INDEX

4E

ADA Lab

1BM22CS284

NAME: Sneha Prasanna STD.: SEC.: ROLL NO.: SUB.:

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
1	3/5/24	Linear Search	1-2	10 Sneha 3/5/24
2	3/5/24	Binary Search	3-4	
3.	3/5/24	Bubble Sort	5-6	
4.	3/5/24	Selection Sort	7-8	
5.	7/6/2024	Merge Sort	11-12	10 Sneha 7/6/24
6.	7/6/2024	Selection Sort	12-13.	
7	30/5/24	Topological Sort, Tower of Hanoi (Fibonacci) GCD, Computing Mean	9-10	
8	13/6/24	Warshall, Floyd Algorithm	11-13	10 Sneha 13/6/24
9	13/6/2024	Johnson Trotter Algorithm	14-16	10 Sneha 20/6/24
10	13/6/2024	knapsack Algorithm	16-18	
11	21/6/2024	Horspool Algorithm, Heap Sort Algorithm	18-20.	10 Sneha 21/6/24
12	5/7/24	knapsack, Kruskals Prims, Dijkstras Algorithm	21-26	10 Sneha 5/7/24
13	12/7/24	N-Queens Algo	27-28	10 Sneha 12/7/24

3/05/2024 Friday

ADA Lab.

Sneha Prasanna
1BM22CS284
Batch-4
ADA Lab-

1. Linear Search.

```
#include <stdio.h>
#include <stdlib.h>
linear_search(int, int[], int);
void main(){
    int i;
    int n;
    int key;
    printf("Enter the number of elements\n");
    scanf("%d", &n);
    printf("Enter the elements\n");
    for(i=0; i<=n; i++){
        scanf("%d", &a[i]);
linearsearch(n,a,key);
    printf("\nEnter search element\n");
    scanf("%d", &key);
}
void linear_search(int n, int a[], int key){
for(int i=0; i<=n; i++){
        if(a[i]==key){
            printf("Element found at %d", i+1);
            printf("Successful search");
        }
        else{
            printf("Element not found\n");
            printf("Unsuccessful search\n");
        }}
```

Output

Case 1 - Enter the value of n

5

Enter the ~~search value~~ elements

4 5 9 9 0

Enter search value

0

0 found at ~~pound~~ position : 5

Case 2 :- Enter the value of n

4

Enter the elements

1 2 3 4

Enter search key

6

Element not found.

2. Binary Search

```
#include <stdio.h>
#include <stdlib.h>
void binary_search(int, int, int[], int);
void main(){
    int i;
    int n, a[n];
    printf("Enter the number of elements\n");
    scanf("%d", &n);
    int l;
    int m, key;
    printf("Enter the value of l\n");
    scanf("%d", &l);
    printf("Enter the value of m\n");
    scanf("%d", &m);
    for(i=0; i<n; i++){
    printf("Enter the sorted elements\n");
    for(i=0; i<=n; i++){
        scanf("%d", &a[i]); }
    binary_search(n, l, a, m, key);
    printf("Entee the key to be searched\n");
    scanf("%d", &key);
}
void binary_search(int n, int l, int a[], int m, int key){
    int l = 0;
    int m = n-1;
    int mid = (l+m)/2;
    int pos = -1;
    while(l<=m){
        mid = (l+m)/2;
        if(a[mid]==key){
            pos = mid+1;
            printf("%d found at pos: %d",
                key, pos);
            exit(0);
        }
```

```
        else if (key > a[mid]){
                l = mid + 1;}
                else if (key < a[mid]){
                m = mid - 1;
                }
        printf("%d not found.\n", key);
        }
    }
  }
}
```

Output

Case 1:- Enter the number of elements
4
Enter the value of l
0
Enter the value of m
2
Enter the sorted elements
2 3 4 5
Enter the key to be searched
3
Element found at pos 2

Case 2:- Enter the number of elements
4
Enter the value of l
0
Enter the value of m
2
Enter the sorted elements
1 2 3 5
Enter the key to be searched
0
Element not found

3. Bubble Sort

```
#include <stdio.h>
#
void bubble_sort(int n, int a[]);
void main(){ int n, i;
    printf("Enter the value of n\n");
    scanf("%d", &n);
    int a[n];
    printf("Enter the elements\n");
    for(i=0; i<=n; i++){
        scanf("%d", &a[i]);
    }
    bubble_sort(n, a);
}
void bubble_sort(int n, int a[]){
    printf("sorted elements");
    for(i=0; i<n; i++){
        printf("%d", array[i]);
    }
    printf("\n");
    return 0;
}
void bubble_sort(int n, int a[]){
    int i, j, temp;
    for(i=0; i<n-1; i++){
        for(j=0; j<n-i-1; j++){
            if(a[j] > a[j+1]){
                temp = a[j];
                a[j] = a[j+1];
                a[j+1] = temp;
```

Output

```
Enter the number of elements: 3
Enter 3 elements:
1 2 0
original array: 1 2 0
Sorted array: 0 1 2
```

---

9. Selection Sort

```
#include<stdio.h>
void SelectionSort(int ~~x~~, int a[]);
void main(){
    int i;
    int n;
    printf("Enter the number of elements\n");
    scanf("%d", &n);
    int a[n];
    printf("Enter the elements of the array\n");
    for(~~int~~ i=0; i<n; i++){
        scanf("%d", &a[i]);
    SelectionSort(n, a);
    }
}
void SelectionSort(int n, int a[]){
    for(int i=0; i<n-1; i++){
        int min=i;
        for(int j=i+1; j<n; j++){
            if(a[j]<a[min])
                min=j;
        }
        int temp=a[i];
        a[i]=a[min];
        a[min]=temp;
    }
}
```

```
printf("Enter the sorted array\n");
for(int i=0; i<n; i++)
    printf("%d", a[i]);
```

Output

Enter the number of elements: 5
Enter the elements in the array:
4 6 5 2 1
Sorted array: 1 2 4 5 6

3/5/24

5 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",
        (float) time_req / CLOCKS_PER_SEC);
}
```

For linear search:
Processor time taken: 0.000170 seconds

30/05/2024

Lab - 02

① Topological sort → dfs method.

Output

```
Enter the number of nodes
5
Enter the 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
Enter the cost matrix:
3 1 2 2 1
4 5 1 2 1
3 1 2 3 1
3 2 2 1 1
3 4 5 1 2
Topologically sorted array
1 0 3 2 4
```

② Tower of Hanoi

Output

```
Enter the numbers of disc: 4
Moves required for Tower of Hanoi with 4 discs:
Move 1 disc from A to B
Move disc from A to C
Move 1 disc from B to C
Move disc from A to B
Move 1 disc from C to A
Move disc from C to B
Move 1 disc from A to B
Move disc from A to C
Move 1 disc from B to C
Move disc from B to A
Move 1 disc from C to A
Move disc from B to C
Move 1 disc from A to B
Move disc from A to C
Move 1 disc from B to C
```

3. GCD of two numbers — Recursive

Output

Enter the value of a

3

Enter the value of b

6

Result : 3

4. Topological ~~sort~~ sort → Source removal method

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

Topological sorted array is

1 4 0 3 2

5. ~~Insertion~~ Computing median selection problem.

Enter no. of elements:

9

Enter array elements:

11 10 8 7 1 2 9 2 15

Enter the value of k:

4

Result : 7

7/6/2024

Lab - 8

Merge Sort

```
#include <stdio.h>
void simplemerge(int [], int, int, int);
void mergesort(int[], int, int);
int c[100];
void main(){
int n, low, high;
int a[];
int mid;
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=0, 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 the no. of elements
8
Enter the array values
8 6 2 4 3 7 5
Sorted elements are:
1 2 3 4 5 6 7 8
```

7/6/24

Quick Sort.

```
# include <stdio.h>
int partition(int[], int, int);
void quicksort(int[], int, int);
void main(){
  int n, low, high;
  int a[];
  int mid;
  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 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;
    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 the no. of elements:
5
Enter the unsorted array values:
7 6 9 4 2
Sorted elements are:
2 4 6 7 9

Sneha
7/6/24

18/06/2024

1. Warshall's Algorithm

Code:-

```
#include <stdio.h>
void Warshall(int [][50], int);
int main(){
    int n;
    int a[50][50];
    printf("Enter the number of vertices of the adjacency matrix\n");
    scanf("%d",&n);
    printf("Enter the adjacency matrix values:\n");
    for(int i=0; i<n; i++){
        for(int j=0; j<n; j++){
            scanf("%d",&a[i][j]);
        }
    }
    printf("\n The path matrix is:\n");
    Warshall(a,n);
    return 0;
}
void warshall(int a[][50], int n){
    int p[50][50];
    for(int i=0; i<=n; i++){
        for(int j=0; j<=n; j++)
            p[i][j]= a[i][j];
    }
    for(int k=0; k<=n; k++){
        for(int i=0; i<=n-1; i++){
            for(int j=0; j<=n-1; j++){
                if(p[i][j]==0 && p[i][k]==0 && p[k][j]==1){
                    p[i][j]=1;
                }
            }
        }
    }
    for(int i=0; i<=n-1; i++){
        for(int j=0; j<=n-1; j++){
            printf("%d\t",p[i][j]);
        }
        printf("\n");
    }
}
```

Output

```
Enter the number of ver
4
Enter the adjacency matrix
0 1 0 0
0 0 0 1
0 0 0 0
1 0 1 0
The path matrix is:
1 1 1 1
1 1 1 1
0 0 0 0
1 1 1 1
```

2. Floyd's Algorithm -

```
#include <stdio.h>
void floyds(int [][50
int main(){
    int n;
    int a[50][50];
    printf("Enter t
    scanf("%d
    printf("Enter
        for(int
            for(int
```

13/06/2024

1. Warshall's Algorithm

Code:-

```
#include <stdio.h>
void Warshall(int [][50], int);
int main(){
    int n;
    int a[50][50];
    printf("Enter the number of vertices of the adjacency matrix:\n");
    scanf("%d",&n);
    printf("Enter the adjacency matrix values:\n");
    for(int i=0; i<n; i++){
        for(int j=0; j<n; j++){
            scanf("%d", &a[i][j]);
        }
    }
    printf("The path matrix is:\n");
    Warshall(a,n);
    return 0;
}
void warshall(int a[][50], int n){
    int p[50][50];
    for(int i=0; i<=n; i++){
        for(int j=0; j<=n; j++)
            p[i][j]= a[i][j];
    }
    for(int k=0; k<=n; k++){
        for(int i=0; i<=n-1; i++){
            for(int j=0; j<=n-1; j++){
                if(p[i][j]==0 && p[i][k]==0 && p[k][j]==1){
                    p[i][j]==1;
                }
            }
        }
    }
```

```
for(int i=0; i<=n-1; i++){
    for(int j=0; j<=n-1; j++){
        printf("%d\t", p[i][j]);
    }
    printf("\n");
}
```

Output

Enter the number of vertices for the adjacency matrix
4
Enter the adjacency matrix values:
0 1 0 0
0 0 0 1
0 0 0 0
1 0 1 0
The path matrix is:
1 1 1 1
1 1 1 1
0 0 0 0
1 1 1 1

2. Floyd's Algorithm -

```
#include <stdio.h>
void floyds(int [][50], int);
int main(){
    int n;
    int a[50][50];
    printf("Enter the number of vertices of the adjacency matrix:\n");
    scanf("%d", &n);
    printf("Enter the adjacency matrix values:\n");
    for(int i=0; i<n; i++){
        for(int j=0; j<n; j++){
            scanf("%d", &a[i][j]);
        }
    }
}
```

```
    printf("The Distance matrix is:\n");
    Floyds(a,n);
    return 0;
}

void Floyds(int a[][50], int n){
    int D[50][50];
    for(int i=0; i<n; i++){
        for(int j=0; j<n; j++){
            D[i,j] = a[i][j];
        }
    }
for(int k=0; k<n; k++){
    for(int i=0; i<n; i++){
        for(int j=0; j<n; j++){
            if(D[i][k] + D[k][j] < D[i][j]){
                D[i][j] = D[i][k] + D[k][j];
            }
        }
    }
for(int i=0; i<n; i++){
    for(int j=0; j<n; j++){
        printf("%d", D[i][j]);
    }
    printf("\n");
}
}
```

Output

Enter the number of vertices for the adjacency matrix
4
Enter the adjacency matrix values:
0 9999 9 9999
2 0 9999 9999
9999 6 0 1
7 9999 9999 0

The distance matrix is:
0 9 3 4
2 0 5 6
8 6 0 1
7 16 10 0

13/6/24

3. ~~ProSorting Algorithm~~
Johnson Trotter Algorithm.

Code

```
#include <stdio.h>
#include<stdlib.h>
#include<stdbool.h>
void printPermutation(int *a, int n){
    for(int i=0; i<n; i++){
        printf("%d", a[i]);
    }
    printf("\n");
}
int factorial(int n){
    int fact = 1;
    for(int i=1; i<=n; i++){
        fact *= i;
    }
    return fact;
}
int searchMobileIndex(int *a, bool *dir, int n){
    int mobileIndex = -1;
    int mobileValue = 0;
```

Output

Enter the number of vertices for the adjacency matrix.
4
Enter the adjacency matrix values:
0 9999 3 9999
2 0 9999 9999
9999 6 0 1
7 9999 9999 0

The distance matrix is :
0 9 3 4
2 0 5 6
8 6 0 1
7 16 10 0

13/6/24

3. ~~Prefixing Algorithm~~
Johnson Trotter Algorithm.

Code

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
void printPermutation(int *a, int n){
    for(int i=0; i<n; i++){
        printf("%d", a[i]);
    }
    printf("\n");
}
int factorial(int n){
    int fact = 1;
    for(int i=1; i<=n; i++){
        fact *= i;
    }
    return fact;
}
int searchMobileIndex(int *a, bool *dir, int n){
    int mobileIndex = -1;
    int mobileValue = 0;
```

```
for (int i = 0; i < n; i++){
    if (dir[a[i]-1] == 0 && i != 0 && a[i] < a[i-1]){
        if (a[i] > mobileValue){
            mobileValue = a[i];
            mobileIndex = i;
        }
    }
    if (dir[a[i]-1] == 1 && i != n-1 && a[i] > a[i+1]){
        if (a[i] > mobileValue){
            mobileValue = a[i];
            mobileIndex = i;
        }
    }
}
return mobileIndex;
}
void swap(int *a, int *b){
    int temp = *a;
    *a = *b;
    *b = temp;
}
void generatePermutations(int n){
    int *a = (int*)malloc(n*sizeof(int));
    bool *dir = (bool*)malloc(n*sizeof(bool));
    for (int i = 0; i < n; i++){
        a[i] = i+1;
        dir[i] = 0;
    }
    printPermutation(a, n);
    for (int i = 1; i < factorial(n); i++){
        int mobileIndex = searchMobileIndex(a, dir, n);
        if (mobileIndex == -1){
            break;
        }
        if (dir[a[mobileIndex]-1] == 0){
            swap(&a[mobileIndex], &a[mobileIndex-1]);
            mobileIndex--;
        }
```

```
else {
    swap(&a[mobileIndex], &a[mobileIndex+1]);
    mobileIndex++;
}
}
for(int j=0; j<n; j++){
    if (a[j] > a[mobileIndex]){
        dir[a[j]-1] = !dir[a[j]-1];
    }
}
print permutation(a, n);
}
free(a);
free(dir);
}
int main(){
    int n;
    printf("Enter the number of elements:");
    scanf("%d", &n);
    generatePermutations(n);
    return 0;
}
```

Output

```
Enter the number of elements 4
1 2 3 4
1 2 4 3
1 4 2 3
4 1 2 3
4 1 3 2
1 4 3 2
1 3 4 2
1 3 4 2
3 1 2 4
3 1 4 2
3 4 1 2
4 3 1 2
4 3 2 1
```

3 4 2 1

3 2 4 1

3 2 1 4

2 3 1 4

2 9 4 1

2 4 3 1

4 2 3 1

4 2 1 3

2 4 1 3

2 1 4 3

2 1 3 4

---

4. Knapsack Algorithm.

```
#include<stdio.h>
void knapsack(int n, int m, int w[], int p[], int v[][50]){
    for(int i=0; i<=n; i++){
        for(int j=0; j<=m; j++){
            if(i==0||j==0){
                v[i][j]=0;
            }
            else if(w[i-1]>j){
                v[i][j]=v[i-1][j];
            }
            else{
                v[i][j]=(p[i-1]+v[i-1][j-w[i-1]]>v[i-1][j])?p[i-1]+v[i-1][j-w[i-1]]:v[i-1][j];
            }
        }
    }
    printf("the optimal solution is %d\n", v[n][m]);
}
```

```
int main(){
    int n;
    printf("Enter the number of items or objects\n");
    scanf("%d",&n);
    int M;
    printf("Enter the capacity of the knapsack\n");
    scanf("%d", &M);
    int p[50];
    printf("Enter the profit of all objects\n");
    for(int i=0; i<n; i++)
        scanf("%d", &p[i]);
    int W[50];
    printf("Enter the weight of all objects\n");
    for(int i=0; i<n; i++)
        scanf("%d", &W[i]);
    int v[50][50];
    knapsack(n, M, W, P, V);
    return 0;
}
```

Output

```
Enter the number of items or objects
4
Enter the capacity of the knapsack
5
Enter the profit of all objects
12 10 20 15
Enter the weight of all objects
2 1 3 2
The optimal solution is 37.
```

20/6/24

```
int main(){
    int n;
    printf("Enter the number of items or objects\n");
    scanf("%d",&n);
    int M;
    printf("Enter the capacity of the knapsack\n");
    scanf("%d", &M);
    int p[50];
    printf("Enter the profit of all objects\n");
    for(int i=0; i<n; i++)
        scanf("%d", &p[i]);
    int w[50];
    printf("Enter the weight of all objects\n");
    for(int i=0; i<n; i++)
        scanf("%d", &w[i]);

    int v[50][50];
    knapsack(n, M, w, p, v);
    return 0;
}
```

Output

```
Enter the number of items or objects
4
Enter the capacity of the knapsack
5
Enter the profit of all objects
12  10  20  15
Enter the weight of all objects
2  1  3  2
The optimal solution is 37.
```

20/6/24

21/06/2024

1. Horspool algorithm

```
#include <stdio.h>
#include <string.h>
void ShiftTable(char p[], int s[]){
    int m = strlen(p);
    for(int i=0; i<127; i++){
        s[i] = m;
    }
    for(int j=0; j<m-1; j++){
        s[(unsigned char)p[j]] = m-1-j;
    }
}
int Horspool(char p[], char t[]){
    int s[127];
    ShiftTable(p, s);
    int m = strlen(p);
    int n = strlen(t);
    int i = m-1;
    while(i<n){
        int k=0;
        while(k<m && p[m-1-k] == t[i-k]){
            k++;
        }
        if(k==m){
            return i-m+1;
        }else{
            i += s[(unsigned char) t[i]];
        }
    }
    return -1;
}
```

```
int main(){
    char p[50];
    char t[200];
    printf("Enter the p
    scanf("%s", p);
    printf("Enter the text
    scanf("%s", t);
    int position = Horsp
    if(position == -1){
        printf("Pattern
    }else{
        printf("Pattern
    }
    return 0;
}
```

Output

```
Enter the pattern string:
barber
Enter the text string:
jim_saw_me_in_a_
Pattern found at posi
```

---

2. HeapSort

```
#include<stdio.h>
void Heapify(int a[],
    for(int k=1; k<
        int key = a[k
        int c = k;
        int p = (c-1)
        while(c>0
            a[c]
            c = p
            p = (
        }
        a[c] = key;
    }
}
```

21/06/2024

1. Horspool algorithm

```
#include <stdio.h>
#include <string.h>
void ShiftTable(char p[], int s[]){
    int m = strlen(p);
    for(int i=0; i<127; i++){
        s[i] = m;
    }
    for(int j=0; j<m-1; j++){
        s[(unsigned char)p[j]] = m-1-j;
    }
}
int Horspool(char p[], char t[]){
    int s[127];
    ShiftTable(p, s);
    int m = strlen(p);
    int n = strlen(t);
    int i = m-1;
    while(i<n){
        int k=0;
        while(k<m && p[m-1-k] == t[i-k]){
            k++;
        }
        if(k==m){
            return i-m+1;
        }else{
            i += s[(unsigned char)t[i]];
        }
    }
    return -1;
}
```

```
int main()
{   char p[50];
    char t[100];
    printf("Enter the pattern string\n");
    scanf("%s", p);
    printf("Enter the text string:\n");
    scanf("%s", t);
    int position = Horspool(p, t);
    if(position == -1){
        printf("pattern not found in the text.\n");
    }else{
        printf("pattern found at position %d\n", position);
    }
    return 0;
}
```

Output

```
Enter the pattern string:
barber
Enter the text string:
jim_saw_me_in_a_barber_shop
Pattern found at position 16.
```

---

2. HeapSort

```
#include<stdio.h>
void Heapify(int a[], int n){
    for(int 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;
    }
}
```

```
int main(){
    int a[50];
    int n;
    printf("Enter the number of elements\n");
    scanf("%d", &n);
    printf("Enter the elements:\n");
    for(int i=0; i<n; i++){
        scanf("%d", &a[i]);
    }
    Heapify(a, n);
    printf("Heapified array:\n");
    for(int i=0; i<n; i++){
        printf("%d", a[i]);
    }
    printf("\n");
    return 0;
}
```

Output

Enter the number of elements: 7
Enter the elements:
50 25 80 75 100 45 80
Heapified array:
100 75 80 25 50 80 45

21/6/24

5/07/2024 Friday

1) Kruskal Algorithm

```
#include <stdio.h>
int i, j, sum, u, v, min = 999, count = 0, k = 0;
int t[20][20], p[20], d[20];
void kruskal(int cost[40][40], int n){
    for(int i=0; i<n; i++){
        p[i] = i;
    }
    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(cost[i][j] < min){
                    min = cost[i][j];
                    u = i;
                    v = j;
                }
            }
        }
        while(p[u] != u){
            u = p[u];
        }
        while(p[v] != v){
            v = p[v];
        }
        if(u != v){
            t[k][0] = u;
            t[k][1] = v;
            k++;
            count++;
            sum += min;
            p[v] = u;
        }
```

```
        cost[u][v] = cost[v][u] = 999;
    }
  }
}
int main(){
    int n;
    int cost[10][10];
    printf("Enter the number of vertices:");
  scanf("%d", &n);
  printf("Enter the cost adjacency matrix:\n");
for(int i=0; i<n; i++){
    for(int j=0; j<n; j++){
            scanf("%d", &cost[i][j]);
        }
    }
  kruskal(cost, n);
printf("The edges of the minimum spanning tree are:\n");
for(int i=0; i<n-1; i++){
        printf("%d - %d\n", t[i][0], t[i][1]);
    }
printf("The minimum cost of the spanning tree is %d\n", sum);
return 0;
}
```

Output

Enter the number 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 of the minimum spanning tree are:

0 - 1
0 - 3
2 - 0

The minimum cost of the spanning tree is 6.

2) Dijkstras Algorithm:

```
#include<stdio.h>
void dijkstras(int a[10][10], int n){
    int d[10],p[10], visited[10], source, u, v, min, s,k;
    printf("Enter the source\n");
    scanf("%d",&s);
    source = s;
    for(int i = 0; i<n; i++){
                d[i] = a[source][i];
                visited[i] = 0;
                p[i] = source;
    }
    visited[source] = 1;
    for(int i = 0; i<n; i++){
        min = 999;
        u = -1;
        for(int j=0;j<n; j++){
            if(!visited[j] && d[j]<min){
                min = d[j];
                u = j;
            }
        }
    }
```

```
if (u == -1){
    break;
}
visited[u] = 1;
for(v = 0; v < n; v++){
    if(!visited[v] && a[u][v] && d[u] + a[u][v] < d[v]){
        d[v] = d[u] + a[u][v];
        p[v] = u;
    }
}
}
printf("\nShortest distances from vertex %d: \n", source);
for(int i=0; i<n; i++){
    if(d[i] == 9999){
        printf("Vertex %d is not reachable.\n", i);
    }else{
        printf("Vertex %d -> Distance = %d, Parent = %d\n",
               i, d[i], p[i]);
    }
}
}
int main(){
    int n;
    int a[10][10];
    printf("Enter the number of vertices (max 10): ");
    scanf("%d", &n);
    printf("Enter the adjacency matrix:\n");
    for(int i=0; i<n; i++){
        for(int j=0; j<n; j++){
            scanf("%d", &a[i][j]);
        }
    }
    dijkstra(a, n);
    return 0;
}
```

Output:-

```
Enter the number of vertices (max 12): 4
Enter the adjacency matrix (9999 for infinity):
0  1  5  2
1  0  9999  9999
5  9999  0  3
2  9999  3  0

Enter the source: 0
Shortest distance from vertex 0:
Vertex 0 → Distance = 0, Parent = 0
Vertex 1 → Distance = 1, Parent = 0
Vertex 2 → Distance = 5, Parent = 0
Vertex 3 → Distance = 2, Parent = 0.
```

3) Knapsack

```
#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 total_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[max_i] * p[max_i];
    printf("Added %d %% (%d/%d) of object %d in the
        bag\n", (int)((float)taken/w[max_i] *
        100), w[max_i], p[max_i]. max_i);
  }
 }
 printf("Filled the bag with objects worth %.2f\n",
}
int main(){
  int n, W;
  printf("Enter the number of objects:");
  scanf("%d", &n);
  int p[n], w[n];
  printf("Enter the profits of all 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 objects: 5 10 15 7 8 9 4
Enter the weights of 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 bag.
Space left 4.
Added object 6 (3,9) completely in bag.
Space left 1.
Added 33% (3,10) of object 2 in the bag.
Filled the bag with objects worth 96.00

Sneha
5/7/24

4) Prims Algo

```
#include <stdio.h>
int i, j, u, v, sum, k, min, source;
int t[10][2], cost[10][10], p[10], d[10], s[10];
void prims(int cost[10][10], int n);
void main(){
    int n;
    printf("Enter the number of vertices: ");
    scanf("%d", &n);
    printf("Enter the adjacency matrix:\n");
    for(i=0; i<n; i++){
        for(j=0; j<n; j++){
            scanf("%d", &cost[i][j]);
        }
    }
    min = 999;
    prims(cost, n);
```

```
printf(" Result :\n");
prims(cost, n);
}
void prims(int cost[][50], int n){
    int d[10], p[10], s[10];
    int min = 999;
    int source = 0;
    int sum, k, 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][
      s[u] = 1;
      for(int v=0; v<n; v++)
      { if(s[v] == 0 && cost[u][v] < d[v])
        { d[v] = cost[u][v];
          p[v] = u;
        }}}
```

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

Output

Enter no-of vertices.
4
Enter cost adjacency matrix:
0 1 5 2
1 0 999 999
5 999 0 3
2 999 3 0
Result:
Shortest path cost:6
Minimum spanning tree vertices:
1, 0
3, 0
2, 3
0, 0.

12/07/2024 Friday

N-Queens Algorithm

Code:-

```
#include <stdio.h>
#include <stdbool.h>
bool place(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\n");
      count++;
      }else{
        k++;
        x[k] = 0;
      }else{
        k--;
      }
    }
    printf("Total solutions: %d\n", count);
    }
```

```
bool place(int x[20], 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 printSolutions(int x[20], 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
```

16/5/24