25-2-25 Tuesday

classmate
Date 25-2-25
Page 3

MODULE-5

→ Min of bin =

$$\text{ceil} - \left\lceil \frac{\text{Total weight}}{\text{No of Bins}} \right\rceil$$

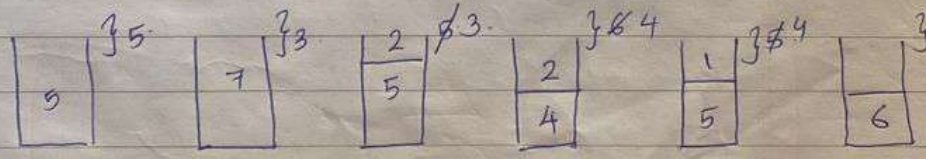
Q ~~Next fit~~

Q) Ne

→ NEXT FIT ALGORITHM:

Q Apply next fit bin packing approximation Algorithm on the following with bin capacity=10, Assuming sizes of items be {5, 7, 5, 2, 4, 2, 5, 1, 6}

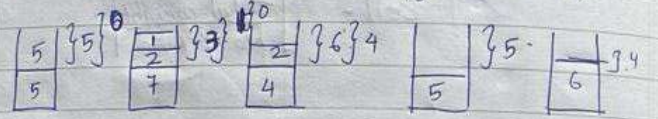
Min of bins = $\left\lceil \frac{37}{10} \right\rceil = 4$



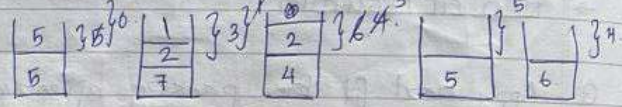
Q) Apply first fit bin packing approximation algorithm on the following with bin capacity=10, Assuming sizes of items be {5, 7, 5, 2, 4, 2, 5, 1, 6} ?

25-2-2025 Tuesday

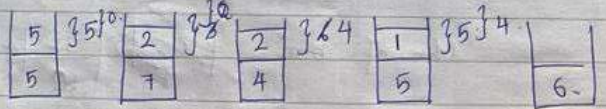
Min no of bins = $\lceil \frac{37}{10} \rceil = \lceil 3.7 \rceil = 4$



→ BEST FIT:



→ WORST

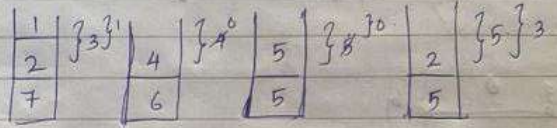


→ OFFLINE ALGORITHM:

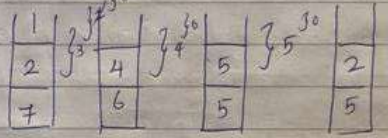
→ FIRST-FIT DECREASING ALGO:

{7, 6, 5, 5, 5, 4, 2, 2, 1}

→ first fit



→ Best fit:



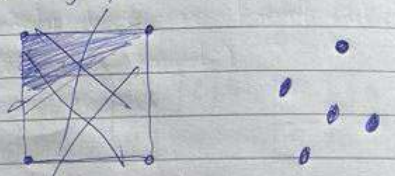
→ Graph colouring:

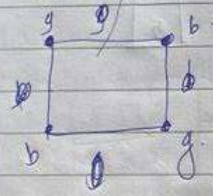
① Vertex colouring
② Edge colouring
③ Face colouring.

chromatic colour → min no of colour used (K, 1)

27-2-2025 Tuesday Thursday

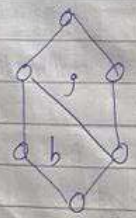
→ One-colourable graph:





Vertex coloured.

$\chi(G) = 2$.



Face-colouring.

28-02-2025 Thursday

→ Approximate-graph-colouring (G, n)

{

for i=1 to n do

for c=1 to n do

if no vertex adjacent to v_i has color c.

{ color v_i with c

Break

}.

}.

→ Randomized Algorithm:

1. Las Vegas Algorithm
2. Monte carlo Algo

① → Las Randomized Las Vegas Algo:

→ O/p is always correct & optimal.

→ Running time is a random number & also it is not bounded.

Eg. Randomized quick sort

Algorithm finding A_LV (A, n)

{

repeat

{
Randomly choose one element out of n elements
} until ('a' is found)

}

③→ Randomized Monte Carlo Algorithm:

→ The algorithm may produce correct o/p with some probability

→ A monte carlo Algorithm runs for a fixed number of steps i.e, the running time is deterministic

Algorithm finding A_MC(A, n, k)
{
i=0
repeat
{
Randomly select one element out of n elements
i=i+1
} until (i=k or 'a' is found)

}

→ Randomized QuickSort:

Algorithm randQuickSort(A[], low, high)

1. if low >= high, then EXIT
2. While pivot 'x' is not a Central Pivot

1. Choose uniformly at random a number from [low, high], let the randomly picked number be x.

2. Count elements in A[low, high] that are smaller than A[x]. Let this count be sc

3. Count elements in A[low, high] that are greater than A[x]. Let this count be gc

4. Let n = (high - low + 1) if sc >= n/4 and gc >= n/4 then x is a central pivot

3. Partition A[low, high] into two subarrays. The first subarray has all the elements of A that are less than x and the second subarray has all those that are greater than x. Now the index of x be pos.
4. randQuickSort(A, low, pos-1)
5. randQuickSort(A, pos+1, high)

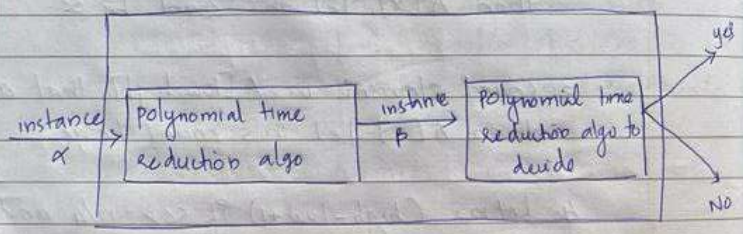
1/3/2025 Saturday

→ Polynomial time reduction:

Suppose we have a deasion problems 'A' & 'B'.

*) problem A: Like to solve in polynomial time is not exist
: Instance is alpha. (α)

*) problem B: Having a polynomial time algo ~~instance is~~ np.
: Instance is Beta (β)



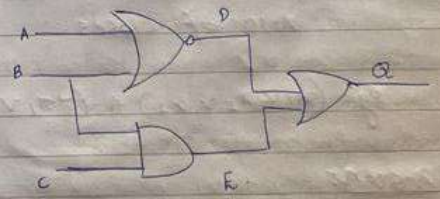
→ NP hard:
→ 'x' is a decision problem
→ 'x' is np hard if every problem in np is polynomial time reducable to 'x'.
→ 'x' is hard as all peoblems in np
→ If 'x' can be solved in polynomial time, then all peoblems np can also solved in polynomial time

1/3/2025 Saturday

→ NP complete:
→ If the problem is 'np' as well as 'np hard' then problem is NP complete.

Eg: → ① Circuit SAT problem

Note: Given a boolean circuit 'C' is there an assignment to the variables that causes the circuit to o/p 1.



Eg: ② → SAT problem:

Given boolean expression ϕ is there an assignment to the variables that causes the expression to o/p to 1

1/3/2025 Saturday

Eg ③ 3-CNF SAT Problem

① Literals: variables and its negations are called literals.

② close: 'OR' of one or more literals are called close.

eg: $x_1 \vee x_2 \vee \neg x_3$.

③ CNF: Conjuctive Normal form: 'AND' of closes

Eg: $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$

④ Arguments.

→ NP complete proof:

1. prove that given problem is np
2. → write a polynomial time verification algo.
   → prove that given problem is np hard
   → we write a polynomial time verification algo from any np problem to given problem.

1/3/25 Saturday

CKT-SAT
↓
SAT
↓
3-CNF SAT
↓ ↓
Clique HAM CYCLE
↓ ↓
Vertex cover TSP
↓
Subset sum

Q) Prove that clique problem is np complete

Step 1 Write polynomial time verification algo to prove that given problem is np hard

Input < G, k, V'>

1. test whether V' is a set of k vertices in the graph G
2. Check whether for each pair (u,v) ∈ V' the edge (u,v) ∈ E
3. Both pass then accept, otherwise reject
   → Algo will execute in poly time.
   ∴ Clique ∈ NP

1/3/2025 Saturday

Step2: Write poly time reduction algo from 3-CNF SAT problem to clique

CNF-SAT problem to CLIQUE problem:

→ Algorithm

1. Let $\phi = C_1 \wedge C_2 \wedge \dots \wedge C_k$ be a Boolean formula in 3CNF with k clauses. Each clauses C_r has exactly three distinct literals l_1^r, l_2^r, l_3^r.

2. Construct a graph G such that ϕ is satisfiable iff G has a click of size k, the graph G is constructed as follows
   1. for each clauses $C_r = (l_1^r \vee l_2^r \vee l_3^r)$ in ϕ, we place a triple of vertices V_1^r, V_2^r & V_3^r in to V.

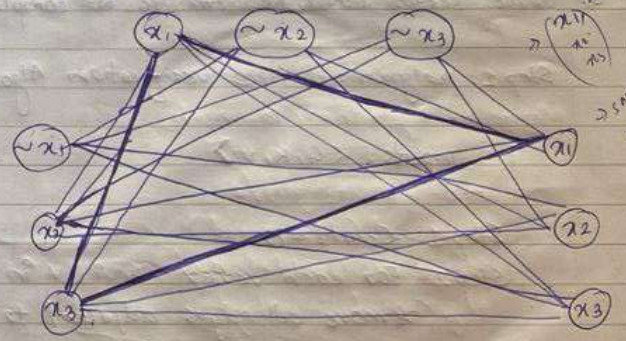
   2. Put an edge between V_i^r to V_j^s if following 2 conditions hold:
      1. V_i^r & V_j^s in different triples (that is $r \neq s$)
      2. l_i^r is not a negation of l_j^s.

1/3/2025 Saturday

3. If G has a clique of size k, then ϕ has a satisfying assignment.

$\phi = (x_1 \vee \sim x_2 \vee \sim x_3) \wedge (\sim x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$

1 0 0 0 1 1 1 0



$x_1 = 1 \quad x_2 = 0 \quad x_3 = 1$

$x_1 \vee \sim x_2 \vee \sim x_3 = 1$

$\sim x_1 \vee x_2 \vee x_3 = 1$

$x_1 \vee x_2 \vee x_3 = 1$

$1 \wedge 1 \wedge 1 = 1$ //

consider literals taken 1 common from 1 & 2 is x_1, x_3

and from 2 & 3 x_1, x_3 and x_3 & x_3.

1/3/25 Saturday

if 'G' has clique of size '3' and 5 has a satisfying assignment at (x_1, x_3, x_1)

Reduction from 3-CNF SAT to clique in polynomial time, ∴ clique belongs to NP hard ee, By step 1 and 2 clique belongs to NP complete.

Q) prove vertex cover problem is NP complete.

Step 1: polynomial time verification Algorithm

Inputs: $\langle G, K, V' \rangle$

Step 1: Count = 0

Step 2: For each vertex 'v' in 'V'
remove all edges adjacent to 'v' from 'E'
2.1: Increment count by 1.

Step 3: If count = k, and 'E' is empty then solution is correct

Step 4: Else solution is incorrect

Therefore vertex covers problems belongs to class NP.

4/3/2025 Tuesday

Step 2: polynomial time reduction Algorithm

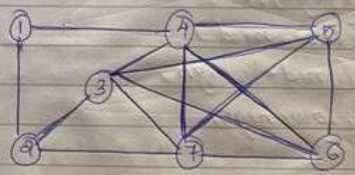
Inputs: $\langle G = (V, E), K \rangle$

Step1: Construct 'G'' which is compliment of G

Step2: If 'G'' has a vertex cover of size $|V| - k$ then 'G' has a clique of size 'k'

Therefore vertex cover belongs to ~~NP~~ class NP hard

$G = (V, E)$



clique = 3, 4, 5, 6, 7

$K = 5$

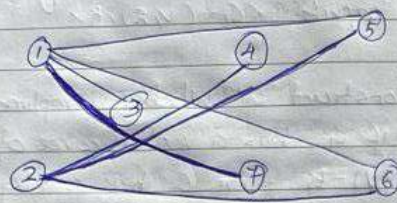
$|V| = 7$

$|V| - k = 7 - 5 = 2$

↳ vertex cover

4/3/2025
Tuesday

G' = (V, E')



(connection which is not in G)

Therefore vertex cover belongs to NP complete.

```
MergeSort Algorithm (low, high)
{
    mid = (low + high)/2
    Mergesort (low, mid)
    Mergesort (mid+1, high)
    Merg (mid, low, high)
}
Algo Merge (low, mid, high)
    i=low, x=low, y=mid+1;
    while (x≤mid and y≤high) do
    {
        if (a[x] ≤ a[y] then
        {
            b[i] = a[x];
            x = x+1
        }
        else
        {
            b[i] = a[y];
            x = x+1
        }
        i = i+1;
    }
    if (x≤mid) then
    {
        for k = x to mid do
        {
            b[i] = a[k]
            i = i+1
        }
    }
    else
        for k = y to high
    i=low, x=low, y=mid+1;
    while (x≤mid and y≤high) do
    {
        if (a[x] ≤ a[y]) then
        {
            b[i] = a[x];
            x = x+1;
        }
        else
        {
            b[i] = a[y];
            y = y+1;
        }
        i = i+1;
    }
    if (x≤mid) then
    {
        for (k = x to mid
        {
            b[i] = a[k]
```

i=low, x=low, y=mid+1
While (x≤mid) and (y≤high)
do
if (a[x] ≤ a[y] then
{
b[i] = a[x];
x = x+1
}
else
{
b[i] = a[y]
y = y+1
}
i = i+1
while if (x≤mid) then
{
for k = x to mid to
{b[i] = a[k]
x = i-1
}
}