UNIT III

Reference Books

- T.H Coreman "Introduct" to Algo"
- S. Sahni "Fundamentals of Computer Algo".

DIVIDE & CONQUER

- Finding Max" & Min" element problem.
- Matrix Multiplication (M.M)
  - Naive method without DAC
  - M.M using DAC
  - Strassen's M.M using DAC
- Convex HULL & searching.
- Binary search, Quick Sort, Merge Sort] covered in Unit I

GREEDY METHODS

- Intro and applications of Greedy
- Optimal Reliability Allocation
- Knapsack < Fractional (OPTIMAL solut" using GREEDY METHOD)
  0/1 knapsack (optimal solut" using DYNAMIC PROG")
- M.S.T < Prim's
  Kruskal's
- Single Source Shortest Paths
  - Dijkstra's Algo
  - Bellman Ford Algo.

DIVIDE & CONQUER

Study Theory Yourself

MATRIX MULTIPLICATION

There are different algo for matrix multiplication of two square matrices A and B of size $n \times n$ each.

1. NAIVE METHOD (Matrix Multiplicat^n Without using DAC)

```
Matrix_Multiply(A[][], B[][], C[][])  // Θ(n^3) T.C.
{ for(i=0 to n)
  { for(j=0 to n)
    { C[i][j] = 0
      for(k=0 to n)
      { C[i][j] += A[i][k] * B[k][j]
    }
  }
}
```

2. Matrix Multiplication Using DAC

Let $C = A * B$ where A, B & C are $n \times n$ matrices.
Assume n is exact power of 2. (i.e 2×2, 4×4, 8×8 ...)
We divide each A, B & C into four matrices of size $\frac{n}{2} \times \frac{n}{2}$
Let $C = A * B$ is written as

$$\begin{bmatrix} r & s \\ t & u \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} * \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

Above equat^n corresponds to four equations

$r = a.e + b.g$ // 2 Multiplicat^n of size $\frac{n}{2} \times \frac{n}{2}$
1 Addition of size $\frac{n}{2} \times \frac{n}{2}$

$s = a.f + b.h$ //

$t = c.e + d.g$ //

$u = c.f + d.h$ //

P.T.O →

From the equations we have

1 Matrix Multiplication of size $n \times n$

8 Matrix Multiplication of size $\frac{n}{2} \times \frac{n}{2}$

4 Matrix Addition of size $\frac{n}{2} \times \frac{n}{2}$

We can define straight forward DIVIDE & CONQUER strategy using those four equations.
We have Recurrence Relation to multiply two nxn matrices.

$$T(n) = \begin{cases} 1 & ; n \leq 2 \\ 8T\left(\frac{n}{2}\right) + n^2 & ; n > 2 \end{cases}$$

$T(n) = \Theta(n^3)$

The solution for $T(n)$ is no faster than the ordinary algo.
Infact Matrix Multiplication using DAC is taking extra space for recursive call.

NOTE * Matrix Multiplication without using DAC i.e NAIVE METHOD is better than the Matrix Multiplicat^n using DAC because NAIVE METHOD don't take extra space in executing the algo.

* $C_{mp} = A_{mn} * B_{np}$ ← multiplicat^n of two matrices

$C_{mp} = A_{mn} + B_{np}$ ← Addit^n of two matrices

No: of elements in matrix = $m * p$

No: of multiplications = $m * n * p$

Time complexity for Mullipl^n = $O(n^3)$

Time complexity for Addit^n = $O(1)$

Q. $A = \begin{bmatrix} \overbrace{10 & 20}^{A_{11}} & \overbrace{30 & 40}^{A_{12}} \\ 50 & 60 & 70 & 80 \\ \underbrace{90 & 100}_{A_{21}} & \underbrace{110 & 120}_{A_{22}} \\ 130 & 140 & 150 & 160 \end{bmatrix}_{4\times4}$ $B = \begin{bmatrix} \overbrace{1 & 2}^{B_{11}} & \overbrace{3 & 4}^{B_{12}} \\ 5 & 6 & 7 & 8 \\ \underbrace{9 & 10}_{B_{21}} & \underbrace{11 & 12}_{B_{22}} \\ 13 & 14 & 15 & 16 \end{bmatrix}_{4\times4}$

Find $C = A * B$

Sol C matrix is given as

$$C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

$$C_{11} = A_{11}B_{11} + A_{12}B_{21}$$

$$C_{12} = A_{11}B_{12} + A_{12}B_{22}$$

$$C_{21} = A_{21}B_{11} + A_{22}B_{21}$$

$$C_{22} = A_{21}B_{12} + A_{22}B_{22}$$

We see that

1 Matrix Multiplications of size 4x4

8 Matrix Multiplications of size 2x2

4 Additions of Matrix of size 2x2

V. Imp

3. STRASSEN'S ALGO (using DAC)

Volker Strassen discovered different recursive approach that required only 7 recursive multiplication of $\frac{n}{2} \times \frac{n}{2}$ matrices and $\theta(n^2)$ scalar Additions & substractions yielding the Recurrence

$$T(n) = \begin{cases} 1 & ; n \leq 2 \\ 7T\left(\frac{n}{2}\right) + n^2 & ; n > 2 \end{cases}$$

$$T(n) = \theta(n^{\log 7}) \approx O(n^{2.81})$$

Strassen's Method has 4 steps:

① Divide the i/p matrices A & B into $\frac{n}{2} \times \frac{n}{2}$ submatrices.

② Using $\theta(n^2)$ scalar Additions & Subtraction, compute 14 matrices $A_1, B_1, A_2, B_2, \ldots\ldots A_7 B_7$ each of which is of size $\frac{n}{2} \times \frac{n}{2}$

③ Recursively compute the seven matrix products $P_i = A_i B_i$ for $i = 1, 2, \ldots\ldots 7$.

④ Compute the desired submatrices r, s, t, u of the result matrix C by adding and/or subtracting various combinations of the P_i matrices, using only $\theta(n^2)$ Scalar Additions and subtractions.

Let $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$

P.T.O →

$B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$

$C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} * \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$

In Strassen's Method we first compute 7 sub-matrix products P1, P2, P3 .... P7 and then C_{ij} are computed.

	Multipl^n	Addit^n	Subtract^n
$P1 = (A_{11} + A_{22})(B_{11} + B_{22})$	1	2	–
$P2 = (A_{21} + A_{22}) B_{11}$	1	1	–
$P3 = A_{11}(B_{12} - B_{22})$	1	–	1
$P4 = A_{22}(B_{21} - B_{11})$	1	–	1
$P5 = (A_{11} + A_{12}) B_{22}$	1	1	–
$P6 = (A_{21} - A_{11})(B_{11} + B_{12})$	1	1	1
$P7 = (A_{12} - A_{22})(B_{21} + B_{22})$	1	1	1
	7	6 + 10	4

C_{ij}'s are computed using following.

$C_{11} = P1 + P4 - P5 + P7$

$C_{12} = P3 + P5$

$C_{21} = P2 + P4$

$C_{22} = P1 + P3 - P2 + P6$

As we can see P1, P2 .... P7 can be computed using
7 Matrix Multiplications and
10 Matrix Additions or subtractions.
C_{ij}'s require an additional 8 Matrix Addition or Substract^n.

The resulting recurrence relation in general is given as

$$T(n) = \begin{cases} b & ; n \le 2 \text{, } b \text{ is const.} \\ 7T\left(\frac{n}{2}\right) + an^2 & ; n > 2 \text{, } a \text{ is const.} \end{cases}$$

- Strassen's method is not preferred for practical application because
  - Constant used in it's method are high and for typical application NAIVE METHOD works better.
  - For sparse Matrices, there are better methods designed for them.
  - The sub-matrices in recursion takes extra space.
  - Large errors accumulate in it.

NOTE:

Algo for Matrix Multiplications	Time Complexity (T.C)	Space Complexity (S.C)
* NAIVE METHOD (without using DAC)	$O(n^3)$	NIL
* Using DAC	$O(n^3)$	$O(\log n)$
* Strassen's Algo (using DAC)	$O(n^{2.81})$	Little bit extra than using DAC S.C

➤ REFERENCE BOOK : S. SAHNI

CONVEX HULL

- Important structure in geometry
- Used in construction of many other geometric structures.
- Convex Hull of a set 'S' of points in plane is defined to be the smallest convex polygon containing all the points of 'S'.
- A polygon is defined to be convex if for any two points P_1 & P_2 inside the polygon, the directed line segment from P_1 to P_2 (denoted as $\langle P_1, P_2 \rangle$) is fully contained in the polygon.

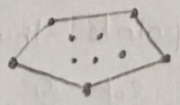


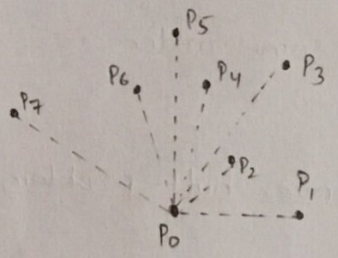
fig. Convex HULL

- The vertices of the convex hull of a set 'S' of points form a subset (not necessarily proper subset) of 'S'.
- There are two variants of the convex hull problem:
  1. obtain the vertices of the convex hull (these vertices are called extreme points)
  2. obtain the vertices of the convex hull in some order i.e clockwise or anticlockwise.

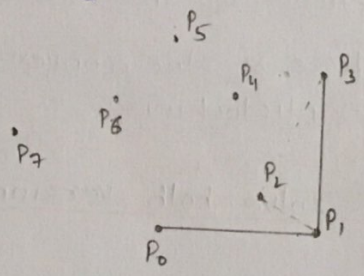
- Using DAC we can solve both versions of convex hull problem in $\theta(n \cdot \log n)$ time.
- **QuickHull** Algo similar to Quick sort algo solves the convex hull problem in $\theta(n \cdot \log n)$ Avg. Case.
  $\theta(n^2)$ worst case.
- **Grahm's Scan** Algo solves Convex Hull prob by maintaining a stack of main points. and runs in $\theta(n \cdot \log n)$ time.
- A **simple DAC** Algo based on divide the points & combine them later runs in $\theta(n \cdot \log n)$ time.

GRAHM'S SCAN

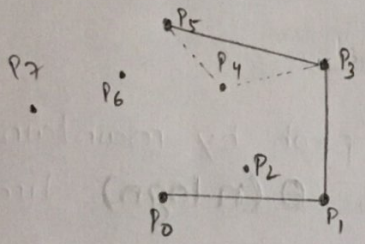
- It solves the convex hull problem by maintaining a stack of main points
- Each points of the input set Q is pushed once into the stack.
- The points that are not the vertices of convex Hull having input set Q are eventually popped.
- When algo terminates, stack contains exactly the vertices of convex Hull input set Q in counter-clockwise order of their appearence on the boundary.



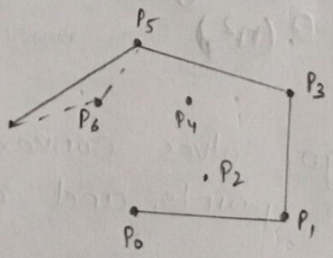
(i)



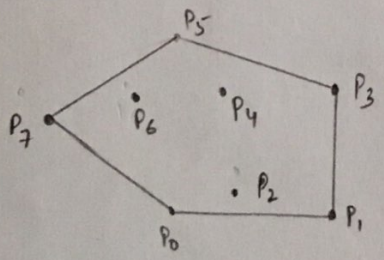
(ii)



(iii)



(iv)



(v)

fig (v) shows convex hull having input set $Q = \{P_0, P_1, P_2 \ldots P_7\}$ and the vertices of convex hull is subset of Q i.e $\{P_0, P_1, P_3, P_5 \& P_7\}$

GREEDY METHOD:

- one of the designing technique of algo
- It can be applied to wide variety of problems

OPTIMIZATION PROBLEM

- Given a problem instance, a set of constraints & an objective function.
  ↳ specify limitations on required solut^n
- Find a feasible solution for the given instance
- either maximum or minimum depending on the problem being solved.

SOLUTION For Optimization Problems

1. Dynamic Programming
2. Greedy Technique } simpler and more efficient for some optimizat^n problem.

DYNAMIC PROG^n Vs GREEDY METHOD

-At each step, the choice is determined based on solutions of sub-prob.	-At each step, it quickly make a choice that currently looks best. A local optimal (greedy) choice.
-Sub-prob. are solved first	-Greedy choice can be best first before solving further sub-prob.
-Bottom-up Approach	- Top-down Approach
-Can be slower & more complex.	-Usually faster & simple but may not give you optimal solution in some cases.

- **Characteristics of GREEDY METHOD**
  - Make a sequence of choices
  - each choice is the one that seems best (locally)
  - Choice produces a smaller problem to be solved.

- **PHASES OF GREEDY METHOD**

  Greedy Algo works in phases. At each phase
  - It takes the best solution right now, without regard for future consequences.
  - It choose a local optimum solut^{n} at each step & end up at a global optimum solut^{n}.

- SOLUTION SPACE: Set of all possible solut^{n} over given 'n' no: of inpul-

  FEASIBLE SOLUTION: Set of solut^{n} which will satisfy our condition.

  OPTIMAL SOLUTION: One of the Feasible solut^{n} which is giving optimal value.

- **APPLICATIONS OF GREEDY METHOD**

  1. Job Sequencing Deadline
  2. Knapsack Problem
  3. Huffman Coding
  4. Optimal Merge Pattern
  5. Minimum Cost Spanning Tree
     - PRIM'S ALGO
     - KRUSKAL'S ALGO
  6. Single Source Shortest Path
     - DIJKSTRA ALGO
     - BELLMAN FORD ALGO
     - BREADTH FIRST TRAVERSAL ALGO

KNAPSACK PROBLEM

1. 0-1 Knapsack Prob.

- It can be solved using DYNAMIC PROG^{n}
- A thief robbing a store finds n items.

  i^{th} item has profit P[i] and weight W[i]

  He can carry atmost 'm' pounds in knapsack

  He can either take item or leave item.

2. Fractional knapsack Prob

- It can be solved using GREEDY METHOD.
- Same as 0-1 knapsack.

  He can take any fraction of item.

OPTIMAL KNAPSACK ALGO (for Fractional knapsack)

```
Fractional_knapsack (Array P, Array w, int m)
{ 1. for (i = 1 to size(P))          // O(n)
  2.    do P[i] = P[i] / w[i]
  3. Sort elements of P[i] in Descending Order   // O(n.logn)
  4. i = 0
  5. while (m > 0)                   // O(n)
  6.    { do amount = min(m, w[i])
  7.        Solution[i] = amount
  8.              m = m - amount
    }
  9. return Solution
}
```

Time Complexity = $O(n) + O(n.\log n) + O(n) = O(n.\log n)$

Que:1. Find Maximum Profit using GREEDY KNAPSACK Prob.
Given capacity of knapsack (m) = 20kg.

Object	1	2	3	4
Profit	200	250	300	180
Weight	10	8	12	5

Sol^{n} Object 1 with 10 unit weight has Profit 200
" " 1 " " " " $\frac{200}{10} = 20$

i.e $A[1] = \frac{P[1]}{W[1]} = \frac{200}{10} = 20$

Similarly

$A[2] = \frac{P[2]}{W[2]} = \frac{250}{8} = 31.25$

$A[3] = \frac{P[3]}{W[3]} = \frac{300}{12} = 25$

$A[4] = \frac{P[4]}{W[4]} = \frac{180}{5} = 36$

Arrange the elements of Array in decreasing order.

$36 > 31 > 25 > 20$

Capacity of knapsack (m) = 20
Subtract weight of object from capacity of knapsack till m is ~~m is~~ full

i.e $20 - 5 = 15$
$15 - 8 = 7$
$7 - 12 \times \frac{7}{12} = 0$

Profit $= 180 + 250 + 300 \times \frac{7}{12}$
$= 605$

$\left(\frac{0}{A1}, \frac{1}{A2}, \frac{7/12}{A3}, \frac{1}{A4} \right)$

Q2. Find Max^m Profit Using GREEDY METHOD
Given capacity of knapsack (m) = 33

OBJECT	1	2	3	4	5	6	7
PROFIT	50	70	30	60	20	55	45
WEIGHT	5	9	8	5	4	5	8

Sol^n

P[i] / w[i]	10	7.7	3.7	12	5	11	5.6

Arrange in decreasing Order

$12 > 11 > 10 > 7.7 > 5.6 > 5 > 3.7$
(obj 4, obj 6, obj 1, obj 2, obj 7, obj 5, obj 3)

Capacity of knapsack (m) = 33
Max^n Profit Initially (P) = 0

$m - \underbrace{w[4]}_{\text{weight of obj 4}} = 33 - 5 = 28$ $\quad$ Profit = $0 + \underbrace{P[4]}_{\text{Profit of obj 4}} = 60$

$28 - w[6] = 28 - 5 = 23$ $\quad = 60 + P[6] = 115$

$23 - w[1] = 23 - 5 = 18$ $\quad = 115 + P[1] = 165$

$18 - w[2] = 18 - 9 = 9$ $\quad = 165 + P[2] = 235$

$9 - w[7] = 9 - 8 = 1$ $\quad = 235 + 45 = 280$

$1 - w[5] = 1 - 4 \times \frac{1}{4} = 0$ $\quad = 280 + 20 \times \frac{1}{4} = 285$

∴ Max^n Profit = **285**

Capacity of knapsack full
will all objects excepts obj 3

$\left(\frac{1}{x_1}, \frac{1}{x_2}, \frac{0}{x_3}, \frac{1}{x_4}, \frac{1/4}{x_5}, \frac{1}{x_6}, \frac{1}{x_7} \right)$

Q3. Find Max^m profit using GREEDY KNAPSACK
Given capacity of knapsack (m) = 20

obj	1	2	3
Profit	24	25	10
Weight	15	19	15

Sol^n $\frac{P[i]}{w[i]}$ = 1.6 1.31 1.5

Arrange in decreasing order.

1.6 > 1.5 > 1.31
obj 1 obj 3 obj 2

Capacity of knapsack (m) = 20 ; Profit = 0 initially.

m - w[1] = 20 - 15 = 5

5 - w[3] = 5 - 15 × $\frac{5}{15}$ = 1 = 0

Profit = 0 + P[1] = 0 + 24 = 24
= 24 + P[3] = 24 + 10 × $\frac{1}{3}$

∴ Max^m Profit = 27.3

capacity of knapsack full

$\left(\frac{1}{x1}, \frac{0}{x2}, \frac{1/3}{x3}\right)$

Q4. Given Capacity of knapsack (m) = 12. find Profit using FRACTIONAL KNAPSACK.

object	1	2	3	4	5
Profit	5	2	2	4	5
weight	5	4	6	2	1

Sol^n $\frac{P[i]}{w[i]}$ 1 0.5 0.3 2 5

Arrange $\frac{P[i]}{w[i]}$ in Decreasing order 5 > 2 > 1 > 0.5 > 0.3
obj 5 obj 4 obj 1 obj 2 obj 3

Capacity of knapsack (m) = 12 ; Profit = 0 intially.

m - w[5] = 12 - 1 = 11
11 - 2 = 9
9 - 5 = 4
4 - 4 = 0

Profit = 0 + 5 + 4 + 5 + 2 = 16

∴ $\left(\frac{1}{x1}, \frac{1}{x2}, \frac{0}{x3}, \frac{1}{x4}, \frac{1}{x5}\right)$

OPTIMAL KNAPSACK ALGO (for 0-1 knapsack)

- 0-1 knapsack can be solved efficiently using Dynamic Progⁿ technique.

```
0_1knapsack(m,n)
{ 1. if(m==0 || n==0)
  2.    return ;
  3. else if(W[n] > m)
  4.    return 0_1_knapsack(m,n-1);
  5. else if(Table[m-W[n], n-1] == NULL)
  6.    { a = 0_1_knapsack(m-W[n], n-1) + P[n]
  7.      b = 0_1_knapsack(m, n-1)
  8.      c = max(a,b)
    }
  9. return c;
}
```

- Recurrence Relation

V.Imp

$$0\_1\_knapsack(m,n) = \begin{cases} 0 & ; m = 0 \text{ or } n = 0 \quad (m: \text{capacity}, n: \text{object}) \\ 0\_1\_knapsack(m, n-1) & ; W[n] > m \\ \max\{0\_1\_knapsack(m-W[n], n-1) + P[n],\ 0\_1\_knapsack(m, n-1)\} & ; w[n] \le m \quad (w[n]: \text{weight of object}) \end{cases}$$

- Time Complexity for Dynamic knapsack Prob $= O(n^2)$

  Space Complexity for " " " $= O(n^2)$

  ⇓

  i/p + extra

  i/p → n

  extra → stack ($n+1$), Table ($m*n$)

Q1. Find the max^m profit for knapsack Problem.

a.) using Fractional knapsack

b.) using 0-1 knapsack

Given capacity of knapsack (m) = 10

object	1	2	3
Profit	35	40	70
weight	5	5	7
$\frac{P_i}{w_i}$ =	7	8	10

Sol^n a.) Fractional knapsack.

Greedy Method.

finds $\frac{P[i]}{w[i]}$

Arrange in decreasing order

$10 > 8 > 7$

capacity of knapsack = 10

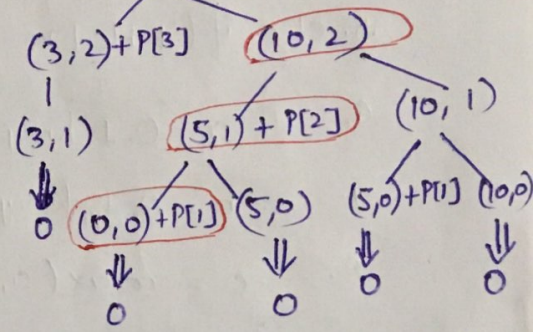
$m - w[3] = 10 - 7 = 3$; Profit = 70

$3 - w[2] = 3 - 5 \times \frac{3}{5} = 0$ $= 70 + 40 \times \frac{3}{5}$

$= 94$

Dynamic Method

0-1Knap(m, n) = 10, 3



Profit = 0 + P[1] + P[2]

= 75

b.) 0-1 knapsack

Greedy Method

Finds $\frac{P[i]}{w[i]}$

Arrange in dec. order.

$10 > 8 > 7$

Capacity of knapsack = 10

$m - w[3] = 10 - 7 = 3$

Now Greedy can't fill the bag with any of the two objects left. because it can't take fraction here

$\therefore$ Profit = 0 + P[3] = 70

Dynamic Method

. . . . . .

Same as above

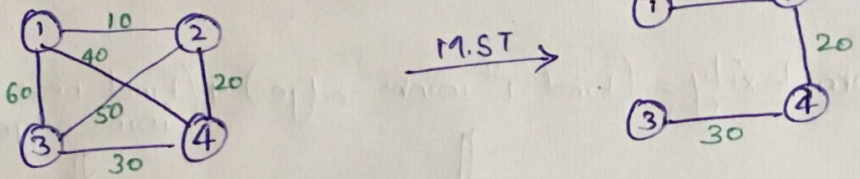
. . . . . .

Profit = 75

MINIMUM COST SPANNING TREE (MST)

- Simple Graph
- Undirected
- Weighted
- Given a graph G(V, E) with V vertices & E edges, then graph G'(V, E') is the MST where G' contains all the vertices of G and the edges E' with minimum weight 'w'

ex:



- To construct M.ST there are two Algo
  1. Kruskal's Algo.
  2. Prim's Algo

• KRUSKAL'S ALGO

- It finds M.S.T where you may get disconnected graph in between but at the end you will get connected graph.
- ALGO:

{
Study the Algo from BOOK.
COREMAN
}

- Time Complexity = (Take minim" edge everytime) + (Add to M.S.T)

$$O(E.\log E) + O(E)$$

$$= O(E.\log E)$$

$$\approx O(E.\log V)$$ using Binary heap

PRIM'S ALGO

- Every step generates connected Graph
- It finds new min^{n} edge adjacent to previous min^{n} edge
- ALGO:

{ :

Study the Algo from BOOK COREMAN

}

- Time Complexity = (Find 1st min^{m} edge) + (Find new adjacent min^{n} edge)

$O(V) + O(V^2)$

$= O(V^2)$ using Array

- NOTE : Time Complexity of Prim's Algo.

$= (V \cdot \log V) + (E \log V) = O((V+E) \log V)$ using BINARY HEAP

$= O(E + V \cdot \log V)$ using FIBONACCI HEAP

$= O(V+E)$ using BINOMIAL HEAP

$= O(V^2)$ using ARRAY

$= O(V^3)$ using SORTED/UNSORTED LINKED LIST.

SINGLE SOURCE SHORTEST PATH

- In Single Source Shortest Path Problem, we consider A Given graph $G=(V,E)$.
  We want to find a shortest path from source 's' to every vertex v.
  where V : Vertices in graph G
  E : Edges in graph G
  $s \in V$
  $v \in V$

- Two algorithms to solve above prob.
  (1) Dijkstra's Algo (2) Bellman-Ford Algo

• DIJKSTRA'S ALGO

- Edsger Dijkstra discovered this algo.
- This algo is greedy because everytime it looks for min^m distance or weight from source to v.
- ALGO : {
  Study Algo from BOOK COREMAN
  }
- Time Complexity =

(Everytime finding min^n distance from source to v) + (Each time delete min^n distance visted & find all it's neighbors)

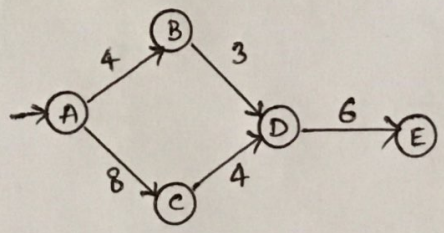
$O(E)$ + $O(V^2)$ = $O(V^2)$

NOTE: Time complexity of Dijkstra's algo can be improved
using Binary Heap = $O(E.\log V) + O(V.\log V) = O(E.\log V)$
using Fibonacci Heap = $O(E) + O(V.\log V) = O(V.\log V)$

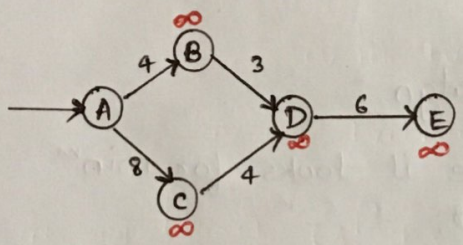
Q1 Solve the shortest path problems using Dijkstra's Algo.

a.) Find the output sequence of vertices.

b.) What will be the cost of shortest path from source 'A' to destination 'E'.



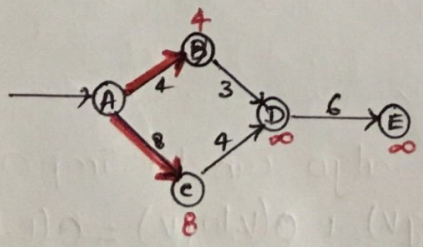
Sol^{n}: Initially, From source 'A' to all vertices, the distance is set to ∞.



V:	A	B	C	D	E
	0	∞	∞	∞	∞

↳ mark the one who is minim^{n}

- Find minimum from the vertices set { i.e Vertex with min^{n} cost from source. } Extract_Min(V) func^{n} is used to find the ~~min^{m}~~ vertex with min^{n} distance from source.
- We get A ← Extract Min(V):
- Let S be the set which contains vertices that comes in the ~~po~~ shortest path from source to Destinat^{n}
- Now Relax all the edges leaving 'A'.

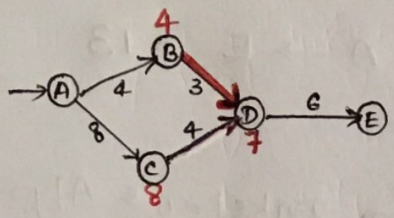


V:	A	B	C	D	E
	0	∞	∞	∞	∞
S={A}	-	4	8	∞	∞

↳ mark the one who is min^{m}

'B' ← EXTRACT_MIN (V)

Relax all the edges leaving 'B'

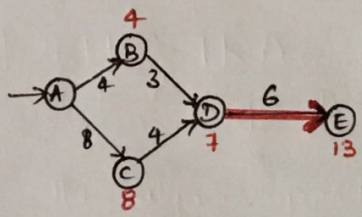


V:	A	B	C	D	E
	0	∞	∞	∞	∞
	-	4	8	∞	∞
S={A,B}	-	-	8	**7**	∞

↳ mark the one who is min^m

'D' ← EXTRACT_MIN (V)

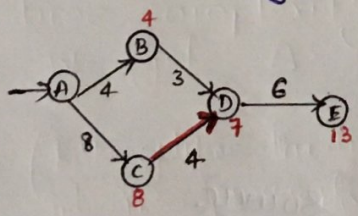
Relax all the edges leaving 'D'



V:	A	B	C	D	E
	0	∞	∞	∞	∞
	-	4	8	∞	∞
	-	-	8	7	∞
S={A,B,D}	-	-	**8**	-	13

'C' ← EXTRACT_MIN (V)

Relax all the edges leaving 'C'



V:	A	B	C	D	E
	0	∞	∞	∞	∞
	-	4	8	∞	∞
	-	-	8	7	∞
S={A,B,D,C}	-	-	-	-	**13**

'E' ← EXTRACT_MIN (V)

Relax all the edges leaving E

Since NO edges leaving vertex E & the destinal" is E itself you get the total cost to reach the destination from the source

↳ Shortest path.

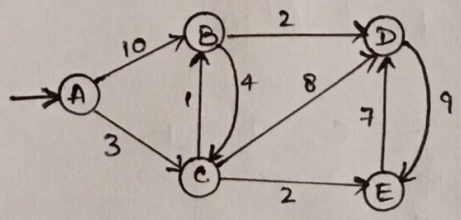
P.T.O →

a.) the output sequence of vertices

$S = \{ A, B, D, C, E \}$

b.) the cost of shortest path from $A \rightarrow E = 13$

Q2. Solve the shortest path problem using Dijkstra's Algo
Find the cost of shortest path from 'A' to 'D'



Sol^{n}

V:	A	B	C	D	E
	0	∞	∞	∞	∞
A, C	-	10	3	∞	∞
A, C, E	-	7	-	11	5
A, C, E, B	-	7	-	11	-
A, C, E, B, D	-	-	-	9	-

Total cost to reach destinat^{n} from source
= 9

Sequence of shortest path from source to dest^{n}

A —3— C —4— B —2— D

3 + 4 + 2 = 9

NOTE : In exam they may ask the sequence of shortest path from source to Dest^{n}

BELLMAN FORD ALGO

- Finds all shortest-path lengths from a source $s \in V$ to all $v \in V$ or determines that a negative-weight cycle exists.

- Solves single source shortest path in more general case in which edge weights can be negative.

- Given a weighted, directed graph $G = (V, E)$ with source s and weight funct^n w.
  - The algo returns a boolean value indicating whether or not there is a negative-weight cycle that is reachable from the source.
  - If there is a cycle, the algo indicates that no solution exits.
  - If there is no such cycle, the algo produces shortest paths & their weights.
  - Algo returns True if and only if the graph contains no-negative-weight cycles that are reachable from the source.

- ALGO : {

  Study from Book
  COREMAN

  }

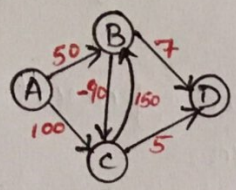
- Time Complexity

  $= O(V * V^2) = O(V^3)$ using Linked List

  $= O(V) * O(V-1) = O(V^2)$ using Arrays

Q1. Find the shortest path using Bellman-Ford Algo from 'A' to D

Sol^n



	A	B	C	D
	0	∞	∞	∞
i = 1	0	50	100	∞
i = 2	0	10	100	57
i = 3	0	10	100	17
i = 4	0	10	100	17

Distance given (i.e. shortest Path from Source)

A — B : 10

A — C : 100

A — D : 17

NOTE :

* Bellman For can find all -ve edge weight cycle present in graph if they are reachable from source

* Does not work for unweighted edge in graph