VASAVI COLLEGE OF ENGINEERING

(AUTONOMOUS)
(Affiliated to Osmania University)
Hyderabad - 500 031.

DEPARTMENT OF : CSE

NAME OF THE LABORATORY : DAA

Name K.S.I. SIVANI Roll No. -052 Page No. 66

PRELAB QUESTIONS-6:

1) What is Knapsack problem? Find the time complexity of Knapsack problem using greedy method?

→ The Knapsack problem refers to the common place problem of packing the useful items without overloading.

0/1 Knapsack Problem does not give an optimal solution.

Fractional Knapsack Problem gives an optimal solution and time complexity is O(nlogn).

2) Find the soln for Knapsack problem instance given below:

Item	1	2	3	4	5
Weight	5	10	15	22	25
Value	30	40	45	77	90

m = 60.

Greedy by profit: $m = 60$.

5 4 3 2 1

$W_5 = 25 < 60$

$p = 90$ $m = 60 - 25 = 35$.

3) What is the procedure to calculate total number of record movements in optimal file merge pattern?

→ Given n number of sorted files.

When 2 (or) more sorted files are to be merged altogether to form a single file; the minimum computations are done to reach this file are known as Optimal merge pattern.

4) How to calculate compression ratio and entropy in lossless compression algorithm?

compression ratio: $\frac{\text{size of o/p file}}{\text{group pages value}}$

5) Find the optimal storage of the programs given below 13 programs on 3 tapes T_0, T_1 & T_2 where the program of lengths 12, 5, 8, 32, 7, 5, 18, 26, 4, 3, 11, 10 & 6.

n=13; m=3

T_0	T_1	T_2
12	5	8
32	7	5
18	26	4
3	11	10
6		

$T_0 \rightarrow 3, 6, 12, 18, 32$

$mRT_0 = \frac{1}{5}[3+9+21+39+71]$

$= \frac{1}{5}[143]$

$= 28.6$

$T_1 \rightarrow 5, 7, 11, 26$

$mRT_1 = \frac{1}{4}[5+12+23+49]$

$= \frac{89}{4} = 22.25$

$T_2 \rightarrow 4, 5, 8, 10$

$mRT_2 = \frac{1}{4}[4+9+17+27]$

$= \frac{57}{4} = 14.25$

PRELAB PROGRAMS-6.

1) Implement Knapsack using Greedy method:

```
#include<stdio.h>
void knapsack(int n, float weight[], float profit[],
              float capacity)
{ float x[20], tp=0;
  int i, j, u;
  u = capacity;
```

```
for(i=0;i<n;i++){ x[i]=0.0; }.
for(i=0;i<n;i++)
 { if(weight[i]>u)   break;
    else
    { x[i]=1.0;
       tp=tp+profit[i];
        u=u-weight[i];}}
 if(i<n)
  { x[i]=u/weight[i];}
  tp=tp+(x[i]*profit[i]);
  printf("\n The result vector is:-");
  for(i=0;i<n;i++){printf("%f\t",x[i]);}
  printf("\n Maximum profit is:%f",tp);}
int main()
{ float weight[20], profit[20], capacity;
   int num,i,j;
   float ratio[20],temp;
   printf("Enter the no. of objects:");
   scanf("%d",&num);
```

Output:

```
Enter the no. of objects :7
Enter the weights and profits of each object:
2 10
3 5
5 15
7 7
1 6
4 18
1 3
Enter the capacity of knapsack :15
The result vector is: 1.000000 1.000000 1.000000 1.000000
1.000000 0.666667 0.000000
Maximum profit: 55.333332
```

VASAVI COLLEGE OF ENGINEERING

(AUTONOMOUS)
(Affiliated to Osmania University)
Hyderabad - 500 031.

DEPARTMENT OF : CSE

NAME OF THE LABORATORY : DAA

Name K.S.I.SIVANI Roll No. -052- Page No. 70

```
printf("\n Enter the weights and profits of each
        object: ");
for(i=0;i<num;i++)
{scanf("%f %f", &weight[i], &profit[i]); }
printf("\nEnter the capacity of knapsack: ");
scanf("%f", &capacity);
for(i=0;i<num;i++) {ratio[i] = profit[i]/weight[i];
for(i=0;i<num;i++)
{for(j=i+1;j<num;j++)
  {if(ratio[i] < ratio[j])
      {temp=ratio[j];
       ratio[j] = ratio[i];
       ratio[i] = temp;

       temp = weight[j];
       weight[j]= weight[i];
       weight[i] = temp;
       temp = profit[j];
       profit[j]=profit[i]; profit[i] = temp;}}}
```

```
knapsack(num, weight, profit, capacity);
return 0; }
```

2) Implement optimal merge patterns using Huffman encoding algorithm:

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int frequency; struct Node *left, *right;};

struct Node * createNode (int frequency)
{ struct Node *newNode =(struct Node*)malloc
                        (sizeof (struct Node));

    newNode → frequency = frequency;
    newNode → left = newNode → right = NULL;
    return newNode; }

int compareNodes (const void *a, const void * b)
 {  struct Node *node1 = *(struct Node **)a;
     struct Node * node2 = * (struct Node **) b;
     return (node1→frequency - node2→frequency); }
```

```
struct Node *buildHuffmannTree(int frequencies[],
                               int size)
{ struct Node * left, *right, * top;
  int capacity = size;
  struct Node **heap = (struct Node **) malloc
                       (capacity *sizeof(struct Node*));

  int i;
  for(i=0; i<size ; ++i)
     {heap[i] = createNode(frequencies[i]) ; }
int heapSize = size;
qsort (heap, heapSize, sizeof(struct Node *), compareNodes);
while (heapSize >1)
   { left = heap[0]; right = heap[1];
     top = createNode (left -> frequency + right -> frequency);
     top-> left = left;
     top->right = right;
     heap[0] = top;
     heap[1] = heap[heapSize-1];
```

```
    --heapSize;
    qsort(heap, heapSize, sizeof(struct Node*),
                                compareNodes); }
    free(heap); return top; }

void assignHuffmanCodes(struct Node *root,
                        int code[], int top)
{ if (root->left)
  { code[top] = 0;
    assignHuffmanCodes(root->left, code, top+1); }
  if(root->right)
  { code[top] = 1;
    assignHuffmanCodes(root->right, code, top+1); }
  if(!(root->left) && !(root->right))
  { printf("Frequency: %d, Code: ", root->frequency);
    int i;
    for(i=0; i<top; ++i) { printf("%d", code[i]); }
    printf("\n"); }}
```

Output:

Optimal Merge Patterns:

Frequency : 10, Code : 0

Frequency : 3, Code : 100

Frequency : 5, Code : 101

Frequency : 8, Code : 11


```
void optimalMergePatterns(int files[], int size)
{ struct Node *root = buildHuffmanTree(files, size);
  int code[100];
  assignHuffmanCodes(root, code, 0); }
int main()
{ int files[] = {3, 5, 8, 18};
  int size = sizeof(files) / sizeof(files[0]);
  printf("Optimal Merge Patterns: \n");
  optimalMergePatterns(files, size);
  return 0; }
```

Implement optimal storage on tapes by using Greedy method:

```
#include <stdio.h>
#include <stdlib.h>
int optimalStorageOnTapes(int files[], int numFiles,
                          int numTapes)
{ int *tapes = (int *)malloc(numTapes * sizeof(int));
```

```
int i, j, minIndex, minSum;
for(i=0; i<numTapes; i++) {tapes[i]=0;}

for(i=0; i<numFiles; i++)
 { minIndex=0;
   minSum=tapes[0];
  for(j=1; j<numTapes; j++)
    { if (tapes[j]<minSum)
        { minIndex=j; minSum=tapes[j]; }}
   tapes[minIndex] += files[i]; }

   int totalTime=tapes[0];
  for(i=1; i<numTapes; i++)
   { if (tapes[i] > totalTime) {totalTime=tapes[i];}}
   free(tapes);
   return totalTime; }
int main ()
{ int files[]={10,5,7,12,18,3};
 int numFiles = sizeof(files)/sizeof(files[0]);
```

```
int numTapes = 3;
int totalTime = optimalStorageOnTapes(files, numFiles,
                                      numTapes);
printf("Total time taken for optimal storage on
        tapes: %d\n", totalTime);
return 0; }
```

Output:

Total time taken for optimal storage on tapes: 25

VASAVI COLLEGE OF ENGINEERING

(AUTONOMOUS)
(Affiliated to Osmania University)
Hyderabad - 500 031.

DEPARTMENT OF : CSE

NAME OF THE LABORATORY : DAA

Name K.S.I.SIVANI Roll No. -052 Page No. 17

PRELAB QUESTIONS-7

1) What is purging rule?

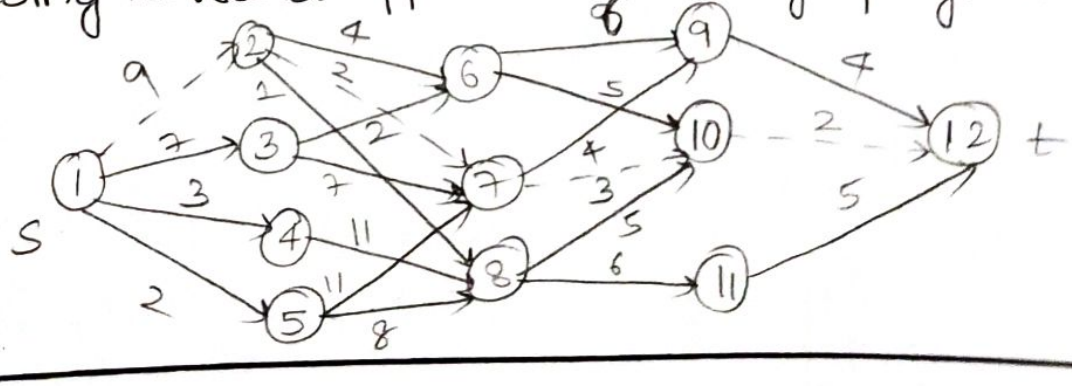
The Purge rule defines how real-time data is stored in system memory. To limit amount of memory to consumed; real time data must occasionally be removed or purged from the system memory.

2) Write the cost function to find shortest path from source to destination in multi-stage graph by using backward approach:

Cost function for backward approach

$$bcost(i,j) = \min\{bcost(i-1,l) + c(l,j)\}$$

3) Find the shortest path from source to destination by using forward approach for the graph given below:



Cost function:

$$Cost(i,j) = \min_{l \in V} \{ c(j,l) + c(i+1,l) \}$$

$cost(4,9) = \min\{c(9,12) + c(5,12)\} = 4+0 = 4$

$cost(4,10) = \min\{c(10,12) + c(5,12)\} = 2+0 = 2$

$cost(4,11) = \min(c(11,12) + c(5,12)) = 5+0 = 5$

$cost(3,6) \Rightarrow l = \{9,10\} = \min\{(c(6,9) + c(4,9)),$
$(c(6,10) + c(4,10))$

$= \min\{(10),(7)\}$

$cost(3,6) = 7 \quad (l = 10)$

$cost(3,7) = l\{9,10\} = \min\{c(7,9) + c(4,9),$
$c(7,10) + c(4,10)\}$

$= \min\{8,5\} = 5.$

$l = 10; cost(3,7) = 5$

$cost(3,8) = l = \{10,11\} = 7 \quad (l=10).$

$cost(2,2) = l = \{6,7,8\} = 7 \quad (l=7)$

cost(2,3) = l = {6,7} = 9 (l=6)

cost(2,4) = l = 8 = 18 (l=8)

cost(2,5) = l = {7,8} = 15 (l=8)

cost(1,1) = l = {2,3,4,5} = 16 (l=2)

Path: 1 → 2 → 7 → 10 → 12

4) Find the solution for the following 0/1 Knapsack instance n=4; W = 5kg; (w1, w2, w3, w4) = (2,3,4,5); (b1, b2, b3, b4) = (3,4,5,6)

n = 4 W = 5

Weights = [2,3,4,5] benefits = [3,4,5,6]

$B(i,j) = \max\{B(i-1,j), b\_i + B(i-1, j-w\_i)\}$

$B(0,j) = 0 \ \forall j$ (no items selected).

$B(i,0) = 0 \ \forall \ i$ (knapsack capacity = 0).

Maximum value that can be achieved is 7.

VASAVI COLLEGE OF ENGINEERING

(AUTONOMOUS)
(Affiliated to Osmania University)
Hyderabad - 500 031.

DEPARTMENT OF : CSE

NAME OF THE LABORATORY : DAA

Name K.S.I.SIVANI Roll No. -052 Page No. 80

PRELAB PROGRAMS-7:

1) Implement 0/1 Knapsack algorithm:

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

int knapSack(int W, int wt[], int val[], int n)
 { int i,w;
   int K[n+1][W+1];

   for(i=0; i<=n; i++)
   { for(w=0; w<=W; w++)
     { if(i==0 || w==0) { K[i][w]=0; }
       else if(wt[i-1]<=w)
        { K[i][w] = max(val[i-1]+K[i-1][w-wt[i-1]],
                        K[i-1][w]); }
       else
        { K[i][w] = K[i-1][w]; } } }
   return K[n][W]; }
```

```
int main()
{ int val[] = {60, 100, 120};
  int wt[] = {10, 20, 30};
  int W = 50;
  int n = sizeof(val)/sizeof(val[0]);
  int result = knapSack(W, wt, val, n);
  printf("Maximum value: %d\n", result);
  return 0; }
```

O/P:

Maximum Value: 220.

---

2) Implement multi-stage graph with backward approach:

```
#include<stdio.h>
#include<stdlib.h>
#define MAX_NODES 100
typedef struct {
    int stage; int cost; int parent; } Node;
```

```
void multiStageGraph(int stages, int nodesPerStage,
                     int *edges, int *costs, int startNode,
                     int endNode)
{ Node graph[MAX_NODES];
  int i, j, k, minCost;
 for(i=0;i<MAX_NODES;i++)
  { graph[i].stage = -1; graph[i].cost = -1;
    graph[i].parent = -1; }
 k = nodesPerStage * (stages-1);
for(i=0;i<nodesPerStage;i++)
{   graph[k+i].stage = stages-1;
    graph[k+i].cost = costs[k+i]; }
for(i=stages-2;i>=0;i--)
 {for(j=0;j<nodesPerStage;j++) { minCost=-1;
  for(k=0;k<nodesPerStage;k++)
   { if(edges[j*nodesPerStage+k]!=-1)
    { int cost = graph[(i+1)*nodesPerStage+k].cost
                 +edges[j*nodesPerStage+k];
```

Output:

Minimum Cost : 13
Path : 0 → 6 → 13 → 19

VASAVI COLLEGE OF ENGINEERING
(AUTONOMOUS)
(Affiliated to Osmania University)
Hyderabad - 500 031.

DEPARTMENT OF : CSE

NAME OF THE LABORATORY : DAA

Name K·S·I·SIVANI Roll No. -052 Page No. 83

```
    if(minCost==-1 || cost < minCost)
      { minCost = cost;
        graph[i * nodesPerStage + j].parent = (i+1)*nodesPerStage
                                              +k;
      }}}
    graph[i*nodesPerStage + j].stage = i;
    graph[i*nodesPerStage + j].cost = minCost; }}
    printf("Minimum Cost : %d\n", graph[startNode].cost);
    printf("Path: ");
    printf("%d", startNode);
    i = startNode;
    while (i != endNode)
      { i = graph[i].parent; printf("→ %d", i); }
    printf("\n"); }
int main()
{ int stages = 4;
  int nodesPerStage = 5;
  int edges[] = { -1, 1, 2, -1, -1,
                  -1, -1, -1, 5, 6,
                  -1, -1, 7, -1, -1
                  -1, -1, -1, -1, 8
                  -1, 9, -1, -1, -1 };
```

```
int cost[] = {0,1,2,3,4,5,6,7,8,9};
int startNode = 0;
int endNode = 19;
multiStageGraph(stages, nodesPerStage, edges, costs,
                startNode, endNode);
return 0;}
```

3) Implement Multi-stage graph with forward approach:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_NODES 100
typedef struct
{ int stage; int cost; int parent; } Node;
void multiStageGraph (int stages, int nodesPerStage,
                      int * edges, int *costs, int startNode,
                      int endNode)
{ Node graph[MAX_NODES],
  int i, j, k, minCost;
```

```
for(i=0; i<MAX_NODES; i++)
{ graph[i].stage = -1; graph[i].cost = -1;
  graph[i].parent = -1; }
k=0;
for(i=0; i<nodesPerStage; i++)
 { graph[k+i].stage =0;
   graph[k+i].cost = costs[k+i]; }

for(i =1; i< stages; i++)
{ for(j=0; j<nodesPerStage; j++)
  { minCost = -1;
    for(k=0; k <nodesPerStage; k++)
    { if (edges[(i-1)*nodesPerStage +k*stages+j]!=-1)
      { int cost = graph[(i-1)* nodesPerStage+k].cost
                   +edges[(i-1)* nodesPerStage+k*stages+j];
        if (minCost ==-1 || cost<minCost)
          { minCost = cost; graph[i*nodesPerStage+j].parent =
                                (i-1)* nodesPerStage+k; }}}}
```

Output:

Minimum cost: 6
Path: 9 3

VASAVI COLLEGE OF ENGINEERING
(AUTONOMOUS)
(Affiliated to Osmania University)
Hyderabad - 500 031.

DEPARTMENT OF : CSE

NAME OF THE LABORATORY : DAA

Name K.S.T.SIVANI Roll No. -052- Page No. 86

```
if(graph[endNode].cost == -1)
{ printf("No valid path from start to end node\n");
  return; }
printf("Minimum Cost : %d\n", graph[endNode].cost);
printf("Path: ");
i = endNode;
while(i != -1)
{ printf("%d", i); i = graph[i].parent; }
printf("\n"), }
int main() { int stages = 4; int nodesPerStage = 5;
int edges[] = { -1, -1, -1, -1, -1,
                 1, -1, -1, -1, -1,
                -1,  2, -1, -1, -1,
                -1,  3, -1, -1, -1,
                -1, -1,  1,  4, -1,
                -1, -1, -1, -1,  5,
                -1, -1, -1,  6, -1,
                -1, -1,  7, -1, -1,
                -1,  8, -1, -1, -1,
                 9, -1, -1, -1, -1 };
int cost[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
```

```
int startNode=0;
int endNode=9;
multistageGraph(stages,nodesPerStage, edges, costs,
                startNode, endNode);

return 0; }.
```

LAB PROGRAMS - 7

→ Alice and Bob take turns playing a game, with Alice starting first. Initially, there is a number n on the chalkboard. On each player's turn, that player makes a move consisting of:

- Choosing any x with $0 < x < n$ and $n \% x == 0$
- Replacing the number n on the chalkboard with $n - x$

Also if a player cannot make a move, they lose the game.

Return true if and only if Alice wins the game, assuming both players play optimally

```
#include <stdio.h>
int main()
{ int x,n;
 printf("Enter the number on the chalkboard:");
 scanf("%d",&n);
 x = 1;
 int i, count=0;
 int A=0,B=0;
 while(n>1)
 { for(i=n-1; i>0; i--)
   {  if(n%i==0)
       { x=i; break;}}
   if(count %2==0)
    {A=1; B=0;}
   else {B=1;A=0;}
    n=n-x; count++;}
 if(A==1)
{ printf("True");}
else { printf("False");}
 return 0;}
```

Output:

```
Enter the number on the
chalkboard: 2

True.
```

VASAVI COLLEGE OF ENGINEERING

(AUTONOMOUS)
(Affiliated to Osmania University)
Hyderabad - 500 031.

DEPARTMENT OF : CSE

NAME OF THE LABORATORY : DAA

Name K.S.I.SIVANI Roll No. -052 Page No. 89

PRELAB QUESTIONS-8

1) What is the memorization? What are the supporting languages to implement high performance algorithms with dynamic programming?

→ Memorization refers to a technique used to optimize the performance of recursive algorithms by storing the results of expensive function calls and reusing them instead of recomputing.

→ Languages for implementing high performance algorithms with dynamic programming are python, c++ Java etc.

2) Distinguish b/w top-down and bottom up approaches used in dynamic programming.

→ Top-down approach starts from the original problem and recursively solves the sub problems.

→ Bottom up approach starts from the smallest subproblems & iteratively builds up the sol^ns to

to larger subproblems until the original problem is solved.

3) What is dynamic programming? What is the optimal stlbstructure and optimal sol^n?

→ Dynamic programming is a technique used in computer science and mathematics to solve complex problems by breaking them down into overlapping sub-problems & solving them in an optimal manner.

→ Optimal substructure refers to the property of a problem where an optimal sol^n to the problem can be constructed from optimal solutions to its subproblems

→ Optimal sol^n in the context of dynamic programming refers to the best possible sol^n to the original problem.

4) Write algorithm to print optimal order sequence of matrix chain multiplication and derive the time complexity: