Greedy strategy

Dr. Priyadarshan Dhabe,

Ph. D (IIT Bombay)

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

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	-

15-1=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

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

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

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

$$\text{Total weight w} = \sum \mathrm{x}_{\mathrm{i}} w_i$$

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

$$\text{Total profit p} = \sum \mathrm{x}_{\mathrm{i}} p_i$$

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

Fractional knapsack problem-greedy approach

Constraint

$$\sum \mathrm{x}_{\mathrm{i}} w_i \leq m = 15$$

Objective function

$$\max \sum \mathrm{x}_{\mathrm{i}} p_i$$

Job sequencing problem-greedy approach

- **Problem definition**:- we have been given some **n number of jobs**, each having the **deadline** and corresponding **profits**. We have fixed number of **maximum slots** to process jobs. **Schedule** the jobs in such a way that to obtain **max profit**.

Job sequencing problem-greedy approach

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

ordered

Assumptions/Information

1. Each job need **1 Hr/slot** to complete
2. Each job has given deadline and need to be completed in it
3. **Job j5** can wait for **3 Hrs/slots**
4. Schedule the jobs within the **deadline** and with **max profit**
5. **Not all jobs** can be scheduled
6. Jobs are arranged in **decreasing order** of profit

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

$$0\overset{j2}{\text{------}}1\overset{j1}{\text{-------}}2\overset{j4}{\text{------}}3$$

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

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

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

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-
- **<u>Step-01:</u>**
- Create a leaf node for each character of the text.
- Leaf node of a character contains the occurring frequency of that character
- **<u>Step-02:</u>**
- 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**.

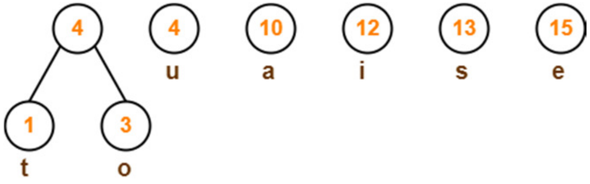
Huffman coding-greedy approach

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

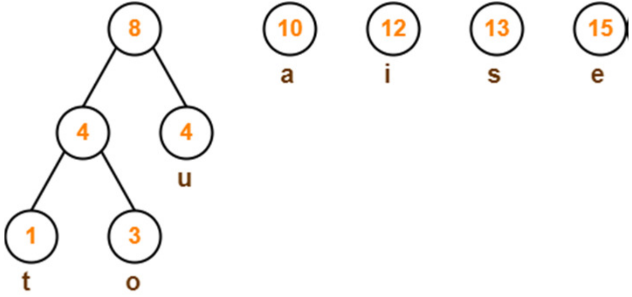
Step-01:

Huffman coding-greedy approach

Step-02:

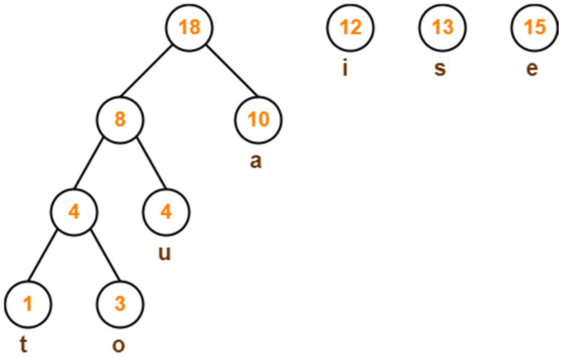


Step-03:

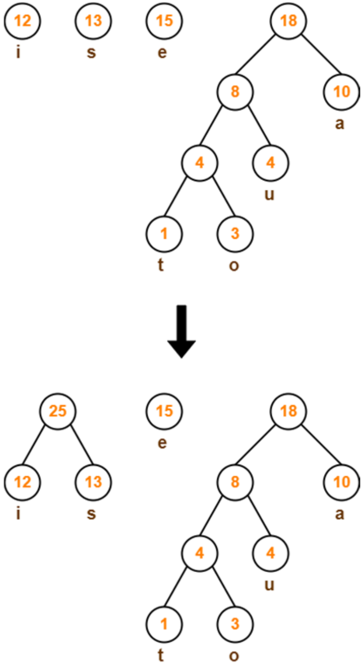


Huffman coding-greedy approach

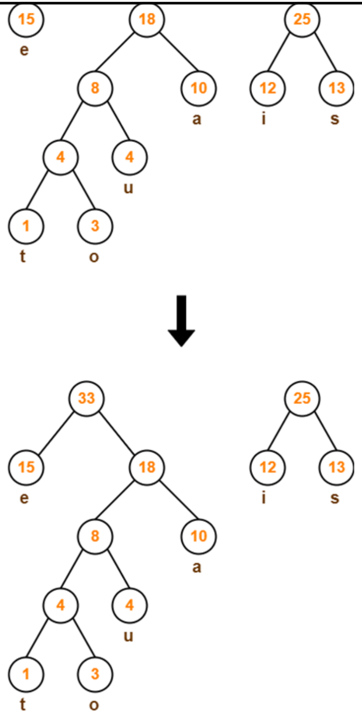
Step-04:



Step-05:



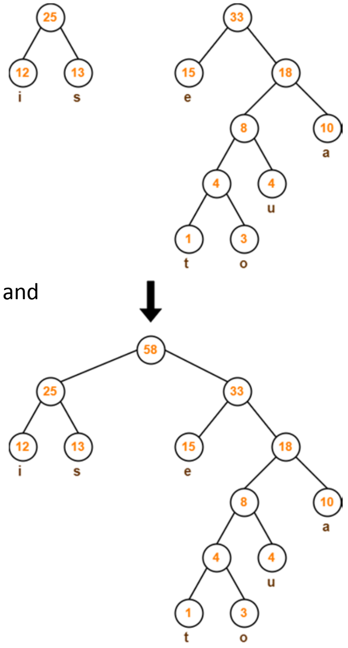
Step-06:



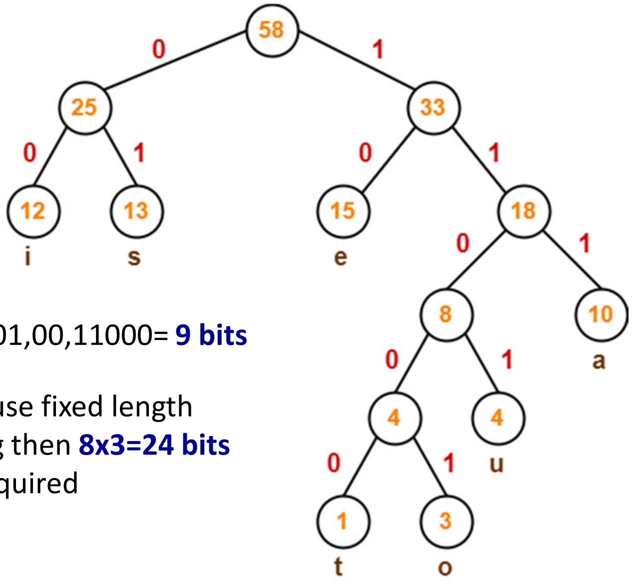
15
e
18
8
10
a
25
12
i
13
s
4
4
u
1
t
3
o
33
15
e
18
8
10
a
4
4
u
1
t
3
o
25
12
i
13
s

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



Huffman Tree

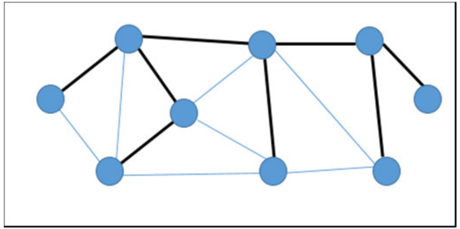
Sit=01,00,11000= **9 bits**

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

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.



Graph and spanning tree

Spanning tree is shown by highlighted edges

Spanning tree definition

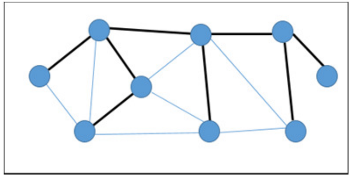
$$\text{Graph } G = (V, E)$$

$$V \text{ - set of vertices and } E \text{ - set of edges}$$

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

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

$$T \text{ contains } |V| - 1 \text{ edges}$$



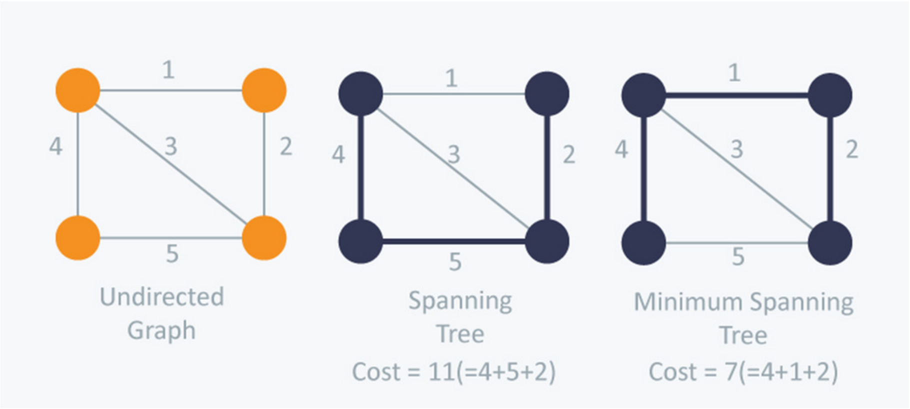
Minimum spanning tree definition

- A **Minimum Spanning Tree** 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 can be **many spanning trees** for a graph but only one **minimum spanning tree (MST)**
- There are **two algorithms** to derive MST using greedy approach

  1.Prim's algorithm

  2. Kruskal's algorithm

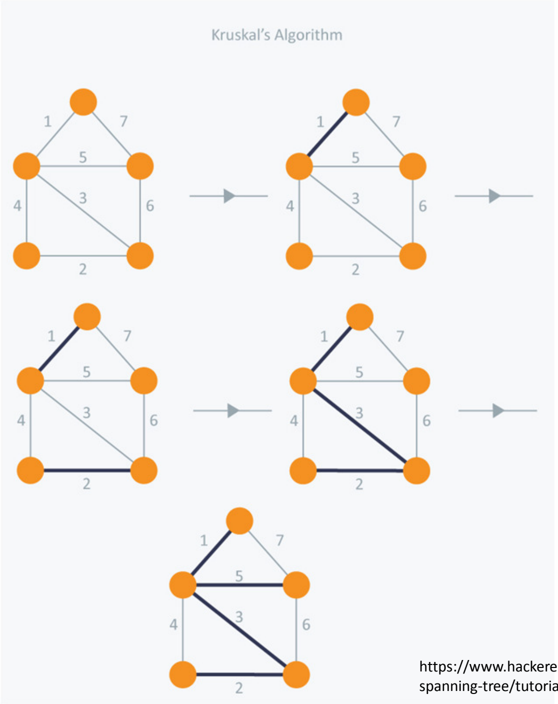
Minimum spanning tree definition



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

- **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

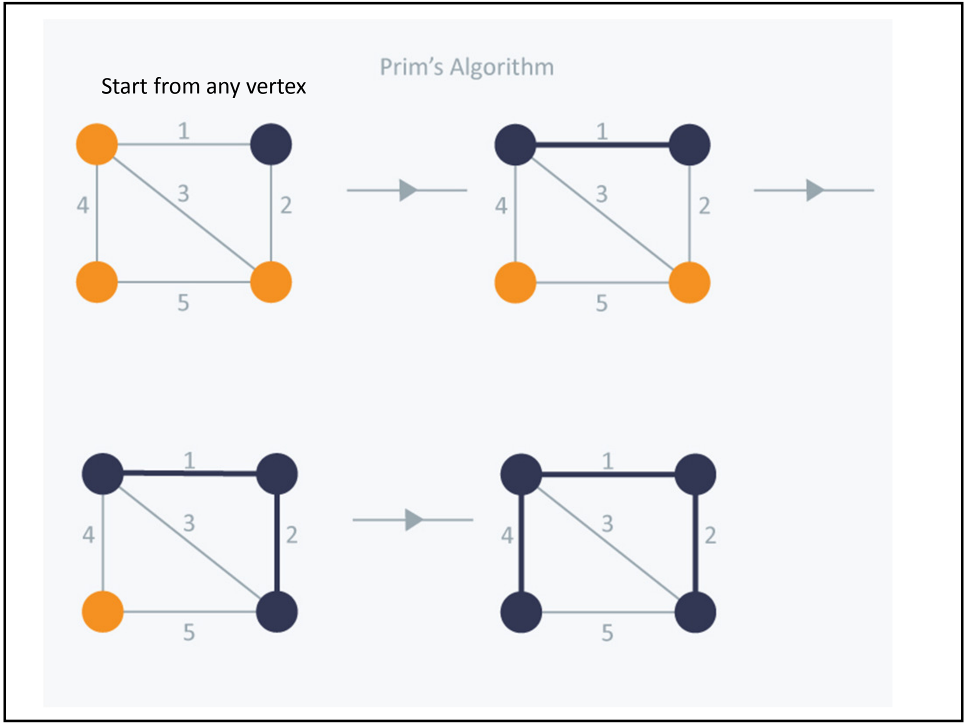
- Time complexity of Kruskal's algorithm- Most of the time is required in sorting the edges, thus it is O(E log E) (sorting time of merge sort O(n log n)).
- This algorithm is proposed by **Joseph Kruskal** in 1956 (wikipedia)

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

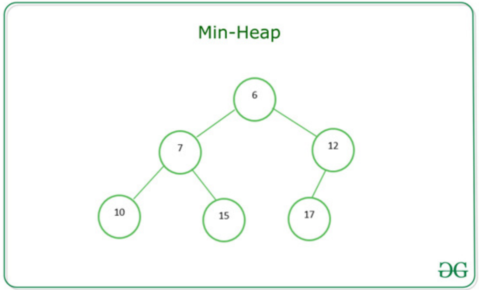
- **Algorithm Steps:**
- Maintain two **disjoint sets** of vertices. One containing vertices that are in the growing spanning tree (A) and other that are not in the growing spanning tree (B). (A int B is empty)
- Select the cheapest vertex that is connected to the growing spanning tree and is not in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


Prim's Algorithm
Start from any vertex
1
3
4
2
5
1
3
4
2
5
1
3
4
2
5
1
3
4
2
5

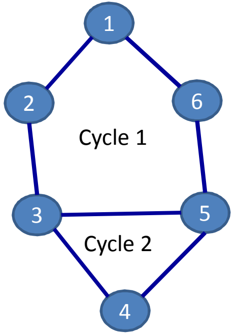
(MST) –Prim’s Algorithm-Greedy approach

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

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 graph



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

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

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

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

Thus, will have $^{7}C_{5} = \dfrac{7!}{5!(7-5)!} - 2 = \dfrac{7*6}{2!} - 2 = 19$

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**.