ADA LAB MANUAL-2024

Prof Sindhu B S

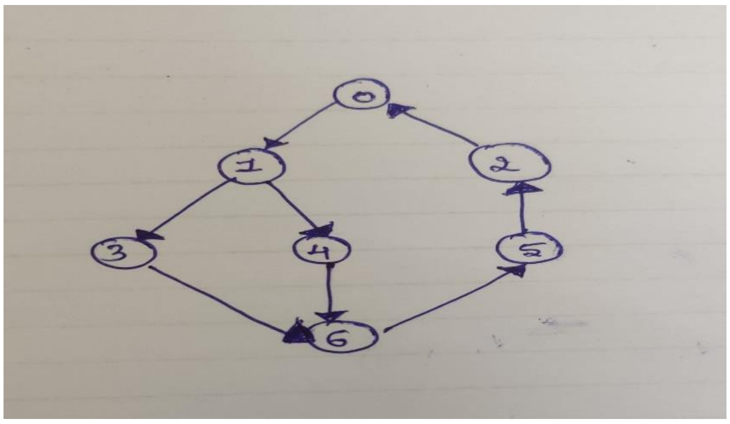
PES COLLEGE OF ENGINEERING ,Mandya

1.Print all the nodes reachable from a given starting node in a digraph using BFS method.

```
#include <stdio.h>
#include <stdlib.h>
int visited[10],G[10][10],v,r=0,f=0,Q[10];
void BFS(int u)
{
   int i;
   visited[u]=1;
   printf(" %d",u);
   for(i=0;i<v;i++)
   {
      if( G[u][i]==1 && visited[i]==0)
      {
         Q[r++]=i;
         visited[i]=1;
      }
    }
      if(r==f)
      return;
      else
         BFS(Q[f++]);
}
int main()
{
   int i,j,s;
   printf("enter the number of vertices: \n");
   scanf("%d",&v);
printf("enter the Adjacency matrix: \n");
   for(i=0;i<v;i++)
   {
       for(j=0;j<v;j++)
```

```
        {
           scanf("%d",&G[i][j]);
        }
        visited[i]=0;
    }
    printf("Enter the source node to begin The BFS :");
    scanf("%d",&s);
      printf("\nBFS Traversal is : ");
    BFS(s);
    return 0;
}
```

Output:

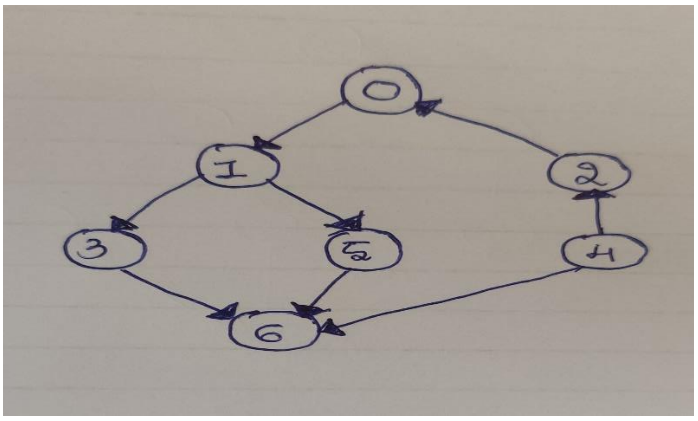


```
enter the number of vertices: 7
enter the Adjacency matrix:
0 1 0 0 0 0 0
0 0 0 1 1 0 0
1 0 0 0 0 0 0
0 0 0 0 0 0 1
0 0 0 0 0 0 1
0 0 1 0 0 0 0
0 0 0 0 0 1 0
Enter the source node to begin The BFS :5
BFS Traversal is:  5 2 0 1 3 4 6
```

2. Obtain the Topological ordering of vertices in a given digraph (DFS Based).

```
#include <stdio.h>
#include <stdlib.h>
int visited[10],G[10][10], v , top=-1 ,S[10];
void DFS(int u)
{
   int i;
   visited[u]=1;
   for(i=0;i<v;i++)
   {
      if( G[u][i]==1 && visited[i]==0){
      DFS(i);
      }
    }
     S[++top]=u;
}
int main()
{
   int i,j,s;
   printf("enter the number of vertices: \n");
   scanf("%d",&v);
   printf("enter the Adjacency Matrix: \n");
   for(i=0;i<v;i++)
   {
       for(j=0;j<v;j++)
       {
          scanf("%d",&G[i][j]);
       }
       visited[i]=0;
   }
     for(i=0;i<v;i++)
     {
          if(visited[i]==0){
                DFS(i);
          }
      }
      printf("\nTopological Sorting using DFS Method : ");
 while(top!=-1)
    printf(" %d",S[top--]);
    return 0;
}
```

Output:



```
enter the number of vertices: 7
enter the Adjacency Matrix:
0 1 0 0 0 0 0
0 0 0 1 0 1 0
1 0 0 0 0 0 0
0 0 0 0 0 0 1
0 0 1 0 0 0 0
0 0 0 0 0 0 1
0 0 0 0 0 0 0
Topological Sorting using DFS Method:  4 2 0 1 5 3 6
```

3. Sort a given set of elements using Merge sort method and determine the time taken to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n.

```
#include <stdio.h>
#define INF 9999
#include<time.h>
void Merge(int A[], int p, int q, int r)
{
   int i, j, k, n1 = q - p + 1, n2 = r - q;
   int L[n1 + 1], R[n2 + 1];
   for (i = 0; i < n1; i++)
      L[i] = A[p + i];
   for (j = 0; j < n2; j++)
      R[j] = A[q + j + 1];
   L[n1] = R[n2] = INF;
   i=j = 0;
   for (k = p; k <= r; k++)
   {
      if (L[i] <= R[j])
         A[k] = L[i++];
      else
         A[k] = R[j++];
   }
}
void MergeSort(int A[], int p, int r)
{
   if (p < r)
   {
      int q = (p + r) / 2;
```

```
        MergeSort(A, p, q);

        MergeSort(A, q + 1, r);

        Merge(A, p, q, r);

    }

}


int main()

{

    int n, i;

    int A[65000];

    clock_t start ,end;

    double duration;

    printf("Enter input size:\n");

    scanf("%d", &n);

    printf("Random elements are:\n");

    for (i = 0; i < n; i++)

    {

        A[i]=rand()%100;

        printf("%d ", A[i]);

    }

    start=clock();

    MergeSort(A, 0, n - 1);

    end=clock();

    duration=((double)(end-start))/CLOCKS_PER_SEC;

    printf("\nSorted array: ");

    for (i = 0; i < n; i++)

        printf("%d ", A[i]);

    printf("\n");

    printf("Time taken: %f ", duration);

    return 0;

}
```

Output:

Enter input size: 6

Random elements are: 41 67 34 0 69 24

Sorted array: 0 24 34 41 67 69

Time taken: 0.000000

4. **Sort a given set of elements using Quick sort method and determine the time taken to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n.**

```
#include <stdio.h>

#include <stdlib.h>

#include<time.h>


int Partition(int A[],int p,int r)

{

   int j,temp,x=A[r],i=p-1;

   for(j=p;j<r;j++)

   {

      if(A[j]<=x)

      {

         i++;

         temp=A[i];

         A[i]=A[j];

         A[j]=temp;

      }

   }

   temp=A[i+1];

   A[i+1]=A[r];

   A[r]=temp;

   return i+1;

}


void QuickSort(int A[],int p,int r)

{

   if(p<r)

   {

      int q=Partition(A,p,r);

      QuickSort(A,p,q-1);
```

```
        QuickSort(A,q+1,r);
    }
}
int main()
{
    int n, i;
    int A[65000];
    clock_t start ,end;
    double duration;
    printf("Enter input size:\n");
    scanf("%d", &n);
    printf("Generated Random Numbers are::\n");
    for (i = 0; i < n; i++)
    {
        A[i]=rand()%100;
        printf("%d ", A[i]);
    }
    start=clock();
    QuickSort(A, 0, n - 1);
    end=clock();
    duration=((double)(end-start))/CLOCKS_PER_SEC;
    printf("\nSorted array: ");
    for (i = 0; i < n; i++)
        printf("%d ", A[i]);
    printf("\n");
    printf("Time taken: %f ", duration);
    return 0;
}
```

Output:

Enter input size:

6

Generated Random Numbers are: 41 67 34 0 69 24

Sorted array: 0 24 34 41 67 69

Time taken: 0.000000

5. Find the Pattern string in a given Text string using Horspool's String-Matching Algorithm.

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

int ST[128];
char txt[200], pat[30];

int main() {
   printf("Enter the Text: \n");
   gets(txt);
   printf("Enter the Pattern: \n");
   gets(pat);
   createST();
   int pos = searchPat();
   if (pos == -1)
      printf("Pattern Not Found\n");
   else
      printf("Pattern Found at Index: %d\n", pos);
   return 0;
}

void createST() {
   int PL = strlen(pat);
   for (int i = 0; i < 128; i++)
      ST[i] = PL;
   for (int j = 0; j < PL - 1; j++)
      ST[(int)pat[j]] = PL - 1 - j;
}
```

```
int searchPat() {
    int TL = strlen(txt) ,PL = strlen(pat) , j;
    for (int i = 0; i <= TL - PL;) {
        for (j = PL - 1; j >= 0; j--) {
            if (pat[j] != txt[i + j])
                break;
        }
        if (j < 0)
            return i;
        i += ST[(int)txt[i + PL - 1]];
    }
    return -1;
}
```

Output:

Enter the Text: Dont Stress Do Best Forget Rest and All the Best

Enter the Pattern: Forget

Pattern Found at Index: 20

6. Sort a given set of elements using Heap Sort algorithm.

```
#include <stdio.h>
void maxHeapify(int A[], int i, int heapsize) {
    int l = 2 * i + 1 ,r = 2 * i + 2 ,big = i;
    if (l < heapsize && A[l] > A[i])
        big = l;
    if (r < heapsize && A[r] > A[big])
        big = r;
    if (big != i) {
        int temp = A[i];
        A[i] = A[big];
        A[big] = temp;
        maxHeapify(A, big, heapsize);
    }
}

void buildMaxHeap(int A[], int n)
{
    for (int i = n/ 2 - 1; i >= 0; i--)
        maxHeapify(A, i, n);
}

void heapSort(int A[], int n)
{
    buildMaxHeap(A, n);
    int hsize = n;
    for (int i = n - 1; i >= 1; i--)
    {
        int temp = A[0];
        A[0] = A[i];
        A[i] = temp;
        hsize--;
        maxHeapify(A, 0, hsize);
    }
}
int main()
{
    int A[10],i,n;
    printf("Enter the size:\n");
    scanf("%d",&n);
    printf("Enter the %d Elements:\n",n);
    for(i=0;i<n;i++)
       scanf("%d",&A[i]);
    heapSort(A,n);
    printf("Sorted array:\n");
    for (int i = 0; i < n; i++)
        printf("%d ", A[i]);
    return 0;
}
```

Output:

```
Enter the size: 11
Enter the 11 Elements:
1 3 5 4 6 13 10 9 8 15 17
Sorted array:
1 3 4 5 6 8 9 10 13 15 17
```

7. Implement 0/1 Knapsack problem using Dynamic Programming

```
#include <stdio.h>
#include <stdlib.h>
int p[30], w[30];
int T[30][30];
int n, max;

int Calculate() {
    for(int i = 0; i <= n; i++) {
        for(int j = 0; j <= max; j++) {
            if (i == 0 || j == 0) {
                T[i][j] = 0;
            }
        }
    }

    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= max; j++) {
            if (w[i] <= j) {
                T[i][j] = (p[i] + T[i-1][j - w[i]] > T[i-1][j]) ? p[i] + T[i-1][j - w[i]] : T[i-1][j];
            } else {
                T[i][j] = T[i-1][j];
            }
        }
    }
    return T[n][max];
}

void FindItems() {
    int i = n, j = max;
    while (i > 0 && j > 0) {
        if (T[i][j] != T[i-1][j]) {
            printf("Item %d (Weight: %d, Profit: %d)\n", i, w[i], p[i]);
            j -= w[i];
        }
        i--;
    }
}

int main() {
    printf("Enter number of items available: \n");
    scanf("%d",&n);

    for (int i = 1; i <= n; i++) {
        printf("Enter weight and profit for item-%d: \n", i);
        scanf("%d %d", &w[i], &p[i]);
    }
     printf("Enter the maximum weight for Knapsack (Bag) to be filled: \n");
    scanf("%d",&max);

    int res = Calculate();
    printf("\nProfit: %d\n\n", res);

    for(int i = 0; i <= n; i++) {
        for(int j = 0; j <= max; j++) {
            printf("%2d ", T[i][j]);
        }
        printf("\n");
    }

    printf("\n\nItems included in the knapsack:\n");
    FindItems();
    return 0;
}
```

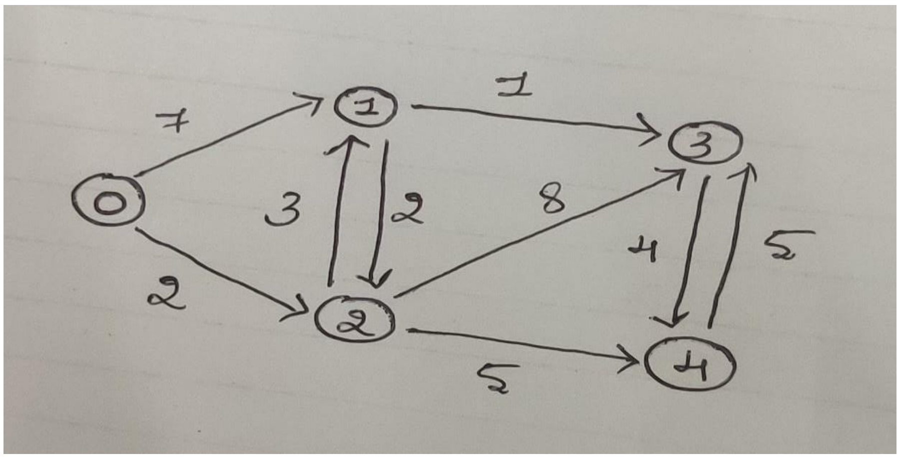
Output:

```
Enter number of items available:      3
Enter weight and profit for item-1:           1          10
Enter weight and profit for item-2:           2          12
Enter weight and profit for item-3: 4          28
Enter the maximum weight for Knapsack (Bag) to be filled:       6

Profit: 40

 0   0   0   0   0   0   0
 0 10 10 10 10 10 10
 0 10 12 22 22 22 22
 0 10 12 22 28 38 40

Items included in the knapsack:
Item 3 (Weight: 4, Profit: 28)
Item 2 (Weight: 2, Profit: 12)
```

8. From a given vertex in a weighted connected graph, find shortest paths to other Vertices using Dijikstra's algorithm.

```
#include <stdio.h>
#define INF 9999
int min(int dist[], int Set[], int n) {
    int min = INF, minIndex = -1;
    for (int v = 0; v < n; v++) {
        if (!Set[v] && dist[v] <= min) {
            min = dist[v];
            minIndex = v;
        }
    }
    return minIndex;
}
void dijkstra(int graph[][100], int n, int src) {
    int dist[100],Set[100];
    for (int i = 0; i < n; i++) {
        dist[i] = INF;
        Set[i] = 0;
    }
    dist[src] = 0;
    for (int c= 0; c< n - 1; c++) {
        int u = min(dist, Set, n);
        Set[u] = 1;
        for (int v = 0; v < n; v++) {
            if (!Set[v] && graph[u][v]!= 0 && graph[u][v]!= 999 && dist[u]!= INF && dist[u] +
graph[u][v] < dist[v]) {
                dist[v] = dist[u] + graph[u][v];
            }
        }
    }
    printf("Vertex \t Distance from Source\n");
    for (int i = 0; i < n; i++) {
        printf("%d \t\t %d\n", i, dist[i]);
    }
}
int main()
{
    int graph[100][100],n, src;
    printf("Enter the number of vertices: ");
    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", &graph[i][j]);
        }
    }
    printf("Enter the source vertex: ");
    scanf("%d", &src);
    dijkstra(graph, n, src);
    return 0;
}
```

Output:



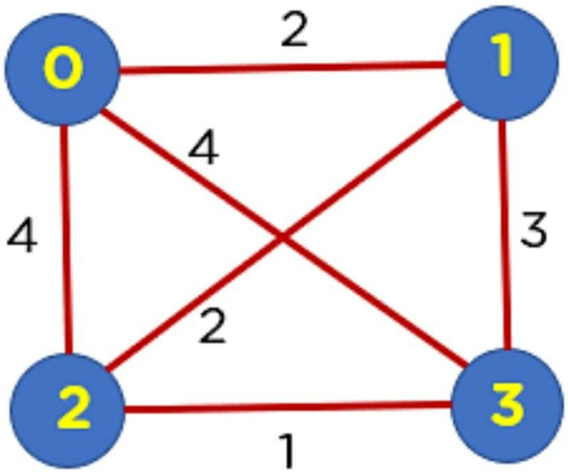
```
Enter the number of vertices: 5
Enter the adjacency matrix:
0       7       2       999     999
999     0       2       1       999
999     3       0       8       5
999     999     999     0       4
999     999     999     5       0
Enter the source vertex: 0
Vertex   Distance from Source
0          0
1          5
2          2
3          6
4          7
```

9 . Find minimum cost spanning tree of a given undirected graph using Kruskal's Algorithm.

```
#include <stdio.h>
int find(int p[], int i) {
    if (p[i] == i)
        return i;
    return find(p, p[i]);
}
void swap(int *a, int *b) {
    int t = *a;
    *a = *b;
    *b = t;
}
void kruskal(int v, int g[100][100], int ec, int e[100][3],int p[100]) {
    int i, j, k = 0;
    int A[100][2];
    for (i = 0; i < ec - 1; i++) {
        for (j = i + 1; j < ec; j++) {
            if (e[i][2] > e[j][2]) {
                swap(&e[i][0], &e[j][0]);
                swap(&e[i][1], &e[j][1]);
                swap(&e[i][2], &e[j][2]);
            }
        }
    }
    int sum = 0;
    for (i = 0; i < ec; i++) {
        int x = find(p, e[i][0]);
        int y = find(p, e[i][1]);

        if (x != y) {
            A[k][0] = e[i][0];
            A[k][1] = e[i][1];
            k++;
            sum += e[i][2];
            p[y] = x;
        }
    }
    printf("Edges in MinSpanTree is:\n");
    for (i = 0; i < k; i++) {
        printf("%d -- %d\n", A[i][0], A[i][1]);
    }
    printf("Sum of span tree: %d\n", sum);
}
int main() {
    int v,e[100][3],g[100][100],ec = 0,p[100];      //ec=edgecount
    printf("Enter num of vertices: ");
    scanf("%d", &v);
    printf("Enter adj matrix:\n");
    for (int i = 0; i < v; i++) {
        for (int j = 0; j < v; j++) {
            scanf("%d", &g[i][j]);
             if (g[i][j] != 0) {
                e[ec][0] = i;
                e[ec][1] = j;
                e[ec][2] = g[i][j];
                ec++;
            }
            p[i]=i;
        }
    }
    kruskal(v, g, ec, e,p);
    return 0;
```

}

Output:



Enter num of vertices: 4

Enter adj matrix:

```
0 2 4 4
2 0 2 3
4 2 0 1
4 3 1 0
```

Edges in MinSpanTree is:

```
2 -- 3
1 -- 2
0 -- 1
```

Sum of span tree: 5

10. Implement Sum-of-Subset problem of a given set S = {s1, s2, ..., sn} of 'n' Positive integers whose sum is equal to a given positive integer .

```
#include <stdio.h>
#include <stdbool.h>
int arr[100], subset[100], n, tgt;

bool subsetSum(int n, int tgt) {
   if (tgt == 0) return true;
   if (n == 0) return false;
   if (arr[n - 1] > tgt) return subsetSum(n - 1, tgt);
   return subsetSum(n - 1, tgt) || subsetSum(n - 1, tgt - arr[n - 1]);
}

void printSubsets(int n, int tgt, int subset[], int size) {
   if (tgt == 0) {
      for (int i = 0; i < size; i++) {
         printf("%d ", subset[i]);
      }
      printf("\n");
      return;
   }
   if (n == 0) return;
   if (arr[n - 1] <= tgt) {
      subset[size] = arr[n - 1];
      printSubsets(n - 1, tgt - arr[n - 1], subset, size + 1);
   }
   printSubsets(n - 1, tgt, subset, size);
}

int main() {
   printf("Enter the number of elements: ");
   scanf("%d", &n);
```

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

    printf("Enter the sum: ");
    scanf("%d", &tgt);

    if (subsetSum(n, tgt)) {
        printf("Found a subset with given sum %d\n", tgt);
        printSubsets(n, tgt, subset, 0);
    } else {
        printf("No subset with given sum %d\n", tgt);
    }
    return 0;
}
```

Output:

```
Enter the number of elements: 5
Enter the elements:
1
2
5
6
8
Enter the sum: 9
Found a subset with given sum 9
8 1
6 2 1
```