1.这一题用dijkstra可以比较容易的解决
2.重要的是各个记录数组的更新
3.一定要注意dijkstra算法每次大循环要检查两次vis数组

1003. Emergency (25)

As an emergency rescue team leader of a city, you are given a special map of your country. The map shows several scattered cities connected by some roads. Amount of rescue teams in each city and the length of each road between any pair of cities are marked on the map. When there is an emergency call to you from some other city, your job is to lead your men to the place as quickly as possible, and at the mean time, call up as many hands on the way as possible.

Input

Each input file contains one test case. For each test case, the first line contains 4 positive integers: N (<= 500) - the number of cities (and the cities are numbered from 0 to N-1), M - the number of roads, C1 and C2 - the cities that you are currently in and that you must save, respectively. The next line contains N integers, where the i-th integer is the number of rescue teams in the i-th city. Then M lines follow, each describes a road with three integers c1, c2 and L, which are the pair of cities connected by a road and the length of that road, respectively. It is guaranteed that there exists at least one path from C1 to C2.

Output

For each test case, print in one line two numbers: the number of different shortest paths between C1 and C2, and the maximum amount of rescue teams you can possibly gather. All the numbers in a line must be separated by exactly one space, and there is no extra space allowed at the end of a line.

Sample Input

```
5 6 0 2
1 2 1 5 3
0 1 1
0 2 2
0 3 1
1 2 1
2 4 1
3 4 1
```

Sample Output

```
2 4
```

- Edit 0:

```
#include <cstdio>
#include <vector>
#include <algorithm>

using std::fill;
using std::vector;

const int maxn=510;
const int INF=0x3fffffff;

int num[maxn]={0};
int weight[maxn];
int graph[maxn][maxn]={{INF}};
int distance[maxn];
int w[maxn]={0};
void dijkstra(int begin_point,const int (&graph)[maxn][maxn],const int node_num,int
(&weight)[maxn]){
    num[begin_point]=1;
    w[begin_point]=weight[begin_point];
    bool visited[maxn]={false};
    fill(distance,distance+maxn,INF);
    distance[begin_point]=0;
    for(int i=0;i<node_num;++i){
        int u=-1;
        int MIN=INF;
        for(int j=0;j<node_num;++j){
            if(visited[j]==false&&MIN>distance[j]){
                MIN=distance[j];
                u=j;
            }
        }
        if(u==-1) return;
        visited[u]=true;
        for(int v=0;v<node_num;++v){
            if(visited[v]==false&&graph[u][v]!=INF){
                if(graph[u][v]+distance[u]<distance[v]){
                    distance[v]=graph[u][v]+distance[u];
                    num[v]=num[u];
                    w[v]=w[u]+weight[v];
                }
                else if(graph[u][v]+distance[u]==distance[v]){
                    if(w[u]+weight[v]>w[v]){
                        w[v]=w[u]+weight[v];
                    }
                    num[v]+=num[u];
                }
            }
        }
    }
}
```

```
void dijkstra(int begin_point,const int (&graph)[maxn][maxn],const int node_num,int
(&weight)[maxn]);
int main(){
    fill(graph[0],graph[0]+maxn*maxn,INF);
    int city_num=0,road_num=0,current_in=0,object=0;
    scanf("%d %d %d %d",&city_num,&road_num,&current_in,&object);
    int value;
    for(int i=0;i<city_num;++i){
        scanf("%d",&value);
        weight[i]=value;
    }
    int id1,id2;
    int len;
    for(int i=0;i<road_num;++i){
        scanf("%d %d",&id1,&id2);
        scanf("%d",&len);
        graph[id1][id2]=len;
        graph[id2][id1]=len;
    }
    dijkstra(current_in,graph,city_num,weight);
    printf("%d %d\n",num[object],w[object]);
    return 0;
}
```

- Edit 1:

```
#include <cstdio>
#include <cstring>
#include <algorithm>

using std::fill;

const int MAXV=510;
const int INF=0x3fffffff;
int n,m,st,ed,G[MAXV][MAXV],weight[MAXV];
int d[MAXV],w[MAXV],num[MAXV];

bool vis[MAXV]={false};

void Dijkstra(int s){
    fill(d,d+MAXV,INF);
    memset(num,0,sizeof(num));
    memset(w,0,sizeof(w));
    d[s]=0;
    num[s]=1;
    w[s]=weight[s];
    for(int i=0;i<n;++i){
        int u=-1;
        int MIN=INF;
        for(int j=0;j<n;++j){
            if(vis[j]==false&&d[j]<MIN){

                u=j;
```

```
                MIN=d[j];
            }
        }

        if(u==-1) return;
        vis[u]=true;
        for(int v=0;v<n;++v){
            if(vis[v]==false&&G[u][v]!=INF){
                if(d[u]+G[u][v]<d[v]){
                    d[v]=d[u]+G[u][v];
                    w[v]=w[u]+weight[v];
                    num[v]=num[u];
                }
                else if(d[u]+G[u][v]==d[v]){
                    if(w[u]+weight[v]>w[v]){
                        w[v]=w[u]+weight[v];
                    }
                    num[v]+=num[u];
                }
            }
        }
    }
}
  int main(){

    scanf("%d%d%d%d",&n,&m,&st,&ed);

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

        scanf("%d",&weight[i]);

    }

    int u,v;

    fill(G[0],G[0]+MAXV*MAXV,INF);

    for(int i=0;i<m;++i){

        scanf("%d%d",&u,&v);

        scanf("%d",&Gu);

        Gv=Gu;

    }

    Dijkstra(st);

    printf("%d %d\n",num[ed],w[ed]);

    return 0;
```

```
    }
```

* Edit 2:

  ```c++
  /*
   *用nowHander记录最短路径上收集到的人手，如果可以在最短的前提下
   *收集到更多的人手，更新
   *优化路径长度时，最短路径的数目num为上一节点最短路径的数目
   *如果出现相同的路径长度，当前节点的路径数目+上一节点的路径数目
   *一定要记住，在dijkstra算法的大循环中，每循环一次，会检查两次visited
   */
  #include <vector>
  #include <string>
  #include <iostream>
  #include <algorithm>
  #include <cstring>
  #include <cstdio>
  using std::fill;
  using std::vector;
  using std::string;
  using std::cin;
  using std::cout;
  using std::endl;

  const int maxn=510;
  const int INF=0x3fffffff;
  int graph[maxn][maxn];
  bool visited[maxn];
  int distance[maxn];
  int hander[maxn];
  int num[maxn];
  int nowHander[maxn];
  void dijkstra(int begin,int ncount){
    memset(visited,false,sizeof(visited));
    fill(distance,distance+maxn,INF);
    fill(nowHander,nowHander+maxn,0);
    fill(num,num+maxn,0);
    distance[begin]=0;
    num[begin]=1;
    nowHander[begin]=hander[begin];
    for(int i=0;i<ncount;++i){
        int u=-1;
        int minDis=INF;
        for(int j=0;j<ncount;++j){
            if(visited[j]==false&&minDis>distance[j]){
                u=j;

                minDis=distance[j];
```

```
            }
        }
        if(u==-1){
            return;
        }
        visited[u]=true;
        for(int v=0;v<ncount;++v){
            if(visited[v]==false&&graph[u][v]!=INF){
                if(distance[u]+graph[u][v]<distance[v]){
                    distance[v]=distance[u]+graph[u][v];
                    num[v]=num[u];
                    nowHander[v]=nowHander[u]+hander[v];
                }
                else if(distance[u]+graph[u][v]==distance[v]){
                    if(nowHander[u]+hander[v]>nowHander[v]){
                        nowHander[v]=nowHander[u]+hander[v];
                    }
                    num[v]+=num[u];
                }
            }
        }
    }
}

int main(){
    int point_count,road_count,begin,end;
    scanf("%d%d%d%d",&point_count,&road_count,&begin,&end);
    for(int i=0;i<point_count;++i){
        scanf("%d",&hander[i]);
    }

    fill(graph[0],graph[0]+maxn*maxn,INF);
    int id1,id2;
    int howLong;
    for(int i=0;i<road_count;++i){
        scanf("%d %d %d",&id1,&id2,&howLong);
        graph[id1][id2]=howLong;
        graph[id2][id1]=howLong;
    }
    dijkstra(begin,point_count);
    printf("%d %d\n",num[end],nowHander[end]);
    return 0;

}
```