Date 16/2/2023
Page 62

# * GREEDY ALGORITHM.

## (1) * Coin Changing

Given currency { 1, 5, 10, 25, 100 }.

Ex :- 34¢ = 25, 5, 1, 1, 1, 1

$2.89 = 100, 100, 25, 25, 25, 10, 1, 1, 1, 1

Algorithm.

Cashier-Algorithm ( $x, c_1, c_2, c_3, \ldots, c_n$ )

1. Sort 'n' denominations so that
   $0 < c_1 < c_2 < \ldots < c_n$
2. $S = \phi$ [Empty set].
3. while ($x > 0$)
4. : $k$ = largest coin denomination $c_k$ such that $c_k \leq x$
5. If no such $c_k$
   Return "No solution"
6. Else.
7. $x = x - c_k$
8. $S = S \cup \{k\}$.
9. Return S.

Is this optimal. —— Yes.
Because of the given denomination.

Is it optimal for all denomination?
↳ No.

Ex:- Postages {1, 10, 21, 34, 70, 100, 350, 1225, 1500}.

Cashier Algo 140 ¢ = 100 + 34 + 1 + 1 + 1 + 1 + 1 = 8.
Optimal = 140 ¢ = 70 + 70 = 2

Sometimes, this algo may not even give feasible solution.

Ex: {7, 8, 9}.
Cashier Algo ⇒ 15 ¢ = 9 + 6
Optimal ⇒ 15 ¢ = 7 + 8 = 2

Property of any optimal solution -

1) No. of pennies ≤ 4. → ¢1
2) No. of nickles ≤ 1 → ¢5
3) No. of quarters ≤ 3 → ¢25
4) No. of N + No. of dimes ≤ 2.
5) No. of Dollar = no limit

| k | $c_k$ | optimal property | max. value in optimal solution |
|---|---|---|---|
| 1 | 1 | ≤ 4 | - |
| 2 | 5 | ≤ 1 | 4 (4 coins) |
| 3 | 10 | N + D ≤ 2 | 4 + 5 = 9 (5 coins) |
| 4 | 25 | ≤ 3 | 20 + 4 = 24 (6 coins) |
| 5 | 100 | no limit | 75 + 24 = 99 (9 coins) |

Optimality

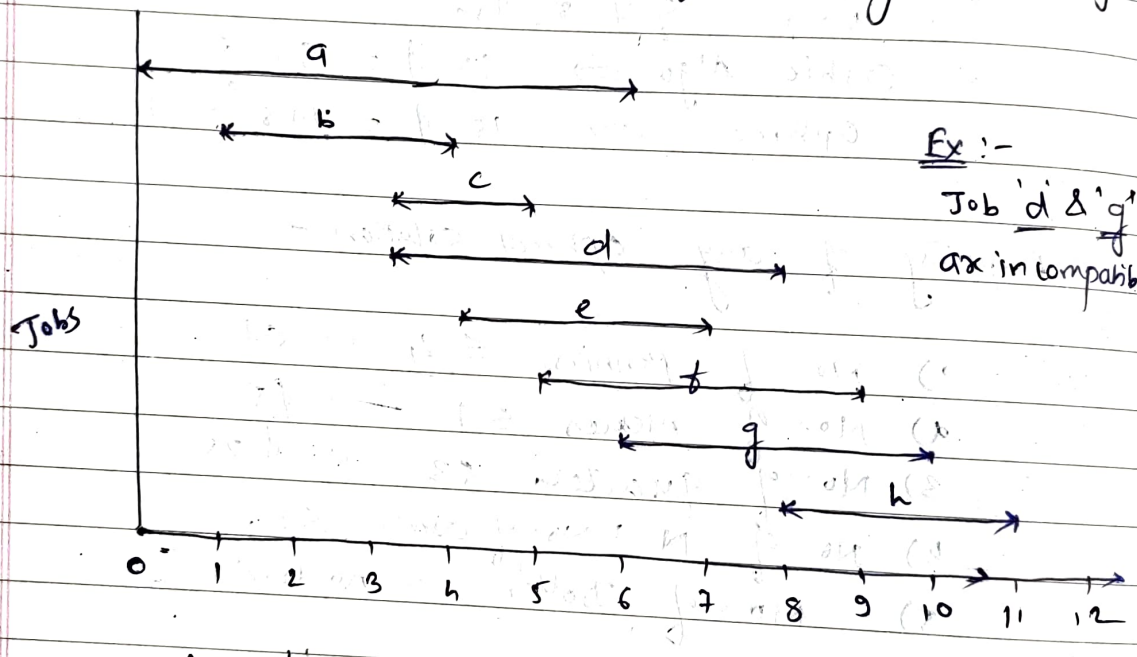
optimal way - $c_k \le x < c_{k+1}$ → greedy takes 'k'
If not - $c_1 \dots c_{k+1}$ should add upto 'x'.
Table above shows - this is not optimal.

## (DS # Interval Scheduling :-

⇒ Job 'j' starts at $s_j$ and finishes at $f_j$

⇒ Two jobs are compatible if they don't overlap.

⇒ Goal :- find max. subset of mutually compatible jobs.




Ex :- Job 'd' & 'g' are incompatible.

| j | $s_j$ | $f_j$ |
|---|---|---|
| a | 0 | 6 |
| b | 1 | 4 |
| c | 3 | 5 |
| d | 3 | 8 |
| e | 4 | 7 |
| f | 5 | 9 |
| g | 6 | 10 |
| h | 8 | 11 |

Three different approach

1) Earliest start time — sorted by $s_j$
2) Earliest finish time — sorted by $f_j$
3) Shortest interval — sorted by $f_j - s_j$.

Earliest_finish_time ( n, $s_1$, .. $s_n$, $f_1$ .. $f_n$ ).
2) $S = \phi$ — [set of jobs selected; initially empty]
3) for j = 1 to n
4) if (job j is compatible with S)
5) $S = S \cup \{j\}$.
6) Return S.

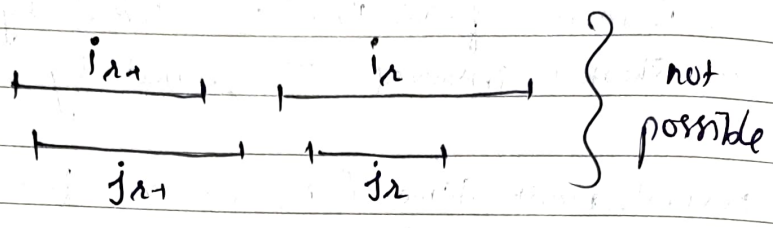
1) Sort jobs by finish time $f_1 \leq f_2 \leq f_3 \leq \dots \leq f_n$.

Proof that algorithm is optimal
[By contradiction].

1) Assume that greedy is not optimal.
2) Let $i_1, i_2, i_3, \dots i_k$ jobs selected by Greedy
Let $j_1, j_2, j_3, \dots j_m$ jobs selected by some optimal solution
3) Thus if greedy is not optimal
then $m > k$.

3) Now for any job r $f(i_r) \leq f(j_r)$
Because:
a) $r = 1$ [Base condition] $f(i_1) \leq f(j_1)$
as greedy selects job with minimum finish time
b) Assum that this is true for r-1
$f(i_{r-1}) \leq f(j_{r-1})$.
c) Now we show that $f(i_r) \leq f(j_r)$.

If this is not the case then it means —
$i_{r+1}$ finish after $j_{r+1}$.




However, this is not possible as the greedy algo always has the option of choosing $j_r$. Thus, it fulfill the induction step.

To be specific —

hypothesis — $\left.\begin{aligned} f(i_{r+1}) &\le f(j_r) \\ f(i_{r+1}) &\le f(j_{r+1}) \end{aligned}\right\}$ Combining these

$$f(i_{r+1}) \le s(j_r)$$

Thus $j_r$ is available when greedy selects $i_r$.
But still greedy selects $i_r$, which means

$$f(i_r) \le f(j_r).$$

4) Since $m > k$, there is job $j_{k+1}$ in O
Thus starts after $j_k$ ends. Means $j_{k+1}$ is compatible with other jobs.
But $f(i_k) \le f(j_k)$ and
greedy still stops at $i_k$ even when $j_{k+1}$ is available.

This contradicts our assumption that $m > k$

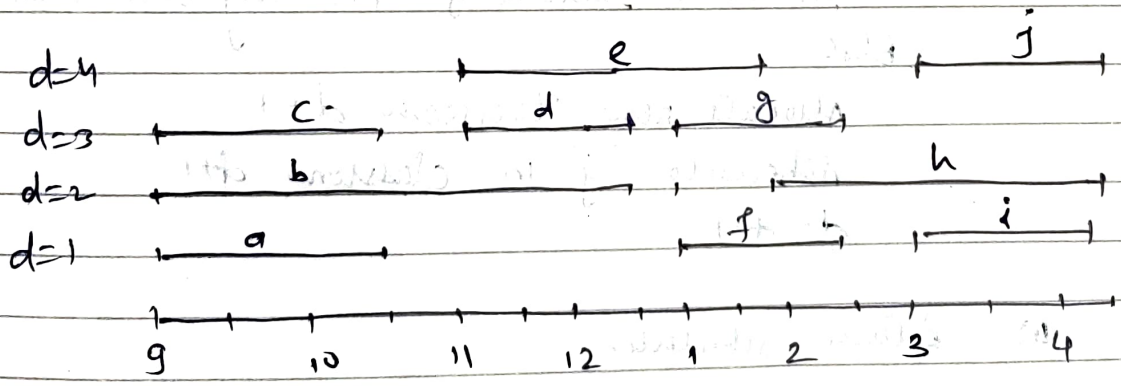
$\boxed{m = k}$.

Hence, algo is optimal.

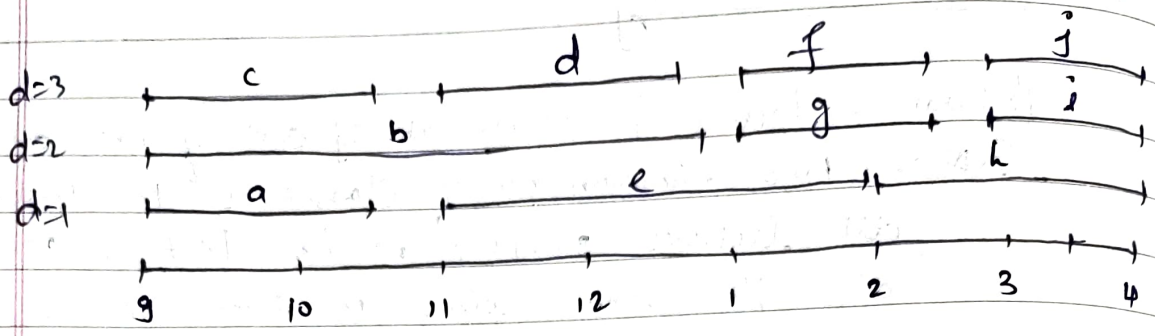
(3) * **Interval Partitioning**

**Problem** :- Lecture `j` starts at $s_j$ and finishes at $f_j$.
**Goal** :- Find minimum no. of classrooms to schedule all lectures so that no two lectures occur at the same time in same room.

| job | $s_j$ | $f_j$ |
|---|---|---|
| a | 9 | 10:30 |
| b | 9 | 12:30 |
| c | 9 | 10:30 |
| d | 11 | 14:30 |
| e | 11 | 2 |
| f | 1 | 2:30 |
| g | 1 | 2:30 |
| h | 2 | 4:30 |
| i | 3 | 4:30 |
| j | 3 | 4:30 |




One instance of interval partitioning.




## Optimal partitioning of classes

Earliest start time $(n, s_1 - s_n, f_1 - f_n)$

1) Sort lectures by start time and renumber so that $s_1 \leq s_2 \leq s_3 \leq \cdots \leq s_n$.

2) $d = 0$

3) for $j = 1$ to $n$
   if (lecture $j$ is compatible with classroom)
   schedule lecture 'j' in any such classroom 'k'.
   else
   Allocate new classroom $d+1$
   schedule 'j' in classroom $d+1$
   $d = d+1$

4) Return schedule

## Proof of optimality

Let d = max. no. of classes in conflict at any point of the time.

To prove – greedy algorithm uses 'd' classrooms to schedule all lectures.

Contradiction :- Let it uses more than 'd' classrooms

Let 'j' be the first lecture that is assigned to d+1.

Since we are allocating lectures according to start time, there are 'd' lectures that starts before 'j'.

Thus, at the start of 'j' there are at least 'd+1' lectures in conflict.

This contradict our assumption that depth is 'd'.

Hence greedy is optimal.

Analysis :-

- Sorting takes time nlogn.
- Store classroom is priority queue. (K = finish time of its last lecture).
- to allocate new classroom - INSERT in queue.
- schedule j in 'k' classroom - INCREASE key to $f_j$
- to find compatible classroom, compare $s_j$ with key of all classroom.

Total no. of operation is - n, each takes logn time.

$O(n \log n)$

(4) * Fractional Knapsack (Greedy).

Problem :-

Given the weights and values of 'N' items, in the form of (value, weight) put these items in a knapsack of capacity 'W'. to get the maximum total value in knapsack.

* Note :- In fractional knapsack, we can break item (to get partial profit) for maximizing total value of knapsack.

⇒ There are basically 3 approaches to solve this-

a) Select items based on maximum profit
b) Select items based on minimum weight
c) Select items based on maximum profit/weight.

Example :- (1)

| Items | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| Profit | 5 | 10 | 15 | 7 | 8 | 9 | 4 |
| weight | 1 | 3 | 5 | 4 | 1 | 3 | 2 |

W = 15.

Approach 1 Max. profit

| Item | Profit | Weight | Remaining weight |
|---|---|---|---|
| 3 | 15 | 5 | 15 - 5 = 10 |
| 2 | 10 | 3 | 10 - 3 = 7. |
| 6 | 9 | 3 | 7 - 3 = 4. |
| 5 | 8 | 1 | 4 - 1 = 3. |
| 4 | 7 * 3/4 | 4 | 3 - 4 * 3/4 = 0. |

47.25

Approach 2

| Item | profit | Weight | Remaining weight |
|---|---|---|---|
| 1 | 5 | 1 | 15 - 1 = 14. |
| 5 | 8 | 1 | 14 - 1 = 13. |
| 7 | 4 | 2 | 13 - 2 = 11 |
| 2 | 10 | 3 | 11 - 3 = 8 |
| 6 | 9 | 3 | 8 - 3 = 5 |
| 4 | 7 | 4 | 5 - 4 = 1 |
| 3 | 15 * 1/5 | 5 | 1 - 5 * 1/5 = 0 |

46.

Approach 3

| Item | profit / weight |
|---|---|
| 1 | 5/1 = 5 |
| 2 | 10/3 = 3.33 |
| 3 | 15/5 = 3 |
| 4 | 7/4 = 1.7 |
| 5 | 8/1 = 8 |
| 6 | 9/3 = 3 |
| 7 | 4/2 = 2 |

| Item | profit | weight | Remaining weight |
|---|---|---|---|
| 5 | 8 | 1 | 15-1 = 14 |
| 1 | 5 | 1 | 14-1 = 13 |
| 2 | 10 | 3 | 13-3 = 10 |
| 3 | 15 | 5 | 10-5 = 5 |
| 6 | 9 | 3 | 5-3 = 2 |
| 7 | 4 | 2 | 2-2 = 0 |

51

Thus, we get max. profit by using 3rd approach.

Algorithm :-

1) Sort items based on their profit/weight value.
   wt=0, x[i]=0 for i=1 to n,
2) for i= to n (OR wt < W)
   if wt + w[i] ≤ W
   x[i] = 1
   wt = wt + w[i].
   else
   x[i] = (W - wt/w[i])
   wt = W
   break
   sum=0
3) for i=1 to n
   sum = sum + x[i] * p[i].
4- Return sum.

5. (2)

| Items | A | B | C | D | E |
|---|---|---|---|---|---|
| weight | 1 | 4 | 3 | 7 | 6 |
| profit | 1 | 16 | 15 | 21 | 12 |

$W = 12$

1) Max Profit

| D | B | C | E | A |
|---|---|---|---|---|
| 7 | 4 | 3 | 6 | 1 |
| 21 | 16 | 15 | 12 | 1 |

| Item | weight | profit | total profit |
|---|---|---|---|
| D | 7 | 21 | 21 |
| B | 7+4=11 | 16 | 21+16=37 |
| C | $11+\frac{1}{3}\times 3$ | $\frac{1}{3}\times 15$ | 37+5 = (42) |
| | = 12 | | |

2) Min weight

| A | C | B | E | D |
|---|---|---|---|---|
| 1 | 3 | 4 | 6 | 7 |
| 1 | 15 | 16 | 12 | 21 |

| Item | weight | profit | total profit |
|---|---|---|---|
| A | 1 | 1 | 1 |
| C | 1+3=4 | 15 | 1+15=16 |
| B | 4+4=8 | 16 | 16+16=32 |
| E | $8+6\cdot\frac{4}{6}$ | $12\times\frac{4}{6}$ | 32+8 = (40) |
| | = 12 | | |

3) Max Ratio

| Item | weight | profit | total |
|---|---|---|---|
| C | 3 | 15 | 15 |
| B | 3+4=7 | 16 | 15+16=31 |
| D | $7+7\times\frac{5}{7}$ | $21\times\frac{5}{7}$ | 31+15 = (46) |
| | = 12 | | |

| C | B | D | E | A |
|---|---|---|---|---|
| 3 | 4 | 7 | 6 | 1 |
| 15 | 16 | 21 | 12 | 1 |
| 5 | 4 | 3 | 2 | 1 |

Date 21/2/2023
Page 4th

(5) * **Minimum Spanning Tree**

It is a spanning tree in which sum of weight of edges is as minimum as possible

A spanning tree is a sub graph of an undirected connected graph, which include all vertices with minimum possible no. of edges.

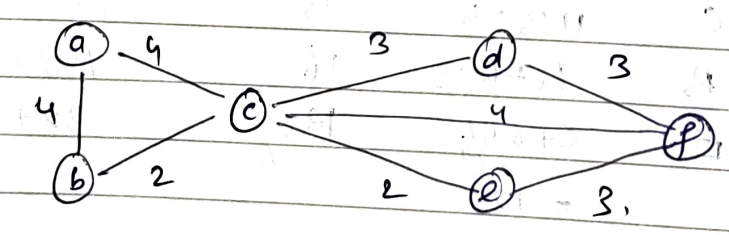
For n vertices, no. of edges in spanning tree = n-1

⇒ **Kruskal's algorithm** -

Following are steps involved in Kruskal's algo -

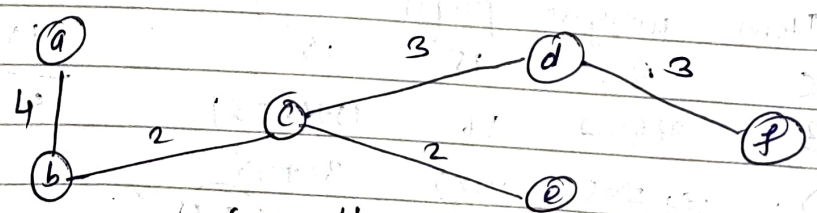
1) Sort all edges from low weight to high
2) Take the edge with lowest edge weight and add it to spanning tree.
   If adding edge creates a cycle, then reject it.
3) Keep adding edges until we have n-1 edges in the MST.

Ex.



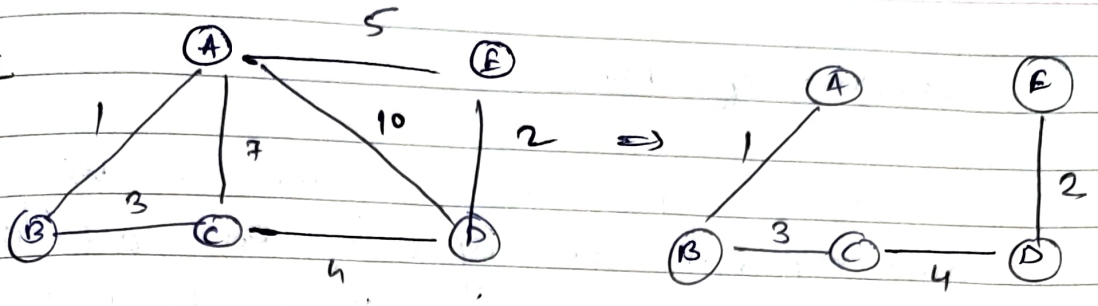

Sort b-e, c-e, c-d, d-f, e-f, a-b, a-c, c-f.
✓ ✓ ✓ ✓ ✗ ✓ ✗ ✗

MST




Cost = 14

Ex 2



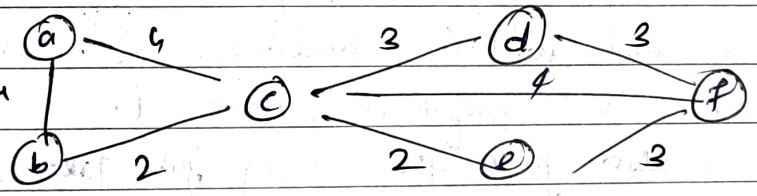

Cost = 10

⇒ Prims Algorithm

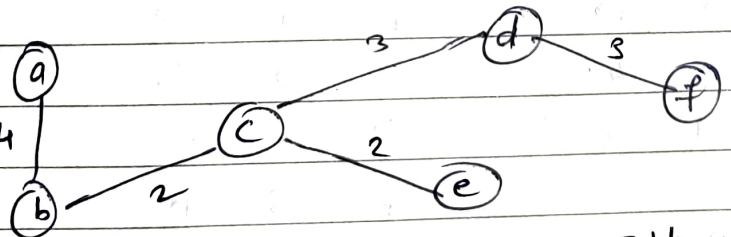
following are steps involved in Prim's algo.

1. Initialize the minimum spanning tree with random vertex.
2. Find all edges that connects the tree to new vertices, find the minimum and add it to the tree.
3. Keep repeating 2 until we get a minimum S.T.

Ex:




Let us start from 'c'.




MST with cost = 14

Complexity of Kruskal — $O(|V||E|) = O(n^2)$ if $|V|=|E|=n$

MSTs can be improved by using min heap.

$O(n \log n)$.

## Complexity of Prims

Simple implementation of Prims — using adjacency matrix or adjacency list and linearly searching an array of weight to find minimum weight edge to add — requires $O(|V|^2)$

This can be improved by using Heaps to implement finding minimum weight edges.

↳ $O(|E| \log |E|)$ — worst case.

## (6) * Job Sequencing with Deadline

A set of 'n' jobs are given with their deadline and profits. Profit is earned if the job is finished within deadline. Schedule jobs such that profit is maximized.

It is also given that every job takes single unit of time.

Ex:

| | | |
|---|---|---|
| A | 4 | 20 |
| B | 1 | 10 |
| C | 1 | 40 |
| D | 1 | 30 |

C, A = 60

| | | |
|---|---|---|
| A | 2 | 100 |
| B | 1 | 19 |
| C | 2 | 27 |
| D | 1 | 25 |
| E | 3 | 15 |

C, A, E = 142

Logic :- Sort jobs by max. profit and schedule jobs with max. profit for every slot.

Algorithm :-

1) Sort all jobs in decreasing order of profit
2) for each job, do following -
   a) find a time slot 'i', such that 'i' is empty, 'i' is ~~most~~ greatest and i < deadline
   b) If such 'i' exist, put job in 'i'.
      Else, ignore the job.

Ex ① A B C D → D B A C

2 1 1 2 → 2 1 2 1

60 80 50 100 → 100 80 60 50.

| | B | D | |
|---|---|---|---|
| 0 | 1 | | 2 |

i=2 for 'd' i≤2 Assign 'D'.

i=1 for B i≤1 Assign B.

Profit = 80 + 100 = 180

Ex ②

| A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|
| 7 | 2 | 5 | 3 | 4 | 5 | 2 | 7 | 3. |
| 15 | 20 | 30 | 18 | 18 | 10 | 23 | 16 | 25 |

Sort = C, I, G, B, D, E, H, A, F

Array =

| B | G | I | E | C | A | H |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7. |

~~7~~, i=5, Empty i≤5 Assign C

i=3 Assign I

i=2 Assign G

i=2 - Not empty for B

i=1 - Empty & i≤2 Assign B.

'D' - Ignore

'E' - Assign to 4.

H - Assign to 7

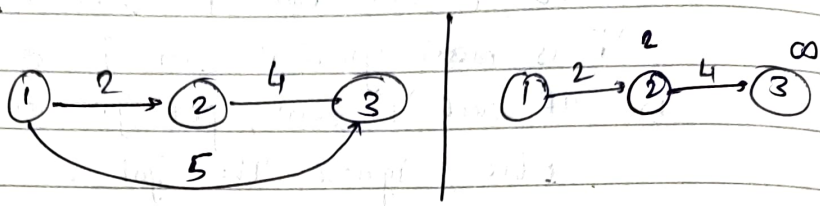
A - Assign to 6.

Profit = 20 + 23 + 25 + 18 + 30 + 15 + 16 = 147.

(7) * **Dijkstra Algorithm**

— Single source Shortest Path Problem

**Approach :-**




Cost to reach 3 = min. cost to reach [2] + C[2,3]

2 + 4 = 6 < ∞.

**In General**

Shortest Path to

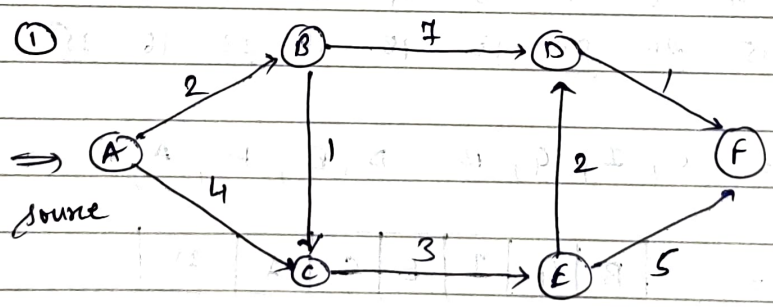
reach any node 'u' ⇒ d[u]

If there is edge u→v. then -

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

then d[v] = d[u] + c [u,v].

**Ex ①**




| Initially | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| | 0 | (2) | 4 | ∞ | ∞ | ∞. |

select minimum.

Edges from 'B' ⇒ B→C , B→D.

for B → C

$d[B] + c[B,C] = 2+1 = 3 < 4$

$d[C]$.

$d[C] = d[B] + c[B,C]$

$= 2+1$

$= 3$.

for B → D.

$d[B] + c[B,D] = 2+7 = 9$

$9 < \infty$ $[d[D]]$.

$\therefore d[D] = d[B] + c[B,D]$

$= 2+7$

$= 9$.

| A | B | C | D | E | f |
|---|---|---|---|---|---|
| 0 | 2 | (3) | 9 | ∞ | ∞ |

Select minimum.

for C → E $d[C] + c[C,E] = 3+3 = 6 < \infty$ $d[E]$

$d[E] = d[C] + c[C,E]$

$= 3+3 = 6$

| A | B | C | D | E | f |
|---|---|---|---|---|---|
| 0 | 2 | 3 | 9 | (6) | ∞ |

Select minimum.

E → D

$d[E] + c[E,D] = 6+2 = 8 < 9$.

$d[D] = 8$.

E → F

$d[E] + c[E,F] = 6+5 = 11 < \infty$.

$d[F] = 11$.

| A | B | C | D | E | F |
|---|---|---|---|---|---|
| 0 | 2 | 3 | (8) | 6 | 11 |

D → F  d[D] + c[D,F] = 8+1 = 9. < 11

d[F] = 9

| A | B | C | D | E | F |
|---|---|---|---|---|---|
| 0 | 2 | 3 | 8 | 6 | 9. |

Overall -

| Selected vertex | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| B | 0 | (2) | 4 | ∞ | ∞ | ∞ |
| C | 0 | 2 | (3) | 9 | ∞ | ∞ |
| E | 0 | 2 | 3 | 9 | (6) | ∞ |
| D | 0 | 2 | 3 | (8) | 6 | 11 |
| F | 0 | 2 | 3 | 8 | 6 | (9) |

Path ⇒ A → B → C → E → D → F.

Algorithm (G, S)

for each V in G

distance[V] = c[S,V] if there is direct edge

else distance[V] = ∞.

previous[V] = NULL

If V != S add V to to priority Q.

while Q is not empty

U = Extract min from Q.

for each neighbor V of U

t = distance[U] + c[U,V].

```
If (t < distance[v])
    distance[v] = t
    previous[v] = u

Return distance[], previous[].
```

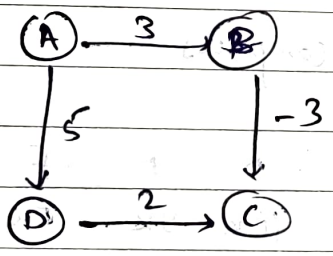
**Complexity -**

To find minimum cost = $|V|$

To update the cost = $|V|-1$

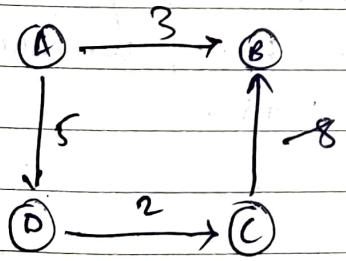
Complexity = $O(|V|^2) = O(n^2)$

**Drawback :-** When we have -ve weight, we may not get optimal solution using Dijkstra.




| | A | B | C | D |
|---|---|---|---|---|
| B | 0 | (3) | ∞ | 5 |
| C | 0 | 3 | (0) | 5 |
| D | 0 | 3 | 0 | (5) |
| | 0 | 3 | 0 | 5 |

Worked even if there is -ve edge




| | A | B | C | D |
|---|---|---|---|---|
| B | 0 | (3) | ∞ | 5 |
| D | 0 | 3 | ∞ | (5) |
| C | 0 | 3 | (7) | 5 |
| | 0 | -1 | 7 | 5 |

C→B Already checked so no need to change.

Thus Dijkstra give d(B) = 3 But it is -1

**Bellman ford** works for -ve edge weight.

## (8) * Huffman Coding

- Data can be encoded efficiently
- Widely used technique for data compressing
- This algorithm use table of frequency of occurrences of each character to build optimal way of representing each character as binary string.

**Motivation** ASCII - A-65 B-66 ... 01000001, 01000010...

Suppose we want to store $10^5$ character in file.

Normal Storage = $8 \times 10^5$ [Each character (ASCII) require 8 bit for storage]

However, we can compress -

| Ex: | A | B | C | D | E | F | Total |
|---|---|---|---|---|---|---|---|
| Freq. | 45k | 13k | 12k | 16k | 9k | 5k | 100k |

Fixed length coding - Each letter is represented by an equal no. of bits. Here, we can represent using 3 bits.

| | |
|---|---|
| A | 000 |
| B | 001 |
| C | 010 |
| D | 011 |
| E | 100 |
| F | 101 |

Total file storage = $3 \times 10^5$

| | |
|---|---|
| 8×6 bits | 3×6 bits |

48+18 = 66 bits for decoding the message

## Variable length coding -

Characters that occurs max no. of times can be represented by less no. of bits and vice versa.

Example -

| | |
|---|---|
| A | 0 |
| B | 101 |
| C | 100 |
| D | 111 |
| E | 1101 |
| F | 1100 |

No. of bits = $(45\times1) + (13\times3) + (12\times3) + (16\times3) + (9\times4) + (5\times4)$

$= 45 + 39 + 36 + 48 + 36 + 20$

$= 224 \times 1000$

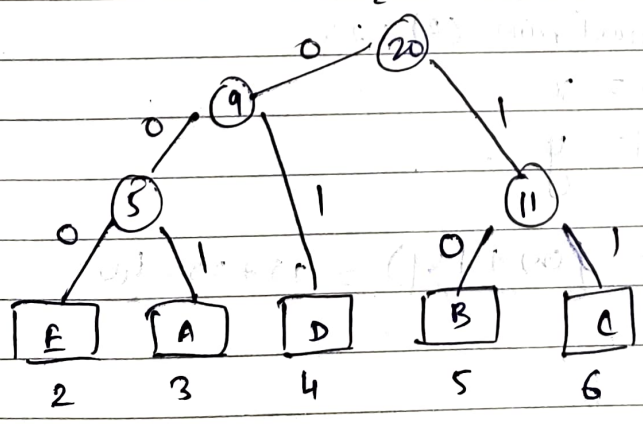
$= \underline{2.24 \times 10^5}$ ← 25% saving

Huffman obtained these variable length code for the given message

Ex: Message - B C C A B B D D A E C C B B A E D D C C

$f(M) = \{3, 5, 6, 4, 2\}$ n = 5 length = 20.

Original size = $20\times8 = 160$




| Char | count | code | |
|---|---|---|---|
| A | 3 | 00 | 3×3=9 |
| B | 5 | 10 | 2×5=10 |
| C | 6 | 11 | 2×6=12 |
| D | 4 | 01 | 2×4=8 |
| E | 2 | 000 | 3×2=6 |

8×5 + 12 = 45 bits Message

[52 bits for table/tree]

Algorithm:-

Huffman (C)

1. n = |C|  Q = C
2. for i = 1 to n-1
3. Z = Allocate New_Node ( )
4. x = left[Z] = Extract_Min (Q)
5. y = right[Z] = Extract_Min (Q)
6. f[Z] = f[x] + f[y]
7. Insert (Q, Z)
8. Return Q.

Ex (2) A: 50 B: 25 C: 15 D: 40 E: 75

C = { A, B, C, D, E }.

f(C) = { 50, 25, 15, 40, 75 }.

n = 5

Q = C

ie.

| C \| 15 | | B \| 25 | | D \| 40 | | A \| 50 | | E \| 75 |
|---|---|---|---|---|---|---|---|---|---|

for i=1 Z = New node

x = Extract min (Q) = 15

y = Extract min (Q) = 25

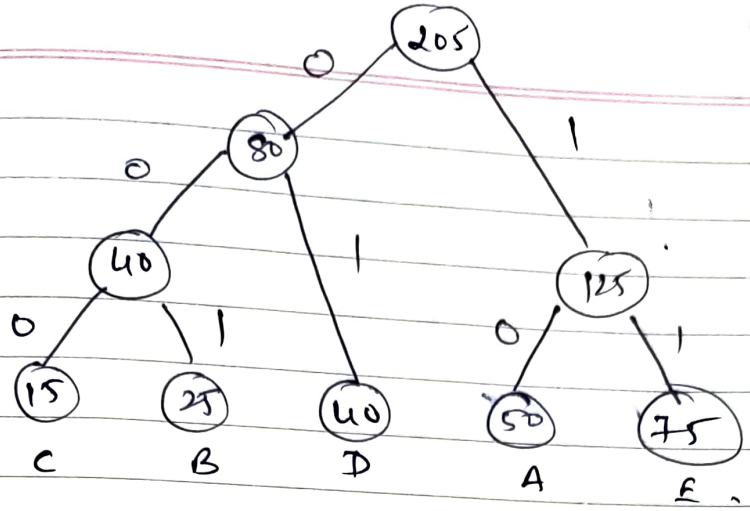
left[Z] = x

Right[Z] = y

f(z) = f(x) + f(y) = 15 + 25 = 40.

Similarly

for i=2 i=3 i=4.




| | | | |
|---|---|---|---|
| A | 10 | 50 | 2×50 = 100 |
| B | 001 | 25 | 3×25 = 75 |
| C | 000 | 15 | 3×15 = 45 |
| D | 01 | 40 | 2×40 = 80 |
| E | 11 | 75 | 2×75 = 150 |
| 5×8 | 12 | | 382 bits Message |

Original length = 205 × 8 = 1640

Compressed length = 382 + 52 = 434 bits
Message Table