INDEX

Name Shreya Bharamanna Patil Sub. ADA

Std.: IV Div. E ~~Roll~~ No. 1BM23CS420

Telephone No. E-mail ID. shreyapatil.cs23@bmsce.ac.i

Blood Group. Birth Day.

Sr.No.	Title	Page No.	Sign./Remarks
1.	Linear, Binary Search, Bubble sort, Selection Sort.	01 - 06	10 Marks 3/5/24
2.	Tower of hanoi, GCD, lomuto partition, Topological Sort using DFS/ Source Removal method	07 - 13	10 Marks 7/6/24
3.	Merge Sort, Quick Sort	14 - 17	
4.	Warshall's algorithm, Floyd's algorithm, knapsack problem, Presorting	18 - 26	10 Marks 21/6/24
5.	Horspool algorithm, Heapify	27 - 30	
6.	Prim's, Kruskal's, Dijkstra's Fractional Knapsack Algorithm	31 - 39	10 Marks 5/7/24
7.	N- Queens	40 - 41	10 Marks 19/7/24

PAGE NO: 01
DATE: 03/05/24

```c
// C program to implement linear search
#include <stdio.h>
#include <stdlib.h>
void main (){
   int num, i, key;
   printf ("Enter the number of elements: ");
   scanf ("%d", &num);
   int arr[num];
   printf ("\nEnter all the elements:");
   for (i=0; i<num; i++){
      scanf ("%d", &arr[i]);
    }

   printf ("\nEnter the key to be searched: ");
   scanf ("%d", &key);

   for(i=0; i<num; i++){
     if (key== arr[i]){
        printf ("element at index %d", i);
        exit(0);
     }
   }
   printf("\nelement not found");
}
```

Output:

Case 1:

Enter the number of elements: 4
Enter all elements: 12 56 78 95
Enter the key to be searched: 56
element at index 1

Case 2:

Enter the number of elements : 4
Enter all the elements : 12 89 56 24
Enter the key to be searched : 100
element not found

```
// C program to implement binary search
# include <stdio.h>
int binarySearch(int arr[], int l, int r, int x)
 {
    while (l <= r) {
      int m = l+(r-l)/2;
      if (arr[m] == x)
          return m;
      if (arr[m] < x)
          l = m+1;
      else
          r = m-1;
    }
    return -1;
}

void main(){
   int num, i, j, temp, x;
   printf("Enter the number of elements: ");
   scanf("%d", &num);
   int arr[num];
   printf("\n Enter all the elements");
   for(i=0; i<num; i++){
      scanf("%d", &arr[i]);
   }
```

```
printf("\n Enter the key to be searched: ");
scanf("%d", &x);

for(i=0; i<n; i++){
    for(j=i+1; j<n; j++){
        temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
}

int n = sizeof(arr) / sizeof(arr[0]);
int result = binarySearch(arr, 0, n-1, x);
if (result == -1){
        printf("\nElement not found");
}else{
        printf("Element is at index %d", result);
    }
}
```

Output:

Case 1:

Enter the number of elements: 4
Enter all the elements: 23 45 78 95
Enter the key to be searched: 95
Element is at index 3

Case 2:

Enter the number of elements: 4
Enter all the elements: 23 45 78 95
Enter the key to be searched: 100
Element not found

```
// C program to implement bubble sort
#include<stdio.h>
void bubble_sort(int arr[], int n){
    int i, j;
    for(i=0; i<n-1; i++){
        for(j=0; j<n-i-1; j++){
            if(arr[j] > arr[j+1]){
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }
}

void main(){
int num, i;
printf("Enter the number of elements: ");
scanf("%d", &num);
int arr[num];
printf("\n Enter all the elements: ");
for(i=0; i<num; i++){
    scanf("%d", arr[i]);
}

bubble_sort(arr, num);
printf("\n Sorted array: ");
for(i=0; i<num; i++){
    printf("%d", arr[i]);
}
}
```

Output :

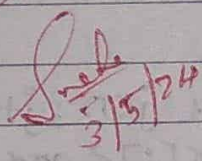
Enter number of elements: 5
Enter all the elements: 12 56 78 32 56
Sorted array: 12 32 56 56 78

```
// C program for selection sort
#include <stdio.h>
void selection (int arr[], int n)
 {
    int i, j, small;
    for (i=0; i<n-1; i++){
       small = i;
       for (j=i+1; j<n; j++){
          if (arr[j]< arr[small]{
                small = j;}
       }

   int temp = arr[small];
   arr[small] = arr[i];
   arr[i] = temp;
  }
}

void main(){
   int num, i;
   printf("Enter the number of elements: ");
   scanf("%d", &num);
   int arr[num];
   printf("\nEnter all the elements:");
   for (i=0; i<num; i++){
      scanf("%d", &arr[i]);
   }
   selection(arr, num);
```

```
    printf("\n Sorted array: ");
    for(i=0; i<num; i++){
        printf("%d ", arr[i]);
    }
}
```

Output:

Enter the number of elements : 5
Enter all the elements: 45 89 23 45 56
Sorted array : 23 45 45 56 89



// C program to implement recursive Tower Of Hanoi Algori

```
#include <stdio.h>
void toh(int, char, char, char);

int main(){
    int n;
    printf("Enter the number of disks: ");
    scanf("%d", &n);
    toh(n, 'A', 'B', 'T');
    return 0;
}

void toh(int n, char s, char d, char t){
    if (n == 1){
        printf("\nDisk %d moved from source %c to
          %c destination", n, s, d);
        return;
    }
    toh(n-1, s, t, d);
    printf("\nDisk %d moved from source %c to %c
        destination", n, s, d);
    toh(n-1, t, d, s);
}
```

Output:

Enter the number of disks: 2
Disk 1 moved from source A to T destination
Disk 2 moved from source A to B destination
Disk 1 moved from source T to B destination

// C program to implement GCD of two numbers

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

```
int gcd(int, int)

int main(){
   int a,b,ans;
   printf("Enter the values for integer a and b:");
   scanf("%d %d", &a, &b);
   ans = gcd(a, b);
   printf("\n GCD of %d and %d is %d", a,b,ans);
   return 0;
}

int gcd(int a, int b){
   if (b==0){
      return a;
   }
   return gcd(b, a%b);
}
```

Output:
Enter the value for integer a and b: 6 10
GCD of 6 and 10 is 2

```
//C program to implement lomuto partition.
# include <stdio.h>
void swap(int*, int*);
int lomutoPartition(int [], int, int);
int quickSelect(int [], int, int);

int main(){
   int a[10], l, r, n, i, res;
   printf("Enter the size:");
   scanf("%d",&n);
```

```
    printf("\n Enter the array elements: ");
    for (i = 0; i<n; i++){
        scanf("%d", &a[i]);
    }
    res = quickSelect(a, 0, n-1, n/2);
    printf("Median is %d", res);
    return 0;
}

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

int lomutoPartition(int a[], int l, int r){
    int p, s, i;
    p = a[l];
    s = l;
    for(i = l+1; i<r; i++){
        if(a[i]<p){
            s = s+1;
            swap(&a[s], &a[i]);
        }
    }
    swap(&a[s], &a[l]);
    return s;
}

int quickSelection(int a[], int l, int r, int k){
    int s = lomutoPartition(a, l, r);
    if(s == k){
        return a[s];
```

```
    }else if(s<k){
        quickselect(a,s+1,r,k);
    }else{
        quickselect(a,l,s-1,k);
    }
}
```

Output:

```
Enter the size: 9
Enter the array elements: 4 1 10 8 7 12 9 2 15
Median is 8
```

```
//C program to implement topological sort using DFS
#include <stdio.h>
int n, a[10][10], res[10], s[10], top=0;
void dfs(int, int, int[][10]);
void dfs_top(int, int[][10]);

int main(){
  printf("Enter the no. of nodes: ");
  scanf("%d", &n);
  int i,j;
  for(i=0;i<n;i++){
    for(j=0;j<n;j++){
      scanf("%d", &a[i][j]);
    }
  }
  dfs_top(n,a);
  printf("Solution: ");
  for(i=n-1; i>=0; i--){
    printf("%d ", res[i]);
  }
}
```

```
    return 0;
}

void dfs_top(int n, int a[][10]){
    int i;
    for(i=0; i<n; i++){
        s[i] = 0;
    }
    for(i=0; i<n; i++){
        if(s[i]==0){
            dfs(i,n,a);
        }
    }
}

void dfs(int j, int n, int a[][10]){
    s[j] = 1;
    int i;
    for(i=0; i<n; i++){
        if(a[j][i]==1 && s[i]==0){
            dfs(i,n,a);
        }
    }
    res[top++] = j;
}
```

Output:

```
Enter the no. of nodes: 5
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
Solution: 1 0 3 2 4
```

```
// C program to implement topological sort using source
  removal method
# include <stdio.h>
int a[10][10], n, t[10], indegree[10];
int stack[10], top=-1;

void computeIndegree(int, int[][10]);
void tps_SourceRemoval(int, int[][10]);

int main(){
   printf("Enter the no. of nodes:");
   scanf("%d",&n);
   int i,j;
   for(i=0;i<n;i++){
      for(j=0;j<n;j++){
            scanf("%d",&a[i][j]);
      }
   }
   computeIndegree(n,a);
   tps_SourceRemoval(n,a);
   printf("Solution:");
   for(i=0;i<n;i++){
      printf("%d",t[i]);
   }
   return 0;
}

void computeIndegree(int n, int a[][10]){
    int i,j,sum=0;
    for(i=0;i<n;i++){
        sum=0;
        for(j=0;j<n;j++){
            sum=sum+a[j][i];
    }
```

```
    indegree[i] = sum;
  }
}

void tps_SourceRemoval (int n, int a[][10]){
  int i,j,v;
  for(i=0; i<n; i++){
    if(indegree[i]==0){
      stack[++top] = i;
    }
  }
  int k=0;
  while(top!=-1){
    v = stack[top--];
    t[k++] = v;
    for(i=0; i<n; i++){
      if(a[v][i]!=0){
        indegree[i] = indegree[i] - 1;
        if(indegree[i]==0){
          stack[++top] = i;
        }
      }
    }
  }
}
```

Output:

```
Enter the no. of nodes: 5
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
Solution: 1 0 3 2 4
```

```
// C program to implement merge sort
#include <stdio.h>
int a[20], n;
void simple_sort(int [], int, int, int);
void merge_sort(int[], int, int);

int main(){
   int i;
   printf("Enter the no. of elements: ");
   scanf("%d", &n);
   printf("Enter the array elements: ");
   for (i=0; i<n; i++){
      scanf("%d", &a[i]);
   }
   merge_sort(a, 0, n-1);
   printf("Sorted array: ");
   for (i=0; i<n; i++){
      printf("%d ", a[i]);
   }
   return 0;
}

void merge_sort(int a[], int low, int high){
      if(low<high){
         int mid = (low+high)/2;
         merge_sort(a, low, mid);
         merge_sort(a, mid+1, high);
         simple_sort(a, low, mid, high);
      }
}

void simple_sort(int a[], int low, int mid, int high){
      int i=low, j=mid+1, k=low;
```

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

Output:

```
Enter the no. of elements: 5
Enter the array elements: 8 9 6 2 3
Sorted array: 2 3 6 8 9
```

```
// C program to implement quick sort
#include<stdio.h>
int a[20],n;
int partition(int[],int,int);
void quick_sort(int[],int,int);
void swap(int*,int*);
```

```
int main(){
    int i;
    printf("Enter the no. of elements: ");
    scanf("%d", &n);
    printf("Enter the array elements:");
    for(i=0; i<n; i++){
        scanf("%d", &a[i]);
    }
    quick_sort(a, 0, n-1);
    printf("Sorted array:");
    for(i=0; i<n; i++){
        printf("%d ", a[i]);
    }
    return 0;
}

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

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

int partition(int a[], int low, int high){
    int pivot = a[low];
    int i = low, j = high+1;
```

```
while(i<=j){
    do{
      i++;
    }while(a[i]<pivot && i<=high);
    do{
      j--;
    }while(a[j]>pivot && j>=low);
    if(i<j){
      swap(&a[i], &a[j]);
    }
  }
  swap(&a[low], &a[j]);
  return j;
}
```

Output :

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

7/6/24

// C program to implement warshall's algorithm

```
#include<stdio.h>
int a[10][10], P[10][10], n;
void warshall(int [][10], int);

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

void warshall(int a[][10], int n){
    int i,j,k;
    for(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;
            }
        }
    }
}
```

Output:

Enter the no. of vertices: 4

Enter the adjacent matrix:

0 1 0 0

0 0 0 1

0 0 0 0

1 0 1 0

Transitive Matrix/ Path matrix:

1 1 1 1

1 1 1 1

0 0 0 0

1 1 1 1

```
// C program to implement floyd's algorithm
#include<stdio.h>
int a[10][10], D[10][10], n;
void floyd(int [][10], int);
int min(int, int);
int main(){
   printf("Enter the no. of vertices: ");
   scanf("%d ", &n);
   printf("Enter the cost adjacency matrix: \n");
   int i, j;
   for(i=0; i<n; i++){
      for(j=0; j<n; j++){
         scanf("%d", &a[i][j]);
      }
   }
   floyd(a, n);
   printf("Distance Matrix: \n");
   for(i=0; i<n; i++){
      for(j=0; j<n; j++){
         printf("%d ", D[i][j]);
      }printf("\n");
   }
   return 0;
}

void floyd(int a[][10], int n){
   int for 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]));
        }
    }
}
}

int min(int a, int b){
    if(a<b){
        return a;
    }else{
        return b;
    }
}
```

Output:

```
Enter the no. of vertices: 4
Enter the cost adjacency matrix:
0   99   3   99
2    0   99  99
99   6    0   1
7   99   99   0
Distance Matrix:
0   9   3   4
2   0   5   6
8   6   0   1
7   16  10  0
```

```
// C program to implement ~~preposting~~ presorting
# include <stdio.h>
int a[20], n;
void simple_sort (int [], int, int, int);
void merge_sort (int[], int, int);
int presorting (int [], int);

int main() {
    int i, result;
    printf("Enter the no. of elements:");
    scanf("%d", &n);
    printf("Enter the array elements:");
    for(i=0; i<n; i++){
        scanf("%d", &a[i]);
    }
    result = presorting(a,n);
    if (result==1){
        printf("Unique array elements");
    }else{
        printf("Duplicate array elements");
    }
    return 0;
}

void merge_sort( int a[], int low, int high){
    if (low<high){
        int mid=(low+high)/2;
        merge_sort(a,low,mid);
        merge_sort(a, mid+1, high);
        simple_sort(a, low, mid, high);
    }
}
```

```
void simple_sort (int a[], int low, int mid, int high
  {
    int i=low, j= mid+1, k=low;
    int c[n];
    while (i<=mid && j<=high) {
        if(a[i]<a[j]) {
            c[k++]=a[i];
            i++;
        }else{
            c[k++]=a[j];
            j++;
        }
    }
    while (i<=mid){
        c[k++]=a[i];
        i++;
    }
    while(j<=high){
        c[k++]=a[j];
        j++;
    }
    for(i=low; i<=high; i++){
        a[i]=c[i];
    }
}
int presorting(int a[], int n){
    merge_sort(a,0,n-1);
    int i;
    for(i=0; i<n-2; i++){
        if(a[i]==a[i+1]){
            return 0;
        }
    } return 1;
}
```

Output:

Case 1:

Enter the no. of elements: 8

Enter the array elements: 25 03 96 78 65 10 65 25

Duplicate array elements.

Case 2:

Enter the no. of elements: 5

Enter the array elements: 12 56 98 74 32

Unique array elements

```
// C program to implement knapsack problem in dynamic
  programming
#include <stdio.h>
int n,m,w[10], p[10], v[10][10];
void knapsack(int, int, int[], int[]);
int max(int, int);
int main(){
  int i,j;
  printf("Enter the no. of items:");
  scanf("%d", &n);
  printf("Enter the capacity of knapsack:");
  scanf("%d", &m);
  printf("Enter weights:");
  for(i=0; i<n; i++){
    scanf("%d", &w[i]);
  }
  printf("Enter profits:");
  for(i=0;i<n;i++){
    scanf("%d", &p[i]);
  }
  knapsack(n,m,w,p);
  printf("Optimal Solution:\n");
  for(i=0; i<n; i++){
    for(j=0; j<n; j++){
      printf("%d ", v[i][j]);
    }
    printf("\n");
  }
  return 0;
}

void knapsack(int n, int m, int w[], int p[]){
    int i,j;
```

```
for(i=0; i<n; i++){
    for(j=0; j<m; j++){
        if (i==0 || j==0){
            v[i][j]=0;
        }else if (w[i]>j){
            v[i][j]= v[i-1][j];
        }else{
            v[i][j]= max(v[i-1][j], ((v[i-1][j-w[i]]
                                    + p[i]));
        }
    }
}
}

int max(int a, int b){
    if(a>b){
        return a;
    }else{
        return b;
    }
}
```

Output:

Enter the no. of items: 4
Enter the capacity of knapsack: 5
Enter weights: 2 1 3 2
Enter profits: 12 10 20 15
Optimal Solution:

0	0	0	0
0	10	10	10
0	10	10	20
0	10	15	25

```
// C program to implement horspool algorithm.
#include <stdio.h>
#include <string.h>
char P[100], T[100];
int S[200];

void shiftTable(char[], int[]);
int horspod(char[], char[]);

int main(){
   printf("Enter the text string: ");
   scanf("%s", T);
   printf("Enter the pattern string: ");
   scanf("%s", P);
   int result = horspool(T, P);
   if(result == -1){
        printf("\nPattern not found\n");
   }else{
        printf("Pattern found at position %d\n",
            result);
   }
   return 0;
}
void shiftTable(char P[], int S[]){
   int m = strlen(P);
   int i;
   for(i=0; i<128; i++){
      S[i] = m;
   }
   for(i=0; i<m-1; i++){
        S[(int)P[i]] = m-1-i;
   }
}
```

```
int horspool(char T[], char P[]){
    shiftTable(P, S);
    int n = strlen(T);
    int m = strlen(P);
    int i = m-1;
    while(i<=n-1){
        int k=0;
        while(k<=m-1 && T[i-k]==P[m-1-k]){
            k++;
        }
        if(k==m){
            return i-m+1;
        }
        i=i+S[(int)T[i]];
    }
    return -1;
}
```

Output:

Enter the text string: jim saw me in barber shop
Enter the pattern string: barber
Pattern found at position 14

Enter the text string: jim saw me in barber shop
Enter the pattern string: cake
Pattern not found.

```
// C program to implement heapify
#include <stdio.h>
int a[10], n;
void heapify (int [], int);
int main(){
    printf("Enter the number of array elements: ");
    scanf("%d", &n);
    int i;
    printf("Enter array elements: ");
    for(i=0; i<n; i++){
        scanf("%d", &a[i]);
    }
    heapify(a,n);
    printf("Array elements: ");
    for(i=0; i<n; i++){
        printf("%d", a[i]);
    }
    return 0;
}

void heapify (int a[], int n){
    int k;
    for(k=1; k<n; k++){
        int key = a[k];
        int c = k;
        int 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 number of array elements : 7

Enter array elements : 50 25 30 75 100 45 80

Array elements : 100 75 80 25 50 30 45

Sneha
21761224

```
// C program to implement Prim's algorithm.
#include <stdio.h>
int cost[10][10], n, t[10][2], sum;
void prims (int cost[10][10], int n);

int main(){
   int i,j;
   printf("Enter the number of vertices: ");
   scanf("%d", &n);
   printf("Enter the cost adjacency matrix: \n");
   for (i=0; i<n; i++){
      for(j= 0; j<n; j++){
         scanf("%d", &cost[i][j]);
      }
   }
   prims (cost, n);
   printf("Edges of the minimal spanning tree: \n");
   for(i=0; i<n-1; i++){
      printf("(%d, %d) ", t[i][0], t[i][1]);
   }
   printf("\n Sum of minimal spanning tree: %d\n",
      sum);
   return 0;
}
void prims (int cost[10][10], int n){
   int i,j,u,v;
   int min, source;
   int p[10], d[10], s[10];
   min = 999;
   source = 0;
   for(i = 0; i<n; i++){
      d[i] = cost[source][i];
      s[i] = 0;
```

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

     if(u!=-1){
       t[k][0]=u;
       t[k][1]=p[u];
       k++;
       sum += cost[u][p[u]];
       s[u]=1;

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

Output:

```
Enter the number of vertices: 4
Enter the cost adjacency matrix:
0  1  5  2
1  99 99 99
5  99 0  3
2  99 3  0
Edges of the minimal spanning tree:
(0,1) (0,3) (2,3)
Sum of minimal spanning tree: 6
```

```
// C program to implement kruskal's algorithm
#include<stdio.h>
int cost[10][10], n, t[10][2], sum;
void kruskal (int cost[10][10], int n);
int find (int parent[10], int i);
int main(){
   int i,j;
   printf("Enter the number of vertices:");
   for(i=0;i<n;i++){
      for(j=0;j<n;j++){
         scanf("%d", &cost[i][j]);
      }
   }
   kruskal(cost,n);
   printf("Edges of the minimal spanning tree:\n");
   for(i=0;i<n-1;i++){
      printf("(%d,%d)", t[i][0], t[i][1]);
   }
   printf("\nSum of minimal spanning tree: %d\n",
         sum);
   return 0;
}
```

```
void kruskal(int cost[10][10], int n){
  int min, u, v, count, k;
  int parent[10];
  k=0;
  sum=0;
  for(int i=0; i<n; i++){
    parent[i]=i;
  }
  count=0;
  while(count<n-1){
    min = 999;
    u=-1;
    v=-1;
    for(int i=0; i<n; i++){
      for(int j=0; j<n; j++){
        if(find(parent,i) != find(parent,j) &&
          cost[i][j] < min){
            min = cost[i][j];
            u=i;
            v=j;
          }
      }
    }
    int root_u = find(parent, u);
    int root_v = find(parent, v);
    if(root_u != root_v){
      parent[root_u] = root_v;
      t[k][0]=u;
      t[k][1]=v;
      sum+=min;
      k++;
      count++;
    }
  }
```

```
int find(int parent[10], int i){
    while (parent[i] != i){
        i = parent[i];
    }
    return i;
}
```

Output:

Enter the number of vertices: 4
Enter the cost adjacency matrix:

0	1	5	2
1	0	99	99
5	99	0	3
2	99	3	0

Edges of the minimal spanning tree:
(0,1) (0,3) (2,3)
Sum of minimal spanning tree: 6

```
// C program to implement Dijkstra's algorithm
#include <stdio.h>
int cost[10][10], n, result[10][2], weight[10];
void dijkstras(int [][10], int);
int main(){
    int i, j, s;
    printf("Enter the number of vertices:");
    scanf("%d", &n);
    printf("Enter the cost adjacency matrix:\n");
    for(i = 0; i<n; i++){
        for(j = 0; j<n; j++){
            scanf("%d", &cost[i][j]);
        }
    }
```

```
    printf("Enter the source vertex:");
    scanf("%d",&s);
    dijkstras(cost,s);
    printf("Path:\n");
    for(i=1;i<n;i++){
        printf("(%d,%d) with weight %d", result[i]
            result[i][1], weight[result[i][1]]);
    }
    return 0;
}

void dijkstras(int cost[][10], int s){
    int d[10], p[10], visited[10];
    int i,j, min, u,v,k;
    for(i=0;i<10;i++){
        d[i]=999;
        visited[i]=0;
        p[i]=s;
    }
    d[s]=0;
    visited[s]=1;
    for(i=0;i<n;i++){
        min=999;
        u=0;
        for(j=0;j<n;j++){
            if(visited[j]==0){
                if(d[j]<min){
                    min=d[j];
                    u=j;
                }
            }
        }
    }
    visited[u]=1;
```

```
        for (v=0; v<n; v++) {
            if(visited[v]==0 && (d[u]+cost[u][v]<d[v])
              ){
              d[v]= d[u]+cost[u][v];
              p[v]=u;
            }
        }
    }
    for (i=0; i<n; i++) {
        result[i][0] = p[i];
        result[i][1] = i;
        weight[i] = d[i];
    }
}
```

Output:

```
Enter the number of vertices: 4
Enter the cost adjacency matrix:
0  1  5  2
1  0  99 99
5  99 0  3
2  99 3  0
Enter the source vertex: 0
Path:
(0,1) with weight 1
(0,2) with weight 5
(0,3) with weight 2
```

```
// C program to implement fractional knapsack problem
#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[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 %2.f
```

```
    \n", tot_v);
}
int main(){
    int n, W;
    printf("Enter the number 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 number 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
object 7 (2, 4). Space left: 9
object 3 (5, 15). Space left: 4
object 6 (3, 9). Space left: 1
33% (3, 10) of object 2 in the bag
Filled the bag with objects worth 36.00.
```

5/7/24

8 //C program to implement "N-Queens-Problem"

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

bool place(int[], int);
void printSolution(int[], int);
void nQueens(int);

int main(){
  int n;
  printf("Enter the number 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;
     while(x[k]<=n && !place(x,k)){
        x[k]=x[k]+1;
     }
     if(x[k]<=n){
        if(k==n){
           printSolution(x,n);
           printf("Solution found");
           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
```

Sneha 19/7/24