ADDA notes

Algorithm and Function

An algorithm can be converted into a function.

Problem: Find smallest element in an array.

Algorithm: Step 1: Set min = A[0]
2. min Index = 0
3. for (i=1 to n-1)
   if (A[i] < A[minIndex]) Set min index = i
4. return minIndex

n-1 steps

2n+1 = n+2

$f(n) = n+2 \longrightarrow$ Linear function

- Quadratic
- cubic
- logarithmic

* **Complexity of Algorithm** is a function describing the efficiency of algorithm in terms of data the algorithm must process.

Problem → Algorithm (function) → Time → no. of machine instructions executed
→ Space → memory used by RAM

* **Time Complexity**: It is a function describing the amount of time an algorithm takes in terms of amount of input to the algorithm.

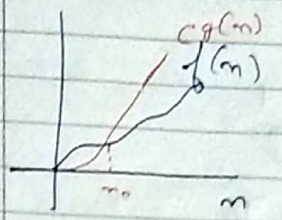
Asymptotic Notation

Formal way to speak about function

Page 2

Upper bound — Lower bound

O (Big-O)



$O(g(n)) = \{f(n) : 0 \le f(n) \le cg(n)\}$

where $c > 0$, $n_0 \ge 1$

eg $f(n) = n+2$

$n+2 \le 3n$ — $g(n)$

$c = 3$, $n \ge n_0$, $n \ge 1$

$O(n)$

eg $g(n) = n^2$

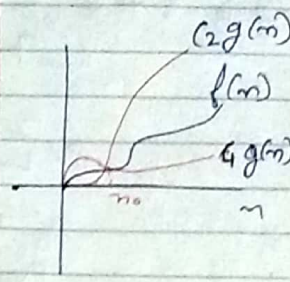
$= n^3$

it can be $O(n^2)$, $O(n^3)$

∴ but closest value is taken

→ It represent worst case because it is upper bound

Θ (Theta)



$\Theta(g(n)) = \{f(n) : c_1 g(n) \le f(n) \le c_2 g(n)\}$

$c_1, c_2 > 0$, $n \ge n_0$

eg $f(n) = n+2$

$c_1 = 1$, $c_2 = 3$

$1 \cdot n \le n+2 \le 3n$

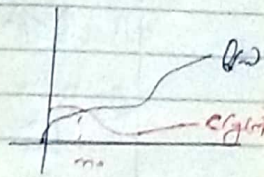
c_1 c_2

$n \ge 1$

$\Theta(n)$

→ It represent average case

Ω (Big-omega)



$\Omega(g(n)) = \{f(n) : 0 \le g(n) \le f(n)\}$

$c > 0$, $n \ge n_0$

eg

~~$\Omega(g(n)) = \{f(n) : 0 \le g(n) \le f(n)\}$~~

~~$c > 0$, $n \ge n_0$~~

$f(n) = n+2$

$1 \cdot n \le n+2$, $n \ge 1$

$\Omega(n)$

→ It represent best case

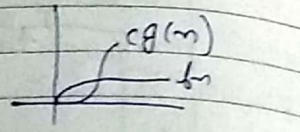
Big-oh notation

$o(g(n))$ is the set of all functions with a smaller or the same order of growth as $g(n)$

$O(n^2) = \{2n^2, 2n+1, \log n \ldots\}$

$2n^2 \simeq O(n^2)$

$2n \simeq o(n^2)$



Little-oh notation : little-oh of g of n is the set of

$o(g(n)) = \{f(n)$: for any positive constant $c>0$, $n_0 \geq 0$ such that $0 \leq f(n) < cg(n)$ $\forall$ $n \geq n_0\}$

or

$o(g(n))$ is the set of all functions with a smaller rate of growth than $g(n)$

$2n \simeq o(n^2)$ but $2n^2 \neq o(n^2)$

Definition

$f(n) = o\,g(n) \Rightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

$o(n^2) = \{100n+5, \log n \ldots\}$ ⇒ $f(n) = 2n \Rightarrow \lim_{n\to\infty} \frac{2n}{n^2} \Rightarrow 0$, $g(n) = n^2$

eg. $\lim_{n\to\infty} \frac{\log n}{n^2}$ (modified) $\Rightarrow \lim_{n\to\infty} \frac{f'(n)}{g'(n)}$ (for logs) $\Rightarrow \frac{1/n}{2n} = \frac{1}{2n^2} = 0$

$= \frac{\infty}{\infty}$

$=$ indefinite

formula

So far

$\Rightarrow T(n)$ is $O(f(n))$ (big) "basically means that $f(n)$ describes upperbound for $T(n)$

$\Rightarrow T(n)$ is $\Omega(f(n))$ "basically means that $f(n)$ describes lowerbound for $T(n)$

$\Rightarrow T(n)$ is $o(f(n))$ (small) "basically means that $f(n)$ is the upperbound for $T(n)$ but that $T(n)$ can never be equal

Little-omega: ω

$\omega(g(n)) = \{f(n)$: for any positive constant $c>0$, n_0 such that $0 \leq cg(n) < f(n)$ $\forall$ $n \geq n_0$

or

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$

eg. $\frac{n^2}{2} = \omega(n)$ $\frac{n^2}{2} \simeq \omega(n^2)$

eg. $f(n) = 3^n$ $g(n) = 2^n$

$\lim_{n\to\infty} \frac{3^n}{2^n} = \lim_{n\to\infty} \left(\frac{3}{2}\right)^n = \infty$

$3^n > \omega(2^n)$

eg: $f(n) = n^2$

$\omega(\log n) = ?$

$g(n) = \log n$

$\lim_{n\to\infty} \frac{n^2}{\log n} = \frac{\infty}{\infty} = \frac{0}{0}$ indefinite

Using L-hospital rule

$\lim_{n\to 0} \frac{f'(n)}{g'(n)} = \infty \Longrightarrow \lim_{n\to 1/n} \frac{2n}{1/n} = \infty$

$n^2 \simeq \omega(\log n)$

Recurrence Relation

A recurrence relation is an equation that recursively define a sequence where next term is a function of previous term.

Example ① $a_1 = a_0 + 2$
$a_2 = a_1 + 2$
$a_3 = a_2 + 2$
$\vdots$
$a_n = a_{n-1} + 2$

② Fibonacci series
$F_n = F_{n-1} + F_{n-2}$

```
int Fib(int n)
{
If (n == 0)
return 0;
else if (n == 1)
return 1
else
return (Fib(n-1)
+Fib(n-2));
}
```

Example $a_k = a_{k-1} + 2$ with $a_0 = 1$
$k = 1 \dots n$

$\Rightarrow a_1 = a_0 + 2 = 1 + 2$
$a_2 = a_1 + 2 = 1 + 2 + 2 = 1 + 2.2$
$a_3 = a_2 + 2 = 1 + 2 + 2 + 2 = 1 + 3.2$
$\dots$
$a_n = 1 + n.2 \Rightarrow a_n = 1 + 2n$

$O(n)$ (Big O complexity)

Brute Force method = Method of iteration for solving Recurrance relation

Master method to Solve recurrence relation:

$T(n) = a\,T(n/b) + f(n)$

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and $T(n)$ be defined non negative integers

$T(n)$ can be bounded as follows

Case 1: If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$ then $T(n) = \Theta(n^{\log_b a})$ $\Theta(f(n))$

Case 2: If $f(n) = \Theta(n^{\log_b a})$ then $T(n) = \Theta(n^{\log_b a} \log n)$ $f(n)\log n$

Case 3: If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$ and if $af(n/b) \leq cf(n)$ $[c < 1]$ then $T(n) = \Theta(f(n))$
regularity condition

Q:- $T(n) = 9T(n/3) + n$

$\Rightarrow T(n) = a \cdot T(n/b) + fn$
$a = 9, \; b = 3, \; f(n) = n$
$n^{\log_b a} = n^{\log_3 9} = n^{\log_3 3^2} = n^2$
$T(n) = \Theta(n^{\log_b a}) = \Theta(n^2)$

Q:- $T(n) = 2\,T(n/2) + n^3$
$\Rightarrow T(n) = a + (n/b) + f(n)$
$a = 2, \; b = 2, \; f(n) = n^3$
$n^{\log_b a} = n^{\log_2 2} = n$
$\Rightarrow$ ~~Regularity condition~~
$2f(n^3/2^3) < c\,f(n^3)$
$c = 1/4 \Rightarrow c > 1/4$

Case 3
$T(n) = \Theta(n^3)$
by master theorem

Imp rules for interation method

1) whenever converting $\doteq$ to $\leq$

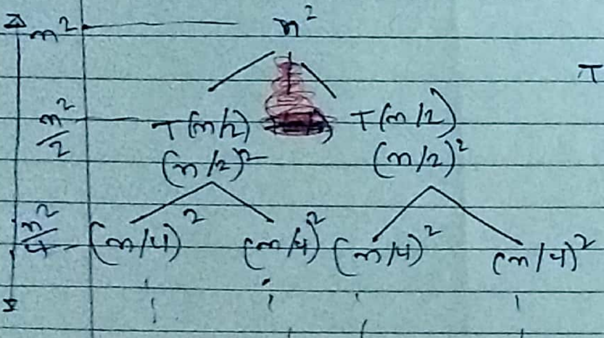
Change i to ∞ in Σ

2) Take $T(1)$ as θ at any pt.

3) for $n < 1$ take $o(n)$
for $n \leq 1$ take $O(n)$

Recursion Tree Method

Q. $T(n) = 8T(n/2) + n^2$ and $T(1) = 1$



$T(n/2) = 8\left(\frac{n}{2}\right)^2$
$= 2n^2$

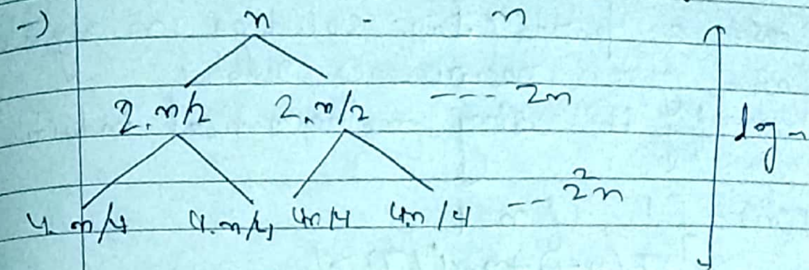
Useful formulae

1) $\sum_{i=1}^{n} \frac{1}{a^i}$ if common ratio is less than 1, then convert it in infinite GP series i.e. $\sum_{i=1}^{\infty} \frac{1}{a^i}$

2) $\sum_{i=0}^{\infty} \frac{1}{a^i}$ for infinite GP series, the formulae is $\frac{a}{1-r}$

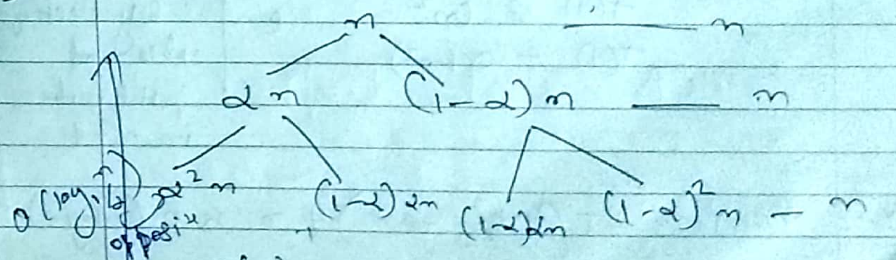
3) $\sum_{i=0}^{n} a^i$ for finite GP series, $\frac{a(r^n-1)}{r-1}$

Ex $T(n) = 4T(\lceil n/2 \rceil) + n$ using Recurrence Tree



$T(n) = n + 2n + 2^2n + \dots\ i^{th}$ $\qquad \frac{n}{2^i} = 1$
$= n(1 + 2 + 2^2 + \dots \log n)$ $\qquad \log n = i$
$= n \sum_{k=0}^{\log n} 2^k$
$= n.\left(\frac{2^{\log_2 n} - 1}{2-1}\right)$ $\quad \because \frac{a(r^n-1)}{r-1}$
$= n(n-1)$
$= n^2 - n$
$= \theta(n^2)$

Ex $T(n) = T(\alpha n) + T[(1-\alpha)n] + n$



$T(n) = n + n + \dots + (\log_{1/\alpha} n + 1)$ th
$= \sum_{i=0}^{\log_{1/\alpha} n} n$
$= \theta(n \log_{1/\alpha} n)$

Rules for intention method

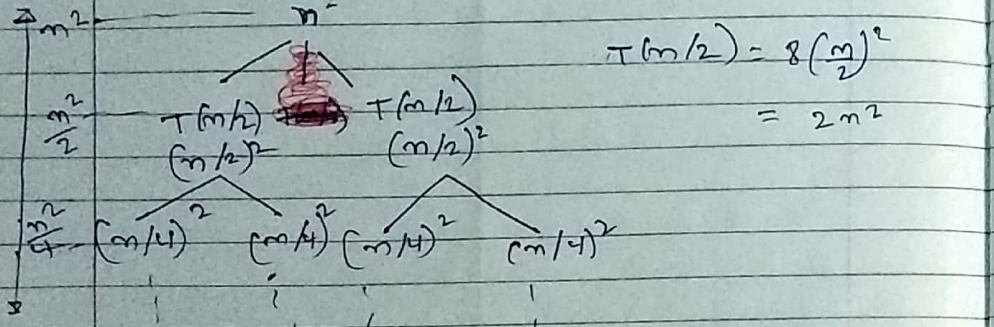
1) Whenever converting $\doteq$ to $\leq$

Change i to ∞ in Σ

2) Take $T(1)$ as 0 at any pt.
3) for $n < 1$ take $o(n)$
for $n \leq 1$ take $O(n)$

Recursion Tree Method

Q. $T(n) = 8T(n/2) + n^2$ and $T(1) = 1$



$T(n/2) = 8\left(\frac{n}{2}\right)^2$
$= 2n^2$

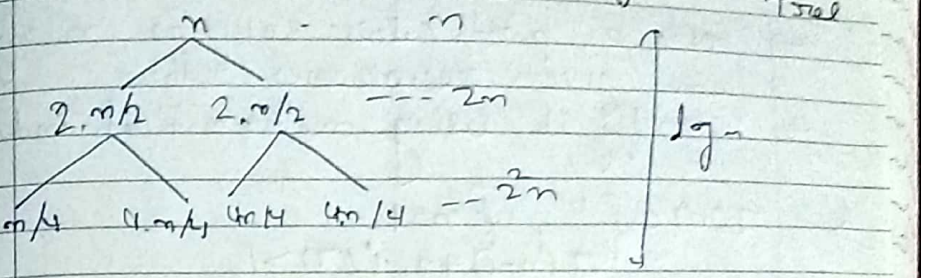
Useful formulae

1) $\sum_{i=1}^{n} \frac{1}{a^i}$ if common ratio is less than 1, then convert it in infinite GP series ie. $\sum_{i=1}^{\infty} \frac{1}{a^i}$

2) $\sum_{i=0}^{\infty} \frac{1}{a^i}$ for infinite GP series, the formulae is $\frac{a}{1-r}$

3) $\sum_{i=0}^{n} a_i$ for finite GP series, $\frac{a(r^n-1)}{r-1}$

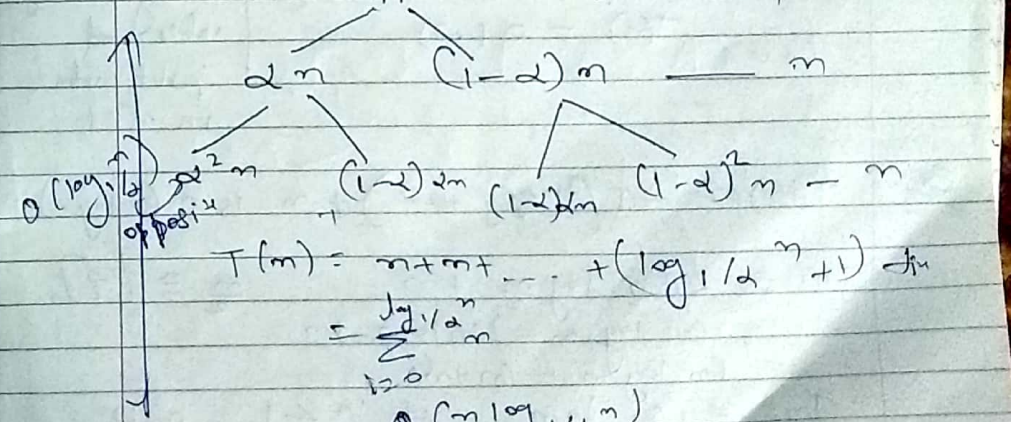
Ex $T(n) = 4T(\lfloor n/2 \rfloor) + n$ using Recursion Tree



$T(n) = n + 2n + 2^2 n + \ldots$ i^{th} $\quad \frac{n}{2^i} = 1$
$= n(1 + 2 + 2^2 + \ldots \log n) \quad \log n = i$
$= n \sum_{k=0}^{\log n} 2^k$
$= n\left(\frac{2^{\log_2 n} - 1}{2-1}\right) \quad \because \frac{a(r^n-1)}{r-1}$
$= n(n-1)$
$= n^2 - n$
$= O(n^2)$

$\log_2 n = \log$

Ex $T(n) = T(\alpha n) + T[(1-\alpha)n] + n$



$T(n) = n + n + \ldots + (\log_{1/\alpha} n + 1)$ times
$= \sum_{i=0}^{\log_{1/\alpha} n} n$
$= O(n \log_{1/\alpha} n)$

Substitution method

→ Guess a particular solution in asymptotic form given recurrence relation
→ verify it using mathematical induction

Q. $T(n) = \begin{cases} 1, & \text{if } n=1 \\ T(n-1)+n, & \text{if } n>1 \end{cases}$

$T(n) = T(n-1) + n$ and $T(1) = 1$ if $n \geq 1$

Solution $T(n) = O(n^2)$

$T(n) \leq cn^2$] Assumption
$c > 0, n_0 > n_0$

Proof:

$T(n) = T(n-1) + n$
$\leq c(n-1)^2 + n$
$\leq cn^2 - 2cn + c + n$
$T(n) \leq cn^2$ ← ∴ By checking values and putting value in $n=1$
$T(n) = O(n^2)$

Ex $T(n) = 2T\left(\left[\frac{n}{2}\right] + 22\right) + n$ is $O(n \log n)$

$T(n) \leq 2c\frac{n}{2}\log\frac{n}{2} + n \qquad \frac{n}{2} = \left[\frac{n}{2}\right]$
$= cn \log n - cn \log 2 + n$
$= cn \log n - cn + n$
$Tn \leq cn \log n$, if $c \leq 1$ Doubt
$Tn = O(n \log n)$ is true

Or by Master method

How to make good guess
→ For given recurrence relation

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

Case 1: If $a = b = 2$, then
$T(n) = O(f(n) \log n)$

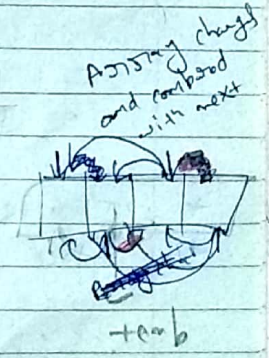
Case 2: If $a = 1$, b = any value, then
$T(n) = O(\log n)$

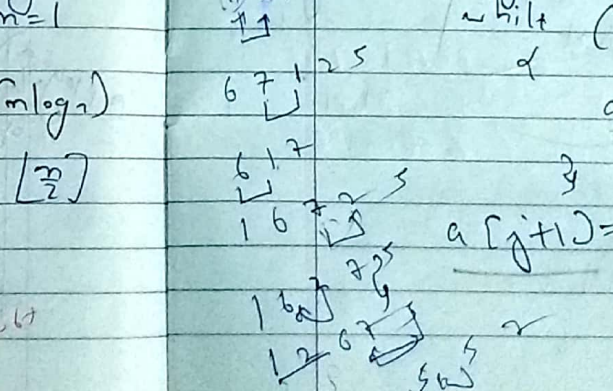
Case 3: if $a \neq 1$, $b > a$
$T(n) = O[f(n) \cdot n]$
$a \neq 1 \quad a > b$

Insertion Sort

Algorithm

```
Insertion-Sort(a)
{ int k, j, n;
  for (k=1; k<=n; k++)
  {
    temp = a[k];
    j = k-1;
    while ((temp < a[j]) && (j >= 0))
    {
      a[j+1] = a[j];
      j = j-1;
    }
    a[j+1] = temp;
```





No. of steps

For worst case

k=1 ⇒ 1+1 = 2×1
k=2 ⇒ 2+2 = 2×2
k=3 ⇒ 3+3 = 2.3

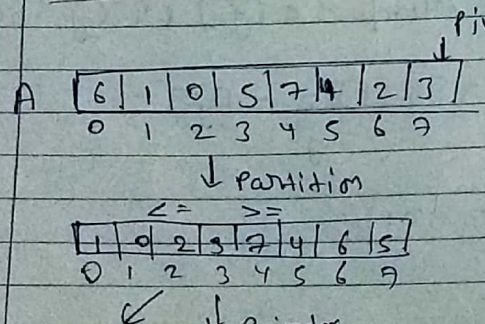
= 2.1 + 2.2 ---- 2.(n-1)
= 2(1+2+ --- n-1)
= 2 (n-1)n / 2 = n²-n
≃ O(n²)

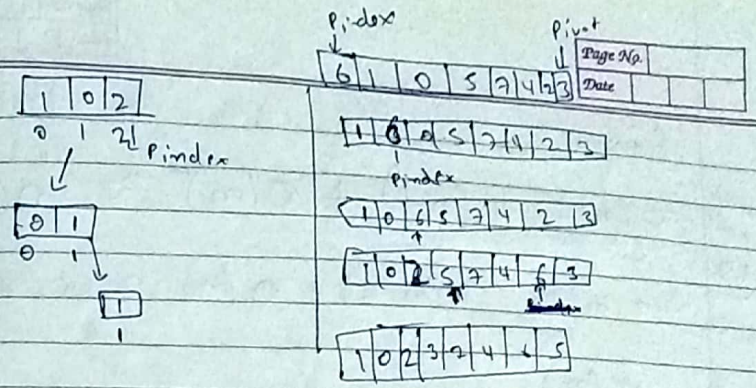
For Best case

6 7 8 9
= 1 + 1 + 1 ---- n-1
= Ω(n)

* It is inplace sorting
* Space complexity O(1)

Quick Sort [inplace sorting]





Algorithm

```
Quick Sort (A, start, end)
{
  if (start < end)
Pindex ← Partition (A, start, end)
    Quick Sort (A, start, Pindex-1)

    Quick Sort (A, Pindex+1, end)
}
}
```

```
Partition (A, start, end)
{
      Pivot ← A [end]
      Pindex ← start
  for (i = start to end-1)
  {   if (A[i] <= Pivot)
      {
          Swap (A[i], A[Pindex])
          Pindex = Pindex+1
      }
  }
Swap (A[Pindex], A[end])
```

Analysis

worst case: $T(n) = T(n-1) + O(n) \approx O(n^2)$

1 2 3 4 5 6

6 5 4 3 2 1: $T(n) = T(n-1) + O(n) \approx O(n^2)$

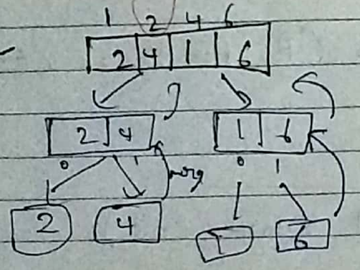
Best: T(n) → T(n/2), T(n/2)

$T(n) = 2T(n/2) + O(n) \approx = \Omega(n \log n)$

Merge Sort

Break (in equally) + merge

Example



Doubt

```
MergeSort (A)
{
  n = length (A)
  if (n < 2) return
  mid = n/2
  left = array of size (mid)
  right = arr of size (n-mid)
  for (i = 0 to mid-1)
     left [i] = A [i]
  for (i = mid to n-1)
     right [i-mid] = A [i]
  MergeSort (left)      T(n/2)
  MergeSort (right)     T(n/2)
  merge (left, right, A)
}

Merge (L, R, A)
{
  nL = Length (L)
  nR = Length (R)
  i = 0, j = 0, k = 0
  while (i < nL && j < nR)
  {
    if (L[i] ≤ R[j])
    {
      A[k] = L[i]
      i = i+1
    }
    else
    {
      A[k] = R[j]
      j = j+1
    }
    k = k+1
  }
  while (i < nL)
  {
    A[k] = L[i]; i++, k++;
  }
  while (j < nR)
  {
    A[k] = R[j], j++, k++;
  }
}
```

C1 (the steps before the copy loops); C2n (the copy loops); recursive call (the MergeSort calls); C3n + C4 extra cost (merge)

? why (beside the while (i < nL) loop)

→ Div → O(1)
→ Conquer → 2T(n/2)
→ O(n)

Analysis

n → n/2, n/2

$T(n) = C_1 + C_2 n + 2T(n/2) + C_3 n + C_4$

~~= O(n)~~ [the $C_1 + C_2 n + C_3 n + C_4$ part is O(n)]

$= 2T(n/2) + O(n)$

$= O(n \log n)$ ← Time complexity

* It is not inplace sort, so space required

n → n/2, n/2 → n/4, n/4 → … → 1

$\frac{n}{2^k} = 1$

$k = \log_2 n$

Space required ↑

$n + \frac{n}{2} + \frac{n}{4} - -$

$= n(1 + \frac{1}{2} + \frac{1}{4} [illegible])$

$= 2n$

$= O(n)$ ← Space complexity

$1 \to n \times \log n = O(n \log n)$

Strassen's Algorithm for matrix multiplication

Matrix multiplication

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix}_{n \times n} \quad Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix} \quad Z = \begin{bmatrix} z^{11} & z^{12} \\ z^{21} & z^{22} \end{bmatrix}$$

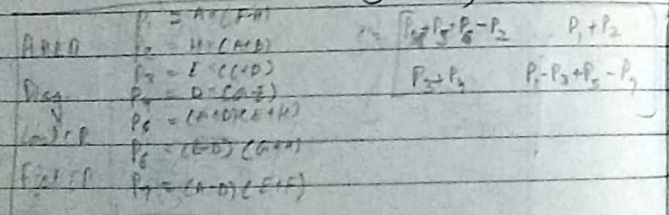
$$z^{11} = AE + B.G$$
$$= x_1 + x_2$$

```
Mmult (X, Y, n)
1.  If n=1  o/p X*Y
2.  Else compute A,B ... H %  by n/2
3.
4.  X1 = Mmult (A,E , n/2)
5.  X2 = Mmult (B,G , n/2)
6.  X3 = Mmult (A,F , n/2)
7.  X4 = Mmult (B,H , n/2)
8.  X5 = Mmult (C,E , n/2)
9.  X6 = Mmult (D,G , n/2)
10. X7 = Mmult (C,F , n/2)
11. X8 = Mmult (D, H, n/2)
12. z11 = X1 + X2
13. z12 = X3 + X4
14. z21 = X5 + X6
15. z22 = X7 + X8
16. output Z
17. End if
```

8 + (n/2)

$\frac{n}{2} \times \frac{n}{2} = \frac{n^2}{4} = \Theta(n^2)$

$T(n) = 8T(n/2) + \Theta(n^2)$
$= \Theta(n^3)$



$Z = \begin{bmatrix} P_5+P_4-P_2+P_6 & P_1+P_2 \\ P_3+P_4 & P_1+P_5-P_3-P_7 \end{bmatrix}$

Strassen (X, Y) Implement in order method to use calculator

```
1.  If n=1 , output X * Y
2.  else
3.  Compute A,B,C ... H% by n/2
4.  P1 = Strassen (A, F-H)
5.  P2 = Strassen (H, A+B)
6.  P3 = Strassen (E, C+D)
7.  P4 = Strassen (D, G-E)
8.  P5 = Strassen (A+D, E+F)
9.  P6 = Strassen (B-D, G+H)
10. P7 = Strassen (A-C, E+F)
11. z11 = P6+P5+P4-P2
12. z12 = P1+P2
13. z21 = P3+P4
14. z22 = P1+P5-P3-P7
15. output Z
```

7 T (n/2)

$P_1 = A * (E-H)$

$\frac{n}{2} * \frac{n}{2} = \frac{n^2}{4} = \Theta(n^2)$

$T(n) = 7T(n/2) + \Theta(n^2)$
$= O(n^{2.32}) = O(n^2)$

Example

$$X = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \quad Y = \begin{pmatrix} E & F \\ G & H \end{pmatrix} \quad Z = X.Y = \begin{pmatrix} P_5+P_4+P_6-P_2 & P_1+P_2 \\ P_3+P_4 & P_5+P_1-P_3-P_7 \end{pmatrix}$$

Row (1) Colm (-)

Memory → A H E D
Diagonals
Last CR
First CR

$P_1 = A * (F-H)$
$P_2 = H * (A+B)$
$P_3 = E * (C+D)$
$P_4 = D * (G-E)$
$P_5 = (A+D) * (E+H)$
$P_6 = (B-D) * (G+H)$
$P_7 = (A-C) * (E+F)$

$P_1 = A * (F-H)$
$P_2 = H * (A+B)$
$P_3 = E * (C+D)$
$P_4 = D * (G-E)$
$P_5 = (A+D) * (E+H)$
$P_6 = (B-D) * (G+H)$
$P_7 = (A-C) * (E+F)$

Page No.
Date

Ex

$$X = \begin{pmatrix} 2 & 4 \\ 3 & 5 \end{pmatrix} \quad Y = \begin{pmatrix} 1 & 3 \\ 4 & 7 \end{pmatrix} \quad Z = X \cdot Y$$

$P_1 = -8$, $P_2 = 42$, $P_3 = 8$, $P_4 = 15$,
$P_5 = 56$, $P_6 = -11$, $P_7 = -4$

$$Z = \begin{pmatrix} 18 & 34 \\ 23 & 44 \end{pmatrix}$$

(Union find DS or Merge find set DS)

Dis-joint Set Data Structure

* Disjoint set DS representing a dynamic collection of set S

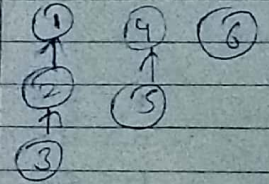
$S = \{ S_1, S_2 \cdots S_n \}$

$\Rightarrow$ $S = \{ a, b, c, d, e \}$ (S_1: a, b, c; S_2: d, e)

1. Make-Set(a) →
2. Find-Set(a) →
3. Union(y, n) →

$SF = \{ S_1, S_2 \}$ — Set related to each other

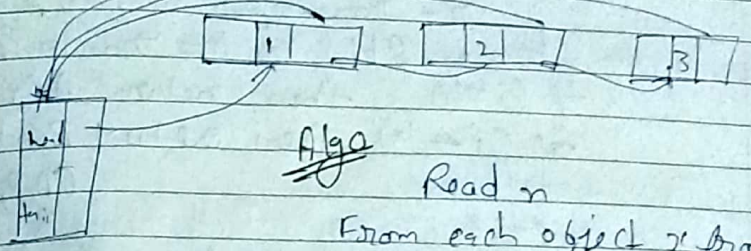
1 2 3 4 5 6



$\Rightarrow$ $S = \{ S_1 = \{1, 2, 3, 6\}, S_2 = \{4, 5\}, S_3 = \{7\} \}$

Page No.
Date

Link list

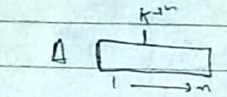


Algo

Read n
From each object x from 1 to n
Make_set(x)
Do untill all related objects are in first
if (x ≈ y) i.e related
if (Find-Set(x) != Find-Set(y))
union(x, y)

To check both set not have same ele

Median and Order Statistics



i^{th} order statistics of a set of n elements is the i^{th} smallest element

$i = 1$ minimum
$i = n$ Maximum
median = $\lfloor \frac{n+1}{2} \rfloor$ or $\lceil \frac{n+1}{2} \rceil$ — half way

lower median — upper median

Finding o.s of i^{th} element → 1) Randomise - Algorithm
2) Worst case linear time

Randomized Algo

Randomized-Select(A, p, q, i) Array Starting Ending looking for

if (p=q) then return A[p] //
r = Randompartition (A, p, q)
k = r - p + 1 // the rank A[r]
if (i=k) then return A[r]
if (i < k) then return Randomize-select (A, P, r-1, i)
else
return Randomized-select (A, r+1, q, i-k)

⟹ | | | | | | (p ← r → q)

i = 2

| 1 | 4 | 2 | 3 |
1 2 3 4

$O(n \log n)$

Q. 7^{th} order statistics i=7

A | 6 | 10 | 13 | 5 | 8 | 3 | 2 | 11 |
(pivot at 6)

| 2 | 5 | 3 | 6 | 8 | 13 | 10 | 11 |
(r at 6 = 4, start at 8)

| 8 | 10 | 11 | 13 |
i = 7

Overlapping Problem
→ If problem can be broken down to sub problem
→ Same subproblem solve many time ... several time
→ Explain eg. Fibonacci Series

Dynamic Programming

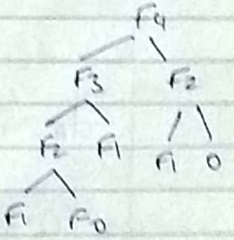
* It solves problems by combining the solutions to subprograms

* It is applied when sub-program overlap
→ Dynamic Programming algorithm solves each subproblem just once and then saves its answer in table.

4 steps for Dynamic Programming

1. Characterize the structure of a optimal sol^n
2. Recursively define the value of an optimal solution
3. Compute the value of an optimal solution (in bottom up fashion)
4. Construct optimal solution from computed information.

Fibonacci Series
0 1 1 2 3 ...
$F_n = F_{n-1} + F_{n-2}$
$F_0 = 0, F_1 = 1$



$O(2^n)$

Solving

DP → m | 0 | 1 | | | | m[0]=1, m[1]=1
0 1 2 3 4

Dictionary m;
m[0]=0, m[1]=1
int fib(n)
if m[n] = null
m[n] = fib[n-1] + fib[n-2]
return m[n]

O(n) — optimal sol change due to dynamic programming

⟹ Solution of optimal sub structure

Page No.
Date

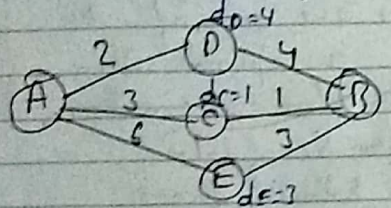
Memorization & optimal Solution

Dynamic Programming :- 1. recursive substance

2. Memorization
"keep track of intermediate result to solve intermediate problem"

optimal Structure
A problem has optimal structure if an optimal solution can be constructed efficiently from optimal solutions of its sub problems.

Ex



$d_A = \min\{d_B, d_C, d_E\} = 4$

$d(x) = \min\{d_{yxw} \dots\}$

Proof by contradiction

Let d_{A-B} be optimal path from A-B
Let assume it passes th. C

d_{A-C} & d_{CB}

assume there is another path $d'_{AC} \leq d_{AC}$
d'_{A-C} & d_{CB}
then $d'_{AC} + d_{CB} < d_{AC} + d_{CB}$
$d'_{AC} + d_{CB} \not< d_{AC} + d_{CB}$
Prove, by contradiction d_{A-B} via C is optimal solution

Matrix Chain Multiplication

→ Decide optimal structure to minimize multiplication cost

$$A_1 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}_{3\times4} \times \begin{bmatrix} 2 & 2 \\ 2 & 2 \\ 2 & 2 \\ 2 & 2 \end{bmatrix}_{4\times2} = \begin{bmatrix} 8 & \\ & \end{bmatrix}_{3\times2}$$

$1-4$
$6-4\times6=24$ multiplications

$[A_1]_{2\times6} * [A_2]_{6\times1} * [A_3]_{4\times1}$

$A_1 \times A_2 \times A_3$

Case 1 $2\times4\times6 + 2\times1\times4 = 48+8$
$= 56$

Case 2 $A_1 * A_2 \times A_3$

$6\times4\times1 + 6\times2\times1 = 36$

Case 3

$A_1 \times A_2 \times A_3$

$A_1 \quad k=2\times3$

Let's take this matrix

	1	2	3
1	0	48	36
2		0	24
3			0

Ind

$A_i \dots A_k \; A_{k+1} \dots A_j$

$A[1,n] \rightarrow A[1,2] + A[3,n] + r_1 c_2 c_n \dots$ Case 2
$A[1,1] + A[2,n] + r_1 c_1 c_n$

$A[1,m-1] + A[n,1] + r_1 c_{n-1} c_n$

$$A[i,j] = \min_{k=i+1}^{j-1} \{A[i,k] + A[k+1,j] + r_i r_k c_j\}$$

15, 20, 21, 77
15, 17, 21, 20

Page No.
Date

Longest Common Sub Sequence [LCS]

Use of LCS :→ we can express a strand of DNA as a string over finite set {A, C, G, T}

A = abcdcdab
B = dcbcaa

LCS
→ bca
→ dcb
→ dca

$$Z_{i,j} = \begin{cases} 0, & i=0 \text{ or } j=0 \\ Z_{i-1,j-1}+1, & a_i = b_j,\ i,j>0 \\ \max(Z_{i,j-1}, Z_{i-1,j}), & i,j>0,\ a_i \neq b_j \end{cases}$$

Z

		0	1 a	2 b	3 c	4 d	5 c	6 d	7 a	8 b
0		0	0	0	0	0	0	0	0	0
1	d	0	0	0	0	1	1	1	1	1
2	c	0	0	0	1	1	2	2	2	2
3	b	0	0	1	1	1	2	2	2	2
4	c	0	0	1	2	2	2	2	2	3
5	a	0	1	1	2	2	2	2	3	3
6	a	0	1	1	2	2	2	2	3	3

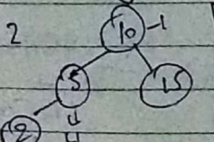
b c a

L.C.S = bca

Optimal BST (Binary Search Tree)

→ Arrange numbers in BST that minimize accessing cost

10, 5, 15, 2



1×1 + 2×4 = 9

4×1 + 1×2 = 6

left ← root → right

A	0	1	2	3
	5	6	8	20
	4	4	4	4
	4	2	6	3

→ 5 6
(4) (2)
4+2=6 + min{2→0, 4→1}

→ 5 6 8 = 8
4 2 6 = 12 + min{...} = 20

	0	1	2	3
0	4	8	(20)[2]	(21)[2]
1		2	10	16
2			6	(2)[2]
3				3

6 8 20 → 11 + min{12 →1, 2+3 →2} = 16[2]

5 6 8 20 = 15 + {16→0, 4+12→1, 3+5+2, 20→3}

Page No. 26

0/1 Knapsack

Problem : thief →

item	1	2	3	4	
w	2	3	4	5	- kg
v	3	7	2	9	- $

Bag - 5 kg

$$C[i][M] = \begin{cases} 0 & \text{if } i=0,\ M=0 \\ C[i-1][M] & \text{if } w_i > M \\ \max(V_i + C[i-1][M-w_i],\ C[i-1][M]) & \text{if } i>0 \text{ and } w_i \le M \end{cases}$$

C[i][M] → value of item for 1 to i with max weight - M

item	w	v
1	2	3
2	3	7
3	4	2
4	5	9

i \ weight	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	7	7	10
3	0	0	3	7	7	10
4	0	0	3	7	7	10
	0					

cost

C[2][3] C[2][5] = max{ 7 + C[1][2], C[1][5] }

7 + 3 + 3 =

Page No.
Date

Binomial Co-efficient through Dynamic Programming

$$^nC_k = \frac{\lfloor n}{\lfloor k \lfloor n-k} \quad \& \quad ^nC_0 = {}^nC_n = 1$$

$$^5C_2 = \frac{5!}{2!\,3!} = 10$$

$$^nC_k = {}^{n-1}C_{k-1} + {}^{n-1}C_k$$

$$^5C_2 = {}^4C_1 + {}^4C_2 = \frac{4!}{1!\,3!} + \frac{4!}{2!\,2!}$$
$$= 4 + 6 = 10$$

Dynamic Programming

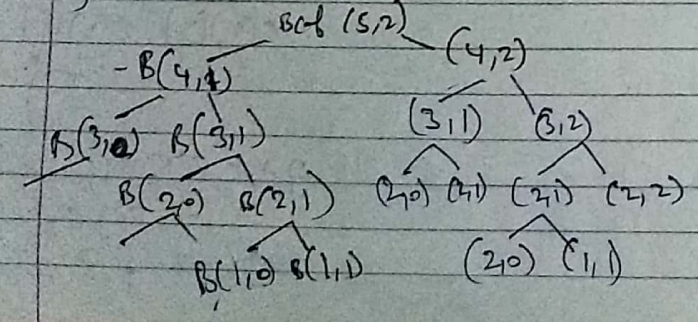
$$^2C_1 = {}^1C_0 + {}^1C_1$$

$$^nC_k = {}^{n-1}C_{k-1} + {}^{n-1}C_k$$

$$^4C_2 = {}^3C_1 + {}^3C_2$$

n ↓ / k →	0	1	2	3	4	5
0	1					
1	1	1				
2	1	2	1			
3	1	3	3	1		
4	1	4	6	4	1	
5	1	5	10	10	5	1

```
B(int n, int k)
{ if (k==0 || k==n) return 1
  else B(n-1,k-1) + B(n-1,k);
}
```



Page No.
Date

FLOYD - WARSHALL ALGORITHM

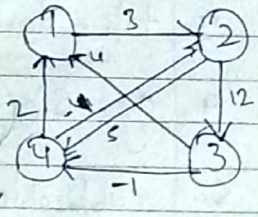
"shortest path between all pairs of vertices"
[Negative edge are allowed]

d_{ij} = weight of shortest path from vertex i to j for which all intermediate node are in {1, 2 ... k} (vertices)

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k=0 \\ \min\left(d_{ij}^{k-1}, d_{ik}^{(k-1)} + d_{kj}^{k-1}\right) & \text{if } k \geq 1 \end{cases}$$

for VxV matrix $D^{(n)} = d_{ij}^{n}$
4x4

$$\text{and } d_{ij} = \begin{cases} 0 & \text{if } i=j \\ w_{ij} & \text{if } i \neq j \\ \infty & \text{otherwise} \end{cases}$$



FLOYD - WARSHALL

Algorithm : running time $O(V^3)$

1. n = rows[W]
2. $D^0 \leftarrow W$
3. for k ← 1 to n
4. do for i ← 1 to n
5. do for j ← 1 to n
6. do $d_{ij} \leftarrow \min\left(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1}\right)$
7. return D^n

$D^0 = W$

	1	2	3	4
1	0	3	∞	∞
2	∞	0	12	5
3	4	∞	0	-1
4	2	-4	∞	0

D^1

	1	2	3	4
1	0	3	∞	∞
2	∞	0	12	5
3	4	7	0	-1
4	2	-4	∞	0

D^2

	1	2	3	4
1	0	3	15	8
2	∞	0	12	5
3	4	7	0	-1
4	2	-4	8	0

D^3

	1	2	3	4
1	0	3	15	8
2	7	0	12	5
3	3	-5	0	-1
4	2	-1	8	0

D^4

	1	2	3	4
1	0	3	15	8
2	7	0	12	5
3	3	-5	0	-1
4	2	-1	8	0

$d_{12} = d_{12} : d_{12}, d_{11} + d_{12}$ (k=3) ... at k=4

Difference b/w Dynamic Programming and Divide & conquer approach

Divide & Conquer Approach	Dynamic Programming
1. Sub-problems are more or less independent	1. Sub-problem are dependent causl overlapping
2. It may not provide an optimal solution	2. It grantees provide an optimal solution
3. top-down approach	3. bottom-top approach
4. eg. Binary Search	4. eg. Floyd-warshall
5. subproblems are recursively solved again and again increasing calculation and time.	5. Subproblems are shared, solution is stored and used for further problem

Operations performed on linked list DS for Disjoint sets

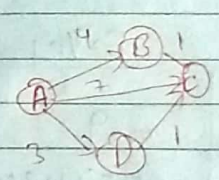
1. Intersection
2. Union of two sets
3. Delete an element from set - depends on implementation
4. Enumerate set i.e. iterate each element in set

Greedy Algorithm

* A greedy algorithm makes the choice that looks best at that moment.

Greedy Algorithm works on following property

1. Greedy choice property:- It makes locally optimal choice in the hope that this choice lead to globally optimal solution.
2. Optimal structure:- optimal solution contain optimal sub solution.



Ex-1

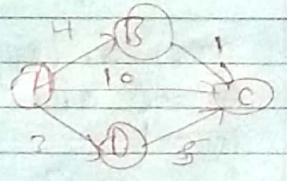
"A to C"

Acc. to greedy

A→B 4

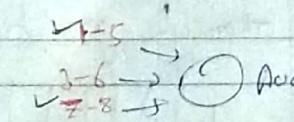
A→D 3 ✓

A→C 9



Ex-2

"A to C"



Activity Selection Problem

* It is the problem of scheduling several activities that require exclusive use of common resource, with a goal of selecting a maximum-size set of mutually compatible devices.

Ex Set of activity = {a1, a2 --- an}

start time = {s1, s2 --- sn}

Finish time = {f1, f2 --- fn}

ai	1	2	3	4	5	6	7	8	9
si	1	2	4	1	5	8	9	11	13
fi	3	5	7	8	9	10	11	14	16

Auditorium

"Select longest mutally compatible"

Greedy - Activity - selector (s, k)

1. n = s.length
2. A = {a1}
3. k = 1
4. for m = 2 to n
5. if s[m] ≥ f[k]
6. A = A ∪ {am}
7. k = m
8. return A

Solving above problem

n = 9

A = {a1} activity

k = 1

loop m = 2

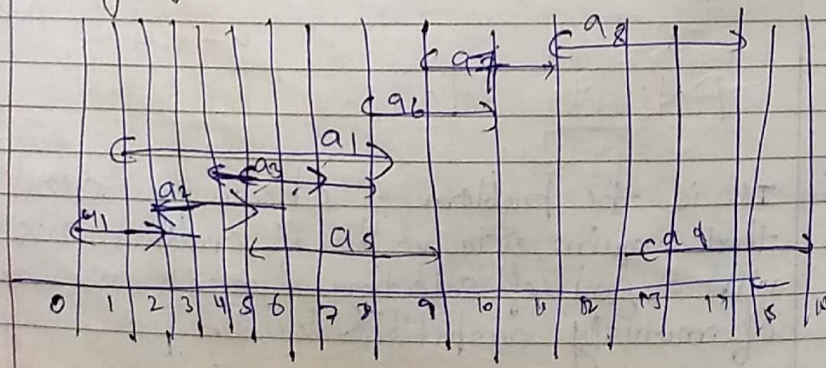
if s(m) > f(k)

s(2) ≥ f(1)

2 ≥ 3

A = {a1, a3, a6, a8}

Through graph



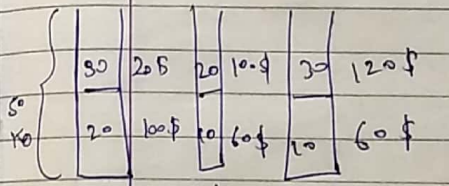
Fractional knapsack Problem

* The thief can take fraction of items, rather than having to make binary (0-1) choice for each item.

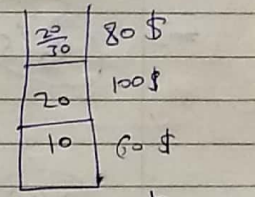
Ex item 1 (10 kg, $60), item 2 (20 kg, $100), item 3 (30 kg, $120)

knapsack(M) = 50

For 0/1



For fractional



Huffman codes

→ Data compression method

→ Lossless (without loss of information) compression

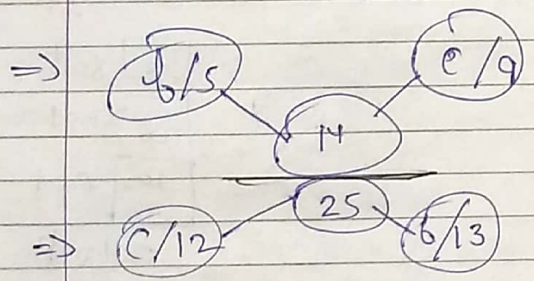
Algorithm C Huffman (C)

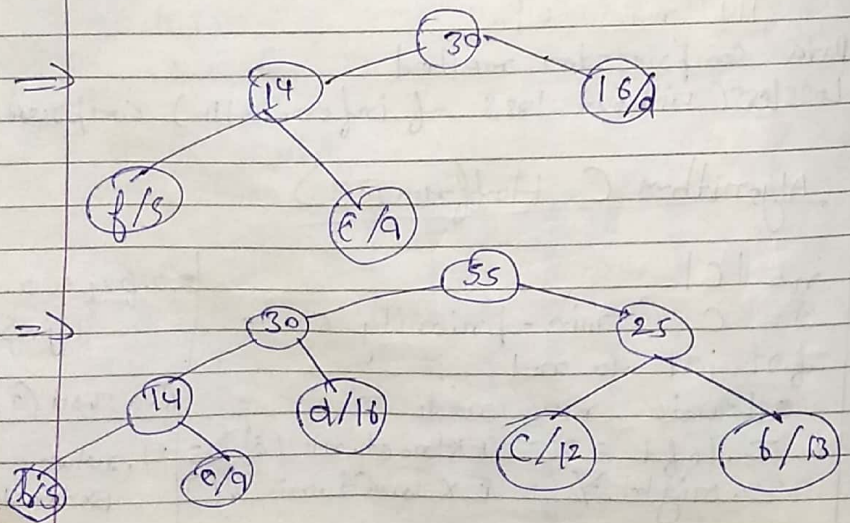
1. n = |C| - 6
2. Q = C // min-priority Queue
3. for i = 1 to n-1
   allocate new node Z
   Z.left = x = Extract.min(Q)
   Z.right = y = Extract.min(Q)
4. z.freq = x.freq + y.freq
5. Insert (Q, Z)
6. return Extract.min(Q)

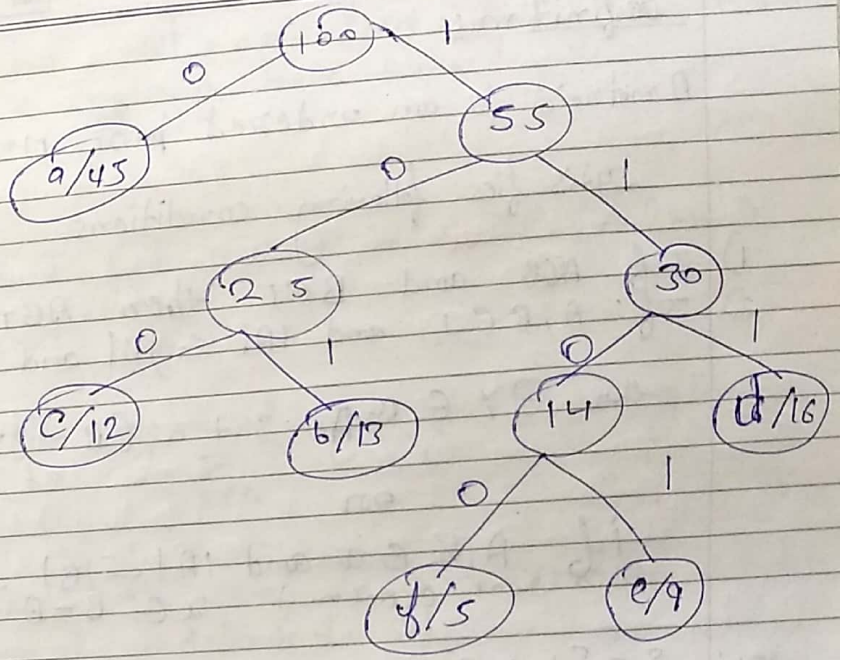
character	Frequency	Fixed length code	Variable length
a	45	000	0
b	13	001	101
c	12	010	100
d	16	011	111
e	9	100	1101
f	5	101	1100

25
55
30
14

Bitsreq. to store a character =
frequency x fixed length code



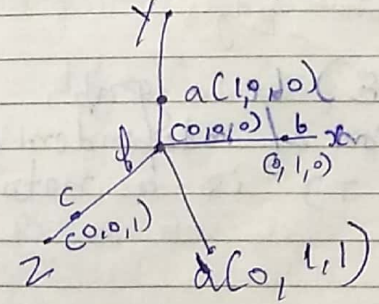




Matroid

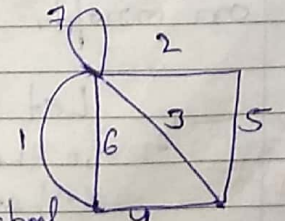
→ It involves combinational structure
→ It provides abstraction for many problems like linear algebra, Graph theory.

Linear Algebra



$\vec{a}, \vec{b}, \vec{c}, \vec{d}$ → Linearly Independent Scalar Multiplication

Graph



cycle should not be there
Linearly independent - No cycle
I = {1, 3, 4, 5}
Not L.I

Definition:

A matroid is an ordered pair $M = [S, I]$ (S: finite set; I: subsets of S) satisfies following conditions

1) If $A \subseteq B$ and $B \in I$ then $A \in I$
2) If $A, B \in I$ and $|A| < |B|$ and $x \in A - B$ then $\exists\, y \in B - A$ s.t $(B - \{y\}) \cup \{x\} \in I$

or

if $A, B \in I$ and $|A| < |B|$ then exist some element $x \in B - A$ s.t $A \cup \{x\} \in I$

Eg: $S = \{1, 2, 3\}$

$I = \{(1), (2), (3), (1,2), (2,3), (3,1)\}$

A (A_1, A_2) above; B (B_1, B_2)

— Both conditions verified, so it is a matroid

Matroid → Greedy

on Graph

Let $G = (V, E)$ be a graph (V: vertices, E: edges)

I be the collection of independent set of edges (i.e. no cycle created)

then (S_G, I) is a matroid

eg: A, B, C

Theorem:- All maximal independent subsets in a a matroid have same length.

Weighted Matroid:

(S, I, w) where $w: S \to R$ [Real no.]

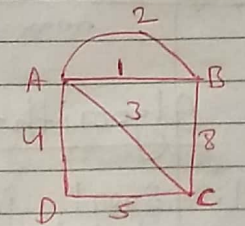
$\in$ weighted L. Ind. fun.

The weight of any $A \in I$ is given by $\sum_{x \in A} w(x)$

Problem: To compute a maximum weight maximal independent set.

Algorithm [weighted Matroid (S, I, w)]

1. Sort the elements S in non-increasing order. Corder x_1, x_2 ...
2. $A = \phi$
3. For $i = 1$ to $|S|$ do
   if $(A \cup \{x_i\} \in I)$
   then $A = A \cup \{x_i\}$
4. Return A



$S = \{AB, AD, BC, CA, CD\}$

8, 5, 4, 3, 2, 1

x_1 x_2

$= \{8\}$

$= \{8, 5\}$

$= \{8, 5, 4\}$ maximal

$= \{8, 5, 4, 3\}$ cycle ✗

BC, DC, AD

L.

Page No.
Date

Task - Scheduling Problem

→ It is problem of optimally scheduling unit time task on single processor

→ A unit time task is a job, such as program to be run on computer that requires exactly one unit of time to complete.

Ex

Index	1	2	3	4	5
Job	J_1	J_2	J_3	J_4	J_5
Deadline	2	1	3	2	1
Profit	30	50	10	20	10

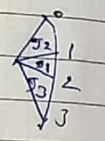
Index	1	2	3	4	5
Job	J_2	J_1	J_4	J_3	J_5
Deadline	1	2	2	3	1
Profit	50	30	20	10	10

Sorted

Deadline – Complete job on or before given time

objective:- Select job for max profit

$J_2 + J_1 + J_3$
$= 50 + 30 + 10$
$= 90$

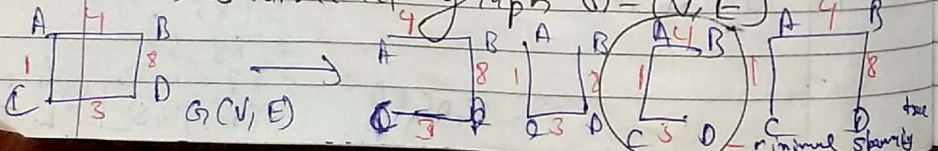


Algorithm

1. Sort jobs in profit decreasing order.
2. Shedule job in latest possible free slot meeting its deadline, (if available)

Spanning Tree :- It is a connected graph using all vertices in which there is no cycle

→ undirected graph $G = (V, E)$



Page No.
Date

Minimum Spanning Tree

We want to find a subset of E with minimum total weight / length that connect all the nodes/vertices in to a tree

undirected graph, weight, Edges

$ST(G, w)$ $w : E \rightarrow R$

1. $A = \phi$
2. while A does not form a S.T
3. find an edge (u, v) that is safe for A.
4. $A = A \cup (u, v)$
5. return A

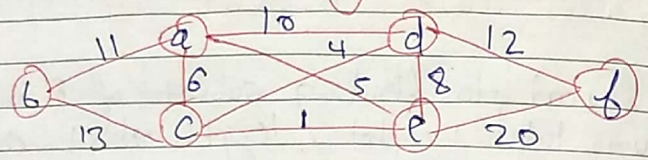
Applications :-

1. Computer Network – Wire connection
2. Travelling sell problems
-

* Minimum spanning tree can be found by Prims & Kruskal

More Applications – Pg 12-2016

Kruskal's Algorithm



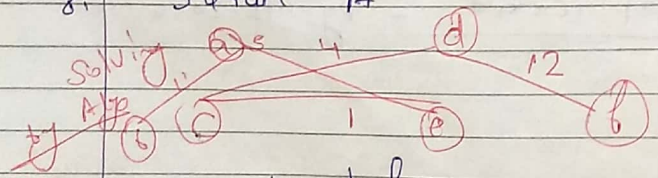
G(V, E)

MST- Kruskal (G, w)

1. $A = \phi$
2. for each vertex $v \in V[G]$

   Make Set(v) - a, b, c, d, e, f
3. Sort Edges weight in increasing order

   1, 4, 5, 6, 8, 10, 11, 12, 13, 20
4. for each edge (u, v) // sorted edge
5. if Find_Set(u) ≠ Find_Set(v)
6. $A = A \cup \{(u, v)\}$
7. Union (u, v)
8. return A



Kruskal

1. Select the shortest edge in a network.
2. Select the next shortest edge in a network, which not create a cycle
3. Repeat 2 until all v connected

Prim's

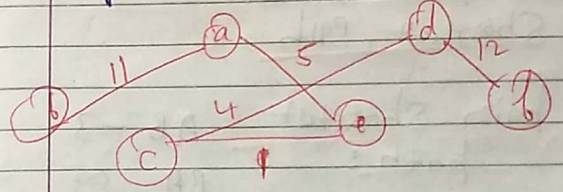
1. Select any vertex.
2. Select shortest edge connected to that vertex.
3. Repeat 2 cycle until all v connected

Prim's Algorithm

V	a, b, c, d, e, f
V-U	~~b c d e f~~
U	a e c d b f

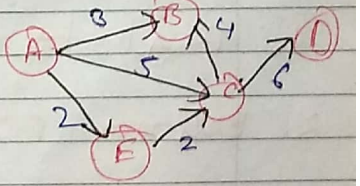
min (edge(a)) = 5

1. Choose initial vertex v and place it in U
2. From among the vertices in V-U choose that vertex, say v, which is connected to some vertex, say u, in U by an edge of least cost. Add v to U
3. Repeat step 2 until U = V



Shortest Path Problem

* G = (V, E) - weighted directed graph, w: E → R



weight w(P) of path $P = (v_0, v_1, \ldots, v_k)$ is the sum of weight of constituent edges

$$w(P) = \sum_{i=1}^{k} w(v_{i-2}, v_i)$$

Page No.
Date

Shortest path weight $\delta(u, v)$ = $\begin{cases} \min \{u(p): & \text{if there is path from } u \text{ to } v \\ \infty & \rightarrow \text{otherwise} \end{cases}$

$\delta(A,D) = \min \begin{cases} 11 \\ 10 \\ 13 \end{cases}$

10

Variant → Single Source Shortest Path : Shortest path from one node to all other node

Single Destination: Shortest Path

All pair Shortest path:

AB - 3
BA - ∞
AC - 5
CD - ∞

Algorithms used:

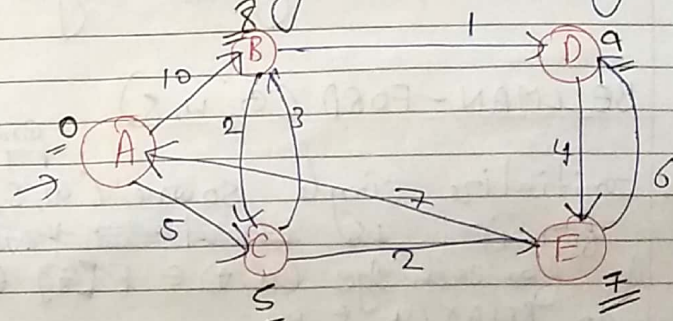
1. Dijkstra - for Single source [negative edges not allowed]
2. Bellman Ford - for Single sour but negate edge are allowed
3. Floyd warshall - for all pair Shortest path

Page No.
Date

Dijkstra's Algorithm

Single Source Shortest Path

"Shortest path from one node to all other node in which weights are non-negative"



Weight matrix / distance Matrix

A	B	C	D	E
0	∞	∞	∞	∞
0	10	5	∞	∞
0	8	5	∞	7
0	8	5	13	7
0	8	5	9	7

Bellman-Ford Algorithm

Single Source Shortest Path

"Shortest path from one node to all other nodes in which negative weights are allowed"

BELLMAN-FORD (G, w, S)

1. Initialize single source (G, S) — |V[G]|
2. for i = 1 to |V[G]| - 1
3. do for each edge (u, v) ∈ E[G] — E[G]
4. do RELAX (u, v, w)
5. for each edge (u, v) ∈ E[G]
6. do if v.d > u.d + w(u, v)
7. then return False
8. return true

Graph, total vertices, edges

Complexity

1 → |V| - 1

↓

|E|

O(|V|, |E|)

Relax (u, v, w)

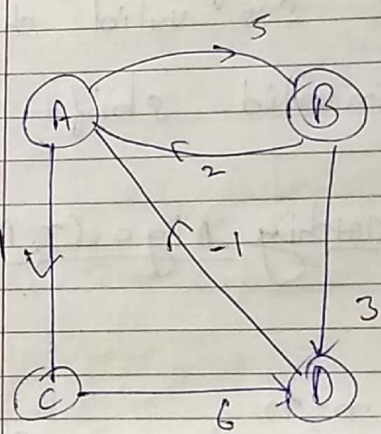
if v.d > u.d + w(u, v) then

v.d = u.d + w(u, v) → weight

v.P = u

↓

Previous value of v goes into u



Previous

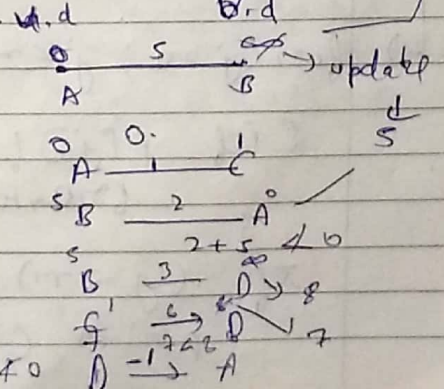
A	B	C	D
-	-	-	-
	A	A	B/C

A	B	C	D
0	∞	∞	∞
0	5	1	87

Edges	w
A → B	5
A → C	1
B → A	2
B → D	3
C → D	6

i = 1

i = 2



Page No. | Date

String Matching

- Pattern - P[m] → P[0, 1, ... m-1] T → | H | I | _ | H | E | L | L | O | _ | H | I | (indices 0 1 2 3 4 5 6 7 8 9 10; n = 11)

T[n] → T[0, 1 ... n-1]

P → | H | I | (indices 0 1; m = 2)

- P occurs with shift S in T if

$0 \le s \le n-m$

and

T[S+0, ... S+m-1] = P[0, 1 ... m-1]

- If P occurs with shift S in T then we call S is valid shift

otherwise S is invalid shift

$O((n-m+1)m)$

Naive - String - Matching Algo (T, P)

0

1. n = T. length
2. m = P. length
3. for s = 0 to n-m
4. If P[0, ... m-1] = T[S+0, ... S+m-1]
5. Print "Pattern occur with shift 's'

↳ for (j = 0; j ≤ m ; j++)

{ if P[j] != T[S+j]
break

}

If (j == m)
print ("pattern found at" s)

Complexity: $O((n-m+1) * m)$ (steps × step)

$\delta(q_0, a) \to q_1$

$\delta(q_1, b) \to q_2$

For aba

q_1

q_2

q_3 is final state (accepted)

Page No. | Date

Rabin-Karp Algorithm : String Matching

$O(m)$ avg; $O((n-m+1)m)$ worst

Rabin-Karp (T, P)

T = 3 1 2 3 4 8 6 2 (h^T)

P = 2 3 4 (h^P = function)

h(P) → 234 mod 13 ⇓ 0 (234 / 13)

1. n = T. length
2. m = P. length
3. h^P = hash (P)
4. h^T = hash (T[0 ... m-1])
5. for S = 0 to n-m
6. If (h^P == h^T)
7. if (P[0 ... m-1] = T[S+0, ... S+m-1])
8. Print "Pattern found with shift" s
9. if (S < n-m)
10. h^T = hash (T (S+1, ... S+m)) // 2nd h^T value will be taken

$O(m|\Sigma|)$ $O(n)$

Finite Automation Matcher

Finite State Matcher (P, δ)

{

1. m = P. length
2. q ← 0
3. for (i = 1 upto m)
4. q ← δ(q, P[i])
5. end loop
6. If (q is final state)
7. string accepted
8. else
rejected

Finite automation M is a 5-tuple (Q, q_0, F, Σ, δ)

→ Q is a finite set of state

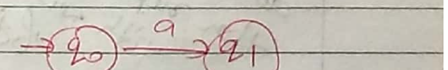
→ $q_0 \in Q$ is the start state

→ F → set of final state

- Σ → input symbol

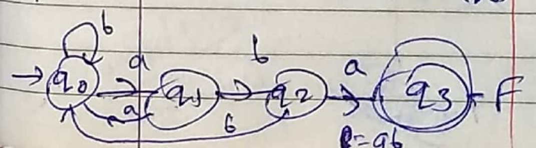
- δ is Transition Function

δ :- Q × Σ → Q



$q_0 \times a \to q_1$

Q = {q_0, q_1, q_2, q_3} Σ = {a, b} F = {q_3}



T = abac bda

P = ac

Page No. Date

Processing time $O(m)$ $O(n)$

Knuth-Morris-Pratt Algorithm (KMP Algo)

T = abc abdeabcb → n
P = abc dabcy → m   O(mn)

KMP → O(m+n)

Longest proper prefix which is also proper suffix

Possible Proper Prefix of abcd
↓
a
ab
abc

Possible Proper suffix of abcd
↓
d
cd
bcd

```
j=0
lps[0]=0
i=1
while (i<P.length)
{
    If (P[j]==P[i])
    {
        lps[i]= j+1;
        i++; j++;
    }
    else
    {
        If (j!=0) {j=lps[j-1]}
        else
        {lps[i]=0; i++;}
    }
}
```

this cond occur → do the cond repeat loop then with new value for j

Example

P	a	b	a	b
lps[]	0	0	0+1	1+1
	0	1	2	3

for abab

eg. abcd abc Y

Ans = 0000 1230

Page No. Date

KMP- Matcher Algorithm

KMP-MATCHER (T,P)

1. n ← |T| // length of text
2. m ← |P| // length of pattern
3. π ← KMP-Prefix (P) // lps
4. j ← 0
5. for i=1 upto n steps // till length of text
6. while (j>0 and P[j+1] ≠ T[i])
7. j ← π [j] // Skip Text using prefix table
8. If P[j+1] = T[i] then
9. j = j+1 // Next character
10. If j = m then // is all of P matched
11. Print "pattern occurs with shift" i-m
12. j = π [j] // Look the next match

Example

	1	2	3	4	5	6	7
Pattern [i]	A	C	A	C	A	G	T
π [i]	0	0	1	2	3	0	0

lps [i]   skip 2 char

how 14-7

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
T:	A	C	A	T	A	C	G	A	C	A	C	A	G	T

P: A C A C A G T

A C A C A G T

A C A C A G T

A C A C A G T

complexity
O(m+n)

P and NP class

class P:- Problem solvable in polynomial time

↳ $O(n^k)$ where k - constant

ie. size = n

Linear search algo

a[10] → LSA → found / not found

any element x in array which we have to find

not found → $O(n)$

⟹ It is a tractable problem

Ex: 2-CNF

Class NP: → It consists of those problem that are "verifiable" in polynomial time

→ Decision problems solvable in Non-Deterministic (guess) polynomial time

Yes / No

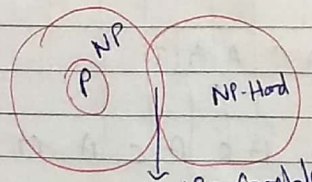
Examples

3-SAT

3-CNF

Travelling S Problem

Knapsack problem



* NP-Hard problems are hard to find solutions or solutions are not feasable.

NP

* A decision problem z is the NP if there exists a polynomial time algorithm A(x, y) such that for every x to problem z

$z(x) = 1 \iff$ there exists some y such that $A(x, y) = 1$ Yes

→ y is called a witness or certificate

Algorithm A is called a verifier or non-deterministic algorithm

↓ Polynomial time

3-SAT

Ex: Soduku

Find $x_1\ x_2\ x_3\ x_4$ value for which eq. is true

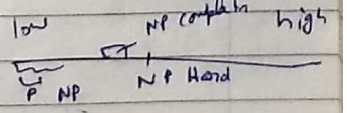
$(x_1 \vee x_2 \neg x_3) \wedge (\neg x_1 \vee x_3 \vee x_4)$

$\wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_1 \vee x_3 \vee \neg x_4)$

= true

N.P → D.P → Polysize certificate + verifier for Yes input

Difficulty level



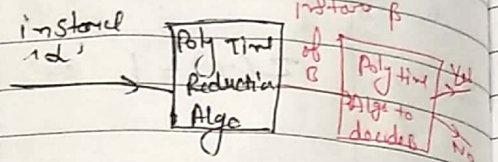
NP-Complete & Polynomial-time-Reduction

How to Prove X is NP complete

1) $X \in NP$ → Non-deterministic algo → certificate & Algo

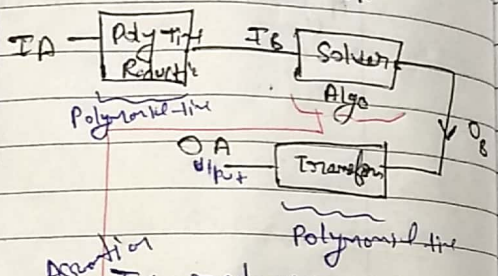
2) Reduce from known NP-complete problem Y to X

Reduction



B → NPC

A - NPC



Assumption: In reduction if

$A \in P$, $B \in P$

$A \in NP$, $B \in NP$

→ So it is also solvable i.p

↓

∴ but A is NPC and not part of P

So, this assumption is contradiction

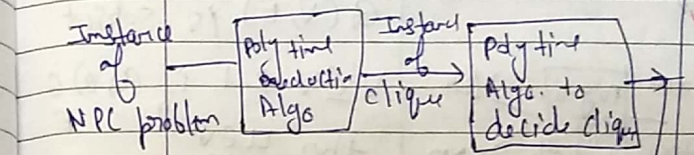
↳ therefore it also NPC

How to prove clique is NP complete

1) Clique $\in$ NP → NDP → C & A

2) A NPC problem can be reduced to clique decision problem

3CNF - SAT



$\phi = C_1 \wedge C_2 \dots C_k$

$(l_1^1 \vee l_2^1 \vee l_3^1) \wedge (l_1^2 \vee l_2^2 \vee l_3^3) \dots$

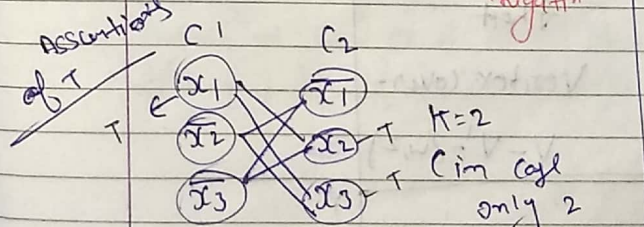
Should be true

$\phi = (x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3)$

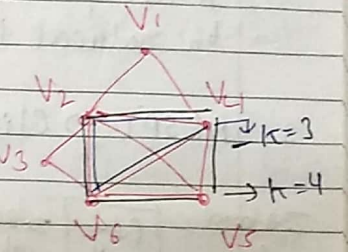
Graph

1) V_i^r — edge — V_j^s → $r \neq s$

2) $l_i^r = l_j^s$; edge $x_i \neq \neg x_i$ negation



Clique



Vertex-connected each other

Max-k-clique

clique is also a NP complete problem

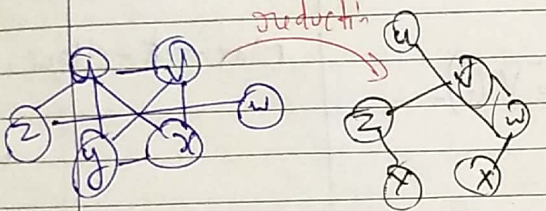
(27)

How to Prove Vertex Cover problem is NP Complete

1) Vertex cover Problem ∈ NP
2) A NPC problem (Clique) can be reduced to Vertex Cover problem
3 - CNF SAT → Clique → Vertex Cover

Instance of Clique — [Poly time reduct.] — Instance of Vertex cover — [poly-time algo to decide Vertex cover] — Yes / No

reduction



G(V,E) ⇒

Clique $V' = \{u, v, x, y\}$

$k = 4$

⇒ Clique $\leq_p$ Vertex cover

$\bar{G}$ = Fully Comp. graph − G

FCG − G

$\bar{G}$ produced by reduction algo that

Vertex cover = $V - V' = \{w, z\}$

Contrapositive of implication

$G \to V' \subseteq V$

↓

$|V'| = |V| - k$

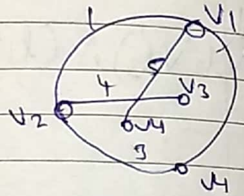
$\forall u, v \in E$

→ $u \in V'$ or $v \in V'$ or both

Vertex Cover Problem

" Vertex cover of undirected graph G(V, E) is subset $V' \subseteq V$ s.t if $(u, v) \in E$ then $u \in V'$ or $v \in V'$ or both $u, v \in V'$

" min. vertex that cover all edges "



$G \Rightarrow V = \{V_1, V_2, V_3, V_4\}$

$V' = \{V_1, V_2\}$

↓

verifies

i.e. NP

Divide & Conquer	Dynamic
6. Solutions are simple as compared to dynamic	6. Dynamic Programming solution can often be quite complex and tricky
7. used for any kind of problems	7. used generally for optimized problems
8. only one decision seq. is ever generated.	8. Many sequences may be generated.

Similarities b/w divide & conquer

1. Divided into stages, where decisions req. at each stage.
2. Each stage has a no. of state associated with it.
3. The decision at one stage transforms one state into a state in the next stage.
4. The optimal decision for each of the remaining states does not depend on the previous states or decision
5. The final stage must be solvable by itself.
6. recursive soln exist that identifies optimal decision for j, if j+1 is already solved.