Greedy Algorithms

- It is used to solve optimization problem.

→ Optimization problem : (maximization or minimization Problem)
It is the problem of finding the best solution from all feasible solutions.

- Feasible solution:
Solution that satisfy the constraint.

- Greedy algorithm always makes the choice that look best at the moment.

- Greedy ~~al~~ algorithms do not always yield optimal solutions, but for many problem they do.

* Elements of Greedy Algorithm

Greedy choice property : Globally optimal solution can be obtained by making a locally optimal choice.

Optimal substructure :
A problem exhibits optimal substructure, if optimal solution to the problem ~~is~~ contains is composed of optimal solutions to subproblems.

→ If we can demonstrate that the problem has these properties, then we can develop greedy algorithm.

Greedy vs Dynamic

→ In Greedy subproblems are independent.

→ It is top down approach

→ After making choice we can't undone.

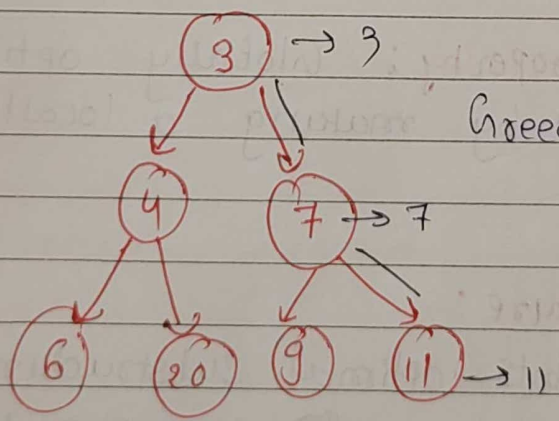
→ It will not depend on future choice

→ Finding local optimal solution at each step.

→ It will not give always optimal solution.

Example

Problem: To find largest sum from root to leaf value.



Greedy answer = 3+7+11
= 21

This is not optimal

3 + 4 + 20 = 27 → optimal solution

Making locally optimal choices ending up with non-optimal solution.

Applications of Greedy strategy

optimal solution:

- making change Problem
- minimum spanning tree (MST): Prim's and Kruskal's
- single-source shortest path: Dijkstra's algo
- Activity selection Problem
- Huffman Code

Approximations / heuristics:

- Travelling salesman Problem (TSP)
- Knapsack Problem

Activity selection problem (Greedy approach) O(nlogn)

→ It is a optimization problem.

Problem is to select the maximum number of activities that can be performed by a single person or machine, assuming that a person can only work on a single activity at a time

constraints : activities should be compatible or non overlapping or non conflicting.

objective : maximum set of nonconflicting activities

i/p → set of activities, start time and finish time

Activities a_i and a_j are compatible if the interval $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap

$0 \leq s < f < \infty$

Greedy Approach of activity selection

O(nlogn) sort the activities according to their finish time

- Select first activity from sorted array.
- The greedy choice is to always pick the next activity whose start time is greater or equal to finish time of priviously selected activity

Greedy-Activity-Selector (s, f) Iterative algorithm

```
n = length[s]
A = {a1}
i = 1
for m = 2 to n
    do if s[m] ≥ f[i]
        then A = A ∪ {am}
             i = m
return A
```

O(n)

Activity selection Problem

→ sort activity with it's increasing Finish time (O(nlogn))

Initially R-A-S(s,f,0,n)

Recursive-Activity-selector (s,f,k,n)

Initially $R_i = 0$

```
Recursive-Activity-selector (s,f,k,n)
    m = k+1
    while m ≤ n and s[m] < f[k]
        m = m+1
    if m ≤ n
        return {am} ∪ Recursive-activity-selector(s,f,m,n)
    else Return ∅
```

Example :

s	1	3	0	5	8	5
f	2	4	6	7	9	9

R-A-S(s,f,0,6)

m = 0+1 = 1

while 1 ≤ 6 and 1 < 0 ×

1 ≤ 6 ✓

{a1} ∪ R-A-S-(sf, 1, 6)

k=1

s	1	3	0	5	8	5
f	2	4	6	7	9	9

m

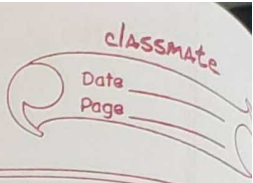
s	1	3	0	5	8	5
f	2	4	6	7	9	9

k m m

3 < 2 ~~0 < 2~~

m

Fractional Knapsack Problem



Knapsack problem

- Fractional
  - Items should be divisible
  - Greedy
- 0-1
  - not divisible
  - Dynamic

Fractional Knapsack Problem

- n items, each have w_i and Profit P_i, and given knapsack capacity C.
- The problem is to ~~find~~ fill the knapsack with object/ items without exceeding knapsack's capacity in order to maximum the total Profit.

maximize $\sum_{i=1}^{n} P_i x_i$

subject to $\sum_{i=1}^{n} w_i x_i <= C$

x_i denote fraction of object i

$0 \le x_i \le 1$

$1 \le i <= n$

Different approaches

1. Maximum Profit
2. minimum weight
3. max value/weight Ratio

Example:

$m=5$ objects and knapsack capacity $W=100$ as in Table 1.

	1	2	3	4	5
W	10	20	50	40	50
V	20	30	66	40	60
V/W	2.0	1.5	2.2	1.0	1.2

Select	x_i					Profit/value
max v_i	0	0	1	0.5	1	146
~~max~~ min w_i	1	1	1	1	0	156
max v_i/w_i	1	1	1	0	0.8	164

Optimal

Greedy approach (W, v, w)

$O(n \log n)$ Sort n objects from large to small base on v_i/w_i
- Assume array $w[1,..n]$ and $v[1...n]$ store the respective weights and values after sorting
- Initialize array $x[1...n]$ to zeros

weight = 0, i = 1, value = 0
while (i <= n and weight < W)
    if weight + w[i] ≤ W
        x[i] = 1
    else
        x[i] = (W - weight)/w[i]
    weight = weight + x[i] * w[i]
    value = value + x[i] * v[i]
    i++
Return value

$O(n \log n)$

Spanning Tree → Does not generate cycle.

Spanning Tree: It is subset of edges of connected undirected graph, that connects all the vertices togather without any cycle

Total No. of possible spanning tree = $E C_{V-1}$ − no of cycle (include those cycle which contain not all vertices)

If $G = (V, E)$, spanning tree $G' = (V', E')$

$V' = |V|$

$E' = |V| - 1$

- Minimum Spanning Tree:
  - Unidirected weighted connected graph.

Spanning tree which has total cost minimum

two methods to find MST

1) Prims
2) Khushkal

Best = $O((V+E)\log v)$ or $O(E \log v)$

worst $O(v^2 \log v)$

* Prims Algorithms (Greedy approach)

Remove all loops and parrallel adges

Prim (G, w, r)

for each u in V[G] } O(v)
  do key[u] = ∞
  π[u] = NIL π for parent

key[r] = 0

Buildheap Q = V[G] Queue → Q → O(v)

while Q ≠ ϕ → O(v) (extract minheap algo)
  do
  u = Extract-MIN(Q) → minheap (extract vertex which has min key value)
  for each v in Adj[u]
    if v in Q and w(u,v) < key[v]
      π[v] = u
      → key[v] = w(u,v) → Decreaskey of minheap

O(v) * log v
v * (v-1)
v * (v-1)
v * (v-1)
v * log v * (v-1) → Dense

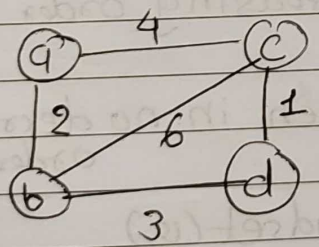
worst case = $v^2 \log v$ (complete graph)

(Q is min Priority Queue)

in worst case for one node adjsane node will be v-1 (in complete graph)

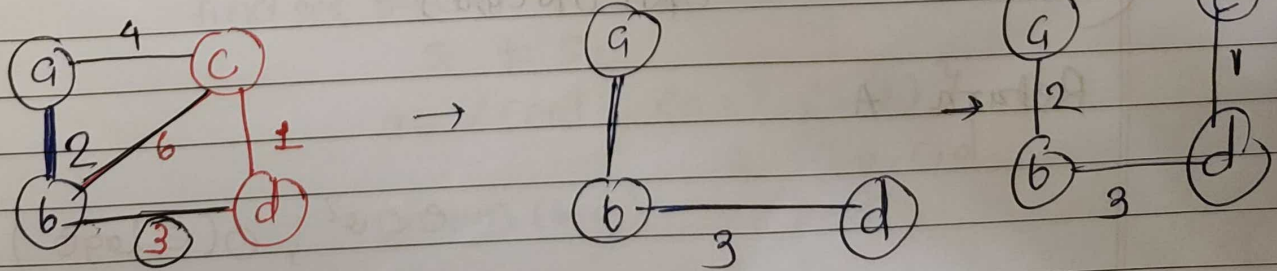
no. of edges in worst case $E = v^2$
in Best case = v
Sparse $E = v$

Example:



- select arbitary vertex

- find minimum adjucancy edges and select in MST (if it create no cycle)



total minimum cost = 6

Krushaka's Algorithm

- Greedy approach
- Sort the edges in non-decreasing order of lengths (weights)
- Build forest of mst by growing tree one edge at a time.
- At each iteration, add the next edge from sorted list unless this would create a cycle. (If it would, skip the edge)
- Repeatedly connect the trees to create a subset of a mst, until all nodes are covered.

Forest tree → more than one

* Kruskal(G, w)

A = ∅ set A = { } O(v)

for each vertex v in V[G] { }

do makeset(v)

O(eloge) sort edges of E in nondecreasing order by weight w

O(v+e)logv for each edge (u,v) in E, taken in no decreasing order by weight

do if Findset(u) ≠ findset(v)

A = A ∪ {(u,v)}

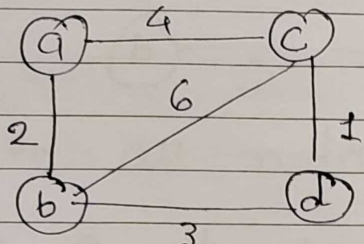
UNION(u,v)

Return A

$e < v^2$; O(e log v)

Example

with Algo;



1 → A = { }

2 → {a}, {b}, {c}, {d}
S1 S2 S3 S4

3 E = {(c,d), (a,b), (b,d), (a,c)}
1 2 3 4

take (c, d)

$S_3 \neq S_4$

if Find set (c) ≠ Find set (d) ✓

both are in different set

A = {(c, d)}

Union (c,d)

{a} {b} {c,d}

S_1	a
S_2	b
S_3	c,d
S_4	~~d~~

take (a,b)

Find set (a) ≠ find set (b) ✓
S_1 S_2

A = {(c,d), (a,b)}

Union {a,b}

{a,b} {c,d}

S_1	a,b
~~S_2~~	~~b~~
S_3	c,d

take (b, d)

find set (b) ≠ find set (d) ✓

$S_1 \neq S_2$

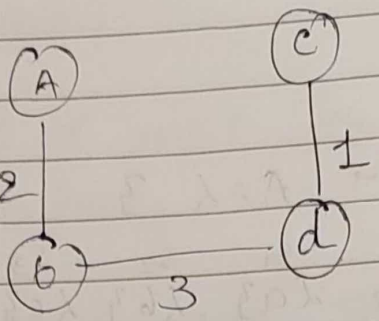
A = {(c,d), (a,b), (b,d)}

Union (b,d)

S_1	a,b,c,d
S_2	~~c,d~~

now all u & v are in one set so we can't go further,

Final MST

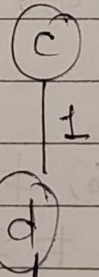


→ ~~Non~~ kruskal ~~os~~ without Algo

step: 1 - short edges in non decreasing order by ω

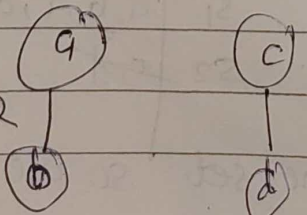
(c,d) → 1
(a,b) → 2
(b,d) → 3
(a,c) → 4
(b,c) → 6

Select 1st edge and include in MST if It will not generate cycle
Repeat till all node are added (Repeat till |V|-1 edges in in MST

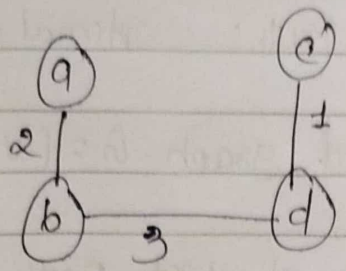


(a,b) will be selected

Forest trees are generated

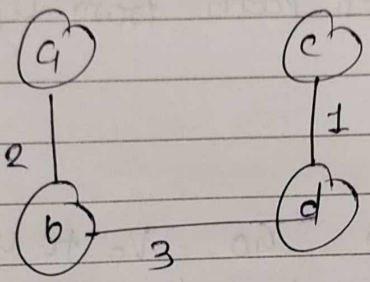


select (b,d) → 3



- Here, all nodes are connected so, stop

∴ Result is



Single source Shortest Path

* lemma → subpaths of shortest path are shortest Paths (optimal substructure)

Given a weighted directed graph $G = (V, E)$ with weight function $w : E \to R$,
Let $p = \langle v_0, v_1 \dots v_k \rangle$ be a shortest path from vertex v_0 to vertex v_k and, for any i and j such that $0 \le i \le j \le k$, let $p_{ij} = \langle v_i, v_{i+1}, \dots v_j \rangle$ be the subpath of p from vertex v_i to v_j.
Then p_{ij} is a shortest path from v_i to v_j

i, j vertex

→ Proof :

If we want to go v_0 to v_k via i and j and if it will give shortest path. then subpath of p_{ij} will be shortest subpath

$$p = p_{0i} + p_{ij} + p_{jk}$$

$$w(p) = w(p_{0i}) + w(p_{ij}) + w(p_{jk}) \quad \text{— (1)}$$

for contradiction, let's assume that p_{ij} is not shortest path. and another p'_{ij} is shortest path.

$$\therefore w(p'_{ij}) < w(p_{ij})$$

Instead of taking $w(p_{ij})$ we will take another path weight $(w(p'_{ij})$ so we will replace $w(p_{ij})$ with $w(p'_{ij})$

$$w(p) = w(p_{0i}) + w(p'_{ij}) + w(p_{jk})$$

So Now our assumption is incorrect
P'ij will always be shortest path between i to j

Shortest Path Problem :

Single Destination shortest path:

- Find a shortest path to a given destination vertex t from each vertex.

Single Pair Shortest path

- Find a shortest path from u to for given vertices u and v (Dijkstra , Bellman-ford)
  +ve edge , both +ve and -ve edge)

All pairs shortest-Paths problem (Floyd warshall)

find a shortest path from u to for every pair of vertices u and v.

* All Single source Shortest Path
→ Greedy approach
- For nonnegative edge weight

→ Dijkstra Algorithm ! (Greedy algo)

we assume that $w(u,v) \geq 0$ for each edge $(u,v) \in E$

directed/undirected weighted graph

~~Int~~
DIJKSTRA(G, w, s)
{
Initialize-single-source(G,s)
S ← ∅ ⟶ Shortest path tree set
Q ← V[G] → Min priority Queue
while Q ≠ ∅
do u ← Extract-MIN(Q) (extract vertex which has minimum distance value)
S ← S ∪ {u}
for each vertex v ∈ Adj[u]
do RELAX(u,v,w)
}

→ Initialize-single-source(G,S)
└ source vertex
{
for each vertex v ∈ V[G]
do d[v] ← ∞
π[v] ← NIL
d[s] = 0
}

```
RELAX(u,v,w)
{
- If d[u]+w(u,v) < d[v]
      then   d[v] ← d[u]+w(u,v)
             π[v] ← u
}
```

Time complexity

Depends upon how we will implement priority queue

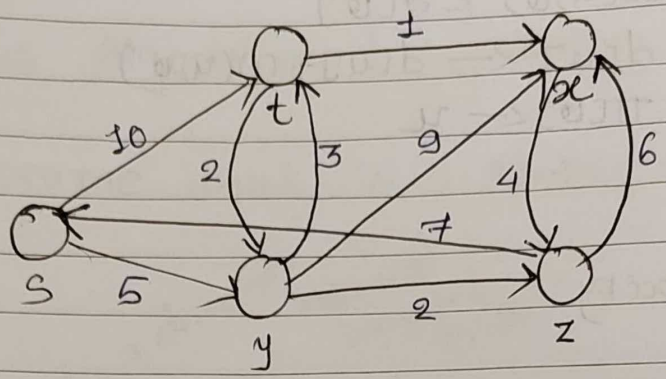
Time complexity
Priority queue

- graph represented by weigh matrix & array for queue
  ↓
  $O(|V|^2)$

- graph represent by adjacency list and minheap for queue
  ↓
  $O(E \log V)$

Example :

Through ~~without~~ Algorithm :



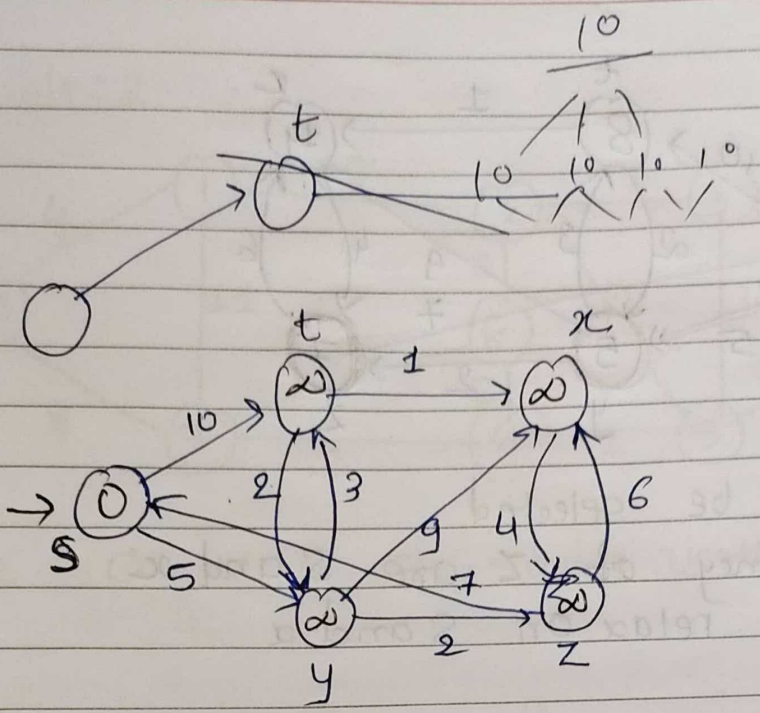
Source vertex = S

Initialization

→ other than source vertex distance of each vertex is ∞ and π value will be NIL

→ distance of source vertex will be 0 Parent π value will be NIL

selected vertex	S	d[v] t	x	y	z
~~S~~	(0)	∞	∞	∞	∞
~~S~~ y		10	∞	(5)	∞
z		~~8~~	14		(7)
x		(8)	13		
z			(9)		



adjacancy vertex of s are t and y

apply Relaxation on t and y

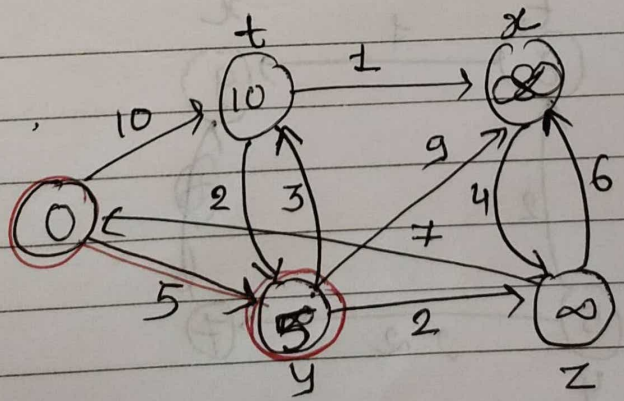
Relax (S, t, 10)

if (d[u] + w(u,v) < d[v])

0 + 10 < ∞ ✓

d[v] = 10 d[t] = 10

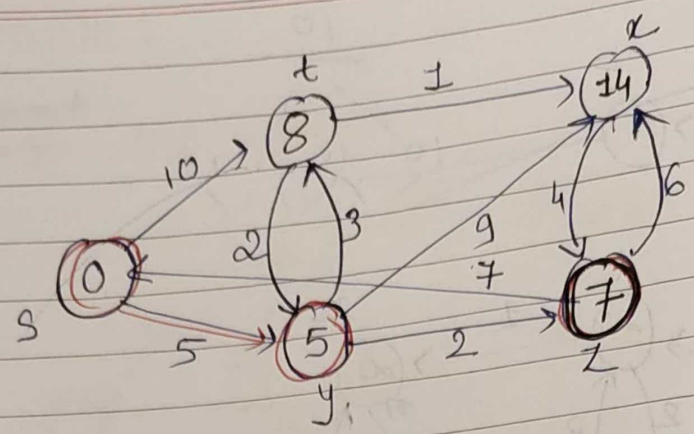
same way d[y] = 5



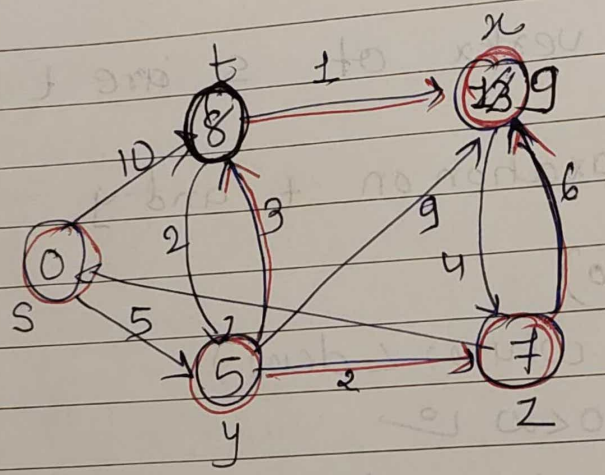
Now y has minimum distance so y will be selected.

adjacancy of y are t, x, z

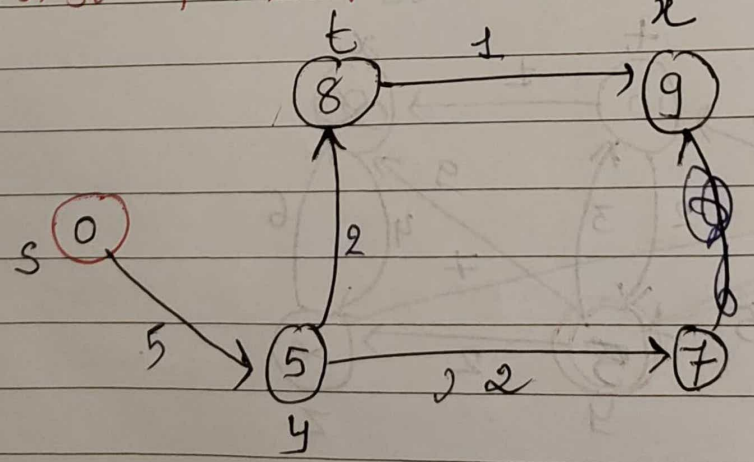
apply Relax (u, v, w) on t, x, z



z will be selected
Adjacancy of z are s and x
apply relax on s and x

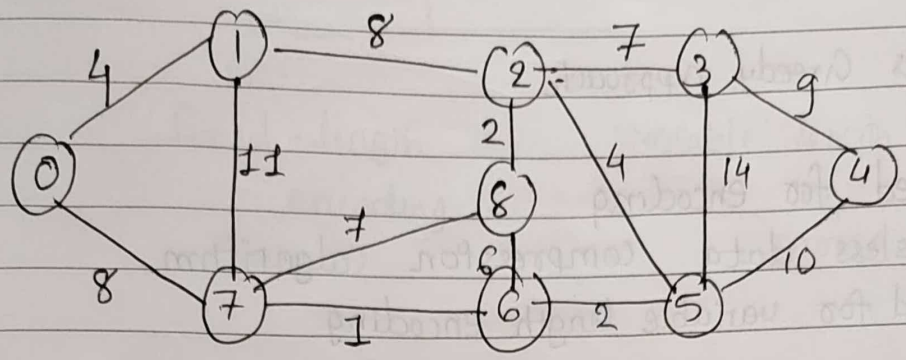


Shortest path from s is

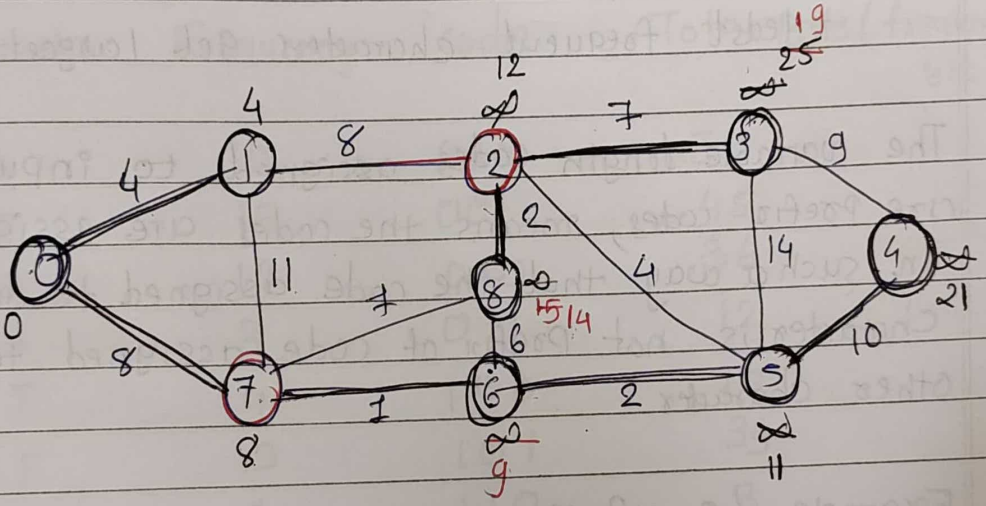


classmate
Date
Page

* Example: 2



take 0 as a ~~shor~~ source vertex



* Huffman coding

- Uses Greedy approach

- Used for encoding
- Lossless data compression algorithm.
- Used for variable length encoding

Idea of variable length encoding
- most frequent characters get smallest code
- least frequent characters get largest code.

The variable length codes assigned to input characters are prefix codes, means the codes are assigned in such a way that the code assigned to one character is not prefix of code assigned to any other character.

Example {0, 11} ✓

{0, 1, 11} ✗ 1 is prefix of 11

{a:00, b:01, c:0, d:1} ✗

code assign to c is prefix of a and b

- Huffman coding makes sure that there is no ambiguity when decoding the generated bit stream.

Two-Types of codes

- fixed-length encoding (ASCII)
- variable length encoding morse code

* Fixed length encoding

character	Frequency	code	Total Bits (frequency × bit length)
A	10	000	30
E	15	001	45
I	12	010	36
S	3	011	12
T	4	100	12
P	13	101	39
Newline	1	110	3
~~8×1=56~~		~~16×3=48~~ 24	total bit used : 174

Here total ~~8~~ 7 characters are there so we required (2^3) 3 bits.

Can we reduce no. of bits ?
Yes using variable length encoding

- when we will send the msg we have to send character & code table along with encoded msg. so d. we can do decoding.
  (~~56~~ + ~~48~~ 21) + 174 bits will be received
  ~~248~~ 252

→ ~~A~~ .

If we will use ASCII for sending msg than $58 \times 8 = 464$ bits are required

Huffman coding procedure

i/p → array of unique characters along with frequency count

o/p → Huffman tree

1) Build a huffman tree
2) Traverse the Huffman tree and assign codes to characters.

Build a huffman tree

1 - Create a leaf node for each character and a build min heap of all leaf nodes

2 - Extract two nodes with minimum frequency from min heap

3.1 - Create a new internal node with frequency equal to sum of two node frequenc.

3.2 - Make first extracted node as its left child and other as it's right child. Add this node to the min heap

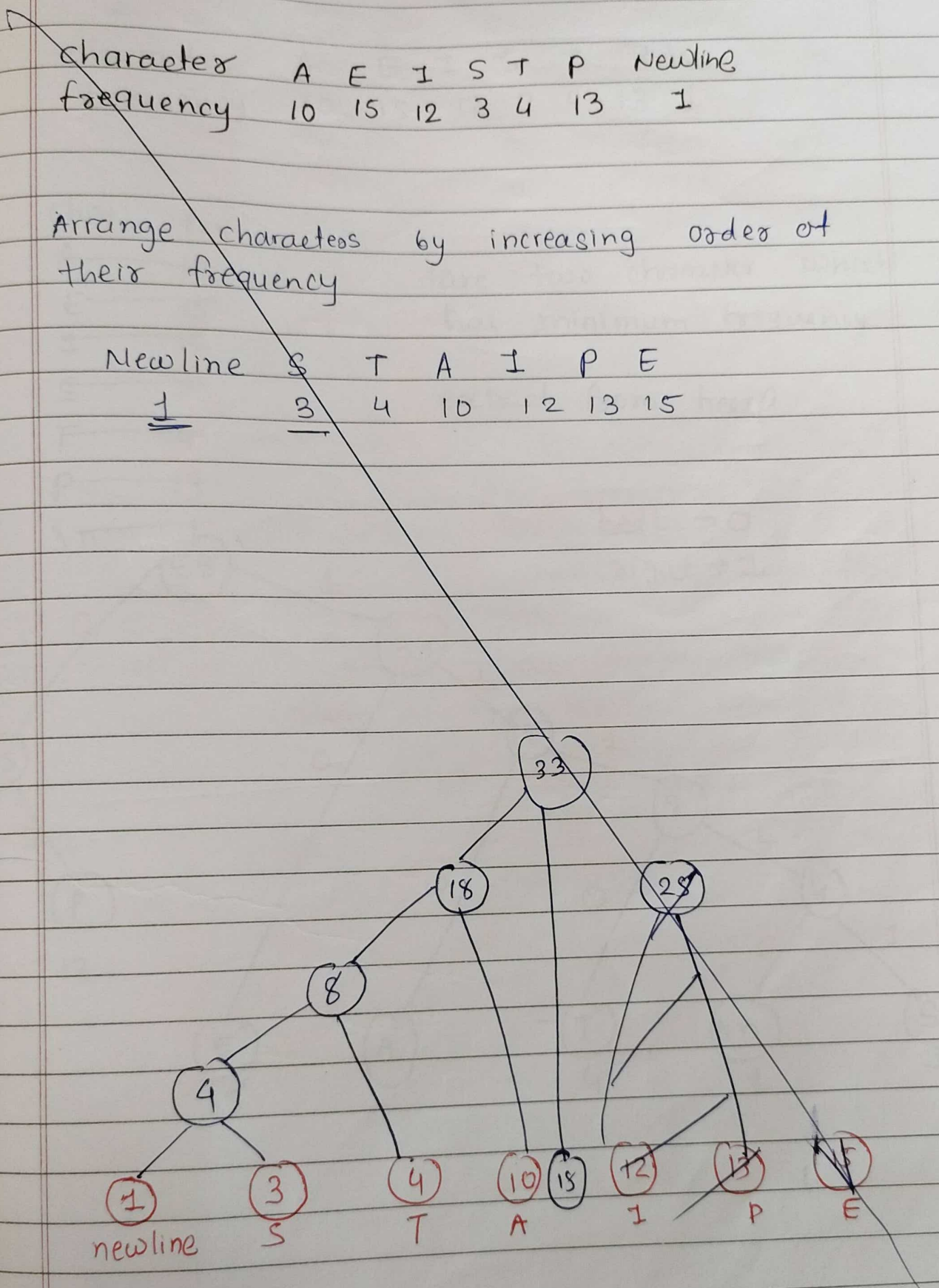
4 - Repeat step 2 & 3 until heap contain only one node. The remaining node is root node and tree is complete.

Example

character	A	E	I	S	T	P	Newline
frequency	10	15	12	3	4	13	1

Arrange characters by increasing order of their frequency

Newline	S	T	A	I	P	E
1	3	4	10	12	13	15



Example

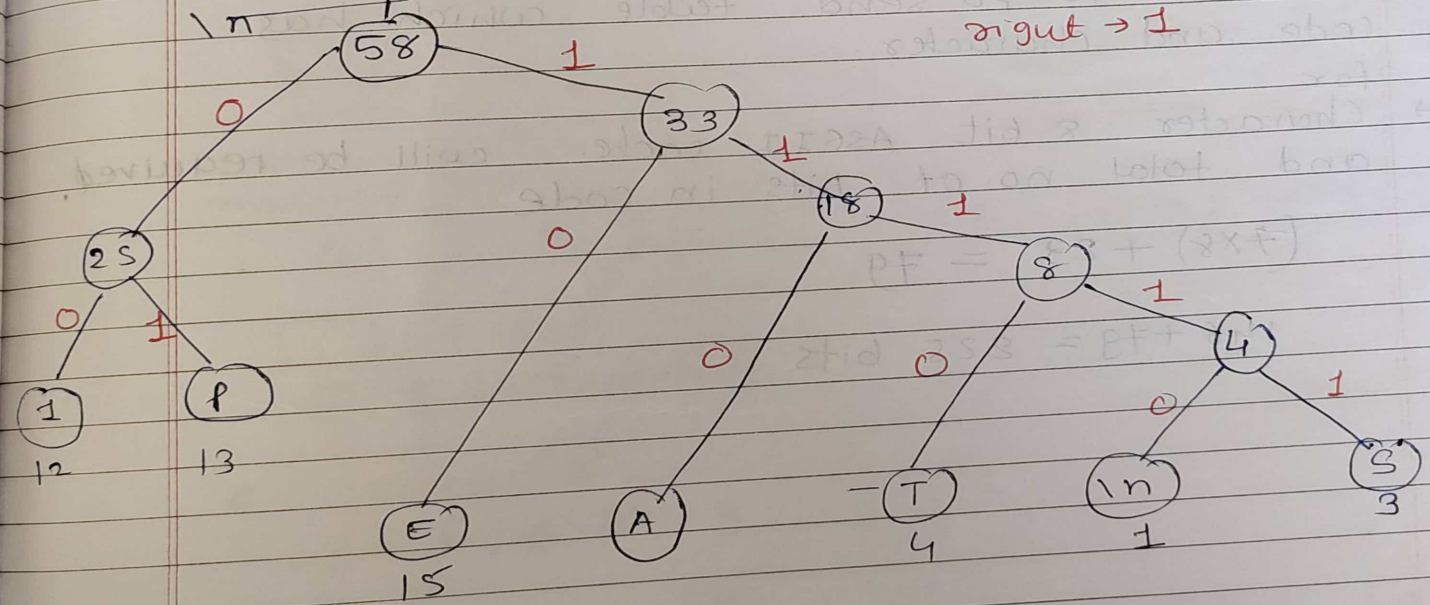
character	A	E	I	S	T	P	\n
freequency	10	15	12	3	4	13	1

chas	fre
A	10
E	15
I	12
S	3
F	4
P	13
\n	1

take two character which has minimum frequency

extract from heap

left → 0
right → 1



Char.	code	freq	Total bits
A	110	10	30
E	10	15	30
I	00	12	24
S	11111	3	15
T	1110	4	16
P	01	13	26
\n	11110	1	5
			146

Total bits for sending massage

we have to ~~pa~~ send table which has code and character.

~~for~~

→ Character 8 bit ASCII code will be required, and ~~total~~ no. of bits in code

$(7 \times 8) + 23 = 79$

$146 + 79 = 225$ bits

Algorithm:

Huffman (C) → always generate full binary tree
{ C is set of n characters and each character has it's frequency

$n = |C|$

min priority queue $Q = C$ → $O(n)$

for $i = 1$ to $n-1$ $(n-1)$

allocate a new node z

z.left = x = Extract_MIN(Q) } $\log n$ $(n-1)$ min heap

z.right = y = Extract_MIN(Q)

z.freq = x.freq + y.freq.

Insert (Q, z)

return Extract_MIN(Q) // return root of the tree

→ $O(n \log n)$

We can reduce running time to $O(n \log\log n)$ by replacing min heap with Van Emde Boas tree.

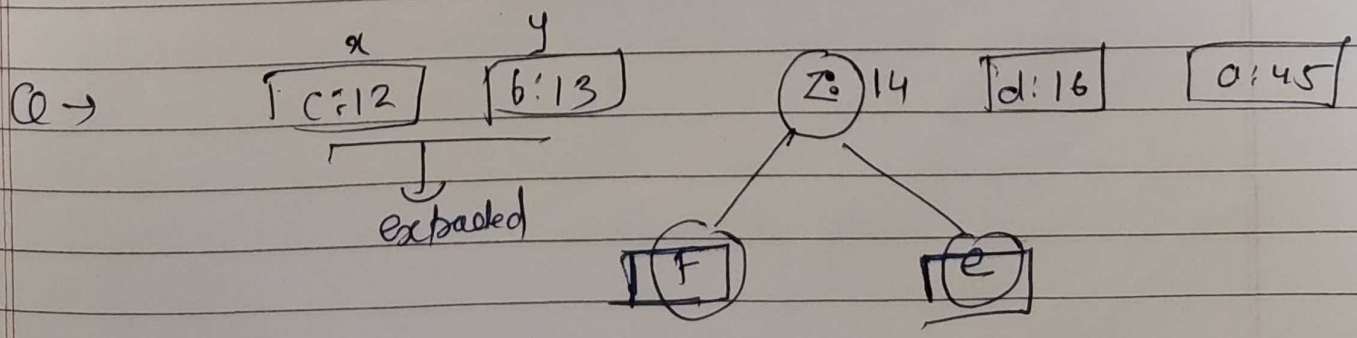
Q: | F:5 | e:9 | c:12 | b:13 | d:16 | a:45 |

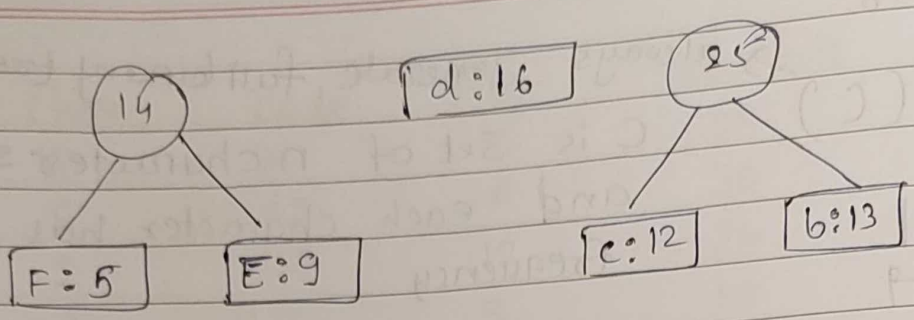
□ → leaf

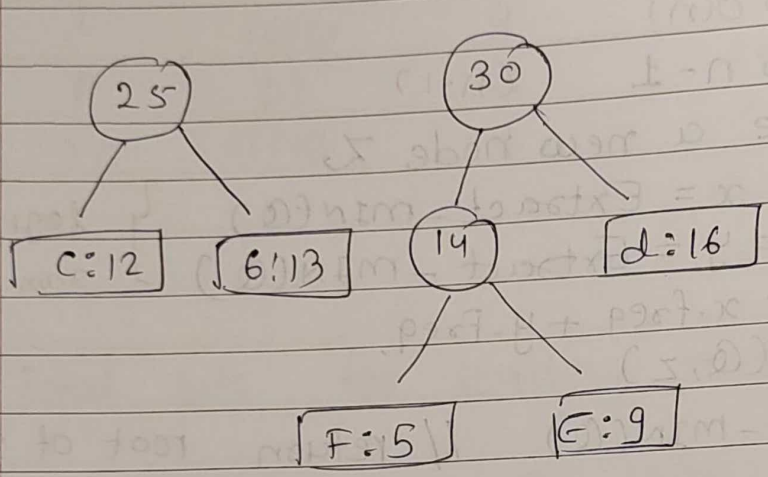
○ → Internal node,

z.left = F = x

z.right = e = y







Extract 2 chara
which has
minimum fre