Greedy algorithms

Contents

- Interval Scheduling
- Scheduling to minimize lateness.
- Optimal Caching

Interval Scheduling-Problem

- Formally Requests **1,2,…,n.** Request **i has start time s(i) and finish time f(i) > s(i)**
  - Requests **i and j are compatible iff either**
    - request **i is for a time entirely before request j**

      f(i) <= s(j)
    - or, request **j is for a time entirely before**

      request **i f(j) <= s(i)**
- Goal: Find maximum size subset **A of compatible** requests

Interval Scheduling- Approaches

- [Earliest start time] Consider jobs in ascending order of s_j.

counterexample for earliest start time

- [Shortest interval] Consider jobs in ascending order of $f_j - s_j$.

counterexample for shortest interval

- [Fewest conflicts] For each job j, count the number of conflicting jobs c_j. Schedule in ascending order of c_j.

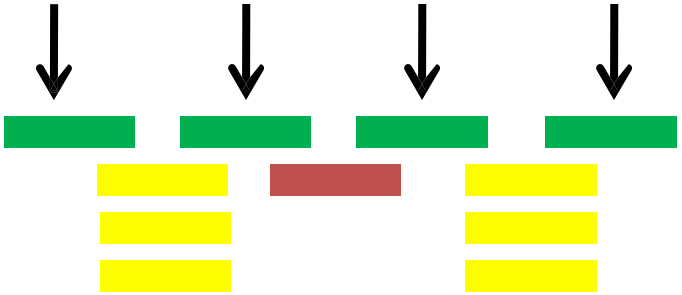
counterexample for fewest conflicts

Interval Scheduling- Approaches

- [Earliest finishtime] Consider jobs in ascending order of f_j.

Solution =4 intervals

Solution =2 intervals



Solution =4 intervals

Interval Scheduling- Algorithm

Initially let R be the set of all requests, and

let A be empty

While R is not yet empty

Choose a request $i \in R$ that has the smallest finishing time — 1

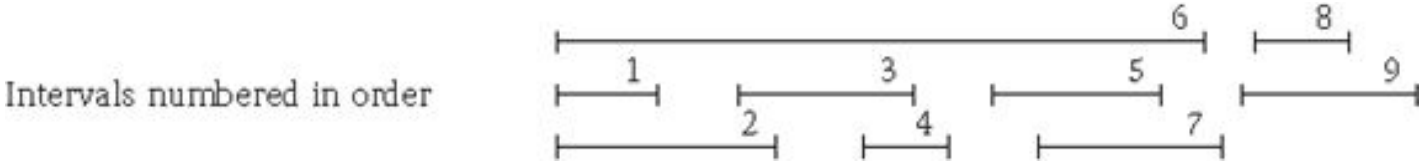
Add request i to A — 2

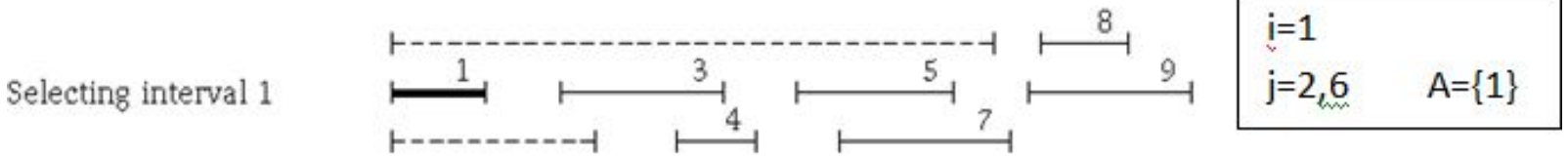
Delete all requests from R that are not compatible with request i — 3

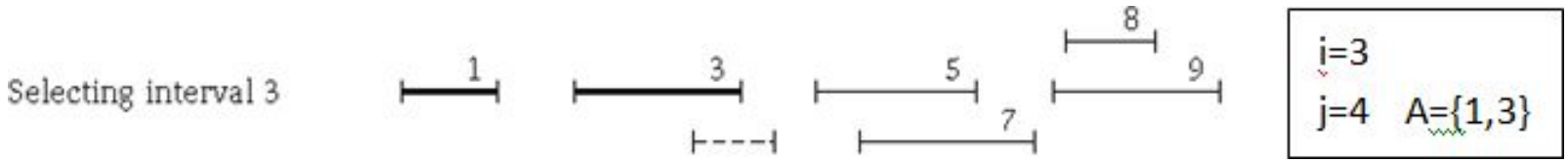
EndWhile

Return the set A as the set of accepted requests

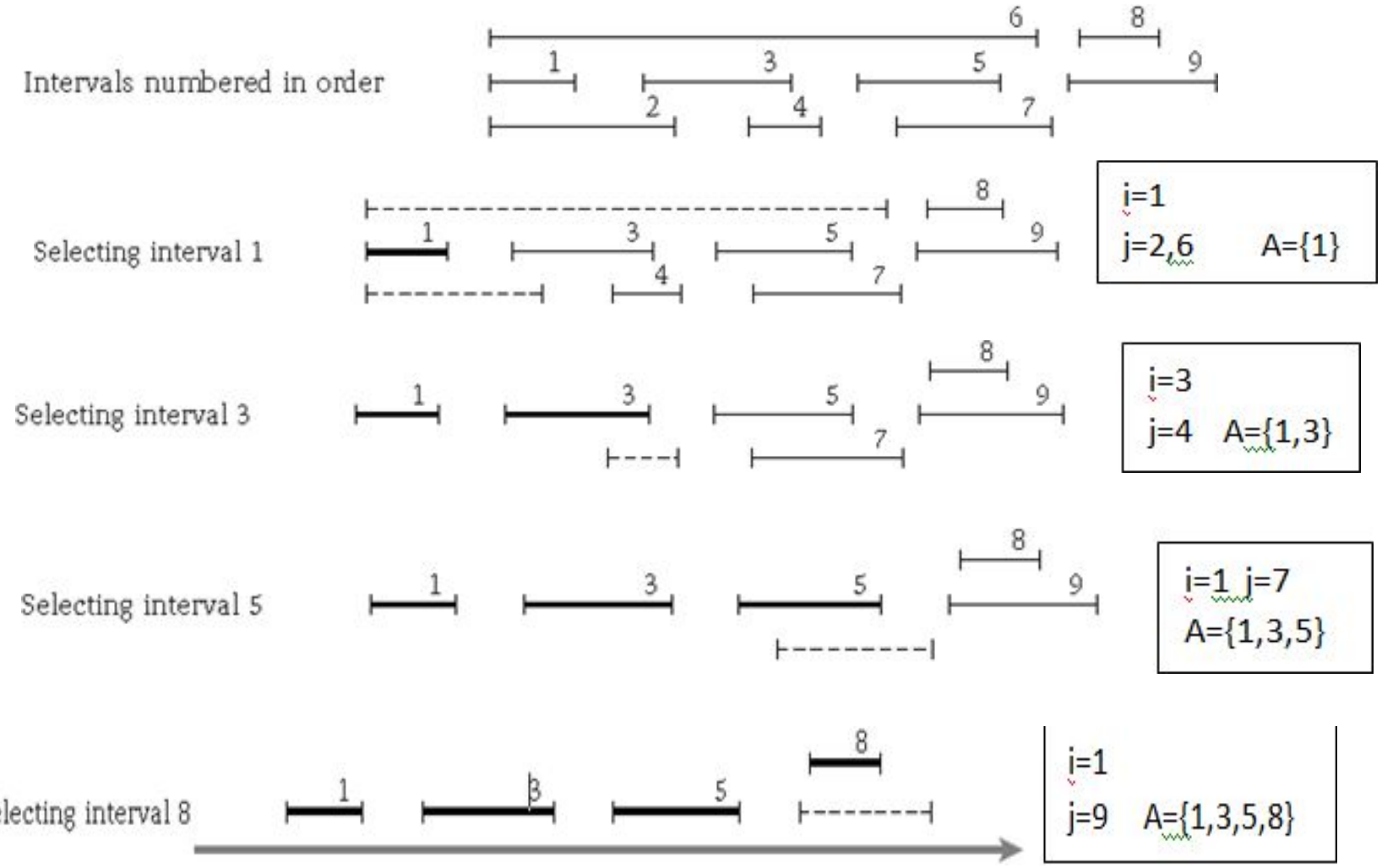
Interval Scheduling-Example





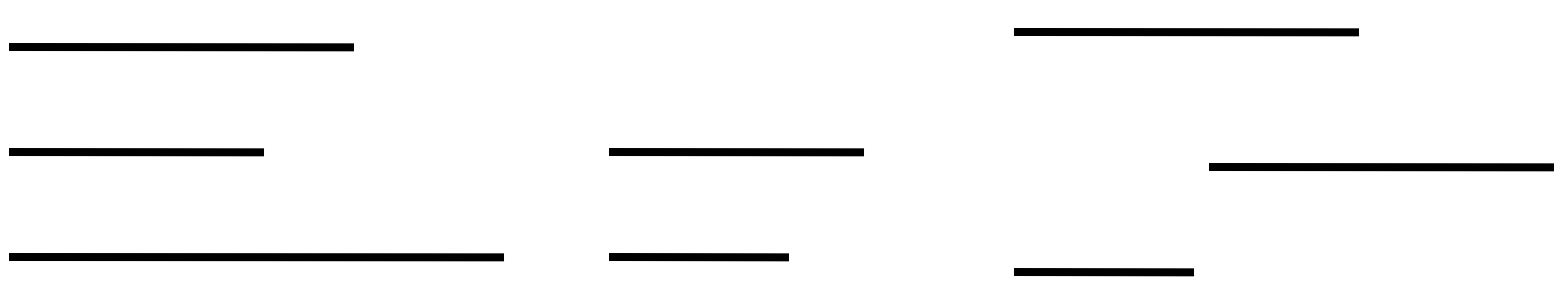


Interval Scheduling-Example



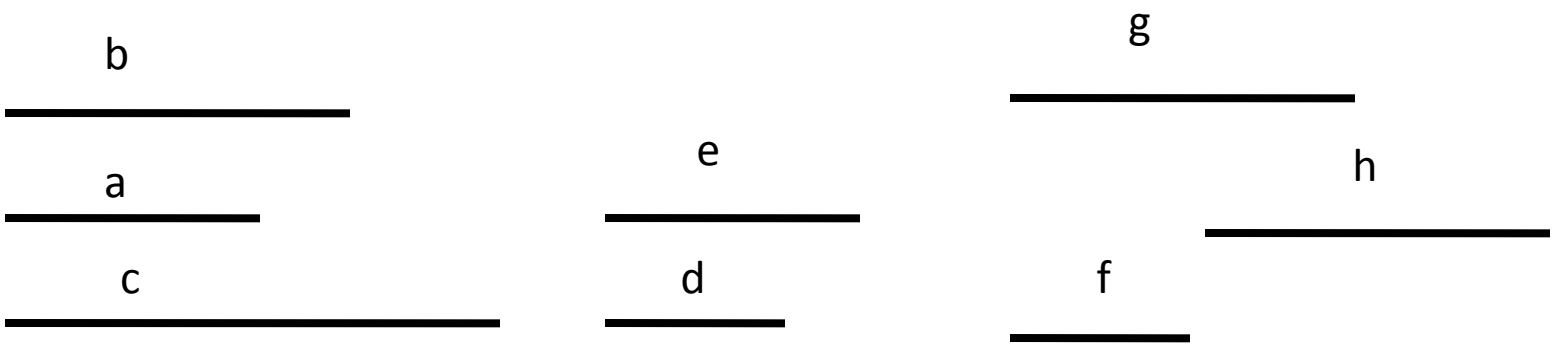
Problem – Interval Scheduling

- Find the intervals that can be scheduled for the following set of requests.



Problem – Interval Scheduling

- Find the intervals that can be scheduled for the following set of requests.



Implementation and running time Analysis

```
f_sorted=sort the n requests in
        order of finish time                                  O(n logn)
    schedule = {}
    last_finish = 0
    for i = 1 to n:
        id=get_request_id(f_sorted[i])                        O(n)
        if s(id)>= last_finish:
                Add i to schedule
                last_finish =
                f_sorted(i)
        endif
    endfor
return
schedule
```

Analysis-Proof of correctness

- *The greedy algorithm returns an optimal set A.*
  - *A is a compatible set of requests.*
  - *When do you say it as compatible?*
  - *$f(i) \leq f(j)$ □ (1)*
  - Let **A=$\{i_1, \ldots, i_k\}$** Note that $|A| = k$. Similarly,
  - **O =$\{j_1, \ldots, j_m\}$** Our goal is to prove that
  - $k = m$.

Analysis-Proof of correctness

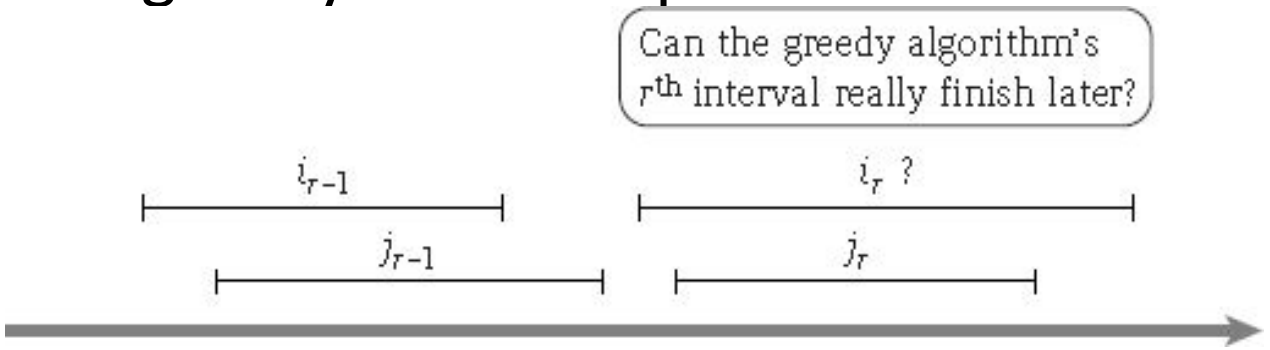
- *A is a compatible set of requests*
  - Let $i_1, \ldots, i_k$ be the set of requests in ***A*** in the order they were added to *A*. Note that $|A| = k$. Similarly,
  - Let the set of requests in optimal set **O** be denoted by $j_1, \ldots, j_m$. Our goal is to prove that $k = m$.
  - we now prove that for each $r \geq 1$, the *r*th accepted request in the algorithm's schedule finishes no later than the *r*th request in the optimal schedule.

    i.e. *For all indices $r \leq k$ we have $f(i_r) \leq f(j_r)$-* □

(2)

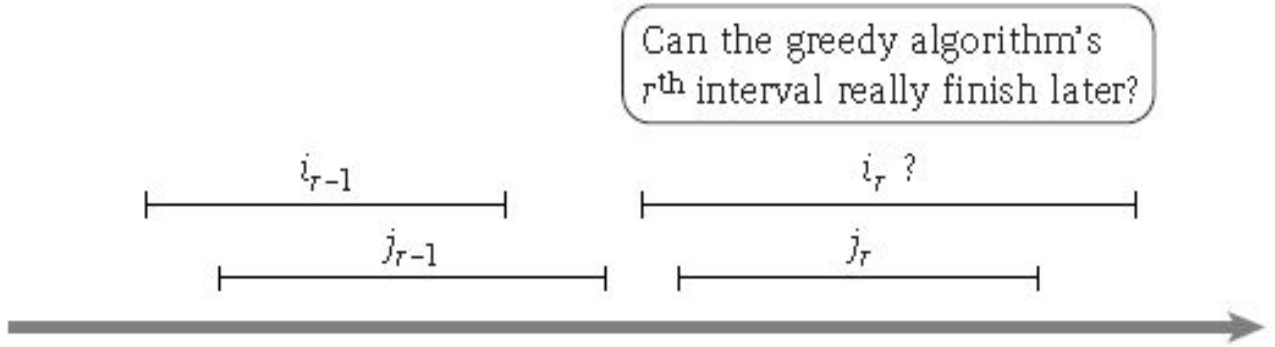
Analysis-Proof of correctness

- *A is a compatible set of requests*
  - i.e. *For all indices $r \le k$ we have $f(i_r) \le f(j_r)$.*
  - For $r = 1$ the statement is clearly true: the algorithm starts by selecting the request i_1 with minimum finish time.
  - let $r > 1$, assume the following set of requests for the greedy and the optimal schedule.



Analysis-Proof of correctness

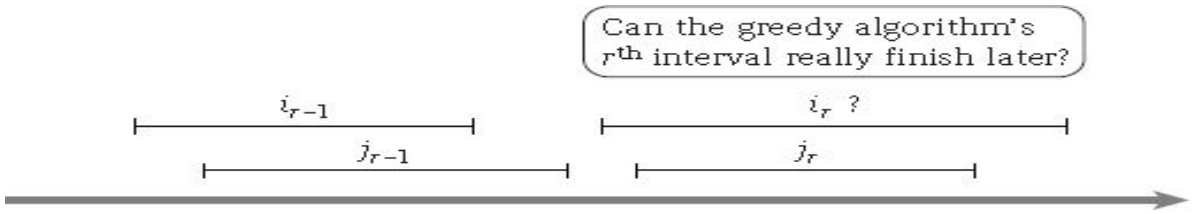
- *A is a compatible set of requests*
  - i.e. *For all indices $r \le k$ we have $f(i_r) \le f(j_r)$.*



  - $f(j_{r-1}) \le s(j_r)$.
  - $f(i_{r-1}) \le f(j_{r-1})$,
  - we get $f(i_{r-1}) \le s(j_r)$.

Analysis-Proof of correctness

- *A is a compatible set of requests*
  - i.e. *For all indices r ≤ k we have f (i_r) ≤ f (j_r).*



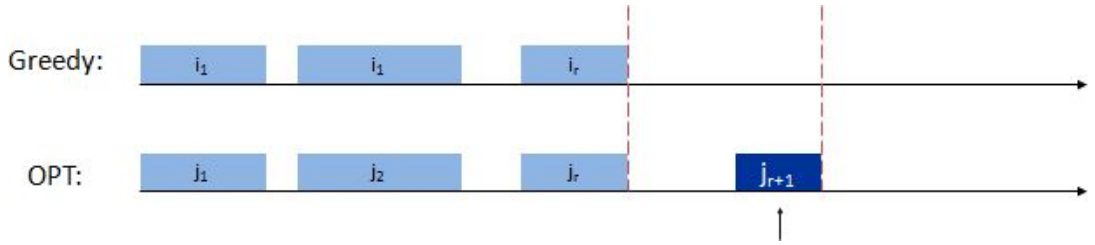
  - *f (j_{r-1}) ≤ s(j_r).*
  - *f (i_{r-1}) ≤ f (j_{r-1}),*
  - we get *f (i_{r-1}) ≤ s(j_r).*
  - The greedy algorithm selects the available interval with *smallest* finish time; since interval j_r is one of these available intervals, we have *f (i_r) ≤ f (j_r).*

Analysis-Proof of correctness

- *The greedy algorithm returns an optimal set A.*
  - *A is set of compatible requests---* *1*
  - *For all indices r ≤ k we have f (i_r) ≤ f (j_r)--* *2*
  - If *A* is not optimal, then an optimal set O must have more requests, that is, we must have *m* > *k*.
  - With *r* = *k*, we get that *f* (i_k) ≤ *f* (j_k). By 2
  - Since *m* > *k*, there is a request j_{k+1} in O

Analysis-Proof of correctness

- *The greedy algorithm returns an optimal set A.*
  - If *A* is not optimal, then an optimal set O must have more requests, that is, we must have $m > k$.
  - With $r = k$, we get that $f(i_k) \leq f(j_k)$. By 2
  - Since $m > k$, there is a request j_{k+1} in O



  - But the greedy algorithm stops with request i_k, and it is only supposed to stop when *R* is empty—a contradiction. Hence, the set A returned is optimal

Summary- Interval Scheduling

- Problem consisted of a set of requests in R and goal is to find the maximum subset from R.
- What approach is suitable?
  - Earliest finish.
- The time complexityof the algorithm discussed is nlogn.
- The optimality of the set of requests A is proved.

Scheduling to minimize lateness

Scheduling to minimize lateness

- Instead of a start time and finish time, the **request *i*** has a **deadline d_i**, and it requires a contiguous time interval of **length t_i**,
- Can be scheduled at any time before the deadline.

1	d=2	t=1
2	d=4	t=4

How can you schedule now for these type of requests??

Scheduling to minimize lateness

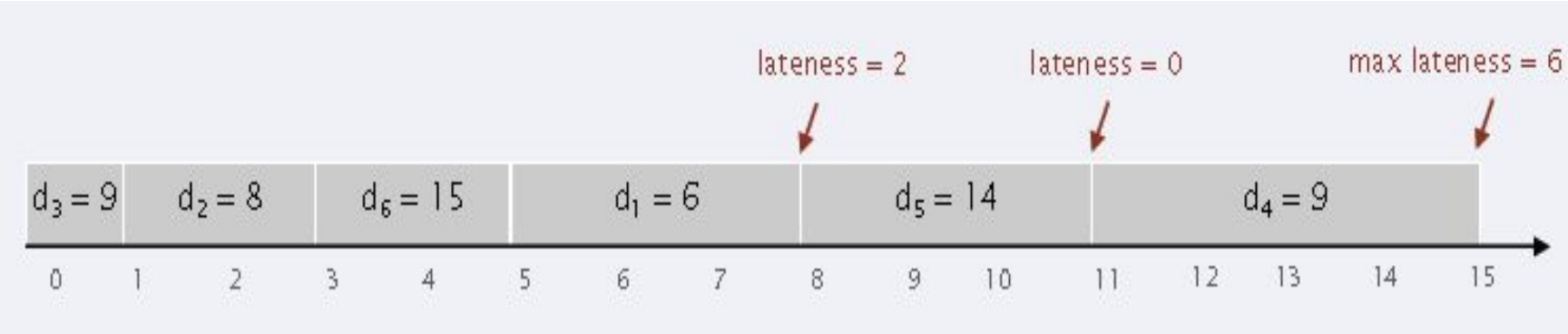
- Each interval is denoted by [*s(i)*, *f (i)*], with

$f(i) = s(i) + t_i$.

  - where, s(i)= start time of the interval i
  - f(i)=finish time of the interval i
  - t_i= time length of the interval I
- We say that a request *i* is *late* if it misses the deadline, that is, if $f(i) > d_i$.
- The *lateness* of such a request *i* is defined to be $l_i = f(i) - d_i$.

Scheduling to minimize lateness- Example

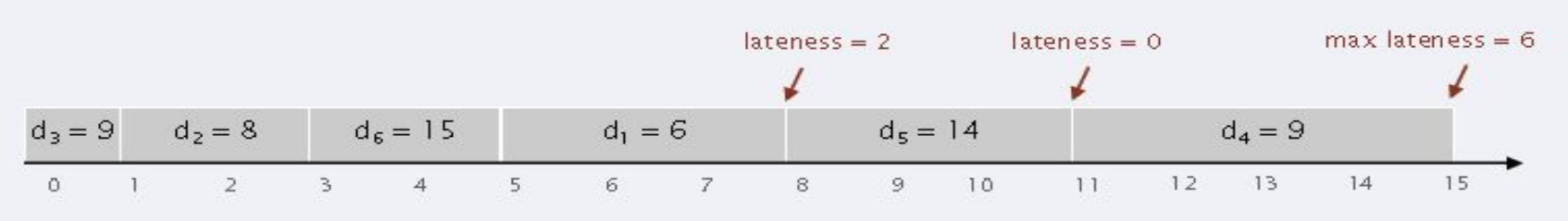
	1	2	3	4	5	6
t(j)	3	2	1	4	3	2
d(j)	6	8	9	9	14	15



Scheduling to minimize lateness- Goal

- The goal in our new optimization problem will be to schedule all requests, using

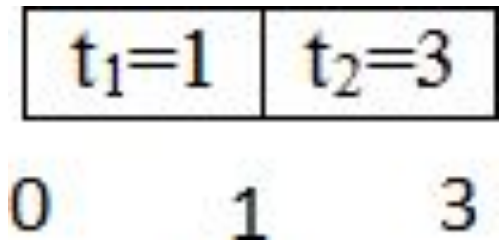
non-overlapping intervals, so as to minimize the *maximum lateness*, $L = \max_i l_i$.



Scheduling to minimize lateness

- **Natural approaches.**
  - increasing length t_i

Request	1	2
t_j	1	3
d_j	2	5



Scheduling to minimize lateness

- Natural approaches.
  - increasing length t_i

Request	1	2
t_j	1	10
d_j	100	10

$t_1=1$, $d_1=100$	$t_2=10$, $d_2=10$

0 1 11

Lateness for the request 2 is 1

Counter example

If request 2 is scheduled first

$t_2=10$ $d_2=10$	$t_1=1$ $d_1=100$

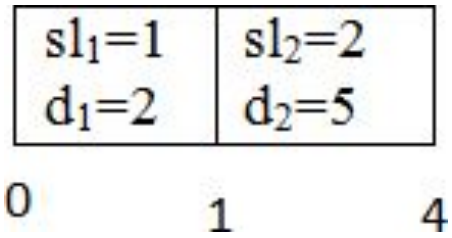
0 10 11

Lateness for the request 2 is 0

Scheduling to minimize lateness

- Natural approaches.
  - ***slack time*** $d_i - t_i$ is very small

Request	1	2
t_j	1	3
d_j	2	5
Sl_i (Slack)	1	2



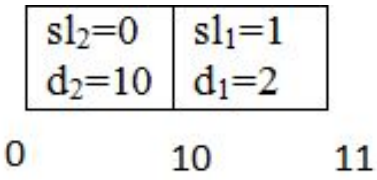
Scheduling to minimize lateness

- Natural approaches.
  - ***slack time*** $d_i - t_i$ is very small

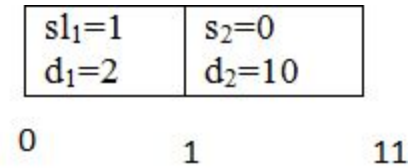
Counter example

Request	1	2
t_j	1	10
d_j	2	10
sl_i	1	0

If request 1 Was scheduled first



Lateness for the request 1 is 9

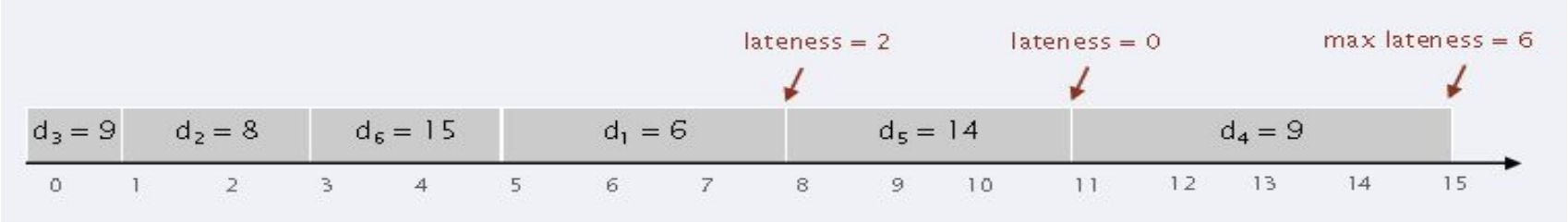


Lateness for the request 2 is 1

Scheduling to minimize lateness- Goal revisited

- The goal in our new optimization problem will be to schedule all requests, using

non-overlapping intervals, so as to minimize the *maximum lateness*, $L = \max_i l_i$.



DON'T MISS THE DEADLINE!

EARLIEST

Scheduling to minimize lateness- Algorithm

Order the jobs in order of their deadlines
Assume for simplicity of notation that $d_1 \leq \ldots \leq d_n$ 1
Initially, $f = 0$

Consider the jobs $i = 1, \ldots, n$ in this order

Assign job i to the time interval from $s(i) = f$ to $f(i) = f + t_i$ 2

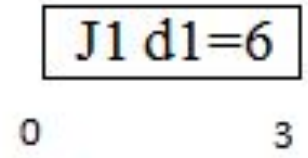
Let $f = f + t_i$ 3

End
Return the set of scheduled intervals $[s(i), f(i)]$ for $i = 1, \ldots, n$

Scheduling to minimize lateness- Example

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15

1. i=1 Add Job J1 to the schedule.



s(1) = f =0

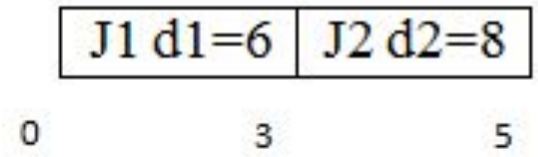
f(1)= s(1)+t(1)=0+3=3

f= f +t(1)=0+3=3

Scheduling to minimize lateness- Example

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15

2. i=2 Add Job J2 to the schedule.



s(2) = f =3

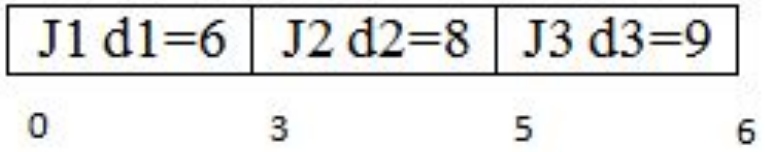
f(2)= s(2)+t(2)=3+2=5

f= f +t(2)=3+2=5

Scheduling to minimize lateness- Example

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15

3. i=3 Add Job J3 to the schedule.



s(3) = f =5

f(3)= s(3)+t(3)=5+1=6

f= f +t(3)=5+1=6

Scheduling to minimize lateness- Example

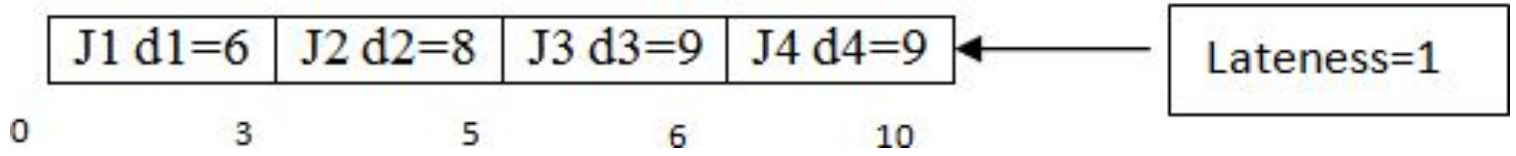
	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15

4. i=4 Add Job J4 to the schedule.

s(4) = f =6

f(4)= s(4)+t(4)=6+4=10

f= f + t(4)=6+4=10

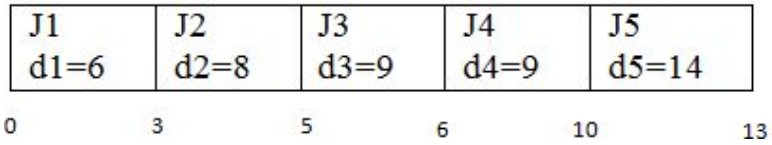


Scheduling to minimize lateness- Example

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15

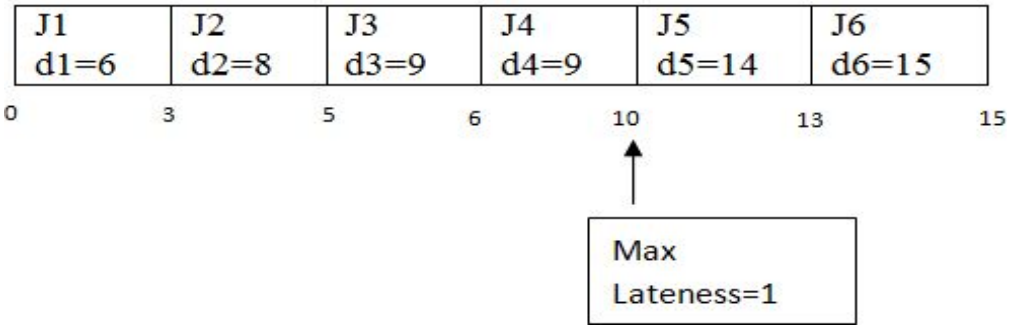
5. i=5 Add Job J5 to the schedule.

s(5) = f =10
f(5)= s(5)+t(5)=10+3=13
f= f +t(5)=10+3=13



6. i=6 Add Job J6 to the schedule.

s(6) = f =13
f(6)= s(6)+t(6)=13+2=15
f= f +t(6)=13+2=15



Scheduling to minimize lateness- Running time

Order the jobs in order of their deadlines **O(N LOGN)**

Assume for simplicity of notation that $d_1 \leq \ldots \leq d_n$

Initially, $f = 0$

Consider the jobs $i = 1, \ldots, n$ in this order **O(N)**

Assign job i to the time interval from $s(i) = f$ to $f(i) = f + t_i$

Let $f = f + t_i$

End

Return the set of scheduled intervals $[s(i), f(i)]$ for $i = 1, \ldots, n$

Scheduling to minimize lateness- Proof of correctness

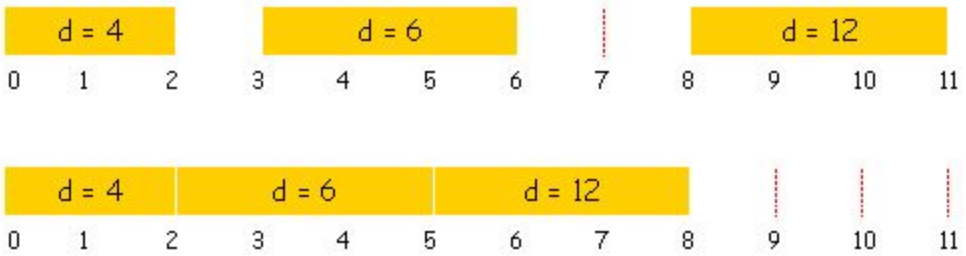
- **There is an optimal schedule with no idle time and no inversions**
  - *(a) There is an optimal schedule with no idle time*
- We will show that if there is another schedule

  O (think optimal schedule)
  - then we can gradually change **O so that**
  - at each step the maximum lateness in **O** never gets worse
  - it eventually becomes the same cost as **A**

Scheduling to minimize lateness- Proof of correctness

- **There is an optimal schedule with no idle time and no inversions**
  - *(a) There is an optimal schedule with no idle time*

Observation. There exists an optimal schedule with no idle time.



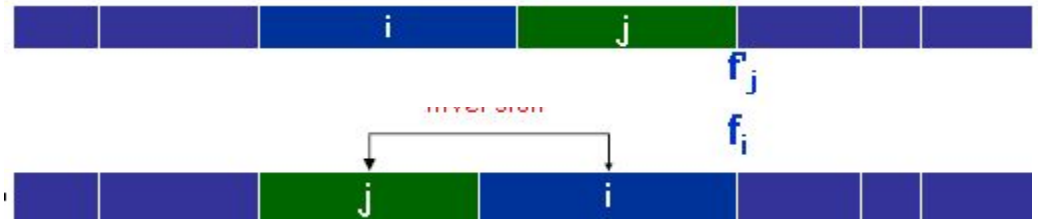
Observation. The greedy schedule has no idle time.

Scheduling to minimize lateness- Proof of correctness

- **There is an optimal schedule with no idle time and no inversions**
  - *(a) There is an optimal schedule with no idle time.*
  - *(b) There is an optimal schedule with no inversions.*

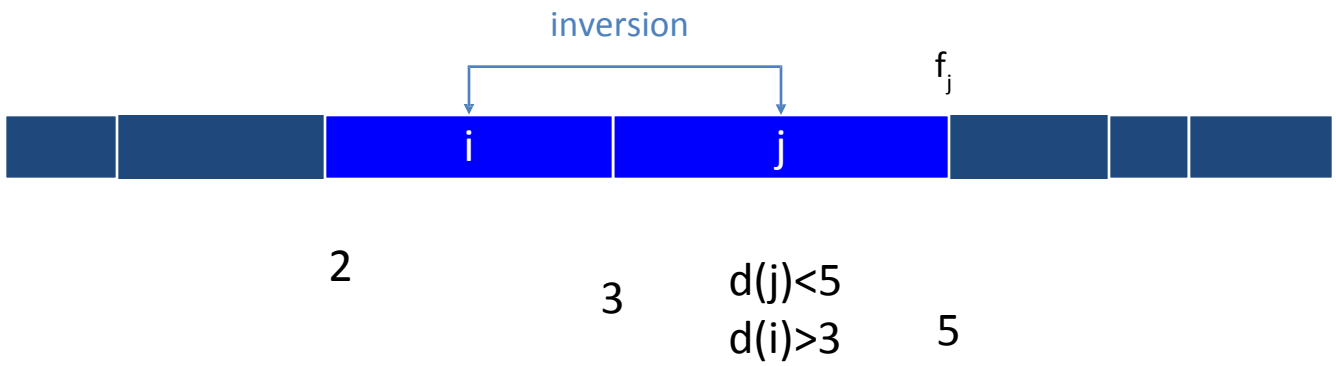
Identical

Non- Identical



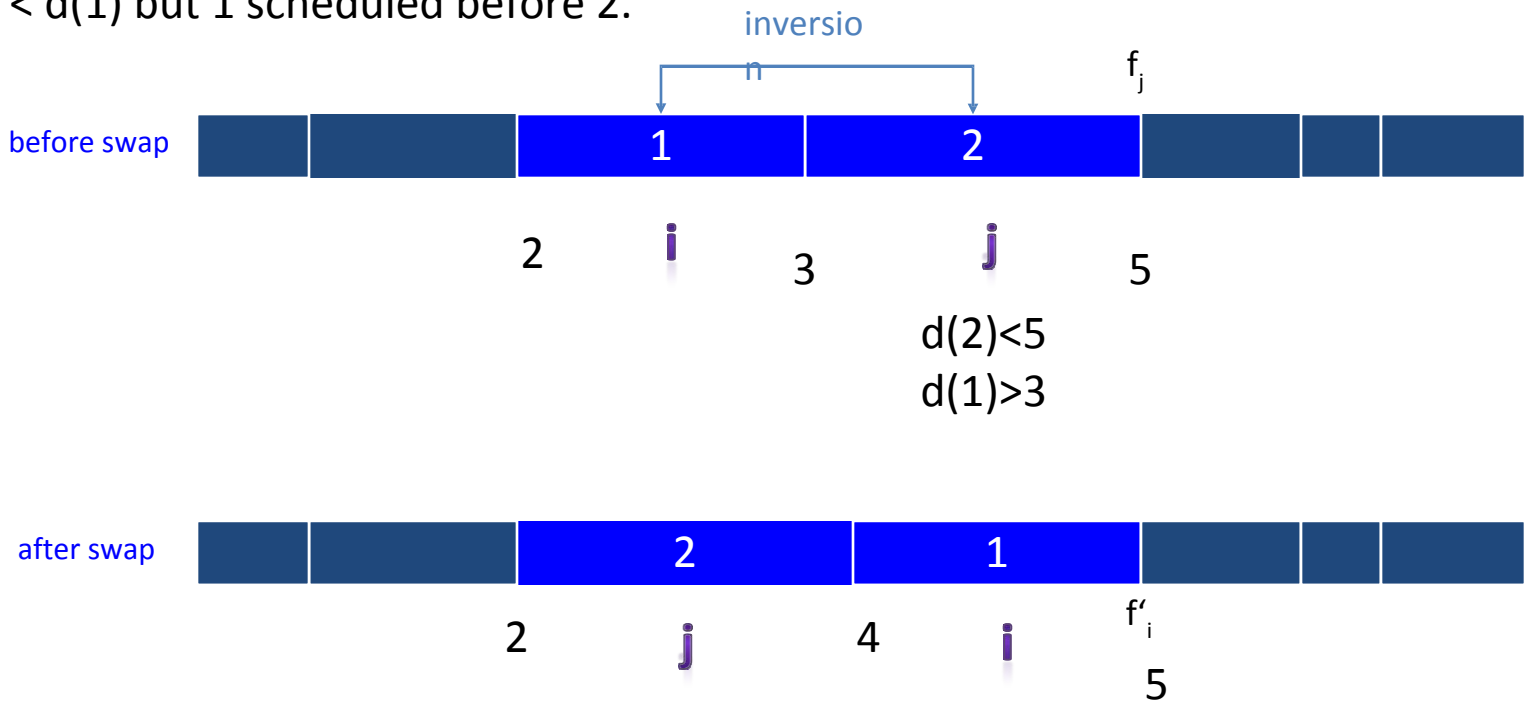
Minimizing Lateness: Inversions

- **Def** Given a schedule S, an inversion is a pair of jobs i and j such that: d(j) < d(i) but i scheduled before j.



Minimizing Lateness: Inversions

- **Def** Given a schedule S, an inversion is a pair of jobs 1 and 2 such that: d(2) < d(1) but 1 scheduled before 2.

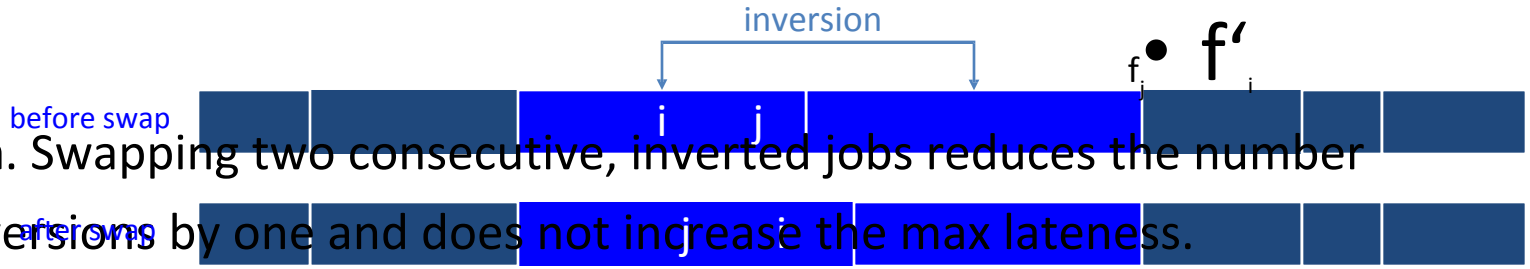


L1'= 5- 4 (say d1)=1
=5-4=1

L2=5- 4(say d2)=1
5-3=2

Minimizing Lateness: Inversions

- **Def.** Given a schedule S, an inversion is a pair of jobs i and j such that: d(j) < d(i) but i scheduled before j.



- **Claim**. Swapping two consecutive, inverted jobs reduces the number
- of inversions by one and does not increase the max lateness.

- Pf. Let l be the lateness before the swap, and let l' be it afterwards.

– $l'_k = l_k$ for all k ≠ i, j.

$\ell_j' \leq \ell_j$

- If job i is late:

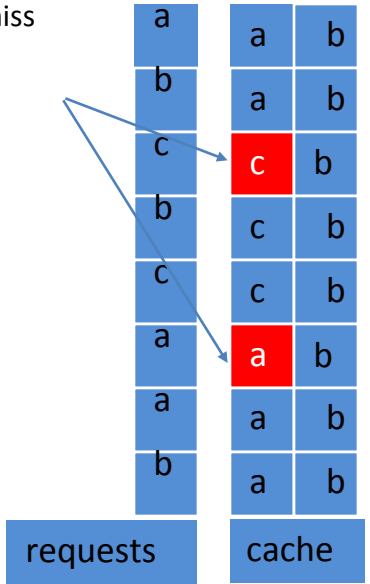
$$
\begin{aligned}
l' &= f' - d && \text{(definition)} \\
&= f_j - d_i && (i \text{ finishes at time } f_j) \\
&\leq f_j - d_j && (j < i) \\
&\leq l_j && \text{(definition)}
\end{aligned}
$$

Optimal caching- Problem

- Caching.
  - Cache with capacity to store k items.
  - Sequence of m item requests $d_1, d_2, \ldots, d_m$.
  - Cache hit: item already in cache when requested.
  - Cache miss: item not already in cache when requested: must bring requested item into cache, and evict some existing item, if full.

- Goal. Eviction schedule that minimizes number of cache misses.

- Ex: k = 2, initial cache = ab,
  requests: a, b, c, b, c, a, a, b.
- Optimal eviction schedule: 2 cache misses.



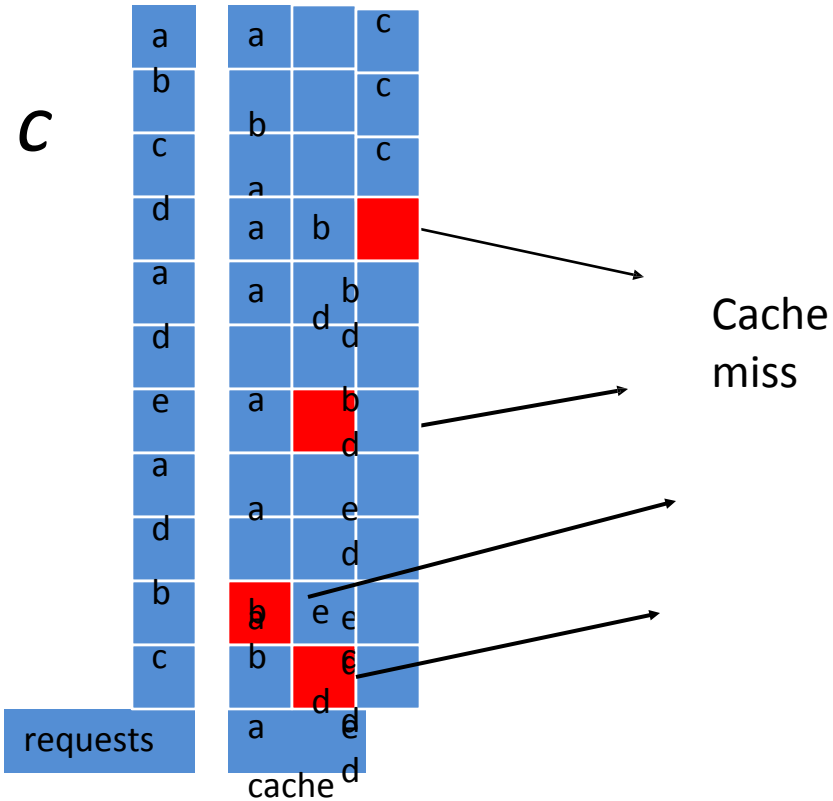
Optimal caching- Algorithm
-Farthest- in-Future

When *di needs to be brought into the cache,*

evict the item that is needed the farthest into the future

Ex: *a, b, c, d, a, d, e, a, d, b, c*

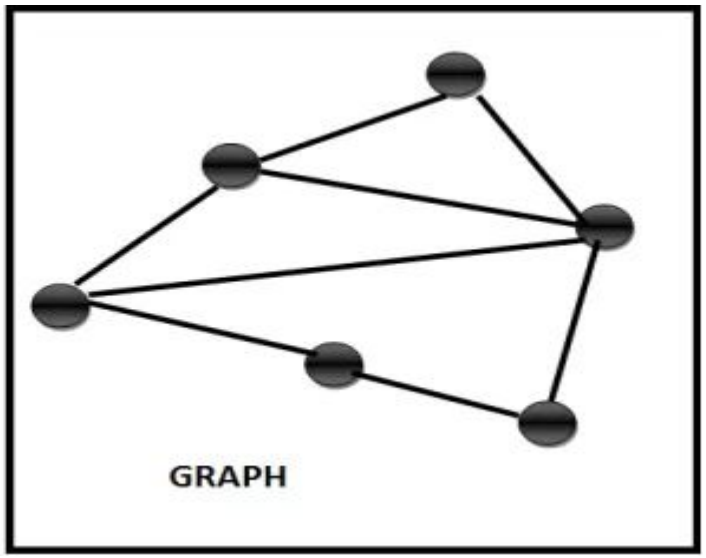
k=3

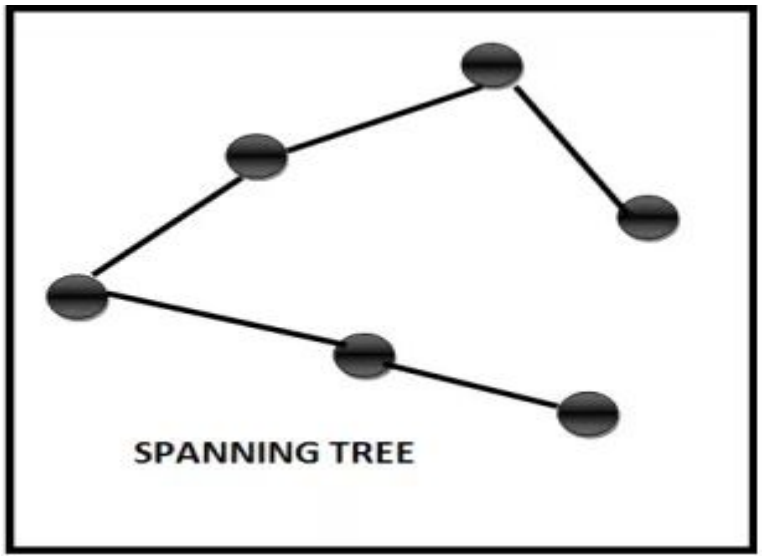


MINIMUM SPANNING TREE PROBLEM

SPANNING TREE

- A spanning tree of an undirected connected graph is its **connected acyclic sub graph** (i.e., a tree) that contains all the vertices of the graph.



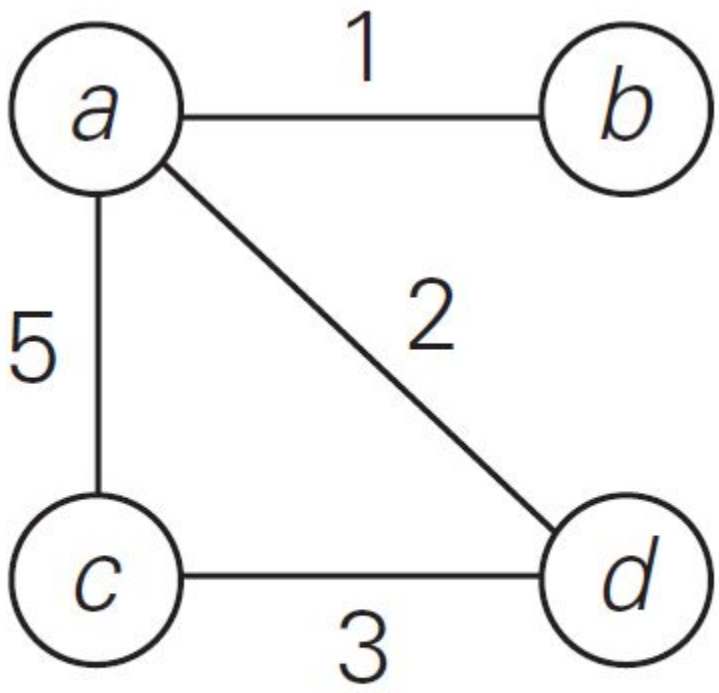


MINIMUM SPANNING TREE

- If such a graph has weights assigned to its edges, a minimum spanning tree is its spanning tree of the smallest weight, where the weight of a tree is defined as the sum of the weights on all its edges

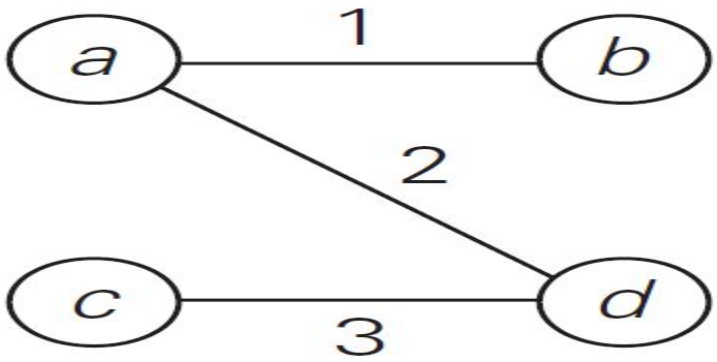
- The minimum spanning tree problem is the problem of finding a minimum spanning tree for a given weighted connected graph.

MINIMUM SPANNING TREE

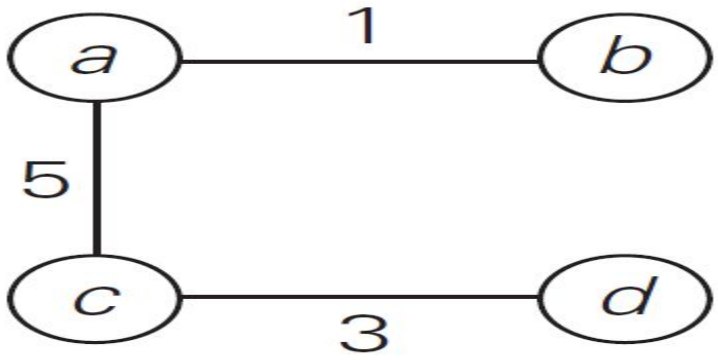


graph

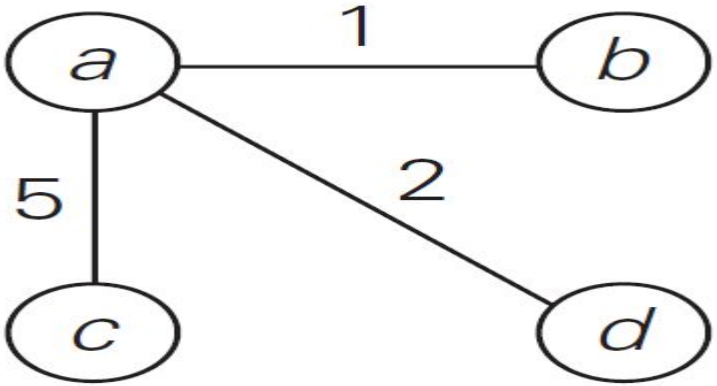
MINIMUM SPANNING TREE



$w(T_1) = 6$



$w(T_2) = 9$



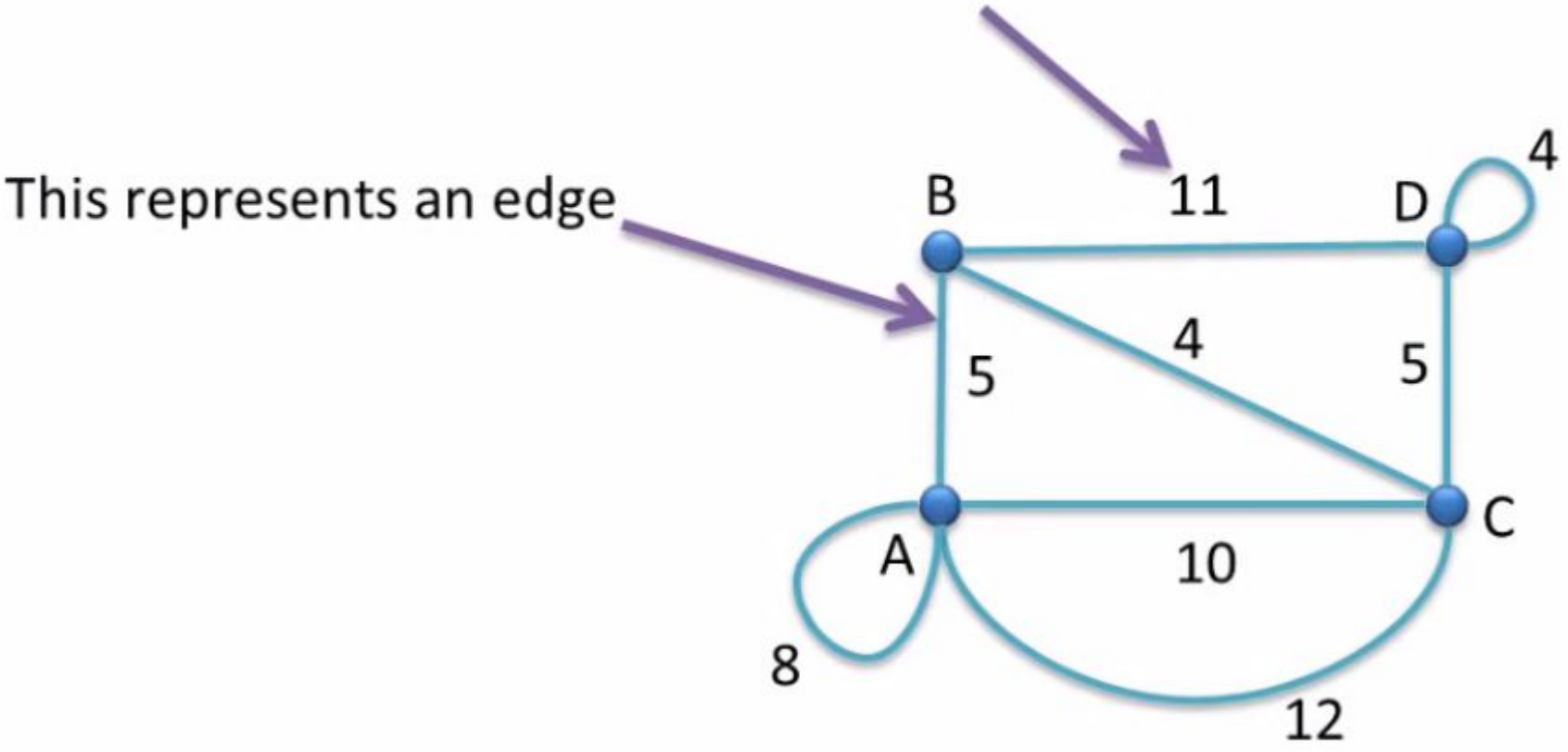
$w(T_3) = 8$

PRIM'S ALGORITHM

Here is our graph

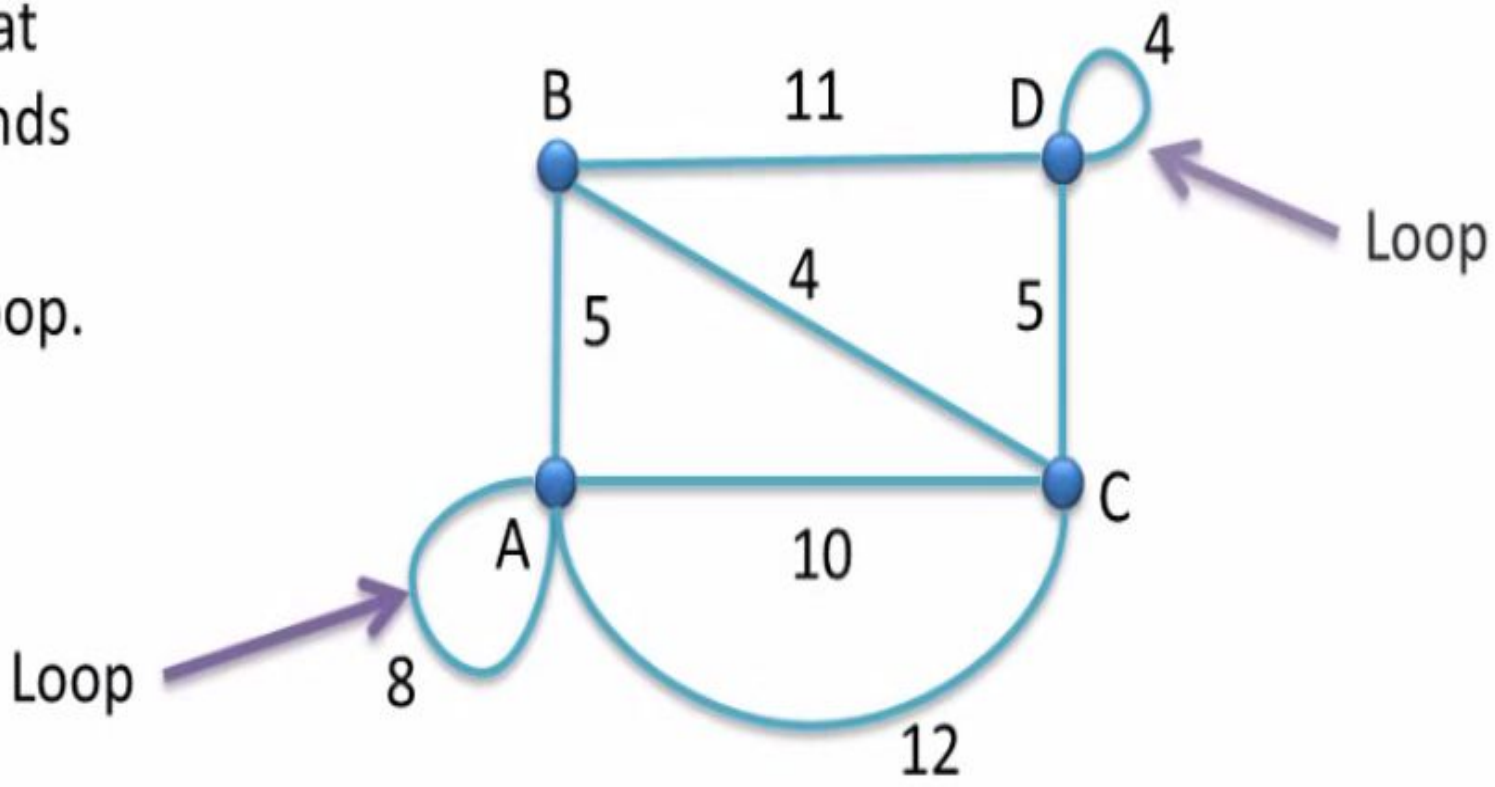
And this represents the weight of the edge

This represents an edge



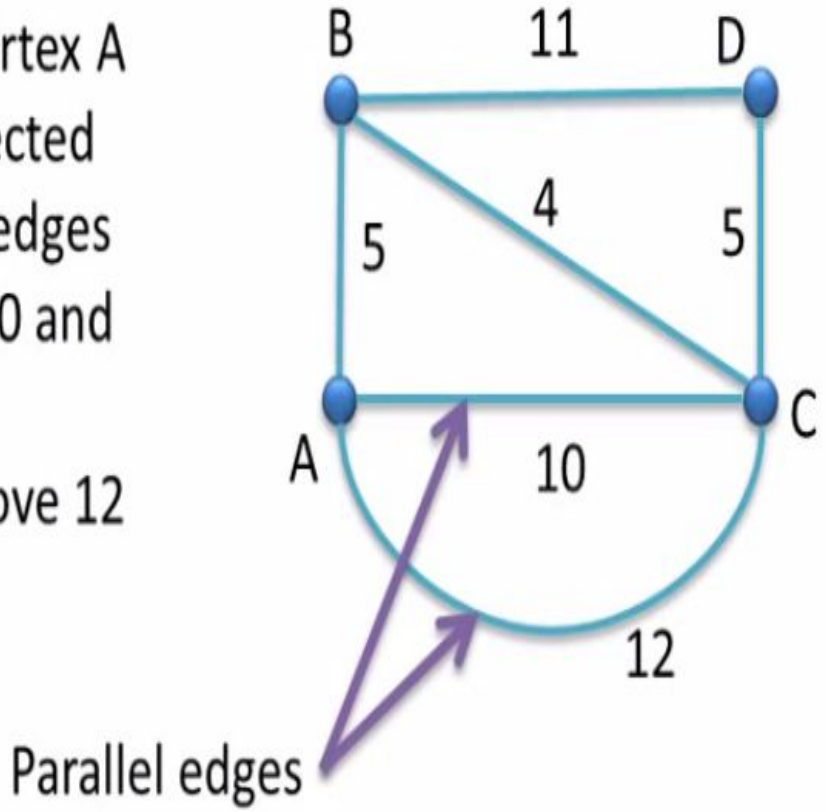
Step 1: Remove all the loops

Note!
Any edge that starts and ends at the same vertex is a loop.



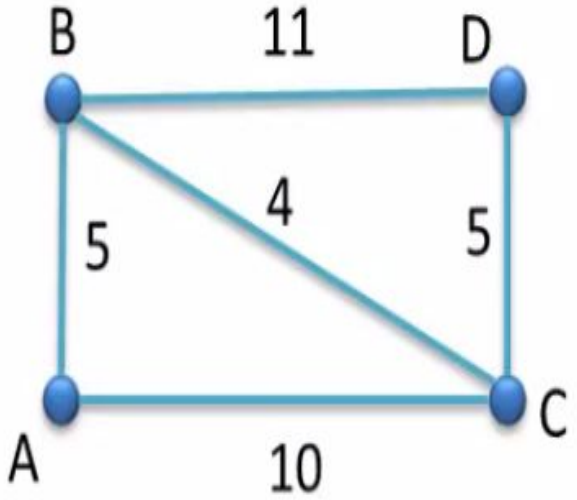
Step 2: Remove all parallel edges between two vertex except the one with least weight

Note!
In this graph, vertex A and C are connected by two parallel edges having weight 10 and 12 respectively.
So, we will remove 12 and keep 10.

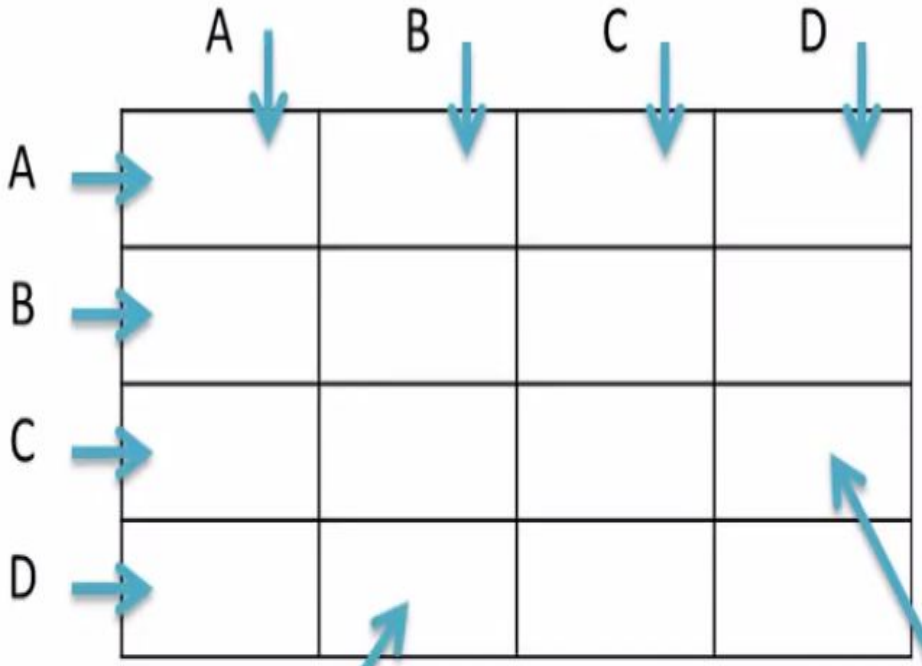


Step 3: Create table

As our graph has 4 vertices, so our table will have 4 rows and 4 columns



These are the columns

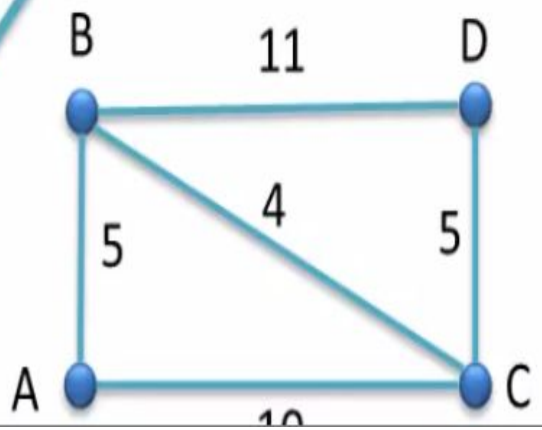


These are the rows

Note!
Row and Column name is same as the name of the vertex.

And this represent a cell CD. Where C is the Row name and D is the column name.

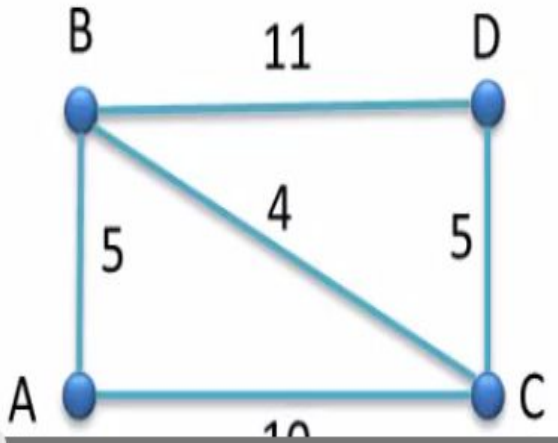
Similarly, this represent a cell DB. Where D is the Row name and B is the column name.



We will now fill the other cells.

	A	B	C	D
A	0			
B		0		
C			0	
D				0

Now, put 0 in cells having same row and column name.

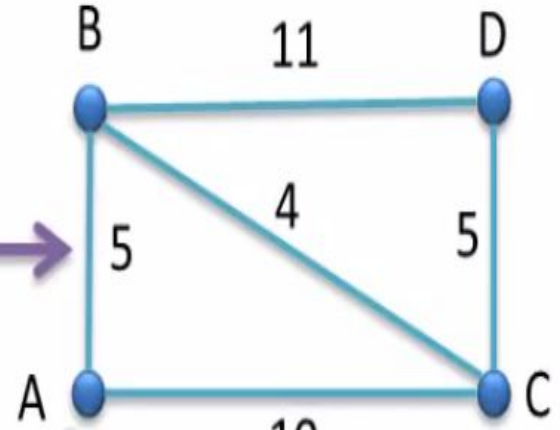


	A	B	C	D
A	0	5		
B	5	0		
C			0	
D				0

So we will write 5 in the cell AB and BA.

Find the edge that **directly** connects vertex A and B.

In this case, we have an edge of weight 5 that directly connect A and B.

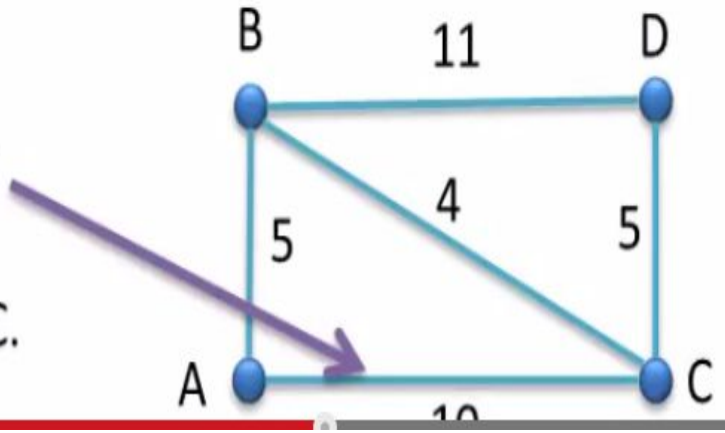


So we will write 10 in the cell AC and CA.

	A	B	C	D
A	0	5	10	
B	5	0		
C	10		0	
D				0

Find the edge that **directly** connects vertex A and C.

In this case, we have an edge of weight 10 that directly connect A and C.



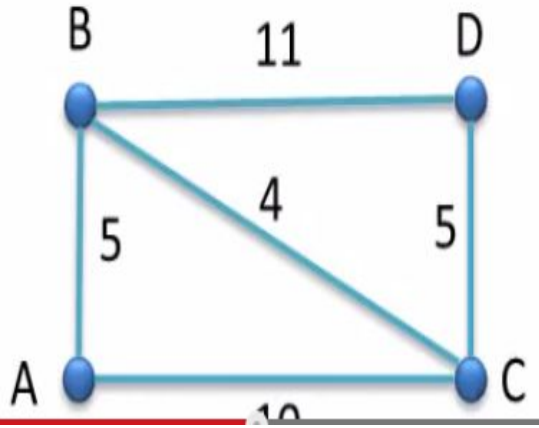
	A	B	C	D
A	0	5	10	∞
B	5	0		
C	10		0	
D	∞			0

So we will write ∞ in the cell AD and DA.

Find the edge that **directly** connects vertex A and D.

∞ denotes **Infinity**.

In this case, we don't have an edge that directly connects A and D

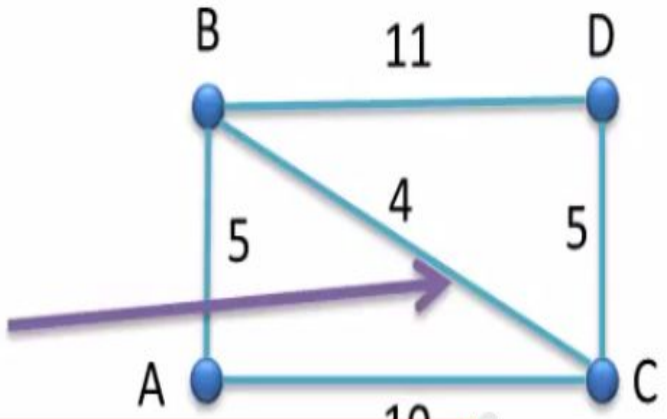


	A	B	C	D
A	0	5	10	∞
B	5	0	4	
C	10	4	0	
D	∞			0

So we will write 4 in the cell BC and CB.

Find the edge that **directly** connects vertex B and C.

In this case, we have an edge of weight 4 that directly connect B and C.

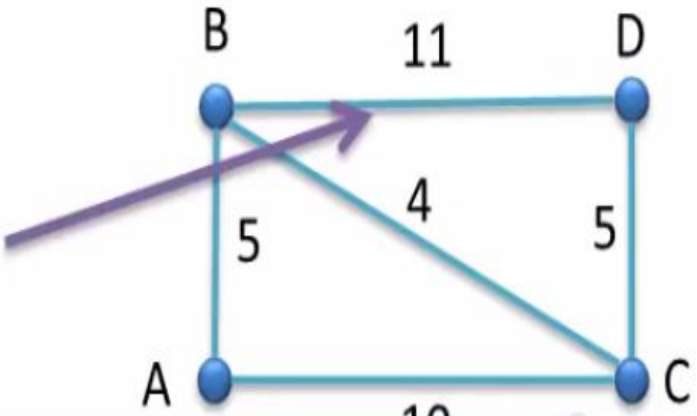


	A	B	C	D
A	0	5	10	∞
B	5	0	4	
C	10	4	0	
D	∞			0

So we will write 11 in the cell BD and DB.

Find the edge that **directly** connects vertex B and D.

In this case, we have an edge of weight 11 that directly connect B and D.

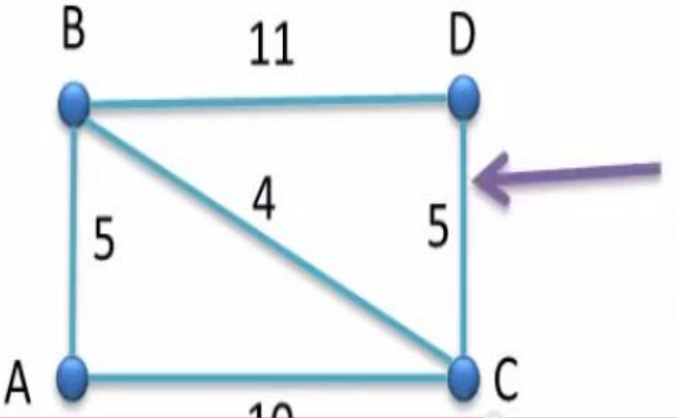


Next

	A	B	C	D
A	0	5	10	∞
B	5	0	4	11
C	10	4	0	5
D	∞	11	5	0

So we will write 5 in the cell CD and DC.

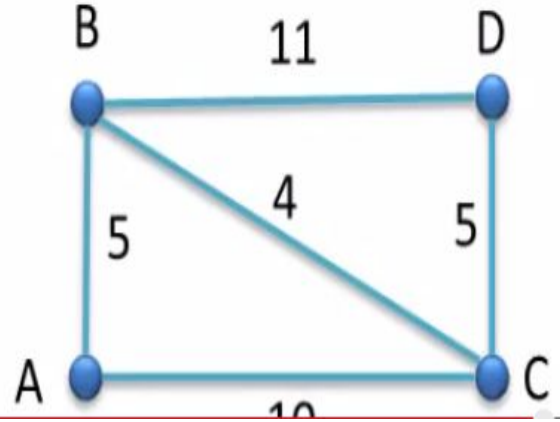
Find the edge that **directly** connects vertex C and D.



In this case, we have an edge of weight 5 that directly connect C and D.

	A	B	C	D
A	0	5	10	∞
B	5	0	4	11
C	10	4	0	5
D	∞	11	5	0

Our table is completely filled, so our next job is to find the MST.



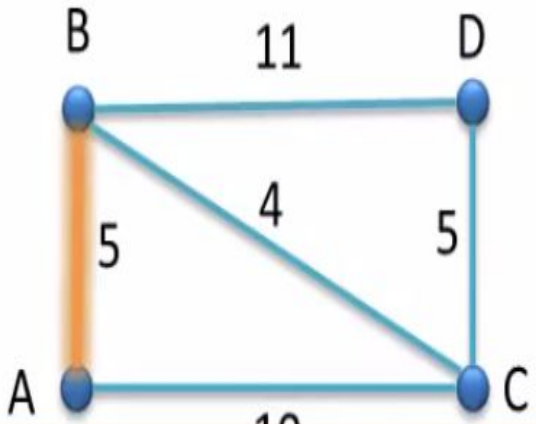
Start from vertex A.

Find the smallest value in the A-row.

Note!
We will not consider 0 as it will correspond to the same vertex.

	A	B	C	D
A	0	5	10	∞
B	5	0	4	11
C	10	4	0	5
D	∞	11	5	0

Smallest value in cell AB



5 is the smallest unmarked value in the A-row.

So, we will mark the edge connecting vertex A and B

Tick 5 in AB and BA cell.

As we connected vertex A and B in the previous step, so we will now find the smallest value in the A-row and B-row.

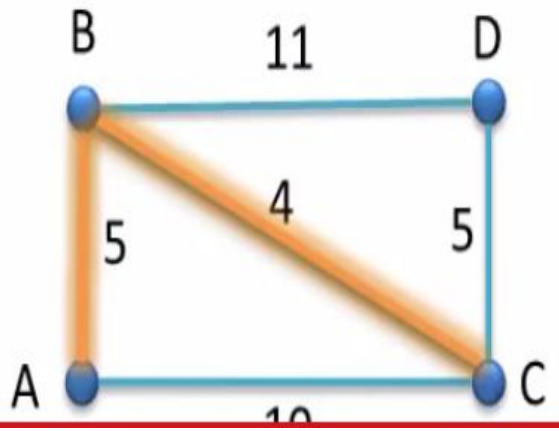
	A	B	C	D
A	0	5✓	10	∞
B	5✓	0	4	11
C	10	4	0	5
D	∞	11	5	0

Smallest value in cell BC

4 is the smallest unmarked value in the A-row and B-row.

So, we will mark the edge connecting vertex B and C

Note!
We will not consider 0 as it will correspond to the same vertex.



As vertex A-B and B-C were connected in the previous steps, so we will now find the smallest value in A-row, B-row and C-row.

	A	B	C	D
A →	0	5✓	10	∞
B →	5✓	0	4✓	11
C →	10	4✓	0	5✓
D	∞	11	5✓	0

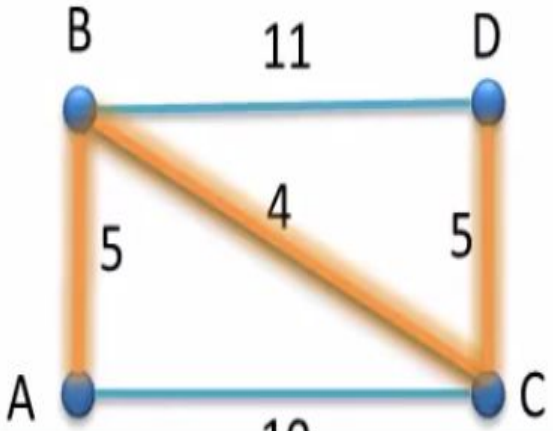
Smallest value in cell CD

5 is the smallest unmarked value in the A-row, B-row and C-row.

So, we will mark the edge connecting vertex C and D

Tick 5 in CD and DC cell.

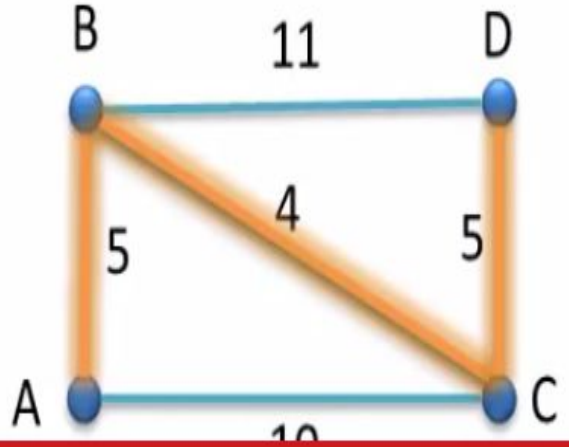
Note!
We will not consider 0 as it will correspond to the same vertex.



	A	B	C	D
A →	0	5✓	10	∞
B →	5✓	0	4✓	11
C →	10	4✓	0	5✓
D	∞	11	5✓	0

Note!
A spanning tree with 4 vertices will have 3 edges.

As we have marked all the 4 vertices, so we will stop here.

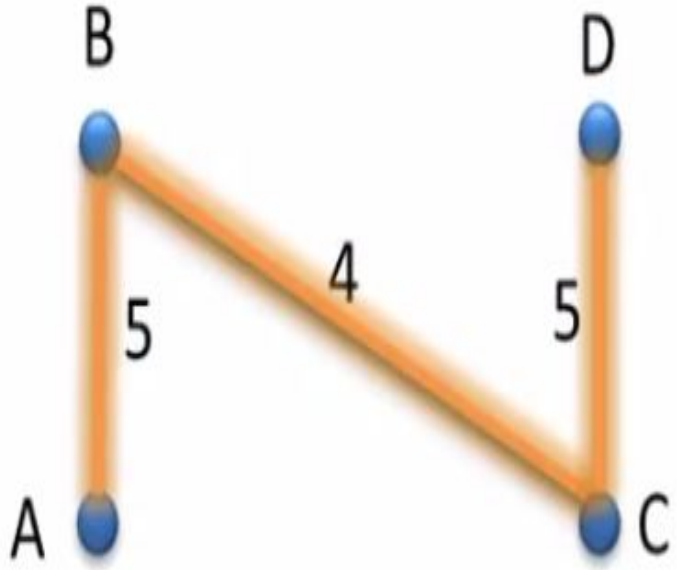


Our required Minimum Spanning Tree (MST) is

Weight of the MST

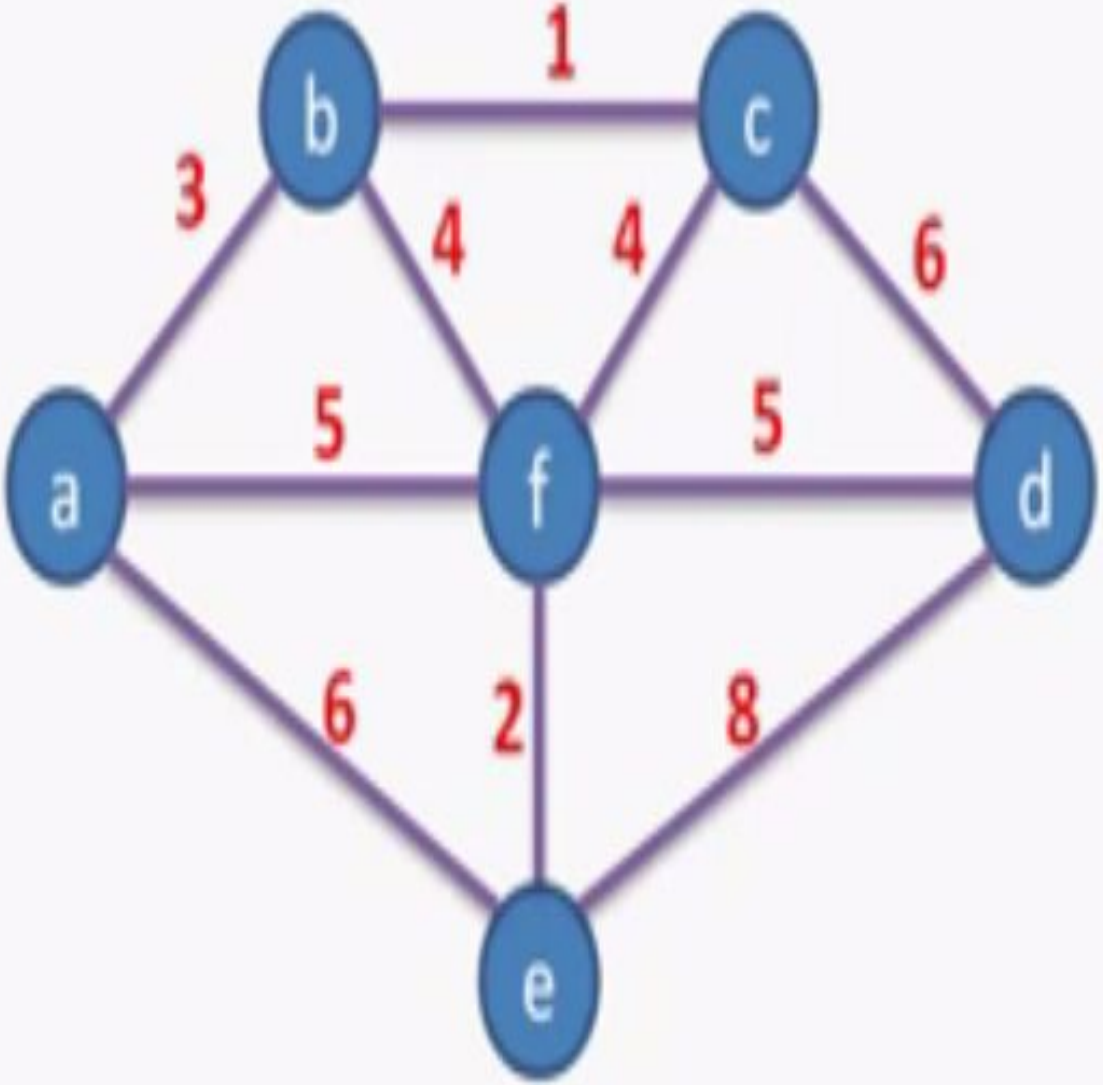
= 5+4+5

= 14 unit

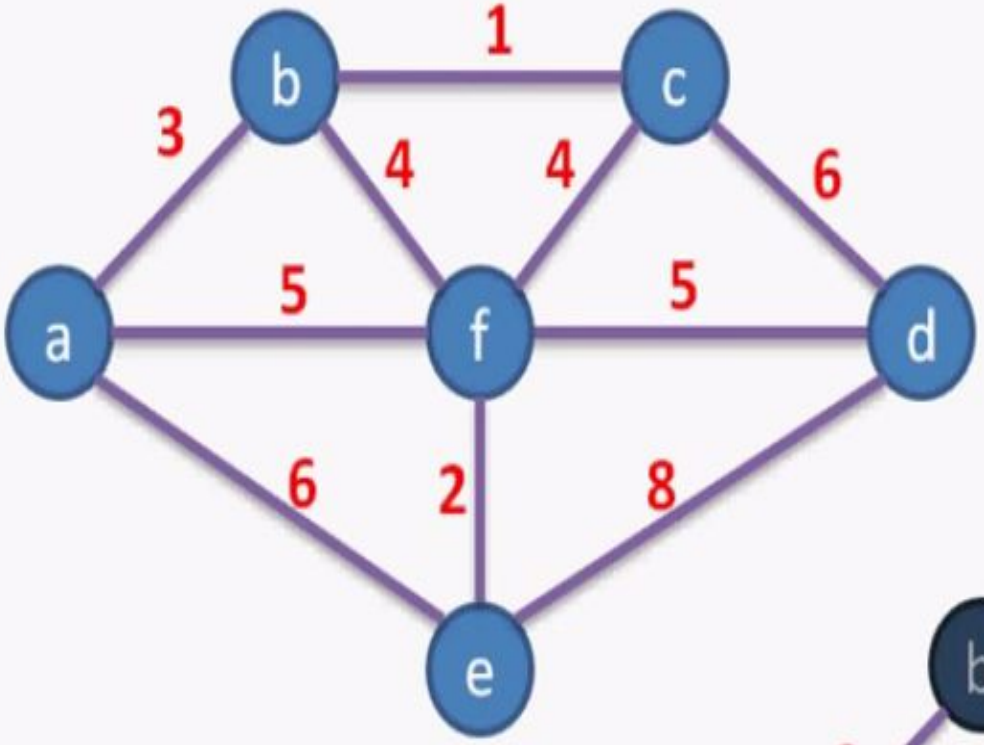


PRIM'S ALGORITHM

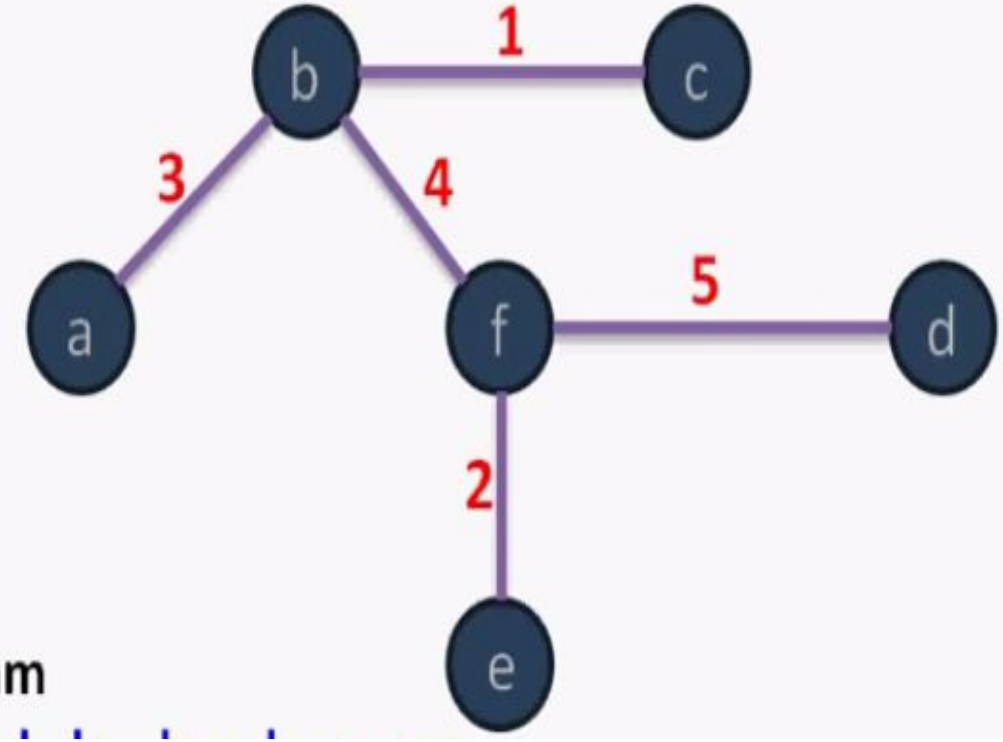
Prim's Algorithm (G, c)
Initially $S = \{s\}$ and $T = \emptyset$
While $S \neq V$
  Select a node $v \notin S$ which has an edge into S and
    for which the attachment cost $\min_{e=(u,v):u \in S} c_e$ is
      as small as possible
  Add v to S
  Add the edge e where the minimum is obtained to T
EndWhile
Return the spanning tree T



Prims Algorithm Example



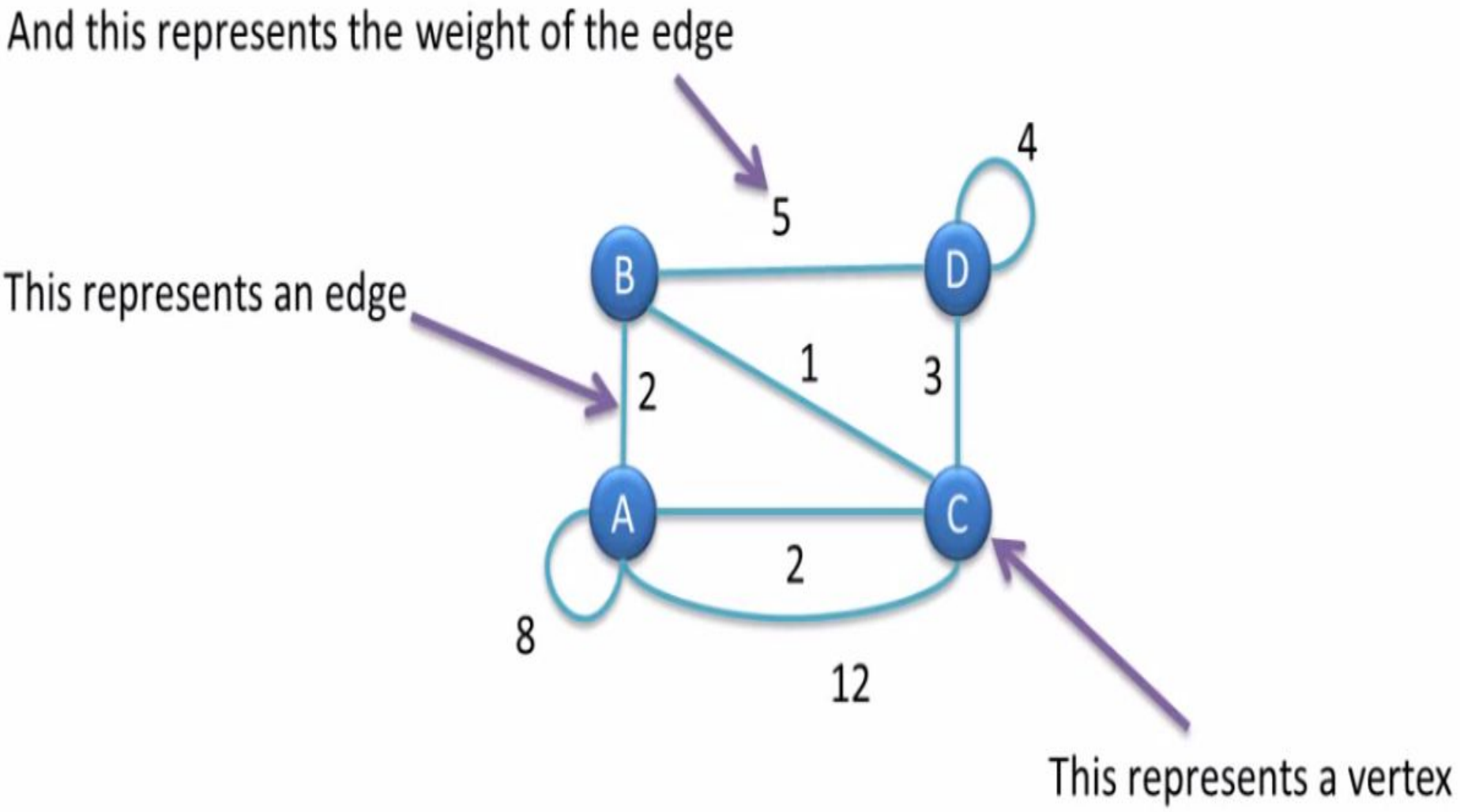
Prims Algorithm
Example



Prims Algorithm

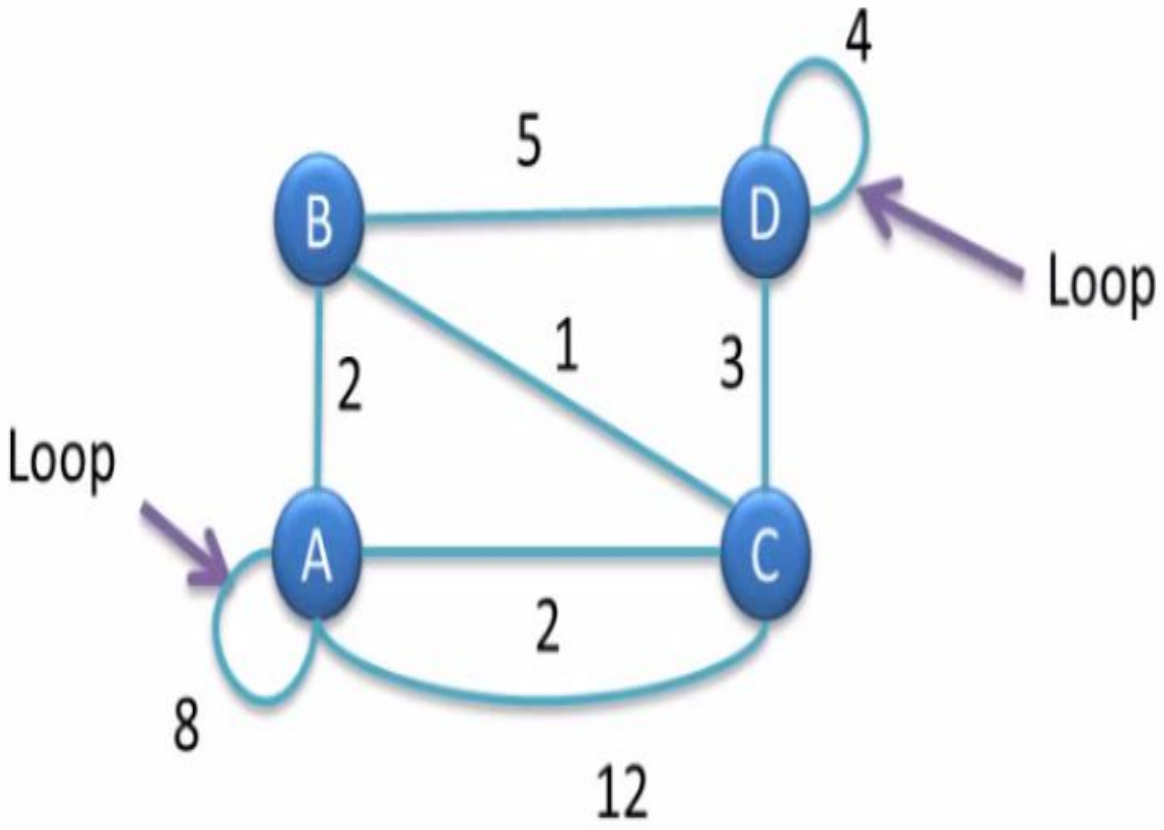
KRUSKAL'S ALGORITHM

Here is our graph

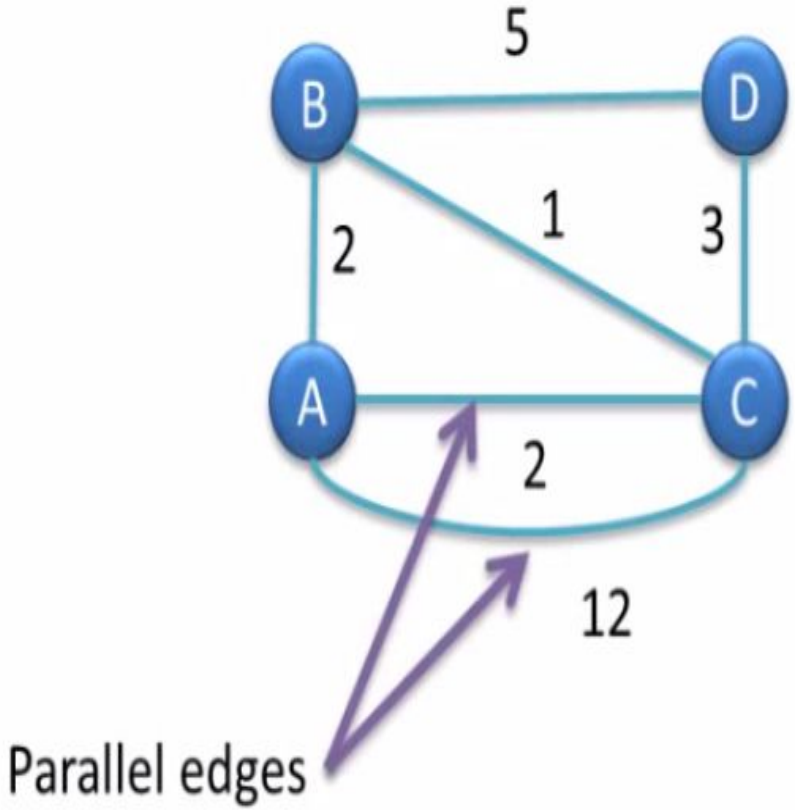


Step 1: Remove all the loops

Note!
Any edge that starts and ends at the same vertex is a loop.

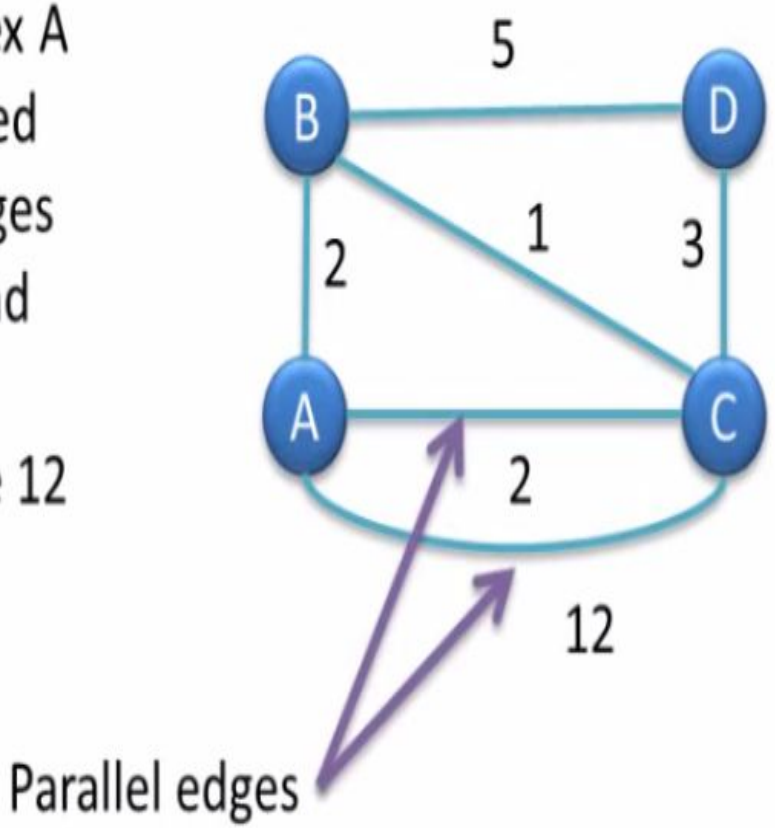


Step 2: Remove all parallel edges between two vertex except the one with least weight



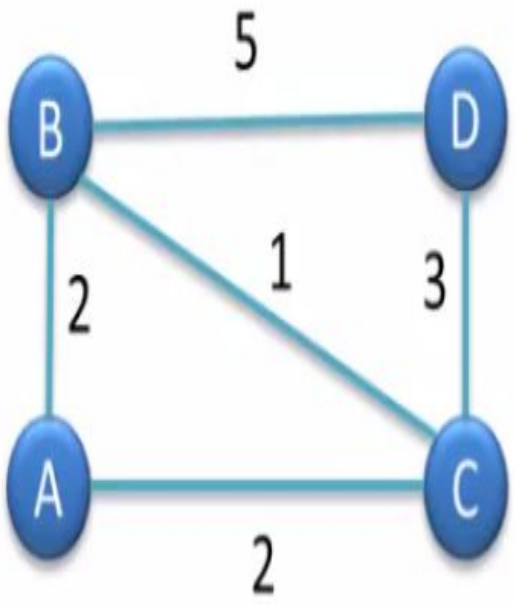
Step 2: Remove all parallel edges between two vertex except the one with least weight

Note!
In this graph, vertex A and C are connected by two parallel edges having weight 2 and 12 respectively.
So, we will remove 12 and keep 2.



Step 3: Create the edge table

An edge table will have name of all the edges along with their weight is ascending order.



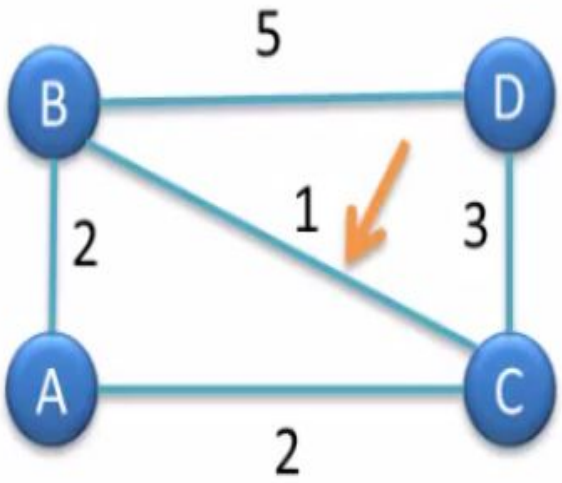
Now if you look at the graph, then you will notice that there are total 5 edges.

So our edge table will have 5 columns.

Edge					
Weight					

Now look at the graph and fill the 1st column with the edge of minimum weight.

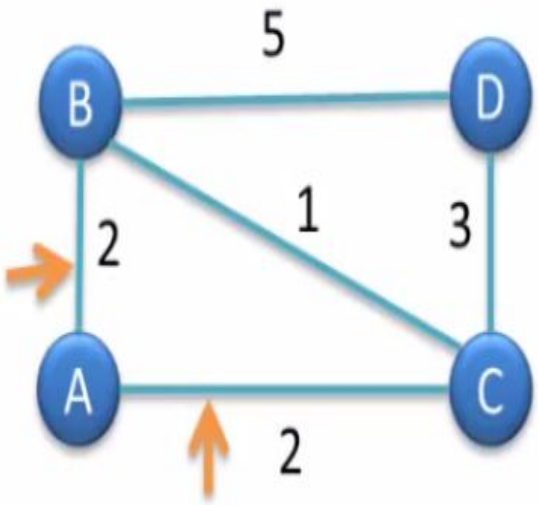
In this case edge BC is of least weight so we will select it.



Edge	BC				
Weight	1				

Now look at the graph again and find the edge of next minimum weight.

In this case edge AB and edge AC are having the minimum weight so we will select them both.

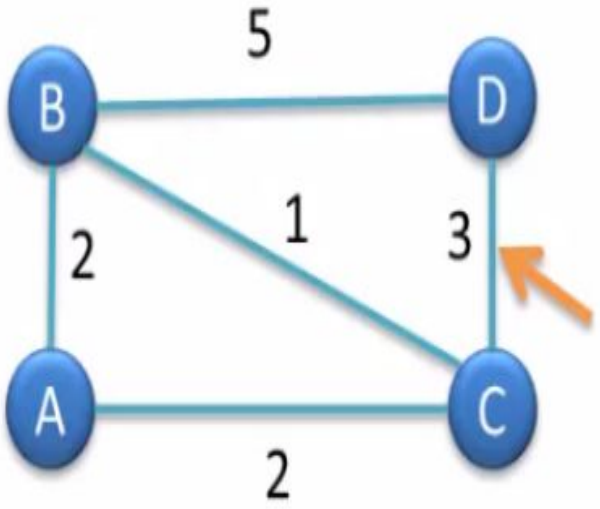


Note!
In case we have two or more edges with same weight then we can write them in any order in the edge table.

Edge	BC	AB	AC		
Weight	1	2	2		

Now look at the graph again and find the edge of next minimum weight.

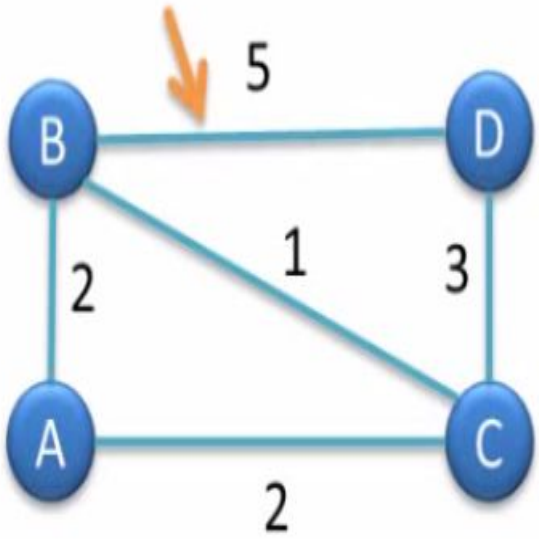
In this case edge DC has the minimum weight so we will select it.



Edge	BC	AB	AC	DC	
Weight	1	2	2	3	

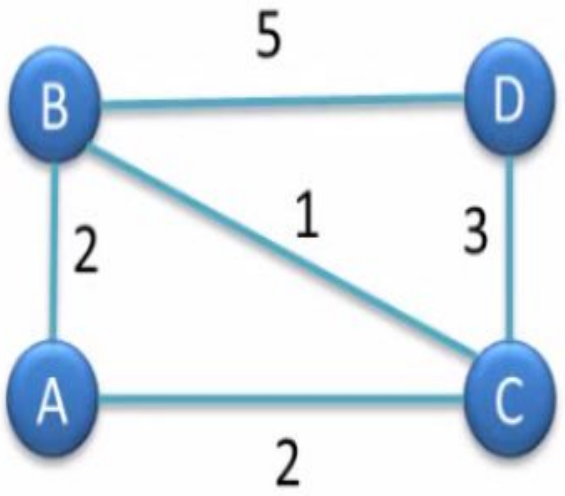
Now look at the graph again and find the edge of next minimum weight.

In this case edge BD has the minimum weight so we will select it.



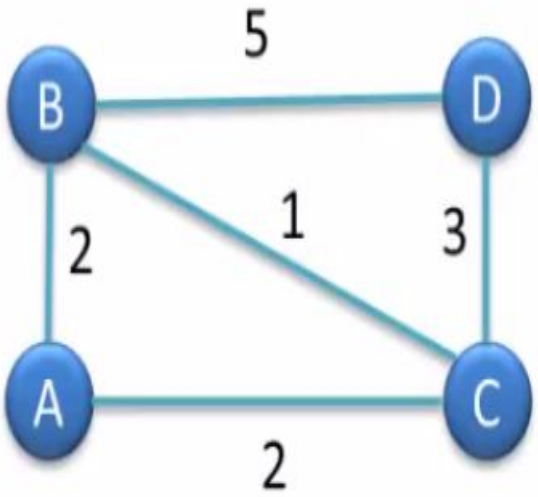
Edge	BC	AB	AC	DC	BD
Weight	1	2	2	3	5

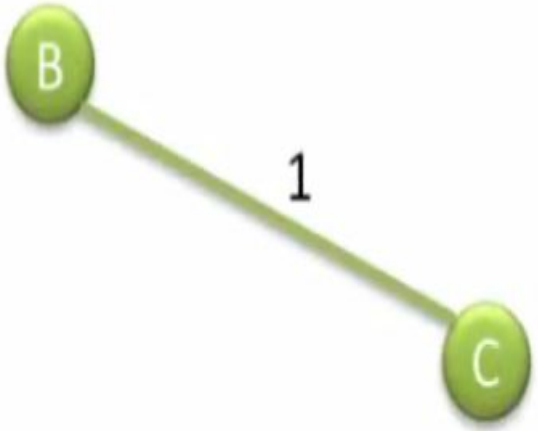
Time to find the minimum spanning tree.



Edge	BC	AB	AC	DC	BD
Weight	1	2	2	3	5

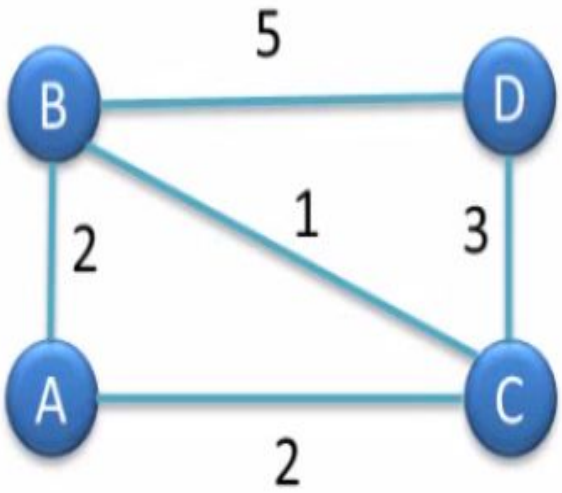
1 is the smallest weight so we will select edge BC

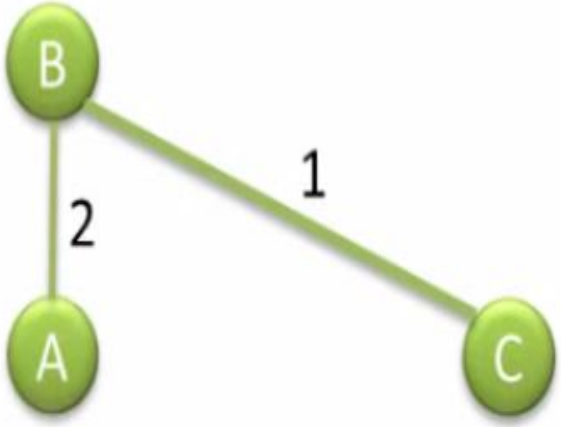




Edge	BC	AB	AC	DC	BD
Weight	1	2	2	3	5

2 is the next smallest weight and edge AB does not form a circuit with the previously selected edges so we will select it.



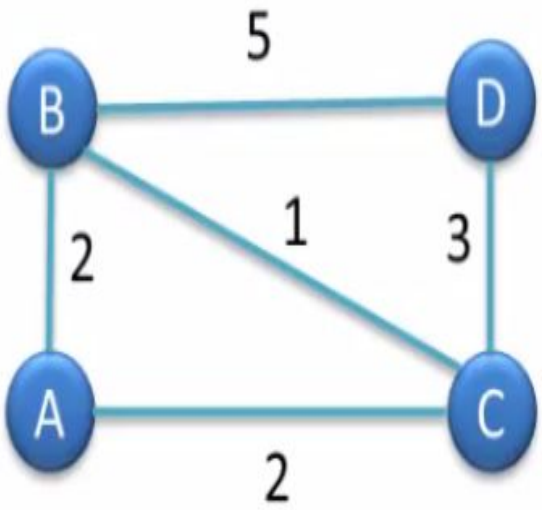


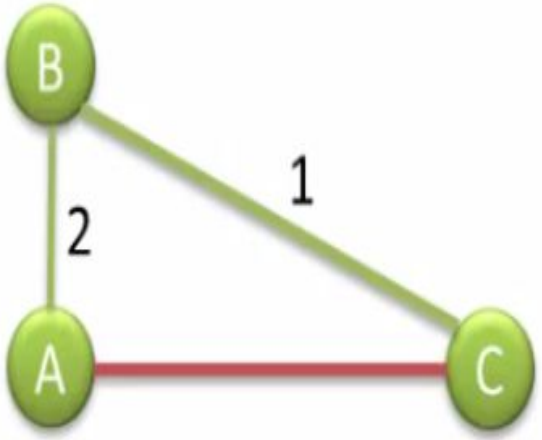
Edge	BC	AB	AC	DC	BD
Weight	1	2	2	3	5

2 is the next smallest weight.

But if we select edge AC then it will form a circuit.

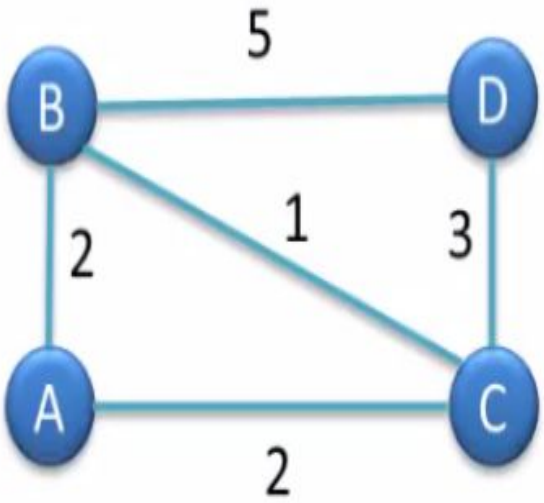
So we are going to reject edge AC.

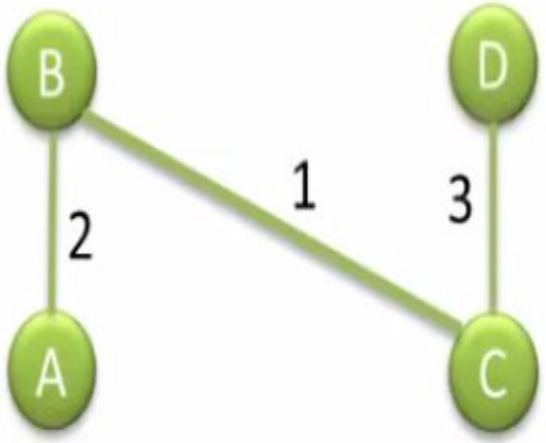




Edge	BC	AB	AC	DC	BD
Weight	1	2	2	3	5

3 is the next smallest weight and edge DC does not form a circuit with the previously selected edges. So we will select it.



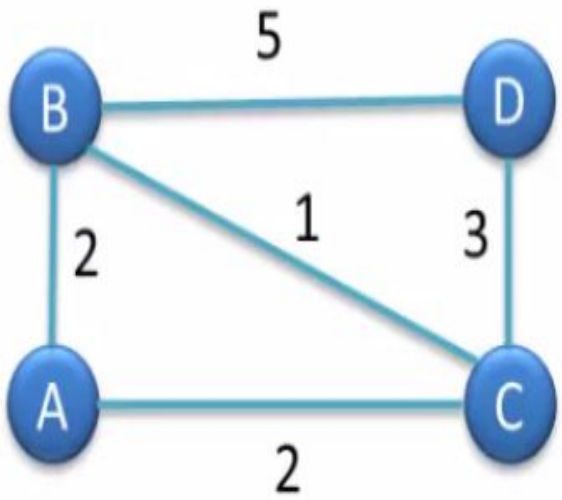


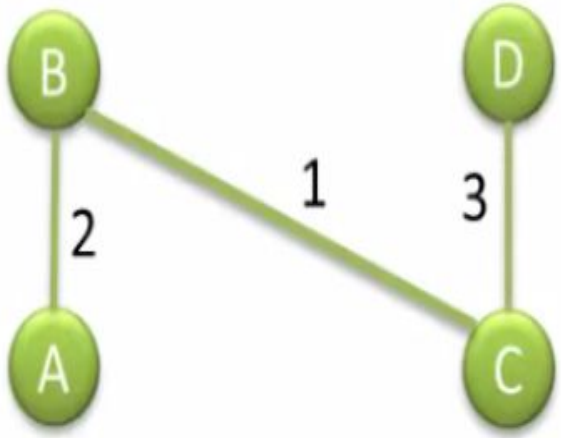
Edge	BC	AB	AC	DC	BD
Weight	1	2	2	3	5

Since we have got the 3 edges so we will stop here.

Note!

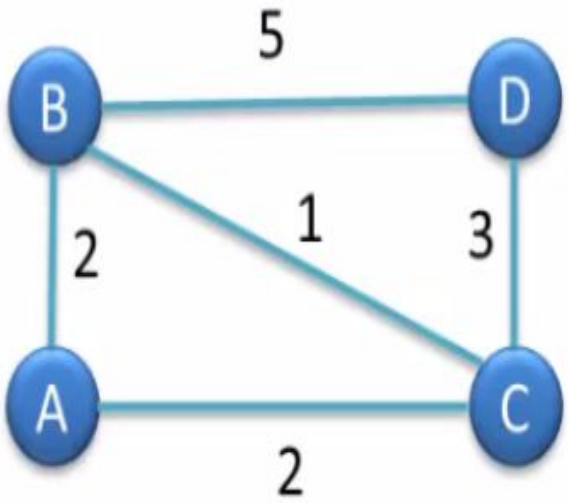
Our graph has 4 vertices so, our MST will have 3 edges.

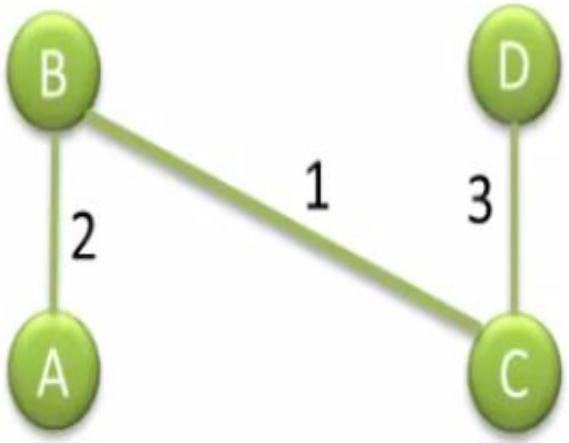




Edge	BC	AB	AC	DC	BD
Weight	1	2	2	3	5

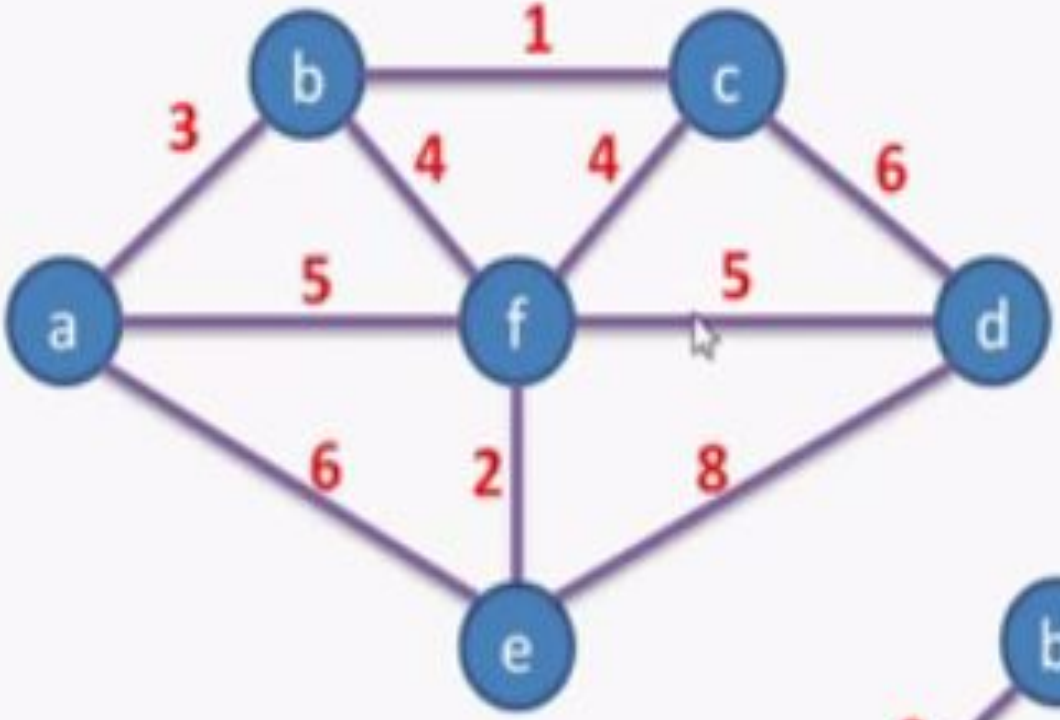
Therefore, our required Minimum Spanning Tree is



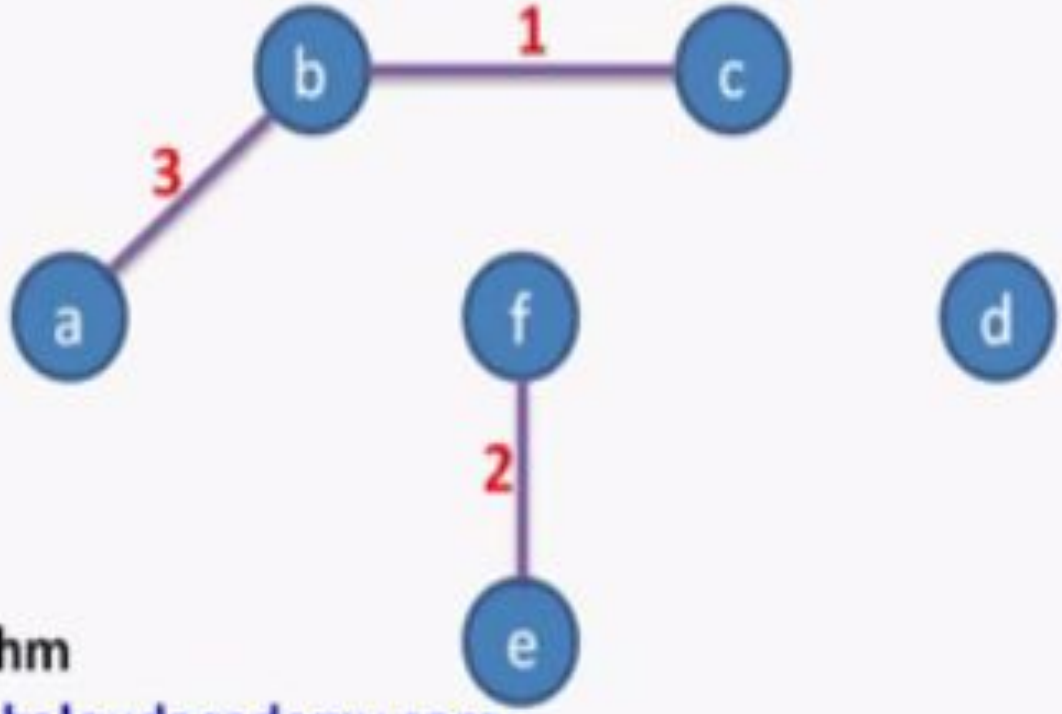


KRUSKAL'S ALGORITHM

Kruskal's Algorithm (G, c)
Sort the edges in order of increasing cost.
Initially $T = \emptyset$
For each edge $e = (v, w)$ in the sorted order
    If there is currently no path from v to w in (V, T) then
        (Adding e won't create a cycle)
        Add e to T.
    EndIf
EndFor
Return the set of edges T



Kruskal's Algorithm
Example



Kruskal's Algorithm

Assume that all the edges are Sorted in ascending order of weights

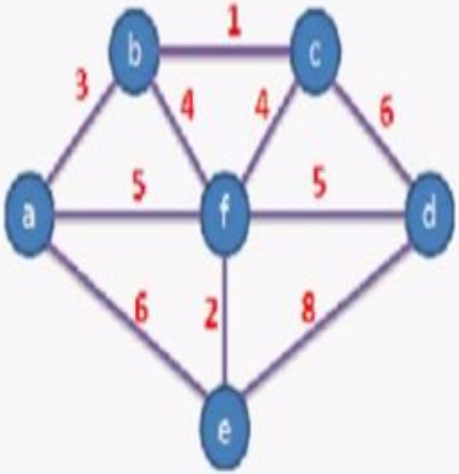
Et Array of edges which is our solution

To keep track of how many edges are there in our edge set.

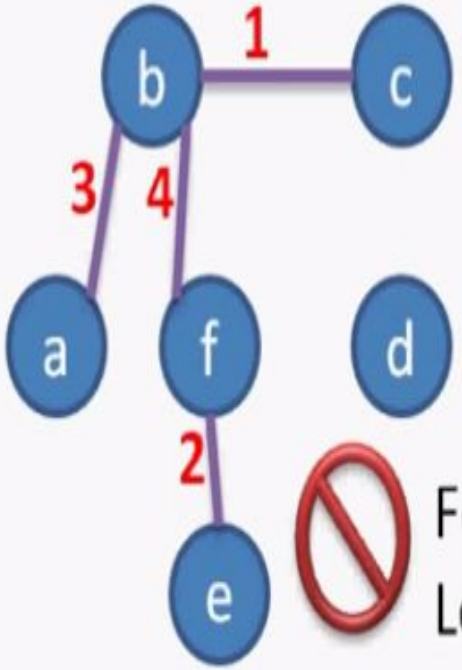
Till we connect all the vertices with edges go ahead

```
Algorithm Kruskal (G)
    Sort E in ascending order of weights
    Et ← 0 //no edges selected
    encounter ← 0 //no of edges selected
    k ← 0
    while encounter < |V| - 1
        k ← k + 1
        if Et U {eik} is acyclic
            Et ← Et U {eik}
            encounter += 1
    return Et
```

Kruskal's Algorithm



Edge	Lengths
b→c	1
f→e	2
a→b	3
b→f	4
c→f	4
a→f	5
d?f	5
a→e	6
c→d	6
e→d	8



Forms Loop !!!

Algorithm Kruskal (G)

Sort E in ascending order of weights

Et ← 0 //no edges selected

encounter ← 0 //no of edges selected

k ← 0

while encounter < |V| - 1

    k ← k + 1

    if Et U {eik} is acyclic

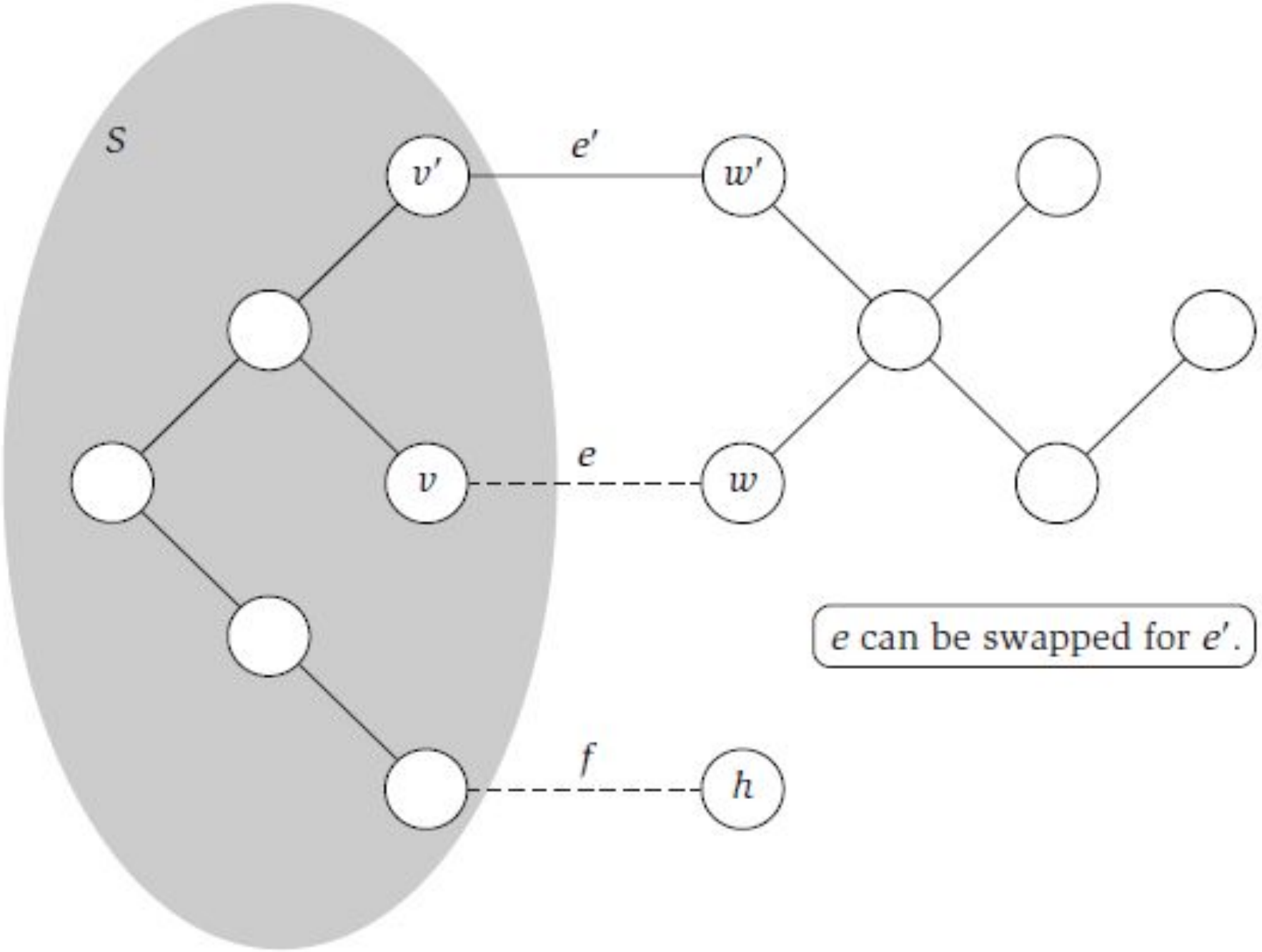
        Et ← Et U {eik}

        encounter += 1

return Et

Kruskal's Algorithm

ANALYSIS


S
v'
e'
w'
v
e
w
f
h
e can be swapped for e'.

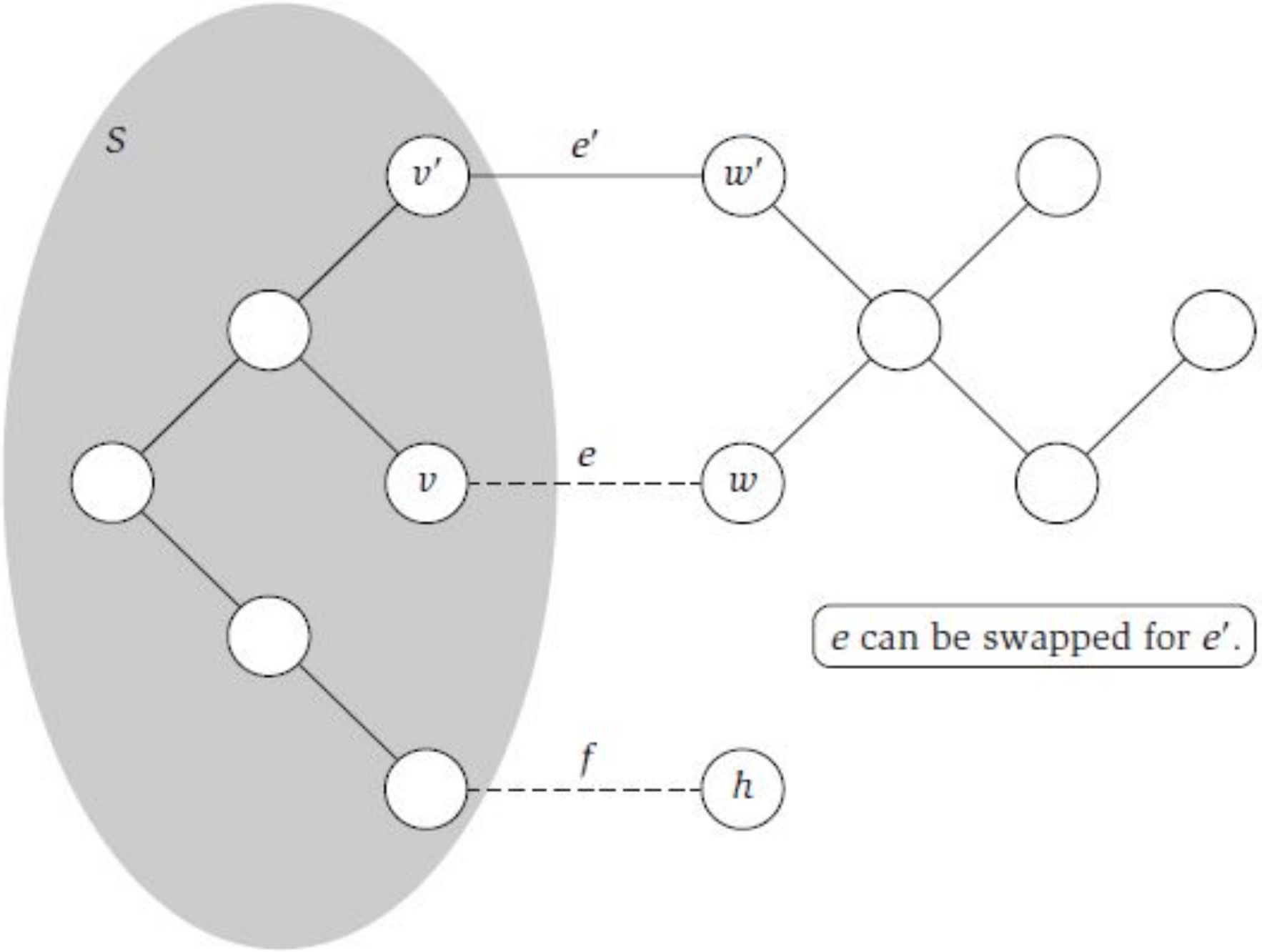
- ***Let T be a minimum-cost solution to the network design problem defined above. Then (V, T) is a tree.***

- **Proof.** By definition, (V, T) must be connected.
- We show that it also will contain no cycles. Indeed, suppose it contained a cycle C.

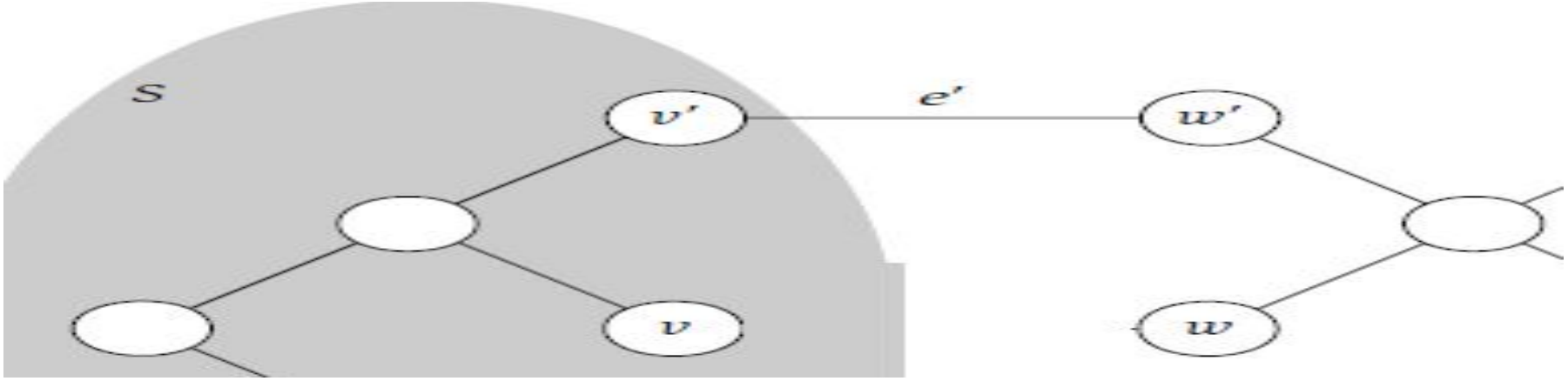
- Let e be any edge on C. We claim that $(V, T - \{e\})$ is still connected.
-
- Since any path that previously used the edge e can now go "the long way" around the remainder of the cycle C instead.

- It follows that $(V, T - \{e\})$ is also a valid solution to the problem, and it is cheaper—a contradiction.

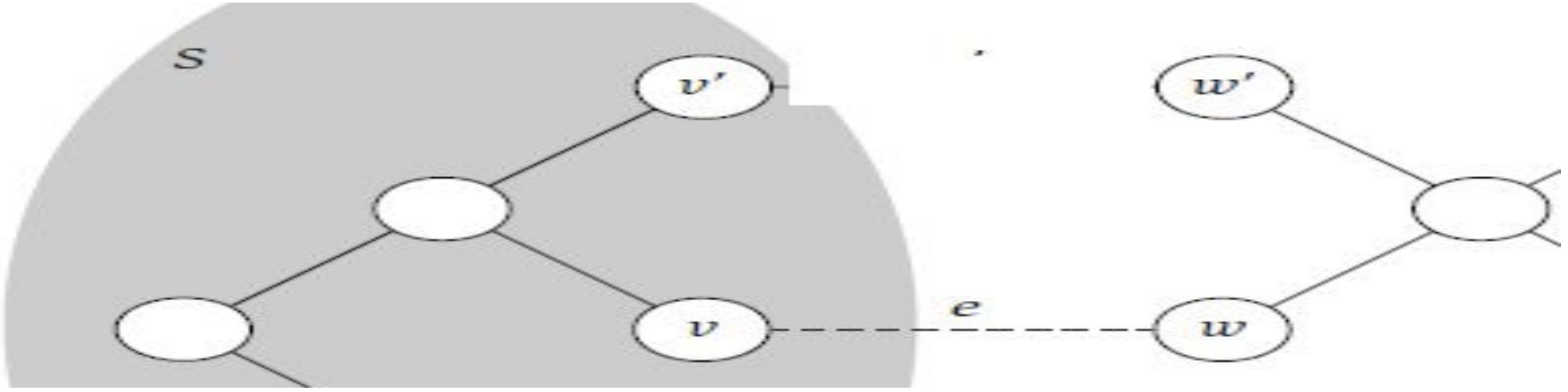
- ***Assume that all edge costs are distinct. Let S be any subset of nodes that is neither empty nor equal to all of V, and let edge e = (v, w) be the minimum cost edge with one end in S and the other in V – S. Then every minimum spanning tree contains the edge e.***


S
v'
e'
w'
v
e
w
f
h
e can be swapped for e'.

- Let *T* be a spanning tree that does not contain *e*; we need to show that *T* does not have the minimum possible cost.



- we'll identify an edge *e'* in *T* that is more expensive than *e*, and with the property exchanging *e* for *e'* results in another spanning tree T'. This resulting spanning tree will then be cheaper than *T.*



- the ends of *e* are *v* and *w*. *T* is a spanning tree, so there must be a path *P* in *T* from *v* to *w*.
- Starting at *v*, suppose we follow the nodes of *P* in sequence; there is a first node *w'* on *P* that is in *V* – *S*.
- Let v' € S be the node just before w' on P, let e'=(v',w') be the edge joining them.

- Let *e* = *(v, w)* be the edge in P.
- *e* is an edge of *T* with one end in *S* and the other in *V* – *S.*
- If we exchange *e* for *e'*, we get a set of edges

  $T' = T - \{e'\} \cup \{e\}$.
- We claim that *T'* is a spanning tree.

- Clearly *(V, T')* is connected, since *(V, T)* is connected, and any path in *(V, T)* that used the edge *e' = (v', w')* can now be "rerouted" in *(V, T')* to follow the portion of *P* from *v'* to *v*,

- then the edge *e*, and then the portion of *P* from *w* to *w'*. To see that *(V, T')* is also acyclic.

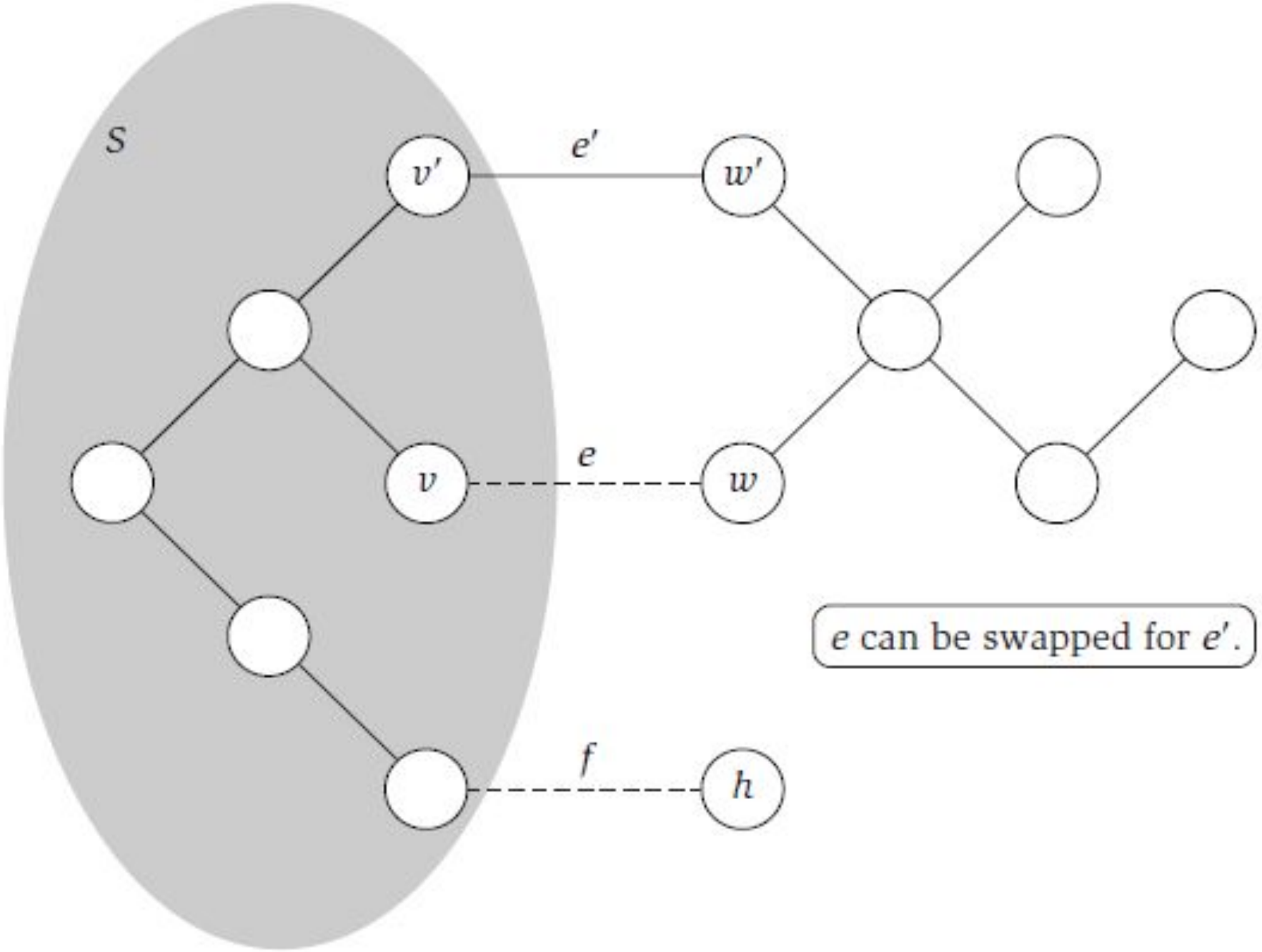
- note that the only cycle in *(V, T* $\cup$ *{e'})* is the one composed of *e* and the path P

- and this cycle is not present in *(V, T')* due to the deletion of *e'*.

- We noted above that the edge *e'* has one end in *S* and the other in *V* – *S*.

- But *e* is the cheapest edge with this property, and so *ce* < *ce'*. (The inequality is strict since no two edges have the same cost.)

- Thus the total cost of *T'* is less than that of *T*, as desired.


S
v'
e'
w'
v
e
w
f
h
e can be swapped for e'.

- ***Kruskal's Algorithm produces a minimum spanning tree of G.***

Proof.

- Consider any edge $e = (v, w)$ added by Kruskal's Algorithm, and let

- S be the set of all nodes to which v has a path at the moment just before e is added.
- Clearly $v \in S$, but $w \notin S$, since adding e does not create a cycle.

- Moreover, no edge from S to $V - S$ has been encountered yet, since any such edge could have been added without creating a cycle, and hence would have been added by Kruskal's Algorithm.

- Thus e is the cheapest edge with one end in S and the other in $V - S$, and so by (4.17) it belongs to every minimum spanning tree.

- So if we can show that the output *(V, T)* of Kruskal's Algorithm is in fact a spanning tree of *G*, then we will be done.

- Clearly *(V, T)* contains no cycles, since the algorithm is explicitly designed to avoid creating cycles.

- Further, if *(V, T)* were not connected, then there would exist a nonempty subset of nodes *S* such that there is no edge from *S* to $V - S$.
- But this contradicts the behavior of the algorithm: we know that since *G* is connected, there is at least one edge between *S* and $V - S$, and the algorithm will add the first of these that it encounters.

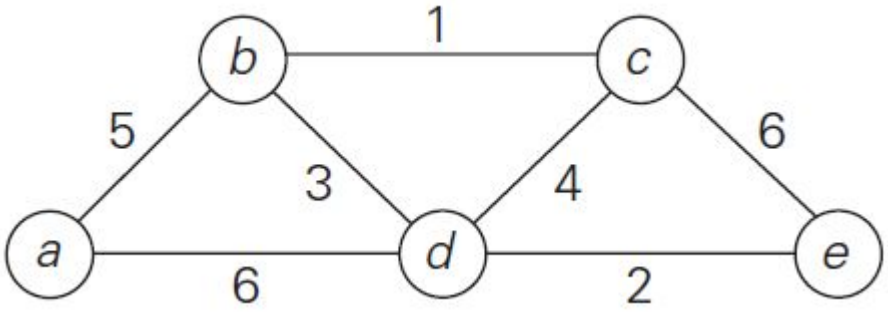
- ***Prim's Algorithm produces a minimum spanning tree of G.***

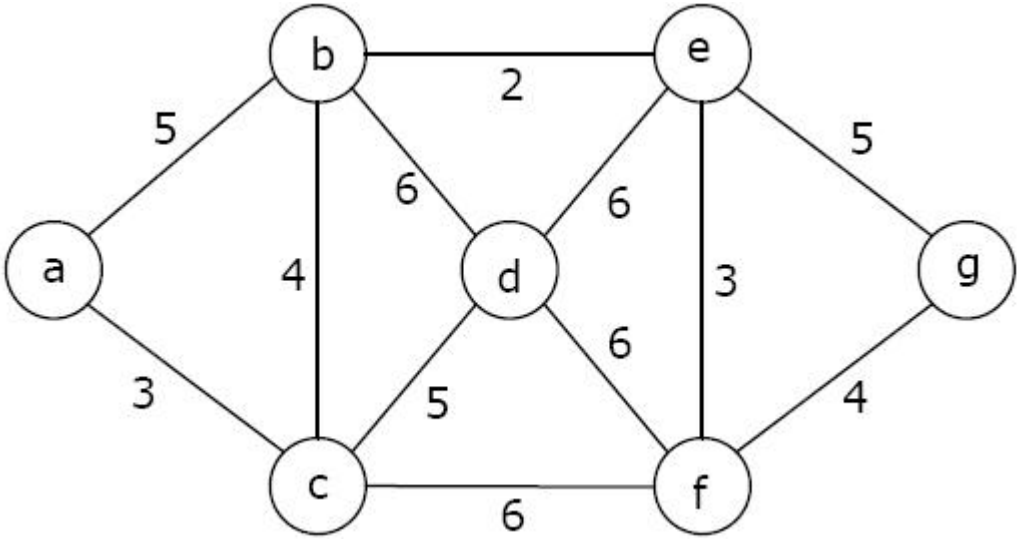
Proof.

- For Prim's Algorithm, it is also very easy to show that it only adds edges belonging to every minimum spanning tree.
- Indeed, in each iteration of the algorithm, there is a set $S \subseteq V$ on which a partial spanning tree has been constructed, and a node v and edge e are added that minimize the quantity
- $\min e=(u,v): u \in S\ c_e$.
- By definition, e is the cheapest edge with one end in S and the other end in $V - S$, and so by the Cut Property (4.17) it is in every minimum spanning tree.

- It is also straightforward to show that Prim's Algorithm produces a spanning tree of G, and hence it produces a minimum spanning tree.

APPLY PRIM'S & KRUSKAL'S

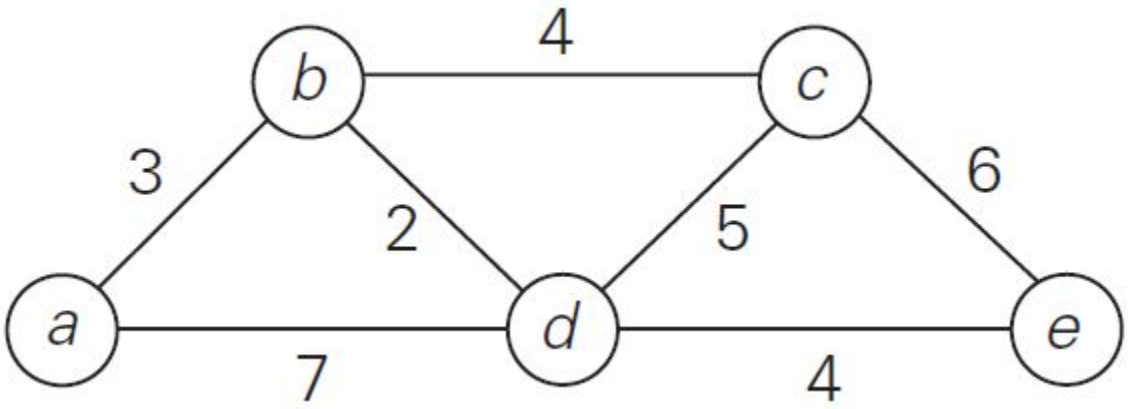




DIJKSTRA ALGORITHM

INTRODUCTION

- Single - Source Shortest - Paths Problem
- For a given vertex called the **Source** in a **weighted connected graph**, find Shortest Paths to all its other vertices.



INTRODUCTION

- The single-source shortest-paths problem asks for a family of paths, each leading from the source to a different vertex in the graph, though some paths may, of course, have edges in common.

DIJKSTRA ALGORITHM

Dijkstra's Algorithm (G, ℓ)

Let S be the set of explored nodes.

For each $u \in S$, we store a distance $d(u)$.

Initially $S = \{s\}$ and $d(s) = 0$.

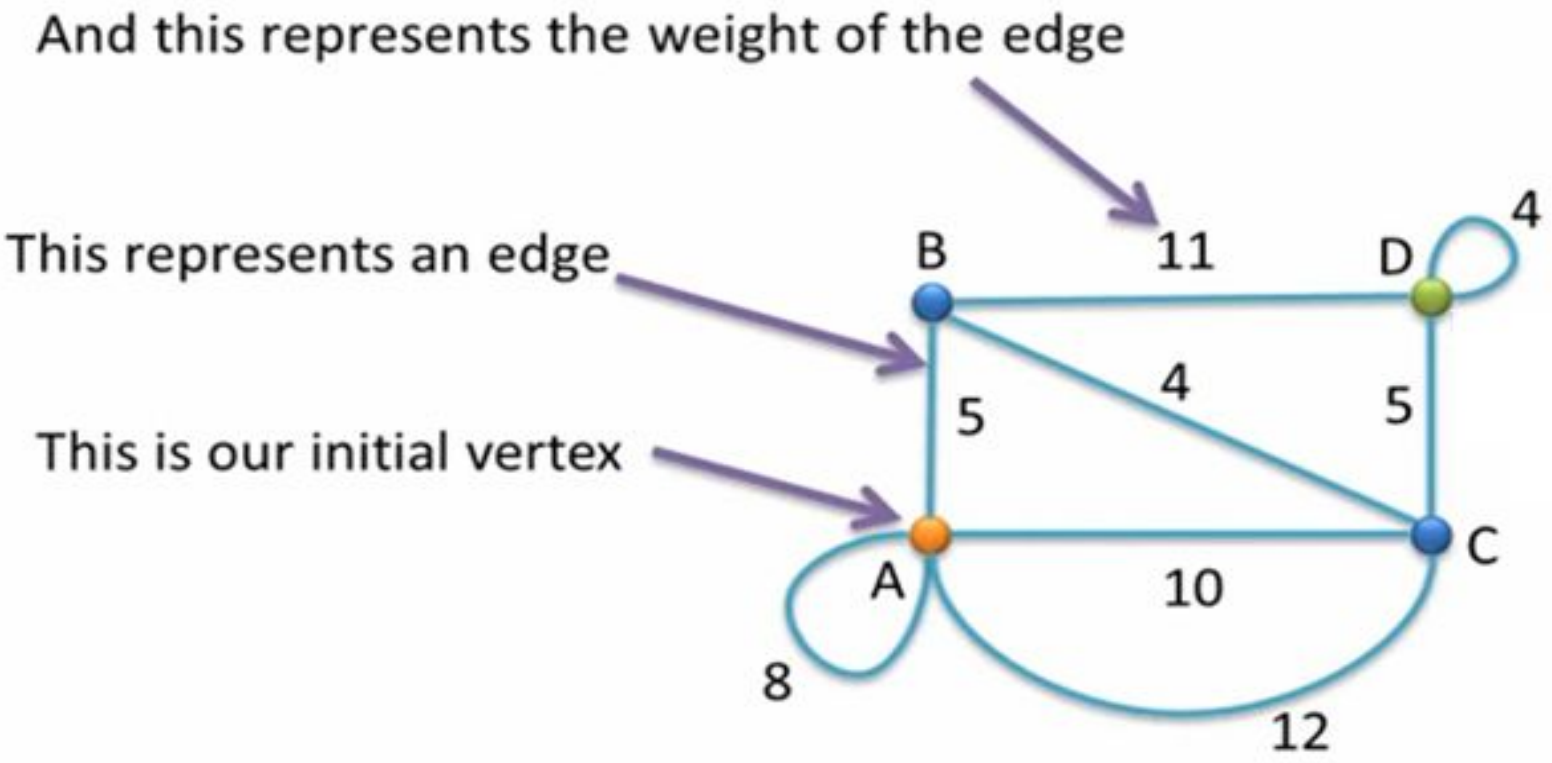
While $S \neq V$

Select a node $v \notin S$ with at least one edge from S for which $d'(v) = \min_{e=(u,v):u \in S} d(u) + \ell_e$ is as small as possible.

Add v to S and define $d(v) = d'(v)$.

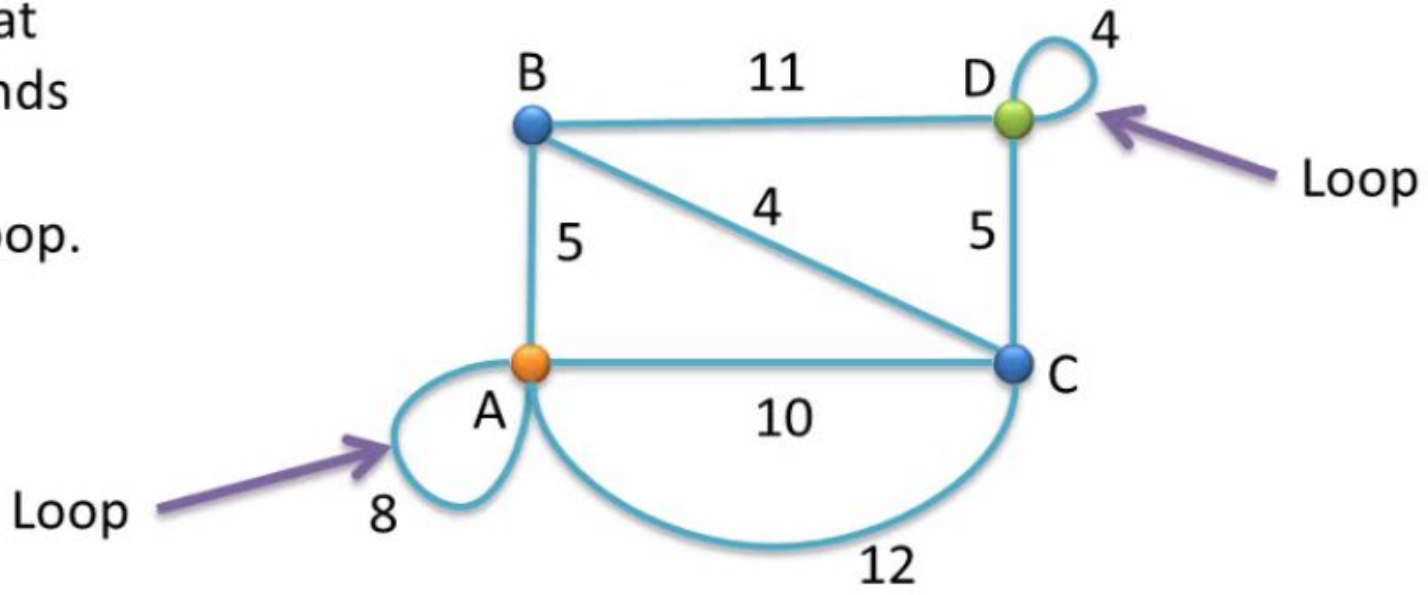
EndWhile

Here is our graph



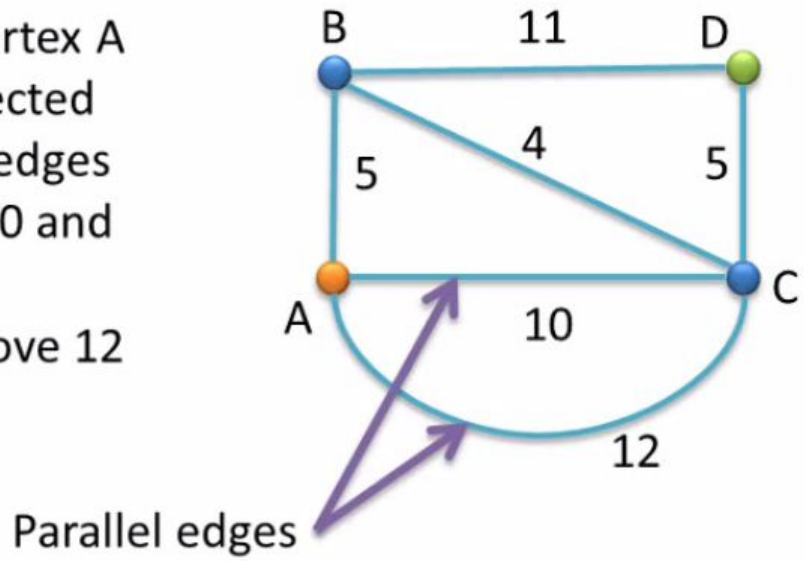
Step 1: Remove all the loops

Note!
Any edge that starts and ends at the same vertex is a loop.

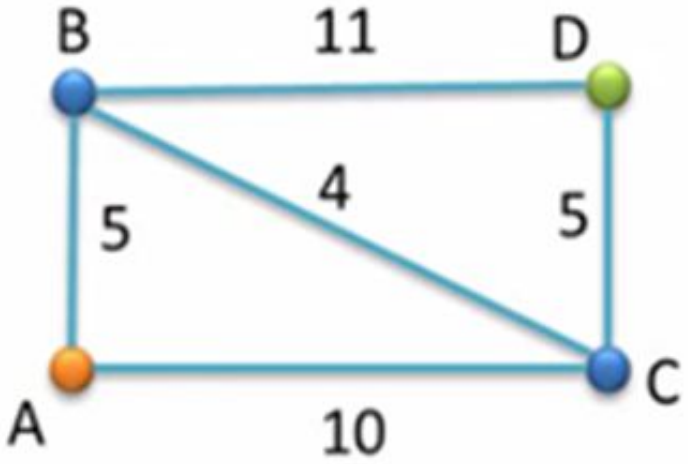


Step 2: Remove all parallel edges between two vertex except the one with least weight

Note!
In this graph, vertex A and C are connected by two parallel edges having weight 10 and 12 respectively.
So, we will remove 12 and keep 10.

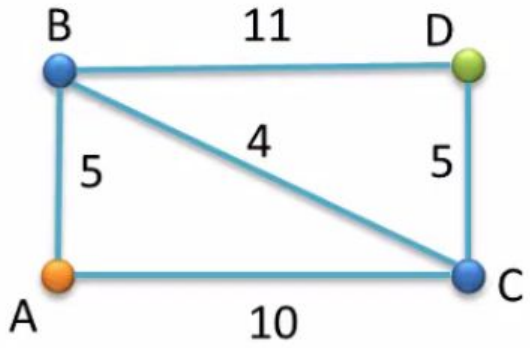


We are now ready to find the shortest path from vertex A



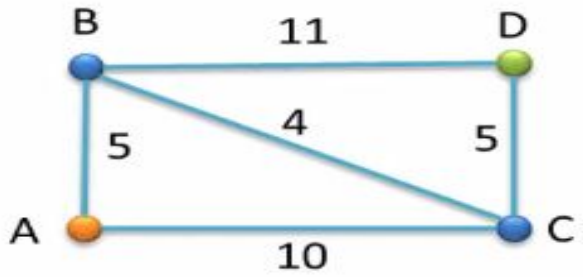
Step 3: Create shortest path table

As our graph has 4 vertices, so our table will have 4 columns



A	B	C	D

Note!
Column name is same as the name of the vertex.



A	B	C	D
0	∞	∞	∞

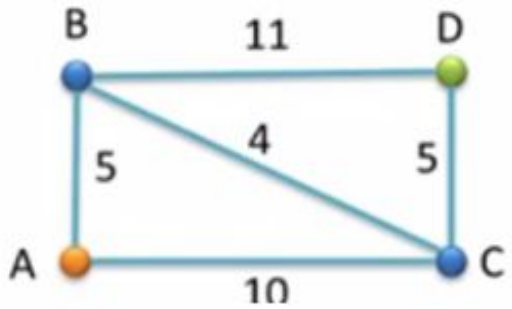
Note!
We have written 0 in column A, as because vertex A is our source.

Now in the 1st row write 0 in column A and ∞ in other columns.

∞ denotes **Infinity**

Value in the columns denote the weight of the shortest path.

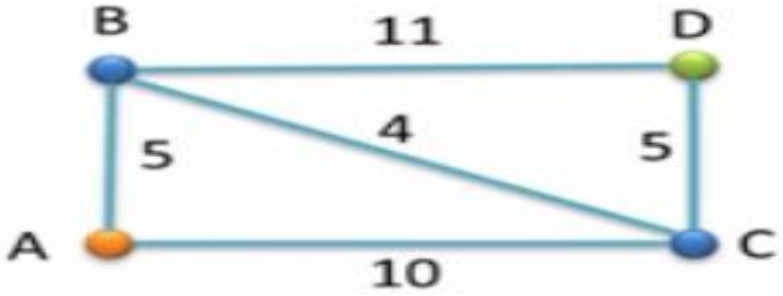
So, 0 in column A denotes we are at vertex A and ∞ in all other columns denote unexplored vertices.



A	B	C	D
0	∞	∞	∞

Now that the 1st row is completely filled, our next job is to find the smallest unmarked value in the 1st row.

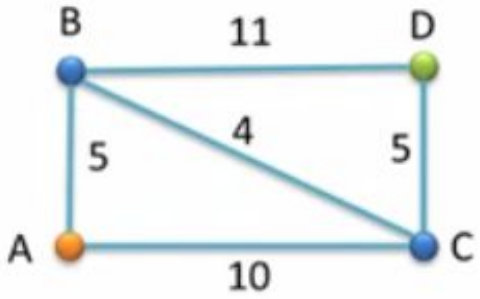
Looking at the table we can say that 0 is the smallest unmarked value in the 1st row. So we will mark it with a square box.



Marked	A	B	C	D
A	0	∞	∞	∞
	0			

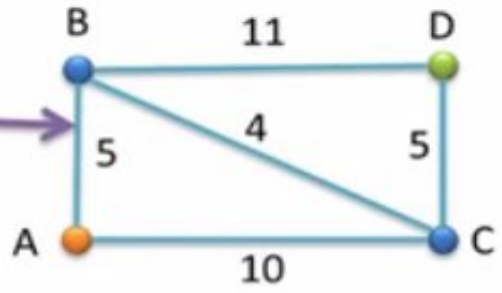
As column A was marked in the previous step, so we will now look for edges that are directly connected with vertex A.

Now draw another row and copy all the marked values in the new row.
In this case 0 is copied.



Marked	A	B	C	D
A	0	∞	∞	∞
	0			

In this case we have an edge of weight 5 that directly connects A and B



Find the edge that **directly** connects vertex A and vertex B

Minimum Value Formula

If we consider two vertices X (Source vertex) and Y (Destination vertex) and an edge that directly connects them.

Then we will have the following formula:

Min(DestValue, MarkedValue + EdgeWeight)

DestValue = The value in the destination vertex (i.e., Y) column.

MarkedValue = The value in the source vertex (i.e., X) column.

EdgeWeight = The weight of the edge that connects the source (i.e., X) and the destination (i.e., Y) vertex.

Minimum Value Formula

Solving:

Min(DestValue, MarkedValue + EdgeWeight)

We will get the minimum value that we will put in the destination vertex (i.e., Y) column.

For example:

If DestValue = 10, MarkedValue = 5 and EdgeWeight = 4

Putting the value we get

Min(10, 5+4)

= Min(10, 9)

= 9 As 9 is smaller than 10.

Marked	A	B	C	D
A	0	∞	∞	∞
	0			

DestValue

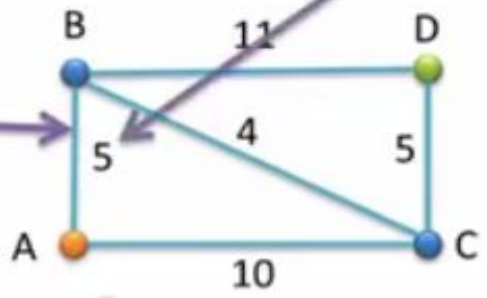
MarkedValue

As we are considering an edge between A and B so,
Source vertex = A
Destination vertex = B

EdgeWeight

Find the edge that **directly** connects vertex A and vertex B

In this case we have an edge of weight 5 that directly connects A and B

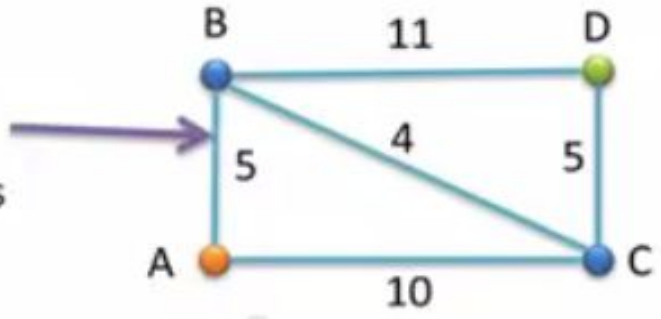


Marked	A	B	C	D
A	0	∞	∞	∞
	0	Min(∞, 0+5)		

So, we will write Min(∞, 0+5) in column B

Solving Min(∞, 0+5)
we get 5.
So, we will put the
value 5 in column B.

In this case we have an edge of weight 5 that directly connects A and B



Find the edge that **directly** connects vertex A and vertex B

Marked	A	B	C	D
A	0	∞	∞	∞
	0	Min(∞, 0+5) 5		

DestValue

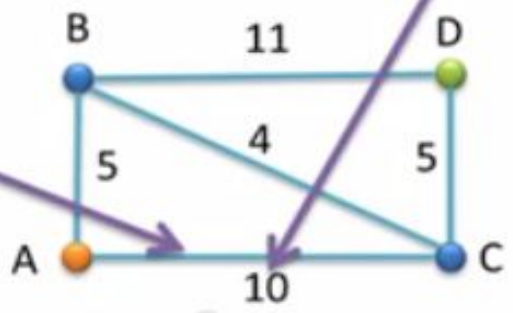
MarkedValue

As we are considering an edge between A and C so,
Source vertex = A
Destination vertex = C

EdgeWeight

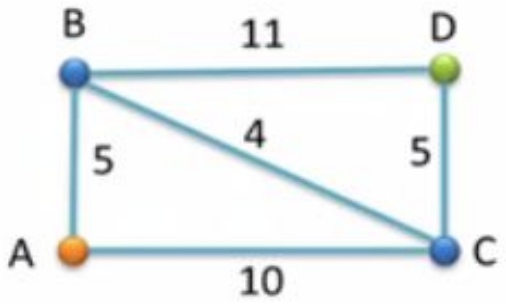
Now find the edge that **directly** connects vertex A and vertex C

In this case we have an edge of weight 10 that directly connects A and C



Marked	A	B	C	D
A	0	∞	∞	∞
	0	Min(∞, 0+5) 5	Min(∞, 0+10) 10	∞

Looking at our graph we find no such edge that directly connects vertex A and D so we will simply copy the previous value in column D i.e., ∞

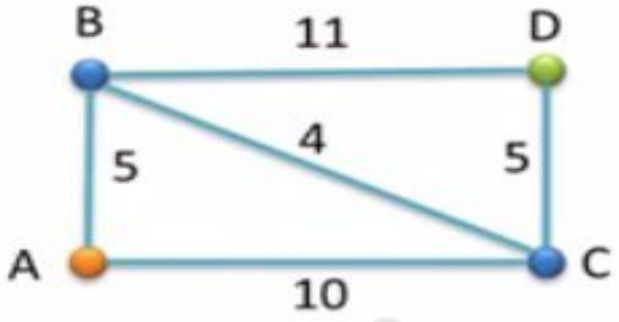


Now find the edge that **directly** connects vertex A and vertex D

Marked	A	B	C	D
A	0	∞	∞	∞
B	0	Min(∞, 0+5) 5	Min(∞, 0+10) 10	∞

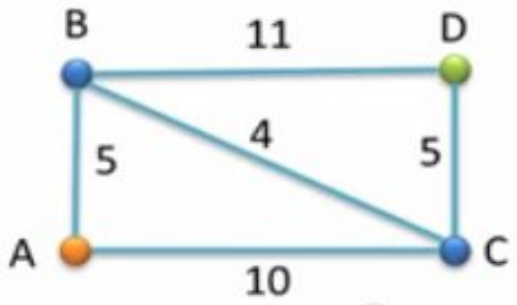
Now that the 2nd row is completely filled, our next job is to find the smallest unmarked value in the 2nd row.

Looking at the table we can say that 5 is the smallest unmarked value in the 2nd row. So we will mark it with a square box.



Marked	A	B	C	D
A	0	∞	∞	∞
B	0	Min(∞, 0+5) 5	Min(∞, 0+10) 10	∞
	0	5		

As column B was marked in the previous step, so we will now look for edges that are directly connected with vertex B.
Note! We don't have to consider vertex A as it is already marked.



Now draw another row and copy all the marked values in the new row.
In this case 0 and 5 are copied.

Marked	A	B	C	D
A	0	∞	∞	∞
B	0	Min(∞, 0+5) 5	Min(∞, 0+10) 10	∞
	0	5		

DestValue

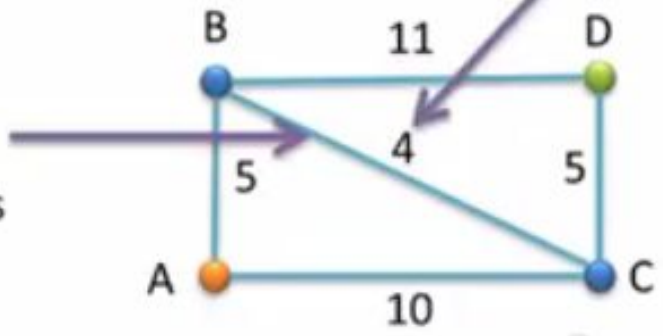
As we are considering an edge between B and C so,
Source vertex = B
Destination vertex = C

MarkedValue

EdgeWeight

Find the edge that **directly** connects vertex B and vertex C

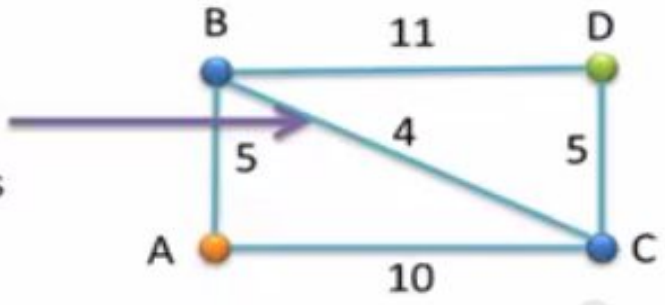
In this case we have an edge of weight 4 that directly connects B and C



Marked	A	B	C	D
A	0	∞	∞	∞
B	0	Min(∞, 0+5) 5	Min(∞, 0+10) 10	∞
	0	5		

So, we will write Min(10, 5+4) in column C.
Solving it we get 9.

In this case we have an edge of weight 4 that directly connects B and C



Find the edge that **directly** connects vertex B and vertex C

Marked	A	B	C	D
A	0	∞	∞	∞
B	0	Min(∞, 0+5) 5	Min(∞, 0+10) 10	∞
	0	5	Min(10, 5+4) 9	

DestValue

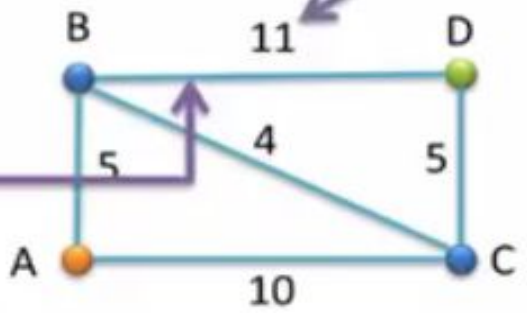
As we are considering an edge between B and D so,
Source vertex = B
Destination vertex = D

MarkedValue

EdgeWeight

Now find the edge that **directly** connects vertex B and vertex D

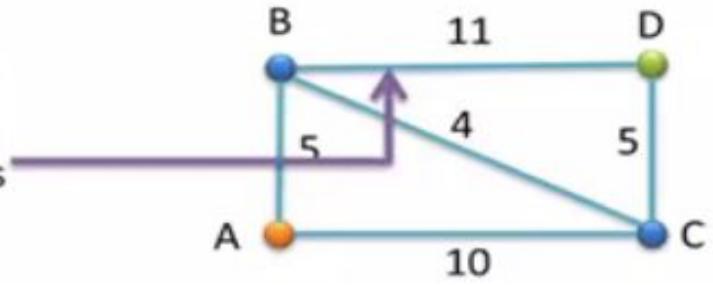
In this case we have an edge of weight 11 that directly connects B and D



Marked	A	B	C	D
A	0	∞	∞	∞
B	0	Min(∞, 0+5) 5	Min(∞, 0+10) 10	∞
	0	5	Min(10, 5+4) 9	Min(∞, 5+11) 16

So, we will write Min(∞, 5+11) in column D.
Solving it we get 16.

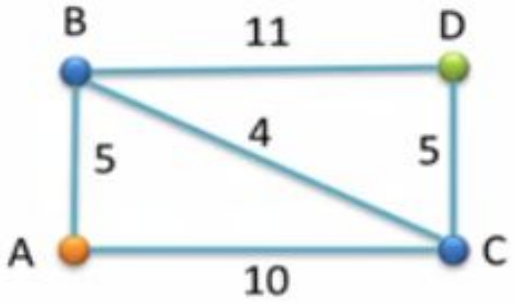
In this case we have an edge of weight 11 that directly connects B and D



Now find the edge that **directly** connects vertex B and vertex D

Marked	A	B	C	D
A	0	∞	∞	∞
B	0	Min(∞, 0+5) **5**	Min(∞, 0+10) 10	∞
C	0	5	Min(10, 5+4) **9**	Min(∞, 5+11) 16

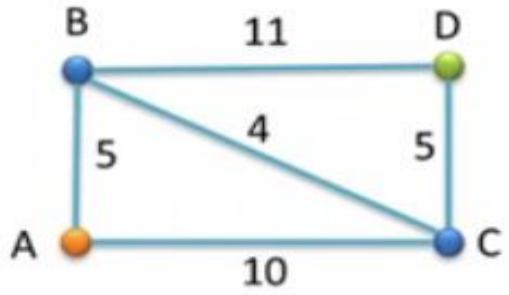
Now that the 3rd row is completely filled, our next job is to find the smallest unmarked value in the 3rd row.



Looking at the table we can say that 9 is the smallest unmarked value in the 3rd row.
So we will mark it with a square box.

Marked	A	B	C	D
A	0	∞	∞	∞
B	0	Min(∞, 0+5) 5	Min(∞, 0+10) 10	∞
C	0	5	Min(10, 5+4) 9	Min(∞, 5+11) 16
	0	5	9	

As column C was marked in the previous step, so we will now look for edges that are directly connected with vertex C.
Note! We don't have to consider vertex A and B as they are already marked.



Now draw another row and copy all the marked values in the new row.
In this case 0, 5 and 9 are copied.

Marked	A	B	C	D
A	0	∞	∞	∞
B	0	Min(∞, 0+5) 5	Min(∞, 0+10) 10	∞
C	0	5	Min(10, 5+4) 9	Min(∞, 5+11) 16
	0	5	9	

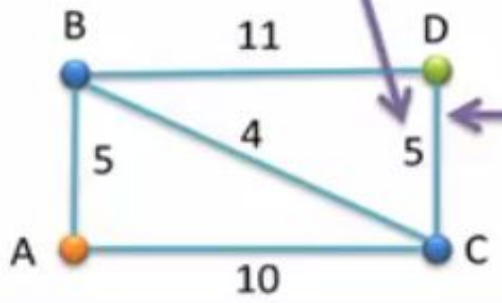
DestValue

MarkedValue

Find the edge that **directly** connects vertex C and vertex D

EdgeWeight

As we are considering an edge between C and D so,
Source vertex = C
Destination vertex = D

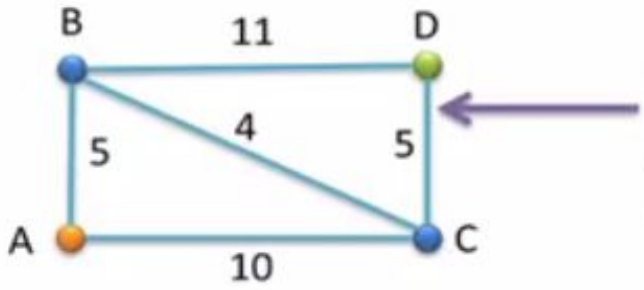


In this case we have an edge of weight 5 that directly connects C and D

Marked	A	B	C	D
A	0	∞	∞	∞
B	0	Min(∞, 0+5) 5	Min(∞, 0+10) 10	∞
C	0	5	Min(10, 5+4) 9	Min(∞, 5+11) 16
	0	5	9	Min(16, 9+5) 14

So, we will write Min(16, 9+5) in column D.
Solving it we get 14.

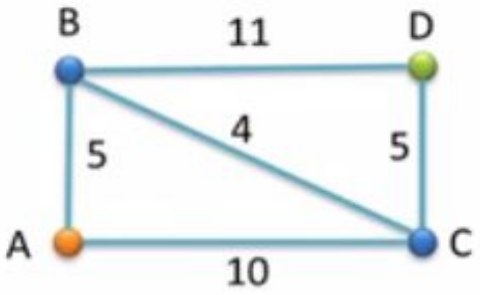
Find the edge that **directly** connects vertex C and vertex D



In this case we have an edge of weight 5 that directly connects C and D

Marked	A	B	C	D
A	0	∞	∞	∞
B	0	Min(∞, 0+5) **5**	Min(∞, 0+10) 10	∞
C	0	5	Min(10, 5+4) **9**	Min(∞, 5+11) 16
	0	5	9	Min(16, 9+5) 14

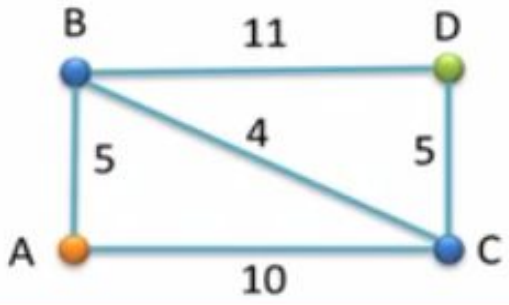
Now that the 4th row is completely filled, our next job is to find the smallest unmarked value in the 4th row.



Looking at the table we can say that 14 is the smallest unmarked value in the 4th row.
So we will mark it with a square box.

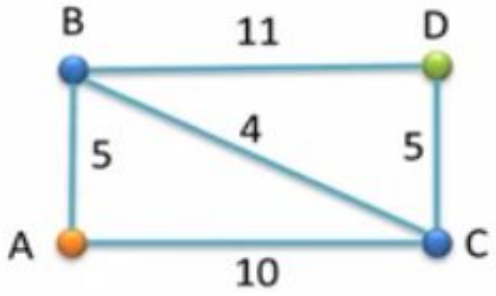
Marked	A	B	C	D
A	0	∞	∞	∞
B	0	Min(∞, 0+5) 5	Min(∞, 0+10) 10	∞
C	0	5	Min(10, 5+4) 9	Min(∞, 5+11) 16
D	0	5	9	Min(16, 9+5) 14

As final vertex **D** is marked so we will stop here.



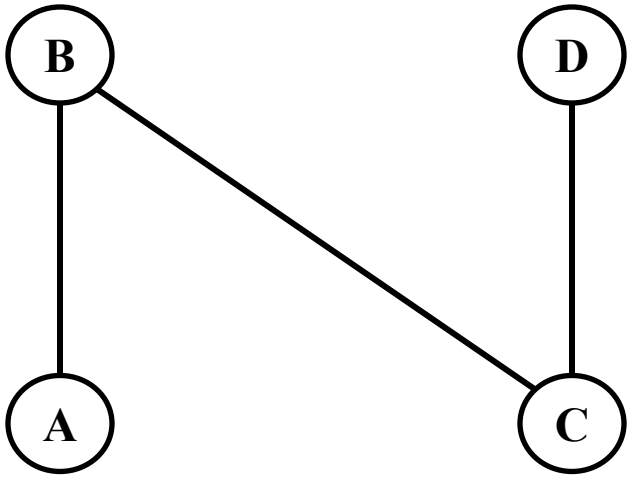
Marked	A	B	C	D
A	0	∞	∞	∞
B	0	Min(∞, 0+5) **5**	Min(∞, 0+10) 10	∞
C	0	5	Min(10, 5+4) **9**	Min(∞, 5+11) 16
D	0	5	9	Min(16, 9+5) **14**

Note! Column D has the marked value 14.
This means our shortest path has the weight 14.



SHORTEST PATHS

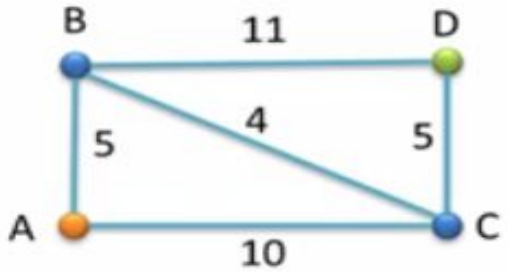
- A □ B
- A □ C
- A □ D



Time to find the shortest path using backtracking

Marked	A	B	C	D
A	0	∞	∞	∞
B	0	Min(∞, 0+5) **5**	Min(∞, 0+10) 10	∞
C	0	5	Min(10, 5+4) **9**	Min(∞, 5+11) 16
D ✓	0	5	9	Min(16, 9+5) **14** ←

Backtracking is very simple.
We will move upwards row by row and will stop only when we find a change in value.



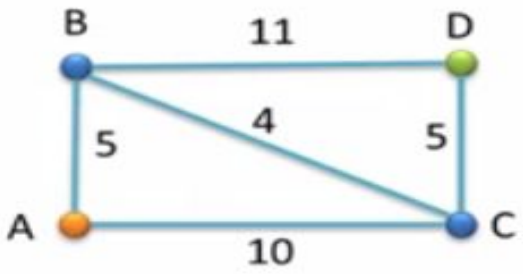
Start from the final marked value 14.
Follow the steps carefully.

Marked	A	B	C	D
A	0	∞	∞	∞
B	0	Min(∞, 0+5) 5	Min(∞, 0+10) 10	∞
C ✓	0	5	Min(10, 5+4) 9	Min(∞, 5+11) 16
D ✓	0	5	9	Min(16, 9+5) 14

Move one row upwards.

Has the value changed?

YES
It is 16.

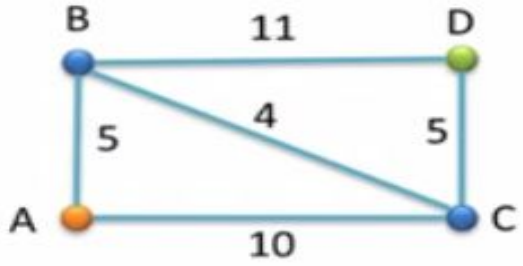


So, tick the vertex that was marked in that row.

i.e., vertex C

Marked	A	B	C	D
A	0	∞	∞	∞
B	0	Min(∞, 0+5) 5	Min(∞, 0+10) 10	∞
C ✓	0	5	Min(10, 5+4) 9	Min(∞, 5+11) 16
D ✓	0	5	9	Min(16, 9+5) 14

Move one row upwards.

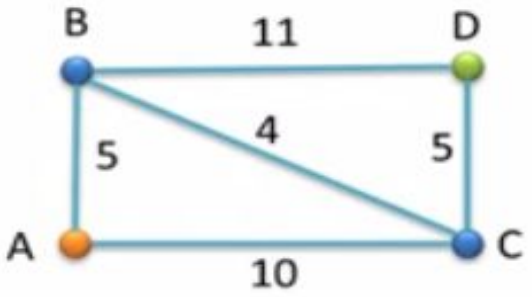


Marked	A	B	C	D
A	0	∞	∞	∞
B ✓	0	Min(∞, 0+5) **5**	Min(∞, 0+10) 10	∞
C ✓	0	5	Min(10, 5+4) **9**	Min(∞, 5+11) 16
D ✓	0	5	9	Min(16, 9+5) **14**

Move one row upwards.

Has the value changed?

YES
It is 10.

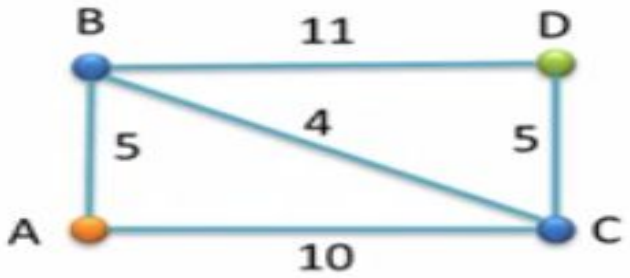


So, tick the vertex that was marked in that row.

i.e., vertex B

Marked	A	B	C	D
A	0	∞	∞	∞
B ✓	0	Min(∞, 0+5) **5**	Min(∞, 0+10) 10	∞
C ✓	0	5	Min(10, 5+4) **9**	Min(∞, 5+11) 16
D ✓	0	5	9	Min(16, 9+5) **14**

Move the pointer to point at value 5 in column B

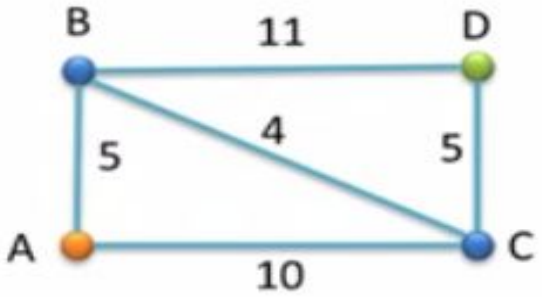


Marked	A	B	C	D
A ✓	0	∞	∞	∞
B ✓	0	Min(∞, 0+5) **5**	Min(∞, 0+10) 10	∞
C ✓	0	5	Min(10, 5+4) **9**	Min(∞, 5+11) 16
D ✓	0	5	9	Min(16, 9+5) **14**

Move one row upwards.

Has the value changed?

YES
It is ∞ (Infinity).

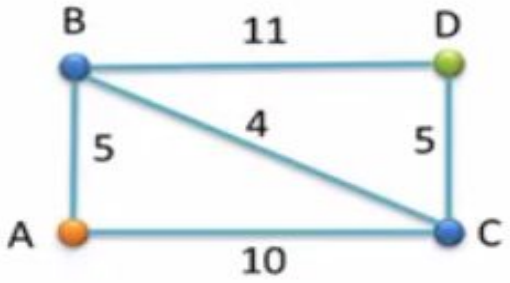


So, tick the vertex that was marked in that row.

i.e., vertex A

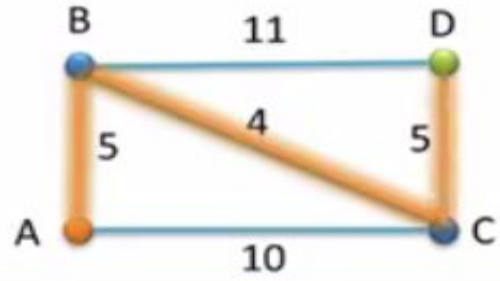
Marked	A	B	C	D
A ✓	0	∞	∞	∞
B ✓	0	Min(∞, 0+5) **5**	Min(∞, 0+10) 10	∞
C ✓	0	5	Min(10, 5+4) **9**	Min(∞, 5+11) 16
D ✓	0	5	9	Min(16, 9+5) **14**

Since the initial vertex A is ticked.
So we will stop here.



Therefore, the required shortest path is

A → B → C → D

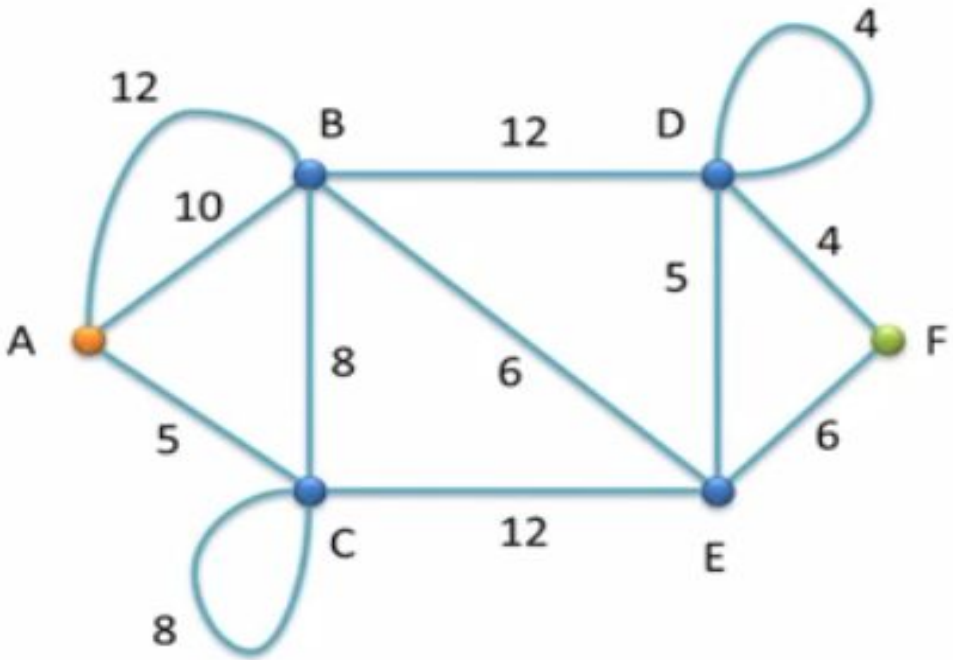


Point to Remember

In case there are two or more columns with the same smallest unmarked value, then you can choose anyone of them.

This only denotes that there will be more than one shortest path in the graph.

Here is our graph



Marked	A	B	C	D	E	F
A ✓	0	∞	∞	∞	∞	∞
C	0	Min(∞, 0+10) 10	Min(∞, 0+5) 5	∞	∞	∞
B ✓	0	Min(10, 5+8) 10	5	∞	Min(∞, 5+12) 17	∞
E ✓	0	10	5	Min(∞, 10+12) 22	Min(17, 10+6) 16	∞
D	0	10	5	Min(22, 16+5) 21	16	Min(∞, 16+6) 22
F ✓	0	10	5	21	16	Min(22, 21+4) 22

Since the source vertex is ticked. So we will stop here.

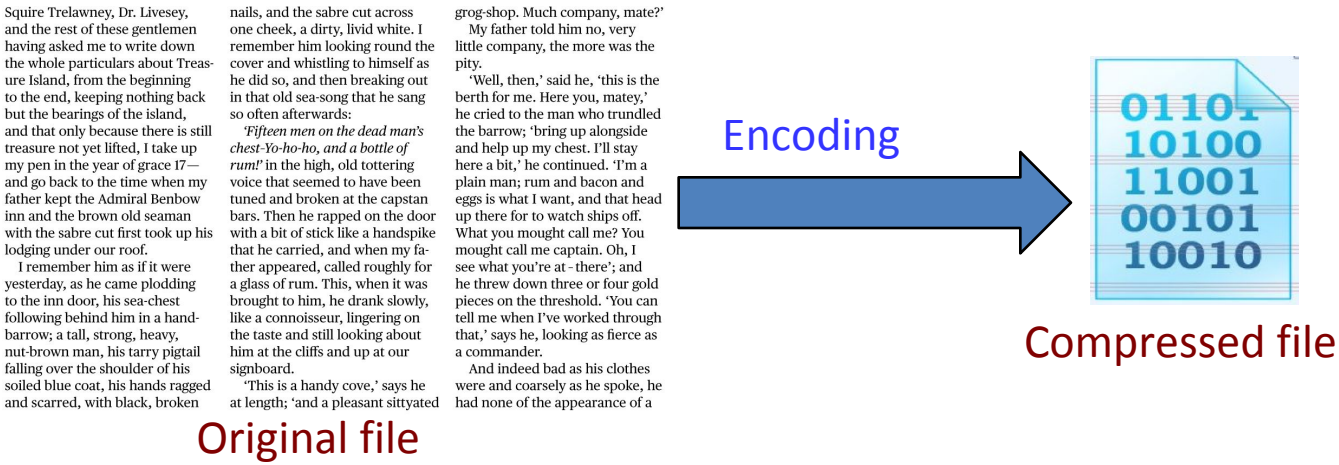
Therefore, the required shortest path is

Optimal Tree Problem

- ✔ Problem is that, we have to encode a text that comprises symbols from some n-symbol alphabet by assigning to each of the text's symbols some sequence of bits called the **codeword**

- ✔ **There are two ways of encoding:**
  - ✔ Fixed-length encoding and
  - ✔ Variable-length encoding**.**

Optimal Tree Problem



- **A technique to compress data effectively**
  - **Usually between 20%-90% compression**

- **Lossless compression**
  - **No information is lost**
  - **When decompress, you get the original file**

Optimal Tree Problem

Squire Trelawney, Dr. Livesey, and the rest of these gentlemen having asked me to write down the whole particulars about Treasure Island, from the beginning to the end, keeping nothing back but the bearings of the island, and that only because there is still treasure not yet lifted, I take up my pen in the year of grace 17— and go back to the time when my father kept the Admiral Benbow inn and the brown old seaman with the sabre cut first took up his lodging under our roof.

I remember him as if it were yesterday, as he came plodding to the inn door, his sea-chest following behind him in a hand-barrow; a tall, strong, heavy, nut-brown man, his tarry pigtail falling over the shoulder of his soiled blue coat, his hands ragged and scarred, with black, broken nails, and the sabre cut across one cheek, a dirty, livid white. I remember him looking round the cover and whistling to himself as he did so, and then breaking out in that old sea-song that he sang so often afterwards:

'Fifteen men on the dead man's chest-Yo-ho-ho, and a bottle of rum!' in the high, old tottering voice that seemed to have been tuned and broken at the capstan bars. Then he rapped on the door with a bit of stick like a handspike that he carried, and when my father appeared, called roughly for a glass of rum. This, when it was brought to him, he drank slowly, like a connoisseur, lingering on the taste and still looking about him at the cliffs and up at our signboard.

'This is a handy cove,' says he at length; 'and a pleasant sittyated grog-shop. Much company, mate?'

My father told him no, very little company, the more was the pity.

'Well, then,' said he, 'this is the berth for me. Here you, matey,' he cried to the man who trundled the barrow; 'bring up alongside and help up my chest. I'll stay here a bit,' he continued. 'I'm a plain man; rum and bacon and eggs is what I want, and that head up there for to watch ships off. What you mought call me? You mought call me captain. Oh, I see what you're at - there'; and he threw down three or four gold pieces on the threshold. 'You can tell me when I've worked through that,' says he, looking as fierce as a commander.

And indeed bad as his clothes were and coarsely as he spoke, he had none of the appearance of a

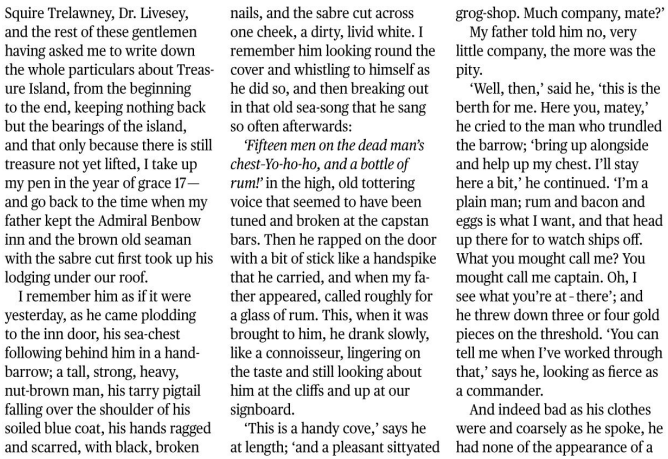
Original file

Huffman coding

Compressed file

- **Saving space**
  - **Store compressed files instead of original files**
- **Transmitting files or data**
  - **Send compressed data to save transmission time and power**
- **Encryption and decryption**
  - **Cannot read the compressed file without knowing the "key"**

Optimal Tree Problem



Squire Trelawney, Dr. Livesey, and the rest of these gentlemen having asked me to write down the whole particulars about Treasure Island, from the beginning to the end, keeping nothing back but the bearings of the island, and that only because there is still treasure not yet lifted, I take up my pen in the year of grace 17— and go back to the time when my father kept the Admiral Benbow inn and the brown old seaman with the sabre cut first took up his lodging under our roof.

I remember him as if it were yesterday, as he came plodding to the inn door, his sea-chest following behind him in a hand-barrow; a tall, strong, heavy, nut-brown man, his tarry pigtail falling over the shoulder of his soiled blue coat, his hands ragged and scarred, with black, broken nails, and the sabre cut across one cheek, a dirty, livid white. I remember him looking round the cover and whistling to himself as he did so, and then breaking out in that old sea-song that he sang so often afterwards:

'Fifteen men on the dead man's chest-Yo-ho-ho, and a bottle of rum!' in the high, old tottering voice that seemed to have been tuned and broken at the capstan bars. Then he rapped on the door with a bit of stick like a handspike that he carried, and when my father appeared, called roughly for a glass of rum. This, when it was brought to him, he drank slowly, like a connoisseur, lingering on the taste and still looking about him at the cliffs and up at our signboard.

'This is a handy cove,' says he at length; 'and a pleasant sittyated grog-shop. Much company, mate?'

My father told him no, very little company, the more was the pity.

'Well, then,' said he, 'this is the berth for me. Here you, matey,' he cried to the man who trundled the barrow; 'bring up alongside and help up my chest. I'll stay here a bit,' he continued. 'I'm a plain man; rum and bacon and eggs is what I want, and that head up there for to watch ships off. What you mought call me? You mought call me captain. Oh, I see what you're at - there'; and he threw down three or four gold pieces on the threshold. 'You can tell me when I've worked through that,' says he, looking as fierce as a commander.

And indeed bad as his clothes were and coarsely as he spoke, he had none of the appearance of a

- Assume in this file only 6 characters appear
  - E, A, C, T, K, N
- The frequencies are:

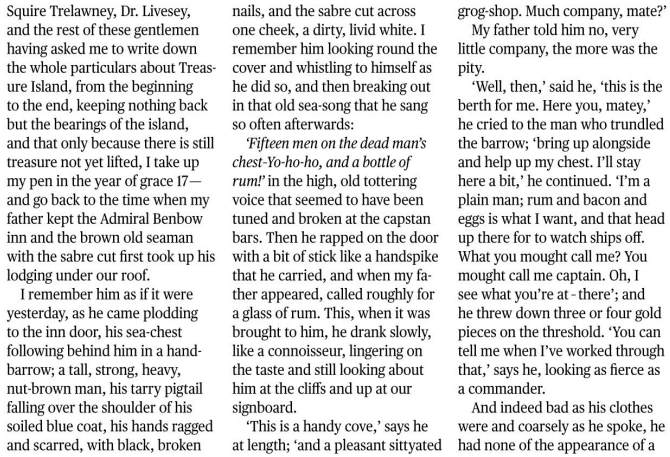
Character	Frequency
E	10,000
A	4,000
C	300
T	200
K	100
N	100

- Option I (No Compression)
  - Each character = 1 Byte (8 bits)
  - Total file size = 14,700 * 8 = 117,600 bits

- Option 2 (Fixed length encoding- compression)
  - We have 6 characters, so we need 3 bits to encode them
  - Total file size = 14,700 * 3 = 44,100 bits

Character	Fixed Encoding
E	000
A	001
C	010
T	100
K	110
N	111

Optimal Tree Problem



Squire Trelawney, Dr. Livesey, and the rest of these gentlemen having asked me to write down the whole particulars about Treasure Island, from the beginning to the end, keeping nothing back but the bearings of the island, and that only because there is still treasure not yet lifted, I take up my pen in the year of grace 17— and go back to the time when my father kept the Admiral Benbow inn and the brown old seaman with the sabre cut first took up his lodging under our roof.

I remember him as if it were yesterday, as he came plodding to the inn door, his sea-chest following behind him in a hand-barrow; a tall, strong, heavy, nut-brown man, his tarry pigtail falling over the shoulder of his soiled blue coat, his hands ragged and scarred, with black, broken nails, and the sabre cut across one cheek, a dirty, livid white. I remember him looking round the cover and whistling to himself as he did so, and then breaking out in that old sea-song that he sang so often afterwards:

'Fifteen men on the dead man's chest-Yo-ho-ho, and a bottle of rum!' in the high, old tottering voice that seemed to have been tuned and broken at the capstan bars. Then he rapped on the door with a bit of stick like a handspike that he carried, and when my father appeared, called roughly for a glass of rum. This, when it was brought to him, he drank slowly, like a connoisseur, lingering on the taste and still looking about him at the cliffs and up at our signboard.

'This is a handy cove,' says he at length; 'and a pleasant sittyated grog-shop. Much company, mate?'

My father told him no, very little company, the more was the pity.

'Well, then,' said he, 'this is the berth for me. Here you, matey,' he cried to the man who trundled the barrow; 'bring up alongside and help up my chest. I'll stay here a bit,' he continued. 'I'm a plain man; rum and bacon and eggs is what I want, and that head up there for to watch ships off. What you mought call me? You mought call me captain. Oh, I see what you're at - there'; and he threw down three or four gold pieces on the threshold. 'You can tell me when I've worked through that,' says he, looking as fierce as a commander.

And indeed bad as his clothes were and coarsely as he spoke, he had none of the appearance of a

- Assume in this file only 6 characters appear
  - E, A, C, T, K, N
- The frequencies are:

Character	Frequency
E	10,000
A	4,000
C	300
T	200
K	100
N	100

- Option 3 (Variable length encoding - compression)
  - Variable-length compression
  - Assign shorter codes to more frequent characters and longer codes to less frequent characters
  - Total file size:

(10,000 x 1) + (4,000 x 2) + (300 x 3) + (200 x 4) + (100 x 5) + (100 x 5) = 20,700 bits

Char.	Variable Length Encoding
E	0
A	10
C	110
T	1110
K	11110
N	11111

Optimal Tree Problem

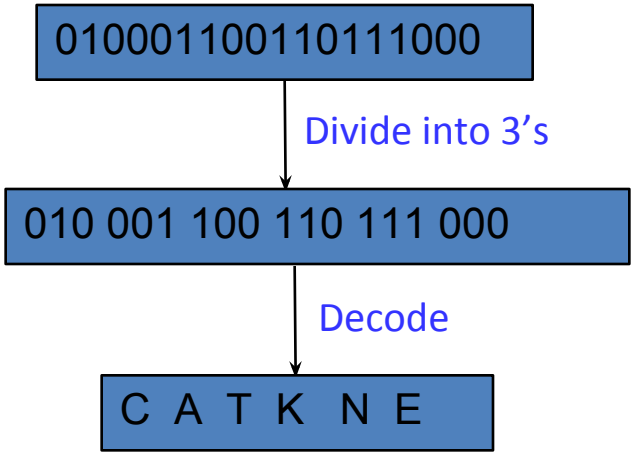
- **A variable-length coding for characters**
  - **More frequent characters shorter codes**
  - **Less frequent characters longer codes**

- **It is not like ASCII coding where all characters have the same coding length (8 bits)**

- **Two main questions**
  - **How to assign codes *(Encoding process)?***
  - **How to decode (from the compressed file, generate the original file) *(Decoding process)?***

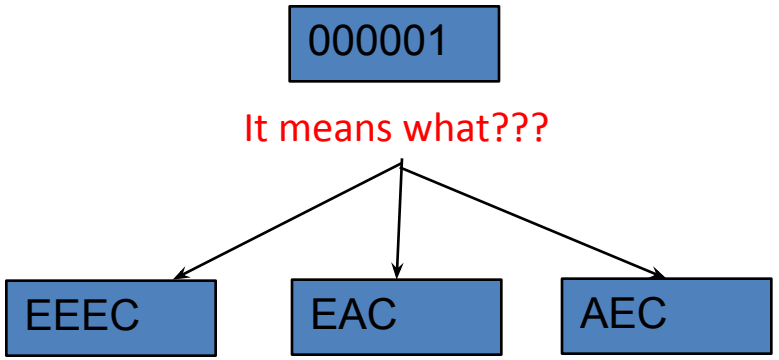
Decoding for fixed-length codes is much easier

Character	Fixed-length Encoding
E	000
A	001
C	010
T	100
K	110
N	111



Decoding for variable-length codes is not that easy…

Character	Variable-length Encoding
E	0
A	00
C	001
…	…
…	…
…	…



Optimal Tree Problem

- To avoid this complication, we can limit ourselves to the so-called ***prefix-free (or simply prefix) codes.***

- In a prefix code, no codeword is a prefix of a codeword of another symbol.

- Hence, with such an encoding, we can simply scan a bit string until we get the first group of bits that is a codeword for some symbol, replace these bits by this symbol, and repeat this operation until the bit string‟s end is reached.

- Example: a = 0, b = 101, c = 100
  - Decode 00100
    - Translates to “aac”

Optimal Tree Problem

- The solution to the problem to create a binary prefix code for some alphabet, is to associate the alphabet's symbols with leaves of a binary tree in which

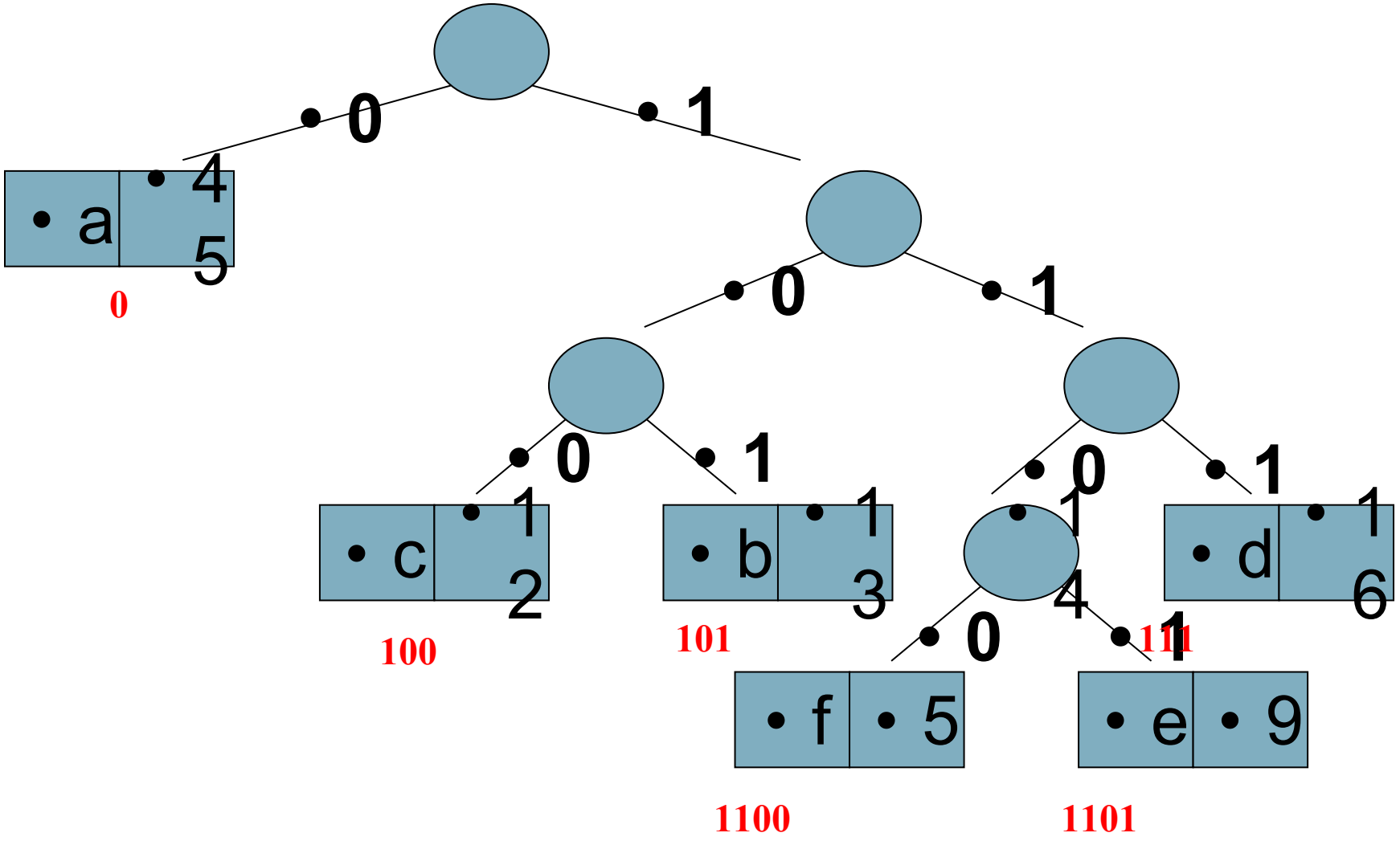
  - ✔ all the left edges are labeled by 0
  - ✔ all the right edges are labeled by 1.

Optimal Tree Problem

- The codeword of a symbol can then be obtained by recording the labels on the simple path from the root to the symbol's leaf.

- Among the many trees that can be constructed to solve this problem – The optimal one can be done by the greedy algorithm, invented by David Huffman known as Huffman algorithm.

Example



Huffman Algorithm

Step 1: Get Frequencies

- Scan the file to be compressed and count the occurrence of each character
- Sort the characters based on their frequency

Step 2: Build Tree & Assign Codes

- Build a Huffman tree (binary tree)
- Traverse the tree to assign codes

Step 3: Encode (Compress)

- Scan the file again and replace each character by its code

Step 4: Decode (Decompress)

- Huffman tree is the key to decompress the file

How to build Huffman tree ??

Step 1: Initialize ‘n’ one-node trees and label them with the symbols of the alphabet given. Record the frequency of each symbol in its tree’s root to indicate the tree’s weight.

Step 2: Repeat the following operation until a single tree is obtained.

- ✔ Find two trees with the smallest weight.
- ✔ Make them the left and right subtree of a new tree and record the sum of their weights in the root of the new tree as its weight.
- ✔ Rearrange the nodes again in ascending order of weights

- ❖ A tree constructed by the above algorithm is called as **Huffman tree.**
- ❖ The codeword defined using this tree is called as **Huffman code.**

Huffman Code- Example

Construct the Huffman Code for the following data:

symbol	A	B	C	D	_
frequency	0.35	0.1	0.2	0.2	0.15

Encode the text **DAD** and decode the text **100110110111101**

Solution

THANK YOU