INDEX

IBM22CS283

Name Sneha N Shastri

Standard Section Roll No.

Subject

S.No.	Date	Title	Page No	Teacher's Sign
22	12/07	N-Queens	10	16/7/24

03/05/2024

1. Linear Search

```
#include <stdio.h>

void lsearch(int, int , int [ ]);

void main( )
{
   int n;
   int key;
   printf("Enter the value of n: \n");
   scanf("%d", &n);
   int arr[n];
   printf("Enter array values: \n");
   int j;
   for(j=0;j<n;j++)
   scanf("%d", &arr[j]);
   printf("Enter search value: \n");
   scanf("%d", &key);
   lsearch(n, key, arr);
}

void lsearch(int n, int key, int arr[ ])
{ int i;
  for (i=0; i<n; i++)
  {
   if (arr[i]==key)
   {
    ("%d found at position : %d \n", key,
                                      i+1);
exit (0);
}
}
printf ("%d not found.", key);
}
```

Output :

```
Case - 1
Enter the value of n:
5
Enter the array values:
4 5 7 9 0
Enter search value:
0
0 found at position : 5

Case - 2
Enter the value of n
3
Enter array values:
1 3 6
Enter search value
7
7 not found.
```

2. Binary Search

```
#include <stdio.h>
void bsearch (int, int, int [ ]);
~~void bsearch (int~~
void main ()
{
  int n;
  int key;
  printf("Enter the value of n: \n");
  Scanf ("%d", &n);
  int arr[n];
  printf ("Enter sorted array values: \n");
  int j;
  for(j=0; j<n; j++)
  Scanf("%d", &arr[j]);
  printf("Enter search value :\n");
  Scanf("%d", &key);
  bsearch (n, key, arr);
}

void bsearch (int n, int key, int arr [ ])
{
  int lb = 0;
  int ub = n-1;
  int mid = (lb + ub) /2;
  int pos = -1;
  while (lb <= ub)
{
  mid=(lb+ub)/2;
  if(arr[mid]==key)
  {
    pos=mid+1;
    printf("%d found at position: %d \n",
             key,pos);
   exit(0);
  }
  else if (key > arr[mid])
     lb=mid+1;
  else if(key < arr[mid])
     ub=mid-1;
 }
 printf("%d not found.\n",key);
}
```

Output:

Case 1:

```
Enter the value of n:
5
Enter sorted array values:
1 4 5 7 9
Enter search value:
5
5 found at position 3
```

Case 2:

```
Enter value of
n:
3
Enter sorted array
Values:
1 4 6
Enter search value:
7
7 not found.
```

3. Bubble Sort

```
#include <stdio.h>
void bubblesort(int n, int []);

void main()
{
 int n;
 printf("Enter the value of n:\n");
 scanf("%d", &n);
 int arr[n];
 printf("Enter unsorted array values:\n");
 int i;
 for(i=0;i<n;i++)
 scanf("%d",&arr[i]);
 bubblesort(n, arr);
}

void bubblesort (int n,int arr[])
{
  for(int i=0;i<n-1;i++)
  {
   for(int j=0;j<n-1-i;j++)
   {
   if(arr[j] > arr[j+1])
    {
     int t=arr[j];
     arr[j]=arr[j+1];
     arr[j+1]=t;
   }
}
}
printf("Sorted array in ascending order : \n");
for(int i=0;i<n;i++)
printf("%d \t", arr[i]);
}
```

i=0→3
j=1→3
9 8 7 6 8

Output:

```
Enter the value of n:
5
Enter unsorted array values:
9 4 7 8 2
Sorted array in ascending order:
2 4 7 8 9
```

4. Selection Sort

```
#include <stdio.h>
void selectionsort(int n, int[]);
void main()
{
  int n;
  printf("Enter the value of n: \n");
  scanf("%d ",&n);
  int arr[n];
  printf("Enter unsorted array values: \n");
  int i;
for(i=0; i<n; i++)
scanf ("%d ", &arr[i]);
selectionsort(n, arr);
}

void selectionsort(int n, int arr[])
{
  for(int i=0; i<n-1, i++)
  {
    int min = i;
   for(int j=i+1; j<n; j++)
   {
     if (arr[j] < arr[min])
     {
       ~~int t=a~~
       min = j;
     }
   }
  int t = arr[i];
  arr[i] = arr[min];
  arr[min] = t;
  }
printf(" Sorted array in ascending order:\n");
for(int i=0; i<n; i++)
printf("%d \t", arr[i]);
}
```

3/5/24

Output:

```
Enter value of n:
5
Enter unsorted values:
9 6 5 7 0
```

```
Sorted array in ascending order:
0 5 6 7 9
```

For time complexity:

Implement the following in main

```
void main()
{
  float a;
  clock_t time_req;
  time_req=clock();
  // Execution statements eg. Linear Search
  time-req=clock()-time_req;
  printf("Processor time taken: %f seconds\n",
         (float) time_req / CLOCKS_PER_SEC);
}
```

For linear search:

```
Processor time taken: 0.000170 seconds
```

31/05/2024

1. ~~ast~~ Topological Sort - DFS

Output:

```
Enter no. of nodes:
5
Enter adjacency matrix:
0 0 1 1 0
1 0 0 1 0
0 0 0 0 1
0 0 1 0 1
0 0 0 0 0

Topologically sorted array is:
1 4 0 3 2
```

2. GCD - Recursive

```
Enter two numbers to calculate gcd:
6 10
Result : 2
```

3. Tower of Hanoi

```
Enter no of discs:
3
Move disc 1 from s to d
Move disc 2 from s to 2
Move disc 1 from d to t
Move disc 3 from s to d
Move disc 1 from t to s
Move disc 2 from t to d
Move disc 1 from s to d
```

4. Topological sort - source removal

```
Output:
Enter no of nodes:
5
Enter adjacency matrix:
0 0 1 1 0
1 0 0 1 0
0 0 0 0 1
0 0 1 0 1
0 0 0 0 0
Topologically sorted array is:
1 4 0 3 2
```

5. Lomuto partition

```
Enter no. of elements:
9
Enter array elements:
4 1 10 8 7 12 9 2 15
~~Enter~~ Enter value of k:
4
Result: 7
```

07-06-2024

1. Merge Sort

```
#include <stdio.h>
void simplemerge(int [], int, int, int);
void mergesort(int [], int, int);
int c[100];
void main()
{
 int ~~to~~ n, low, high;
 printf("Enter the no. of elements :\n");
 Scanf("%d", &n);
 int a[n];
 low=0;
 high=n-1;
 printf("Enter the unsorted array values:\n");
 for (int i=0; i<n; i++)
  Scanf("%d",&a[i]);
 printf("Sorted elements are :\n");
 mergesort(a, low, high);
 for(int i=0;i<n;i++)
 {
  printf("%d\t", a[i]);
 }
}
```

```
Void mergesort(int a[], int low, int high)
{
   int mid;
   if (low<high)
   {
     mid =(low+high)/2;
     mergesort(a, low, mid);
    mergesort (a, mid+1, high);
   Simplemerge (a,low,mid,high);
   }
}
```

```
void simplemerge (int a[],int low,int mid,
                              int high)
 {
   int i= low, k=0,j;
   j=mid+1;
   int n=high+1;
   while(i<=mid && j<=high)
   {
    if (a[i]<a[j])
    {
      c[k++]=a[i++];
    }
   else
    {c[k++]=a[j++];
    }
}
```

```
while(i<=mid)
    c[k++]=a[i++];
while(j<=high)
    c[k++]=a[j++];
for (int i=low;i<n; i++)
 {
   a[i] = c[i-low];
 }
}
```

Output:

```
Enter no.of elements:
8
Enter unsorted array values:
8 6 2 4 3 1 7 5

Sorted elements are:
1 2 3 4 5 6 7 8
```

2. Quick Sort

```
#include<stdio.h>
int partition (int [], int,int);
void quicksort (int [], int, int);
void main()
{
```

```
Void mergesort(int a[], int low, int high)
{
  int mid;
  if(low<high)
  {
    mid = (low+high)/2;
    mergesort (a, low, mid);
    mergesort (a, mid+1, high);
    simplemerge (a,low,mid,high);
  }
}

void simplemerge (int a[], int low, int mid,
                              int high)
{
  int i= low, k=0, j;
  j= mid+1;
  int n=high+1;
  while(i<=mid && j<=high)
  {
    if (a[i]<a[j])
    {
      c[k++]=a[i++];
    }
    else
    {c[k++]=a[j++];
    }
  }
while(i<=mid)
  c[k++]=a[i++];

while(j<=high)
  c[k++]=a[j++];

for(int i=low;i<n;i++)
{
  a[i] = c[i-low];
}
}
```

Output:

```
Enter no. of elements:
8
Enter unsorted array values:
8 6 2 4 3 1 7 5

Sorted elements are:
1 2 3 4 5 6 7 8
```

2. Quick Sort

```
#include<stdio.h>
int partition (int [], int,int);
void quicksort (int [], int, int);
void main ()
{
```

```
int n, low, high;
printf("Enter no. of elements: \n");
Scanf("%d", &n);
int a[n];
low=0;
high=n-1;

printf("Enter the unsorted array values: \n").
for(int i=0; i<n; i++)
{
    scanf("%d", &a[i]);
}

printf("sorted elements are :\n").
quicksort(a, low, high);
for(int i=0; i<n; i++).
 {
    printf("%d \t", a[i]);
 }
}

void quicksort(int a[], int low, int high)
 {
    int mid;
    if (low<high)
    {
      mid= partition(a, low, high);
      quicksort(a, low, mid-1);
      quicksort(a, mid+1, high);
```

```
    }
 }

int partition(int a[], int low, int high)
 {
   int i=low;
   int j=high+1;
   int pivot=a[low];
  while(i<=j)
  {
    do
    {
      i=i+1;
    }while(a[i]<pivot && i<=high);

   do
   {
     j=j-1;
   }while(a[j]>pivot && j>=low);
   if(i<j)
   {
     int t=a[i];
     a[i]=a[j];
     a[j]=t;
   }
 }
 int k= a[low];
 a[low]=a[j];
 a[j]=k;
 return j;
}
```

Output:

```
Enter no. of elements:
5
Enter unsorted array values:
7 6 9 4 2
Sorted elements are:
2 4 6 7 9
```

7/6/24

13-06-2024

1. Warshall Algorithm:

```
#include <stdio.h>
void warshall(int [][100], int);

void main()
{
 int n;
 printf("Enter the no. of vertices:\n");
 scanf("%d", &n);
 printf("Enter the adjacency matrix:\n");
 int a[100][100];
 for(int i=0; i<n; i++)
 {
  for(int j=0; j<n; j++)
  {
    scanf("%d", &a[i][j]);
  }
 }
 printf("Path Matrix is:\n");
 warshall(a,n);
}

void warshall(int a[][100], int n)
{
 int P[n][n];
 int i,j,k;
for(~~int~~ i=0;i<n;i++)
for (j=0; j<n; j++)
{
  P[i][j]=a[i][j];
  }
}

for (k=0; k<n; k++)
 {
  for (i=0; i<n; i++)
  {
   for (j=0; j<n; j++)
   {
    if((P[i][j]==0 && (P[i][k]==1 &&
        P[k][j]==1))
      P[i][j]=1;
   }
  }
}

for(i=0; i<n; i++)
{
 for(j=0;j<n;j++)
 {
  printf("%d\t", P[i][j]);
 }
 printf("\n");
}
}
```

Output:

```
Enter the no. of vertices:
4
Enter the adjacency matrix:
0 1 0 0
0 0 0 1
0 0 0 0
1 0 1 0
Path matrix is:
1 1 1 1
1 1 1 1
0 0 0 0
1 1 1 1
```

2. Floyds algorithm

```
#include<stdio.h>
void floyds(int [][100], int);
int min(int, int);
void main()
{
 int n;
 printf("Enter no. of vertices:\n");
 scanf("%d",&n);
 printf("Enter the cost adjacency matrix:\n");
 int a[100][100];
 for(int i=0; i<n; i++)
 {
```

```
for(int j=0;j<n;j++)
 {
  scanf("%d",&a[i][j]);
 }
}
printf("Distance Matrix is:\n");
floyds(a,n);
}

void floyds(int a[][100],int n)
{
 int D[n][n];
 int i,j,k;
 for(i=0;i<n;i++)
 {
  for(j=0;j<n;j++)
  {
   D[i][j]=a[i][j];
  }
 }
for(k=0;k<n;k++)
{
 for(i=0;i<n;i++)
 {
  for(j=0;j<n;j++)
  {
   D[i][j]=min(D[i][j],(D[i][k]+D[k][j]));
  }
 }
}
for(i=0;i<n;i++)
 {
 for(j=0;j<n;j++)
  {
  printf("%d\t",D[i][j]);
  }
 printf("\n");
 }
}
int min(int a,int b)
 {
  if(a<=b)
   return a;
  else
   return b;
 }
```

Output:

```
Enter no.of vertices:
4
Enter the cost adjacency matrix:
0 999 3 999
2 0 999 999
999 6 0 1
7 999 999 0
```

Distance matrix :

```
0   9   3   4
2   0   5   6
8   6   0   1
7   16  10  0
```

13/6/24

3. Johnson Tortter

```
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>

#define MAXN 20

int p[MAXN];
int pi[MAXN];
int dir[MAXN];

void swap(int *a, int *b)
 {
   int temp = *a;
   *a = *b.
   *b = temp;
 }

void printPermutation(int n)
{
 for(int i=0; i<n; i++)
 {
   printf("%d", p[i]);
}
printf("\n");
}

void printAllPermutations(int n)
 {
  for(int i=0, i<n ; i++)
  {
   p[i]= i+1;
   p[i] = i;
   dir[i]= -1;
  }

printPermutation(n);

int mobile, mobileIndex;
bool found;
while(1)
{
  mobile = -1;
  found = false;
  for(int i=0;i<n;i++)  // finding largest mobile element
  {
   int next = i+dir[i];
   if(next >= 0 && next<n && p[i]> p[next])
   {
     if(p[i]> mobile)
     {
       mobile = p[i];
       ~~p[i]=~~ mobileIndex = i;
     found = true; } } }
if(!found)
    break;

//Printing next permutation
int next = mobileIndex + dir[mobileIndex];
Swap(&p[mobileIndex], &p[next]);
Swap(&pi[p[mobileIndex]], &pi[p[next]]);
Swap(&dir[mobileIndex], &dir[next]);
printPermutation(n);

//Reverse direction of all elements larger than mobile elements
for(int i=0;i<n;i++)
  {
   if(p[i]>mobile)
    {
     dir[i]*=-1;
    }
  }
 }
}
}
int main()
{
  int n;
  printf("Enter the value of n(max 20):");
  Scanf("%d", &n);
  if(n>MAXN ||n<1)
  {
  printf("Invalid input.\n");
  return 1;
}
```

```
printAllPermutations(n);
return 0;
}
```

Output:

```
Enter the value of n(max 20):3
1 2 3
1 3 2
3 1 2
3 2 1
2 3 1
2 1 3
```

4. Knapsack Problem

```
#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 n;
    printf("Enter the no.of items:\n");
    scanf("%d", &n);
    int wt[100];
   int val[100];

   int W = 50;
   printf("Enter the weight of each item and
              its corresponding value:\n");
 for(int i=0;i<n;i++)
 {
    scanf("%d %d", &wt[i], &val[i]);
 }
 printf("Maximum value that can be obtained
              is %d \n", knapsack(W, wt, val,n));
 return 0;
}
```

Output:

```
Enter no.of items:
3
Enter the weight of each item and its corresponding
value:
10
60
20
100
30
120
Maximum value that can be obtained is 220
```

21/06/2024

1. Horspool algorithm

```
#include <stdio.h>
#include <stdlib.h>
#include < string h>

void shifttable (char [] , int []);
int horspool (char [], char[]);

int s[128];
int n,m;

int main()
{
 char t[100];
 char p[100];
 printf ("Enter the text string: \n");
 fgets(t, sizeof (t), stdin);
 t[strcspn (t, "\n")]='\0';

 printf ("Enter the pattern string: \n");
 fgets (t, sizeof (p), stdin);
 p[strcspn(p, "\n")]='\0';
int x=horspool(p, t);
if (x = = -1)
  printf ("Pattern not found in the text. \n");
else
  printf ("~~Pattern~~ The position where the pattern
          starts is: %d\n", x);
```

```
return 0;
}
int horspool (char p[], char t[])
{
  int i;
  shifttable (p, s);
   n=strlen (t);
  m=strlen ( p);
  i=m-1;
  int k;
 while (i<=n-1)
 {
  k=0;
  while (k<=m-1 && t[i-k]==p[m-1-k])
  {
     k=k+1;
  }
  if (k==m)
     return i-m+1;
 else
     i=i+s[t[i]];
 }
 return -1;
}
```

```
void shifttable(char p[], int s[])
{
   int i;
   m=strlen(p);
   for(i=0;i<=127;i++)
   {
     s[i]=m;
   }
   for(i=0;i<=m-2;i++)
   {
     s[p[i]]=m-1-i;
   }
}
```

Output:

```
Enter the text string:
jim saw me at a barber shop
Enter the pattern string:
barber
The position where the pattern starts is: 16
```

2. Heap Sort

```
#include <stdio.h>

void heapify(int [],int);

void main()
{
 int n;
 int a[50];
 printf("Enter the no. of elements: \n");
 scanf("%d", &n);
 printf("Enter array elements to heap sort: \n");
 for(int i=0; i<n; i++)
 {
  scanf("%d",&a[i]);
 }
 printf("The sorted array elements are: \n");
 heapify(a,n);
 for(int i=0;i<n;i++)
 {
   printf("%d \t",a[i]);
 }
}
```

```
   heapify (int a[], int n)
 {
   int k;
   int c;
   int key;
  int p;
  for (k=1; k<=n-1; k++)
  {
   key = a[k];
   c = k;
   p = (c-1)/2;
   while (c>0 && key>a[p])
   {
     a[c] = a[p];
     c = p;
     p = (c-1)/2;
   }
   a[c] = key;
 }
}
```

Output:
Enter the no of elements:
7
En

Enter array elements to heap sort:
50 25 30 75 100 45 80

The sorted elements are:
100 75 80 25 50 30 45

21/6/24

1. Prims algorithm

```
#include<stdio.h>
void prims(int [][50], int);
void main()
{
    int cost[50][50];
    int n;
    printf("Enter no.of vertices:\n");
    scanf("%d",&n);
    printf("Enter cost adjacency matrix:\n");
    for(int i=0;i<n;i++)
     {
      for(int j=0;j<n;j++)
       {
         scanf("%d",&cost[i][j]);
       }
    }
  printf("Result:\n");
  prims(cost,n);
}

void prims(int cost[][50],int n)
{
    int d[10];
    int p[10];
    int s[10];
    int min=999;
    int source=0;
int sum;
int k;
int u,v;
int t[10][10];

for(int i=0;i<n;i++)
 {
  for(int j=0;j<n;j++)
  {
   if(cost[i][j]!=0 && cost[i][j]<min)
    {
      min=cost[i][j];
      source=i;
    }

  }
}

for(int i=0;i<n;i++)
 {
  d[i]=cost[source][i];
  s[i]=0;
  p[i]=source;
  }

 s[source]=1;
 sum=0;
 k=0;
```

```
for(int i=1; i<n; i++)
 {
    u=-1;
    min=999;
    for(int j=0; j<n; j++)
    {
        if(s[j]==0)
         {
           if(d[j]<min)
           {
              min=d[j];
              u=j;
           }
        }
    }

t[k][0]=u;
t[k][1]=p[u];
k=k+1;
sum=sum+cost[u][p[u]];
s[u]=1;

for(int v=0; v<n; v++)
{
  if(s[v]==0 && cost[u][v]<d[v])
   {
   ~~if(d[v~~
   d[v]=cost[u][v];
   p[v]=u;
   }
}
}

printf("Shortest path cost : %d \n", sum);
printf("Minimum spanning tree vertices :\n");
for(int i=0; i<n; i++)
{
   printf("%d , %d \n", t[i][0], t[i][1]);

}
}
```

Output:

```
Enter no. of vertices:
4
Enter cost adjacency matrix:
0  1    5    2
1  0    9999 9999
5  9999 0    3
2  9999 3    0
```

Result:

```
Shortest path cost : 6
Minimum spanning tree vertices
```

```
3,0
2,3
0,0
```

2. Kruskal algorithm

```
#include <stdio.h>
int count=0;
int i, j, u, v, k=0, min, sum=0;
int t[10][10], cost[10][10];
int p[10], d[10];
void kruskal (int cost[10][10], int n);
int find (int i);
int union1 (int i, int j);

void main()
{
 int n;
 printf("Enter no. of vertices = ");
 scanf("%d", &n);
 printf("Enter the cost adjacency matrix:
        \n");
 for(~~int~~ i=0; i<n; i++)
   {
    for(j=0; j<n; j++)
     {
      scanf(" %d ", &cost[i][j]);
      if (cost[i][j]==0)
        {
         cost[i][j]=999;
        }
     }
   }
 kruskal (cost, n);
 printf("The edges in the minimum spanning
  tree are:\n");
 for(i=0; i<k; i++)
 {
  printf("(%d, %d)\n", t[i][0], t[i][1]);
 }
 printf("Minimum cost : %d \n", sum);
}

void kruskal (int cost[10][10], int n)
{
 for(int i=0; i<n; i++)
 {
  p[i]=i;
 }
 while(count<n-1)
 {
```

```
1,0
3,0
2,3
0,0
```

2. Kruskal algorithm

```
#include <stdio.h>
int count=0;
int i,j,u,v,k=0, min, sum=0;
int t[10][10], cost[10][10];
int p[10], d[10];
void kruskal(int cost[10][10], int n);
int find(int i);
int union1(int i, int j);

void main()
{
  int n;
  printf("Enter no.of vertices: ");
  scanf("%d", &n);
  printf("Enter the cost adjacency matrix:
        \n");
  for(~~int~~ i=0; i<n; i++)
    {
      for(j=0;j<n;j++)
        {
          scanf("%d", &cost[i][j]);
          if(cost[i][j]==0)
            {
              cost[i][j]=999;
            }
        }
    }

  kruskal(cost, n);
  printf("The edges in the minimum spanning
   tree are:\n");
  for(i=0; i<k; i++)
  {
   printf("(%d,%d)\n", t[i][0], t[i][1]
  }
  printf("Minimum cost: %d\n", sum);
}

void kruskal(int cost[10][10], int n)
{
  for(int i=0; i<n; i++)
  {
    p[i]=i;
  }
 while(count<n-1)
 {
```

```
for(int i=0;i<n;i++)
{
 for(int j=0;j<n;j++)
 {
   if(cost[i][j]<min)
   {
     min = cost[i][j];
     u=i;
     v=j;
  }
 }
}
if(find(u)!=find(v))
 {
   t[k][0]=u;
    t[k][1]=v;
    k++;
    count++;
    sum+=min;
    union1(u,v);
  }
 cost[u][v]=cost[v][u]=999;
}
}
```

```
int find(int i)
{
 while(p[i]!=i)
  {
    i=p[i];
  }
  return i;
}

void union1(int i, int j)
{
  int a=find(i);
  int b=find(j);
  p[a]=b;
 }
```

Output:

```
Enter the no of vertices : 4
Enter the cost adjacency matrix:
0  1  5  2
1  0  9999 9999
5  9999  0  3
2  9999  3  0

The edges in minimum spanning tree are:
(0,1)
(0,3)
(2,3)
```

3. Dijkstras algorithm

```
#include <stdio.h>
void dijkstras(int [][50], int);
void main()
{
    int cost[50][50];
    int n;
    printf("Enter no. of vertices :\n");
    scanf("%d", &n);
    printf("Enter cost adjacency matrix :\n");
    for(int i=0;i<n;i++)
    {
     for(int j=0;j<n;j++)
      {
        scanf("%d", &cost[i][j]);
      }
    }
  printf("Result :\n");
  dijkstras(cost, n);
}
```

```
void dijkstras(int a[][50], int n)
{
  int d[10];
  int visited[10];
  int p[10];
  int u,v;
  int s;
  int min;
  printf("Enter the source :\n");
  scanf("%d", &s);

  for(int i=0; i<n; i++)
  {
    d[i] = a[s][i];
    visited[i]=0;
    p[i]=s;
  }

visited[s]=1;
for(int i=0;i<n;i++)
{
  min = 999;
  u=0;
 for(int j=0;j<n;j++)
 {
```

```
if (visited [j] = = 0)
  {
    if (d [j] < min)
     {
       min = d [j];
       u = j;
     }
  }
}
visited [u] = 1;
for (int v=0; v<n; v++)
{
  if (visited [v] = = 0  &&  d [u] + a[u][v]
                              < d[v])
  {
    d [v] = d[u] + a[u][v];
    p[ v] = u;
  }
}
}
printf (" The shortest paths from vertex %d
          are : \n", s);
for (int i=0; i<n; i++)
{
```

```
if (i ! = s)
{
  printf ("To vertex %d : Distance = %d,
          Path = %d ", i, d[i], i );
  int j= i;
  while (p[j] ! = s)
  {
    j = p[j];
    printf (" <- %d ", j);
  }
  printf (" <- %d \n", s);
}
}
}
```

Output :

```
Enter no. of vertices :
4
~~Ent~~
Enter cost adjacency matrix :
0 1 5 2
1 0 9999 9999
5 9999 0 3
2 9999 3 0
```

Result :

```
Enter the source :
0
The shortest paths from vertex 0 are:
To vertex 1 : Distance = 1, Path = 1<— 0
To vertex 2 : Distance = 5, Path = 2 <— 0
To vertex 3 : Distance = 2, Path = 3<— 0.
```

4. Knapsack (Fractional : p/w ratio method)

```
# include<stdio.h>

void knapsack(int n, int p[], int w[],
                    int W)
~~{~~
{
   int used [n];
   for(int i=0;i<n ; i++)
      used [i] = 0;
  int cur-w = W;
  float tot-v = 0.0;
  int i, maxi;
  while (cur-w >0)
  {
maxi = -1;
for(i = 0; i<n; i++)
   if ((used [i]==0) && ((maxi == -1)||
   ((float) w[i] /p [i] > (float)w[maxi]/
                                p[maxi])))

   maxi = i;

   ~~used~~
used [maxi]= 1;
if (w[maxi]<=cur-w)
   {
     cur -w -= w [maxi];
     tot_v+= p[maxi];
     printf ("Added object %d (%d,
            %d) completely in the bag
         Space left: %d \n", maxi + 1,
         w[maxi], p[maxi], cur-w);

   }
else
 {

  int taken=cur-w;
  cur-w = 0;
 tot_v+=(float)taken/p[maxi]*p[maxi];
printf("Added %d %% (%d, %d) of
  object %d in the bag.\n", (int)((float)
  taken / w[maxi] * 100), w[maxi], p[maxi],
  maxi + 1);
}
}
printf("Filled the bag with objects worth
        %.2f.\n", tot_v);

}

int main()
{
  int n, W;
  printf("Enter the no. of objects: ");
  scanf("%d", &n);
  int p[n], w[n];
  printf("Enter the profits of the objects: ");
  for(int i=0; i<n; i++)
  {
    scanf("%d", &p[i]);
  }
printf("Enter the weights of the
        objects: ");
for(int i=0; i<n; i++)
{
  scanf("%d", &w[i]);
}
printf("Enter the maximum weight of
        the bag: ");
scanf("%d", &W);
knapsack(n, p, w, W);
return 0;
}
```

Output:

```
Enter the no. of objects: 7
Enter the profits of the objects: 5 10 15
7 8 9 4
Enter the weights of the objects: 1 3 5 4 1
3 2
Enter the maximum weight of the
bag: 15
Added object 4(4,7) completely in the
bag. Space left: 11
Added object 7 (2,4) completely in the
bag. Space left: 9
```

```
printf("Added %d%% (%d, %d) of
 object %d in the bag.\n", (int)(((float)
 taken / w[maxi] * 100), w[maxi], p[maxi],
 maxi + 1);
}
}

printf("Filled the bag with objects worth
        %.2f.\n", tot_v);

}

int main()
{
  int n, W;
  printf("Enter the no. of objects: ");
  scanf("%d", &n);
  int p[n], w[n];
  printf("Enter the profits of the objects: ");
  for(int i=0; i<n; i++)
  {
    scanf("%d", &p[i]);
  }
printf("Enter the weights of the
        objects: ");
for(int i=0; i<n; i++)
{
 scanf("%d", &w[i]);

}
printf("Enter the maximum weight of
        the bag: ");

scanf("%d", &W);
knapsack(n, p, w, W);
return 0;
}
```

Output:

```
Enter the no. of objects: 7
Enter the profits of the objects: 5 10 15
 7 8 9 4
Enter the weights of the objects: 1 3 5 4 1
3 2
Enter the maximum weight of the
bag: 15
Added object 4 (4, 7) completely in the
bag. Space left: 11
Added object 7 (2, 4) completely in the
bag. Space left: 9
```

Added object 3 (5,15) completely in the bag. Space left : 4.

Added object 6 (3,9) completely in the bag. Space left : 1.

Added 33% (3,10) of the object 2 in the bag.

Filled the bag with objects worth 36.00

5/7/24

12/07/2024

Q. Implementation of n-Queens :

```
#include <stdio.h>
#include <stdbool.h>
bool place (int [], int);
void printSolution (int [], int);
void nQueens (int);
int main()
{
 int n;
 printf ("Enter the no. of queens : ");
 scanf ("%d", &n);
 nQueens (n);
 return 0;
}
void nQueens (int n)
{
 int x[10];
 int count = 0;
 int k=1;
 while (k!=0)
 {
   x[k] = x[k] +1;
 }
 if (x[k] <=n)
 {
```

```
printSolution (x, n);
printf("Solution found\n");
count ++;
}
else
{
 k++;
 x[k]=0;
 }

}
else
{
 k--;
}
}
printf("Total Solutions: %d\n", count);
}
bool place (int x[10], int k)
{
 int i;
 for(i=1; i<k; i++)
 {
  if ((x[i]==x[k]) || (i-x[i]==k-x[k])
     || (i+x[i]==k+x[k]))
 {
  return false;
```

```
}
}
return true;

}
void printSolution (int x[10], int n)
{
 int i;
 for(i=1; i<=n; i++)
  {
   printf("%d", x[i]);
  }
 printf("\n");
}
```

Output:

```
Enter the number of queens : 4
2 4 1 3
Solution found
3 1 4 2
Solution found
Total solutions: 2
```

16/7/24