SNS COLLEGE OF TECHNOLOGY

Coimbatore-35
An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF MCA

19CAT602 – DATA STRUCTURES & ALGORITHMS

I YEAR I SEM

UNIT IV – Greedy and Backtracking

TOPIC 21- Greedy Method: Prim's and Kruskal's Algorithm

Greedy Algorithms:

- Many real-world problems are optimization problems in that they attempt to find an optimal solution among many possible candidate solutions.
- An optimization problem is one in which you want to find, not just *a* solution, but the *best* solution
- A "greedy algorithm" sometimes works well for optimization problems
- A greedy algorithm works in phases. At each phase: You take the best you can get right now, without regard for future consequences. You hope that by choosing a *local* optimum at each step, you will end up at a *global* optimum
- A familiar scenario is the change-making problem that we often encounter at a cash register: receiving the fewest numbers of coins to make change after paying the bill for a purchase.

Greedy Technique:

- Constructs a solution to an optimization problem piece by
- piece through a sequence of choices that are:

    1. feasible, i.e. satisfying the constraints

    2. locally optimal (with respect to some neighborhood definition)

    3. greedy (in terms of some measure), and irrevocable

- For some problems, it yields a globally optimal solution for every instance. For most, does not but can be useful for fast approximations. We are mostly interested in the former case in this class.

Greedy Techniques:

- Optimal solutions:
  - change making for “normal” coin denominations
  - minimum spanning tree (MST)
    - **Prim’s MST**
    - **Kruskal’s MST**
  - simple scheduling problems
  - **Dijkstra’s algo**
  - Huffman codes

- Approximations/heuristics:
  - traveling salesman problem (TSP)
  - knapsack problem
  - other combinatorial optimization problems

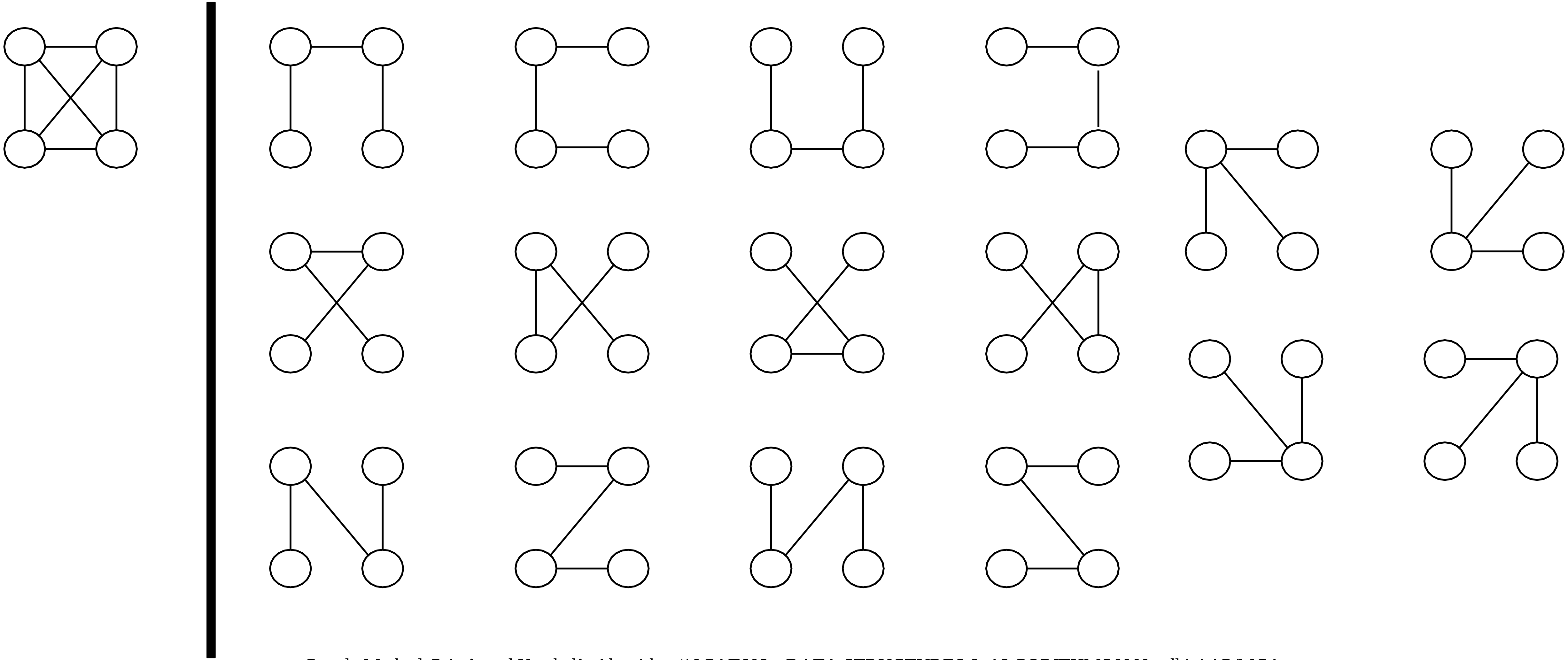
Greedy Scenario:

- **Feasible**
  - Has to satisfy the problem's constraints
- **Locally Optimal**
  - Has to make the best local choice among all feasible choices available on that step
    - If this local choice results in a global optimum then the problem has optimal substructure
- **Irrevocable**
  - Once a choice is made it can't be un-done on subsequent steps of the algorithm
- **Simple examples**:
  - Playing chess by making best move without look-ahead
  - Giving fewest number of coins as change

- Simple and appealing, but don't always give the best solution

Minimum Spanning Tree (MST):

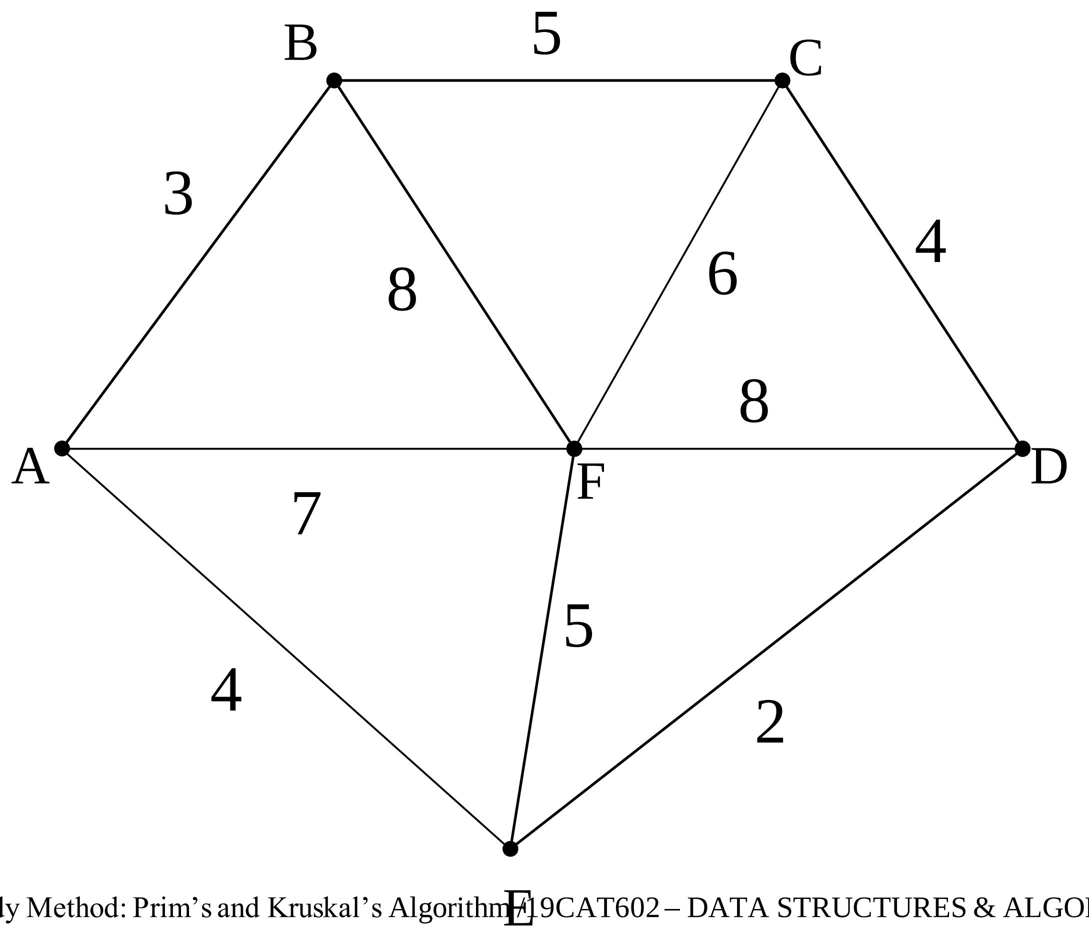
16 states of Spanning tree can happened



Solution for MST:

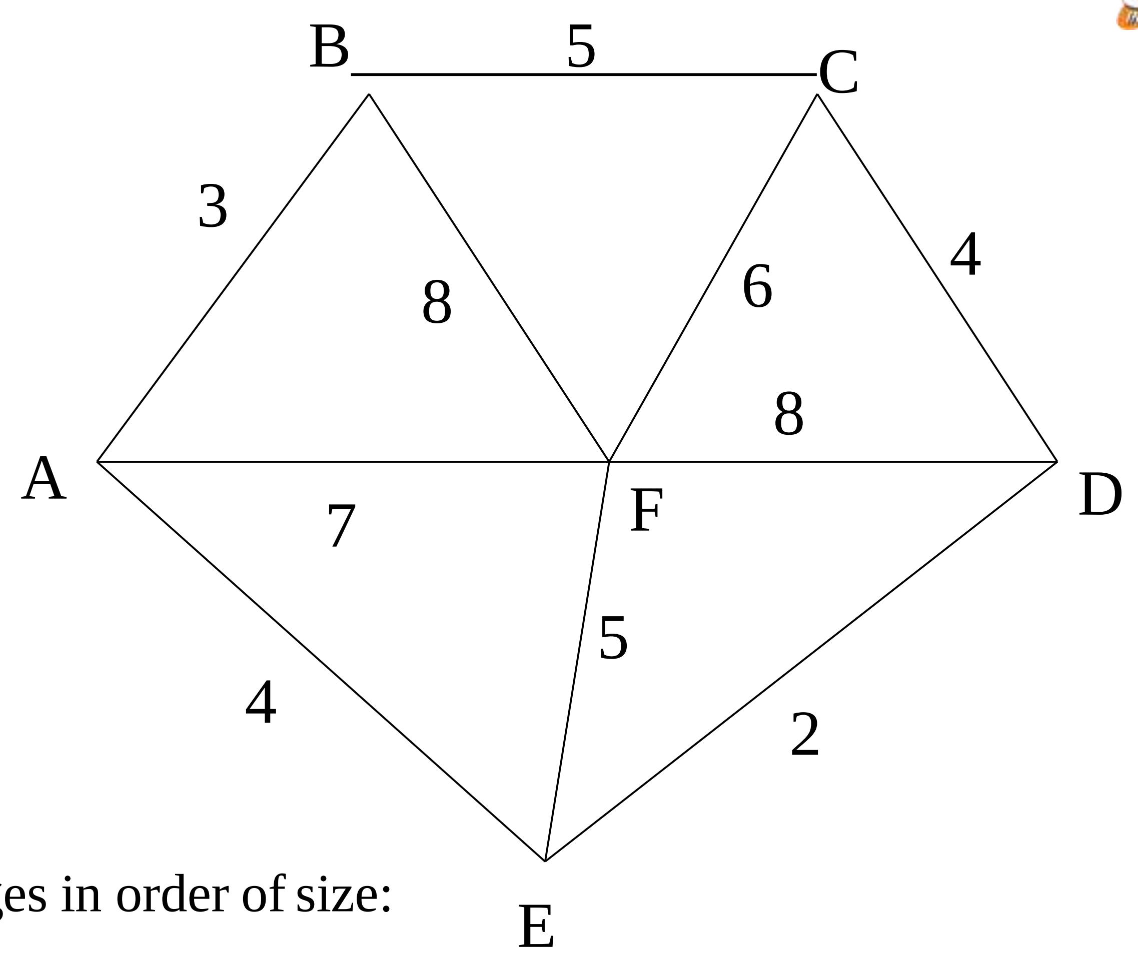
Example A cable company want to connect five villages to their network which currently extends to the market town

What is the minimum length of cable needed?



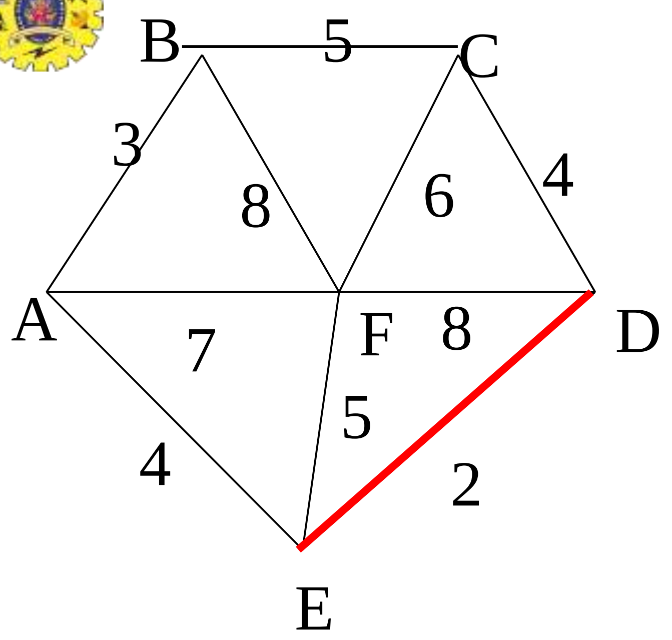
Kruskal’s Algorithm:

MST-KRUSKAL(G, w)
1. A ← Ø
2. for each vertex v V[G]
3. do MAKE-SET(v)
4.sort the edges of E into nondecreasing order by weight w
5.for each edge (u, v) E, taken in nondecreasing order by weight
6. do if FIND-SET(u) ≠ FIND-SET(v)
7. then A ← A {(u, v)}
8. UNION(u, v)
9. return A



List the edges in order of size:

ED 2	AB 3
AE 4	CD 4
BC 5	EF 5
CF 6	AF 7
BF 8	CF 8

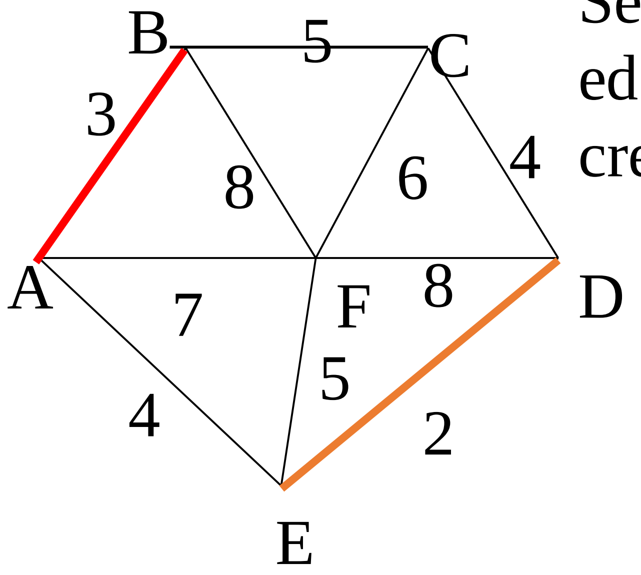


Select the shortest edge in the network

ED 2

1

2

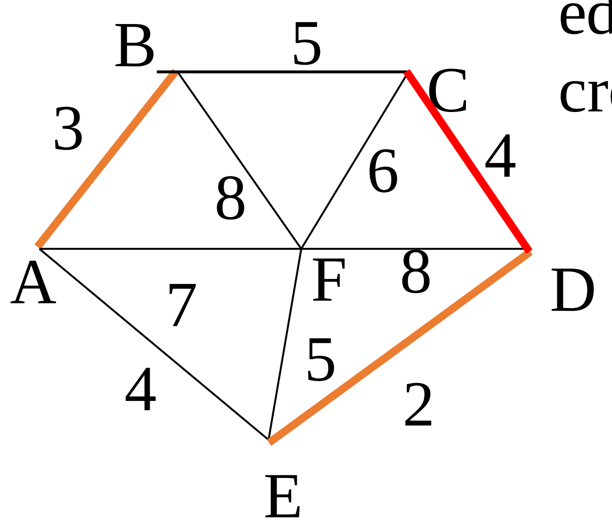


Select the next shortest edge which does not create a cycle

ED 2
AB 3

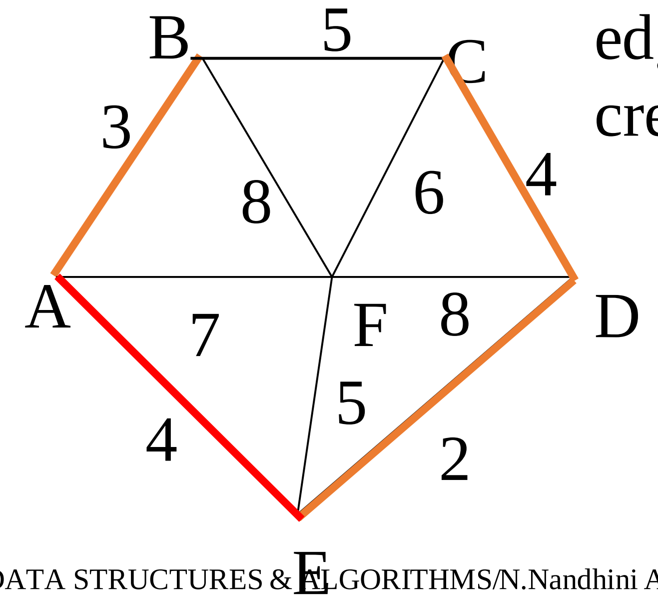
3

Select the next shortest edge which does not create a cycle



ED 2
AB 3
CD 4 (or AE 4)

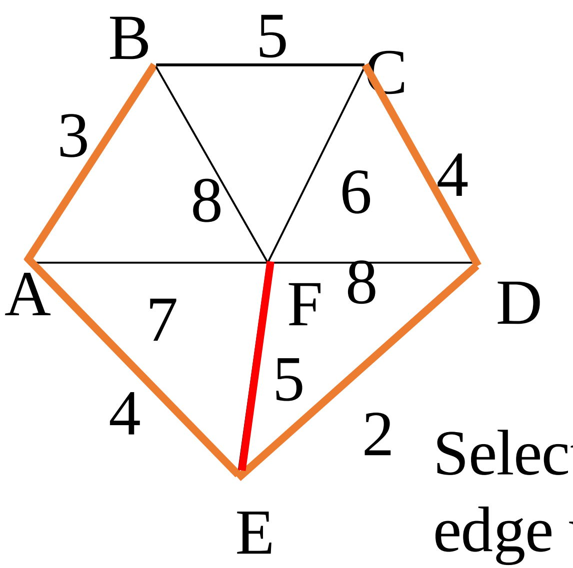
4



Select the next shortest edge which does not create a cycle

ED 2
AB 3
CD 4
AE 4

Kruskal's Algorithm:



Select the next shortest edge which does not create a cycle

ED 2

AB 3

CD 4

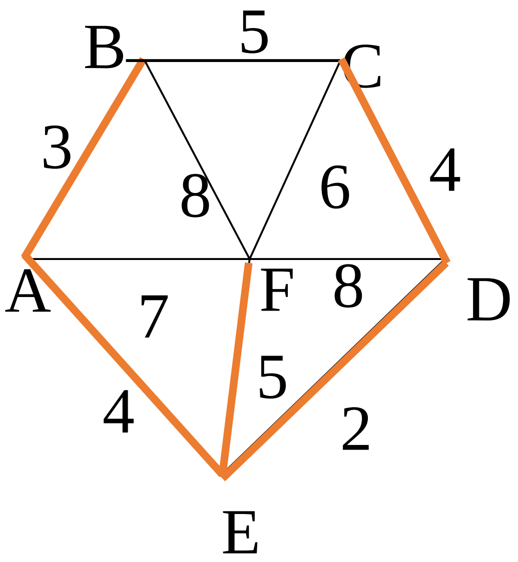
AE 4

BC 5 – forms a cycle

EF 5

5

6



All vertices have been con

The solution is

ED 2

AB 3

CD 4

AE 4

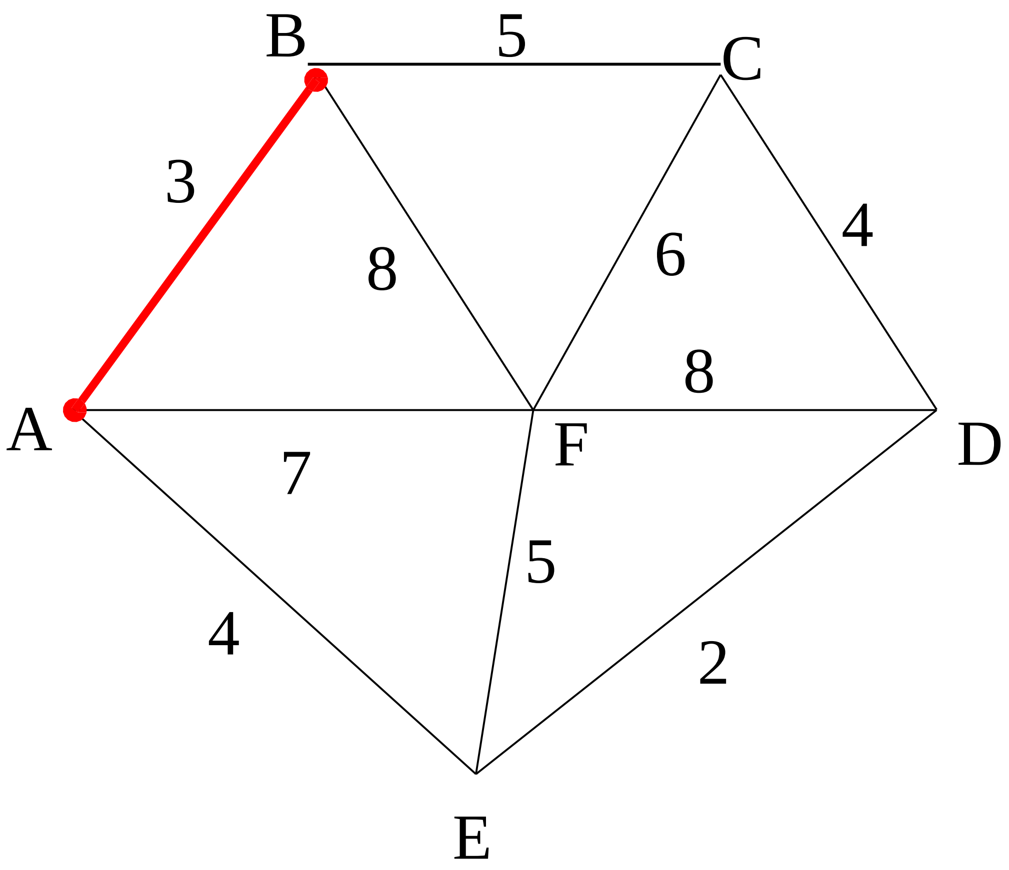
EF 5

Total weight of tree: 18

Prim’s Algorithm:

MST-PRIM(G, w, r)
1. for each u   V [G]
2.   do key[u] ← ∞
3. π[u] ← NIL
4. key[r] ← 0
5. Q ← V [G]
6. while Q ≠ Ø
7.   do u ← EXTRACT-MIN(Q)
8.   for each v Adj[u]
9.   do if v Q and w(u, v) < key[v]
10.   then π[v] ← u
11.   key[v] ← w(u, v)

Prim’s Algorithm:

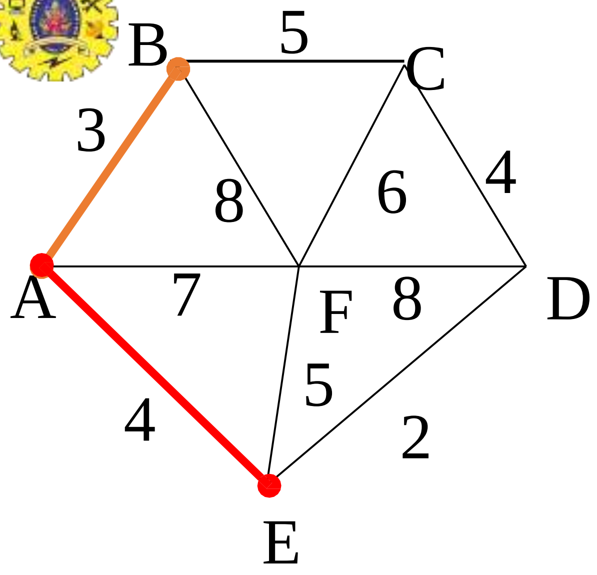


Select any vertex

A

Select the shortest edge connected to that vertex

AB 3

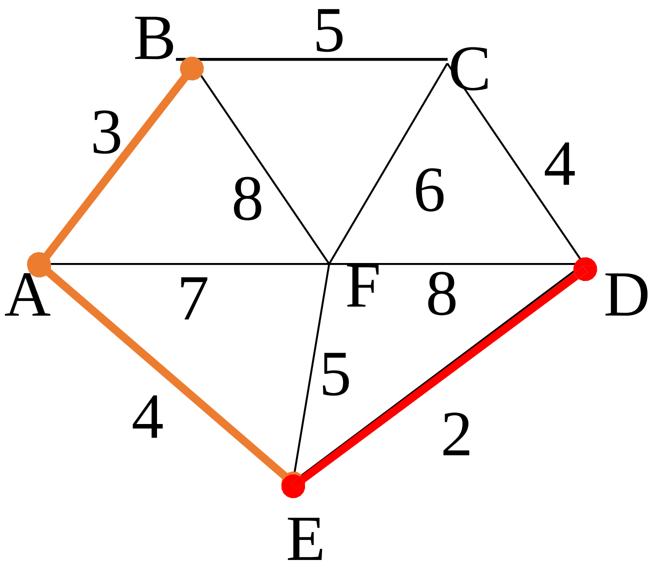


Select the shortest edge connected to any vertex already connected.

AE 4

1

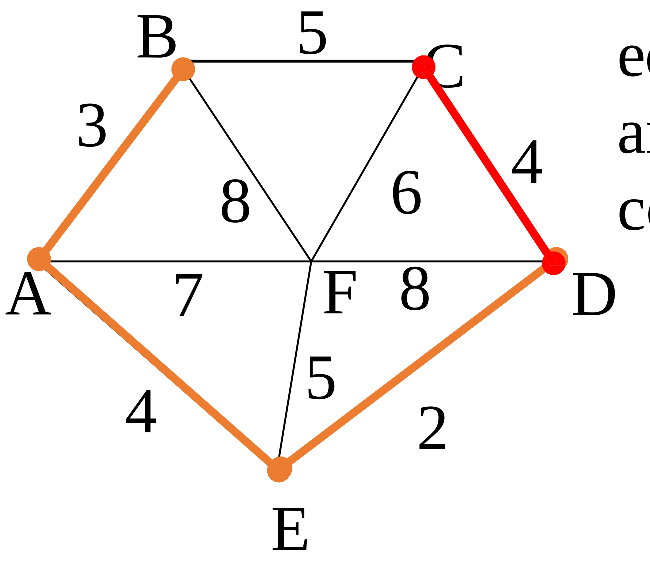
2



Select the shortest edge connected to any vertex already connected.

ED 2

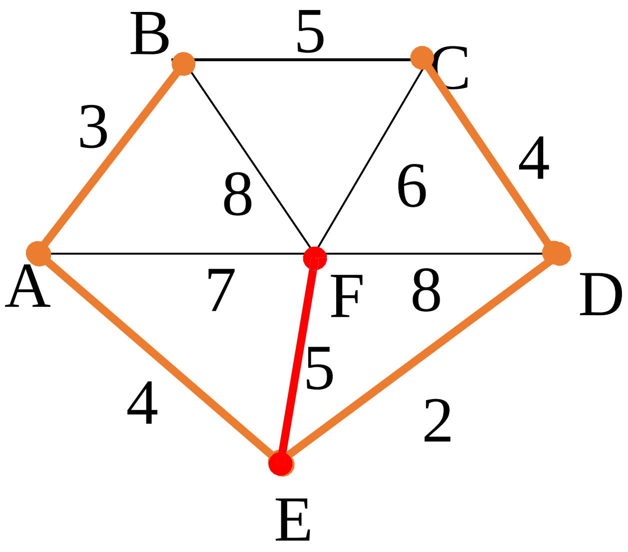
3



Select the shortest edge connected to any vertex already connected.

DC 4

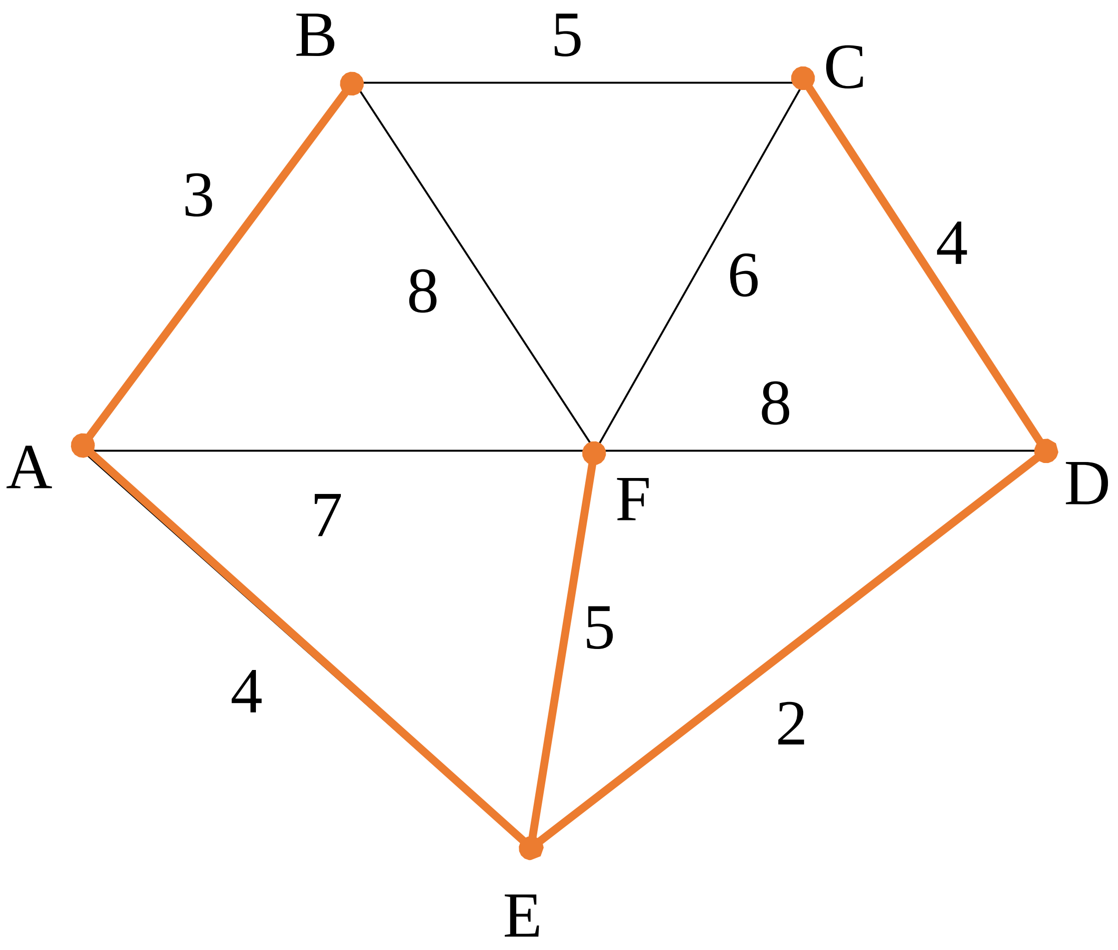
4



Select the shortest edge connected to any vertex already connected.

EF 5

Prim’s Algorithm:



All vertices have been connected.

The solution is

AB 3
AE 4
ED 2
DC 4
EF 5

Total weight of tree: 18

Greedy Algorithms:

There are some methods left:

- **Dijkstra’s algorithm**
- Huffman’s Algorithm
- Task scheduling
- Travelling salesman Problem etc.
- Dynamic Greedy Problems

We can find the optimized solution with Greedy method which may be optimal sometime.

THANK YOU