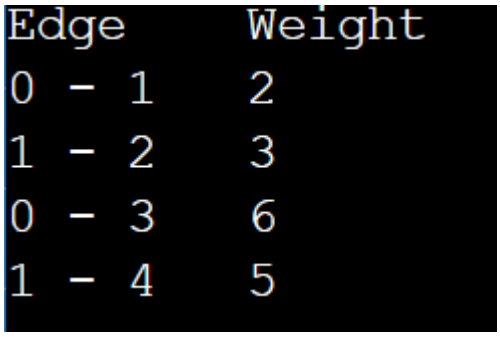
Practical-4.1 : Prim's Minimum Spanning Tree

```
#include <bits/stdc++.h>
using namespace std;
#define V 5
int minKey(int key[], bool mstSet[])
{
	int min = INT_MAX, min_index;
	for (int v = 0; v < V; v++)
		if (mstSet[v] == false && key[v] < min)
			min = key[v], min_index = v;
	return min_index;
}
void printMST(int parent[], int graph[V][V])
{
	cout << "Edge \tWeight\n";
	for (int i = 1; i < V; i++)
		cout << parent[i] << " - " << i << " \t"
			<< graph[i][parent[i]] << " \n";
}

void primMST(int graph[V][V])
{
	int parent[V];
	int key[V];
	bool mstSet[V];
	for (int i = 0; i < V; i++)
		key[i] = INT_MAX, mstSet[i] = false;
		key[0] = 0;
		parent[0] = -1;
	for (int count = 0; count < V - 1; count++) {
		int u = minKey(key, mstSet);
```

```
                mstSet[u] = true;
                for (int v = 0; v < V; v++)
                        if (graph[u][v] && mstSet[v] == false
                                && graph[u][v] < key[v])
                                parent[v] = u, key[v] = graph[u][v];
        }
        printMST(parent, graph);
}
int main()
{
        int graph[V][V] = { { 0, 2, 0, 6, 0 },
                            { 2, 0, 3, 8, 5 },
                            { 0, 3, 0, 0, 7 },
                            { 6, 8, 0, 0, 9 },
                            { 0, 5, 7, 9, 0 } };
        primMST(graph);
        return 0;
}
```

Output:-



Prac-5.1 : Kruskal's Minimum Spanning Tree

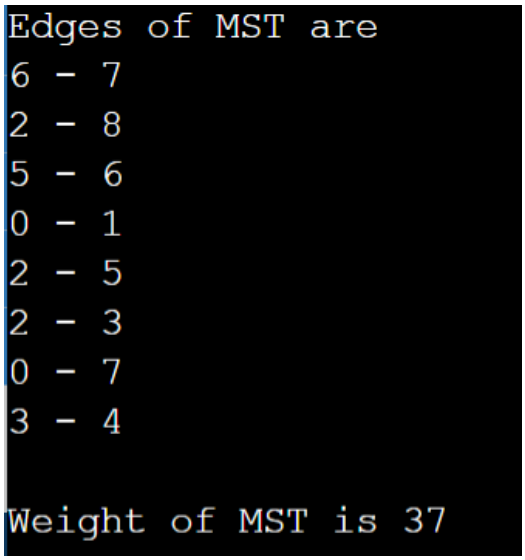
```
#include<bits/stdc++.h>
using namespace std;
typedef pair<int, int> iPair;
struct Graph
{
	int V, E;
	vector< pair<int, iPair> > edges;
	Graph(int V, int E)
	{
		this->V = V;
		this->E = E;
	}
	void addEdge(int u, int v, int w)
	{
		edges.push_back({w, {u, v}});
	}

	int kruskalMST();
};
struct DisjointSets
{
	int *parent, *rnk;
	int n;
	DisjointSets(int n)
	{
		this->n = n;
		parent = new int[n+1];
		rnk = new int[n+1];
		for (int i = 0; i <= n; i++)
		{
```

```
                rnk[i] = 0;
                parent[i] = i;
            }
        }

        int find(int u)
        {
            if (u != parent[u])
                parent[u] = find(parent[u]);
            return parent[u];
        }
        void merge(int x, int y)
        {
            x = find(x), y = find(y);
            if (rnk[x] > rnk[y])
                parent[y] = x;
            else
                parent[x] = y;

            if (rnk[x] == rnk[y])
                rnk[y]++;
        }
};
int Graph::kruskalMST()
{
        int mst_wt = 0; // Initialize result
        sort(edges.begin(), edges.end());
        DisjointSets ds(V);
        vector< pair<int, iPair> >::iterator it;
        for (it=edges.begin(); it!=edges.end(); it++)
        {
```

```
            int u = it->second.first;
            int v = it->second.second;

            int set_u = ds.find(u);
            int set_v = ds.find(v);
            if (set_u != set_v)
            {

                cout << u << " - " << v << endl;

                mst_wt += it->first;
                ds.merge(set_u, set_v);
            }
        }
        return mst_wt;
}
int main()
{
        int V = 9, E = 14;
        Graph g(V, E);
        g.addEdge(0, 1, 4);
        g.addEdge(0, 7, 8);
        g.addEdge(1, 2, 8);
        g.addEdge(1, 7, 11);
        g.addEdge(2, 3, 7);
        g.addEdge(2, 8, 2);
        g.addEdge(2, 5, 4);
        g.addEdge(3, 4, 9);
        g.addEdge(3, 5, 14);
        g.addEdge(4, 5, 10);
        g.addEdge(5, 6, 2);
```

```
        g.addEdge(6, 7, 1);
        g.addEdge(6, 8, 6);
        g.addEdge(7, 8, 7);
        cout << "Edges of MST are \n";
        int mst_wt = g.kruskalMST();
        cout << "\nWeight of MST is " << mst_wt;
        return 0;
}
```

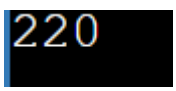
Output:



Prac-6.1 : Knapsack Problem Using Dynamic Programming

```
#include <bits/stdc++.h>
using namespace std;
int max(int a, int b) { return (a > b) ? a : b; }
int knapSack(int W, int wt[], int val[], int n)
{
	if (n == 0 || W == 0)
		return 0;
	if (wt[n - 1] > W)
		return knapSack(W, wt, val, n - 1);
	else
		return max(
			val[n - 1]
				+ knapSack(W - wt[n - 1],
					wt, val, n - 1),
			knapSack(W, wt, val, n - 1));
}
int main()
{
	int val[] = { 60, 100, 120 };
	int wt[] = { 10, 20, 30 };
	int W = 50;
	int n = sizeof(val) / sizeof(val[0]);
	cout << knapSack(W, wt, val, n);
	return 0;
}
```

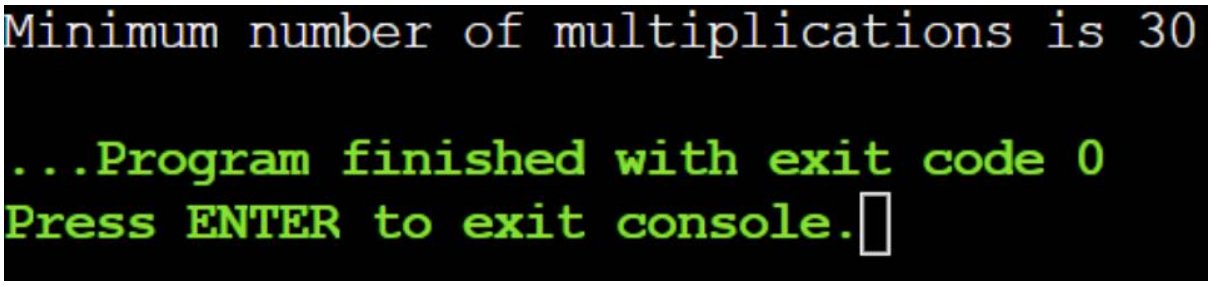
Output:-



Prac-7.1 :- Matrix Chain Multiplication using Dynamic Programming

```
#include <bits/stdc++.h>
using namespace std;
int MatrixChainOrder(int p[], int i, int j)
{
if (i == j)
return 0;
int k;
int min = INT_MAX;
int count;
for (k = i; k < j; k++)
{
count = MatrixChainOrder(p, i, k)+ MatrixChainOrder(p, k + 1, j)+ p[i - 1] * p[k] * p[j];
if (count < min)
min = count;
}
return min;
}
int main()
{
int arr[] = { 1, 2, 3, 4, 3 };
int n = sizeof(arr) / sizeof(arr[0]);
cout << "Minimum number of multiplications is "<< MatrixChainOrder(arr, 1, n - 1);
}
```

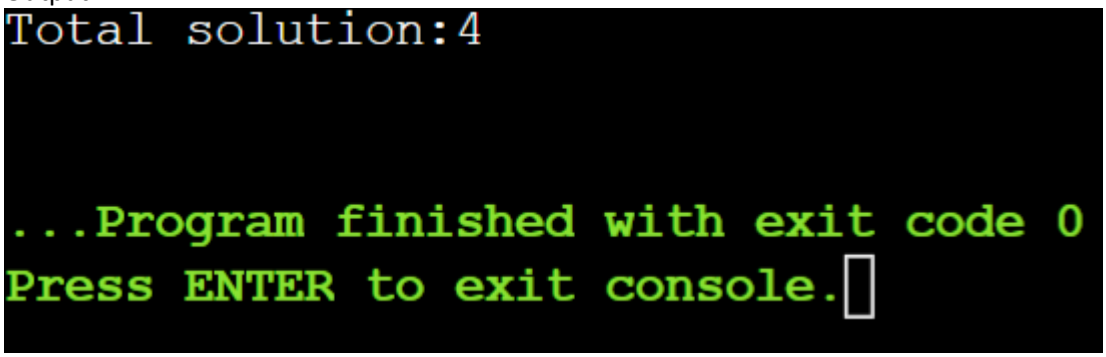
Output:-



Prac-8.1 :- Making A Change Problem Using Dynamic Programming

```
#include<iostream>
using namespace std;
int coins[]={1,2,3};
int NOF=3, sum=4;
int solve(int s,int i)
{
   if(NOF==0 || s>sum || i>=NOF){
      return 0;
   }
   else if(s==sum){
      return 1;
   }
   else{
      return solve(s+coins[i],i)+solve(s,i+1);
   }
}
   int main()
   {
      cout<<"Total solution:"<<solve(0,0)<<endl;
      return 0;
   }
```

Output:-



Prac-9.1 :- Depth First Search(DFS)

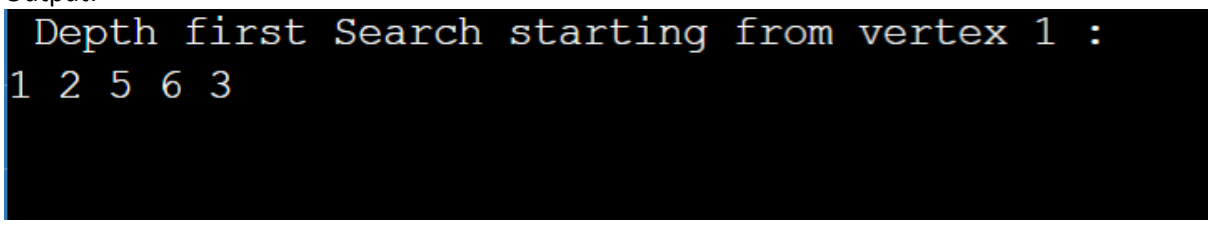
```
#include <iostream>
#include <ctime>
#include <malloc.h>
using namespace std;
struct nod
{
   int info;
   struct nod *next;
};

class stak
{
   struct nod *top;
   public:
      stak();
      void push(int);
      int pop();
      bool isEmpty();
      void display();
};

stak::stak()
{
   top = NULL;
}

void stak::push(int data)
{
   nod *p;
   if((p=(nod*)malloc(sizeof(nod)))==NULL){
      cout<<"Memory Exhausted";
      exit(0);
   }
   p = new nod;
   p->info = data;
   p->next = NULL;
   if(top!=NULL)
   {
      p->next = top;
   }
   top = p;
}

int stak::pop()
{
   struct nod *temp;
   int value;
   if(top==NULL){
      cout<<"\nThe stak is Empty"<<endl;
```

```
    }
    else
    {
        temp = top;
        top = top->next;
        value = temp->info;
        delete temp;
    }
    return value;
}

bool stak::isEmpty()
{
    return (top == NULL);
}

void stak::display()
{
    struct nod *p = top;
    if(top==NULL){
        cout<<"\nNothing to Display\n";
    }
    else
    {
        cout<<"\n The contents of stak\n";
        while(p!=NULL){
            cout<<p->info<<endl;
            p = p->next;
        }
    }
}

class Graph
{
    private:
        int n;
        int **A;
    public:
        Graph(int siz = 2);
        ~Graph();
        bool isConnected(int, int);
        void addEdge(int x, int y);
        void DFS(int , int);
};

Graph::Graph(int siz)
{
    int i, j;
    if (siz < 2) n = 2;
    else n = siz;
    A = new int*[n];
```

```
   for (i = 0; i < n; ++i)
      A[i] = new int[n];
   for (i = 0; i < n; ++i)
      for (j = 0; j < n; ++j)
         A[i][j] = 0;
}

Graph::~Graph()
{
   for (int i = 0; i < n; ++i)
   delete [] A[i];
   delete [] A;
}

bool Graph::isConnected(int x, int y)
{
   return (A[x-1][y-1] == 1);
}

void Graph::addEdge(int x, int y)
{
   A[x-1][y-1] = A[y-1][x-1] = 1;
}

void Graph::DFS(int x, int required)
{
   stak s;
   bool *visited = new bool[n+1];
   int i;
   for(i = 0; i <= n; i++)
      visited[i] = false;
   s.push(x);
   visited[x] = true;
   if(x == required) return;
   cout << " Depth first Search starting from vertex ";
   cout <<  x << " : " << endl;
   while(!s.isEmpty())
   {
      int k = s.pop();
      if(k == required) break;
      cout<<k<<" ";
      for (i = n; i >= 0 ; --i)
         if (isConnected(k, i) && !visited[i])
         {
            s.push(i);
            visited[i] = true;
         }
   }
   cout<<endl;
   delete [] visited;
}
```

```
int main()
{
   Graph g(8);
   g.addEdge(1, 2); g.addEdge(1, 3); g.addEdge(1, 4);
   g.addEdge(2, 5); g.addEdge(2, 6); g.addEdge(4, 7);
   g.addEdge(4, 8);
   g.DFS(1, 4);
   return 0;
}
```

Output:-



Prac-9.2: Breadth First Search(BFS)

```
#include<bits/stdc++.h>
using namespace std;
class Graph
{
	int V;
	vector<list<int>> adj;
public:
	Graph(int V);
	void addEdge(int v, int w);
	void BFS(int s);
};

Graph::Graph(int V)
{
	this->V = V;
	adj.resize(V);
}

void Graph::addEdge(int v, int w)
{
	adj[v].push_back(w); // Add w to v's list.
}
void Graph::BFS(int s)
{
	vector<bool> visited;
	visited.resize(V,false);
	list<int> queue;
	visited[s] = true;
	queue.push_back(s);

	while(!queue.empty())
	{
		s = queue.front();
		cout << s << " ";
		queue.pop_front();
		for (auto adjecent: adj[s])
		{
			if (!visited[adjecent])
			{
				visited[adjecent] = true;
				queue.push_back(adjecent);
			}
		}
	}
}
int main()
{
	Graph g(4);
	g.addEdge(0, 1);
```

```
        g.addEdge(0, 2);
        g.addEdge(1, 2);
        g.addEdge(2, 0);
        g.addEdge(2, 3);
        g.addEdge(3, 3);

        cout << "Following is Breadth First Traversal "
             << "(starting from vertex 2) \n";
        g.BFS(2);
        return 0;
}
```

Output:-

```
Following is Breadth First Traversal (starting from vertex 2)
2 0 3 1
```

Prac-10.1 :- Longest Common Subsequence

```
#include <bits/stdc++.h>
using namespace std;
int N, M;
int lcs(string S, string T, int i, int j)
{
   if (i == N || j == M)
      return 0;
   if (S[i] == T[j])
      return (lcs(S, T, i + 1, j + 1) + 1);
   else
      return max(lcs(S, T, i + 1, j), lcs(S, T, i, j + 1));
}
int main()
{
   string S = " abcde";
   string T = "acek";
   N = S.length();
   M = T.length();
   cout << "Length of longest common subsequence " << lcs(S, T, 0, 0);
   return 0;
}
```

Output:-