8/3/18

MODULE-3

SPACE & TIME TRADE-OFF'S.

- An algorithm must be time efficient and space efficient.
- To achieve both may not be possible for some algorithms, in some situations space may be an important factor or time.
- Thus space and time trade off is a situation in which either time efficiency can be achieved at the cost of extra memory usage or space efficiency can be achieved at the cost of execution speed.
- Methods using which the time efficiency is achieved at the cost of space.

  i) Input enhancement.

  ii) Prestructuring.

  iii) Dynamic programming.

Input Enhancement :-

- Given a problem and various inputs, the input is preprocessed to get additional information about the problem.
- The additional information thus obtained may be stored in the form of a table which may be used by algorithm to get the required results with less ti
- For eg: sorting by counting technique.

ii) Prestructuring :-

- Its a method of achieving time efficiency that uses extra space to facilitate faster and flexible accessing of data.

  Eg : B-Trees, Hash table.

iii) Sorting by counting technique :-

- There are 2 methods :-
  - i) sorting by comparision
  - ii) sorting by distribution

★ SORTING BY COMPARISION :

step 1 : For each element a_i in the given list, find the total no. of elements c_i that are less than a_i

step 2 : The count c_i obtained in step 1 will be the position of a_i in the final sorted list

eg :

efficiency = n^2

a

25	45	10	20	50	15

c

0	1	2	3	4	5
3	4	0	2	5	1

$b[c[i]] = a[i]$

$b[3] = 25$

$b[4] = 45$

Algorithm -

```
for i ← 0 to n-1
    c[i] ← 0
for i ← 0 to n-2
    for j ← i+1 to n-1
```

```
if ( a[i] < a[j] )
    c[j] ← c[j] + 1
else
    c[i] ← c[i] + 1
end if
end for
end for
for i ← 0 to n-1
    b[c[i]] ← a[i]
end for
```

* SORTING BY DISTRIBUTION :-

- All the elements b/w the upper and lower bound should be there in the list otherwise this method is not possible.

$j \leftarrow a[i] - lb$ ⇒ to get the Distribution indices.

$d[j] \leftarrow d[j] - 1$.

a =

10	13	13	11	11	12	12
0	1	2	3	4	5	6

ub = 13.
lb = 10.

freq =

0	1	2	3
1	2	2	2

D = AF = Accumulated freq.

0	1	2	3
1	3	~~4~~ 5	7
1	~~1~~ 3	3	6.

10	11	11	12	12	13	13

$i \leftarrow n-1$ to 0

$a[i] =$

$j \leftarrow a[i] - lb$

$j \leftarrow 13 - 10$

$= 3$.

$d[j] \leftarrow d[j] - 1$

$B[d[j]] \leftarrow a[i]$

→ Efficient algorithm.

→ efficiency = n.

Algorithm :-

```
lb = min(a, n)
ub = max(a, n)
for i <- 0 to ub-lb
    d[i] <- 0.
end for.
for i <- 0 to n-1 do          // to obtain frequency
    j <- a[i] - lb            -preprocessing phases.
    d[j] <- d[j] + 1
end for.
for i <- 1 to ub-lb           // Accumulatede frequency
d[i] <- d[i] + d[i-1]         -preprocessing phases.
end for.
for i <- n-1 down to 0.
    j <- a[i] - lb.
    d[j] <- d[j] - 1
    B[d[j]] <- a[i]
end for.
```

String Matching :-

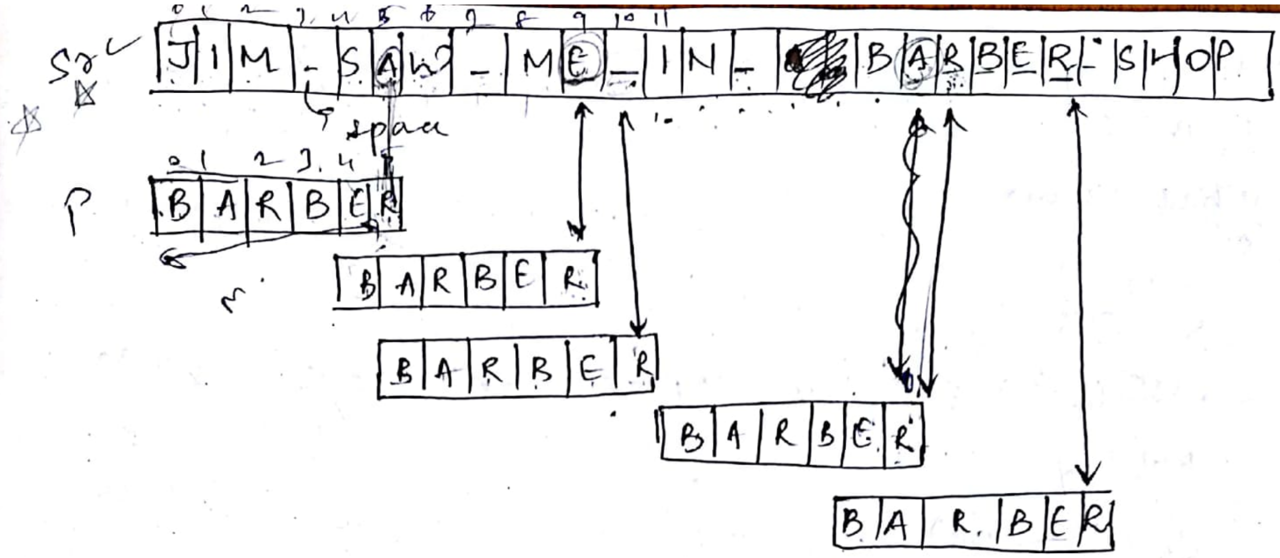
shift table :

J	T	H	* → any character.
2	1	3	3

← calculating the distance from right side of substring. considering (m-1) element first initially filled with no. of characters in substring (len

eg- Barber :-

B	A	R	E	*
~~6~~	~~6~~	~~6~~	~~6~~	6
2	4	3	1	



LABPROGRAM : 5

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#define MAX 126
int t[max] ;
void shifttable(char P[])
{
    int i,j,m;
    m= strlen(P);
    for(i=0;i<max;i++)
        t[i]=m;
    for(j=0;j<(m-1);j++)
        t[P[j]]=m-1-j; ✓
}
int horspool(char S[],char P[])
{
    int i,j,k,m,n;
    n=strlen(S);
    m=strlen(p);
printf("length of text = %d \n", n);
```

t → shift table array → size 126

P → pattern to be searched array.

S → length of the string.

```
printf("length of pattern=%d\n", m);
i=m-1;
while(i<n)
{
    k=0;
    while((k<m) && (p[m-1-k]==s[i-k]))
    k++;
    if(k==m)
    return(i-m+1);
    else
    i=i+t[s[i]];  → for shifting.
}
return -1;
}
void main()
{
  char src[100], p[100];
  int pos;
  clrscr();
  pf("Enter the text in which pattern is to be
       searched\n");
  ~~pf("%~~ gets(src);
  pf("Enter the pattern to be searched\n");
  gets(p);
  shifttable(p);
  pos = horspool(src, p);
  if (pos>=0)
   pf("The desired pattern was found
           starting from position %d", pos+1);
```

```
else
  pf("Pattern was not found \n");
getch();
}.
```

12/3/18

Dynamic Programming :-

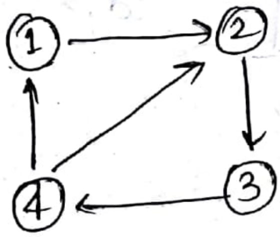
It's a method of solving a problem by recording the soln. of overlapped sub-problems of a given problem in a table and use these recorded soln. to get the soln. for the given problem.

*** Warshall's Algorithm :- (Transitive closure)**

~ If there is a path b/w i to j then $p(i,j) = 1$.

~ If $p(i,j) = 0$, this shows that there is no direct path from vertex $i \to j$. In this situation if there exist a path from $i \to k$ and $k \to j$. Then there exist a path from i to j.

~ Example :-



$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix} \end{array}$$

LAB PROGRAM 4(b) :-

```
step 1: for k <- 0 to (n-1)
            for i <- 0 to (n-1)
              for j <- 0 to (n-1)
                if a[i,j] = 0 && (if a[i,k] = 1 && a[k,j] = 1
                    a[i,j] = 1.
              end if.
        end for
```

⇒ Compute ${}^6C_3$ using Dynamic Programming:-

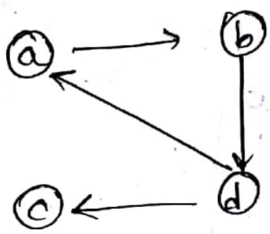
$${}^nC_k = {}^{n-1}C_k + {}^{n-1}C_{k-1}$$

$${}^0C_0 = {}^1C_0 = {}^2C_0 = {}^3C_0 = {}^4C_0 = {}^5C_0 = {}^6C_0 = 1.$$

$${}^1C_1 = {}^2C_2 = {}^3C_3 = 1$$

i \ j →	j=0	1	2	3
i=0	1	x	x	x
1	1	1	x	x
2	1	2	1	x
3	1	3	3	1
4	1	4	6	4
5	1	5	10	10
6	1	6	15	20

⇒ Apply Warshall's algorithm to compute transitive closure for the given graph:



⇒

$$\begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & 1 & 0 & 0 \\ b & 0 & 0 & 0 & 1 \\ c & 0 & 0 & 0 & 0 \\ d & 1 & \cancel{0}\,1 & 1 & 0 \end{array}$$

(a, b) = 1
(d, a) = 1
(d, a) = (a,
(d, b) = 1

$$\begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & 1 & 0 & \cancel{0}\,1 \\ b & 0 & \cancel{0}\,1 & 0 & 1 \\ c & 0 & 0 & 0 & 0 \\ d & 1 & 1 & 1 & \cancel{0}\,1 \end{array}$$

$$\begin{array}{c|cccc} & a & b & c & d \\ \hline a & \cancel{0}1 & 1 & \cancel{0}1 & 1 \\ b & \cancel{b}1 & 1 & \cancel{0}1 & 1 \\ c & 0 & 0 & 0 & 0 \\ d & 1 & 1 & 1 & 1 \end{array}$$

(a, d) = 1
(b, d) = 1
(d, a) = 1
(d, b) = 1
(d, c) = 1
(d, d) = 1

~~(d, c) = 1~~ . Transitive :-

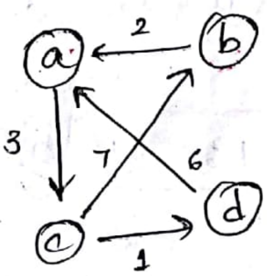
(a, d) = 1 = (d, a) = 1 ⇒ (a, a) = 1
(a, d) = 1 = (d, c) = 1 ⇒ (a, c) = 1
(b, d) = 1 , (d, a) = 1 ⇒ (b, a) = 1
(b, d) = 1 , (d, c) = 1 ⇒ (b, c) = 1

* Floyd's Algorithm :-
(All pairs shortest path algorithm).

- We find the shortest distance from all nodes to all other nodes.
- We assign a cost for each edge, $c[i, i] = 0$ (cost to reach the same node is 0)
- $c[i, j] = \infty$ if there is no edge from i to j.
- The input will be cost adjacency matrix.

Example :

Solve the all pair shortest path problem for the given graph



$$\begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & \infty & 3 & \infty \\ b & 2 & 0 & \cancel{\infty}5 & \infty \\ c & \infty & 7 & 0 & 1 \\ d & 6 & \infty & \cancel{\infty}9 & 0 \end{array}$$

take min $\{\infty, (2+3)\}$
= 5

(a, c) = 3.
(b, a) = 2.
(d, a) = 6.

(b, a) = 2 , (a, c) = 3 ⇒ (b, c) = 5.
(d, a) = 6 , (a, c) = 3 ⇒ (d, c) = 9
min $\{\infty, 9\}$ = 9

Scanned by CamScanner

$$
\begin{array}{c} \\ a \\ b \\ c \\ d \end{array}
\begin{array}{c}
\begin{array}{cccc} a & b & c & d \end{array} \\
\begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & 5 & \infty \\ \cancel{\infty}9 & 7 & 0 & 1 \\ 6 & \infty & 9 & 0 \end{bmatrix}
\end{array}
$$

(b, a) = 2
(b, c) = 5.
(c, b) = 7

(c, b) = 7, (b, a) = 2 ⇒ (c, a) =
(c, b) = 7, (b, c) = 5 ⇒ (c, c) = 12
min (0, 12) = 0

⇓

$$
\begin{array}{c} \\ a \\ b \\ c \\ d \end{array}
\begin{array}{c}
\begin{array}{cccc} a & b & c & d \end{array} \\
\begin{bmatrix} 0 & \cancel{\infty}10 & 3 & \cancel{\infty}4 \\ 2 & 0 & 5 & \cancel{\infty}6 \\ 9 & 7 & 0 & 1 \\ 6 & \cancel{\infty} & 9 & 0 \end{bmatrix} \\
\quad 16.
\end{array}
$$

(a, c) = 3
(b, c) = 5.
(d, c) = 9
(c, a) = 9.
(c, b) = 7
(c, d) = 1.

(a, c) = 3, (c, b) = 7 ⇒ (a, b) = 1
(a, c) = 3, (c, d) = 1 ⇒ (a, d) = 4
(b, c) = 5, (c, a) = 9 ⇒ (b, a) = 14
min(2, 1
(b, c) = 5, (c, d) = 1 ⇒ (b, d) = 6
(d, c) = 9, (c, a) = 9 ⇒ (d, a) = 18
min (6, 18) =
(d, c) = 9, (c, b) = 7 ⇒ (d, b) = 16
~~(d, c) = 9~~
~~(c, a) = 9, (c, a) = 9 = (d, a)~~,

⇓

$$
\begin{array}{c} \\ a \\ b \\ c \\ d \end{array}
\begin{array}{c}
\begin{array}{cccc} a & b & c & d \end{array} \\
\begin{bmatrix} 0 & 10 & 3 & 4 \\ 2 & 0 & 5 & 6 \\ \cancel{9}7 & 7 & 0 & 1 \\ 6 & 16 & 9 & 0 \end{bmatrix}
\end{array}
$$

(a, d) = 4.
(b, d) = 6.
(c, d) = 1.
(d, a) = 6
(d, b) = 16
(d, c) = 9

(a, d) = 4, (d, b) = 16 ⇒ (a, b) = 20
min (10, 20) = 1
(a, d) = 4, (d, c) = 9 ⇒ (a, c) = 13
min (3, 13) = 3
(b, d) = 6, (d, a) = 6 ⇒ (b, a) =
min (2, 12) = 2
(b, d) = 6, (d, c) = 9 ⇒ (b, c)
min (5, 15) = 5.
(c, d) = 1, (d, a) = 6 ⇒ (c, a)
min (9, 7) = 7

LAB PROGRAM - 4 (a):

Design, develop and execute a prog. in C to create a f[unction] called floyd's that takes cost adjacency matrix and implement all pair shortest path problem using floyds algorithms.

```
#include <stdio.h>
#include <conio.h>
void floyd (int a[10][10], int n)
 {
    int i, j, k;
    for(k = 0; k < n; k++)
      for (i=0 ; i < n; i ++)
        for (j = 0; j < n; j ++)
           if (a[i][j] > a[i][k] + a[k][j])
                a[i][j] = a[i][k] + a[k][j];
 }
```

Time efficiency $= n^3$

To input ∞ use 999.

```
void main()
{
 int n, i, j, k, a[10][10] ;
 clrscr();
 pf ("Enter the no. of nodes\n");
 sf ("%d", &n);
 pf (" Enter the cost adjacency matrix\n");
 ~~sf~~ for(i=0; i<n; i++)
        for(j=0 ; j<n; j++)
          ~~for(k=0; k<n; k++)~~
            sf("%d", &a[i][j]);
            floyd(a, n);
  pf(" The all pair shortest path matrix is:-\n");
  for (i=0; i<n ; i++)
   {
    for(i=0 ; j<n ; j++)
    {
      pf (" %5d", a[i][j]);
    } pf ("/n");
```

getch();
}

Time efficiency of floyd's algorithm :-

$$\sum_{k=0}^{n-1} \sum_{j=0}^{n-1} \sum_{i=0}^{n-1} 1.$$

$$\sum_{k=0}^{n-1} \sum_{j=0}^{n-1} (n-\cancel{1}-0+\cancel{1})$$

$$\sum_{k=0}^{n-1} n(n-\cancel{1}-0+\cancel{1})$$

$$\cancel{\sum_{k=0}} \; n(n)(n-1-0+1) = \underline{\underline{n^3}}.$$

LAB PROGRAM : 6

max = 15

n = 5.

0 | 1 knapsack.

0 → reject, 1 → accept.

w	4	5	8	7	3	→ weight
P	12	16	11	20	18.	→ cost.

→ wt. can be max. 15 choose the pairs such that cost is max from wt. 15. only.

~~# practice~~

Implement 0 or 1 knapsack problem using dynamic programming and obtain optimal soluⁿ.

```
#include <stdio.h>
#include <conio.h>
int v[10][10];
int max (int a, int b)
{
 return((a>b)? a : b);
}
```

```
int knapsack (int n, int m, int p[]{}, int w[]{})
{
   int i, j;
   for (i=0 ; i<=n ; i++)
   {
     for (j=0 ; j<=m ; j++)
      {
         if (i==0 || j==0)
            v[i][j] = 0;
         elseif (j - w[i] >= 0)
         v[i][j] = max(v[i-1][j], p[i]+v[i-1][j-w[i]
        else
        v[i][j] = v[i-1][j];
      }
   }
  return v[n][m];
}
void optimalsubset (int n, int m, int v[][], int w[])
 {
   int i, j;
    i = n;
    j = m;
   while ((i!=0) && (j!=0))
   {
     if (v[i][j] != v[i-1][j])
      {
        pf("Item= %d \n", i);
        j = j-w[i];
      }
    }
 }
```

i \ j	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0					
2	0					
3	0					
4	0					

```
void main()
{
 int n, m, p[10], w[10], i, j, value;
 clrscr();
 pf(" Enter the no. of items \n");
 sf("%d", &n);
 pf(" Enter wts of item \n");
 ~~sf("%d", &wt);~~
 for (i = 0; i<n; i++)
   sf("%d", &w[i]);
 pf(" Enter the value of each item \n");
 for (i=0; i<n; i++).
   sf("%d", &p[i]);
 pf(" Enter knapsack capacity \n");
   sf("%d", &m);
 ~~kna~~
   value = knapsack (n, m, p, w);
 pf("The soln. of knapsack problem is:- \n");
 for(i = 0; i <=n; i++)
 { for (j = 0; j <= m; j++)
      pf("%5d"; ~~value~~ v[i][j]);
      pf("\n");
 }
 pf(" The max. value is %d \n", value);
 pf(" the items of optimal subsets are \n");
 optimal subset (n, m, v, w);
 getch();
}
```