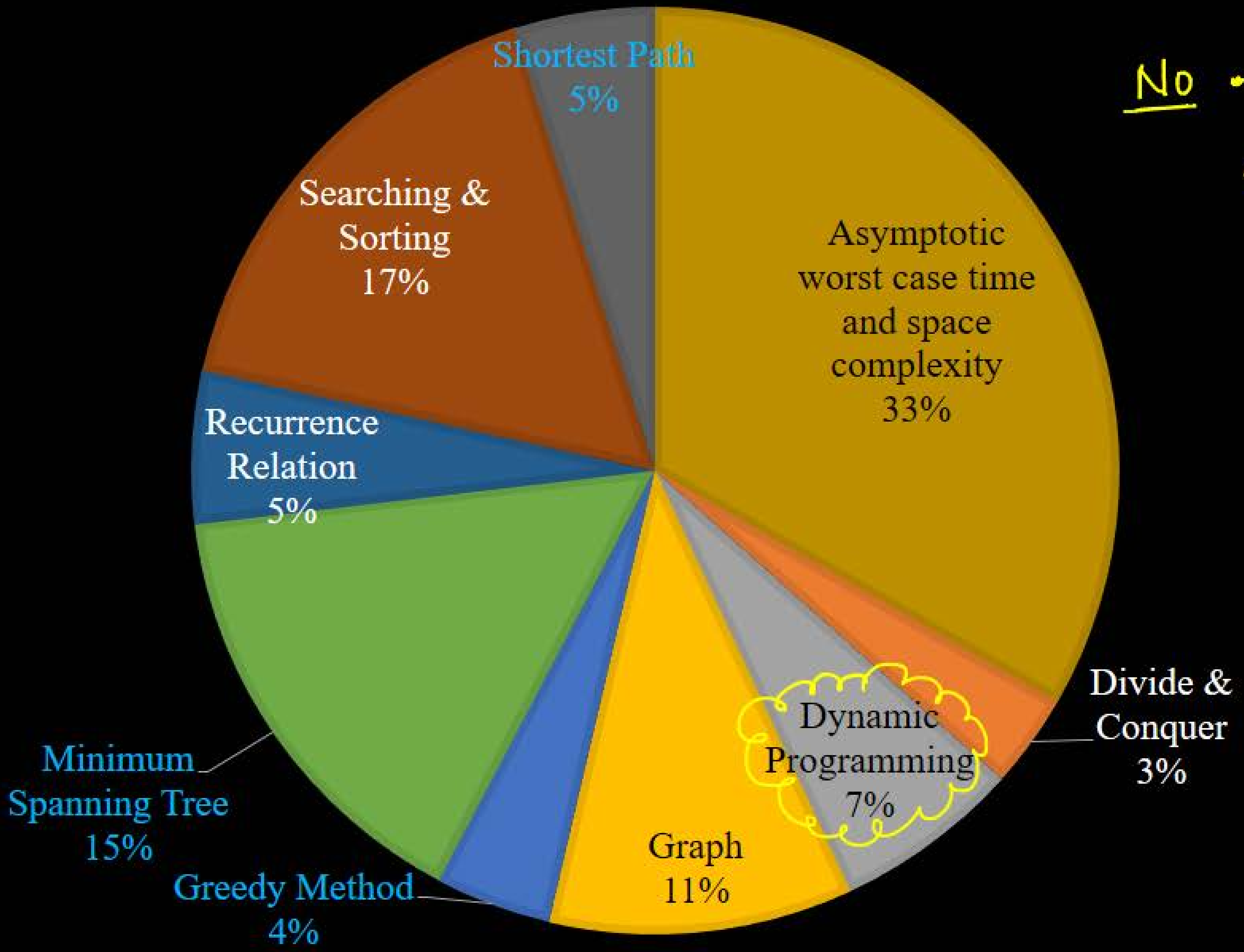
Recursion - ~~Mathatics~~ Maths

- Greedy - Simple ~~op~~ objective function

order the set n element

select one

Dynamic Programming Analysis



No · powerful

Sub-Problem

Divide & Conquer Algorithm also divides the problem into subproblem.

Smaller instance of original problem is called Subproblem.

Main difference

optimization -

Greedy -

D&C - No optimization.

Sub-Problem

- A subproblem is a subparts of the main problem that is an integral part of the main problem.
- Example: Calculation of Fib(6) required calculation of Fib(5)

Sub-Problem

- Dynamic programming, like the divide-and-conquer method, solves problems by combining the solutions to subproblems.

Dynamic Programming

- Dynamic programming, like the divide-and-conquer method, solves problems by combining the solutions to subproblems.
- “Programming” in this context refers to ***a tabular method, not to writing computer code***

Overlapping Subproblem

fib(5)

fib(4) fib(3)

divided in to subproblem

Subproblem - fib(5)

fib(4) + fib(3)

fib(3) fib(2)

Question is ~~fib~~ for solving fib(4) do i need to solve fib(3).

fib(3) Solved multiple times (2 times in computing fib(5))

Overlapping Subproblem : In Dynamic Programming problem divided into Subproblem . and ~~pr~~ Subproblem Share Common ~~Subproble~~ sub - sub problem. The common problem can be Solved once and Reuse the Result again.

Overlapping Subproblem

- In contrast, dynamic programming applies when the ***subproblems overlap***—that is, when subproblems share subsubproblems.
- Solving fib(6) , how many times fib(4) is solved?

Overlapping Subproblem

- In this context, a divide-and-conquer algorithm does more work than necessary, ***repeatedly solving the common subsubproblems***.

Optimization problem

- Subproblem
- overlapping Sub problem
-

- Given all fesible Solution the Solution that maximize & minimize the objective function is called optimization problem.

Optimization problem

- In mathematics, computer science and economics, an optimization problem is **the problem of finding the best solution from all feasible solutions**.

Optimization problem

- We typically apply ***dynamic programming to optimization problems***. Such problems can have many possible solutions.

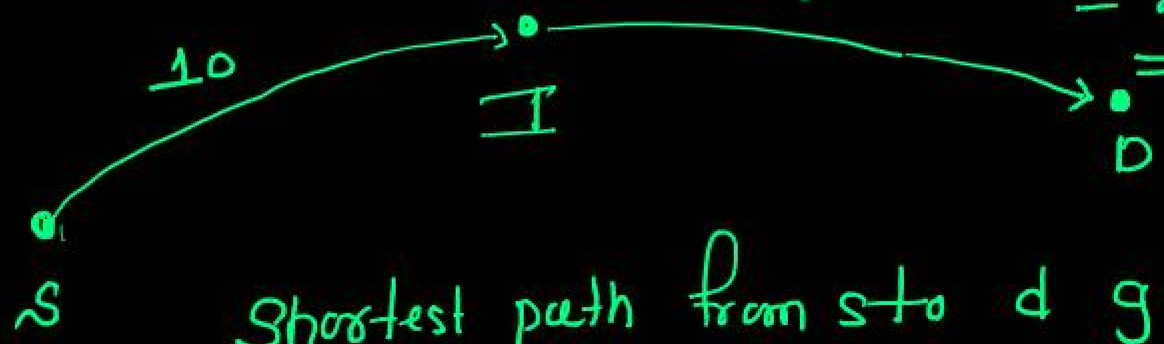
Optimization problem

- Each solution has a value, and we wish to find a solution with the optimal (minimum or maximum) value. We call such a solution an optimal solution to the problem, since there may be several solutions that achieve the optimal value.

Optimal Substructure

- Subproblem
- overlapping Subproblem
- optimization
- ~~Sup~~ Subproblem & optimization.

Shortest path



10+15
= 25 yes

Original proble
shortest path S — D

Shortest path from s to d goes through
I.

Break this in subproblem

S → I
I → D

Optimal Substructure

optimal solution of the problem can be found by combining optimal solution of Subproblem

Optimal Substructure

- Optimal Substructure: A given problems has Optimal Substructure Property *if optimal solution of the given problem can be obtained by using optimal solutions of its subproblems.*

Optimal Substructure

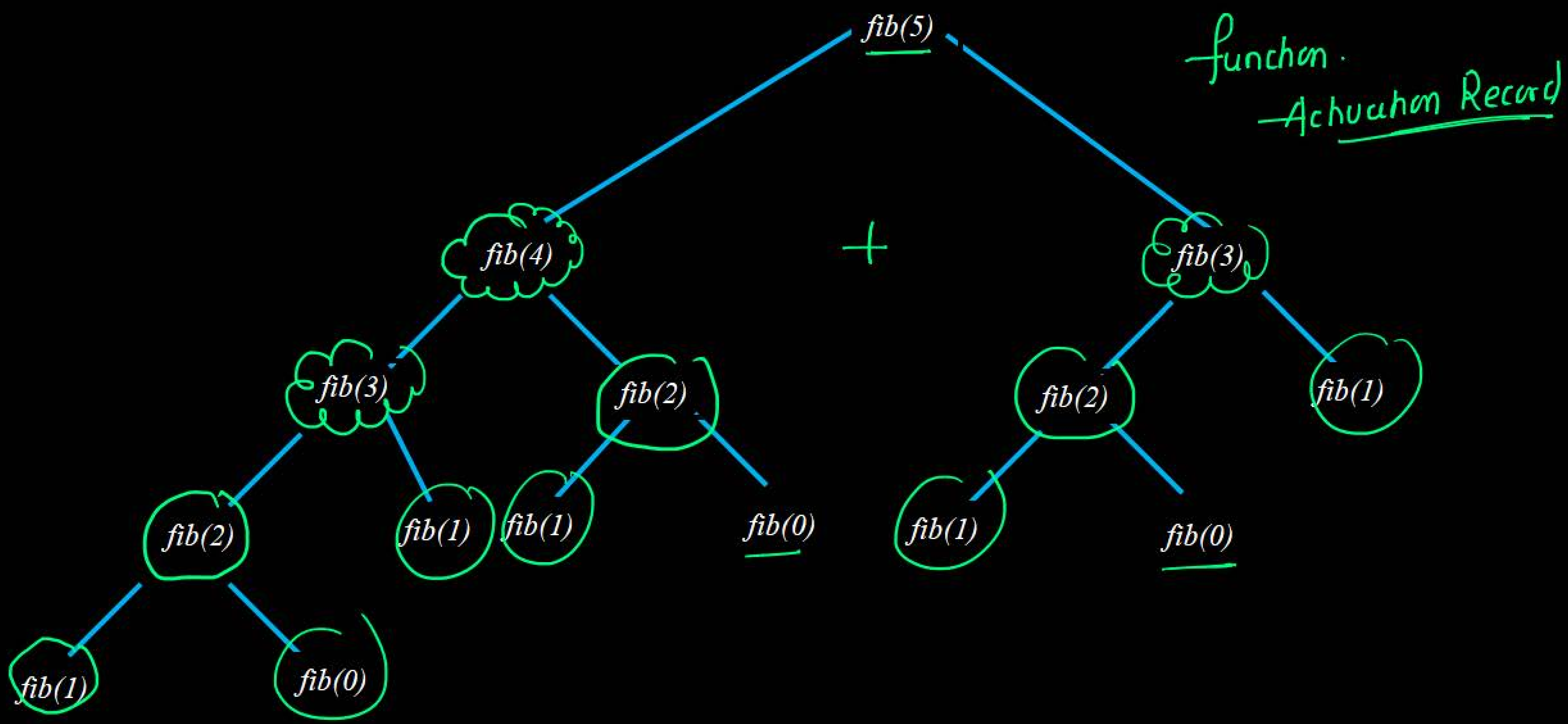
- For example, the Shortest Path problem has following optimal substructure property:
- If a node x lies in the shortest path from a source node u to destination node v then the shortest path from u to v is combination of shortest path from u to x and shortest path from x to v.

Dynamic Programming

- Subproblem
- Overlapping subproblem
- Optimization Problem
- Optimization substructure exists

problem

The Fibonacci function



The Fibonacci function

Overlapping Sub pr

The Fibonacci function

We can see that the function fib(3) is being called 2 times. If we would have stored the value of fib(3), then instead of computing it again, we could have reused the old stored value.

Store the value fib(3)

Second instance

Table

Dynamic Programming

- There are following two different ways to store the values so that these values can be reused:

  Example

  - Top Down (Memorization) → Changing Recursive structure of program we will store value once it computed
  - Bottom up (Tabulation Method) ←

Memorization (Top Down)

- Memoization (Top Down): The memoized program for a problem is similar to the recursive version with a small modification that looks into a lookup table before computing solutions. We initialize a lookup array with all initial values as NIL.
- Whenever we need the solution to a subproblem, we first look into the lookup table.
- If the precomputed value is there then we return that value, otherwise, we calculate the value and put the result in the lookup table so that it can be reused later.

Algorithm using Memorization

fib(0) - 0
fib(1) = 1

array ← maximum

```
int lookup[MAX];
int fib(int n)
{
    if (lookup[n] == NIL) {
        if (n <= 1)
            lookup[n] = n;
        else
            lookup[n] = fib(n - 1) + fib(n - 2);
    }
    return lookup[n];
}
```

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

returned

fib(5)

look

lookup[5] = fib(4) + fib(3)

Lookup(4) = fib(3) + fib(2)

Lookup[3] = fib(2) + fib(1)

Lookup[2] = fib(1) + fib(0)

Tabulation (Bottom Up)

- Tabulation (Bottom Up): The tabulated program for a given problem builds a table in bottom-up fashion and returns the last entry from the table. For example, for the same Fibonacci number, we first calculate fib(0) then fib(1) then fib(2) then fib(3), and so on. So literally, we are building the solutions of subproblems bottom-up.

Tabulation (Bottom Up)

```
int fib(int n) {
    int f[n + 1];
    int i;
    f[0] = 0;
    f[1] = 1;
    for (i = 2; i <= n; i++)
        f[i] = f[i - 1] + f[i - 2];
     return f[n];
}
```

Smallest value to Largest

fib(5)

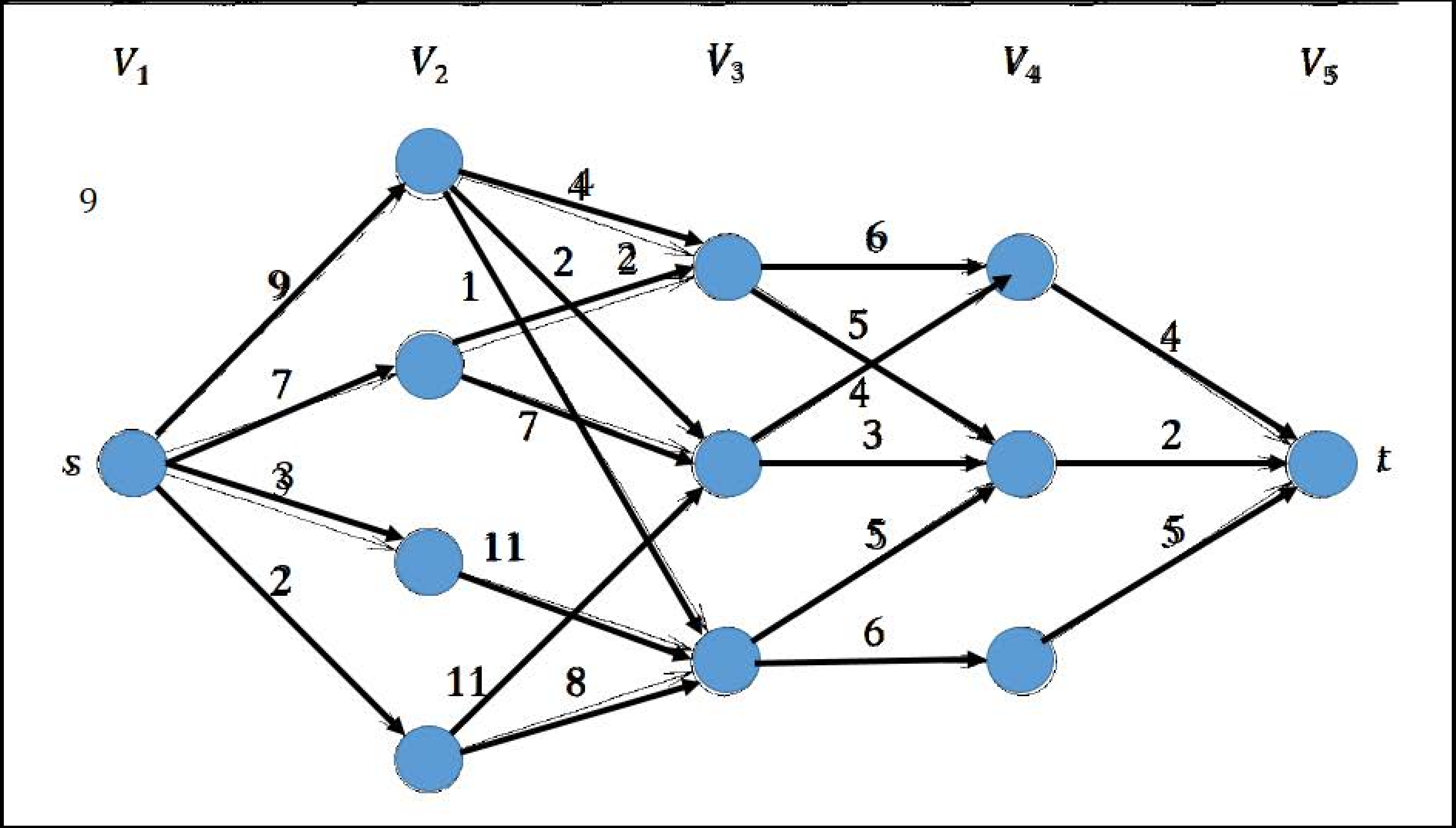
fib(1) fi(0)

Steps to follow

Steps: we follow a sequence of four steps:

1. Characterize the structure of an optimal solution.
2. Recursively define the value of an optimal solution.
3. Compute the value of an optimal solution, typically in a bottom-up fashion.
4. Construct an optimal solution from computed information.

Multistage Graph

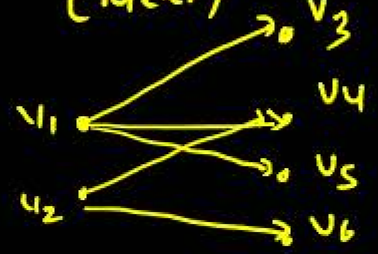


V_1
V_2
V_3
V_4
V_5
9
4
6
9
2
2
1
5
4
7
4
7
3
2
s
3
t
11
5
5
2
6
11
8

Multistage Graph

- A Multistage graph is a directed graph in which the nodes can be divided into *a set of stages such that all edges are from a stage to next stage only* (In other words there is no edge between vertices of same stage and from a vertex of current stage to previous stage).

- Vertex divided into stages Such that there are edges going from one stage to another (Next) and Not on same stage



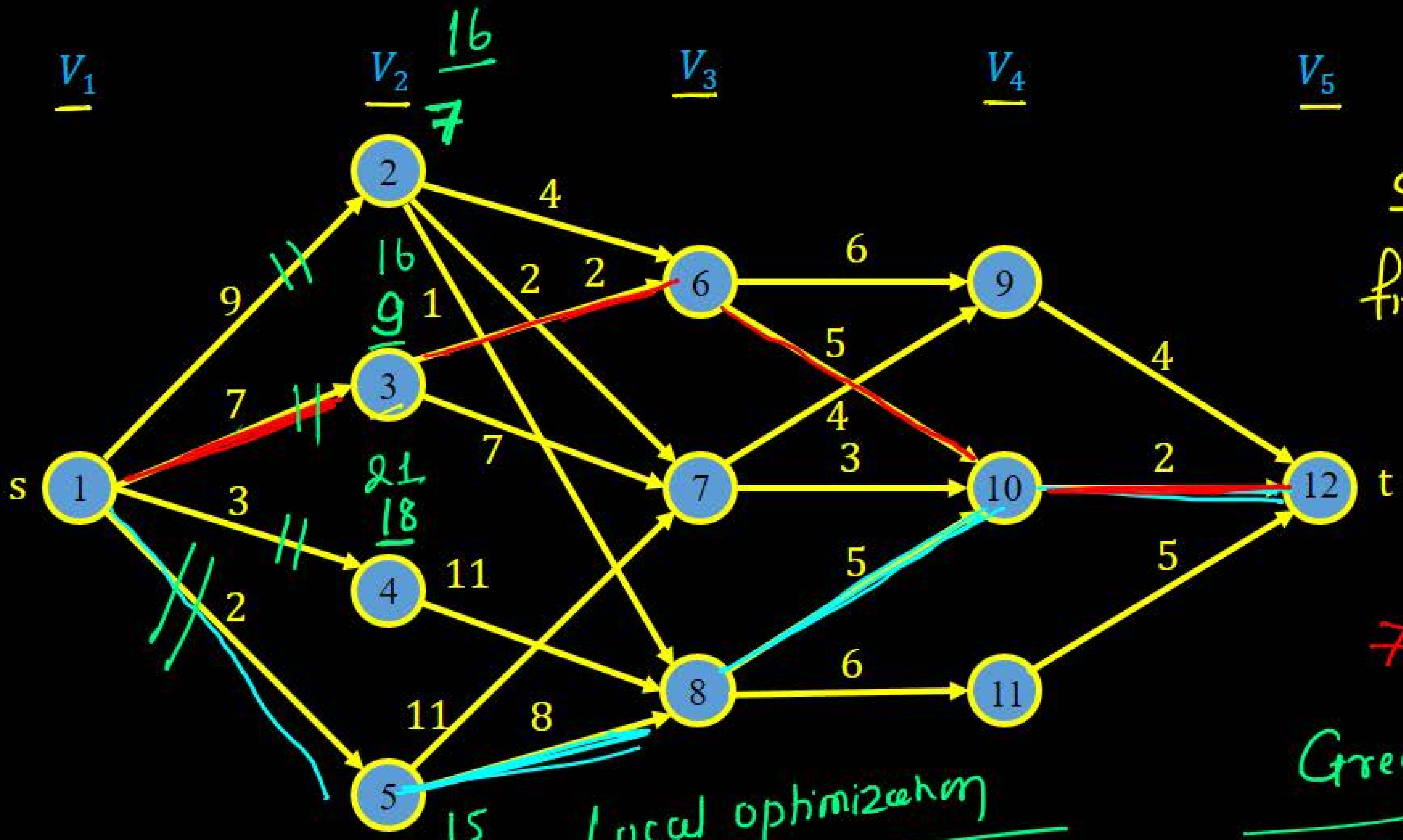
- Single pair shortes path
- we Need to find shortest path between a pair of vertices
- Greedy Method dijktra's Method is Single Source shortest path to all vertices (Extra work)

Multistage Graph

- We are given a multistage graph, a source and a destination, *we need to find shortest path from source to destination*. By convention, we consider source at stage 1 and destination as last stage.

pair of vertex
we Need to find
shortest pat

Multistage Graph



shortest path between first stage & Last stage.

$2+8+5+2$

$=17$

$7+2+5+2=16$

Greedy Method

15

Local optimization

17

Now Global optimization

Local information

Different strategies

- The ***Brute force*** method of finding all possible paths between Source and Destination and then finding the minimum. That's the WORST possible strategy.

Different strategies

- ***Dijkstra's Algorithm*** of Single Source shortest paths. This method will find shortest paths from source to all other nodes which is not required in this case. So it will take a lot of time and it doesn't even use the SPECIAL feature that this MULTI-STAGE graph has.

Different strategies

Simple Greedy Method – At each node, choose the shortest outgoing path. If we apply this approach to the example graph give above we get the solution as 17. But a quick look at the graph will show much shorter paths available than 16. So the greedy method fails !

Different strategies

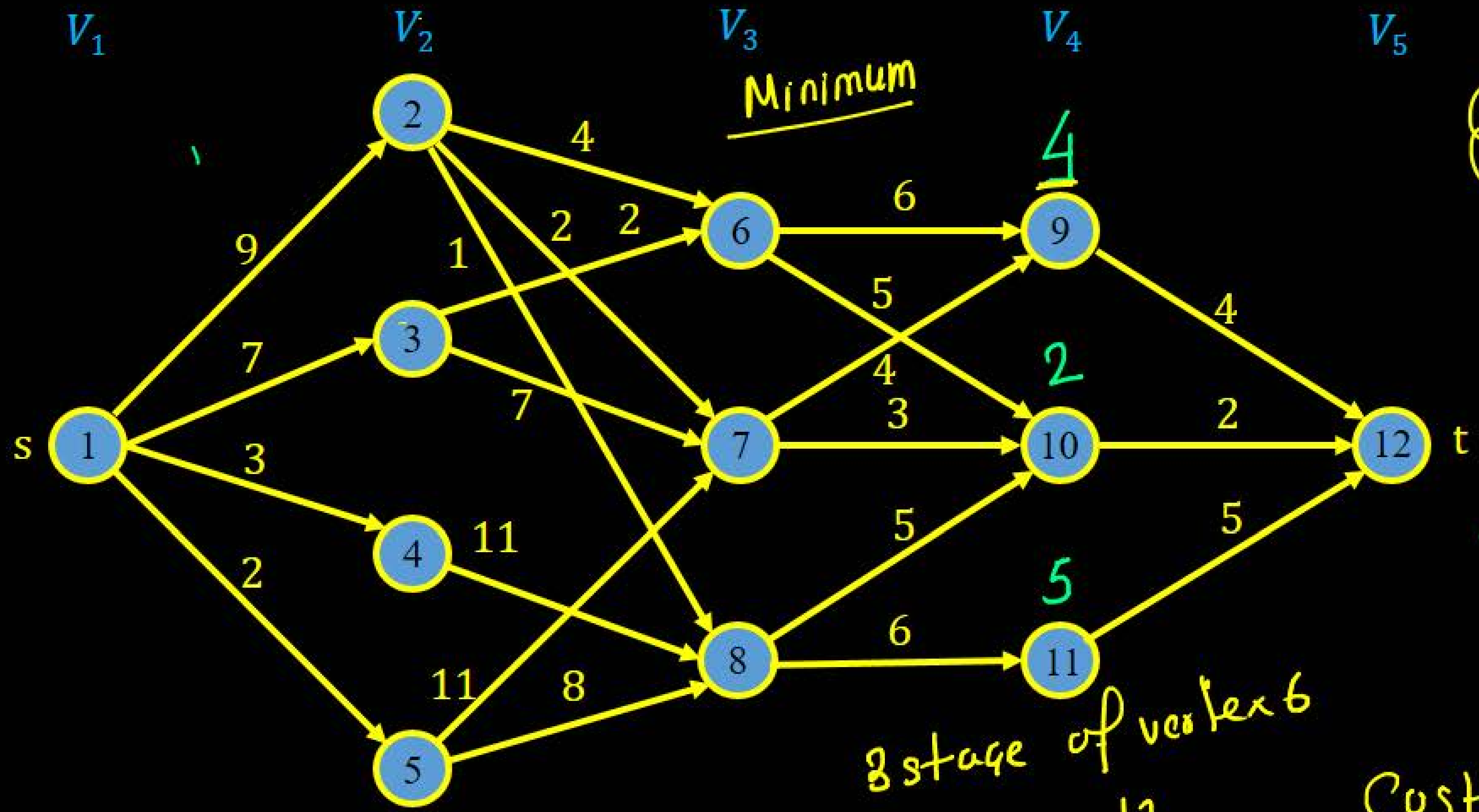
The best option is Dynamic Programming. So we need to find *Optimal Sub-structure, Recursive Equations and Overlapping Sub-problems.*

Optimal Substructure Properties

Simple Greedy Method – At each node, choose the shortest outgoing path. If we apply this approach to the example graph give above we get the solution as 17. But a quick look at the graph will show much shorter paths available than 16. So the greedy method fails !

Optimal Substructure Properties

The best option is Dynamic Programming. So we need to find *Optimal Sub-structure, Recursive Equations and Overlapping Sub-problems.*



$Cost(4, 9)$

stage vertex

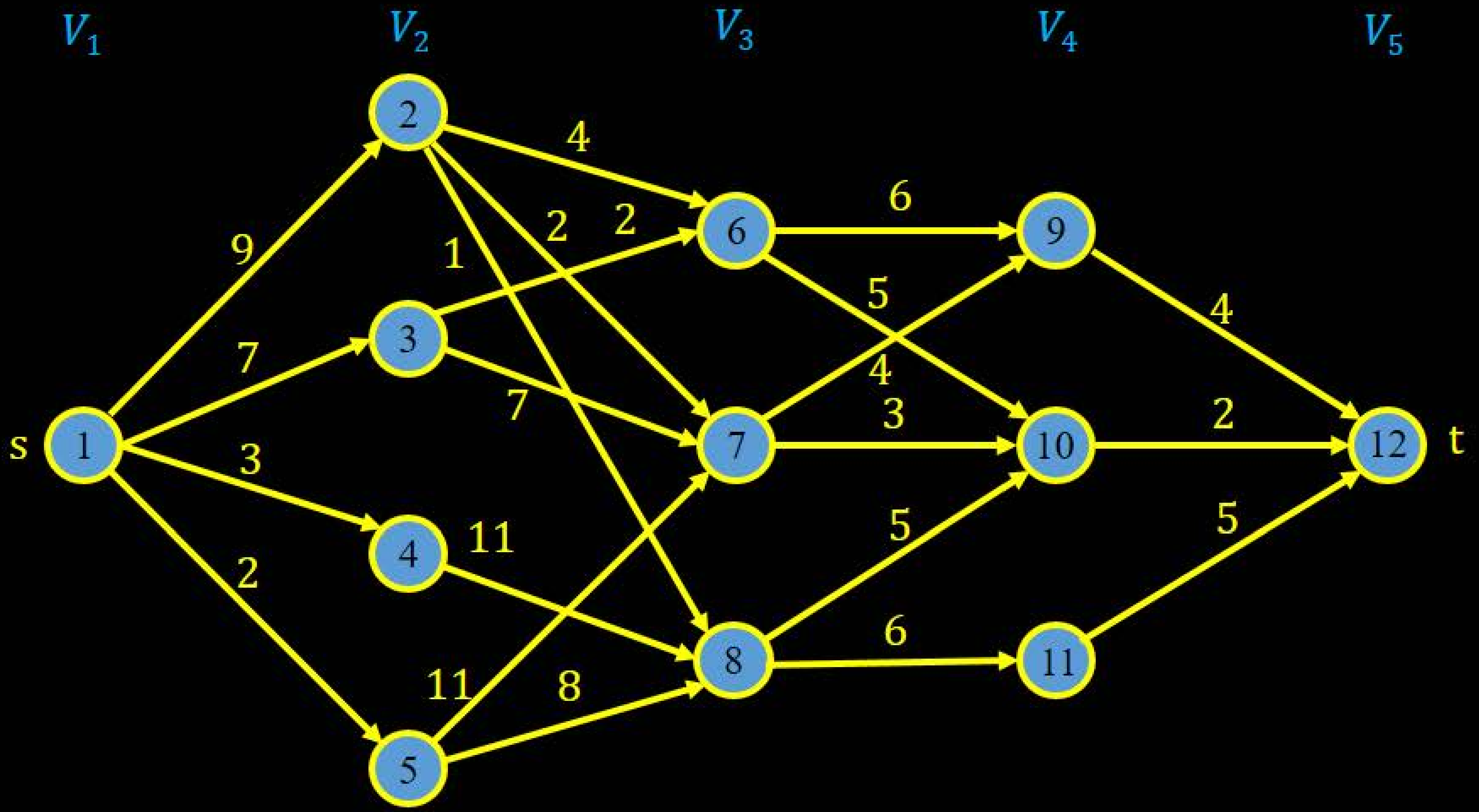
= 9 to Reaching vertex - 12

- Smaller instance of problem I have solved

3 stage of vertex 6 to 12

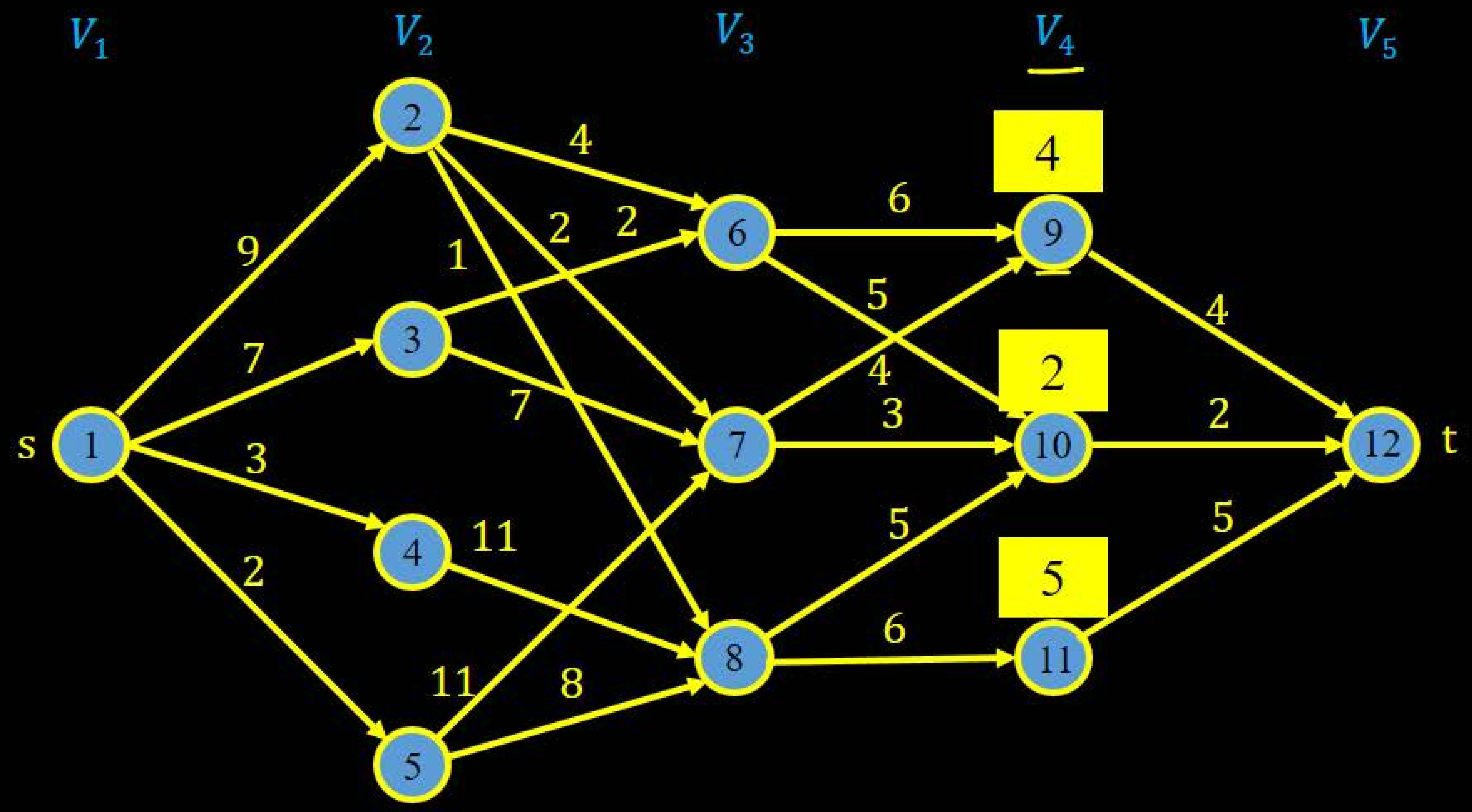
$Cost(3, 6)$ -

$$Cost(3,6) = \min \begin{cases} 6 + cost(4,9) \\ 5 + cost(4,10) \end{cases}$$



tabluhon

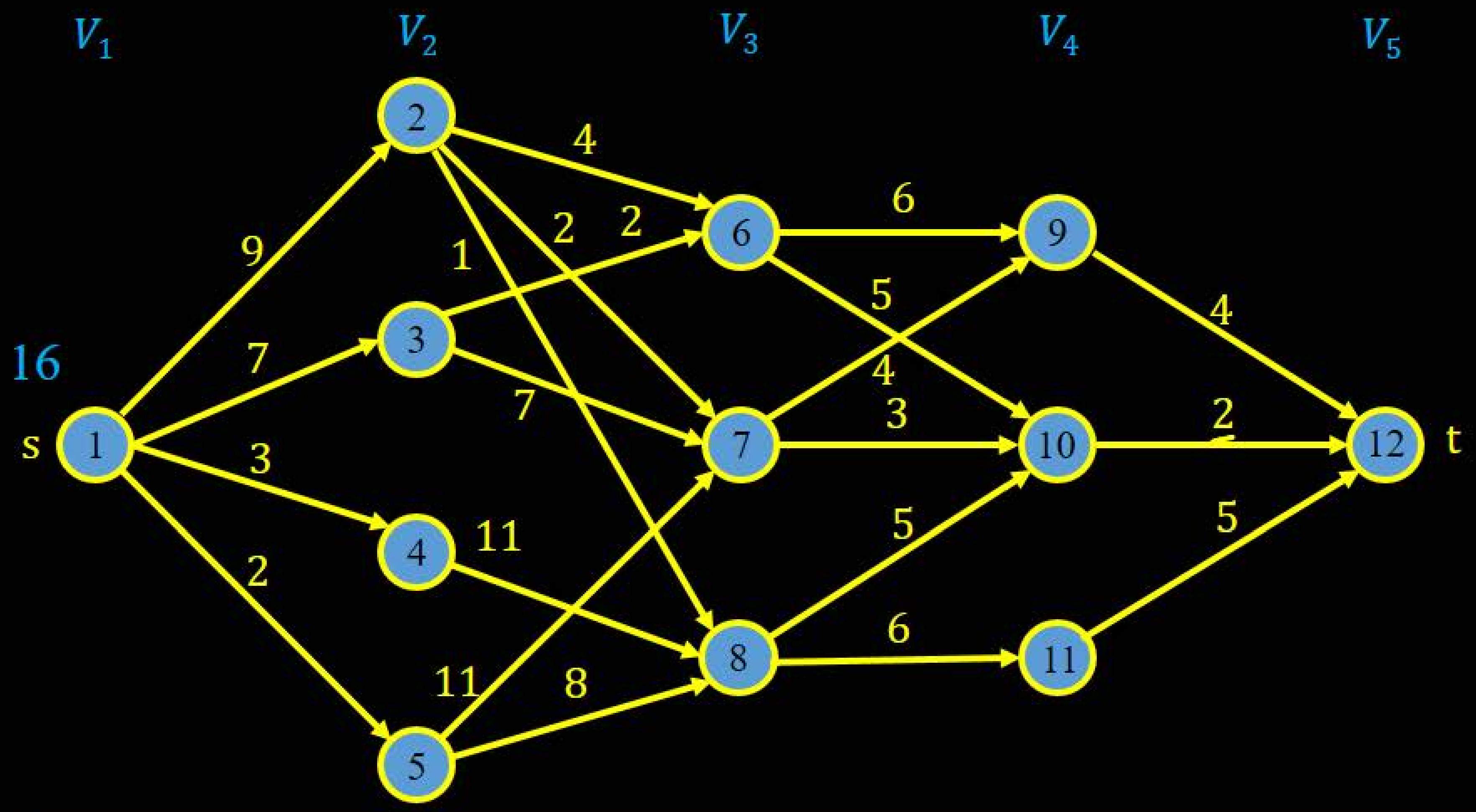
Vertex	1	2	3	4	5	6	7	8	9	10	11	12
Cost												0
d.												12



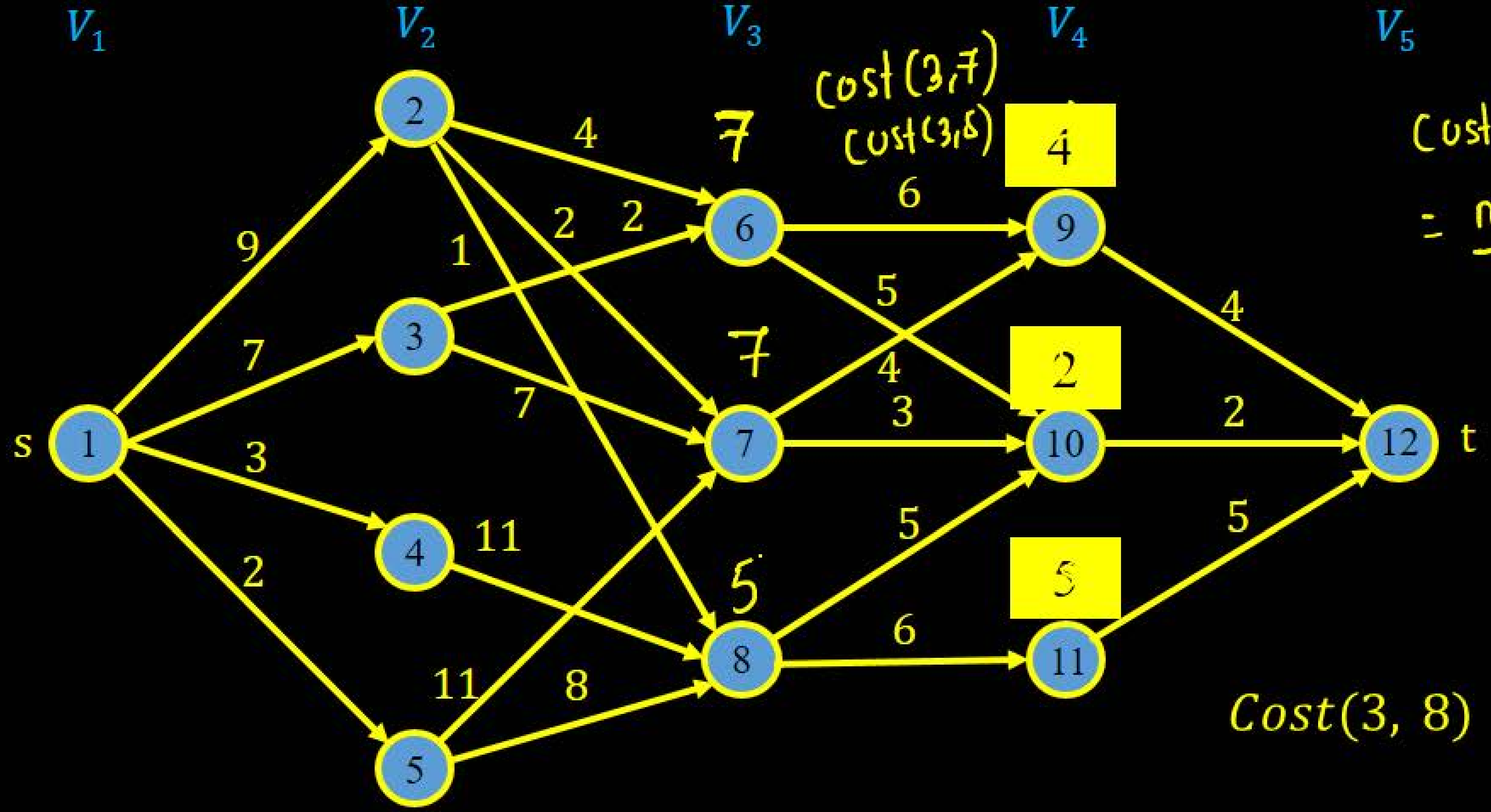
$Cost(4,\ 9)\ = 4$

$Cost(4,\ 10)\ = 2$

$Cost(4,\ 11)\ = 5$



Vertex	1	2	3	4	5	6	7	8	9	10	11	12
Cost									$\boxed{4}$	2	5	0
d									$\boxed{12}$	$\boxed{12}$	$\boxed{12}$	12



$$\text{cost}(3,6) = \min \begin{cases} 6 + \text{cost}(4,9) = 10 \\ 5 + \text{cost}(4,10) = 7 \end{cases}$$

$$Cost(3,\ 8) = \min \begin{cases} 5 + \text{cost}(4,10) \\ 6 + \text{cost}(4,11) \end{cases}$$

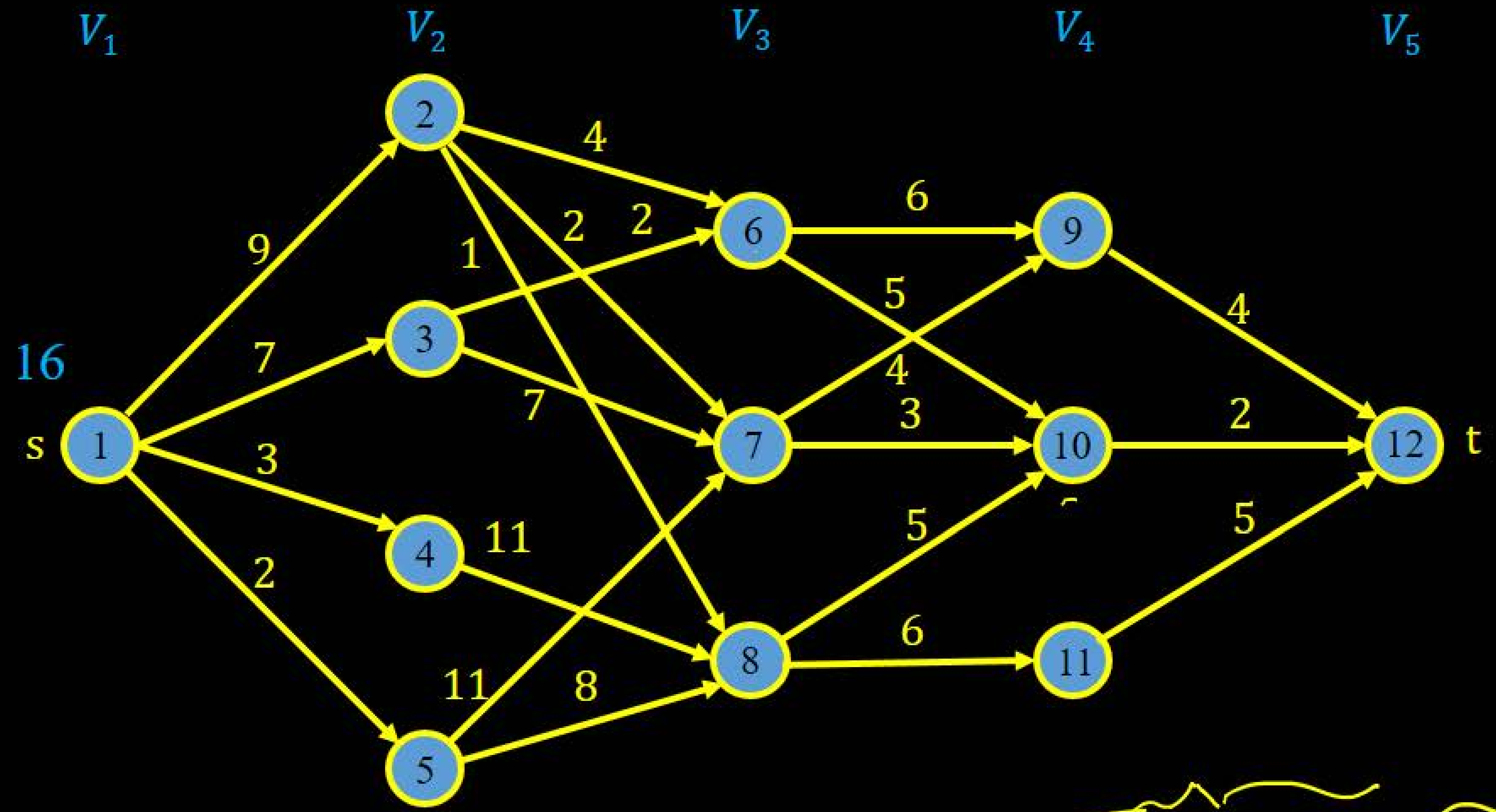
7

$$Cost(3,\ 6) = \min \begin{cases} 6 + \text{cost}(4,9) \\ 5 + \text{cost}(4,10) \end{cases}$$

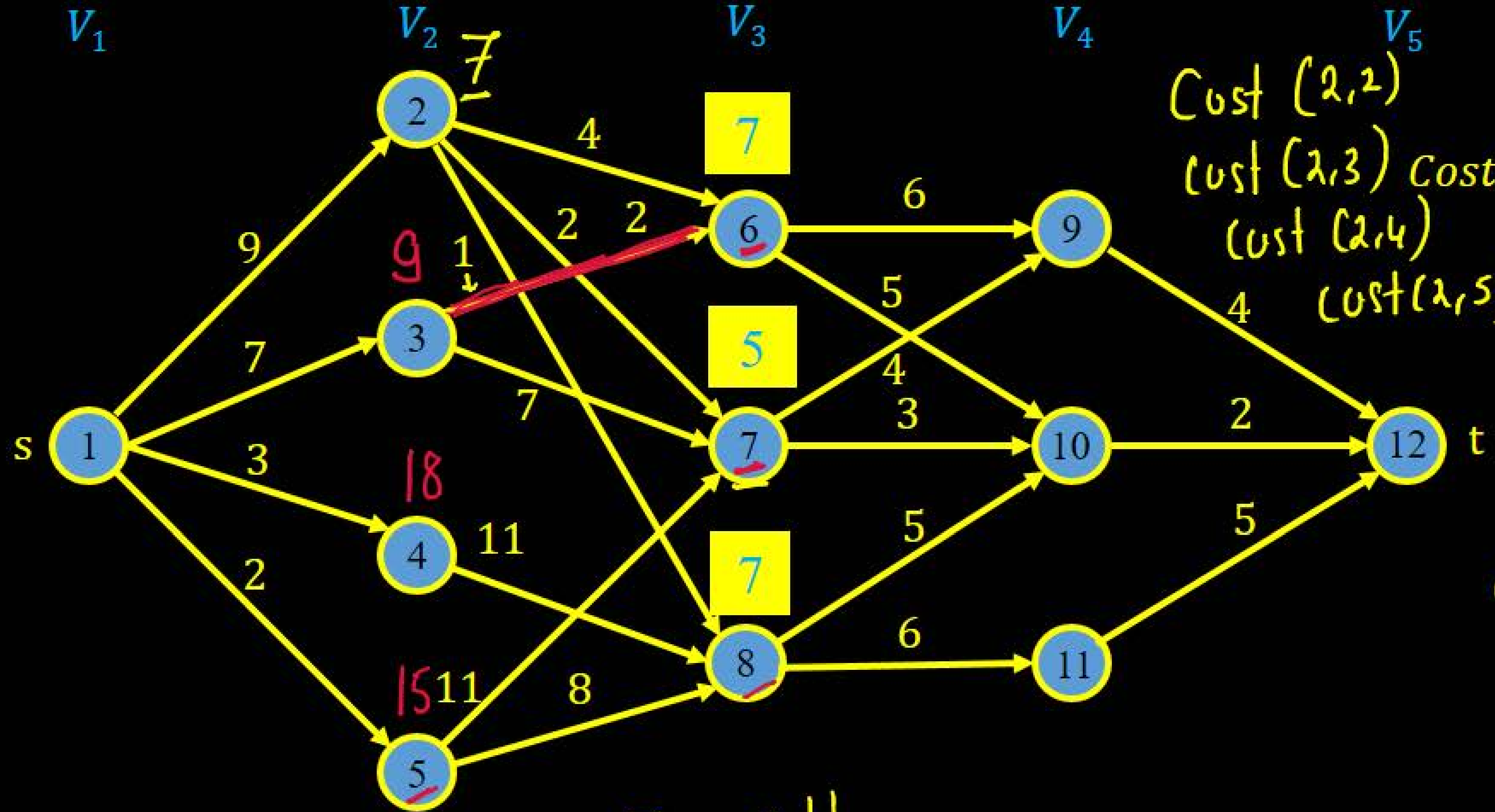
7

$$Cost(3,\ 7) = \min \begin{cases} 4 + \text{cost}(4,9) \\ 3 + \text{cost}(4,10) \end{cases}$$

5



Vertex	1	2	3	4	5	6	7	8	9	10	11	12
Cost						7	5	7	4	2	5	0
d						10	10	10	12	12	12	12



Cost (2,2)
cost (2,3)
cost (2,4)
cost (2,5)

$$Cost(2,3) = \min\begin{cases} 2 + cost(3,6) & 9 \\ 7 + cost(3,7) & 12 \end{cases}$$

9

7 =18

$$Cost(2,4) = \underline{11} + \underline{cost(3,8)}$$

18

7 =11

$$Cost(2,2) = \underline{\min}\begin{cases} 4 + cost(3,6) & 7 \\ 2 + cost(3,7) & 7 \\ 1 + cost(3,8) & 8 \end{cases}$$

5

7

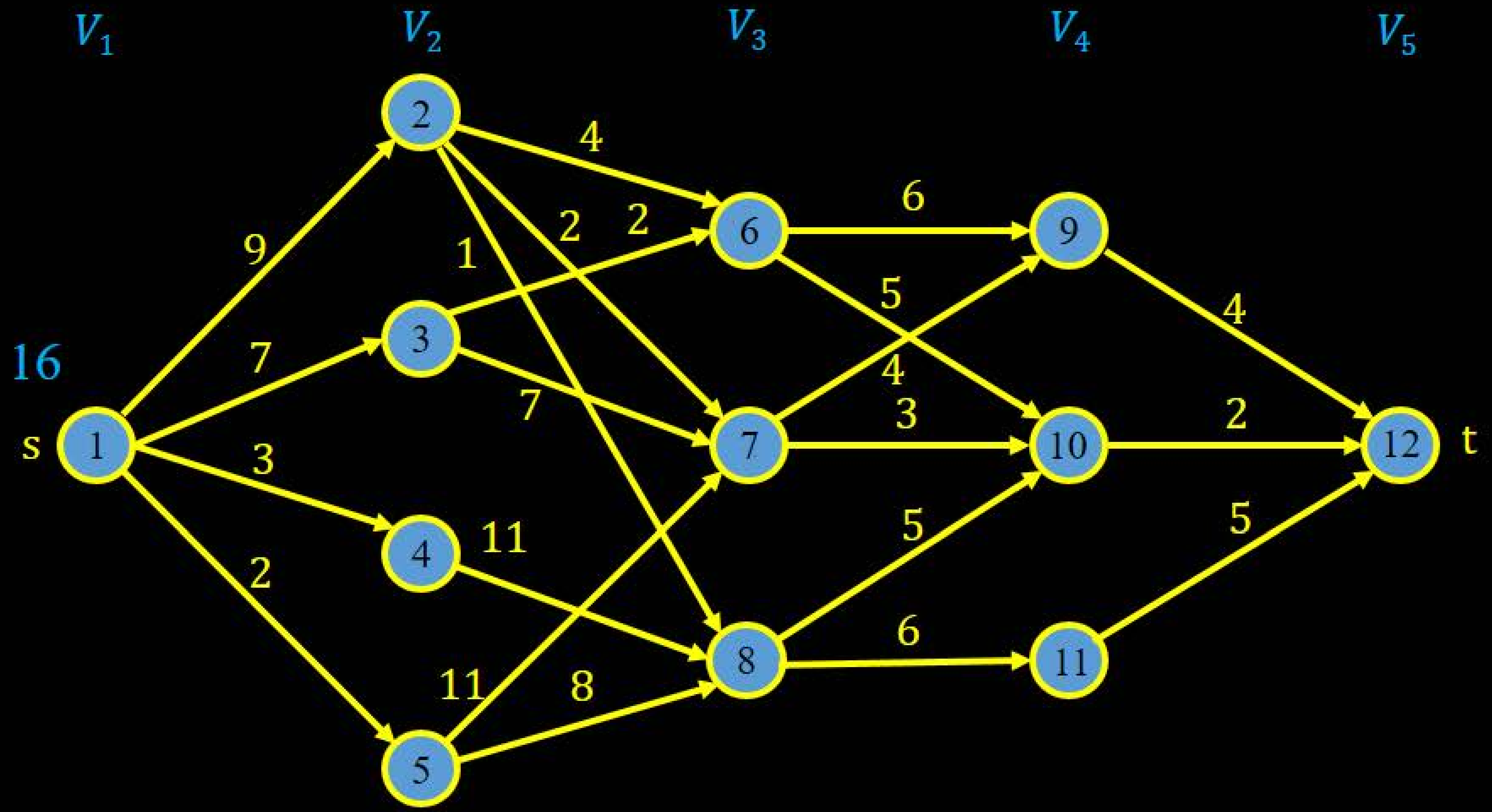
7

16

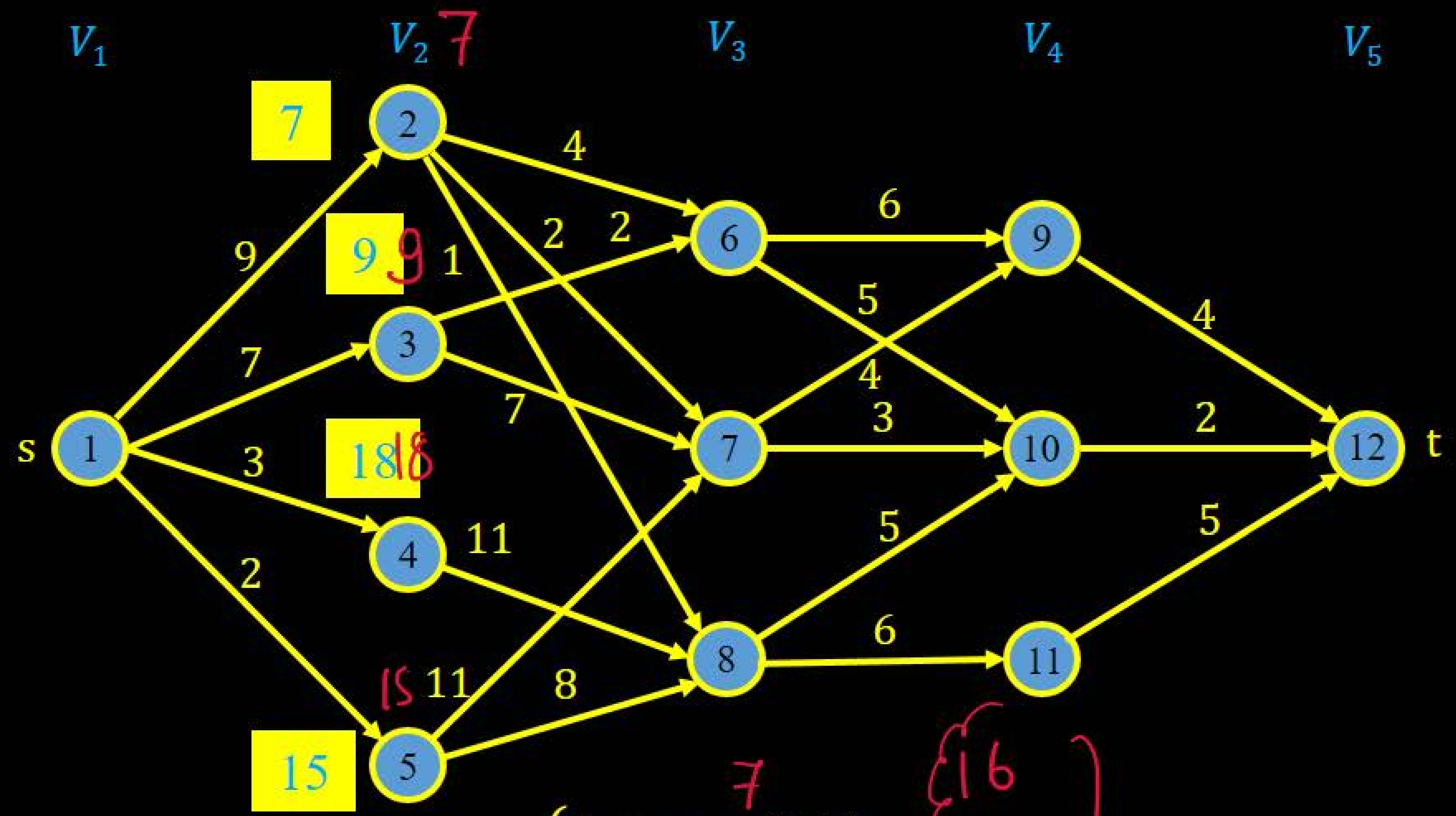
$$Cost(2,5) = \min\begin{cases} 11 + cost(3,7) \\ 8 + cost(3,8) \end{cases}$$

7 =15

15



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



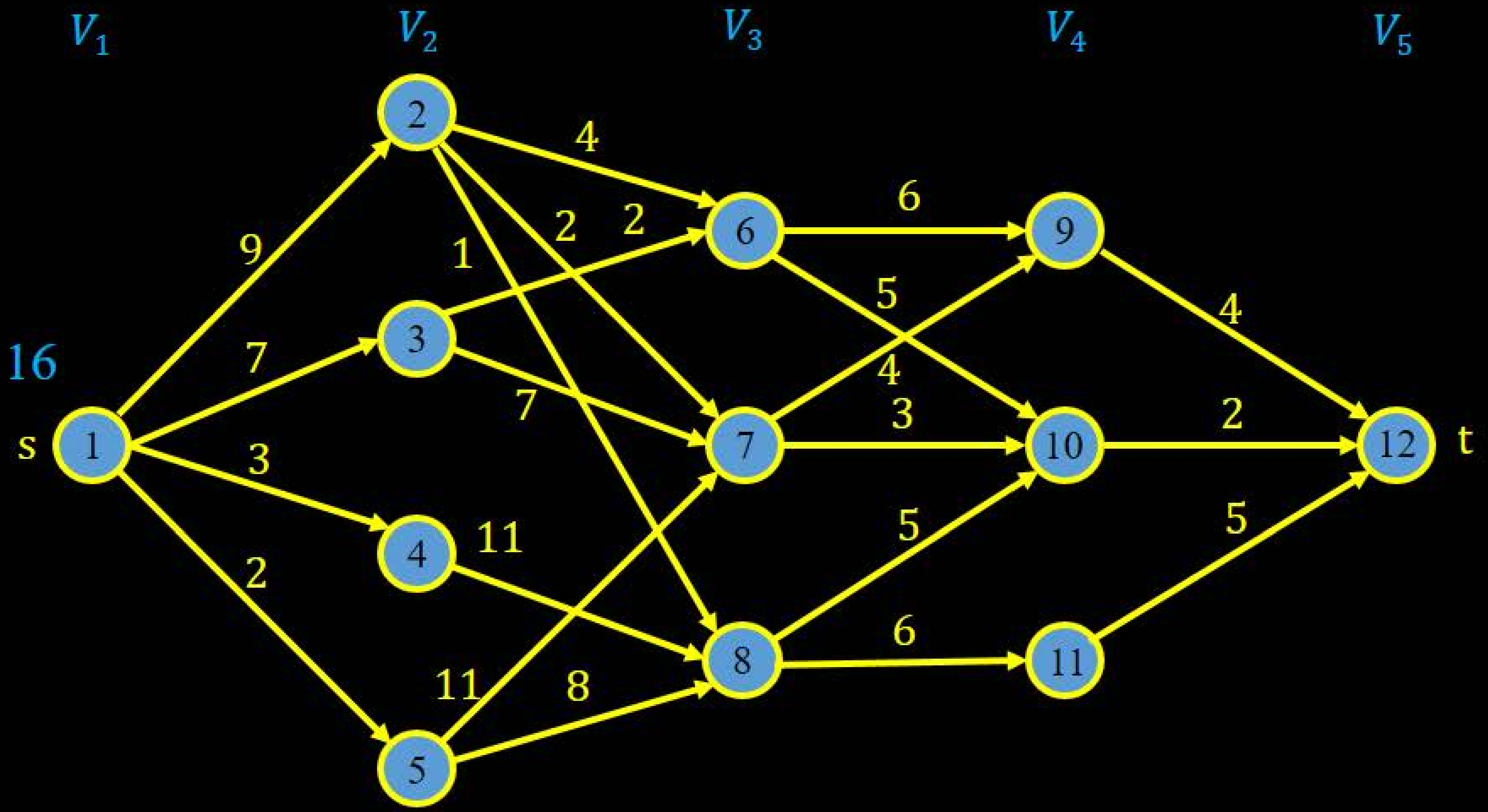
7

$$Cost(1,\ 1)\ = \min\begin{cases} 9 + cost(2,2) \\ 7 + cost(2,3) \\ 3 + cost(2,4) \\ 2 + cost(2,5) \end{cases}$$

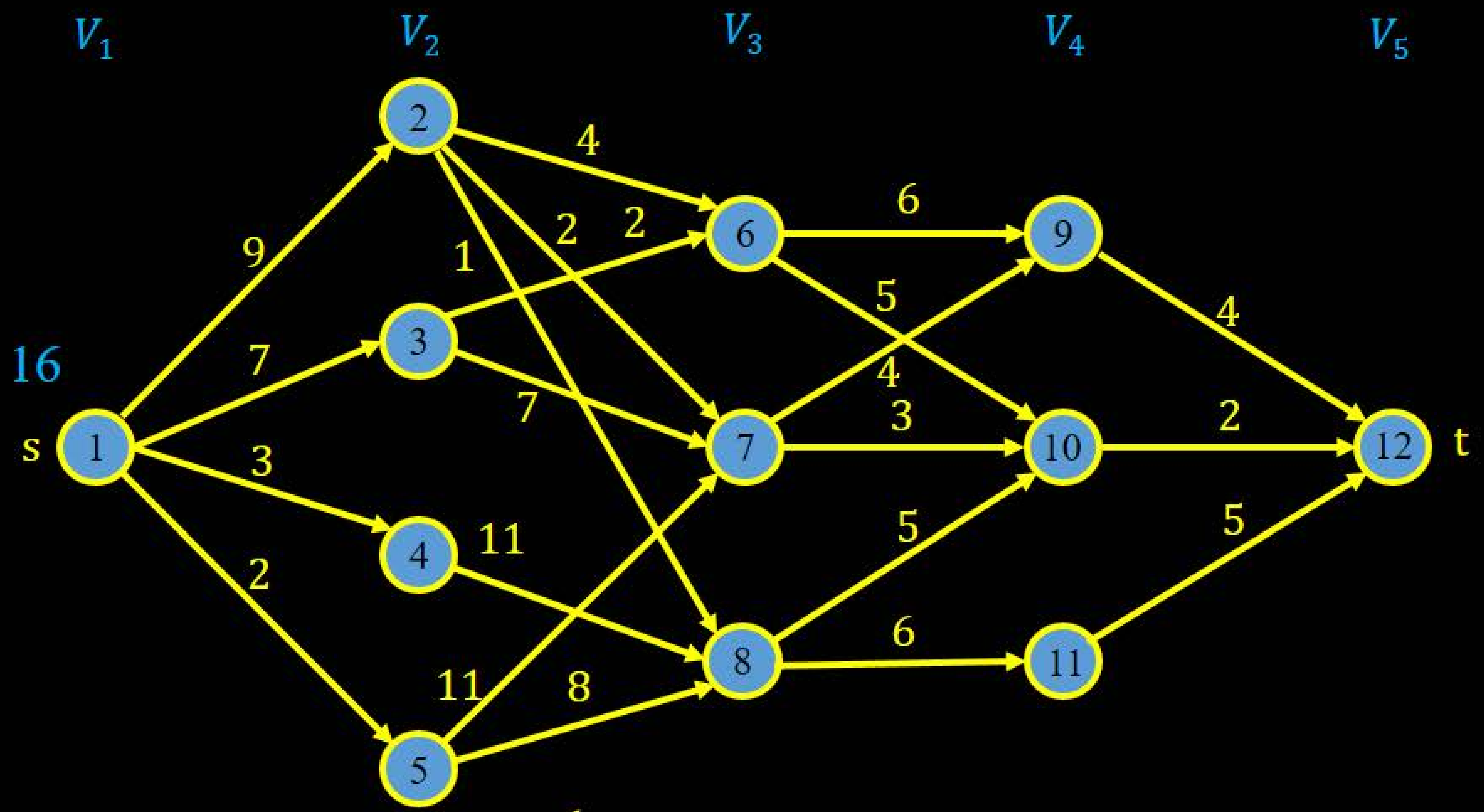
15

9 16
16
18 21
17

16

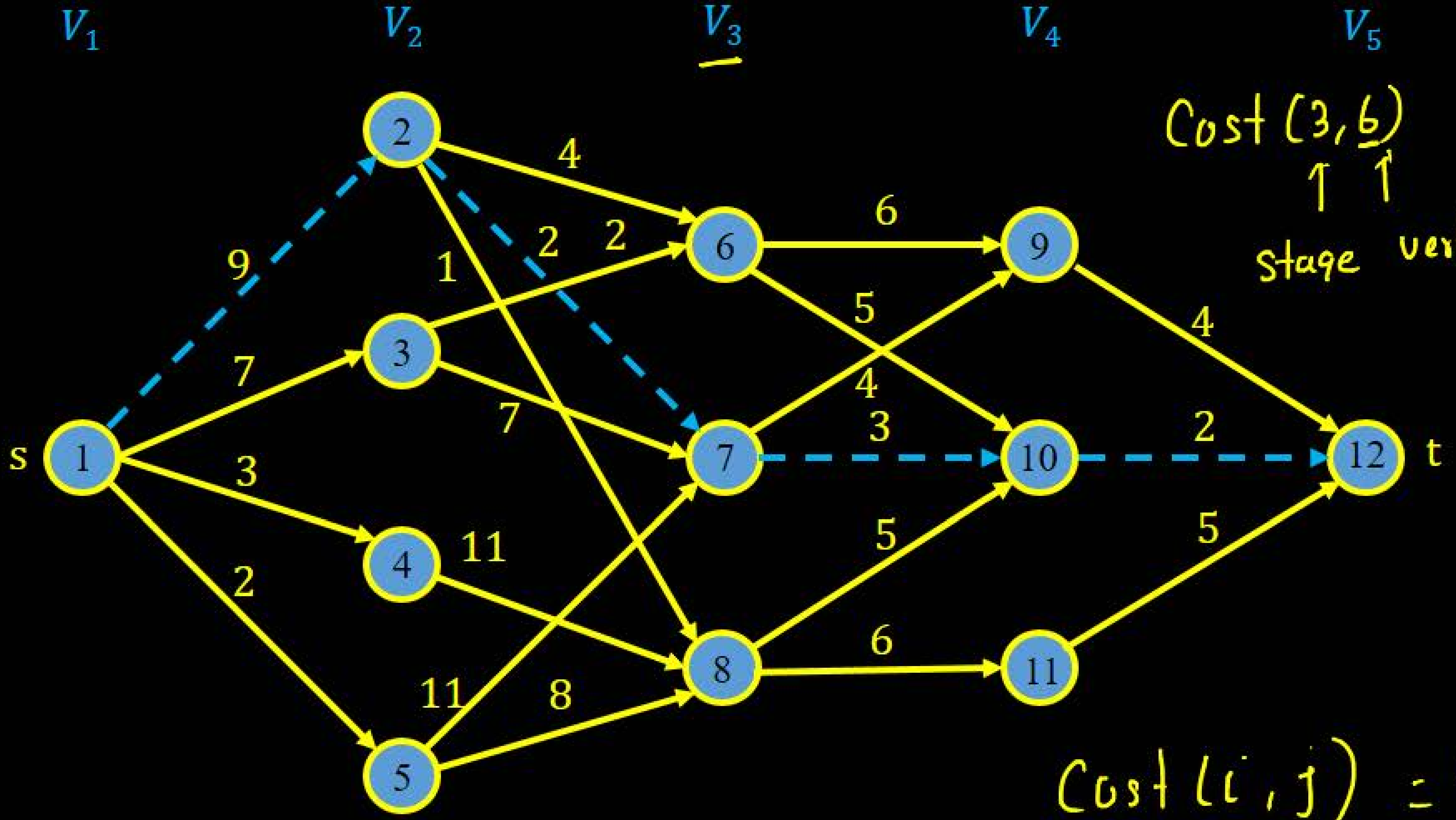


Vertex	1	2	3	4	5	6	7	8	9	10	11	12
Cost	16	7	9	18	15	7	5	7	4	2	5	0
d	2	7	6	8	8	10	10	10	12	12	12	12



$$Cost(1,\ 1)\ = \min \begin{cases} 9 + cost(2,2) \\ 7 + cost(2,3) \\ 3 + cost(2,4) \\ 2 + cost(2,5) \end{cases}$$

16



$$Cost(3, \underline{6}) = \underline{min} \begin{cases} w(\underline{6,9}) + cost(4, 9) \\ \underline{w}(6,10) + cost(4,10) \end{cases}$$

$\uparrow$ stage $\uparrow$ vertex

edge weight

$$Cost(i, \underline{j}) = min \left\{ w(j, l) + \underline{cost(i+1, l)} \right.$$

$j \in \underline{V_i}$

Forward Approach

Dynamic Programming Formulation

Dynamic Programming Formulation

The ith decision involves determining which vertex in V_{i+1}, $1 \leq i \leq k-2$.

Shortest distance from stage 1, node 1 to destination, i.e., 12 is using forward approach

$$Cost(i,\ j)\ =\ \min_{\substack{l \in V_{i+1} \\ (j,l) \in E}} \{c(j,l) + Cost(i+1,l)\}$$

Dynamic Programming Formulation

The ith decision involves determining which vertex in V_{i+1}, $1 \leq i \leq k-2$.

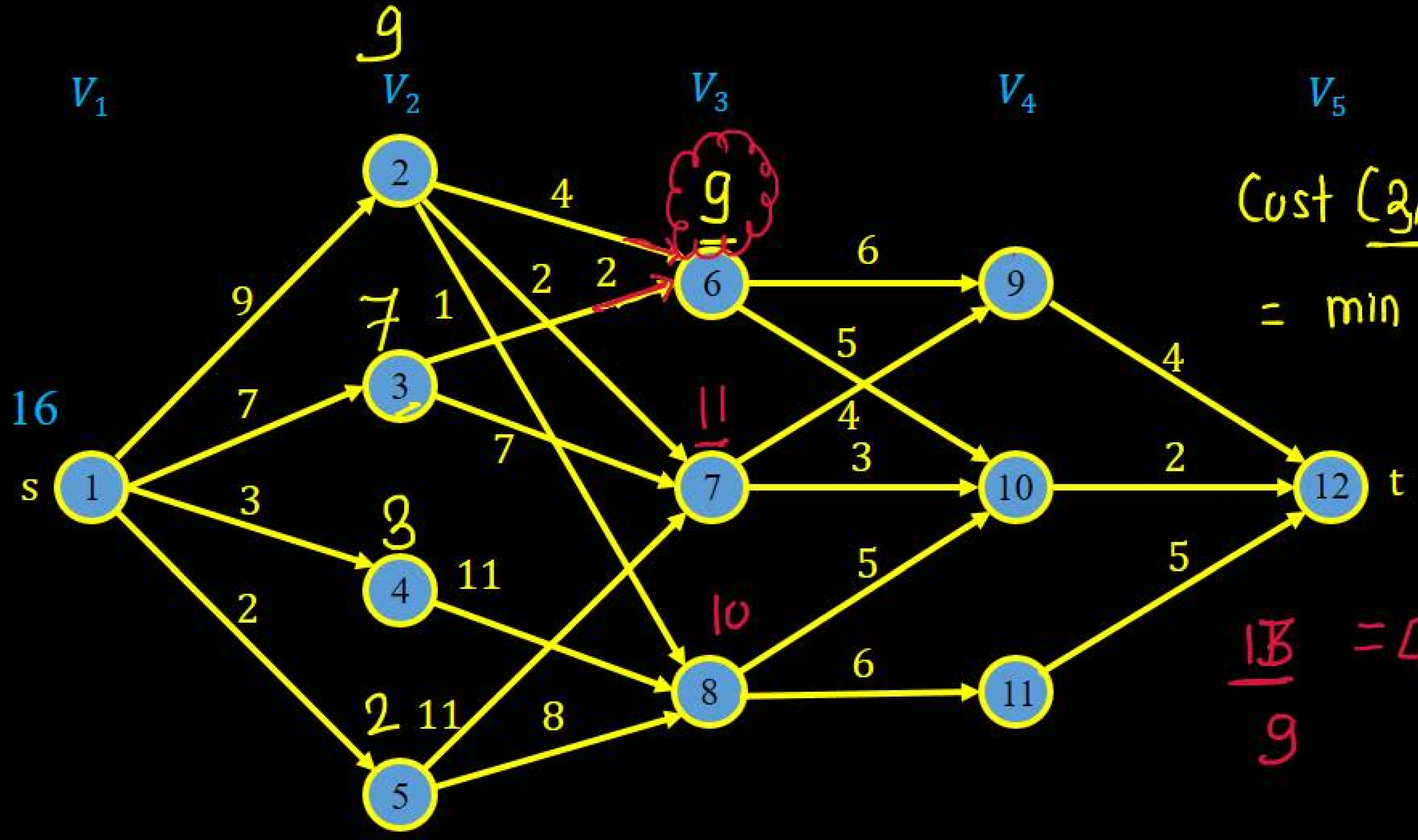
Shortest distance from stage 1, node 1 to destination, i.e., 12 is using forward approach

$$Cost(i, j) = \min_{\substack{l \in V_{i+1} \\ (j,l) \in E}} \{c(j, l) + Cost(i + 1, l)\}$$

Backward Approach

The multistage graph can also be solved by using backward approach

$$Bcost(i,\ j)\ =\ \min_{\substack{l \in V_{i-1} \\ (l,j) \in E}}\ \{Bcost(i-1, l) + c(j, l)\}$$



Cost $(3,6)$ 13

$w(2,6)$

$= \min \begin{cases} 4 + \text{cost}(2,2) \\ \phantom{2+{}} 9 \\ 2 + \text{cost}(2,3) \\ 2 + 7 = 9 \end{cases}$

$\underline{13}$ $= 4 + \text{Cost}(2,3)$ (9)

9 $\quad 2 + \text{Cost}\,\underline{(2,2)}$ 7

$$B\cos t(3,\ 6) = \min\{4 + B\cos t(2,\ 2), 2 + B\cos t(2,3)\}$$

$$= \min\{4+9, 2+7\}$$

$$= \min\{13, 9\}$$