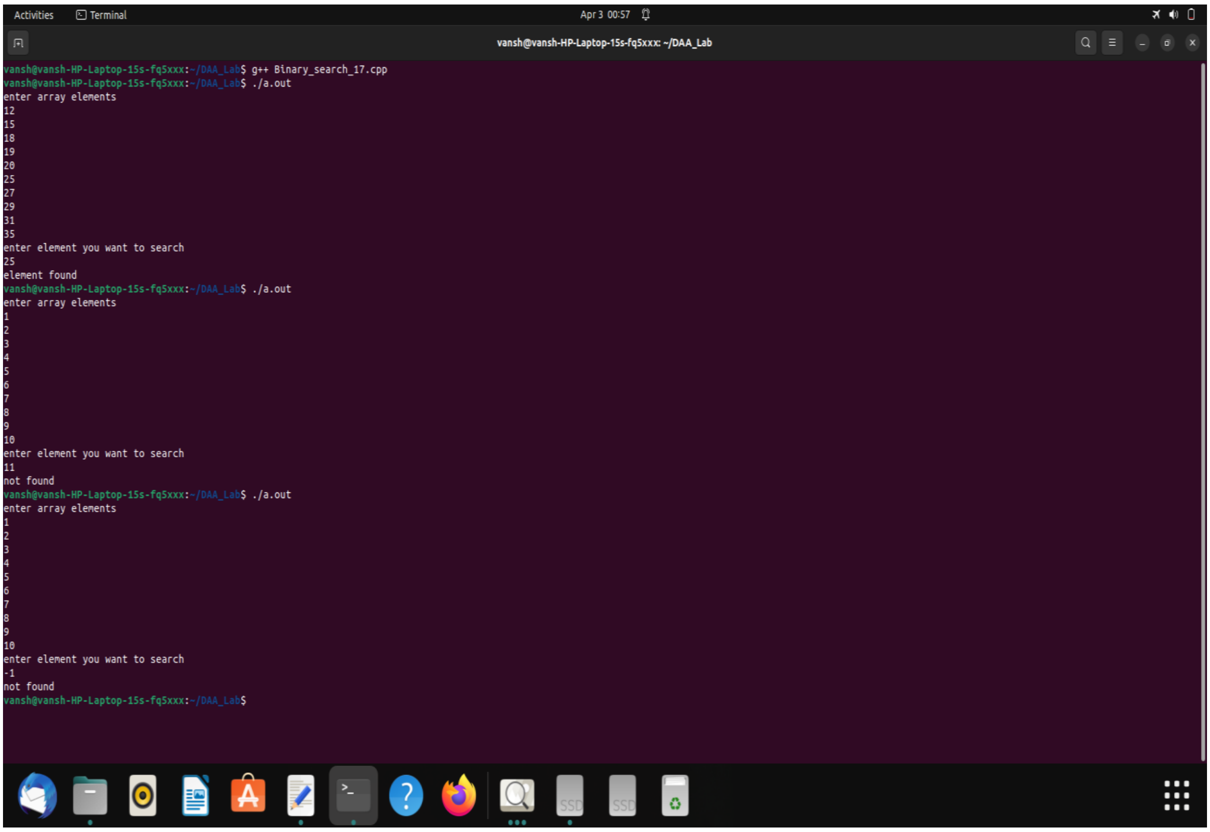
DAA LAB

Title: Binary Search using Divide and Conquer

```
#include <iostream>
using namespace std;
int main()
{
    int array[10],i,search;
    cout<<"enter array elements"<<endl;
    for(i=0;i<10;i++)
    {
       cin>>array[i];
    }
    cout<<"enter element you want to search"<<endl;
    int l=0;
    int up=9;
    int mid=(l+up)/2;
    cin>>search;
    while(l<=up)
    {
        if(search>array[mid])
        {
            l=mid+1;
        }
        else if(search==array[mid])
        {
            cout<<"element found"<<endl;
            break;
        }
        else
        {
            up=mid-1;
        }
     mid=(l+up)/2;
    }
    if(l>up)
    {
        cout<<"not found"<<endl;
    }
    return 0;
}
```



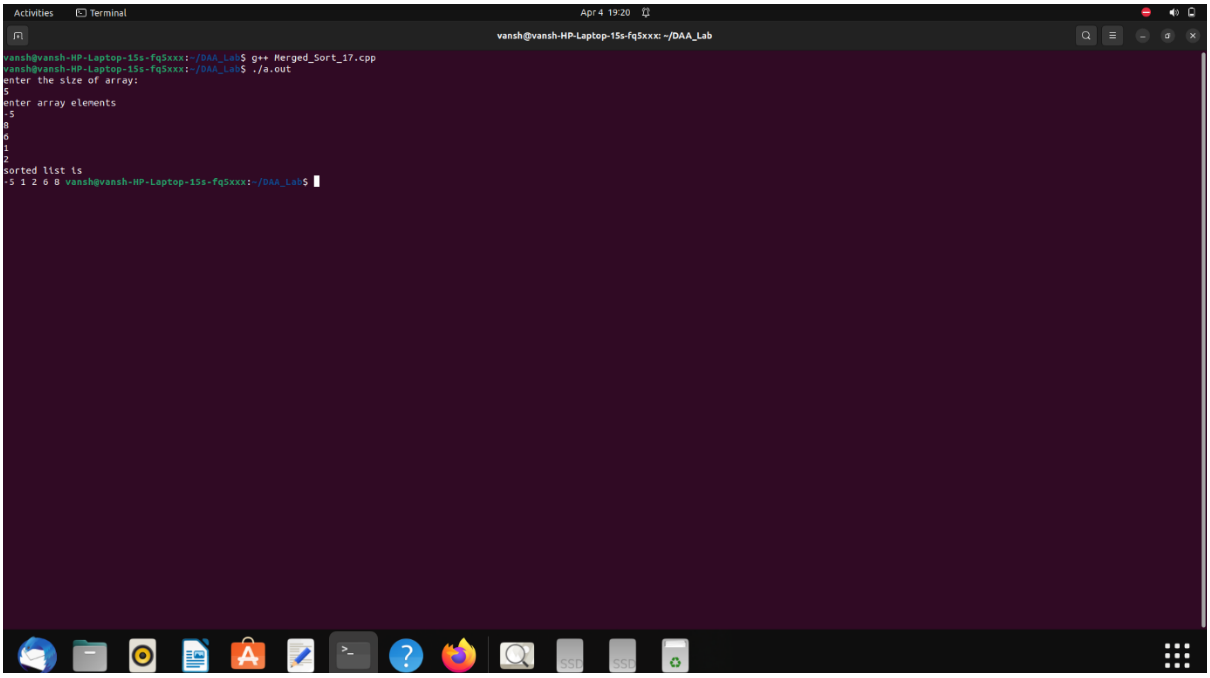
```
vansh@vansh-HP-Laptop-15s-fq5xxx:~/DAA_Lab$ g++ Binary_search_17.cpp
vansh@vansh-HP-Laptop-15s-fq5xxx:~/DAA_Lab$ ./a.out
enter array elements
12
15
18
19
20
25
27
29
31
35
enter element you want to search
25
element found
vansh@vansh-HP-Laptop-15s-fq5xxx:~/DAA_Lab$ ./a.out
enter array elements
1
2
3
4
5
6
7
8
9
10
enter element you want to search
11
not found
vansh@vansh-HP-Laptop-15s-fq5xxx:~/DAA_Lab$ ./a.out
enter array elements
1
2
3
4
5
6
7
8
9
10
enter element you want to search
-1
not found
vansh@vansh-HP-Laptop-15s-fq5xxx:~/DAA_Lab$
```

DAA LAB

Title: Merge Sort using Divide and Conquer

```
#include<iostream>
using namespace std;

#define max 100

void merge_sort(int arr[],int low,int up);
void merge_s(int arr[],int temp[],int low1,int up1,int low2,int up2);
void copy_s(int arr[],int temp[],int low,int up);

int main()
{
   int i,n,arr[max];
   cout<<"enter the size of array:"<<endl;
   cin>>n;
   cout<<"enter array elements "<<endl;
   for(i=0;i<n;i++)
   {
      cin>>arr[i];
   }
   merge_sort(arr,0,n-1);
   cout<<"sorted list is "<<endl;
```

```
    for(i=0;i<n;i++)
        cout<<arr[i]<<" ";

    return 0;
}

void merge_sort(int arr[],int low,int up)
{
    int mid;
    int temp[max];
    if(low<up)
    {
        mid=(low+up)/2;
        merge_sort(arr,low,mid); //left sublist
        merge_sort(arr,mid+1,up); //right sublist
        merge_s(arr,temp,low,mid,mid+1,up);
        copy_s(arr,temp,low,up);
    }
}

void merge_s(int arr[],int temp[],int low1,int up1,int low2,int up2)
{
    int i=low1;
    int j=low2;
    int k=low1;
    while((i<=up1)&&(j<=up2))
    {
        if(arr[i]<=arr[j])
            temp[k++]=arr[i++];
        else
            temp[k++]=arr[j++];
    }
    while(i<=up1)
        temp[k++]=arr[i++];
    while(j<=up2)
        temp[k++]=arr[j++];
}

void copy_s(int arr[],int temp[],int low,int up)
{
    int i;
    for(i=low;i<=up;i++)
        arr[i]=temp[i];
}
```



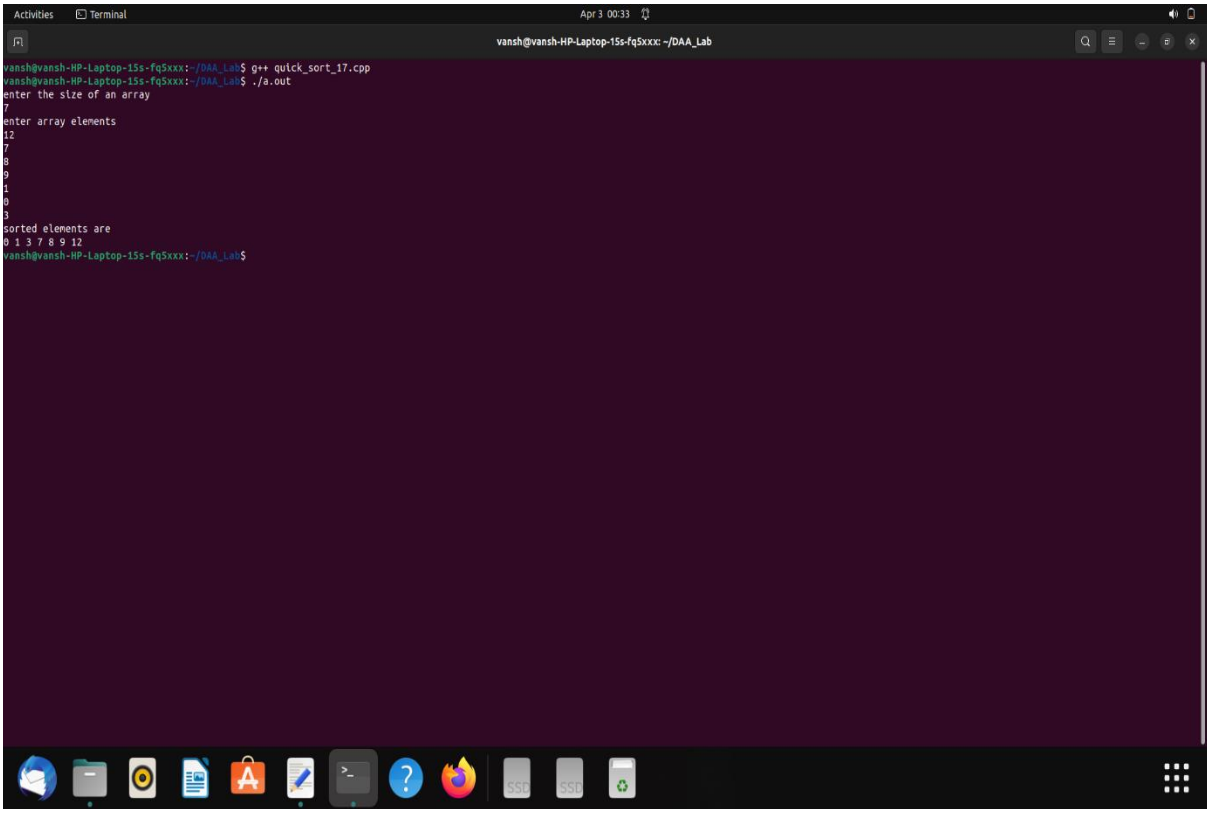
```
vansh@vansh-HP-Laptop-15s-fq5xxx:~/DAA_Lab$ g++ Merged_Sort_17.cpp
vansh@vansh-HP-Laptop-15s-fq5xxx:~/DAA_Lab$ ./a.out
enter the size of array:
5
enter array elements
-5
8
6
1
2
sorted list is
-5 1 2 6 8 vansh@vansh-HP-Laptop-15s-fq5xxx:~/DAA_Lab$ 
```

DAA LAB

Title: Quick Sort using Divide and Conquer

```
#include<iostream>
using namespace std;

void quick(int a[], int l, int up);
int partition(int a[], int l, int up);

int main()
{
    int n;
    cout<<"enter the size of an array"<<endl;
    cin>>n;
    int arr[n];
    cout<<"enter array elements"<<endl;
    for(int i=0;i<n;i++)
    {
      cin>>arr[i];
    }

    int low = 0;
    int up = n-1;
    quick(arr, low, up);

    cout << "sorted elements are" << endl;
    for (int i = 0; i <= up; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    return 0;
}

void quick(int a[], int l, int up) {
    if (l >= up) {
        return;
    }
    int pvtloc = partition(a, l, up);
    quick(a, l, pvtloc - 1); //left sublist
    quick(a, pvtloc + 1, up); //right sublist
}

int partition(int a[], int l, int up) {
    if (l >= up) {
        return l;
    }
    int temp, pvt;
    int i = l + 1;
    int j = up;
    pvt = a[l];
    while (i <= j) {
        while (a[i] < pvt) {
            i++;
        }
        while (a[j] > pvt) {
            j--;
        }
        if (i < j) {
            temp = a[i];
            a[i] = a[j];
            a[j] = temp;
            i++;
            j--;
        } else {
            i++;
        }
    }
    // Swap pivot with element at position j
    temp = a[l];
    a[l] = a[j];
    a[j] = temp;
    return j;
}
```

DAA LAB



```
vansh@vansh-HP-Laptop-15s-fq5xxx:~/DAA_Lab$ g++ quick_sort_17.cpp
vansh@vansh-HP-Laptop-15s-fq5xxx:~/DAA_Lab$ ./a.out
enter the size of an array
7
enter array elements
12
7
8
9
1
0
3
sorted elements are
0 1 3 7 8 9 12
vansh@vansh-HP-Laptop-15s-fq5xxx:~/DAA_Lab$
```

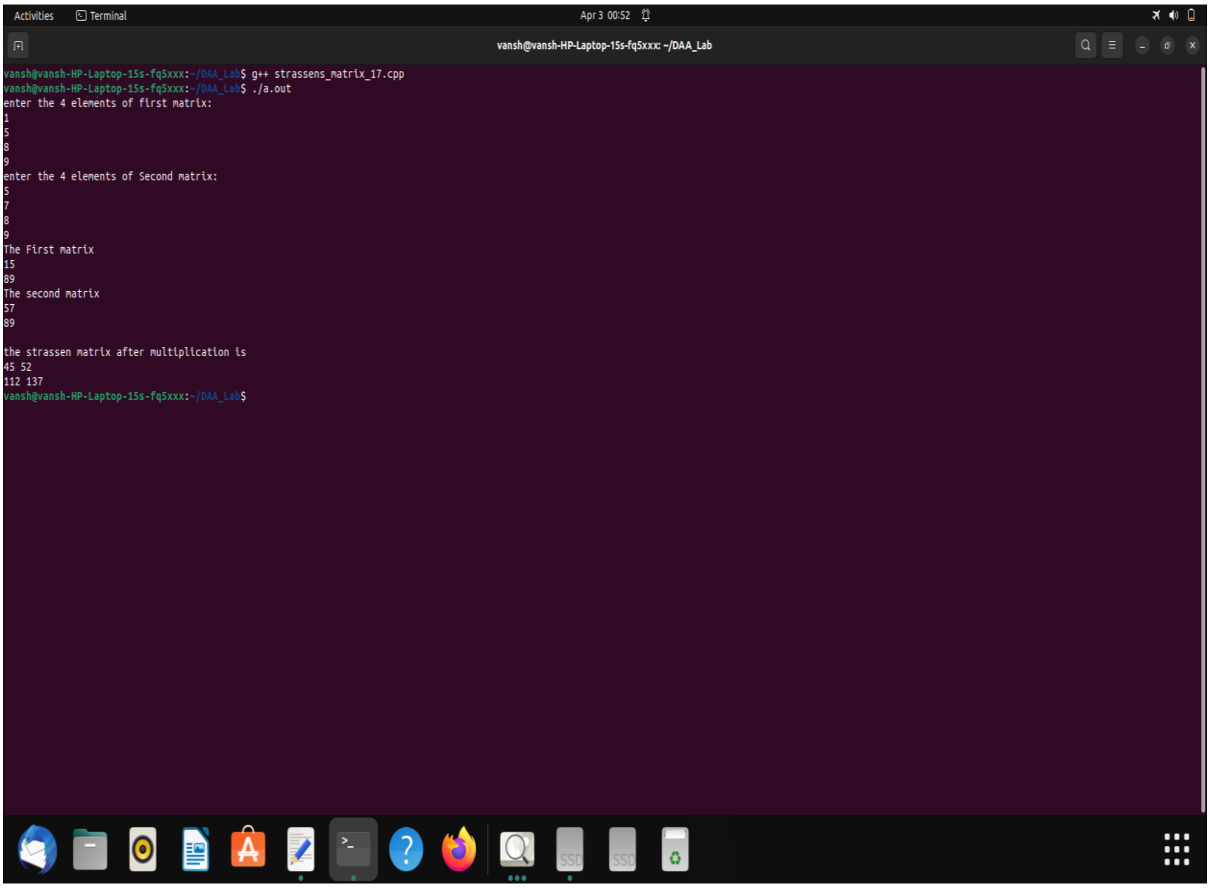
DAA LAB

Title: Strassen’s Matrix Multiplication using Divide and Conquer

```
#include <iostream>

using namespace std;

int main()
{
    int a[2][2],b[2][2],c[2][2],i,j;
    int m1,m2,m3,m4,m5,m6,m7;
    cout<<"enter the 4 elements of first matrix: "<<endl;
    for(i=0;i<2;i++)
        for(j=0;j<2;j++)
        cin>>a[i][j];
    cout<<"enter the 4 elements of Second matrix: "<<endl;
    for(i=0;i<2;i++)
        for(j=0;j<2;j++)
        cin>>b[i][j];
    cout<<"The First matrix"<<endl;
    for(i=0;i<2;i++)
    {
        for(j=0;j<2;j++)
        {
        cout<<a[i][j];
        }
    cout<<endl;
    }
```

```
cout<<"The second matrix"<<endl;
for(i=0;i<2;i++)
{
   for(j=0;j<2;j++)
   {
   cout<<b[i][j];
   }
cout<<endl;
}
m1= (a[0][0] + a[1][1]) * (b[0][0] + b[1][1]);
m2= (a[1][0] + a[1][1]) * b[0][0];
m3= a[0][0] * (b[0][1] - b[1][1]);
m4= a[1][1] * (b[1][0] - b[0][0]);
m5= (a[0][0] + a[0][1]) * b[1][1];
m6= (a[1][0] - a[0][0]) * (b[0][0]+b[0][1]);
m7= (a[0][1] - a[1][1]) * (b[1][0]+b[1][1]);

c[0][0] = m1 + m4- m5 + m7;
c[0][1] = m3 + m5;
c[1][0] = m2 + m4;
c[1][1] = m1 - m2 + m3 + m6;

cout<<endl<<"the strassen matrix after multiplication is "<<endl;
for(i=0;i<2;i++)
{
   for(j=0;j<2;j++)
```

```
        {
        cout<<c[i][j]<<" ";
        }
    cout<<endl;
    }

    return 0;
}
```

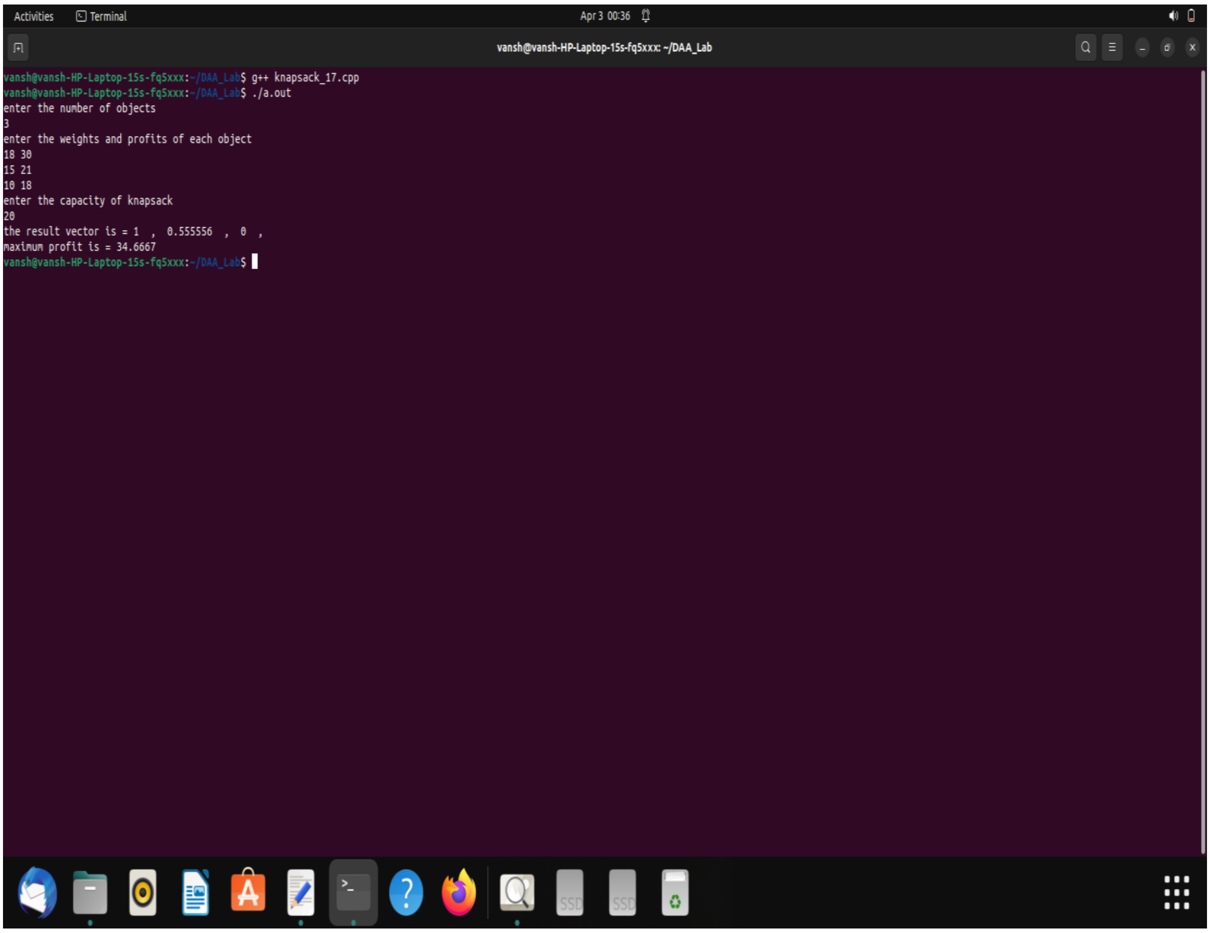


```
vansh@vansh-HP-Laptop-15s-fq5xxx:~/DAA_Lab$ g++ strassens_matrix_17.cpp
vansh@vansh-HP-Laptop-15s-fq5xxx:~/DAA_Lab$ ./a.out
enter the 4 elements of first matrix:
1
5
8
9
enter the 4 elements of Second matrix:
5
7
8
9
The First matrix
15
89
The second matrix
57
89

the strassen matrix after multiplication is
45 52
112 137
vansh@vansh-HP-Laptop-15s-fq5xxx:~/DAA_Lab$
```

DAA LAB

Title: Fractional Knapsack Problem Using Greedy Method

```
#include<iostream>
using namespace std;
void knapsack(int n,float weight[],float profit[],float capacity);
int main()
{
   float weight[20],profit[20],capacity;
   int num,i,j;
   float ratio[20],temp;
   cout<<"enter the number of objects"<<endl;
   cin>>num;
   cout<<"enter the weights and profits of each object"<<endl;
   for(i=0;i<num;i++)
   {
      cin>>weight[i]>>profit[i];
   }
   cout<<"enter the capacity of knapsack"<<endl;
   cin>>capacity;
   for(i=0;i<num;i++)
   {
      ratio[i]=profit[i]/weight[i];
   }
   for (i = 0; i < num; i++)
      {
         for (j = i + 1; j < num; j++)
         {
                if (ratio[i] < ratio[j])
                {
                    temp = ratio[j];
                    ratio[j] = ratio[i];
                    ratio[i] = temp;

                    temp = weight[j];
                    weight[j] = weight[i];
                    weight[i] = temp;

                    temp = profit[j];
                    profit[j] = profit[i];
                    profit[i] = temp;
                }
            }
        }

    knapsack(num,weight,profit,capacity);
    return 0;
}
void knapsack(int n,float weight[],float profit[],float capacity)
{
    float x[20],tp=0;
    int i,j,u;
    u=capacity;
    for(i=0;i<n;i++)
    {
        x[i]=0.0;
    }
    for(i=0;i<n;i++)
    {
        if(weight[i]>u)
            break;
        else
        {
            x[i]=1.0;
            tp=tp+profit[i];
            u=u-weight[i];
        }
    }

    if(i<n)
        x[i]=u/weight[i];
    tp=tp+(x[i]*profit[i]);
    cout<<"the result vector is = ";
    for(i=0;i<n;i++)
        cout<<x[i]<<"  ,  ";
    cout<<endl<<"maximum profit is = "<<tp<<endl;
}
```



```
vansh@vansh-HP-Laptop-15s-fq5xxx:~/DAA_Lab$ g++ knapsack_17.cpp
vansh@vansh-HP-Laptop-15s-fq5xxx:~/DAA_Lab$ ./a.out
enter the number of objects
3
enter the weights and profits of each object
18 30
15 21
10 18
enter the capacity of knapsack
20
the result vector is = 1  ,  0.555556  ,  0  ,
maximum profit is = 34.6667
vansh@vansh-HP-Laptop-15s-fq5xxx:~/DAA_Lab$ 
```

DAA LAB

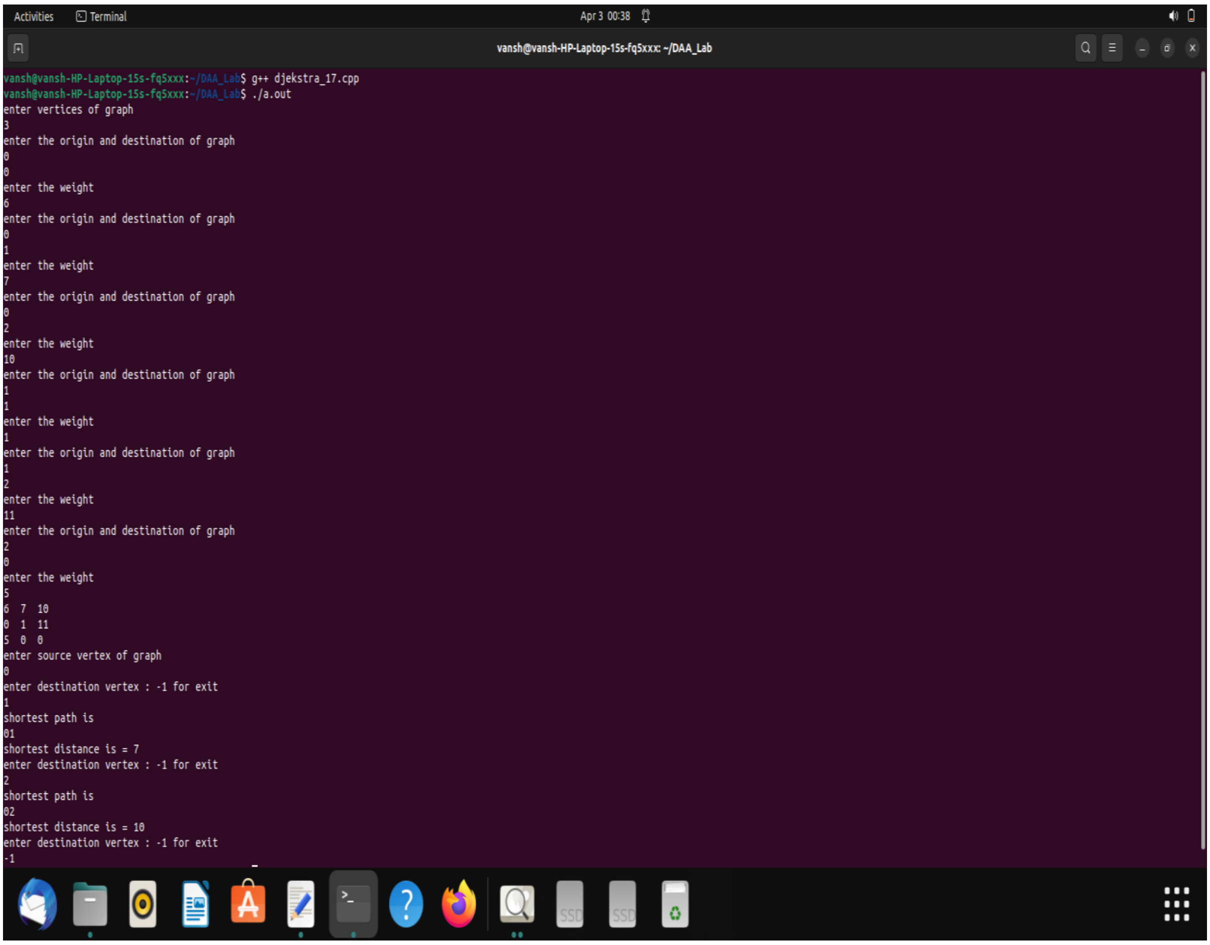
Title: Single Source Shortest Path Problem Dijkstra's Algorithm

```
#include <iostream>

using namespace std;

#define max 100
#define infinity 9999
#define nil -1
#define temp 0
#define permanent 1

int adj[max][max];
int predecer[max];
int pathlength[max];
int status[max];

int create_graph();
void djekstra(int src,int n);
int min_path(int n);
int findpath(int s,int v);

int main()
{
   int src,v;
   int n=create_graph();
   cout<<"enter source vertex of graph"<<endl;
   cin>>src;
   djekstra(src,n);
```

```
    while(1)
    {
        cout<<"enter destination vertex : -1 for exit"<<endl;
        cin>>v;
        if(v==-1)
            break;
        if((v<0)||(v>=n))
            cout<<"this vertex does not exist"<<endl;
        else if(v==src)
            cout<<"source and destination vertices are same"<<endl;
        else if(pathlength[v]==infinity)
            cout<<"there is no path from source to destination vertex"<<endl;
        else
            findpath(src,v);
    }

    return 0;
}

int create_graph()
{
    int n,max_e,i,origin,destination,wt,j;
    cout<<"enter vertices of graph"<<endl;
    cin>>n;
    max_e=n*(n-1);
    for(i=0;i<max_e;i++)
    {
        cout<<"enter the origin and destination of graph"<<endl;
        cin>>origin>>destination;
        cout<<"enter the weight"<<endl;
        cin>>wt;
        adj[origin][destination]=wt;
    }
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            cout<<adj[i][j]<<"  ";
        }
        cout<<endl;
    }
    return n;
}

void djekstra(int src,int n)
{
    int i,current;
    // 1 make all vertices temporary and initiliase pathlenght with infinity and predecer as nil

    for(i=0;i<n;i++)
    {
        status[i]=temp;
        pathlength[i]=infinity;
        predecer[i]=nil;
    }

    // 2 make source vertex pathlenght is 0
    pathlength[src]=0;

    while(1)
    {
        //3 from all temporary vertices find min pathlengh of vertices make it current and permanent
```

```
      current=min_path(n);
      if(current==nil)
         return;
      status[current]=permanent;

      //from all adjacy temporary vertices from current
      for(i=0;i<n;i++)
      {
         if((adj[current][i]!=0)&&(status[i]==temp))
         {
            if(pathlength[current]+adj[current][i]<pathlength[i])
            {
               predecer[i]=current;
               pathlength[i]=pathlength[current]+adj[current][i];
            }
         }
      }
   }
}

int min_path(int n)
{
   int i;
   int min=infinity;
   int k=nil;
   for(i=0;i<n;i++)
   {
      if((status[i]==temp)&&(pathlength[i]<min))
      {
         min=pathlength[i];
         k=i;
```

```
        }
    }
    return k;
}

int findpath(int s,int v)
{
    int i,u;
    int path[max];
    int shortdist=0;
    int count=0;
    while(v!=s)
    {
        count++;
        path[count]=v;
        u=predecer[v];
        shortdist+=adj[u][v];
        v=u;
    }
    count++;
    path[count]=s;
    cout<<"shortest path is "<<endl;
    for(i=count;i>=1;i--)
    {
        cout<<path[i];
    }
    cout<<endl;
    cout<<"shortest distance is = "<<shortdist<<endl;
    return 0;
}
```



```
vansh@vansh-HP-Laptop-15s-fq5xxx:~/DAA_Lab$ g++ djekstra_17.cpp
vansh@vansh-HP-Laptop-15s-fq5xxx:~/DAA_Lab$ ./a.out
enter vertices of graph
3
enter the origin and destination of graph
0
0
enter the weight
6
enter the origin and destination of graph
0
1
enter the weight
7
enter the origin and destination of graph
0
2
enter the weight
10
enter the origin and destination of graph
1
1
enter the weight
1
enter the origin and destination of graph
1
2
enter the weight
11
enter the origin and destination of graph
2
0
enter the weight
5
6  7  10
0  1  11
5  0  0
enter source vertex of graph
0
enter destination vertex : -1 for exit
1
shortest path is
01
shortest distance is = 7
enter destination vertex : -1 for exit
2
shortest path is
02
shortest distance is = 10
enter destination vertex : -1 for exit
-1
```

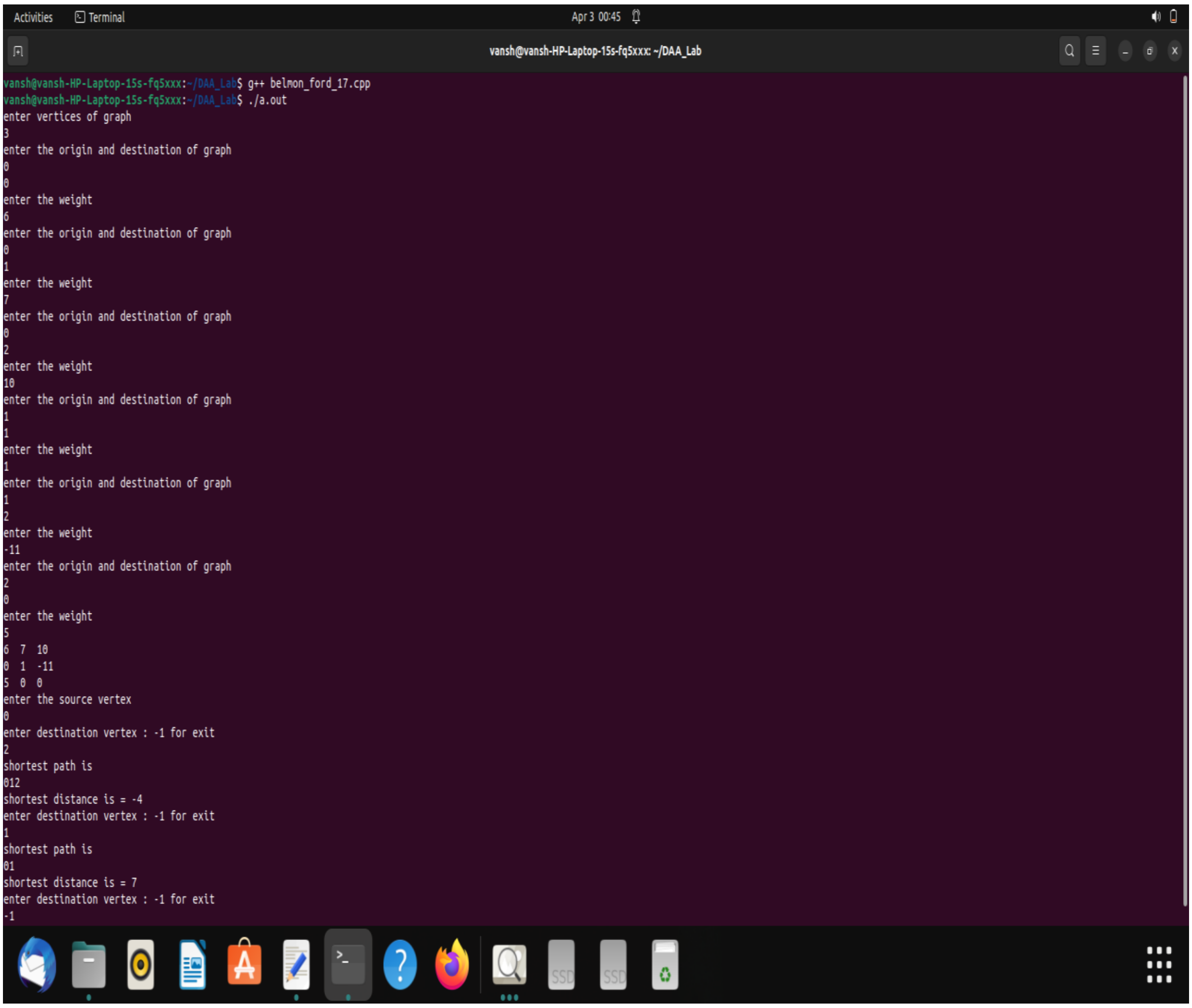
DAA LAB

Title: Single Source Shortest Path Problem Bellman Ford Algorithm

```
#include <iostream>
using namespace std;

#define max 100
#define infinity 9999
#define nil -1
#define true 1
#define false 0

int n; //number of vertices in graph
int adj[max][max];
int predecessor[max];
int pathlength[max];
int ispresent_in_queue[max];
int queue[max];
int front,rear;

int create_graph();
int bellmonford(int s);
void initilize_queue();
void insert_queue(int added_item);
int is_empty_queue();
int delete_queue();
int findpath(int s,int v);
int main()
{
    int s,flag,v;
    create_graph();
    cout<<"enter the source vertex"<<endl;
    cin>>s;
    flag=bellmonford(s);
    if(flag==-1)
    {
        cout<<"ERRor : negative cycle in graph"<<endl;
        exit(1);
    }
        while(1)
    {
        cout<<"enter destination vertex : -1 for exit"<<endl;
        cin>>v;
        if(v==-1)
            break;
        if((v<0)||(v>=n))
            cout<<"this vertex does not exist"<<endl;
        else if(v==s)
            cout<<"source and destination vertices are same"<<endl;
        else if(pathlength[v]==infinity)
            cout<<"there is no path from source to destination vertex"<<endl;
        else
            findpath(s,v);
    }
    return 0;
}



int create_graph()
{
    int max_e,i,origin,destination,wt,j;
    cout<<"enter vertices of graph"<<endl;
    cin>>n;
    max_e=n*(n-1);
    for(i=0;i<max_e;i++)
    {
        cout<<"enter the origin and destination of graph"<<endl;
        cin>>origin>>destination;
        cout<<"enter the weight"<<endl;
        cin>>wt;
        adj[origin][destination]=wt;
    }
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            cout<<adj[i][j]<<"  ";
        }
        cout<<endl;
    }
    return 0;
}

int bellmonford(int s)
{
    int k=0,i,current;
    // 1 initialise pathlength by infinity and predecerr is nil and not any vertex is present in queue
    for(i=0;i<n;i++)
    {
        predecessor[i]=nil;
        pathlength[i]=infinity;
        ispresent_in_queue[i]=false;
    }
    initilize_queue();

    // 2 make path length of source vertex equal to 0 and insert it into queue
    pathlength[s]=0;
    insert_queue(s);
    ispresent_in_queue[s]=true;
    while(!is_empty_queue())
    {
        // 3 delete the vertex from queue and make it current
        current=delete_queue();
        ispresent_in_queue[current]=false;
        if(s==current)
            k++;
        if(k>=n)
            return -1; //negative cycle can be reachable form source vertex
        for(i=0;i<n;i++)
        {
            if(adj[current][i]!=0)
            {
                if(pathlength[i]>adj[current][i]+pathlength[current])
                {
                    pathlength[i]=adj[current][i]+pathlength[current];
                    predecessor[i]=current;
                    if(!ispresent_in_queue[i])
                        {
                            insert_queue(i);
                            ispresent_in_queue[i]=true;
                        }
                }

            }

        }


    }
    return 1;
}

void initilize_queue()
```

```
{
   int i;
   for(i=0;i<max;i++)
   {
      queue[i]=0;
   }
   rear=-1;
   front=-1;
}

void insert_queue(int added_item)
{
   if(rear==max-1)
   {
      cout<<"queue is overflow"<<endl;
      exit(1);
   }
   else
   {
      if(front==-1)
         front=0;
      rear+=1;

      queue[rear]=added_item;
   }


}
```

```
int is_empty_queue()
{
   if((front==-1)||(front>rear))
      return 1;
   else
      return 0;
}

int delete_queue()
{
   int d;
   if(is_empty_queue())
   {
      cout<<"queue is underflow"<<endl;
      exit(1);
   }
   else
   {
      d=queue[front];
      front=front+1;
   }
   return d;
}

int findpath(int s,int v)
{
   int i,u;
   int path[max];
   int shortdist=0;
   int count=0;
   while(v!=s)
   {
      count++;
      path[count]=v;
      u=predecessor[v];
      shortdist+=adj[u][v];
      v=u;
   }
   count++;
   path[count]=s;
   cout<<"shortest path is "<<endl;
   for(i=count;i>=1;i--)
   {
      cout<<path[i];
   }
   cout<<endl;
   cout<<"shortest distance is = "<<shortdist<<endl;
   return 0;
}
```



```
vansh@vansh-HP-Laptop-15s-fq5xxx:~/DAA_Lab$ g++ belmon_ford_17.cpp
vansh@vansh-HP-Laptop-15s-fq5xxx:~/DAA_Lab$ ./a.out
enter vertices of graph
3
enter the origin and destination of graph
0
0
enter the weight
6
enter the origin and destination of graph
0
1
enter the weight
7
enter the origin and destination of graph
0
2
enter the weight
10
enter the origin and destination of graph
1
1
enter the weight
1
enter the origin and destination of graph
1
2
enter the weight
-11
enter the origin and destination of graph
2
0
enter the weight
5
6  7  10
0  1  -11
5  0  0
enter the source vertex
0
enter destination vertex : -1 for exit
2
shortest path is
012
shortest distance is = -4
enter destination vertex : -1 for exit
1
shortest path is
01
shortest distance is = 7
enter destination vertex : -1 for exit
-1
```

DAA LAB

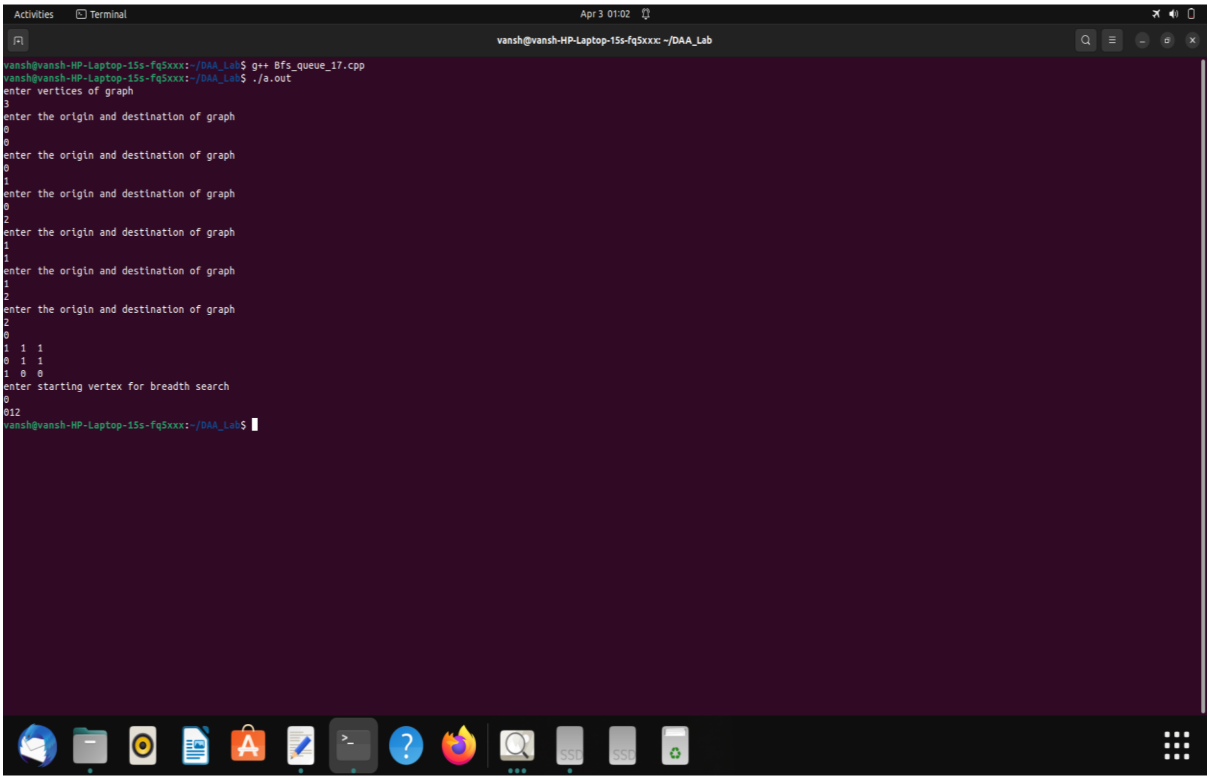
Title: Breadth First Search Using Queue

```
#include<iostream>
using namespace std;

# define MAX 100

# define initial 1
#define waiting 2
#define visited 3

int n;
int adj [MAX] [MAX];
int state[MAX];

void create_graph();
void BF_Traversal ();
void BFS(int v);

int queue [MAX],front = -1, rear = -1;
void insert_queue(int vertex);
int delete_queue ();
int isEmpty_queue();

int main()
{
    create_graph();
    BF_Traversal();
}

void create_graph()
{
    int max_e,i,origin,destination,j;
    cout<<"enter vertices of graph"<<endl;
    cin>>n;
    max_e=n*(n-1);
    for(i=0;i<max_e;i++)
    {
        cout<<"enter the origin and destination of graph"<<endl;
        cin>>origin>>destination;
        adj[origin][destination]=1;
    }
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            cout<<adj[i][j]<<"  ";
        }
        cout<<endl;
    }
}

void BF_Traversal()
```

```
{
    int v;
    for(v=0;v<n;v++)
        state[v]=initial;
    cout<<"enter starting vertex for breadth search"<<endl;
    cin>>v;
    BFS(v);
}
void BFS (int v)
{
    int i;
    insert_queue(v);
    state[v]=waiting;
    while(!isEmpty_queue())
    {
        v=delete_queue();
        cout<<v;
        state[v]=visited;
        for(i=0;i<n;i++)
        {
            if(adj[v][i]==1&&state[i]==initial)
            {
                insert_queue(i);
                state[i]=waiting;
            }
        }
    }
```

```
   cout<<endl;
}
void insert_queue(int vertex)
{
   if(rear==MAX-1)
      cout<<"queue is overflow"<<endl;
   else
   {
      if(front==-1)
         front=0;
      rear+=1;
      queue[rear]=vertex;
   }
}
int isEmpty_queue()
{
   if(front==-1||front>rear)
      return 1;
   else
      return 0;
}
int delete_queue()
{
   int del_item;
   if(front==-1||front>rear)
   {
      cout<<"queue is underflow"<<endl;
```

```
        exit(1);
    }
    del_item=queue[front];
    front+=1;
    return del_item;
}
```



```
vansh@vansh-HP-Laptop-15s-fq5xxx:~/DAA_Lab$ g++ Bfs_queue_17.cpp
vansh@vansh-HP-Laptop-15s-fq5xxx:~/DAA_Lab$ ./a.out
enter vertices of graph
3
enter the origin and destination of graph
0
0
enter the origin and destination of graph
0
1
enter the origin and destination of graph
0
2
enter the origin and destination of graph
1
1
enter the origin and destination of graph
1
2
enter the origin and destination of graph
2
0
1  1  1
0  1  1
1  0  0
enter starting vertex for breadth search
0
012
vansh@vansh-HP-Laptop-15s-fq5xxx:~/DAA_Lab$ 
```

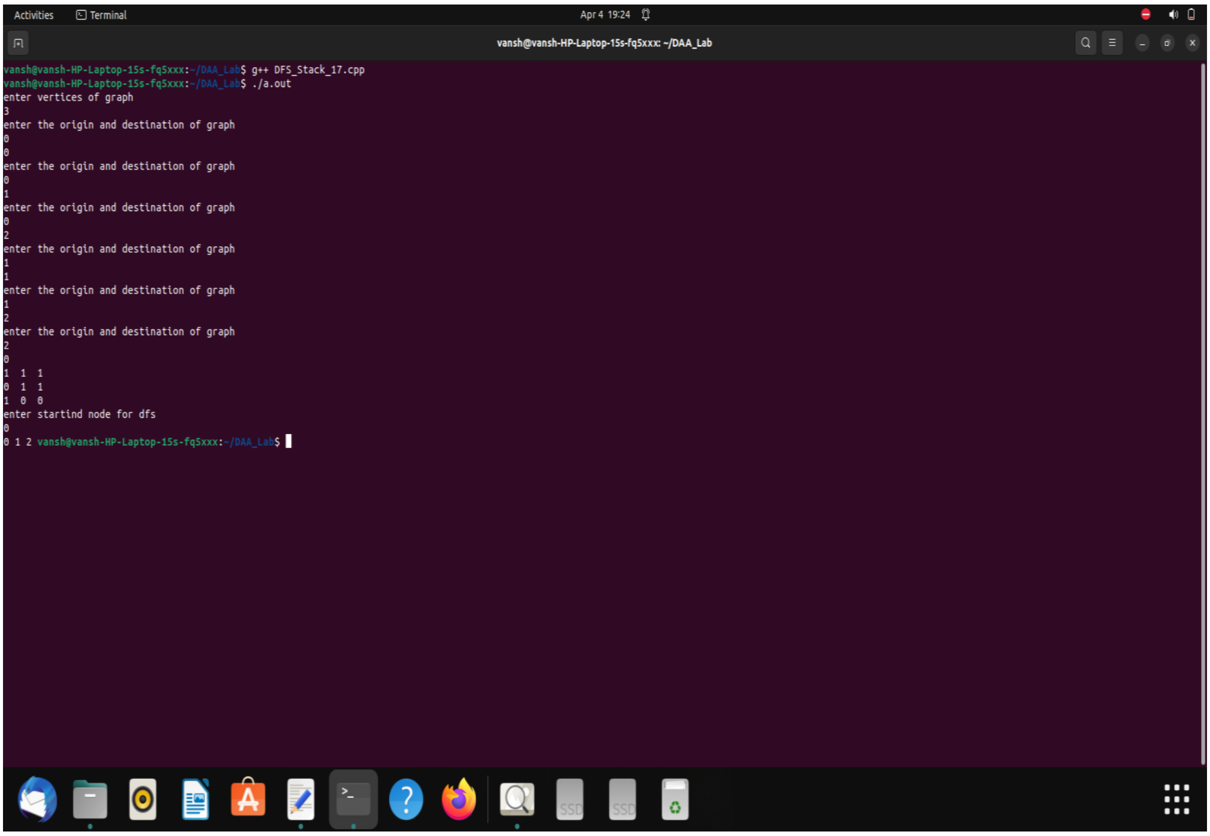
DAA LAB

Title: Depth First Search Using Stack

```
#include <iostream>

using namespace std;

#define max 100
#define initial 1
#define visited 2;

int n;
int adj[max][max];
int state[max];
void create_graph();
void df_traversal();
void dfs(int v);

int stack[max];
int top=-1;
void push(int v);
int pop();
int isEmpty_stack();

int main()
{
   create_graph();
   df_traversal();
```

```
    return 0;
}

void create_graph()
{
    int max_e,i,origin,destination,j;
    cout<<"enter vertices of graph"<<endl;
    cin>>n;
    max_e=n*(n-1);
    for(i=0;i<max_e;i++)
    {
        cout<<"enter the origin and destination of graph"<<endl;
        cin>>origin>>destination;
        adj[origin][destination]=1;
    }
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            cout<<adj[i][j]<<"  ";
        }
        cout<<endl;
    }
}

void df_traversal()
{
    int v;
    for(v=0;v<n;v++)
    {
        state[v]=initial;
    }
    cout<<"enter startind node for dfs"<<endl;
    cin>>v;
    dfs(v);
}

void dfs(int v)
{
    int i;
    push(v);
    while(!isEmpty_stack())
    {
        v=pop();
        if(state[v]==initial)
        {
            cout<<v<<" ";
            state[v]=visited;
        }
        for(i=n-1;i>=0;i--)
        {
            if(adj[v][i]==1&&state[i]==initial)
                push(i);
        }
    }
}
int pop()
{
    int v;
    if(top==-1)
    {
        cout<<"stack underflow"<<endl;
        exit(1);
    }
    else
    {
    v=stack[top];
    top=top-1;
    return v;
    }
}
void push(int v)
{
    if(top==(max-1))
    {
        cout<<"Stack is Overflow"<<endl;
        return;
    }
    top+=1;
    stack[top]=v;
}
```

```
int isEmpty_stack()
{
 if(top==-1)
      return 1;
 else
   return 0;
}
```



```
vansh@vansh-HP-Laptop-15s-fq5xxx:~/DAA_Lab$ g++ DFS_Stack_17.cpp
vansh@vansh-HP-Laptop-15s-fq5xxx:~/DAA_Lab$ ./a.out
enter vertices of graph
3
enter the origin and destination of graph
0
0
enter the origin and destination of graph
0
1
enter the origin and destination of graph
0
2
enter the origin and destination of graph
1
1
enter the origin and destination of graph
1
2
enter the origin and destination of graph
2
0
1  1  1
0  1  1
1  0  0
enter startind node for dfs
0
0 1 2 vansh@vansh-HP-Laptop-15s-fq5xxx:~/DAA_Lab$ 
```

DAA LAB

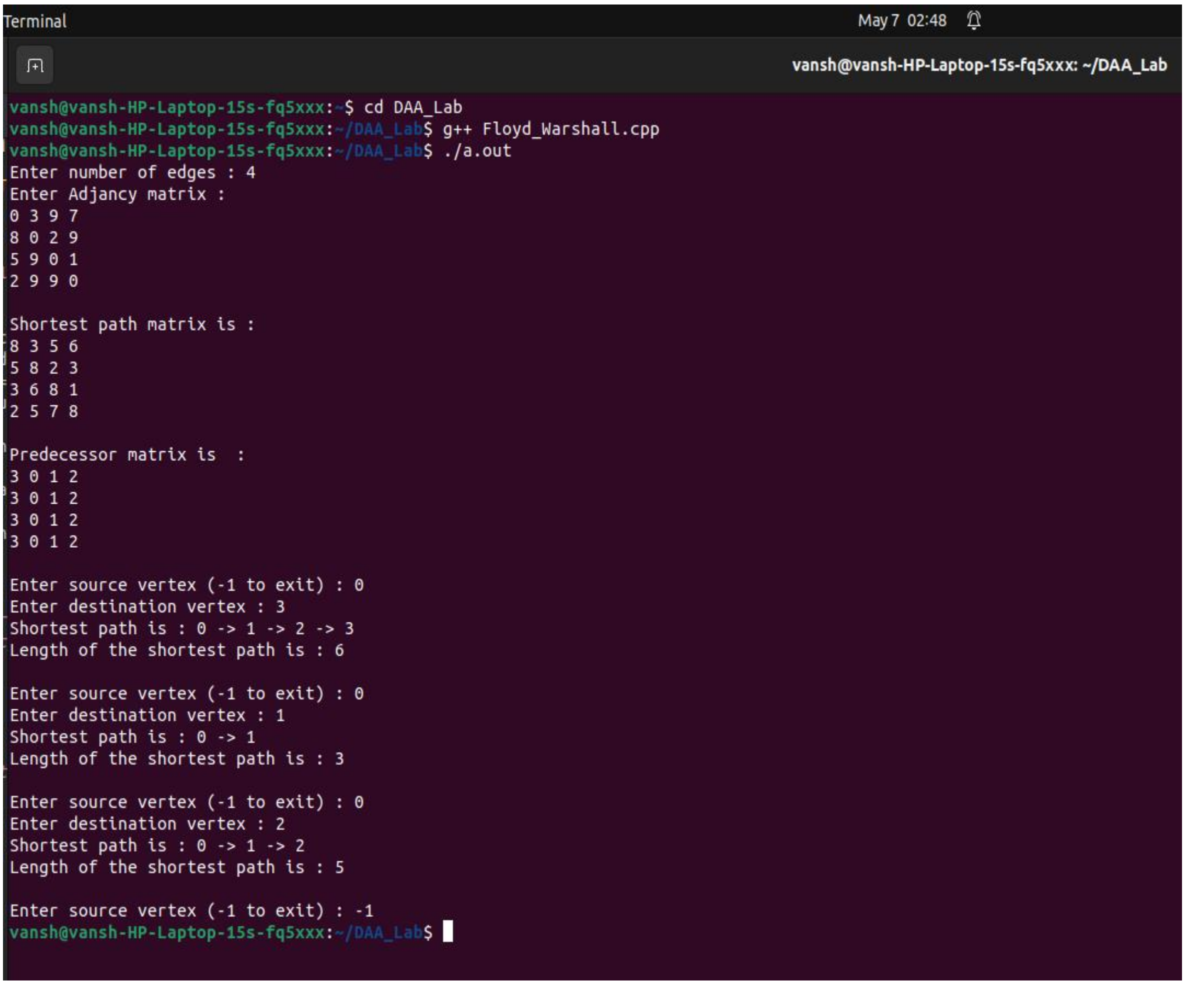
Title: All Pair Shortest Path Algorithm (Floyd-Warshall Algorithm)

```
#include<iostream>
using namespace std;
#define infinity 9999
#define MAX 100
int n;              //Number of vertices
int adj[MAX][MAX];    //Weighted Adjancy matrix
int D[MAX][MAX];      //Shortest path matrix
int pred[MAX][MAX];   //Predecessor matrix
void create_graph();
void floyd_warshalls();
void findpath(int, int);
void display(int m[MAX][MAX], int);
int main()
{
 int s, d;
 create_graph();     //Called function for taking graph as a input
 floyd_warshalls();  //Called function to perform Floyd-Warshall's algorithm

 while(1)
 {
  cout<<"\nEnter source vertex (-1 to exit) : ";
  cin>>s;
  if(s==-1)
  {
   break;
```

```
    }
    cout<<"Enter destination vertex : ";
    cin>>d;

    if(s<0 || s>n-1 || d<0 || d>n-1)
    {
      cout<<"Enter valid vertices\n\n";
      continue;
    }

    cout<<"Shortest path is : ";
    findpath(s, d);
    cout<<"Length of the shortest path is : "<<D[s][d]<<endl;
  }

  return 0;
}

//Function taking graph as an input
void create_graph()
{
  int o,d;
  cout<<"Enter number of edges : ";
  cin>>n;
  cout<<"Enter Adjancy matrix :\n";
  for(o=0; o<n; o++)
    for(int d=0; d<n; d++)
```

```
      cin>>adj[o][d];
}

//Function implementing Floyd-Warshall's algoritm
void floyd_warshalls()
{
  int i, j, k;
  for(i=0; i<n; i++)
  {
    for(j=0; j<n; j++)
    {
      if(adj[i][j]==0)
      {
        D[i][j] = infinity;
        pred[i][j] = -1;
      }
      else
      {
        D[i][j] = adj[i][j];
        pred[i][j] = i;
      }
    }
  }

  for(k=0; k<n; k++)
  {
    for(i=0; i<n; i++)
  {
   for(j=0; j<n; j++)
   {
    if(D[i][k] + D[k][j] < D[i][j])
    {
     D[i][j] = D[i][k] + D[k][j];
     pred[i][j] = pred[k][j];
    }
   }
  }
 }

 cout<<"\nShortest path matrix is :\n";
 display(D, n);
 cout<<"\nPredecessor matrix is  :\n";
 display(pred, n);

 for(i=0; i<n; i++)
 {
  if(D[i][j]<0)
  {
   cout<<"Error : negative cycle\n";
   exit(1);
  }
 }
}
```

```
//Function displays the matrix
void display(int m[MAX][MAX], int n)
{
 int i, j;
 for(i=0; i<n; i++)
 {
  for(j=0; j<n; j++)
  {
   cout<<m[i][j]<<" ";
  }
  cout<<"\n";
 }
}

//Function finds path from source to destination
void findpath(int s, int d)
{
 int i, path[MAX], count;
 if(D[s][d]==infinity)
 {
  cout<<"There is no path between "<<s<<" to "<<d<<"\n";
  return;
 }

 count = -1;
 do
 {
   path[++count] = d;
   d = pred[s][d];
  }while(d!=s);
  path[++count] = s;

  for(i=count; i>0; i--)
  {
   cout<<path[i]<<" -> ";
  }
  cout<<path[i]<<endl;
}
```



```
vansh@vansh-HP-Laptop-15s-fq5xxx:~$ cd DAA_Lab
vansh@vansh-HP-Laptop-15s-fq5xxx:~/DAA_Lab$ g++ Floyd_Warshall.cpp
vansh@vansh-HP-Laptop-15s-fq5xxx:~/DAA_Lab$ ./a.out
Enter number of edges : 4
Enter Adjancy matrix :
0 3 9 7
8 0 2 9
5 9 0 1
2 9 9 0

Shortest path matrix is :
8 3 5 6
5 8 2 3
3 6 8 1
2 5 7 8

Predecessor matrix is  :
3 0 1 2
3 0 1 2
3 0 1 2
3 0 1 2

Enter source vertex (-1 to exit) : 0
Enter destination vertex : 3
Shortest path is : 0 -> 1 -> 2 -> 3
Length of the shortest path is : 6

Enter source vertex (-1 to exit) : 0
Enter destination vertex : 1
Shortest path is : 0 -> 1
Length of the shortest path is : 3

Enter source vertex (-1 to exit) : 0
Enter destination vertex : 2
Shortest path is : 0 -> 1 -> 2
Length of the shortest path is : 5

Enter source vertex (-1 to exit) : -1
vansh@vansh-HP-Laptop-15s-fq5xxx:~/DAA_Lab$
```

DAA LAB

Title : Minimum Cost Spanning Tree of a given connected undirected graph using Prim's Algorithm.

```
#include <iostream>

using namespace std;


// Define constants

#define MAX 10          // Maximum number of vertices

#define TEMP 0          // Temporary status for vertices

#define PERM 1          // Permanent status for vertices

#define infinity 999    // Infinity value for distances

#define NIL -1          // Represents no predecessor


// Structure to represent an edge

struct edge

{

  int u;    // Source vertex of the edge

  int v;    // Destination vertex of the edge

};


// Global variables

int n;                  // Number of vertices

int adj[MAX][MAX];       // Adjacency matrix

int predecessor[MAX];    // Predecessor array for vertices

int status[MAX];         // Status array for vertices (TEMP or PERM)
int length[MAX];          // Distance array for vertices

// Function prototypes
void create_graph();
void maketree(int r, struct edge tree[MAX]);
int min_temp();

int main()
{
  int wt_tree = 0;
  int i, root;
  struct edge tree[MAX];

  // Create the graph
  create_graph();

  // Input root vertex from user
  cout << "Enter root vertex : ";
  cin >> root;

  // Generate the minimum spanning tree
  maketree(root, tree);

  // Display the edges of the spanning tree
  cout << "Edges to be included in spanning tree are : \n ";
  for (i = 1; i <= n - 1; i++)
  {
    cout << tree[i].u << "  ";
    cout << tree[i].v;
    cout << "\n";
    wt_tree += adj[tree[i].u][tree[i].v]; // Calculate total weight of the tree
  }
  cout << "Weight of spanning tree is :" << wt_tree;

  return 0;
}

// Function to generate the minimum spanning tree
void maketree(int r, struct edge tree[MAX])
{
  int current, i;
  int count = 0;

  // Initialize predecessor, length, and status arrays
  for (i = 0; i < n; i++)
  {
    predecessor[i] = NIL;
    length[i] = infinity;
    status[i] = TEMP;
  }

  // Set length of root vertex to 0
  length[r] = 0;
```

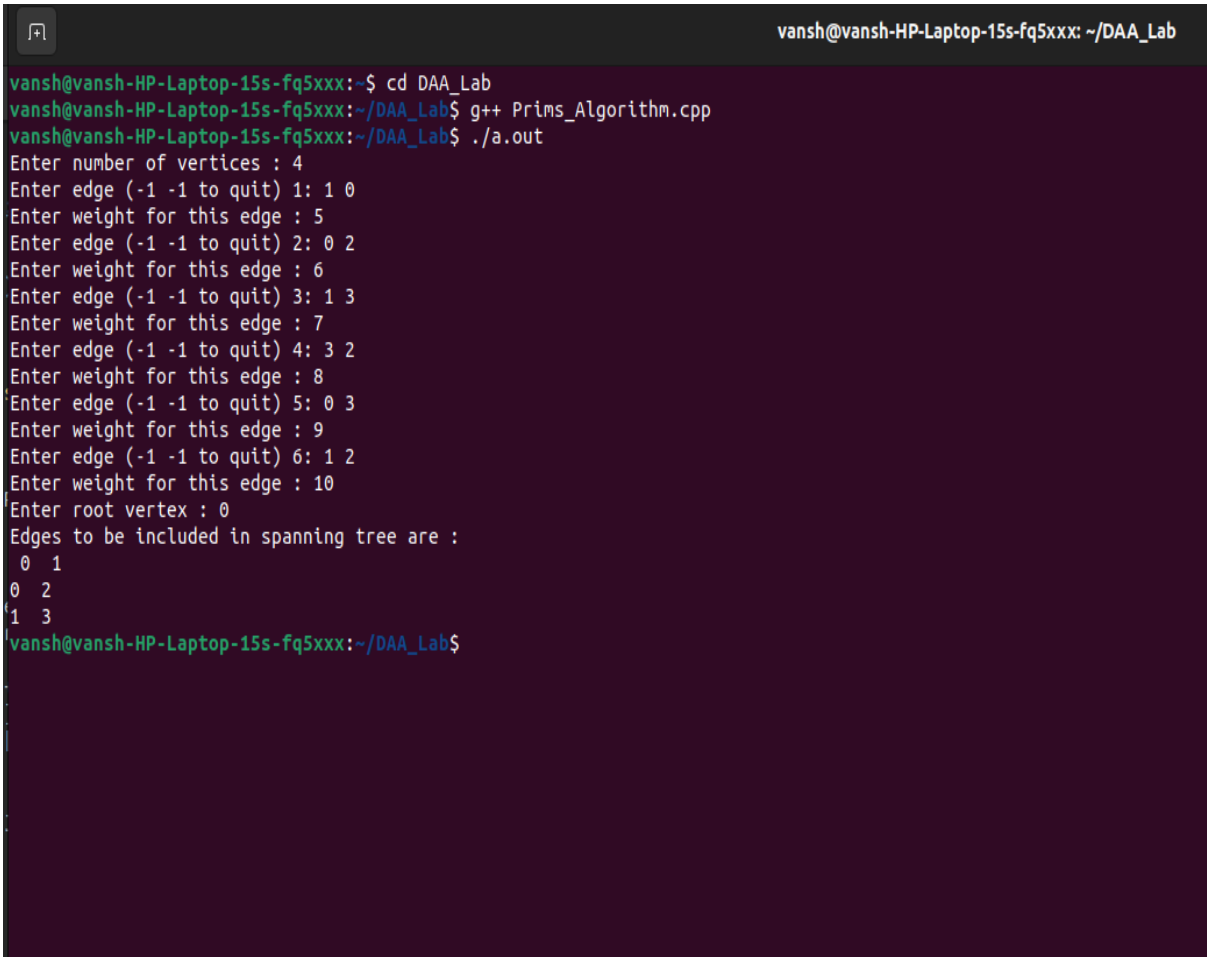
```
// Loop until all vertices are visited
while (1)
{
 // Find the vertex with minimum temporary length
 current = min_temp();

 // If all vertices are permanently visited or graph is disconnected
 if (current == NIL)
 {
  if (count == n - 1)
   return;
  else
  {
   cout << "Graph is not connected , No spanning tree is possible \n";
   exit(1);
  }
 }

 // Mark the current vertex as permanent
 status[current] = PERM;

 // If current vertex is not the root, add edge to tree
 if (current != r)
 {
  count++;
  tree[count].u = predecessor[current];
  tree[count].v = current;
```

```
    }

    // Update lengths and predecessors of adjacent vertices
    for (i = 0; i < n; i++)
    {
      if (adj[current][i] > 0 && status[i] == TEMP)
      {
        if (adj[current][i] < length[i])
        {
          predecessor[i] = current;
          length[i] = adj[current][i];
        }
      }
    }
  }
}

// Function to find the vertex with minimum temporary length
int min_temp()
{
  int i;
  int min = infinity;
  int k = -1;

  // Iterate through all vertices
  for (i = 0; i < n; i++)
  {
```

```
  // If vertex is temporary and its length is less than current minimum
  if (status[i] == TEMP && length[i] < min)
  {
   min = length[i];  // Update minimum length
   k = i;            // Update index of vertex
  }
 }
 return k;  // Return index of vertex
}

// Function to create the graph
void create_graph()
{
 int i, max_edges, origin, destin, wt;

 // Input number of vertices from user
 cout << "Enter number of vertices : ";
 cin >> n;

 // Calculate maximum possible edges
 max_edges = n * (n - 1) / 2;

 // Input edges and weights from user
 for (i = 1; i <= max_edges; i++)
 {
  cout << "Enter edge (-1 -1 to quit) " << i << ": ";
  cin >> origin >> destin;
```

```
    // If input is (-1 -1), exit loop
    if ((origin == -1) && (destin == -1))
      break;

    // Input weight for the edge
    cout << "Enter weight for this edge : ";
    cin >> wt;

    // Check for valid edge inputs
    if (origin >= n || destin >= n || origin < 0 || destin < 0)
    {
      cout << "Invalid edge! \n";
      i--;
    }
    else
    {
      // Update adjacency matrix with weight for the edge
      adj[origin][destin] = wt;
      adj[destin][origin] = wt;
    }
  }
}
```

DAA LAB



```
vansh@vansh-HP-Laptop-15s-fq5xxx:~$ cd DAA_Lab
vansh@vansh-HP-Laptop-15s-fq5xxx:~/DAA_Lab$ g++ Prims_Algorithm.cpp
vansh@vansh-HP-Laptop-15s-fq5xxx:~/DAA_Lab$ ./a.out
Enter number of vertices : 4
Enter edge (-1 -1 to quit) 1: 1 0
Enter weight for this edge : 5
Enter edge (-1 -1 to quit) 2: 0 2
Enter weight for this edge : 6
Enter edge (-1 -1 to quit) 3: 1 3
Enter weight for this edge : 7
Enter edge (-1 -1 to quit) 4: 3 2
Enter weight for this edge : 8
Enter edge (-1 -1 to quit) 5: 0 3
Enter weight for this edge : 9
Enter edge (-1 -1 to quit) 6: 1 2
Enter weight for this edge : 10
Enter root vertex : 0
Edges to be included in spanning tree are :
 0  1
0  2
1  3
vansh@vansh-HP-Laptop-15s-fq5xxx:~/DAA_Lab$
```