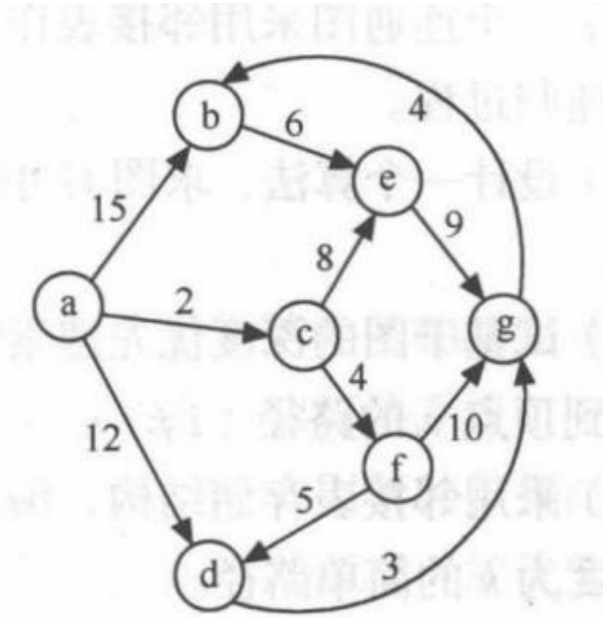
homework10nd

应用题

- (4)
  - 

终点&D	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$
b	$15:(a,b)$	$15:(a,b)$	$15:(a,b)$	$15:(a,b)$	$15:(a,b)$	$15:(a,b)$
c	$2:(a,c)$	$2:(a,c)$	$2:(a,c)$	$2:(a,c)$	$2:(a,c)$	$2:(a,c)$
d	$12:(a,d)$	$12:(a,d)$	$11:(a,c,f,d)$	$11:(a,c,f,d)$	$11:(a,c,f,d)$	$11:(a,c,f,d)$
e	∞	$10:(a,c,e)$	$10:(a,c,e)$	$10:(a,c,e)$	$10:(a,c,e)$	$10:(a,c,e)$
f	∞	$6:(a,c,f)$	$6:(a,c,f)$	$6:(a,c,f)$	$6:(a,c,f)$	$6:(a,c,f)$
g	∞	∞	$16:(a,c,f,g)$	$16:(a,c,f,g)$	$14:(a,c,f,d,g)$	$14:(a,c,f,d,g)$
S终点集	$\{a,c\}$	$\{a,c,f\}$	$\{a,c,f,e\}$	$\{a,c,f,e,d\}$	$\{a,c,f,e,d,g\}$	$\{a,c,f,e,d,g,b\}$

算法设计题

- (3)
  - ```
#include<bits/stdc++.h>
using namespace std;
void ERROR(const string& s){
    cerr<<"ERROR!!!"<<s<<endl;
    exit(114514);
}
template<class T>class Heap{
    int mxsz,sz;
    T* val;
    void Up(){
```
```

```
        int p=sz;
        T key=val[p];
        while(p>>1&&val[p>>1]<key){
            val[p]=val[p>>1];
            p>>=1;
        }val[p]=key;
    }
    void Down(){
        int p=1;
        T key=val[p];
        while((p<<1)<=sz){
            if(key<val[p<<1]&&((p<<1|1)>sz||val[p<<1|1]
<val[p<<1])){
                val[p]=val[p<<1];
                p<<=1;
            }else if((p<<1|1)<=sz&&key<val[p<<1|1]){
                val[p]=val[p<<1|1];
                p=p<<1|1;
            }else break;
        }val[p]=key;
    }
public:
    Heap():mxsz(0),sz(0),val(NULL){}
    Heap(int sz):mxsz(sz),sz(0),val(new T[sz+1]){}
    ~Heap(){delete[] val;}
    bool empty()const{return !sz;}
    void Push(const T& v){
        if(sz==mxsz)ERROR("The heap is full!");
        val[++sz]=v;
        Up();
    }
    void Pop(){
        if(!sz)ERROR("The heap is empty!");
        val[1]=val[sz--];
        if(sz)Down();
    }
    T Top()const{
        if(!sz)ERROR("The heap is empty!");
        return val[1];
    }
};
template<class T>class Graph{
    int vecs,arcs;
    struct Arc{
        int vec;
        T dis;
        Arc* next;
        Arc(int vec,T dis,Arc*
next):vec(vec),dis(dis),next(next){}
```

```
        ~Arc(){free(next);}
    };
    Arc** head;
public:
    Graph():vecs(0),head(NULL){}
    Graph(int vecs,int arcs):vecs(vecs),arcs(arcs),head(new
Arc*[vecs]){
        for(int i=0;i<vecs;i++)head[i]=NULL;
    }
    ~Graph(){delete[] head;}
    int Vecs()const{return vecs;}
    int Arcs()const{return arcs;}
    void AddArc(int u,int v,const T& d){
        Arc* arc=new Arc(v,d,head[u]);
        head[u]=arc;
    }
    const void* GetNextArc(int u,const void* arc)const{
        if(u>=vecs||u<0)ERROR("Unknow vec!");
        if(!arc)return (void*)head[u];
        return ((Arc*)arc)->next;
    }
    int GetVec(const void* arc)const{
        if(!arc)ERROR("Can't get the vec from NULL!");
        return ((Arc*)arc)->vec;
    }
    T GetDis(const void* arc)const{
        if(!arc)ERROR("Can't get the dis from NULL!");
        return ((Arc*)arc)->dis;
    }
};
namespace Dijkstra{
    template<class T>struct node{
        int v;
        T d;
        node(){}
        node(int v,T d):v(v),d(d){}
        bool operator <(const node& _)const{
            return d>_.d;
        }
    };
    template<class T>void solve(const Graph<T>& G,T* dis,int
s,T inf){
        for(int i=0;i<G.Vecs();i++)dis[i]=inf;
        Heap<node<T> >q(G.Arcs());
        q.Push(node<T>(s,dis[s]=0));
        while(!q.empty()){
            node<T> cur=q.Top();q.Pop();
            int u=cur.v;
            if(dis[u]<cur.d)continue;
```

```
            const void* edge=NULL;
            while(edge=G.GetNextArc(u,edge)){
                int v=G.GetVec(edge);
                T d=G.GetDis(edge);
                if(dis[u]+d<dis[v]){
                    q.Push(node<T>(v,dis[v]=dis[u]+d));
                }
            }
        }
    }
}
int main(){
    int vecs,arcs,s;
    cin>>vecs>>arcs>>s;
    Graph<int>G(vecs,arcs);
    int u,v,d;
    while(arcs--){
        cin>>u>>v>>d;
        G.AddArc(u,v,d);
    }
    int* dis=new int[vecs];
    Dijkstra::solve(G,dis,s,INT_MAX);
    int mx=s;
    for(int i=0;i<vecs;i++)if(dis[i]>dis[mx])mx=i;
    cout<<mx<<" dis="<<dis[mx];
    delete[] dis;
}
/*
7 11 0
0 1 15
0 2 2
0 3 12
1 4 6
2 4 8
2 5 4
3 6 3
4 6 9
5 3 5
5 6 10
6 1 4
*/
```

- **(5)**

```
#include<bits/stdc++.h>
using namespace std;
void ERROR(const string& s){
    cerr<<"ERROR!!!"<<s<<endl;
    exit(114514);
```

```
}
template<class T>class Graph{
    int vecs,arcs;
    struct Arc{
        int vec;
        T dis;
        Arc* next;
        Arc(int vec,T dis,Arc*
next):vec(vec),dis(dis),next(next){}
        ~Arc(){free(next);}
    };
    Arc** head;
public:
    Graph():vecs(0),head(NULL){}
    Graph(int vecs,int arcs):vecs(vecs),arcs(arcs),head(new
Arc*[vecs]){
        for(int i=0;i<vecs;i++)head[i]=NULL;
    }
    ~Graph(){delete[] head;}
    int Vecs()const{return vecs;}
    int Arcs()const{return arcs;}
    void AddArc(int u,int v,const T& d){
        Arc* arc=new Arc(v,d,head[u]);
        head[u]=arc;
    }
    const void* GetNextArc(int u,const void* arc)const{
        if(u>=vecs||u<0)ERROR("Unknow vec!");
        if(!arc)return (void*)head[u];
        return ((Arc*)arc)->next;
    }
    int GetVec(const void* arc)const{
        if(!arc)ERROR("Can't get the vec from NULL!");
        return ((Arc*)arc)->vec;
    }
    T GetDis(const void* arc)const{
        if(!arc)ERROR("Can't get the dis from NULL!");
        return ((Arc*)arc)->dis;
    }
};
void dfs(const Graph<int>& G,int u,int len,int end,bool* instk)
{
    if(len==0&&u==end){
        puts("YES");
        exit(0);
    }
    if(len<=0||u==end)return;
    instk[u]=1;
    const void* edge=NULL;
    while(edge=G.GetNextArc(u,edge)){
```

```
            int v=G.GetVec(edge),d=G.GetDis(edge);
            if(!instk[v])dfs(G,v,len-d,end,instk);
        }instk[u]=0;
}
int main(){
    int vecs,arcs,s,t,k;
    cin>>vecs>>arcs>>s>>t>>k;
    Graph<int>G(vecs,arcs);
    int u,v,d;
    while(arcs--){
        cin>>u>>v>>d;
        G.AddArc(u,v,d);
    }
    bool* instk=new bool[vecs];
    memset(instk,0,sizeof(instk));
    dfs(G,s,k,t,instk);
    puts("NO");
    return 0;
}
/*
7 11 0 1 18
0 1 15
0 2 2
0 3 12
1 4 6
2 4 8
2 5 4
3 6 3
4 6 9
5 3 5
5 6 10
6 1 4
*/
```