



Design and Analysis of Algorithms

Lecture – 18

Fractional Knapsack Problem & Minimum Cost Spanning Tree

Success is always inevitable with Hard Work and Perseverance

N. Ravitha Rajalakshmi

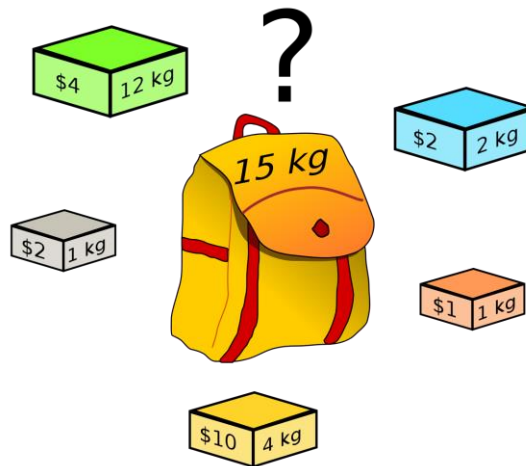
Learning Objective

- Knapsack Problem
- When Greedy is applicable?
- Spanning Trees

Fractional Knapsack Problem

There are n items with weights $w_1, w_2, w_3 \cdots w_n$ and values $v_1, v_2, v_3 \dots v_n$ and a bag with capacity

What will be the **maximum total value of fraction of items** that fit into a bag with capacity W .

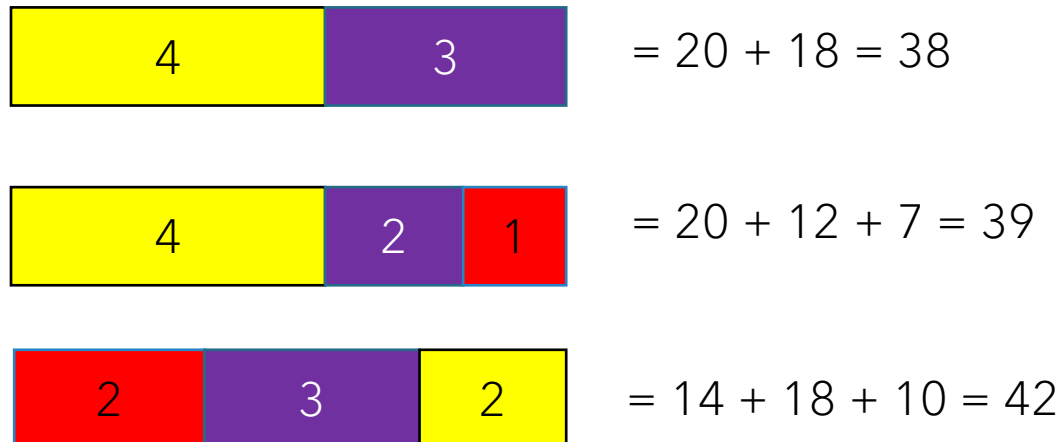


Example



Consider a knapsack with Weight W as 7

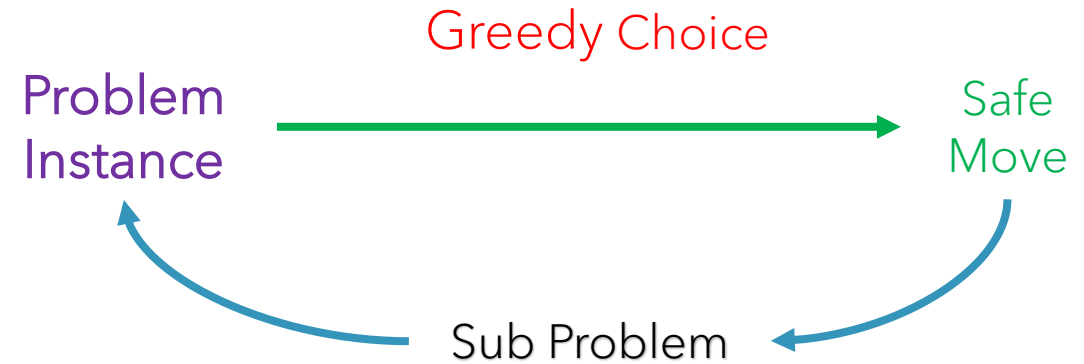
Different ways to fill the knapsack



Greedy Strategy

- Solution is built in stages

1. Make a greedy choice
2. Reduce the problem
3. Iterate until the problem can directly be solved



A safe move is a move which is consistent with some optimal solution

Pause & Think

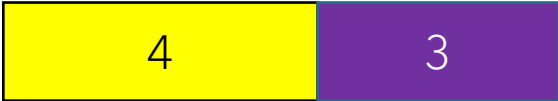
Which Greedy Choice offers a safe move?

Possible Greedy Choice




1. Choose the item with maximum weight
2. Choose the item with maximum cost
3. Choose items with maximum profit (maximum value per unit weight)

Proof

There exists an optimal solution that uses as much as possible of an item with maximal value per unit of weight


$$= 20 + 18 = 38$$


$$= 14 + 10 + 18 = 42$$

Weight	Value	Value per unit weight
	20	5
	18	6
	14	7

Function FillKnapsack(w, v, n, W)

#w and v weight and value of the n items

value = 0 , weight = 0, l = 0

sort(w, v) based on the profit v_i/w_i

while(weight < W){

if(w[i] + weight < W){

value = value + v[i]

weight = weight + w[i]

}

else{

*value = value + (W - weight) * v[i]/w[i]*

weight = W

}

i++

}

Time Complexity

Input Size : n

Basic Operation :

Assignment (sort) , Addition
(Update value and weight)

$T(n) = \text{Sort } O(n \log n) +$
 $\text{Addition } O(n) = O(n \log n)$

Pause & Think

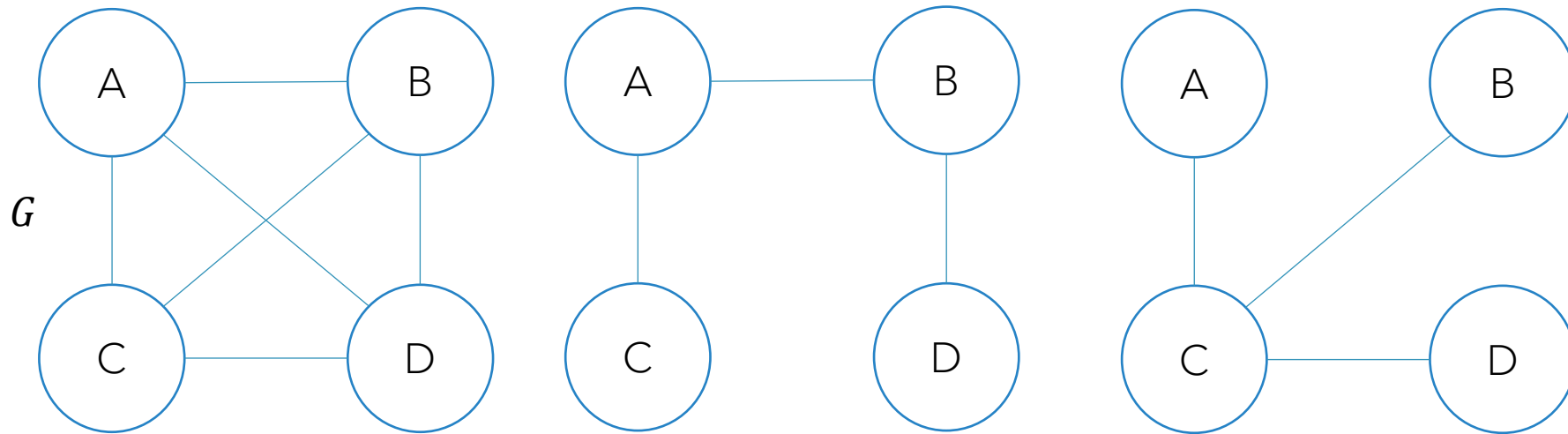
Can Greedy Choice offers a safe move in all problems?

Consider the problem of knapsack where the items cannot be taken in fraction. Do you think the greedy strategy which we discussed before will work?

Consider 3 items $w = \{5, 3, 3\}$ $v = \{10, 9, 9\}$ and $W = 5$

Spanning Trees

Given a undirected connected graph G , spanning tree represents a subset of G which contains all the vertices in the graph with a minimum possible number of edges.



Pause & Think

- In a connected graph with n vertices , how many edges will be there?

$$\frac{n(n-1)}{2}$$

- What will be the minimum number of edges in a spanning Tree?

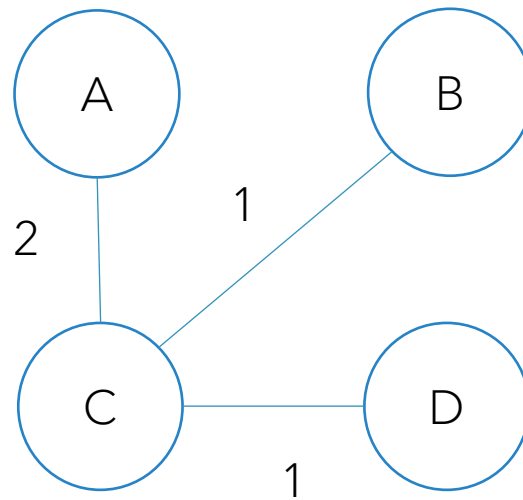
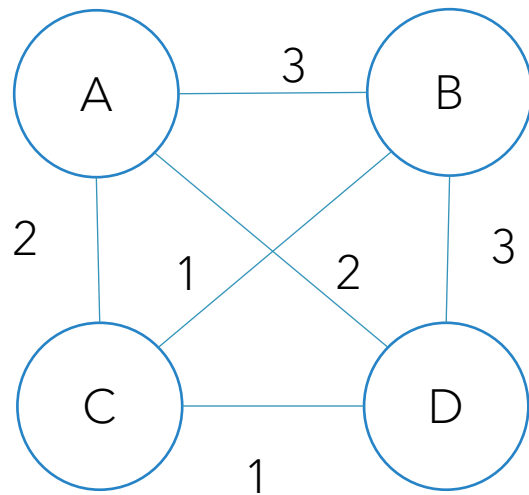
$$n - 1$$

- Can a tree contain a cycle?

No

Minimum cost spanning tree

- There are $n^{(n-2)}$ possible spanning trees for a graph with vertices n
- If the edges are labelled with a cost / weight , then is it possible to find minimum cost spanning tree



Greedy Strategy (Kruskal's Algorithm)

- The minimum cost spanning tree contain edges with lesser cost
- Choose the edges as long as they do not form a cycle

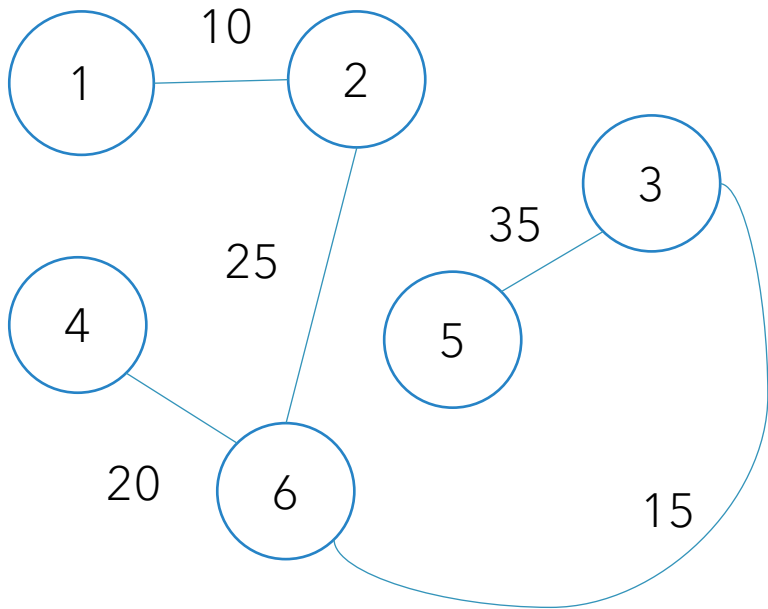
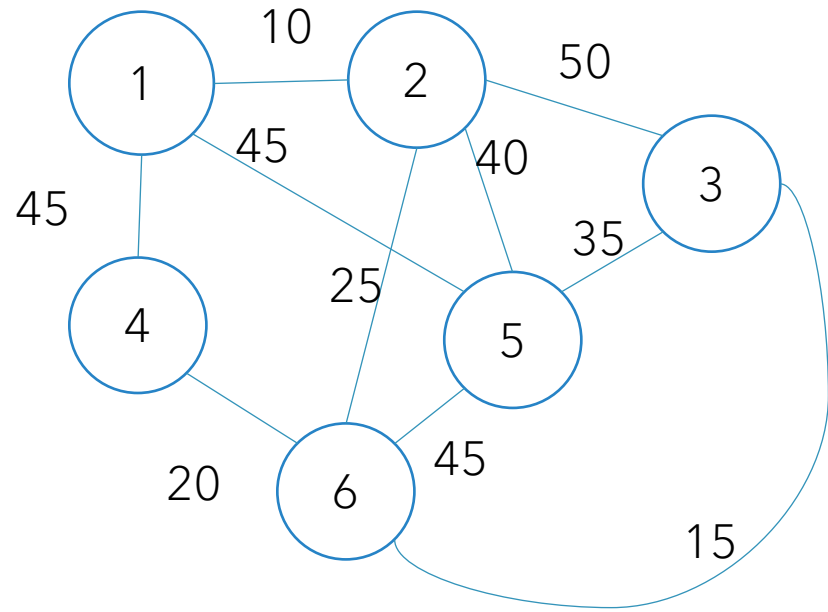
Algorithm:

Edges are sorted according to weights [Use heap]

Least cost edge is initially added to set

While (number of edges $< n-1$)

 Add edges to tree if they do not form a cycle



Edges	Cost	Forests
		{1}{2}{3}{4}{5}{6}
{1 2}	10	{12}{3}{4}{5}{6}
{3 6}	15	{1 2}{3 6}{4}{5}
{4 6}	20	{1 2}{3 4 6}{5}
{2 6}	25	{1 2 3 4 6}{5}
{1 4}	30	Reject
{3 5}	35	{1 2 3 4 5 6}

Total cost = 105

Pause & Think

- Kruskal's is implemented using special kind of data structure known as disjoint sets
- Union and Find are the two major operations
- Union – Joins two sets , their representative is updated
- Find – returns representative for a element

- Using disjoint set, when will you reject an edge ?
Edge endpoints belong to same set

Function `kruskal(E, cost, n, t)`

#E is set of edges in G

#cost [u,v] cost of edge (u,v)

t set of edges with min cost spanning tree

Construct a Heap with the edge costs

i = 0 mincost = 0

```
while (i < n-1 and heap is not empty) {  
    delete edge (u,v) with minimum edge  
    j = Find(u) k = Find(v)  
    if(j ≠ k){  
        t[i-1,0] = u t[i-1,1] = v  
        i = i+1  
        mincost = mincost + cost[u,v]  
        Union(j,k)  
    }  
}
```

Time Complexity

Input Size : V (vertices)

E(edges)

Basic Operation :

Construction of heap

Delete elt from heap

Find and Union

Operation

$T(n)$

$= \text{Heap 0}(E)$

$+ E(\log E(\text{delete}))$

$+ \log E(\text{Find and Union}))$

$= O(E \log E)$

Pause & Think

- How do we detect cycle in Kruskal

For an edge (u,v) if both u and v belong to the same set

- What will be intermediate result of Kruskal's algorithm

Forest

- How do you find whether spanning tree exists for a graph?

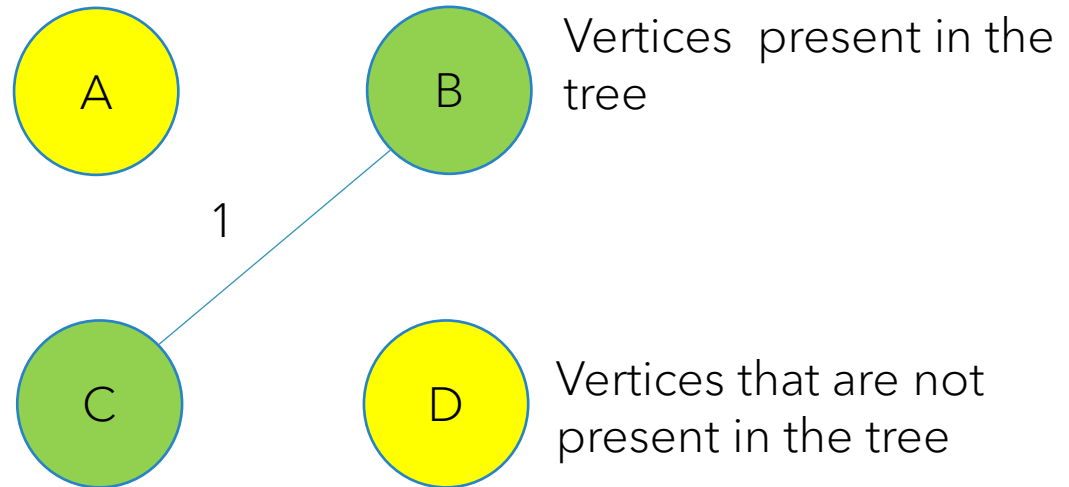
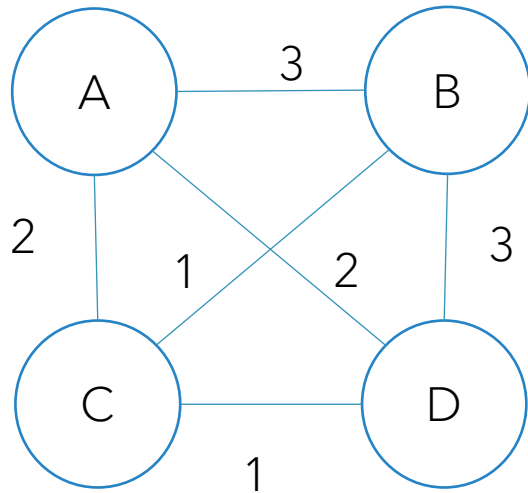
After all the edges are considered

if the tree contains edges less than $n-1$ in the tree, then it does not form a spanning tree

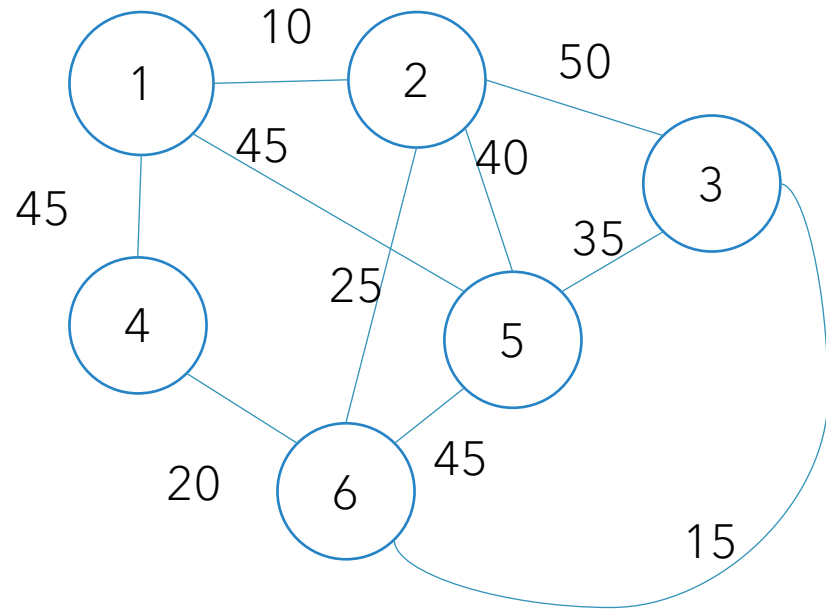
Prim's Algorithm

- It does not consider all the edges for the selection of an edge
- It expands the tree by only considering the edge that connects the vertices that are in the tree with the vertices that are not in the tree

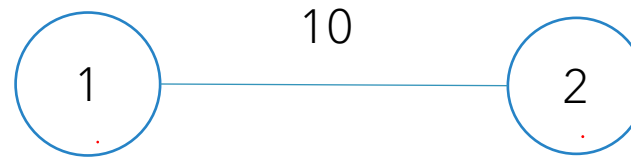
This prevents the cycle in a tree !!



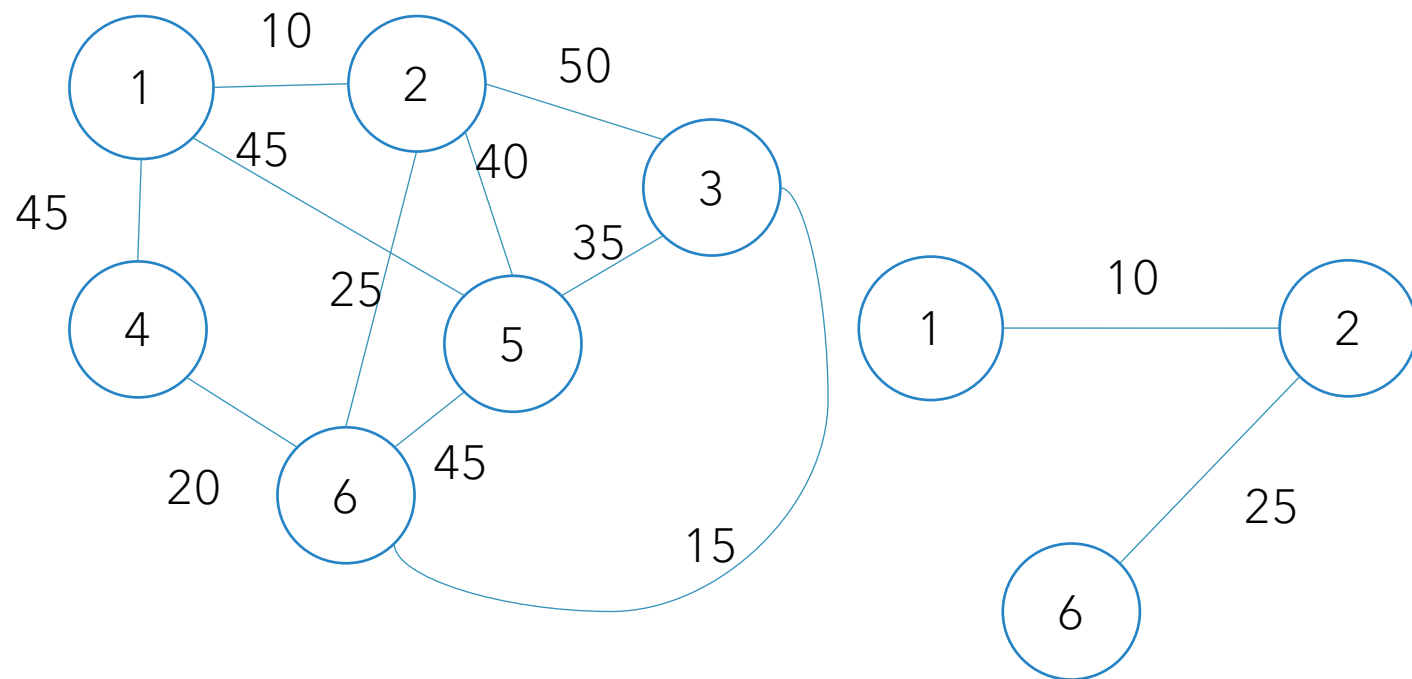
Prim's Algorithm



Vertex	Nearby Vertex	Cost
1	0	
2	0	
3	2	50
4	1	45
5	2	40
6	2	25

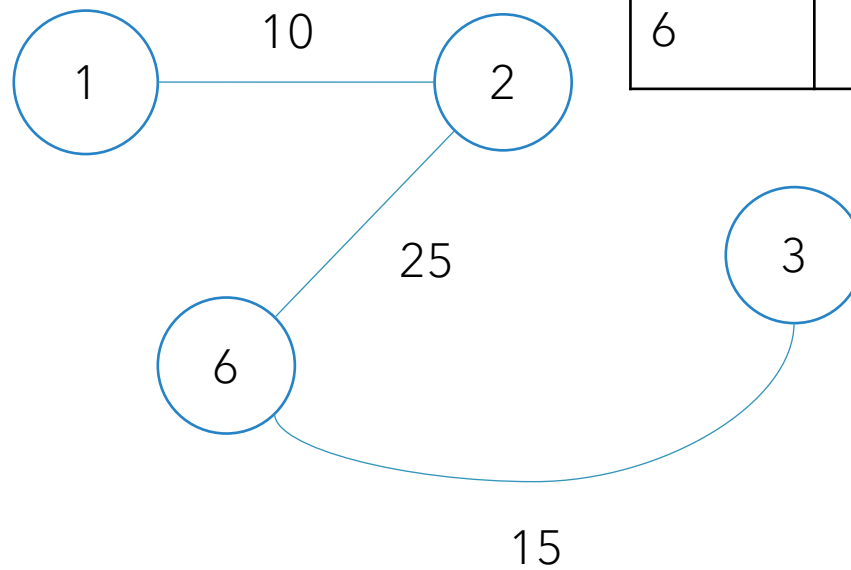
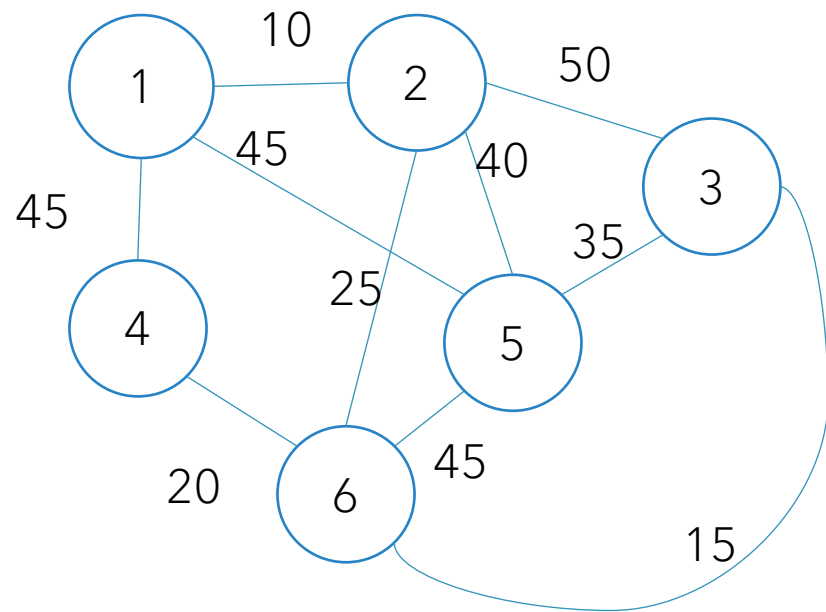


Prim's Algorithm



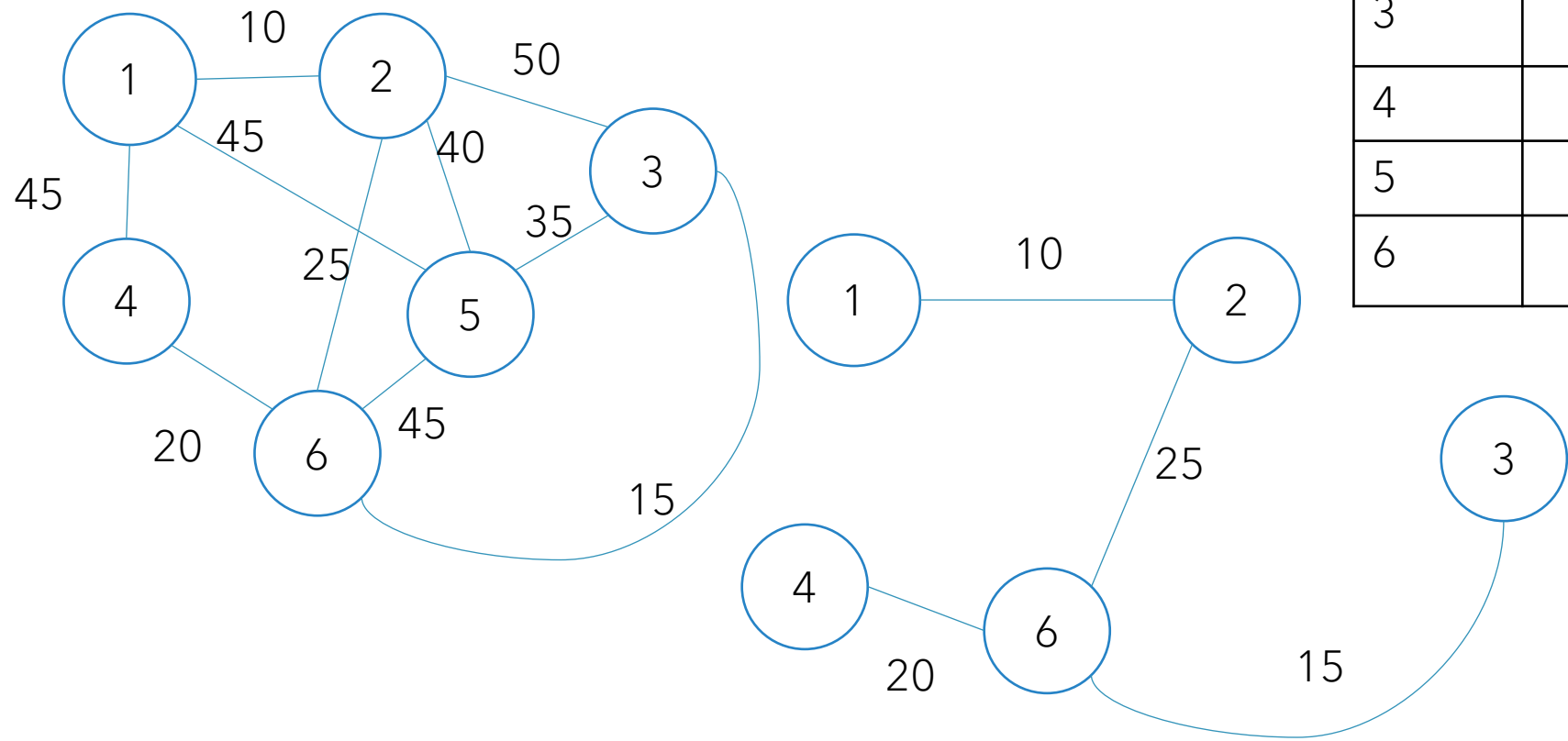
Vertex	Nearby Vertex	Cost
1	0	
2	0	
3	6	15
4	6	20
5	2	40
6	0	

Prim's Algorithm



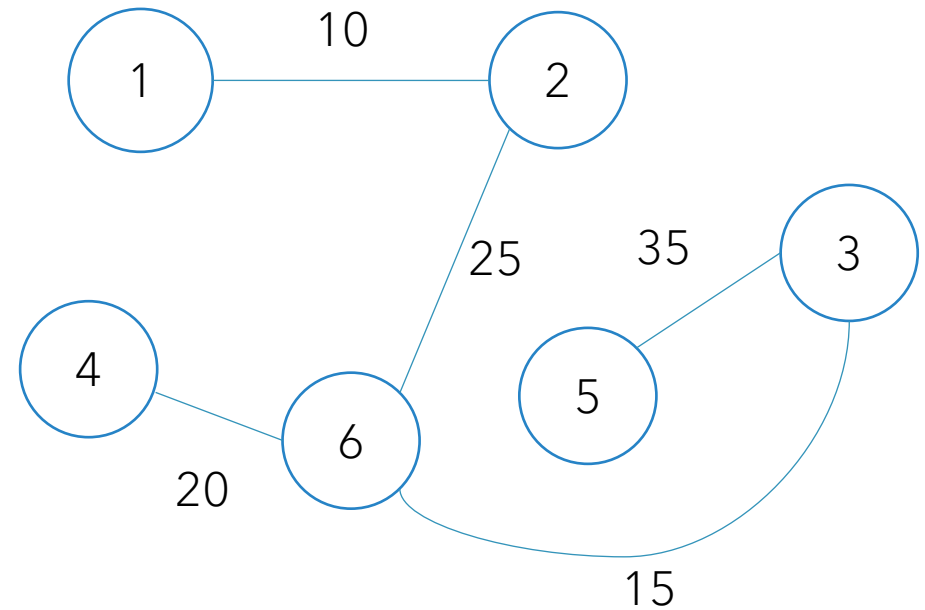
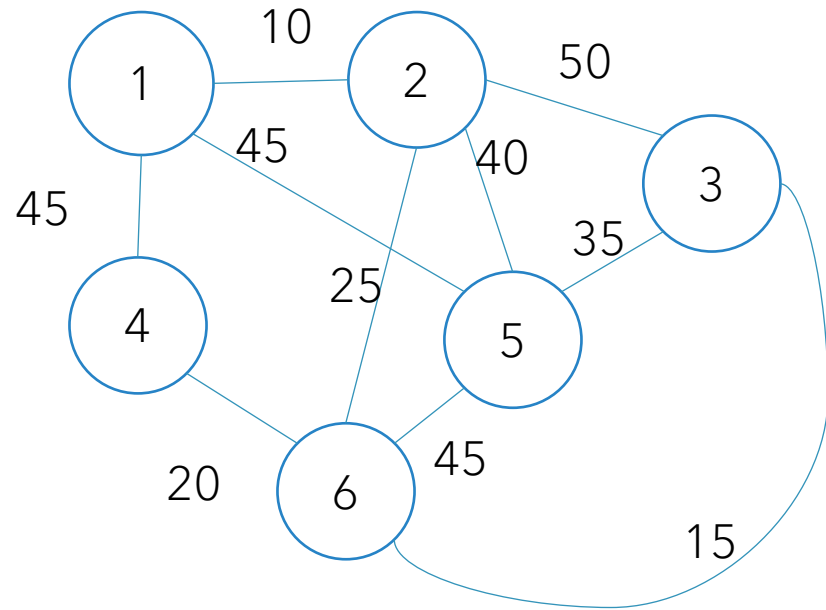
Vertex	Nearby Vertex	Cost
1	0	
2	0	
3	0	
4	6	20
5	3	35
6	0	

Prim's Algorithm



Vertex	Nearby Vertex	Cost
1	0	
2	0	
3	0	
4	0	
5	3	35
6	0	

Prim's Algorithm



Pause & Think

- How do we detect cycle in Prim's

Consider edges that connect the tree to vertices not in the tree

This prevents the cycle

Function Prim(E, cost, n, t)

#E is set of edges in G

#cost [u,v] cost of edge (u,v)

t set of edges with min cost spanning tree

#near array which contains the nearby vertex

i = 0 mincost = 0

#initialization of near vertex

Let (k,l) be the edge with min cost

near[k] = 0 near[l] = 0

t[i][0] = k t[i][1] = l i = i+1

for i = 1 to n-1 :

if(cost[i, k] < cost[i, l])

near[i] = k

else

near[i] = l

```

while ( i < n-1){
    Find vertex j such that near[j] ≠ 0 and cost[j,near[j]] is minimum
    t[i][0] = j t[i][1] = near[j] i = i+1
    mincost = mincost + cost[j, near[j]]
    near[j]=0
    for k = 0 to n-1
        if(near[k] ≠ 0 and cost[k,near[k]]>cost[k,j])
            near[k] = j
    }

```

Time Complexity

Input Size : V (vertices) E(edges)

Basic Operation : Find vertex and Update near matrix
 $O(n^2)$

Summary

- Discussed about Fractional Knapsack and Spanning Trees

Thank You
Happy Learning

Success is always inevitable with Hard Work and Perseverance