Greedy strategy

Dr. Priyadarshan Dhabe,
Ph. D (IIT Bombay),
Professor in Information Technology

Greedy algorithms –General strategy

- A ***greedy algorithm*** always makes the choice that looks best at the moment. That is, it makes a locally optimal choice in the hope that this choice will lead to a **globally optimal** solution.
- For some **optimization problems** greedy algorithms provide optimal solutions.
- Greedy algorithms do not always yield optimal solutions, but for many problems they do.
- The greedy method is quite powerful and works well for a wide range of problems like knapsack, Huffman coding, shortest path, job sequencing and minimum spanning tree.

Fractional knapsack problem-greedy approach

- We have been given n objects and a knapsack with capacity of m kg, select the objects (fraction also) such that we can get **maximum profit.**

n=7 objects and capacity M=15

Objects	1	2	3	4	5	6	7
Profit-p	10	5	15	7	6	18	3
Weight-w	2	3	5	7	1	4	1

$$\text{generally} \sum w_i > 15, \text{ thus cant put all objects}$$

https://www.youtube.com/watch?v=oTTzNMHM05I

Fractional knapsack problem-greedy approach

- **Greedy method working**
  - 1. Computes ratio (p/w) for each object
  - 2. Picks objects/fraction from highest to lower (p/w), such that knapsack becomes full

Objects	1	2	3	4	5	6	7
Profit-p	10	5	15	7	6	18	3
Weight-w	2	3	5	7	1	4	1
p/w	**5**	**1.66**	**3**	**1**	**6**	**4.5**	**3**

Fractional knapsack problem-greedy approach

- We will use a vector to denote which object is picked or not and has the following format

X=	x1	x2	x3	x4	x5	x6	x7

$$\text{where } 0 \le x_i \le 1$$

0 - Object not picked

1 - Object completely picked

0.5 - Object picked 50 %

Fractional knapsack problem-greedy approach

Objects	1	2	3	4	5	6	7
Profit-p	10	5	15	7	6	18	3
Weight-w	2	3	5	7	1	4	1
p/w	**5**	**1.66**	**3**	**1**	**6**	**4.5**	**3**

X=	-	-	-	-	1	-	-

15 kg

15-1=14

X=	1	-	-	-	1	-	-

15 kg

15-1=14
14-2=12

Fractional knapsack problem-greedy approach

Objects	1	2	3	4	5	6	7
Profit-p	10	5	15	7	6	18	3
Weight-w	2	3	5	7	1	4	1
p/w	**5**	**1.66**	**3**	**1**	**6**	**4.5**	**3**

X=	1	-	-	-	1	1	-

15 kg

15-1=14
14-2=12
12-4=8

X=	1	-	1	-	1	1	1

15-1=14
14-2=12
12-4=8
8-5=3

15 kg

Fractional knapsack problem-greedy approach

Objects	1	2	3	4	5	6	7
Profit-p	10	5	15	7	6	18	3
Weight-w	2	3	5	7	1	4	1
p/w	**5**	**1.66**	**3**	**1**	**6**	**4.5**	**3**

X=	1	-	1	-	1	1	1

15-1=14
14-2=12
12-4=8
8-5=3
3-1=2

15 kg

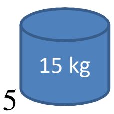
Fractional knapsack problem-greedy approach

Objects	1	2	3	4	5	6	7
Profit-p	10	5	15	7	6	18	3
Weight-w	2	3	5	7	1	4	1
p/w	**5**	**1.66**	**3**	**1**	**6**	**4.5**	**3**

Now highest(p/w) is of object2, but it has weight of 3 kg and thus cant take fully. Opt for 2 kg/3 kg of object2

X=	1	2/3	1	0	1	1	1

15-1=14
14-2=12
12-4=8
8-5=3
3-1=2
2-2=0



$$\text{Total weight w} = \sum \mathrm{x_i} w_i$$

$$w = 1\times 2 + (2/3\times 3) + 1\times 5 + 0\times 7 + 1\times 1 + 1\times 4 + 1\times 1 = 15$$

$$\text{Total profit p} = \sum \mathrm{x_i} p_i$$

$$P = 1\times 10 + (2/3\times 5) + 1\times 15 + 0 + 1\times 6 + 1\times 18 + 1\times 3 = 55.33$$

Fractional knapsack problem-greedy approach

Constraint

$$\sum \mathrm{x_i} w_i \le m = 15$$

Objective function

$$\sum_{\max} \mathrm{x_i} p_i$$

Fractional knapsack problem

- N=3, (objects), M=50 (knapsack capacity)
- P={60,100,120}
- W={10,20,30}
- **Ans:- 240 x={1,1,20/30}**

Job sequencing/scheduling problem with deadlines -greedy approach

- **The sequencing of jobs on a single processor/machine with deadline constraints is called as Job Sequencing with Deadlines.**
- **Here-**
- **You are given a set of jobs.**
- **Each job has a defined deadline (waiting time) and some profit associated with it.**
- **The profit of a job is given only when that job is completed within its deadline.**
- **Only one processor/machine is available for processing all the jobs.**
- **Processor takes one unit of time to complete a job.**

- **The problem - "How can the total profit be maximized if only one job can be completed at a time?"**

Job sequencing problem with deadlines-greedy approach

Jobs	j1	j2	j3	j4	j5
Profit	20	15	10	5	1
deadline	2	2	1	3	3

ordered

Assumptions/Information

1. Prepare **Gantt chart** up to maximum deadline
2. Each job need **1 Hr or 1 slot** to complete
3. Each job has given deadline and need to be completed in it
4. **Job j5** can wait for **3 Hrs or 3 slots**
5. Schedule the jobs within the **deadline** and with **max profit to minimum one (decreasing order).**
6. Not all jobs can be scheduled
7. Jobs are arranged in **decreasing order** of profit
8. All the jobs are submitted to the system at the same time

Job sequencing problem-greedy approach

Jobs	j1	j2	j3	j4	j5
Profit	20	15	10	5	1
deadline	2	2	1	3	3

We can have max 3 slots=max of all deadlines

Job slots

```
   j2      j1       j4
0------1-------2------3
```

Gantt chart

Possible Job sequencing can be
(j2->j1->j4)= profit=15+20+5=40
(j1->j2->j4)= profit=20+15+5=40

Job sequencing problem-greedy approach

Jobs	j1	j2	j3	j4	j5
Profit	20	15	10	5	1
deadline	2	2	1	3	3

j2 j1 j4

0------1-------2------3

Job	Slot assigned	solution	Profit
-	-	Empty	0
j1	[1,2]	j1	20
j2	[0,1][1,2]	J1,j2	20+15
j3 x	[0,1][1,2]	J1,j2	20+15
j4	[0,1][1,2] [2,3]	J1,j2,j4	20+15+5
j5 x	[0,1][1,2] [2,3]	J1,j2,j4	20+15+5

Max Profit=40

Job sequencing problem-greedy approach

Time complexity for n jobs : -

1. Brute force approach we need to check all possible subsets of given set of n jobs, thus, it is of $O(2^n)$.
2. Using greedy approach, the max work done is O(nxm) for m < n slots. Each job need to searched for all the slots. There can be maximum $m = (n-1)$ slots, thus $O(nxm) = O(n \text{ x } (n-1)) = O(n^2)$

Job sequencing problem-greedy approach

- Example to solve with n=7 jobs

Jobs	j1	j2	j3	j4	j5	j6	j7
Profit	35	30	25	20	15	12	5
deadline	3	4	4	2	3	1	2

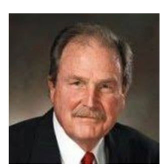
Find possible job sequencing and max profit

Slot allocation-J4-j3-j1-j2
Profit=20+25+35+30=110

https://www.youtube.com/watch?v=zPtI8q9gvX8

Huffman coding-greedy approach

- This code is proposed by ***David A. Huffman***.
- He was a Sc.D. student at MIT, and published his paper in 1952 titled "*A Method for the Construction of Minimum-Redundancy Codes*"
- This code is variable length and lossless compression code.



https://en.wikipedia.org/wiki/Huffman_coding

Huffman coding-greedy approach

- **Huffman coding** is a **variable length coding greedy** approach, where each character in a message is written with minimum number of bits so that whole message can be transmitted using fewer bits.
- The basic idea is that to compute frequency of appearance of each character and assign **lesser** number of bits to **more frequently** used character.
- We need to construct **Huffman coding tree** to decide codes of each character in the message.
- Since we are finding most frequently used character and assigning it minimum number of bits, method is **greedy method.**
- **It is lossless compression technique**

Huffman coding-greedy approach

- There are **two major steps** in Huffman Coding-
  1. **Building a Huffman Tree** from the input characters.
  2. **Assigning code** to the characters by traversing the Huffman Tree.
- The steps involved in the construction of **Huffman Tree** are as follows-
- **Step-01:**
- Create a leaf node for each character of the text.
- Leaf node of a character contains the occurring frequency of that character
- **Step-02:**
- Arrange all the nodes in **increasing order** of their frequency value.

https://www.gatevidyalay.com/huffman-coding-huffman-encoding/#:~:text=Huffman%20Coding%20is%20a%20famous,It%20uses%20variable%20length%20encoding.

Huffman coding-greedy approach

- **Step-03:**
- Considering the first two nodes having minimum frequency,
- Create a new internal node.
- The frequency of this new node is the sum of frequency of those two nodes.
- Make the first node as a left child and the other node as a right child of the newly created node.
- **Step-04:**
- Keep repeating Step-02 and Step-03 until all the nodes forms a single tree.
- The tree finally obtained is the desired **Huffman Tree**.

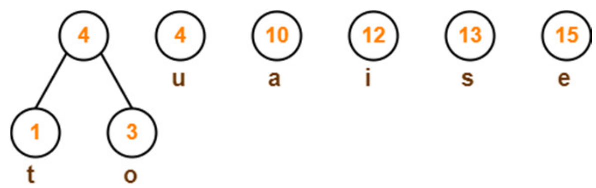
Huffman coding-greedy approach

Characters	Frequencies
a	10
e	15
i	12
o	3
u	4
s	13
t	1

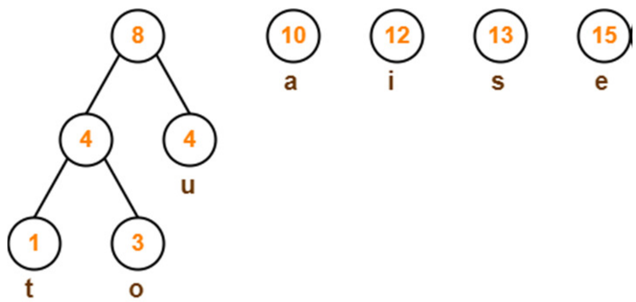
Step-01:

Huffman coding-greedy approach

Step-02:

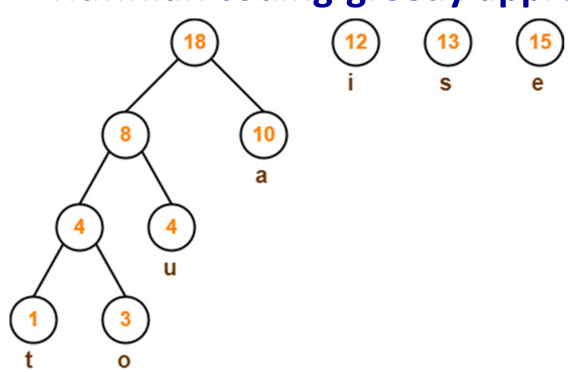


Step-03:

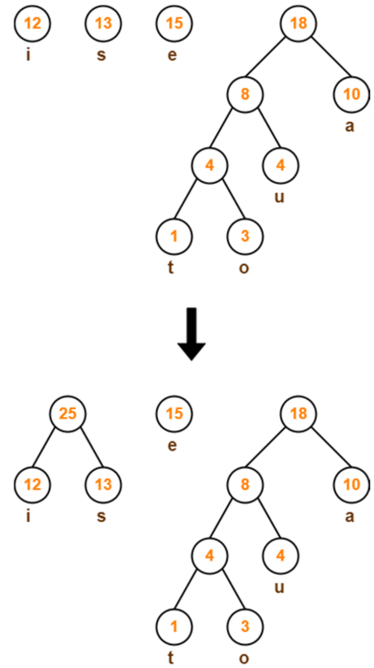


Huffman coding-greedy approach

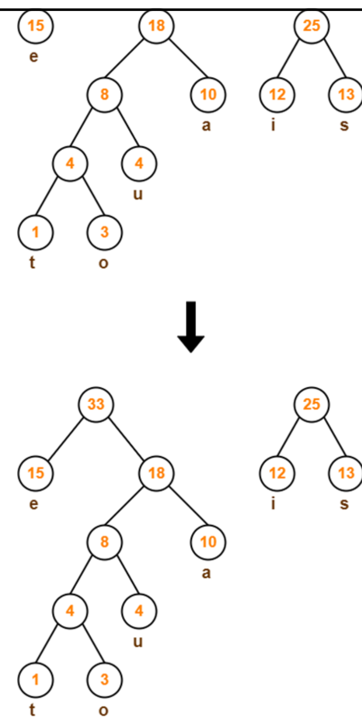
Step-04:



Step-05:

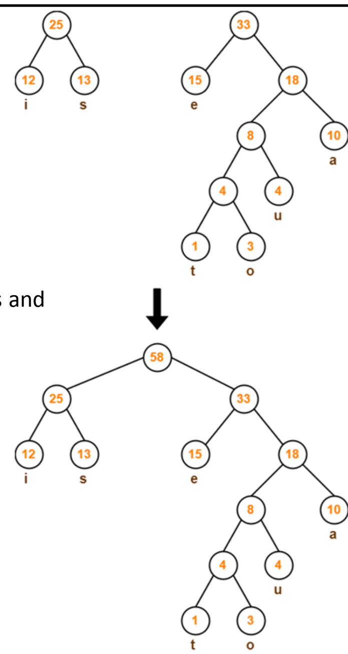


Step-06:

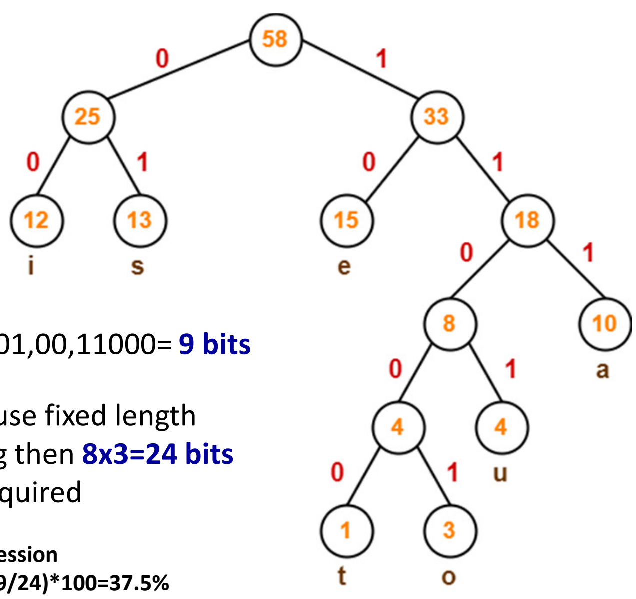


Step-07:

1.We assign weight to all the edges of the constructed Huffman Tree.
2. Let us assign weight ‘0’ to the left edges and weight ‘1’ to the right edges.



Huffman Tree



Sit=01,00,11000= **9 bits**

If we use fixed length coding then **8x3=24 bits** are required

Compression ratio=(9/24)*100=37.5%

Huffman Tree

Huffman coding-greedy approach

- For a **n- character** message, we need to select two nodes with minimum value **(n-1)** times and let **log n** be the height of the binary tree (**hosting min heap**), then the total time is
- **(n-1)* log n**, thus **O(n log n).**

https://www.youtube.com/watch?v=UbYLUmYazTM

https://www.youtube.com/watch?v=NCuaebwQLKU

Huffman coding-greedy approach-problem

Characters and their frequencies

A	35
B	25
C	20
D	12
E	8

Find the code for ABCDCEE

Huffman coding-greedy approach-problem

Characters and their frequencies

A	35
B	25
C	20
D	12
E	8

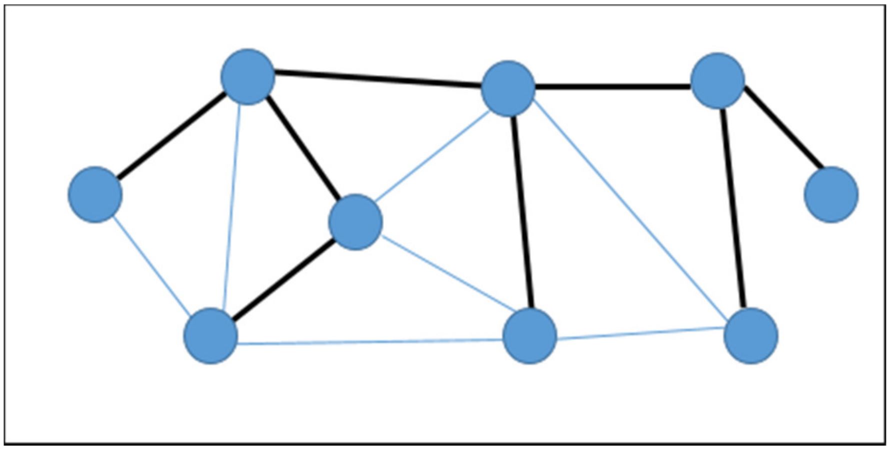
Characters and their codes

Character	Prefix Code
A	11
B	001
C	000
D	10
E	01

Minimum spanning tree-Greedy approach

- A **spanning tree** is a **subset** of an undirected Graph that has all the vertices connected by **fewer number of edges**.
- If all the vertices are connected in a graph, then there exists at **least one** spanning tree. In a graph, there may exist **more than one** spanning tree.
- **Properties of spanning tree**
  1. A spanning tree does not have any **cycle**.
  2. Any vertex can be **reached** from any other vertex.

Spanning tree is shown by highlighted (black) edges



Graph and spanning tree

Spanning tree definition

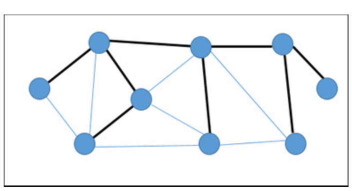
Graph $G = (V, E)$

V - set of vertices and E - set of edges

Spanning tree $T = (V^{'}, E^{'})$ such that

$$V^{'} = V \text{ and } E^{'} \subset E$$

T contains $|V| - 1$ edges



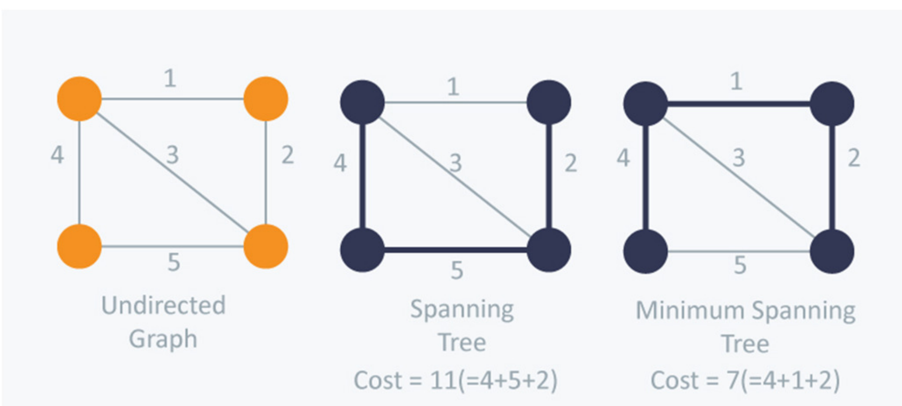
Minimum spanning tree (MST) definition

- A **Minimum Spanning Tree (MST)** is a subset of edges of a connected **weighted** undirected graph that connects **all the vertices** together with the **minimum** possible **total edge weight.**
- There are **two algorithms** to derive MST using **greedy approach**

   1.Prim's algorithm

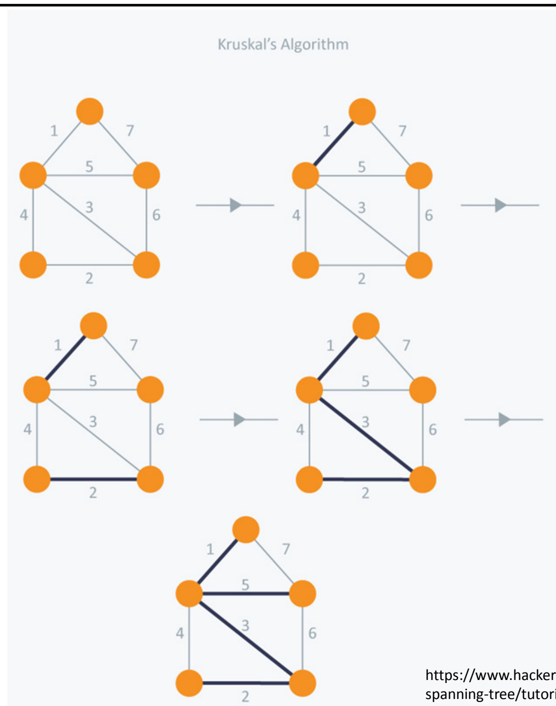
   2. Kruskal's algorithm

Minimum spanning tree definition



Minimum spanning tree (MST) –Kruskal's Algorithm -Greedy approach

- This algorithm first appeared in *Proceedings of the American Mathematical Society*, pp. 48–50 in 1956, and was written by Joseph Kruskal.
- **Kruskal's Algorithm** builds the spanning tree by **adding edges** one by one into a growing spanning tree.
- **Kruskal's algorithm** follows greedy approach as in each iteration it finds an edge which has **least weight** and add it to the growing spanning tree.
- **Algorithm Steps:**
  - **Sort** the **graph edges** with respect to their **weights**.
  - Start adding edges to the MST from the edge with the smallest weight until the edge of the largest weight.
  - Only add edges which **doesn't form a cycle** & edges which connect only **disconnected components**.



Kruskal's Algorithm

https://www.hackerearth.com/practice/algorithms/graphs/minimum-spanning-tree/tutorial/

Minimum spanning tree (MST) –Kruskal's Algorithm -Greedy approach

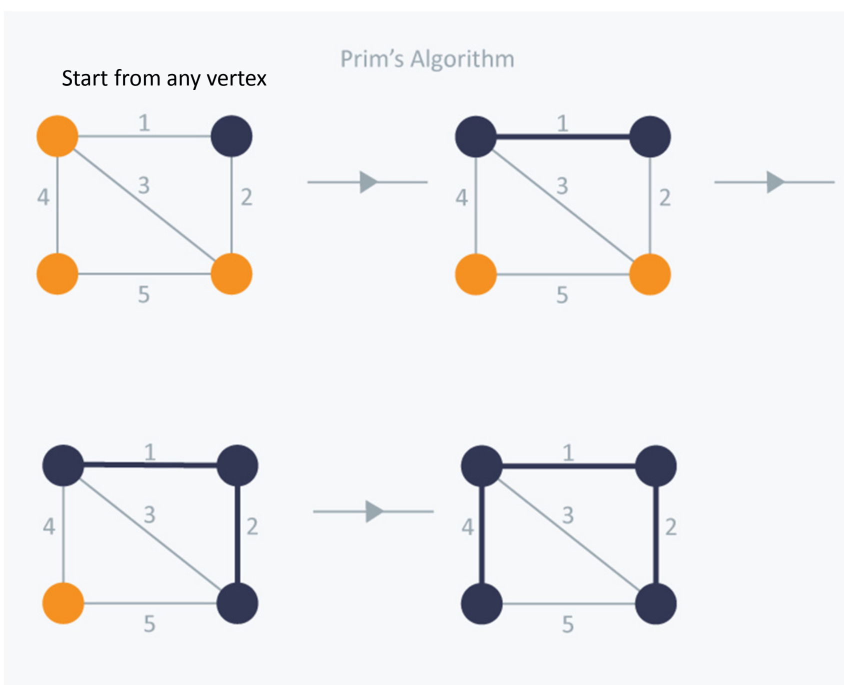
- Time complexity of Kruskal's algorithm- Most of the time is required in sorting the edges, thus it is O(E log E) (sorting time of merge sort O(n log n)).

Minimum spanning tree (MST) –Prim's Algorithm -Greedy approach

- **Prim's Algorithm** also use **Greedy approach** to find the minimum spanning tree.
- In Prim's Algorithm we grow the spanning tree from a starting position. Unlike an **edge** in Kruskal's, **we add vertex to the growing spanning tree in Prim's algorithm.**
- Also called as Jarnik's algorithm and proposed in 1930 (wikipedia) and republished by **Robert C. Prim** in 1957

(MST) –Prim's Algorithm-Greedy approach

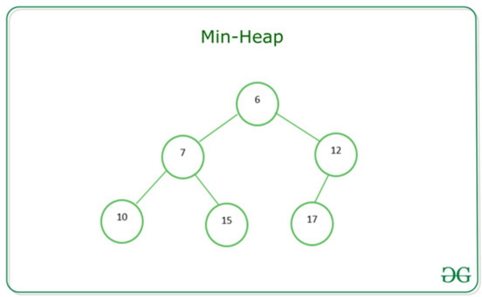
- **Algorithm Steps:**
- Maintain two **disjoint sets** of vertices, say **A and B**. One containing vertices that are **in the** growing spanning tree (A) and other that are **not in the** growing spanning tree (B). **(A intersection B is empty)**
- Select the **cheapest vertex** that is connected to the growing spanning tree and is not in the growing spanning tree and add it into the growing spanning tree. This can be done using **Priority Queues**.
- Check for cycles and connected components.



(MST) –Prim's Algorithm-Greedy approach

- **Time complexity:-** If we implement Priority queue to find the cheapest vertex using ***minheap*** then it is O(E log V) (For each edge E x time to reorder items in heap)

A min-heap is a binary tree such that - the data contained in each node is less than (or equal to) the data in that node's children.

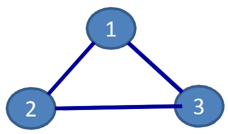


Number of possible spanning trees from a given complete graph

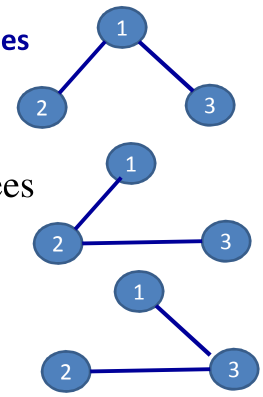
- A **complete graph** is undirected graph, in which every pair of distinct vertices are connected by a unique edge.

Total number of spanning trees possible for a complete graph with n vertices $= n^{(n-2)}$

Find and draw the number of possible spanning trees from a given complete graph

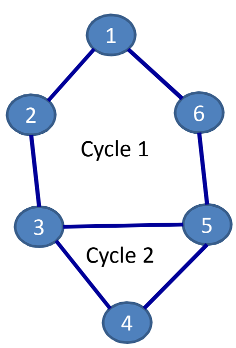


3 spanning trees



$$n = 3, \text{ thus} = n^{(n-2)} = 3^{(3-2)} = 3 \text{ spanning trees}$$

Number of possible spanning trees from a given non-complete graph



The number of spanning trees possible for a non-complete graph $G = (V, E)$ is given as

$$^{|E|}C_{(|V|-1)} - \text{Number of cycles k in G}$$

(the no of ways of selecting $(|V|-1)$ edges out of E edges - internal cycles k)

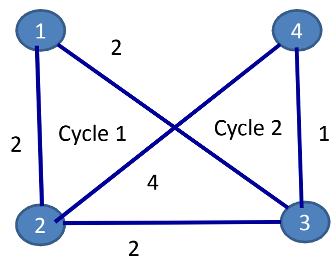
where $^{n}C_{r} = \dfrac{n!}{r!(n-r)!}$

For the graph shown $|E| = 7$ and $|V| - 1 = 6 - 1 = 5$

Thus, will have $\left(^{7}C_{5} - k\right)$

$$= \left(\frac{7!}{5!(7-5)!} - 2\right) = \left(\frac{7*6}{2!} - 2\right) = 19$$

Find the no. of possible spanning trees from a given non-complete graph



$$\text{no of spanning trees} =$$
$$^{|E|}C_{(|V|-1)} - \text{Number of cycles k in G}$$
$$|E| = 5, |V| = 4 \text{ and } k = \text{cycles} = 2$$

$$= \left(\frac{5!}{3!(5-3)!} - 2\right) = \left(\frac{5*4*3!}{3!*2!} - 2\right) = (10-2) = 8$$

Find all the spanning trees and minimum spanning tree

3 minimum Sp. trees with 5 as sum of weights

Applications of minimum spanning trees

- Minimum cost road connectivity
- Minimum cost cable connections in computer networks/ cable TVs
- Telecommunication networks
- Water supply networks
- Electrical grids

minimum spanning tree-further readings

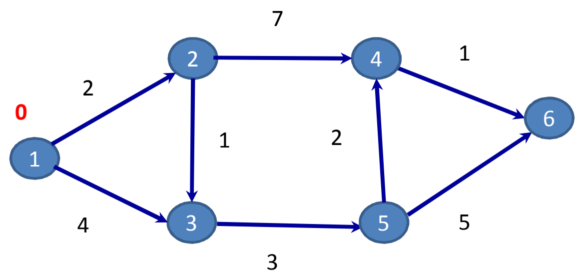
- Read **reverse delete algorithm** from wikipedia and compare it with **Kruskal's algorithm**.

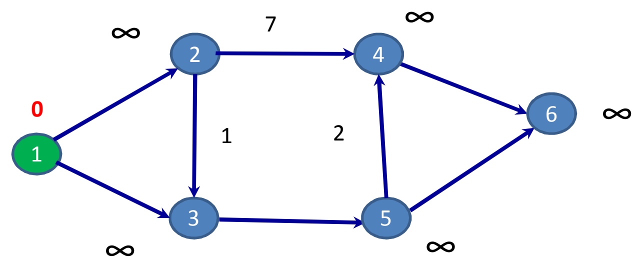
https://en.wikipedia.org/wiki/Reverse-delete_algorithm

Dijkstra's algorithm- finding shortest path from a single node- Greedy approach

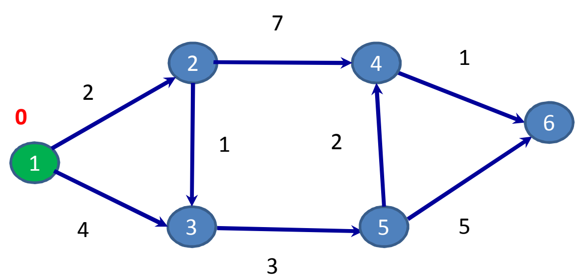
- Proposed by Dutch computer scientist, **Edsgar Dijkstra**, 1956.
- It takes input as a **weighted directed graph** and provides output as **shortest paths** (direct or indirect) to remaining all the nodes from a **single source node**
- It works on directed and undirected graph as well
- It start with a **source node** with **distance 0** and choose the **closest node** directly reachable from it. If a node is not directly reachable it assumes the distance as **infinity** (INF). ∞
- It **updates /relaxes** the nodes which are directly connected with the chosen node
- It repeats the steps till **all the nodes are chosen/selected**

Dijkstra's algorithm- finding shortest path from a single node- Greedy approach

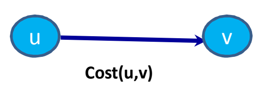




Dijkstra's algorithm- finding shortest path from a single node- Greedy approach



Relaxation



if $(d[u]+\text{cost}(u,v) < d[v])$

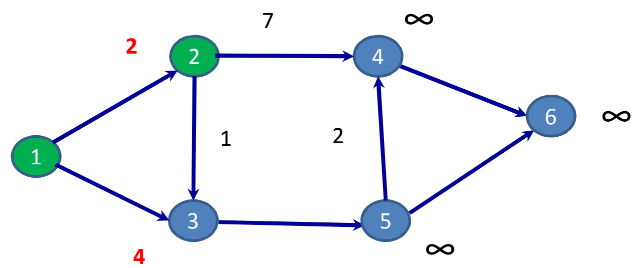
$d[v] = d[u]+\text{cost}(u,v)$

1. $d[2] = d[1]+\text{cost}(1,2) = 0+2 = 2 < \infty$

2. $d[3] = d[1]+\text{cost}(1,3) = 0+4 = 4 < \infty$

d[u]-distance of node u from source

Dijkstra's algorithm- updating a node or relaxation



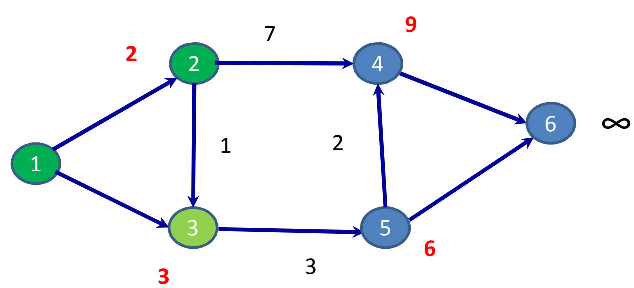
Relaxation

if $(d[u] + cost(u, v) < d[v])$

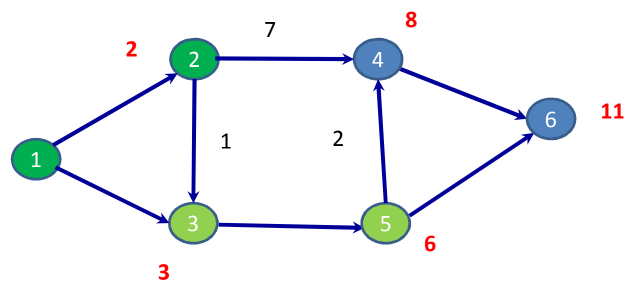
$d[v] = d[u] + cost(u, v)$

$1.\, d[4] = d[2] + cost(2,4) = 2 + 7 = 9 < \infty$

$2.\, d[3] = d[2] + cost(2,3) = 2 + 1 = 3 < 4$

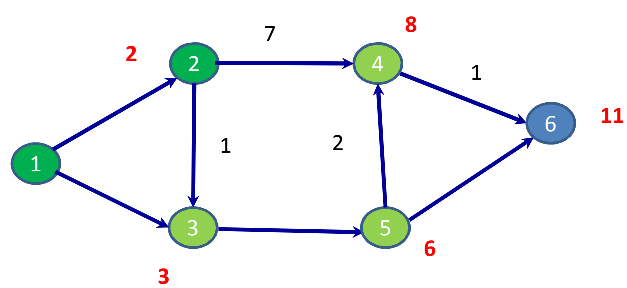


$1.\, d[5] = d[3] + cost(3,5) = 3 + 3 = 6 < \infty$

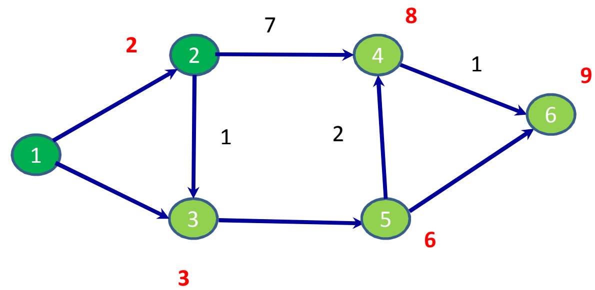


$$1.\ d[4] = d[5] + \mathrm{cost}(5,4) = 6 + 2 = 8 < 9$$

$$2.\ d[6] = d[5] + \mathrm{cost}(5,6) = 6 + 5 = 11 < \infty$$



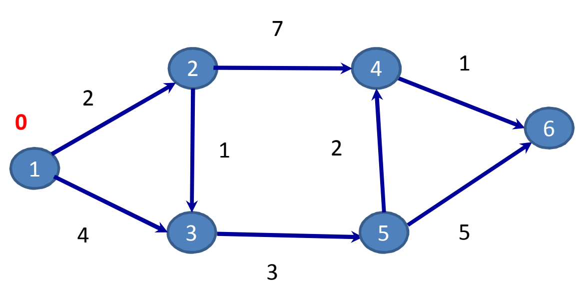
$$1.\ d[6] = d[4] + \mathrm{cost}(4,6) = 8 + 1 = 9 < 11$$



node	d[v] shortest distance from 1
2	2
3	3
4	8
5	6
6	9

Destinations

source	2	3	4	5	6
1	∞	∞	∞	∞	∞
1	2	4	∞	∞	∞
1,2	2	3	9	∞	∞
1,2,3	2	3	∞	6	∞
1,2,3,5	2	3	8	6	11
1,2,3,5,4	2	3	8	6	9



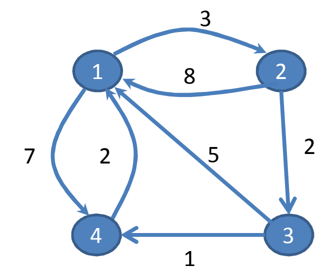
Time complexity of Dijkstra's algorithm

- For a **n node graph**, for each node there can be maximum **n times relaxation** needed and thus it is **O(n^2)**.
- If we use minheap, then it is **O(E/n log n),** choose **maximum** of E (edges) or (/) n (vertices).

All pair shortest path (Floyd-Warshall algorithm)

- This algorithm is used for finding **all pair shortest paths** of a directed and weighted graph.
- Published by Robert Floyd in 1962, Bernard Roy in 1959 and Stephen Warshall in 1962.
- It uses **dynamic programming** approach to find all pair shortest paths

All pair shortest path (Floyd-Warshall algorithm)



We need to generate $n+1$, (nxn) matrices viz. A^0, A^1, A^2, A^3, and A^4

A^0 – gives shortest distance without going through any intermediate node

Matrix A^k is computed from $A^{(k-1)}$ and A^0 is computed from graph

$$A^0 = \begin{bmatrix} 0 & 3 & \infty & 7 \\ 8 & 0 & 2 & \infty \\ 5 & \infty & 0 & 1 \\ 2 & \infty & \infty & 0 \end{bmatrix}$$

All pair shortest path (Floyd-Warshall algorithm)

$$A^0 = \begin{bmatrix} 0 & 3 & \infty & 7 \\ 8 & 0 & 2 & \infty \\ 5 & \infty & 0 & 1 \\ 2 & \infty & \infty & 0 \end{bmatrix}$$

Keep first row and first column same as A0 in A1 and calculate remaining values of A1

$$A^1 = \begin{bmatrix} 0 & 3 & \infty & 7 \\ 8 & 0 & & \\ 5 & & 0 & \\ 2 & & & 0 \end{bmatrix}$$

1.$A^0[2,3] \quad A^0[2,1] + A^0[1,3]$

$2 \quad < \quad 8 + \infty$

thus $A^1[2,3] = 2$

$$A^1 = \begin{bmatrix} 0 & 3 & \infty & 7 \\ 8 & 0 & 2 & \\ 5 & & 0 & \\ 2 & & & 0 \end{bmatrix}$$

2.$A^0[2,4] \quad A^0[2,1] + A^0[1,4]$

$\infty \quad > \quad 8 + 7 = 15$

thus $A^1[2,4] = 15$

$$A^1 = \begin{bmatrix} 0 & 3 & \infty & 7 \\ 8 & 0 & 2 & 15 \\ 5 & & 0 & \\ 2 & & & 0 \end{bmatrix}$$

All pair shortest path (Floyd-Warshall algorithm)

$$A^0 = \begin{bmatrix} 0 & 3 & \infty & 7 \\ 8 & 0 & 2 & \infty \\ 5 & \infty & 0 & 1 \\ 2 & \infty & \infty & 0 \end{bmatrix}$$

$$\begin{aligned} &3.\mathrm{A}^0[3,2] \quad \mathrm{A}^0[3,1] + \mathrm{A}^0[1,2] \\ &\quad \infty \quad > \quad 5+3=8 \\ &\text{thus } \mathrm{A}^1[3,2] = 8 \end{aligned}$$

$$A^1 = \begin{bmatrix} 0 & 3 & \infty & 7 \\ 8 & 0 & 2 & 15 \\ 5 & 8 & 0 & \\ 2 & & & 0 \end{bmatrix}$$

$$\begin{aligned} &4.\mathrm{A}^0[3,4] \quad \mathrm{A}^0[3,1] + \mathrm{A}^0[1,4] \\ &\quad 1 \quad < \quad 5+7=12 \\ &\text{thus } \mathrm{A}^1[3,4] = 1 \end{aligned}$$

$$A^1 = \begin{bmatrix} 0 & 3 & \infty & 7 \\ 8 & 0 & 2 & 15 \\ 5 & 8 & 0 & 1 \\ 2 & & & 0 \end{bmatrix}$$

$$\begin{aligned} &5.\mathrm{A}^0[4,2] \quad \mathrm{A}^0[4,1] + \mathrm{A}^0[1,2] \\ &\quad \infty \quad > \quad 2+3=5 \\ &\text{thus } \mathrm{A}^1[4,2] = 5 \end{aligned}$$

$$A^1 = \begin{bmatrix} 0 & 3 & \infty & 7 \\ 8 & 0 & 2 & 15 \\ 5 & 8 & 0 & 1 \\ 2 & 5 & & 0 \end{bmatrix}$$

All pair shortest path (Floyd-Warshall algorithm)

$$A^0 = \begin{bmatrix} 0 & 3 & \infty & 7 \\ 8 & 0 & 2 & \infty \\ 5 & \infty & 0 & 1 \\ 2 & \infty & \infty & 0 \end{bmatrix}$$

$$\begin{aligned} &6.\mathrm{A}^0[4,3] \quad \mathrm{A}^0[4,1] + \mathrm{A}^0[1,3] \\ &\quad \infty \quad ? \quad 2+\infty=\infty \\ &\text{thus } \mathrm{A}^1[4,3] = \infty \end{aligned}$$

$$A^1 = \begin{bmatrix} 0 & 3 & \infty & 7 \\ 8 & 0 & 2 & 15 \\ 5 & 8 & 0 & 1 \\ 2 & 5 & \infty & 0 \end{bmatrix}$$

Keep 2nd row, 2 nd column and diagonal of A2 same as A1 and find remaining values of A2

$$A^2 = \begin{bmatrix} 0 & 3 & & \\ 8 & 0 & 2 & 15 \\ & 8 & 0 & \\ & 5 & & 0 \end{bmatrix}$$

$$\begin{aligned} &1.\mathrm{A}^1[1,3] \quad \mathrm{A}^1[1,2] + \mathrm{A}^1[2,3] \\ &\quad \infty \quad > \quad 3+2=5 \\ &\text{thus } \mathrm{A}^2[1,3] = 5 \end{aligned}$$

$$A^2 = \begin{bmatrix} 0 & 3 & 5 & \\ 8 & 0 & 2 & 15 \\ & 8 & 0 & \\ & 5 & & 0 \end{bmatrix}$$

$$A^1 = \begin{bmatrix} 0 & 3 & \infty & 7 \\ 8 & 0 & 2 & 15 \\ 5 & 8 & 0 & 1 \\ 2 & 5 & \infty & 0 \end{bmatrix}$$

$$A^2 = \begin{bmatrix} 0 & 3 & 5 & \\ 8 & 0 & 2 & 15 \\ & 8 & 0 & \\ & 5 & & 0 \end{bmatrix}$$

$2.\mathrm{A}^1[1,4] \quad \mathrm{A}^1[1,2] + \mathrm{A}^1[2,4]$

$7 \quad < \quad 3 + 15 = 18$

thus $\mathrm{A}^2[1,4] = 7$

$$A^2 = \begin{bmatrix} 0 & 3 & 5 & 7 \\ 8 & 0 & 2 & 15 \\ & 8 & 0 & \\ & 5 & & 0 \end{bmatrix}$$

$3.\mathrm{A}^1[3,1] \quad \mathrm{A}^1[3,2] + \mathrm{A}^1[2,1]$

$5 \quad < \quad 8 + 8 = 16$

thus $\mathrm{A}^2[3,1] = 5$

$$A^2 = \begin{bmatrix} 0 & 3 & 5 & 7 \\ 8 & 0 & 2 & 15 \\ 5 & 8 & 0 & \\ & 5 & & 0 \end{bmatrix}$$

$3.\mathrm{A}^1[3,4] \quad \mathrm{A}^1[3,2] + \mathrm{A}^1[2,4]$

$1 \quad < \quad 8 + 15 = 23$

thus $\mathrm{A}^2[3,4] = 1$

$$A^2 = \begin{bmatrix} 0 & 3 & 5 & 7 \\ 8 & 0 & 2 & 15 \\ 5 & 8 & 0 & 1 \\ & 5 & & 0 \end{bmatrix}$$

$$A^1 = \begin{bmatrix} 0 & 3 & \infty & 7 \\ 8 & 0 & 2 & 15 \\ 5 & 8 & 0 & 1 \\ 2 & 5 & \infty & 0 \end{bmatrix}$$

$5.\mathrm{A}^1[4,1] \quad \mathrm{A}^1[4,2] + \mathrm{A}^1[2,1]$

$2 \quad < \quad 5 + 8 = 13$

thus $\mathrm{A}^2[4,1] = 2$

$$A^2 = \begin{bmatrix} 0 & 3 & 5 & 7 \\ 8 & 0 & 2 & 15 \\ 5 & 8 & 0 & 1 \\ 2 & 5 & & 0 \end{bmatrix}$$

$6.\mathrm{A}^1[4,3] \quad \mathrm{A}^1[4,2] + \mathrm{A}^1[2,3]$

$\infty \quad < \quad 5 + 2 = 7$

thus $\mathrm{A}^2[4,3] = 7$

$$A^2 = \begin{bmatrix} 0 & 3 & 5 & 7 \\ 8 & 0 & 2 & 15 \\ 5 & 8 & 0 & 1 \\ 2 & 5 & 7 & 0 \end{bmatrix}$$

Using A2 compute A3

$$A^3 = \begin{bmatrix} 0 & 3 & 5 & 6 \\ 7 & 0 & 2 & 3 \\ 5 & 8 & 0 & 1 \\ 2 & 5 & 7 & 0 \end{bmatrix}$$

Using A3 compute A4

$$A^4 = \begin{bmatrix} 0 & 3 & 5 & 6 \\ 5 & 0 & 2 & 3 \\ 3 & 6 & 0 & 1 \\ 2 & 5 & 7 & 0 \end{bmatrix}$$

A4 shows all pair shortest paths

Formula can be determined as

$$A^k[i, j] = \min\left(A^{k-1}[i, j], \left(A^{k-1}[i,k] + A^{k-1}[k, j]\right)\right)$$

Time complexity of all pair shortest path (Floyd-Warshall algorithm)

for k = 0 to n — For creation of all matrices Ak

    for i = 1 to n

        for j = 1 to n — For creation of one matrix Ak

$$A^k[i,j] = \min\left(A^{k-1}[i,j], \left(A^{k-1}[i,k] + A^{k-1}[k,j]\right)\right)$$

        end

    end

end

$$O(n^3)$$