Greedy Method

By

Prof. Shaik Naseera

Department of CSE

JNTUA College of Engg., Kalikiri

Objectives

- Greedy Method
- Applications
  - Optimal Storage on Tapes
  - Job Sequencing with Deadlines
  - Knapsack Problem
  - Single Source Shortest Path
- Previous Gate Questions

Greedy Method

- The Greedy method is a most straight forward design technique which can be applied to a wide variety of problems.
- This algorithm works in steps. In each step it selects the best available options until all options are finished.
- Most of the problems have n inputs and require us to obtain a subset that satisfies some constraints.
- Any subset that satisfies these constraints is called as a ***feasible solution***.
- A feasible solution that either minimizes or maximizes a given objective function is called as ***Optimal Solution***.

Greedy Algorithm

- The Greedy method suggest that one can devise an algorithm that work in stages, considering one input at a time.
- At each stage, a decision is made regarding whether a particular input is an optimal solution or not.
- This is done by considering the inputs in an order determined by some selection procedure.
- If the inclusion of the next input into the partially constructed optimal solution results sub-optimal/infeasible solution, then that input is not added to the partial solution. Otherwise, it is added. The selection procedure itself is based on some optimization measures.

Control Abstraction for ***Greedy Method***

```
procedure GREEDY(A,n)
   //A(1:n) contains the n inputs//
   solution ← φ    //initialize the solution to empty//
   for i ← 1 to n do
      x ← SELECT(A)
      if FEASIBLE(solution,x)
         then solution ← UNION(solution,x)
      endif
   repeat
   return (solution)
end GREEDY
```

Select selects an input from a[] and removes it. The selected input's value is assigned to x.
Feasible is a Boolean-valued function that determines whether x can be included into the solution vector or not.
Union combines x with the solution and updates the objective function.

Types of Greedy Problems

- Subset Paradigm
  - To solve a problem (or possibly find the optimal/best solution), greedy approach generate subset by selecting one or more available choices. Eg. includes **Knapsack problem, job sequencing with deadlines**. In both of the problems greedy create a subset of items or jobs which satisfies all the constraints.
- Ordering Paradigm
  - In this, greedy approach generate some arrangement/order to get the best solution. Eg. includes: **Minimum Spanning tree**

Applications

- Fractional knapsack algorithm
- Optimal Storage on tapes
- Job sequencing with deadline
- Single source shortest path
  - Dijkstra's SSSP algorithm
- Activity Selection Problem
- Minimum Cost Spanning Tree
- ...

Optimal Storage on Tapes

- Optimal Storage on Tapes is one of the application of the Greedy Method.
- The objective is to find the Optimal retrieval time for accessing programs that are stored on tape.

l_{i_k}

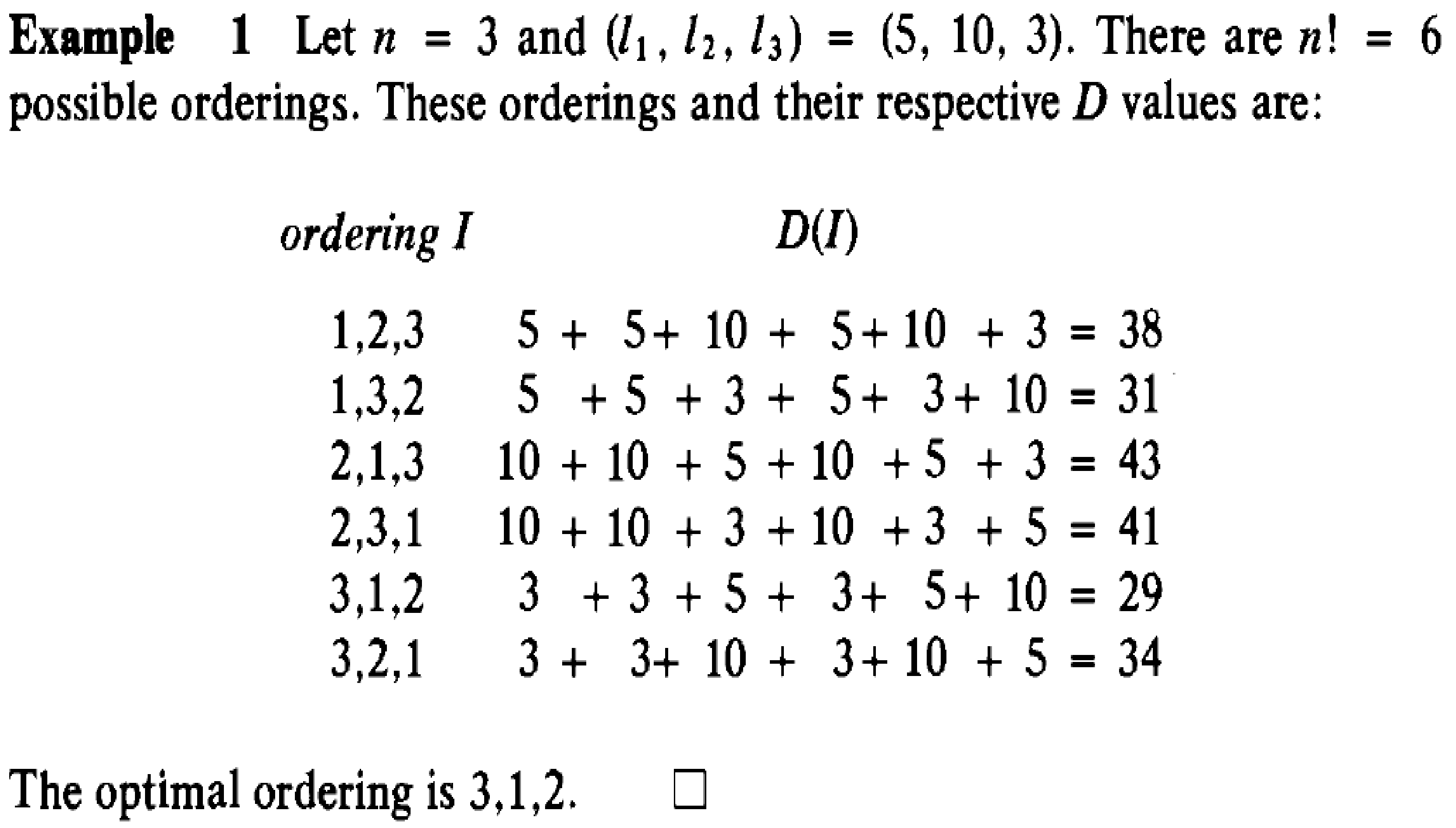
Description

- There are *n programs that are to be stored on a computer tape of length L.*
- *Associated with each program i is a length l;*
- *Clearly, all* programs can be stored on the tape if and only if the sum of the lengths of the programs is at most *L.*
- *We shall assume that whenever a program is to* be retrieved from this tape, the tape is initially positioned at the front.
- Hence' if the programs are stored in the order $I = i_1, i_2, i_3 \ldots i_n$ *the time t_j needed to retrieve program* i_j *is proportional to* l_{ik} .
- *If all programs* are retrieved equally often then the expected or *mean retrieval time (MRT) is*

$$t_j = \sum_{1 \le k \le j} l_{i_k}$$

$$MRT = 1/n \sum_{1 \le j \le n} t_j$$

Example



Example 1 Let $n = 3$ and $(l_1, l_2, l_3) = (5, 10, 3)$. There are $n! = 6$ possible orderings. These orderings and their respective D values are:

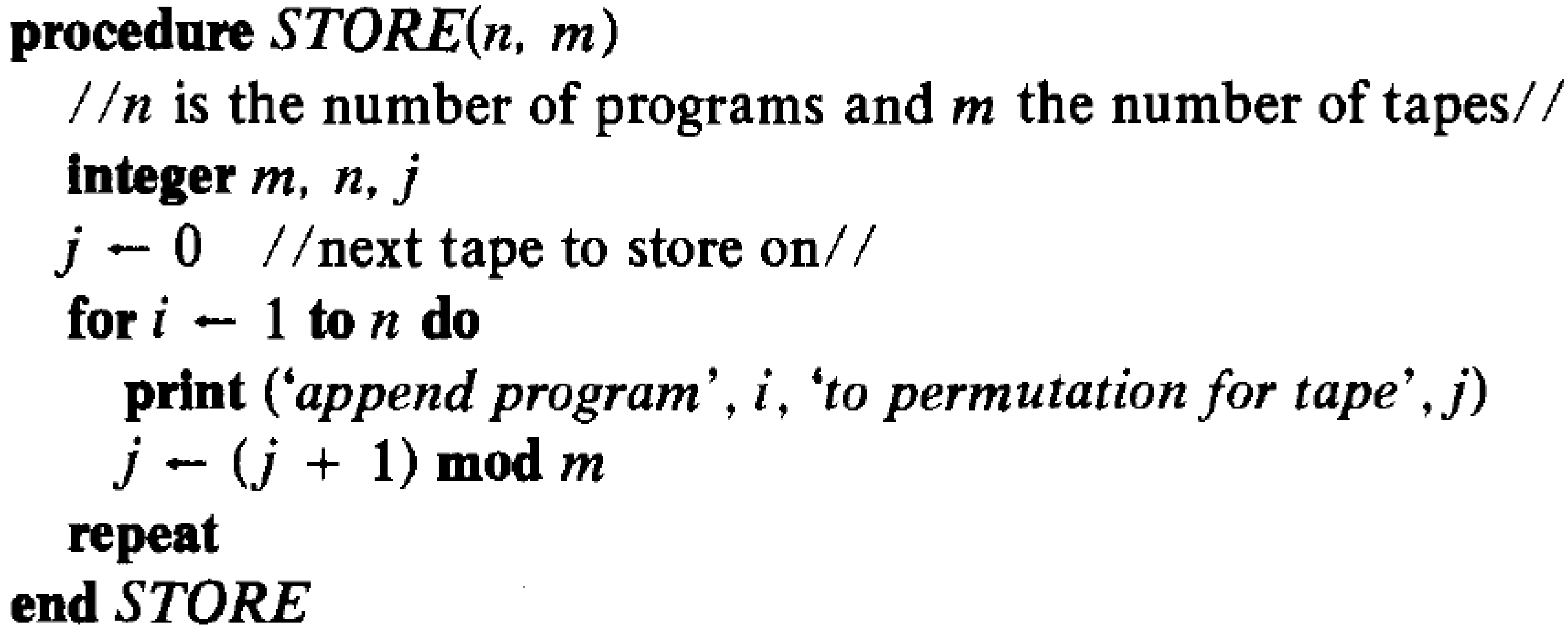
ordering I	$D(I)$
1,2,3	5 + 5 + 10 + 5 + 10 + 3 = 38
1,3,2	5 + 5 + 3 + 5 + 3 + 10 = 31
2,1,3	10 + 10 + 5 + 10 + 5 + 3 = 43
2,3,1	10 + 10 + 3 + 10 + 3 + 5 = 41
3,1,2	3 + 3 + 5 + 3 + 5 + 10 = 29
3,2,1	3 + 3 + 10 + 3 + 10 + 5 = 34

The optimal ordering is 3,1,2. ☐

Method

- The greedy method simply requires us to store the programs in non-decreasing order of their lengths.
- This ordering (sorting) can be carried out in *O(n log n) time using an efficient sorting algorithm*

Algorithm for multiple tapes



```
procedure STORE(n, m)
    //n is the number of programs and m the number of tapes//
    integer m, n, j
    j ← 0    //next tape to store on//
    for i ← 1 to n do
        print ('append program', i, 'to permutation for tape', j)
        j ← (j + 1) mod m
    repeat
end STORE
```

Note: The programs are assumed to be in increasing order of their lengths

Knapsack Problem

- Let, we are given n objects and a Knapsack or Bag.
- Object i has weight W_i and the Knapsack has a capacity M.
- if a fraction X_i of object i is placed into Knapsack, then a profit of P_iX_i is earned.
- The objective is to obtain a filling of Knapsack that maximizes the total profit earned.

- Maximize $\sum_{i \le i \le n} PiXi$ (A)
- Subject to $\sum_{i \le i \le n} WiXi \le M$ (B)
-
- And $0 \le X_i \le 1$, $1 \le i \le n$ (C)
- The profit and weights are the positive numbers.
- Here, A feasible solution is any set (X1, X2, …, Xn) satisfying above rules (B) and (C).
- And an optimal solution is feasible solution for which rule (A) is maximized.

Here, N=3, M=20, (P1, P2, P3)=(25, 24, 15) and (W1, W2, W3)=(18, 15, 10)

Different feasible solutions are:

(X1, X2, X3)	$\sum WiXi$	$\sum_{1 \le i \le n} PiXi$
1. (1/2, 1/3, ¼)	16.5	24.25
2. (1, 2/15, 0)	20	28.2
3. (0, 2/3, 1)	20	31
4. (0, 1, 1/2)	20	31.5
5. (1/2, 2/3, 1/ 10)	20	30
6. (1, 0, 2/10)	20	28

- Of these Six feasible solutions, solution 4 yields the maximum profit. Therefore solution 4 is optimal for the given problem instance.
- *Consideration* 1 - In case the sum of all the weights is ≤ M, then Xi=1, 1 ≤ i ≤n is an optimal solution.
- *Consideration* 2 - All optimal solutions will fill the knapsack exactly.

The knapsack algorithm

- The greedy algorithm:
  Step 1: Sort p_i/w_i into nonincreasing order.
  Step 2: Put the objects into the knapsack according
  to the sorted sequence as possible as we can.
- e. g.

p[i]/w[i]≥p[i+1]≥w[i+1]

$n = 3$, $M = 20$, $(p_1, p_2, p_3) = (25, 24, 15)$
$(w_1, w_2, w_3) = (18, 15, 10)$
Sol: $p_1/w_1 = 25/18 = 1.39$
$p_2/w_2 = 24/15 = 1.6$
$p_3/w_3 = 15/10 = 1.5$
Optimal solution: $x_1 = 0$, $x_2 = 1$, $x_3 = 1/2$
total profit = 24 + 7.5 = 31.5

Algorithm

- Algorithm GreedyKnapsack(m,n)

```
//order the n objects such that p[i]/w[i]≥p[i+1]≥w[i+1]
{
for i:=1 to n do x[i]:=0.0;
    U:=m;
for i:=1 to n do
 {
     if(w[i] > U) then break;
    x[i]:=1.0; U:=U-w[i];
}
if( i ≤ n) then x[i]:=U/w[i];
}
```

Example problem

- N=7
- M=15(15-1=14-2=12-4=8-5=3-1=2)
- (p1,p2…p7)=(10,5,15,7,6,18,3)
- (w1,w2..w7)=(2,3,5,7,1,4,1)
- The solution vector is (1,2/3,1,0,1,1,1)
- P1/w1=5 p2/w2=1.66 p3/w3=3 p4/w4=1 p5/w5=6 p6/w6=4.5 p7/w7=3
- x5,x1,x6,x3,x7,x2,x4
- Weight is 1+2+4+5+1+2/3*3+0=13+2/3*3=15
- Profit is 6+10+18+15+3+2/3*5+0=55.34

(1,2/3,1,0,1,1,1)

Job Sequencing with Deadlines

- We are given a set of n jobs.
- Di is a deadline given to complete ith job for profit Pi where $Pi>0$ & $Di \geq 0$.
- For any job i profit Pi is earned iff the job is completed within its deadline.
- To complete a job, one has to process the job on a machine for one unit of time.
- Only one machine is available for processing jobs.
- ***A feasible solution for this problem is a subset J of jobs such that each job in this subset can be completed by its deadline.***
- The value of a feasible solution J is the sum of the profits of the job in J.
- An optimal solution is a feasible solution with maximum value.

- **Example** –Suppose on a single machine four jobs with profit values (100, 10, 15 and 27) and their respective deadline unit values (2, 1, 2, 1) are given. Calculate the different feasible solutions to complete the jobs with optimal solution.
- **Solution:**
- Number of Jobs (n)= 4
- Profit values of four jobs (P1, P2, P3, P4)=(100, 10, 15, 27)
- Process deadlines for respective jobs are

  (D1, D2, D3, D4)=(2, 1, 2, 1)

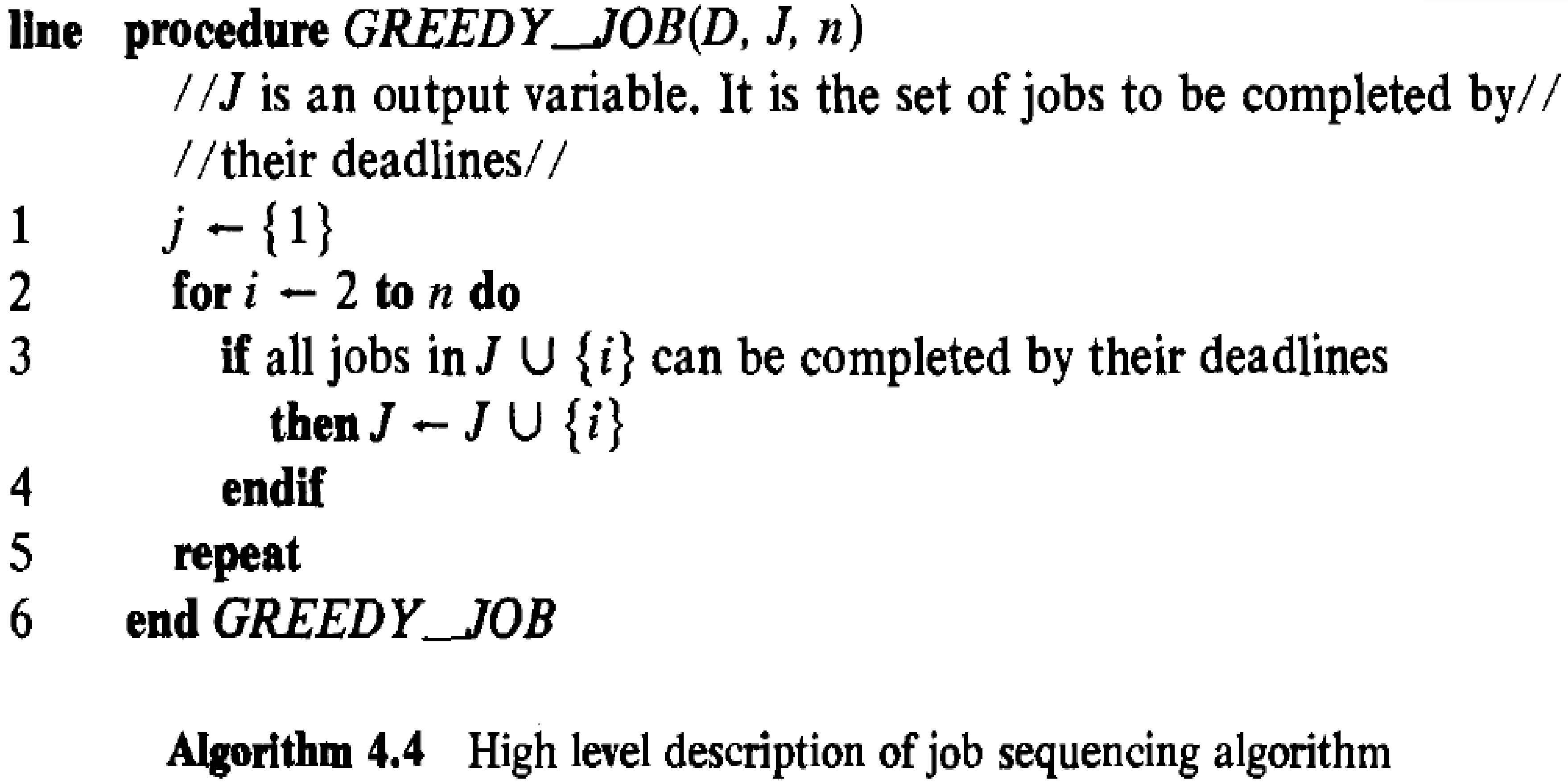
(P1, P2, P3, P4)=(100, 10, 15, 27)   (D1, D2, D3, D4) =(2, 1, 2, 1)

The feasible solutions and their values are –

Feasible sol. subset	Processing Sequence	Values
(1, 2)	(2, 1)	110
(1, 3)	(1, 3 or 3, 1)	115
(1, 4)	(4, 1)	127
(3, 2)	(2, 3)	25
(3, 4)	(4, 3)	42
(1)	(1)	100
(2)	(2)	10
(3)	(3)	15
(4)	(4)	27

Here, Solution 3 is optimal solution. In this solution only job 1 and 4 are processed and the profit value is 127. These jobs must be processed in the order Job 4 followed by job 1. Thus the processing of job 4 begins at time zero and that of job 1 is completed at time 2.

Algorithm



```
line   procedure $GREEDY\_JOB(D, J, n)$
          //$J$ is an output variable. It is the set of jobs to be completed by//
          //their deadlines//
1         $j \leftarrow \{1\}$
2         for $i \leftarrow 2$ to $n$ do
3            if all jobs in $J \cup \{i\}$ can be completed by their deadlines
                then $J \leftarrow J \cup \{i\}$
4            endif
5         repeat
6      end $GREEDY\_JOB$
```

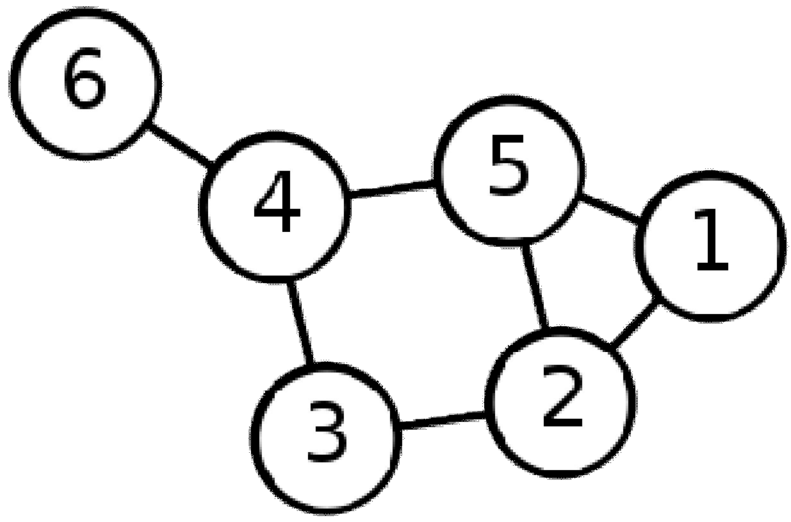
Algorithm 4.4 High level description of job sequencing algorithm

- The algorithm constructs an optimal set J of jobs that can be processed by their deadlines.

Single-Source Shortest Path
(Dijkstra's algorithm)

Single-Source Shortest Path Problem

Single-Source Shortest Path Problem - The problem of finding shortest paths from a source vertex *v* to all other vertices in the graph.



Dijkstra's algorithm

Dijkstra's algorithm - is a solution to the single-source shortest path problem in graph theory.

Works on both directed and undirected graphs. However, all edges must have nonnegative weights.

Approach: Greedy

Input: Weighted graph G={E,V} and source vertex *v*∈V, such that all edge weights are nonnegative

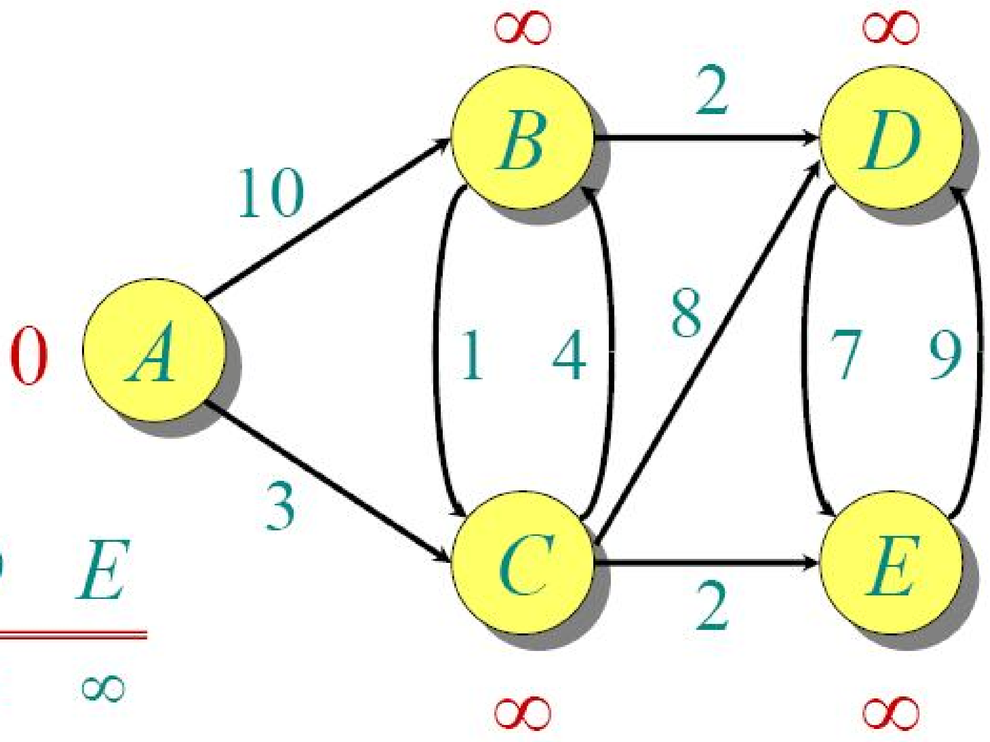
Output: Lengths of shortest paths (or the shortest paths themselves) from a given source vertex *v*∈V to all other vertices

Dijkstra's algorithm - Pseudocode

```
dist[s] ←0                                  (distance to source vertex is zero)
for all v ∈ V–{s}
      do dist[v] ←∞                         (set all other distances to infinity)
S←∅                                         (S, the set of visited vertices is initially empty)
Q←V                                      (Q, the queue initially contains all vertices)
while Q ≠∅                                  (while the queue is not empty)
do   u ← mindistance(Q,dist)                (select the element of Q with the min.
distance)
     S←S∪{u}
     Q=Q-{u}                         (add u to list of visited vertices)
     for all v ∈ neighbors[u]
          do if   dist[v] > dist[u] + w(u, v)                  (if new shortest path found)
                then      d[v] ←d[u] + w(u, v)      (set new value of shortest path)
                      (if desired, add traceback code)
return dist
```

Dijkstra Animated Example

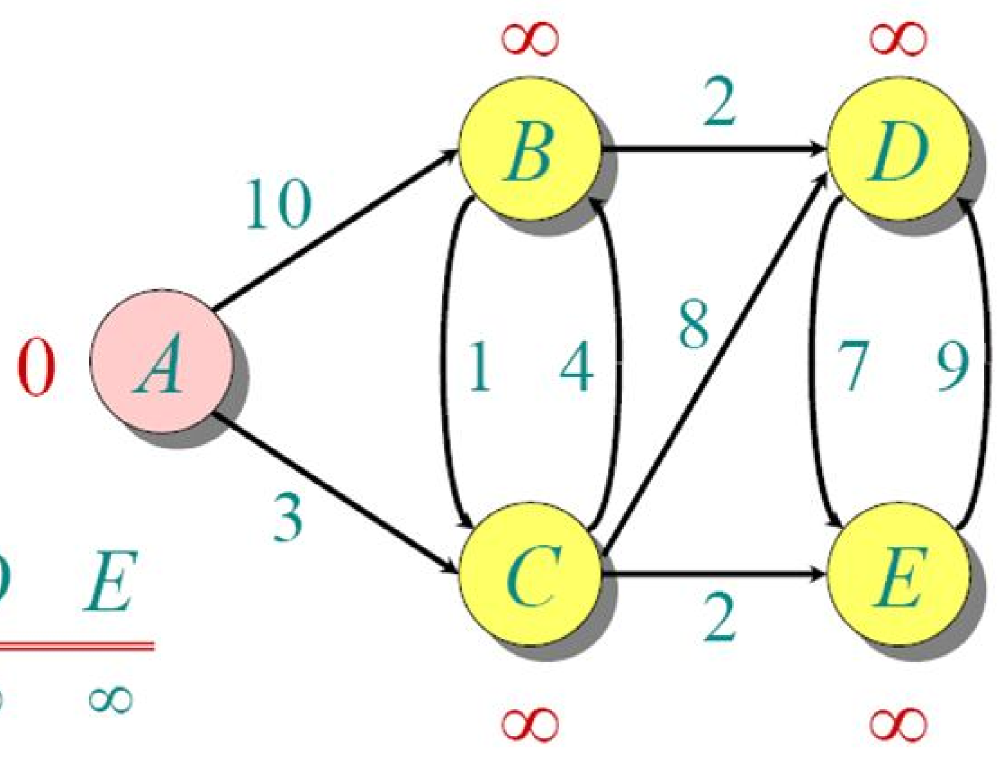
Initialize:



Q:	A	B	C	D	E
	0	∞	∞	∞	∞

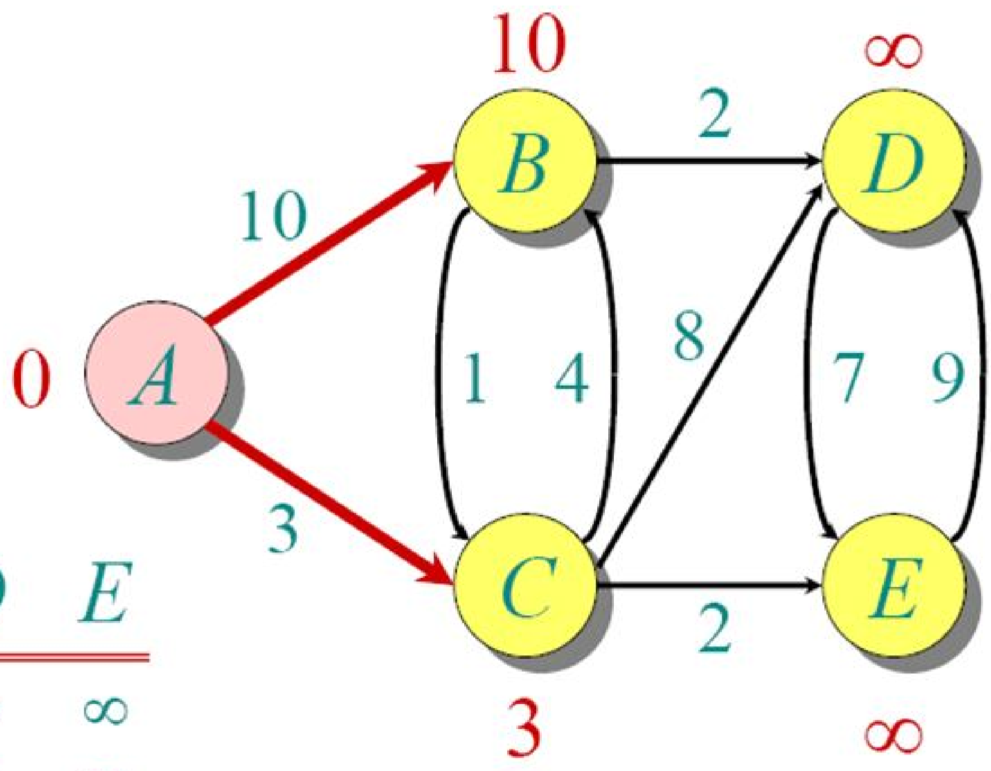
S: {}

Dijkstra Animated Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞

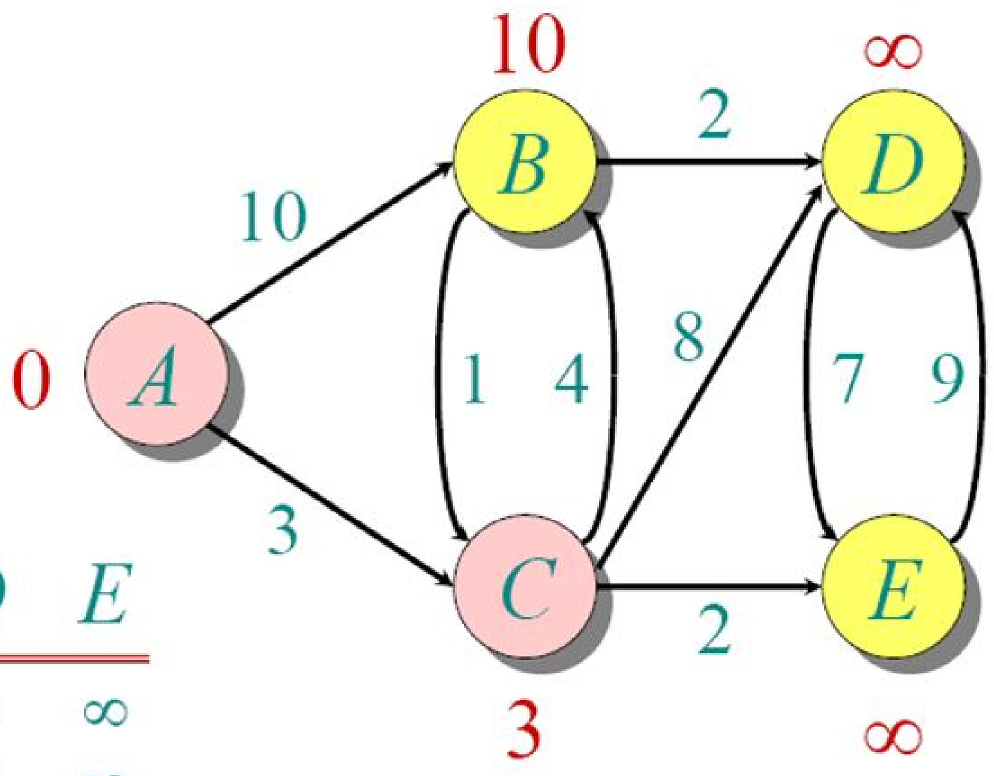
Dijkstra Animated Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A }

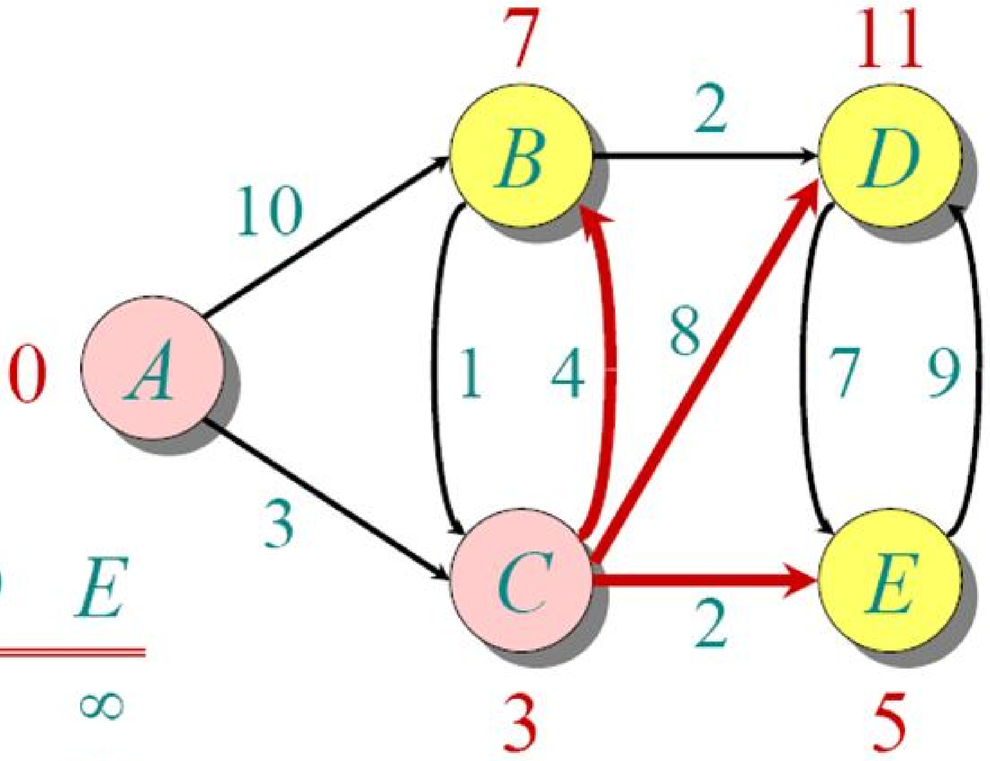
Dijkstra Animated Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A, C }

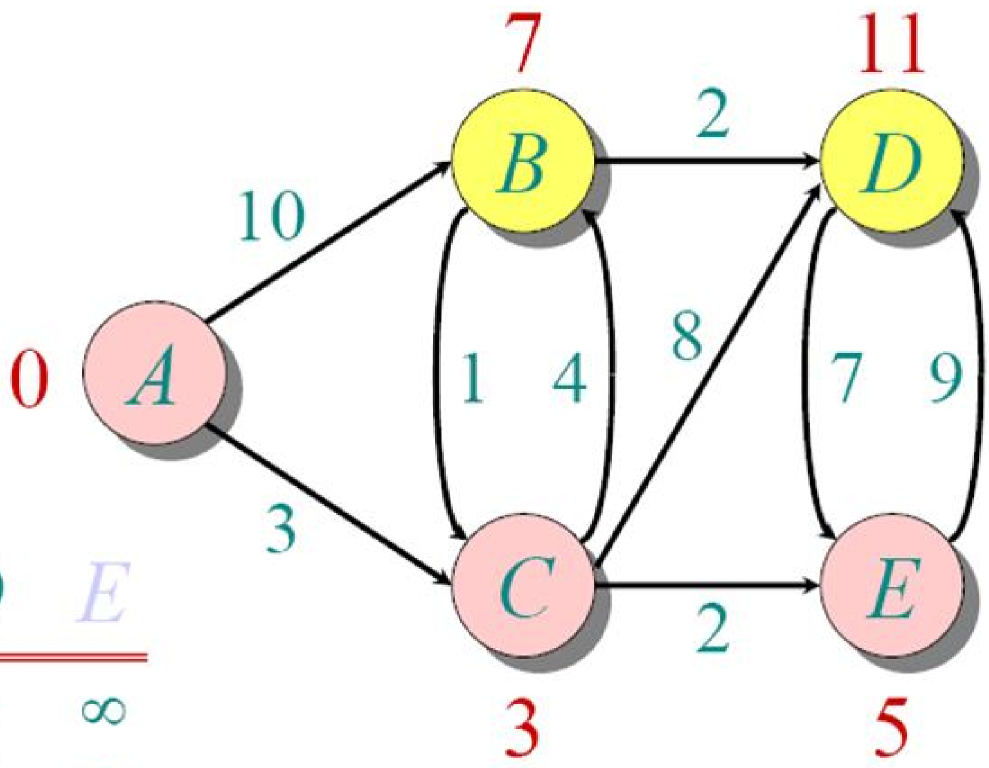
Dijkstra Animated Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { A, C }

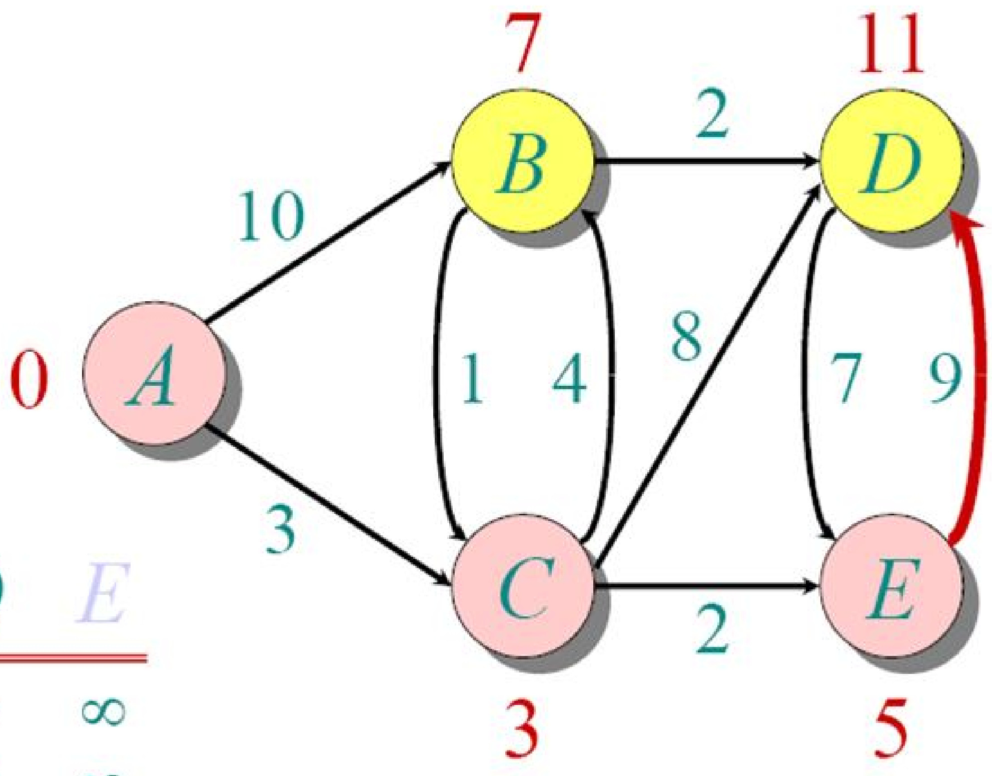
Dijkstra Animated Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { *A*, *C*, *E* }

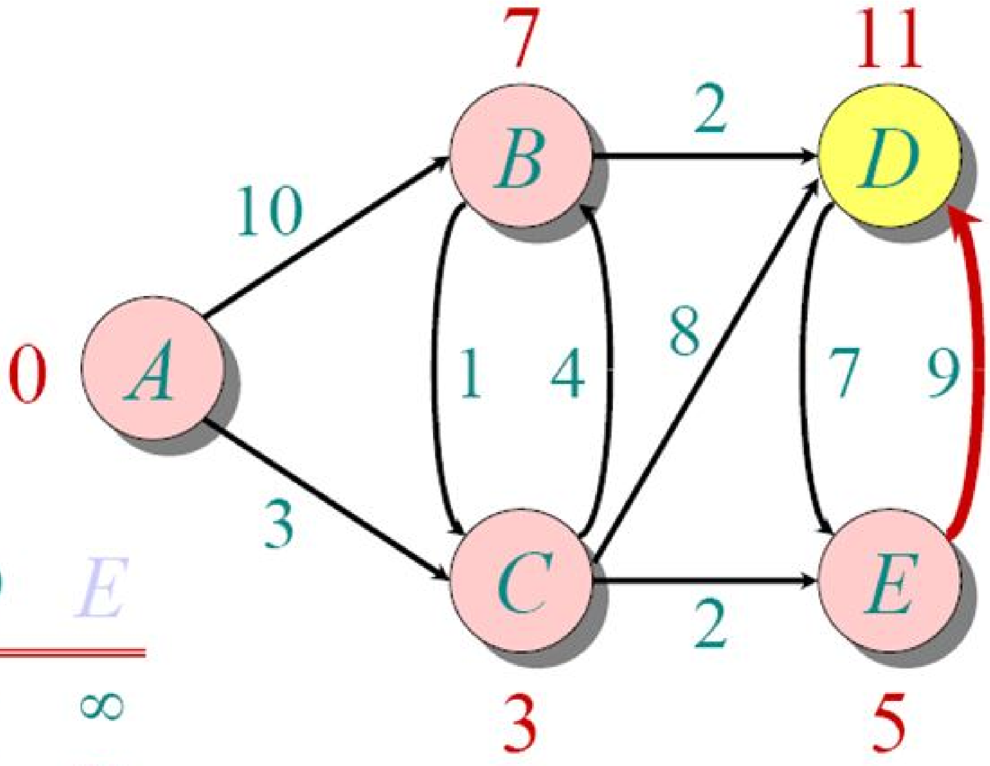
Dijkstra Animated Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { A, C, E }

Dijkstra Animated Example

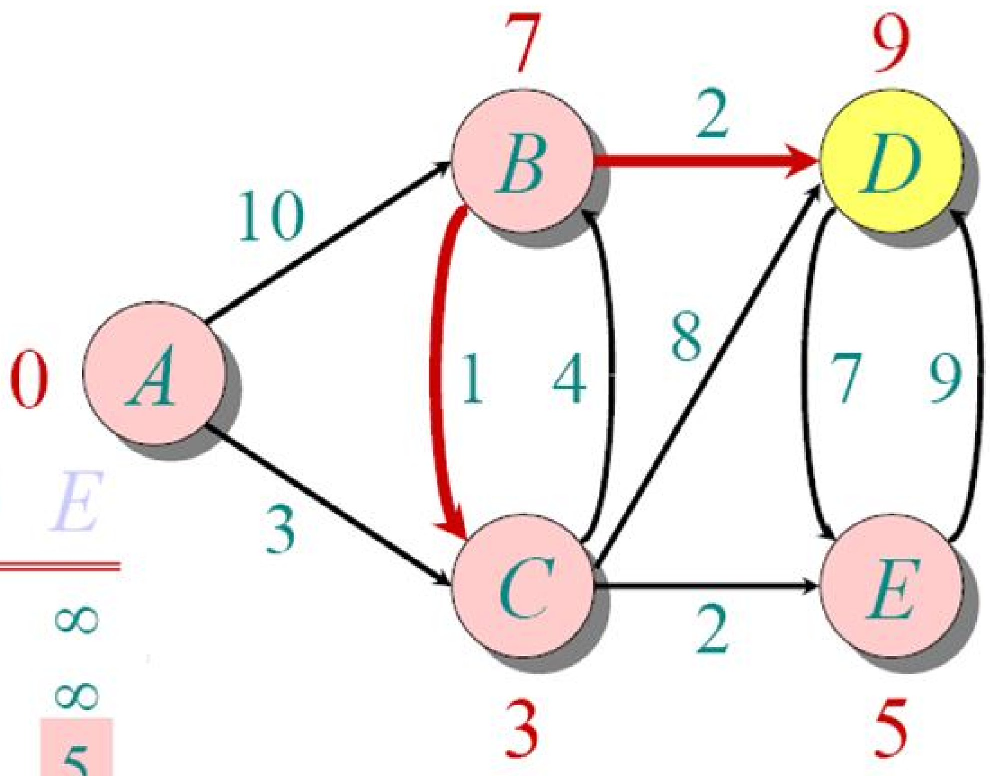


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { *A*, *C*, *E*, *B* }

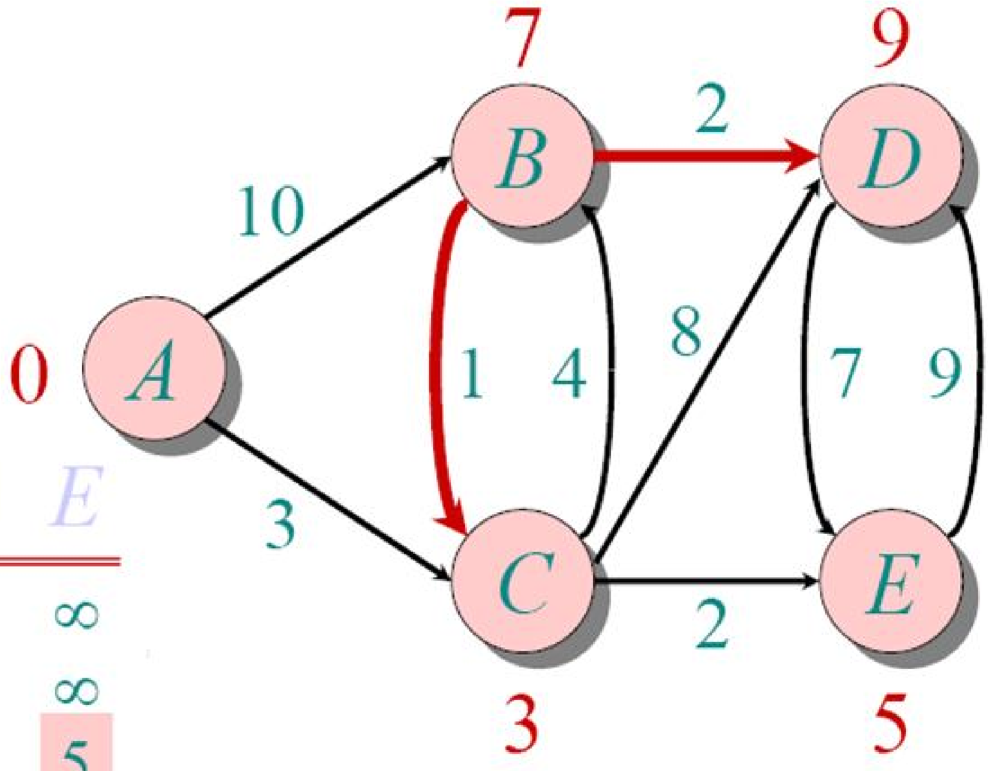
Dijkstra Animated Example

Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	



S: { A, C, E, B }

Dijkstra Animated Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	

S: { A, C, E, B, D }

Implementations and Running Times

The simplest implementation is to store vertices in an array or linked list. This will produce a running time of

$O(|V|^2 + |E|)$

Where |E| is for search needed to find the minimum distance node. //A node may be connected to a maximum of E nodes

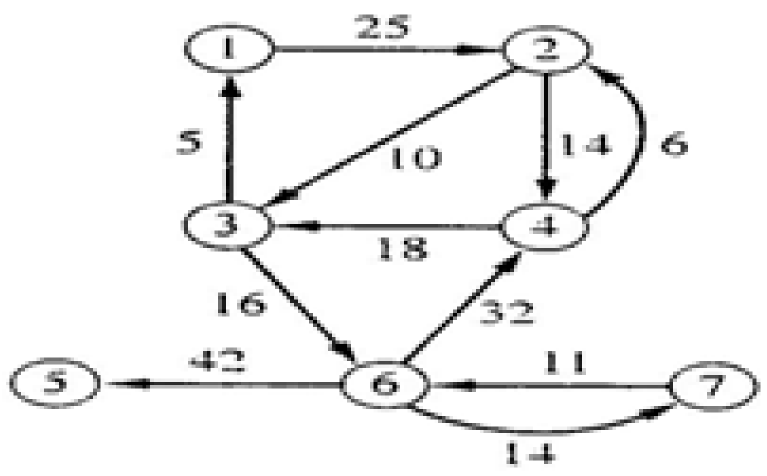
For sparse graphs, or graphs with very few edges and many nodes, it can be implemented more efficiently storing the graph in an adjacency list using a binary heap or priority queue. This will produce a running time of

$O((|E|+|V|) \log |V|)$ //inner loop is replaced by a heap

Dijkstra's Algorithm - Why use it?

- As mentioned, Dijkstra’s algorithm calculates the shortest path to every vertex.
- Therefore, any time we want to know the optimal path to some other vertex from a determined origin, we can use Dijkstra’s algorithm.

Example

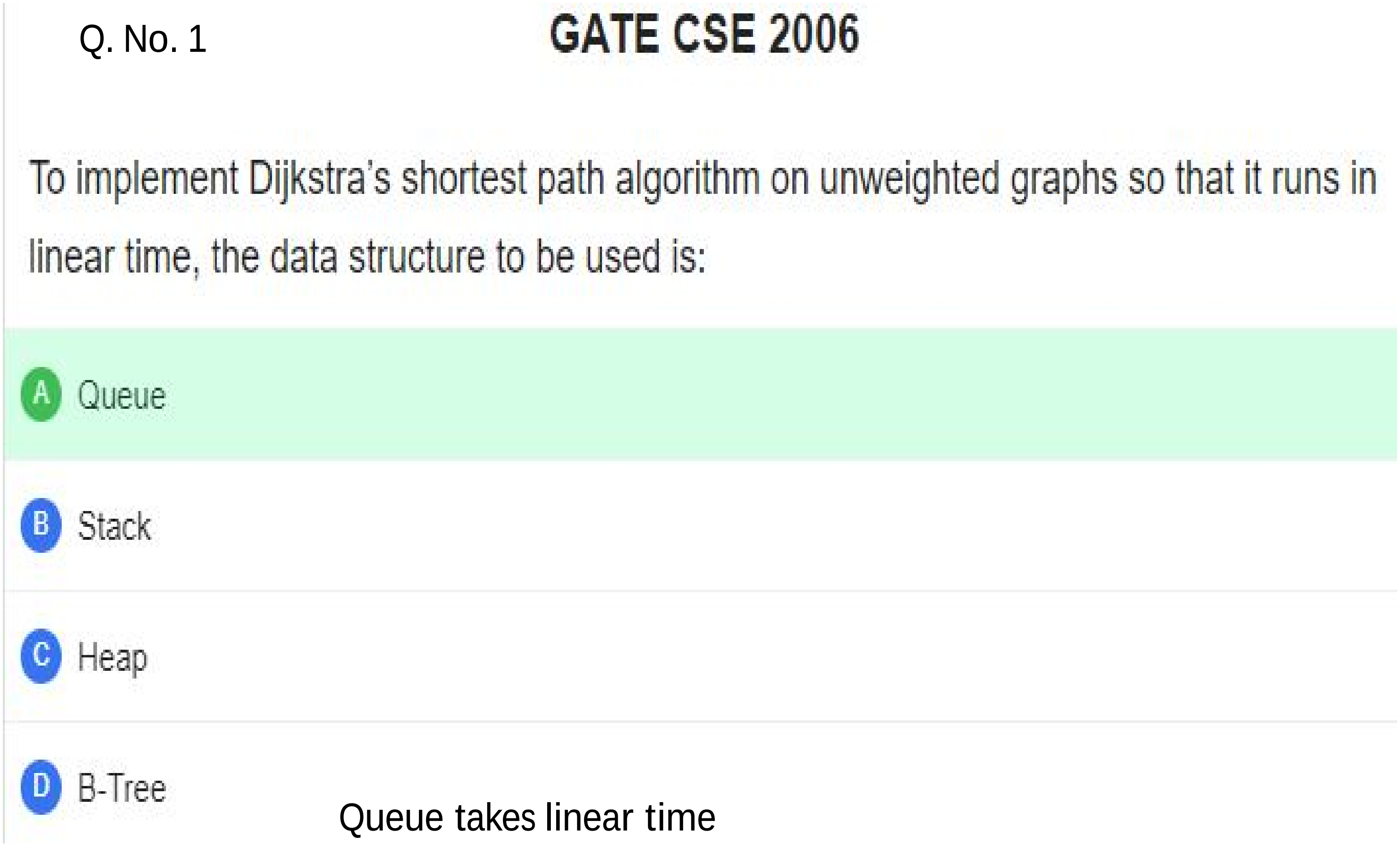


- Vertex 1 is the source

1	2	3	4	5	6	7
0	inf	inf	inf	inf	inf	Inf
	25	inf	inf	inf	inf	inf
		35	39	inf	inf	inf
			39	inf	51	Inf
				inf	51	inf
				93		65
				93		

Previous Gate Questions

Q. No. 1



GATE CSE 2006

To implement Dijkstra's shortest path algorithm on unweighted graphs so that it runs in linear time, the data structure to be used is:

[✓ A Queue]

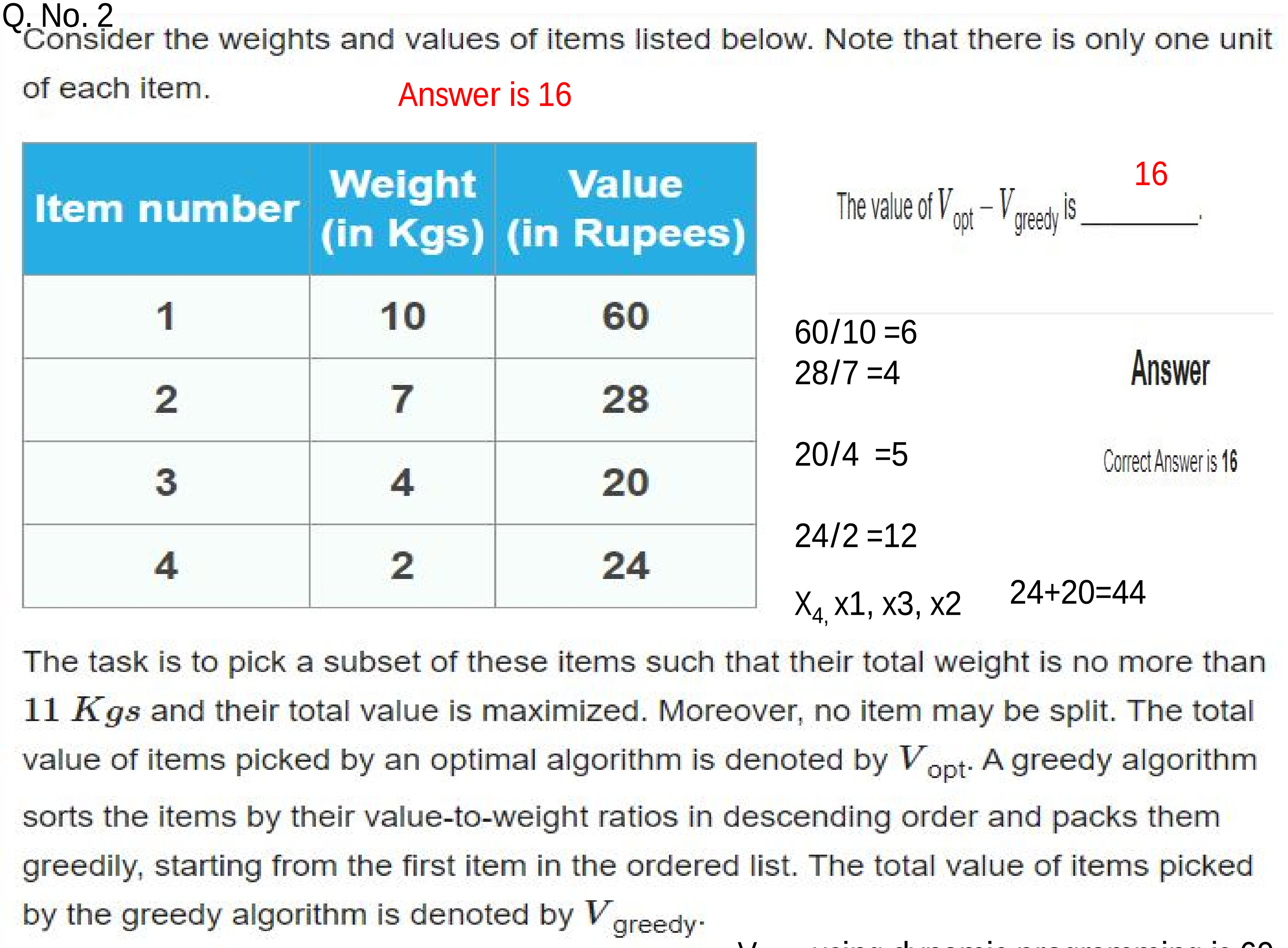
B Stack

C Heap

D B-Tree

Queue takes linear time

Q. No. 2

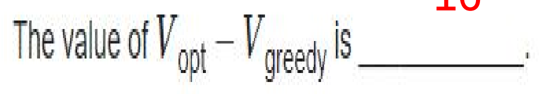


Consider the weights and values of items listed below. Note that there is only one unit of each item.

Answer is 16

Item number	Weight (in Kgs)	Value (in Rupees)
1	10	60
2	7	28
3	4	20
4	2	24

The task is to pick a subset of these items such that their total weight is no more than $11\ Kgs$ and their total value is maximized. Moreover, no item may be split. The total value of items picked by an optimal algorithm is denoted by V_{opt}. A greedy algorithm sorts the items by their value-to-weight ratios in descending order and packs them greedily, starting from the first item in the ordered list. The total value of items picked by the greedy algorithm is denoted by V_{greedy}.



The value of $V_{opt} - V_{greedy}$ is ________. 16

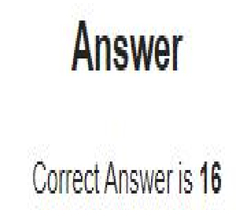
60/10 =6

28/7 =4

20/4 =5

24/2 =12

$X_{4,}$ x1, x3, x2   24+20=44



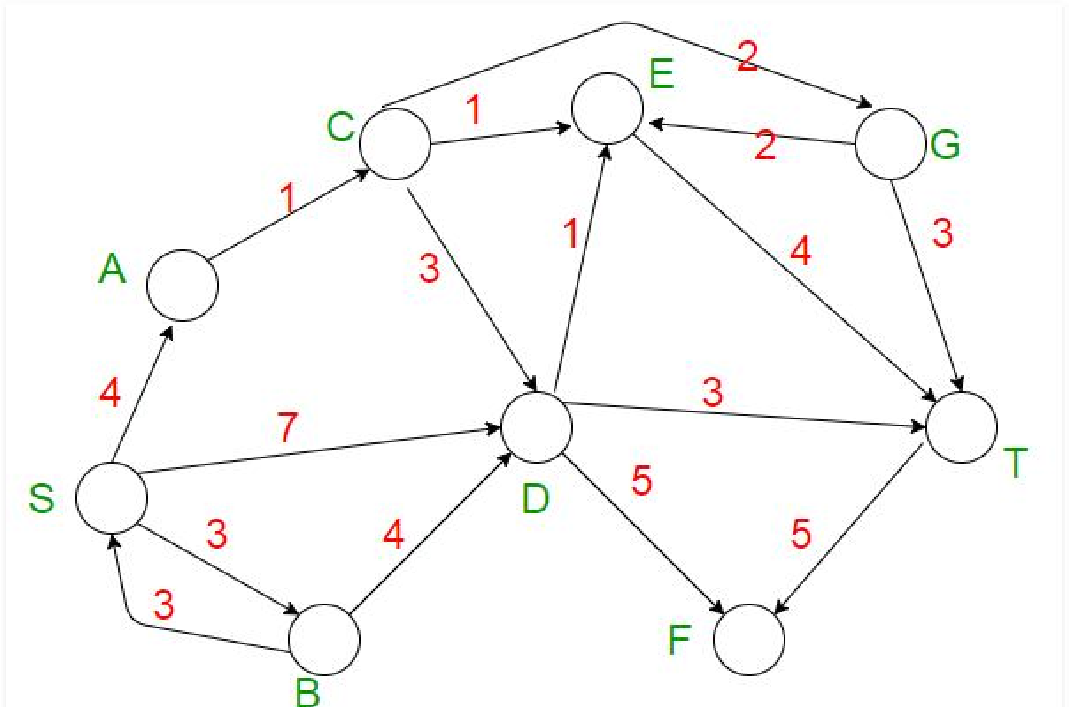
Answer

Correct Answer is **16**

V_{opt}= using dynamic programming is 60

Q. No. 3

Consider the directed graph shown in the figure below. There are multiple shortest paths between vertices S and T. Which one will be reported by Dijstra?s shortest path algorithm? Assume that, in any iteration, the shortest path to a vertex v is updated only when a strictly shorter path to v is discovered.



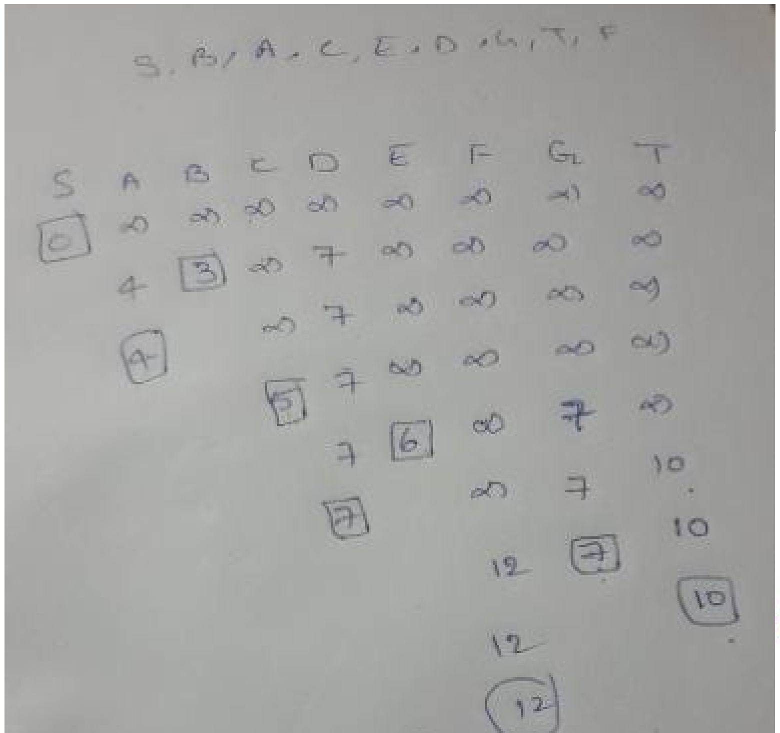
(A) SDT
(B) SBDT
(C) SACDT
(D) SACET

Answer: (D)

Explanation:

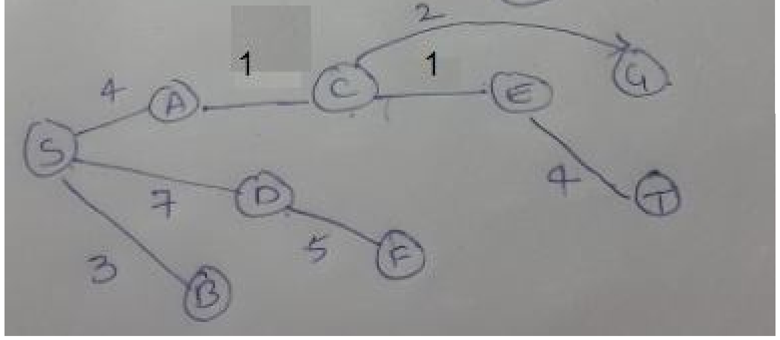
Background Required – Dijkstra's Single Source Shortest Path Algorithm

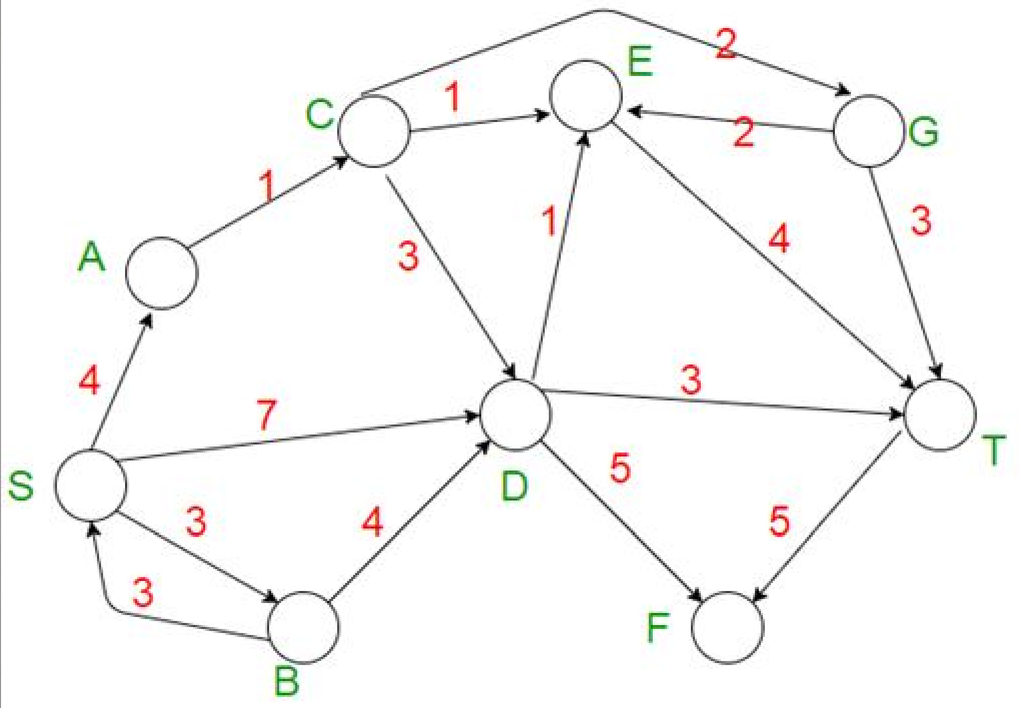
Explanation – Applying Dijkstra's algorithm to compute the shortest distances from S and finally generating the Tree as given below in the diagram.



S, B, A, C, E, D, G, T, F

S	A	B	C	D	E	F	G	T
$\boxed{0}$	∞	∞	∞	∞	∞	∞	∞	∞
	4	$\boxed{3}$	∞	7	∞	∞	∞	∞
	$\boxed{4}$		∞	7	∞	∞	∞	∞
			$\boxed{5}$	7	∞	∞	∞	∞
				7	$\boxed{6}$	∞	7	∞
				$\boxed{7}$		∞	7	10
						12	$\boxed{7}$	10
								$\boxed{10}$
						12		
						$\boxed{12}$		

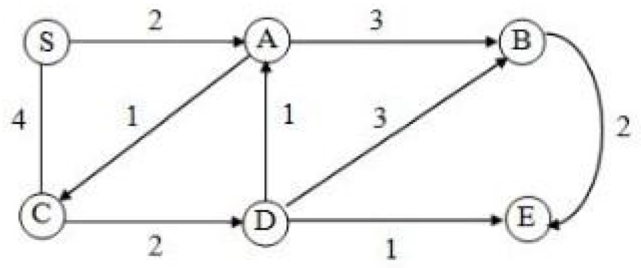




Q. No. 4

If we run disjkstra algorithm on source vertex (S) to the following graph then which of the following is possible order of visiting nodes?

I. S, A, C, D, E, B   II. S, A, D, C, E, B   III. S, A, D, E, C, B



(A) only I, III   (B) only II, III

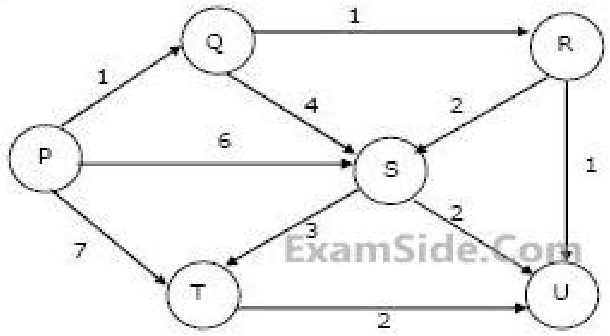
(C) only I, II   (D) I, II, III

Answer: None of the Above

Q. No. 5

GATE CSE 2004

Suppose we run Dijkstra's single source shortest-path algorithm on the following edge-weighted directed graph with vertex P as the source.



In what order do the nodes get included into the set of vertices for which the shortest path distances are finalized?

A P, Q, R, S, T, U

[✓ B] P, Q, R, U, S, T

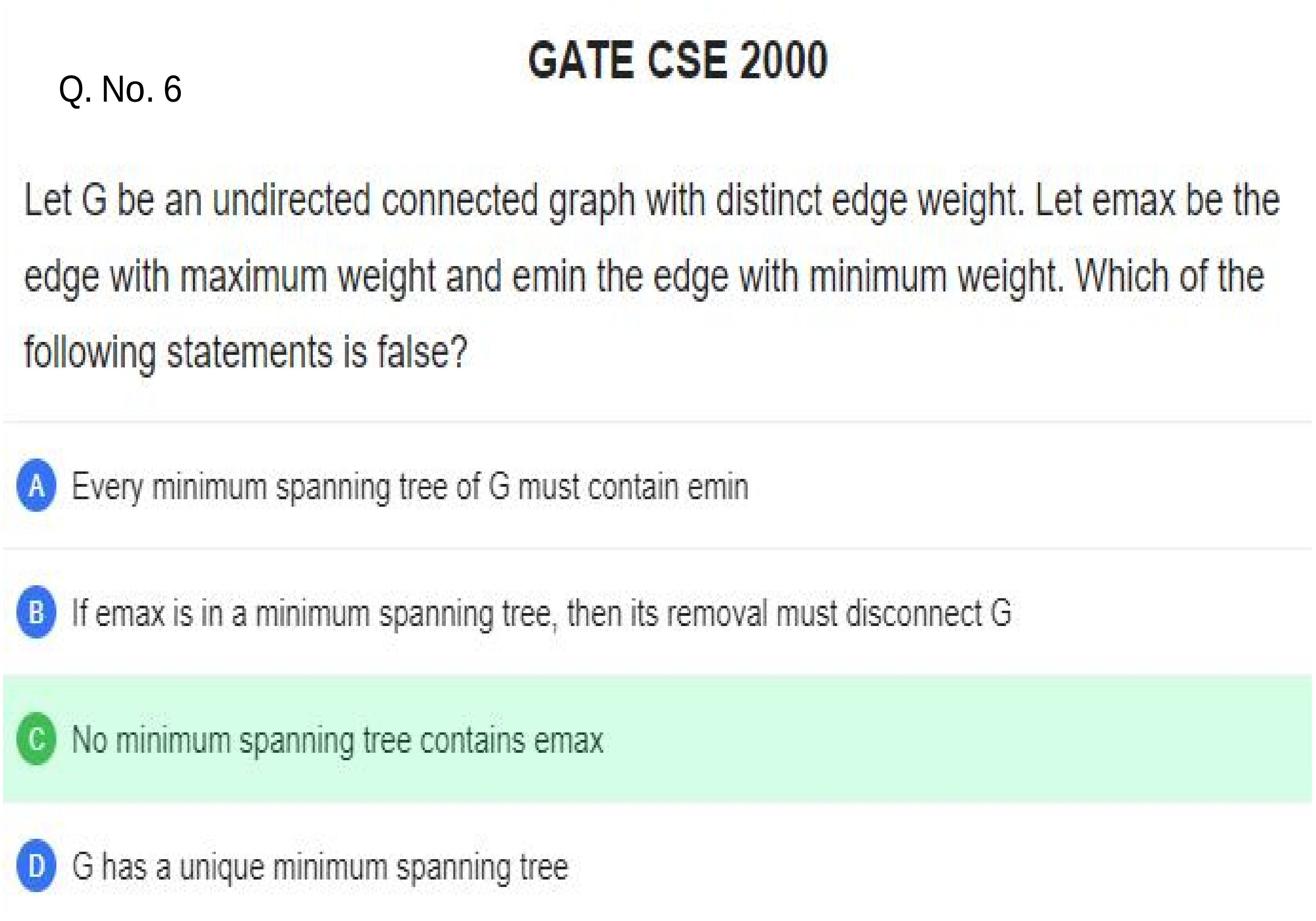
C P, Q, R, U, T, S

D P, Q, T, R, U, S

p	q	r	s	t	u
0	inf	inf	inf	inf	inf
	1	inf	6	7	inf
		2	5	7	8
			4		3

p, q, r, u, s,t

Q. No. 6



GATE CSE 2000

Let G be an undirected connected graph with distinct edge weight. Let emax be the edge with maximum weight and emin the edge with minimum weight. Which of the following statements is false?

A Every minimum spanning tree of G must contain emin

B If emax is in a minimum spanning tree, then its removal must disconnect G

[✓ C] No minimum spanning tree contains emax

D G has a unique minimum spanning tree

Q. No. 7 Given below are some algorithms, and some algorithm design paradigms.

List-I

A. Dijkstra’s Shortest Path 3

B. Floyd-Warshall algorithm to compute all pairs shortest path 2

C. Binary search on a sorted array 1

D. Backtracking search on a graph 4

List-II

1. Divide and Conquer
2. Dynamic Programming
3. Greedy design
4. Depth-first search
5. Breadth-first search

Match the above algorithms on the left to the corresponding design paradigm they follow Codes:

A B C D

(a) 1 3 1 5

(b) 3 3 1 5

(c) 3 2 1 4

(d) 3 2 1 5

Options:

A) a

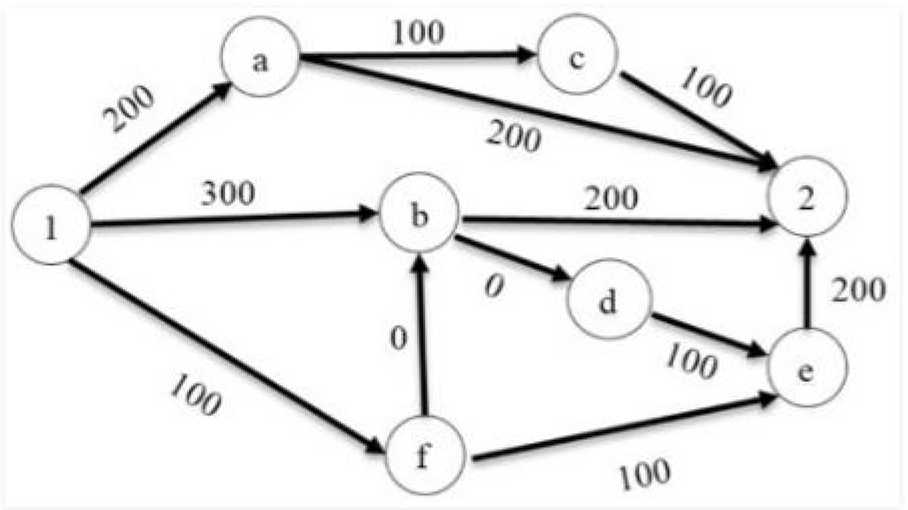
B) b

C) c

D) D Answer C

Q. No. 9

There are multiple routes to reach from node 1 to node 2, as shown in the network.



The cost of travel on an edge between two nodes is given in rupees. Nodes ‘a’, ‘b’, ‘c’, ‘d’, ‘e’, and ‘f’ are toll booths. The toll price at toll booths marked ‘a’ and ‘e’ is Rs. 200, and is Rs. 100 for the other toll booths. Which is the cheapest route from node 1 to node 2 ?

(A) 1-a-c-2

(B) 1-f-b-2

(C) 1-b-2

(D) 1-f-e-2

1	a	b	c	d	e	f	2
0	inf	inf	inf	inf	inf	inf	inf
	200	300	inf	inf	inf	100	inf
		100	inf	inf	200		inf
				100	200		300
					200		300
							300

Answer: (B)

1-f-b-2

Q. No. 10

GATE CSE 2005

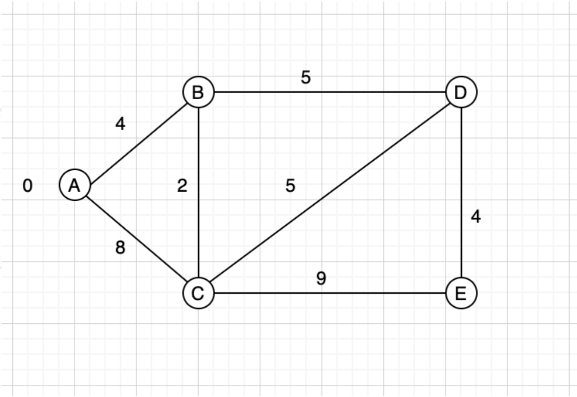
Let G(V, E) an undirected graph with positive edge weights. Dijkstra's single-source shortest path algorithm can be implemented using the binary heap data structure with time complexity:

A $O\left(|V|^2\right)$

B $O(|E| + |V| \log |V|)$

C $O(|V| \log |V|)$

[✓ D] $O((|E| + |V|) \log |V|)$



	A	B	C	D	E
A	0	4	8	0	0
B	4	0	2	5	0
C	8	2	0	5	9
D	0	5	5	0	4
E	0	0	9	4	0