An O(m+nK) implementation of Dijkstra's algorithm

Single Source Shortest Path

```
#include <iostream>
using namespace std;
#include<stdio.h>
#include<stdlib.h>
#include<limits.h>
#include<map>
#include<fstream>
#include<chrono>
using namespace std::chrono;
#define SIZE 25
struct edge
{
    int i,j;
    edge *next;
};
struct ets
{
    struct edge *curr;
};
void insert(ets *e,int i,int j)
{
    struct edge *d=(struct edge*)malloc(sizeof(edge));
    d->i=i;
    d->j=j;
    d->next=NULL;
    if(e->curr==NULL)
    {
        e->curr=d;
        return;
    }
    edge *t=e->curr;
    while(t->next!=NULL)
    {
        t=t->next;
    }
    t->next=d;
}
int argmin(int f[],int k)
{
    int i,m=0;
    for(i=1;i<k;i++)
    {
        if(f[i]<f[m])
```

```
            m=i;
    }
    return m;
}
void print(int d[],int pred[],int V)
{
   ofstream mf;
   mf.open("output1.txt");
   mf<<"Vertex Distance from Source Parent\n";
   for (int i = 0; i < V; i++)
      mf<<i<<" \t\t "<<d[i]<<" \t\t\t   "<<pred[i]<<"\n";
    mf.close();
}

void New_Dijkstra(int g[][SIZE],int src,map <int,int> l,int V)
{
    //initialize
    int d[V],pred[V];
    bool s[V];
    for(int i=0;i<V;i++)
    {
        d[i]=INT_MAX;pred[i]=-1;s[i]=false;
    }
    d[src]=0;s[src]=true;
    int k=l.size();
    int f[k];
    ets e[k];
    for(int i=0;i<k;i++)
    {
        e[i].curr=NULL;
    }
    for(int v=0;v<V;v++)
    {
        if(g[src][v])
        {
            insert(&e[l[g[src][v]]],src,v);
        }
    }

    for(int i=0;i<k;i++)
    {
        //update
        edge *t=e[i].curr;
        if(t!=NULL){
            if(!s[t->j])
            {
                f[i]=d[t->i]+g[t->i][t->j];
                continue;
            }
            while(s[t->j]&&t->next!=NULL)
            {
                t=t->next;
                e[i].curr=t;
            }
            if(!s[t->j])
            {
                f[i]=d[t->i]+g[t->i][t->j];
            }
            else
            {
                e[i].curr=NULL;
                f[i]=INT_MAX;
            }
        }
        else
        {
            f[i]=INT_MAX;
        }
    }
    for(int count=0;count<V-1;count++)
    {
        int r=argmin(f,k);
        edge *t=e[r].curr;
        d[t->j]=d[t->i]+g[t->i][t->j];
        pred[t->j]=t->i;
        s[t->j]=true;
        for(int v=0;v<V;v++)
        {
            if(g[t->j][v])
            {
                insert(&e[l[g[t->j][v]]],t->j,v);
            }
        }
        for(int i=0;i<k;i++)
        {
            //update
            edge *t=e[i].curr;
            if(t!=NULL){
                if(!s[t->j])
                {
                    f[i]=d[t->i]+g[t->i][t->j];
                    continue;
                }
                while(s[t->j]&&t->next!=NULL)
```

```
                    {
                        t=t->next;
                        e[i].curr=t;
                    }
                    if(!s[t->j])
                    {
                        f[i]=d[t->i]+g[t->i][t->j];
                    }
                    else
                    {
                        e[i].curr=NULL;
                        f[i]=INT_MAX;
                    }
                }
            }
        }
    print(d,pred,V);
}
int main() {
    int V,E;
    std::fstream dfile("input1.txt",std::ios_base::in);
    dfile>>V>>E;
    int k;
    dfile>>k;
    int g[SIZE][SIZE];
    for(int i=0;i<V;i++)
    {
        for(int j=0;j<V;j++)
        {
            dfile>>g[i][j];
        }
    }
    map <int,int> l;
    int t;
    for(int i=0;i<k;i++)
    {
        dfile>>t;
        l[t]=i;
    }
    auto start=high_resolution_clock::now();
    New_Dijkstra(g,0,l,V);
    auto end = high_resolution_clock::now();
    duration<double> diff = end-start;
    cout<< diff.count() << " s\n";
    dfile.close();
    return 0;
}
```

All Pair Shortest Path

```
#include <iostream>
using namespace std;
#include<stdio.h>
#include<stdlib.h>
#include<limits.h>
#include<map>
#include<fstream>
#include<chrono>
using namespace std::chrono;
#define SIZE 25
struct edge
{
    int i,j;
    edge *next;
};
struct ets
{
    struct edge *curr;
};
void insert(ets *e,int i,int j)
{
    struct edge *d=(struct edge*)malloc(sizeof(edge));
    d->i=i;
    d->j=j;
    d->next=NULL;
    if(e->curr==NULL)
    {
        e->curr=d;
        return;
    }
    edge *t=e->curr;
    while(t->next!=NULL)
    {
        t=t->next;
    }
    t->next=d;
}
int argmin(int f[],int k)
{
    int i,m=0;
    for(i=1;i<k;i++)
    {
        if(f[i]<f[m])
            m=i;
    }
    return m;
}
void print(int d[],int pred[],int V,int src)
{
   ofstream mf;
   mf.open("o1.txt",std::ios_base::app);
   mf<<"From "<<src<<":\n";
   mf<<"Vertex Distance from Source Parent\n";
   for (int i = 0; i < V; i++)
      mf<<i<<" \t\t "<<d[i]<<" \t\t\t   "<<pred[i]<<"\n";
    mf.close();
}

void New_Dijkstra(int g[][SIZE],int src,map <int,int> l,int V)
{
    //initialize
    int d[V],pred[V];
    bool s[V];
    for(int i=0;i<V;i++)
    {
        d[i]=INT_MAX;pred[i]=-1;s[i]=false;
    }
    d[src]=0;s[src]=true;
    int k=l.size();
    int f[k];
    ets e[k];
    for(int i=0;i<k;i++)
    {
        e[i].curr=NULL;
    }
    for(int v=0;v<V;v++)
    {
        if(g[src][v])
        {
            insert(&e[l[g[src][v]]],src,v);
        }
    }

    for(int i=0;i<k;i++)
    {
        //update
        edge *t=e[i].curr;
        if(t!=NULL){
            if(!s[t->j])
            {
                f[i]=d[t->i]+g[t->i][t->j];
                continue;
            }
            while(s[t->j]&&t->next!=NULL)
            {
                t=t->next;
                e[i].curr=t;
            }
            if(!s[t->j])
            {
                f[i]=d[t->i]+g[t->i][t->j];
            }
            else
            {
                e[i].curr=NULL;
                f[i]=INT_MAX;
            }
        }
        else
        {
            f[i]=INT_MAX;
        }
    }
    for(int count=0;count<V-1;count++)
    {
        int r=argmin(f,k);
        edge *t=e[r].curr;
        d[t->j]=d[t->i]+g[t->i][t->j];
        pred[t->j]=t->i;
        s[t->j]=true;
        for(int v=0;v<V;v++)
        {
            if(g[t->j][v])
            {
                insert(&e[l[g[t->j][v]]],t->j,v);
            }
        }
        for(int i=0;i<k;i++)
        {
            //update
            edge *t=e[i].curr;
            if(t!=NULL){
                if(!s[t->j])
                {
                    f[i]=d[t->i]+g[t->i][t->j];
                    continue;
                }
                while(s[t->j]&&t->next!=NULL)
                {
```

```
                        t=t->next;
                        e[i].curr=t;
                    }
                    if(!s[t->j])
                    {
                        f[i]=d[t->i]+g[t->i][t->j];
                    }
                    else
                    {
                        e[i].curr=NULL;
                        f[i]=INT_MAX;
                    }
                }
            }
        }
        print(d,pred,V,src);
    }
    int main() {
        int V,E,k;
        std::fstream dfile("input1.txt",std::ios_base::in);
        dfile>>V>>E>>k;
        int g[SIZE][SIZE];
        for(int i=0;i<V;i++)
        {
            for(int j=0;j<V;j++)
            {
                dfile>>g[i][j];
            }
        }
        map <int,int> l;
        int t;
        for(int i=0;i<k;i++)
        {
            dfile>>t;
            l[t]=i;
        }
        auto start=high_resolution_clock::now();
        for(int i=0;i<V;i++)
        {
            New_Dijkstra(g,i,l,V);
        }
        auto end = high_resolution_clock::now();
        duration<double> diff = end-start;
        cout<< diff.count() << " s\n";
        dfile.close();
        return 0;
    }
```

A faster algorithm if K is permitted to grow with problem size

Single Source Shortest Path

```
#include <iostream>
using namespace std;
#include<stdio.h>
#include<stdlib.h>
#include<limits.h>
#include<map>
#include<queue>
#include<fstream>
#include<chrono>
using namespace std::chrono;
#define SIZE 25
struct edge
{
    int i,j;
    edge *next;
};
struct ets
{
    struct edge *curr;
};
struct dl
{
    int i,f;
};
struct compare
{
  bool operator()(dl& lhs,dl& rhs)
   {
      return lhs.f>rhs.f;
   }
};
void insert(ets *e,int i,int j)
{
    struct edge *d=(struct edge*)malloc(sizeof(edge));
    d->i=i;
    d->j=j;
    d->next=NULL;
    if(e->curr==NULL)
    {
        e->curr=d;
        return;
    }
    edge *t=e->curr;
```

```
        while(t->next!=NULL)
        {
            t=t->next;
        }
        t->next=d;
    }
    int argmin(priority_queue<dl,vector<dl>,compare > m[],int
    g[][SIZE],bool s[],ets e[],int d[],int h,int b[][SIZE])
    {
        int i,min;
        for(int c=0;c<h;c++)
        {
            if(m[c].empty())
                continue;
            dl x=m[c].top();
            while(s[e[x.i].curr->j])
            {
                m[c].pop();
                int i=x.i;
                //update
                edge *t=e[i].curr;
                if(t!=NULL){
                    while(s[t->j]&&t->next!=NULL)
                    {
                        t=t->next;
                        e[i].curr=t;
                    }
                    if(!s[t->j])
                    {
                        dl a;
                        a.f=d[t->i]+g[t->i][t->j];
                        a.i=i;
                        m[c].push(a);
                        b[t->i][t->j]=1;
                    }
                    else
                    {
                        e[i].curr=NULL;
                    }
                }
                if(m[c].empty()) break;
                x=m[c].top();
            }
        }
        for(i=0;i<h;i++)
            if(!m[i].empty())
            {
                min=i;
                break;
            }
        for(i=0;i<h;i++)
        {
            if((!m[i].empty())&&(m[i].top().f<m[min].top().f))
                min=i;
        }
        return m[min].top().i;
}
void print(int d[],int pred[],int V)
{
    ofstream mf;
    mf.open("output2.txt");
    mf<<"Vertex Distance from Source Parent\n";
    for (int i = 0; i < V; i++)
        mf<<i<<" \t\t "<<d[i]<<" \t\t\t   "<<pred[i]<<"\n";
     mf.close();
}
void New_Dijkstra1 (int g[][SIZE],int src,map <int,int> l,int
V,int E)
{
     //initialize
     int d[V],pred[V];
     bool s[V];
     int b[SIZE][SIZE]={{0}};
     for(int i=0;i<V;i++)
     {
         d[i]=INT_MAX;pred[i]=-1;s[i]=false;
     }
     d[src]=0;s[src]=true;
     int k=l.size();
     int q=V*k/E;
     int h=k/q;
     ets e[k];
     for(int i=0;i<k;i++)
     {
         e[i].curr=NULL;
     }
     for(int v=0;v<V;v++)
     {
         if(g[src][v])
         {
             insert(&e[l[g[src][v]]],src,v);
         }
     }
     priority_queue<dl,vector<dl>,compare > m[h];
    for(int i=0;i<k;i++)
    {
        edge *t=e[i].curr;
        if(t!=NULL){
            dl a;
            a.f=d[t->i]+g[t->i][t->j];
            a.i=i;
            m[i/q].push(a);
            b[t->i][t->j]=1;
        }
    }
    for(int count=0;count<V-1;count++)
    {
        int r=argmin(m,g,s,e,d,h,b);
        edge *t=e[r].curr;
        d[t->j]=d[t->i]+g[t->i][t->j];
        pred[t->j]=t->i;
        s[t->j]=true;
        for(int v=0;v<V;v++)
        {
            if(g[t->j][v])
            {
                insert(&e[l[g[t->j][v]]],t->j,v);
            }
        }
        for(int i=0;i<k;i++)
        {
            edge *t=e[i].curr;
            if(t!=NULL&&!b[t->i][t->j]){
                dl a;
                a.f=d[t->i]+g[t->i][t->j];
                a.i=i;
                m[i/q].push(a);
                b[t->i][t->j]=1;
            }
        }
    }
    print(d,pred,V);
}
int main() {
    int V,E;
    std::fstream dfile("input2.txt",std::ios_base::in);
    dfile>>V>>E;
    int k;
    dfile>>k;
    int q=V*k/E;
    if(k%q!=0)
    {
        printf("k should be divisible by V*k/E");
        return 0;
    }
    int g[SIZE][SIZE];
    for(int i=0;i<V;i++)
    {
        for(int j=0;j<V;j++)
        {
            dfile>>g[i][j];
        }
    }
    map <int,int> l;
    int t;
    for(int i=0;i<k;i++)
    {
        dfile>>t;
        l[t]=i;
    }
    auto start=high_resolution_clock::now();
    New_Dijkstra1(g,0,l,V,E);
    auto end = high_resolution_clock::now();
    duration<double> diff = end-start;
    cout<< diff.count() << " s\n";
    dfile.close();
    return 0;
}
```

All Pair Shortest Path

```
#include <iostream>
using namespace std;
#include<stdio.h>
#include<stdlib.h>
#include<limits.h>
#include<map>
#include<queue>
#include<fstream>
#include<chrono>
using namespace std::chrono;
#define SIZE 25
struct edge
{
    int i,j;
    edge *next;
};
struct ets
{
    struct edge *curr;
};
struct dl
{
    int i,f;
};
struct compare
{
  bool operator()(dl& lhs,dl& rhs)
   {
     return lhs.f>rhs.f;
   }
};
void insert(ets *e,int i,int j)
{
    struct edge *d=(struct edge*)malloc(sizeof(edge));
    d->i=i;
    d->j=j;
    d->next=NULL;
    if(e->curr==NULL)
    {
        e->curr=d;
        return;
    }
    edge *t=e->curr;
    while(t->next!=NULL)
    {
        t=t->next;
    }
    t->next=d;
}
int argmin(priority_queue<dl,vector<dl>,compare > m[],int
g[][SIZE],bool s[],ets e[],int d[],int h,int b[][SIZE])
{
    int i,min;
    for(int c=0;c<h;c++)
    {
        if(m[c].empty())
            continue;
        dl x=m[c].top();
        while(s[e[x.i].curr->j])
        {
            m[c].pop();
            int i=x.i;
            //update
            edge *t=e[i].curr;
            if(t!=NULL){
                while(s[t->j]&&t->next!=NULL)
                {
                    t=t->next;
                    e[i].curr=t;
                }
                if(!s[t->j])
                {
                    dl a;
                    a.f=d[t->i]+g[t->i][t->j];
                    a.i=i;
                    m[c].push(a);
                    b[t->i][t->j]=1;
                }
                else
                {
                    e[i].curr=NULL;
                }
            }
            if(m[c].empty()) break;
            x=m[c].top();
        }
    }
    for(i=0;i<h;i++)
        if(!m[i].empty())
        {
            min=i;
            break;
        }
    for(i=0;i<h;i++)
    {
        if((!m[i].empty())&&(m[i].top().f<m[min].top().f))
            min=i;
    }
    return m[min].top().i;
}
void print(int d[],int pred[],int V,int src)
{
   ofstream mf;
   mf.open("o2.txt",std::ios_base::app);
   mf<<"From "<<src<<":\n";
   mf<<"Vertex Distance from Source Parent\n";
   for (int i = 0; i < V; i++)
      mf<<i<<" \t\t "<<d[i]<<" \t\t\t   "<<pred[i]<<"\n";
    mf.close();
}
void New_Dijkstra1(int g[][SIZE],int src,map <int,int> l,int
V,int E)
{
    //initialize
    int d[V],pred[V];
    bool s[V];
    int b[SIZE][SIZE]={{0}};
    for(int i=0;i<V;i++)
    {
        d[i]=INT_MAX;pred[i]=-1;s[i]=false;
    }
    d[src]=0;s[src]=true;
    int k=l.size();
    int q=V*k/E;
    int h=k/q;
    ets e[k];
    for(int i=0;i<k;i++)
    {
        e[i].curr=NULL;
    }
    for(int v=0;v<V;v++)
    {
        if(g[src][v])
        {
            insert(&e[l[g[src][v]]],src,v);
        }
    }
    priority_queue<dl,vector<dl>,compare > m[h];
    for(int i=0;i<k;i++)
    {
        edge *t=e[i].curr;
        if(t!=NULL){
            dl a;
            a.f=d[t->i]+g[t->i][t->j];
            a.i=i;
            m[i/q].push(a);
            b[t->i][t->j]=1;
        }
    }
    for(int count=0;count<V-1;count++)
    {
        int r=argmin(m,g,s,e,d,h,b);
        edge *t=e[r].curr;
        d[t->j]=d[t->i]+g[t->i][t->j];
        pred[t->j]=t->i;
        s[t->j]=true;
        for(int v=0;v<V;v++)
        {
            if(g[t->j][v])
            {
                insert(&e[l[g[t->j][v]]],t->j,v);
            }
        }
        for(int i=0;i<k;i++)
        {
            edge *t=e[i].curr;
            if(t!=NULL&&!b[t->i][t->j]){
                dl a;
                a.f=d[t->i]+g[t->i][t->j];
                a.i=i;
                m[i/q].push(a);
                b[t->i][t->j]=1;
            }
        }
    }
    print(d,pred,V,src);
}
int main() {
    int V,E;
    std::fstream dfile("input2.txt",std::ios_base::in);
    dfile>>V>>E;
    int k;
    dfile>>k;
    int q=V*k/E;
    if(k%q!=0)
    {
        printf("k should be divisible by V*k/E");
        return 0;
    }
    int g[SIZE][SIZE];
    for(int i=0;i<V;i++)
    {
        for(int j=0;j<V;j++)
        {
            dfile>>g[i][j];
        }
    }
    map <int,int> l;
    int t;
    for(int i=0;i<k;i++)
    {
        dfile>>t;
        l[t]=i;
    }
    auto start=high_resolution_clock::now();
    for(int i=0;i<V;i++)
    {
        New_Dijkstra1(g,i,l,V,E);
    }
    auto end = high_resolution_clock::now();
    duration<double> diff = end-start;
    cout<< diff.count() << " s\n";
    dfile.close();
    return 0;
}
```

Dijkstra

Single Source Shortest Path

```
#include<stdio.h>
#include<limits.h>
#define SIZE 25
#include<fstream>
#include<iostream>
#include<chrono>
using namespace std::chrono;
using namespace std;
int findmin(int d[],bool s[],int V)
{
   int min=INT_MAX,min_index;
   for(int v=0;v<V;v++)
     if(s[v]==false&&d[v]<=min)
     {
        min=d[v];
        min_index=v;
     }
   return min_index;
}
void print(int d[],int pred[],int V)
{
   ofstream mf;
   mf.open("op2.txt");
   mf<<"Vertex Distance from Source Parent\n";
   for (int i = 0; i < V; i++)
      mf<<i<<" \t\t "<<d[i]<<" \t\t\t   "<<pred[i]<<"\n";
    mf.close();
}
void Dijkstra(int g[][SIZE], int src,int V)
{
     int d[V];
     bool s[V];
     int pred[V];
     for(int i=0;i<V;i++)
     {
         d[i]=INT_MAX;
         pred[i]=-1;
         s[i]=false;
     }
     d[src]=0;
     for(int count=0;count<V-1;count++)
     {
       int u=findmin(d,s,V);
```

```
        s[u]=true;
        for(int v=0;v<V;v++)
          if(!s[v]&&g[u][v]&&d[u]!=INT_MAX&&d[u]+g[u][v]<d[v])
          {
              d[v]=d[u]+g[u][v];
              pred[v]=u;
          }
      }
      print(d,pred,V);
}
int main()
{
    int V,E;
    std::fstream file("input1.txt",std::ios_base::in);
    file>>V>>E;
    int k;
    file>>k;
    int g[SIZE][SIZE];
    for(int i=0;i<V;i++)
    {
        for(int j=0;j<V;j++)
        {
            file>>g[i][j];
        }
    }
    auto start=high_resolution_clock::now();
    Dijkstra(g,0,V);
    auto end = high_resolution_clock::now();
    duration<double> diff = end-start;
    cout<< diff.count() << " s\n";
    file.close();
    return 0;
}
```

All Pair Shortest Path

```
#include<stdio.h>
#include<limits.h>
#define SIZE 25
#include<fstream>
#include<iostream>
#include<chrono>
using namespace std::chrono;
using namespace std;
int findmin(int d[],bool s[],int V)
{
   int min=INT_MAX,min_index;
   for(int v=0;v<V;v++)
     if(s[v]==false&&d[v]<=min)
     {
         min=d[v];
         min_index=v;
     }
   return min_index;
}
void print(int d[],int pred[],int V,int src)
{
   ofstream mf;
   mf.open("op1.txt",std::ios_base::app);
   mf<<"From "<<src<<":\n";
   mf<<"Vertex Distance from Source Parent\n";
   for (int i = 0; i < V; i++)
      mf<<i<<" \t\t "<<d[i]<<" \t\t\t   "<<pred[i]<<"\n";
    mf.close();
}
void Dijkstra(int g[][SIZE], int src,int V)
{
     int d[V];
     bool s[V];
     int pred[V];
     for(int i=0;i<V;i++)
     {
         d[i]=INT_MAX;
         pred[i]=-1;
         s[i]=false;
     }
     d[src]=0;
     for(int count=0;count<V-1;count++)
     {
       int u=findmin(d,s,V);
       s[u]=true;
```

```
        for(int v=0;v<V;v++)
          if(!s[v]&&g[u][v]&&d[u]!=INT_MAX&&d[u]+g[u][v]<d[v])
          {
              d[v]=d[u]+g[u][v];
              pred[v]=u;
          }
      }
      print(d,pred,V,src);
}
int main()
{
    int V,E,k;
    std::fstream file("input1.txt",std::ios_base::in);
    file>>V>>E>>k;
    int g[SIZE][SIZE];
    for(int i=0;i<V;i++)
    {
        for(int j=0;j<V;j++)
        {
            file>>g[i][j];
        }
    }
    auto start=high_resolution_clock::now();
    for(int i=0;i<V;i++)
    {
        Dijkstra(g,i,V);
    }
    auto end = high_resolution_clock::now();
    duration<double> diff = end-start;
    cout<< diff.count() << " s\n";
    file.close();
    return 0;
}
```