山东大学 计算机科学与技术 学院

数据结构与算法 课程实验报告

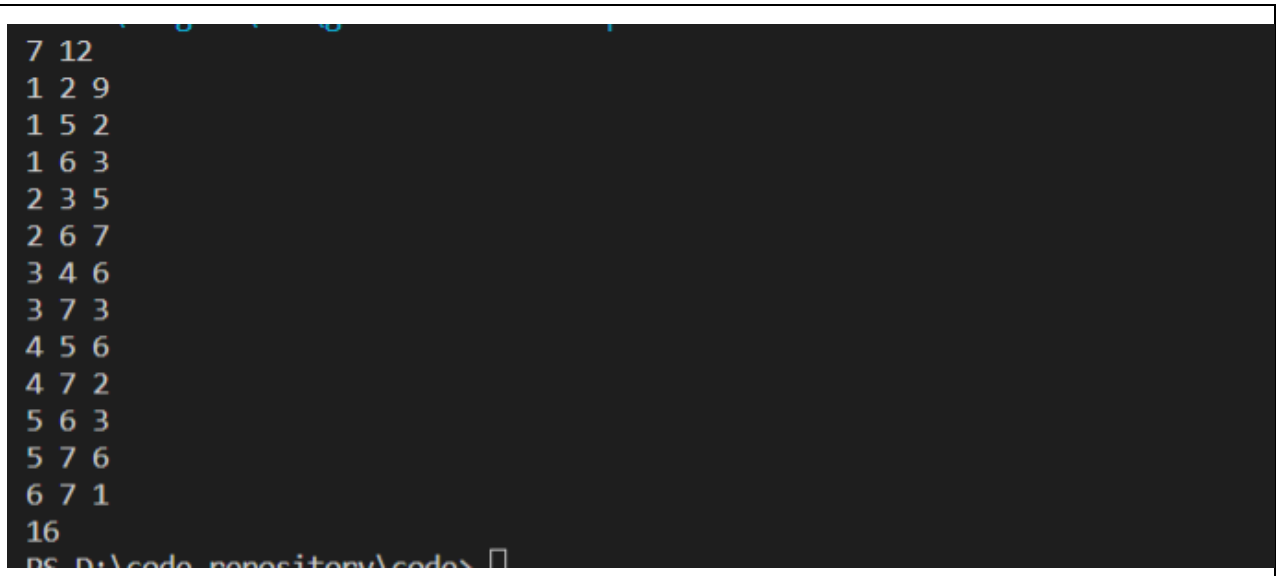
学号：202200130048	姓名： 陈静雯	班级： 6
实验题目：最小生成树		
实验学时：2	实验日期： 12.20	
实验目的： 使用 prim 算法实现最小生成树 使用 kruskal 算法实现最小生成树		
软件开发工具： Vscode		

1. 实验内容
使用 prim 算法实现最小生成树
使用 kruskal 算法实现最小生成树

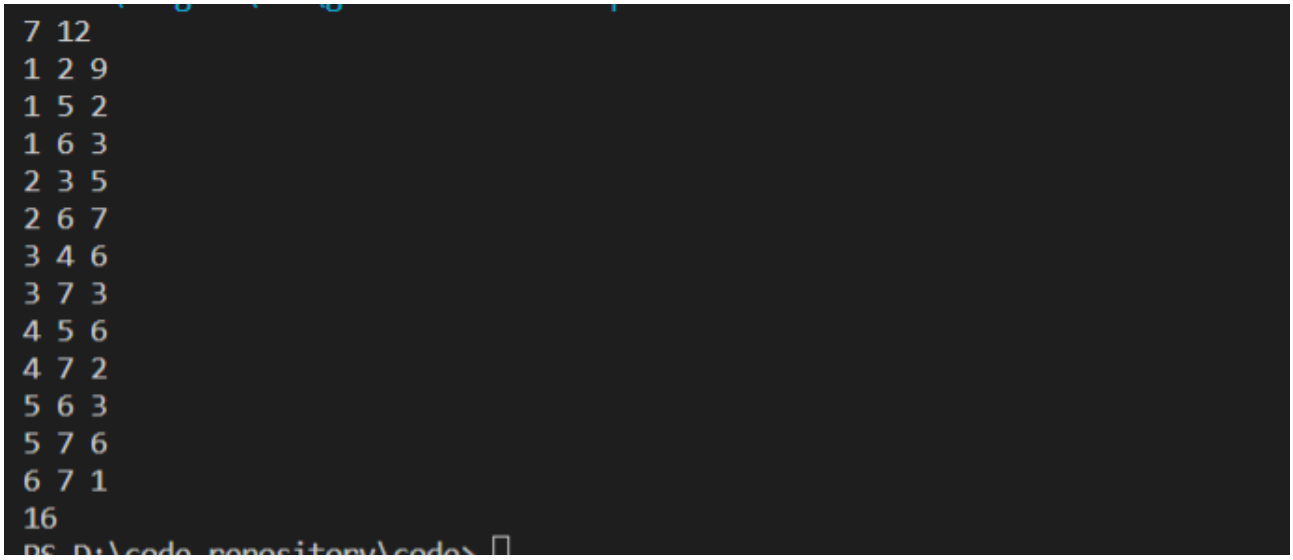
2. 数据结构与算法描述 （整体思路描述，所需要的数据结构与算法）
（1）prim 算法
存图，用数组指针，每个点相邻的边都用一个数组来存
先把点 1 相邻的边都放入最小堆，再 pop 出一个边权最小的边且该条边的另一个端点是未被访问过的，把边权加到最后的结果里，然后更新堆，把 pop 出的边的另一端点相邻的边且那条边的端点未被访问，都 push 入堆，直到找到的边为点数-1，即最小生成树，输出 ans

（2）kruskal 算法
把所有边都放入最小堆，每次 pop 堆顶，即边权最小的边，若两个端点不在同一个子树里，则把边加入结果集，判断是否在同一颗子树通过并查集查找，即两个点的 father 是否是同一个，如果边可以放入结果集，则更新其中一个点的 father，把它们并到同一个子树里，直到边=点-1，即找到最小生成树为止

3. 测试结果（测试输入，测试输出）
（1）Prim



（2）Kruskal



4. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）

时间超时：prim 一个堆优化即可，kruskal 在 find father 时要进行路径压缩，找的同时更新 father

要考虑 array 开的空间，以及堆开辟的空间，

且 ans 是 long long

5. 附录：实现源代码（本实验的全部源程序代码，程序风格清晰易理解，有充分的注释）

（1）prim 算法

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

struct edge{
    int u,v,weight;
    bool operator>(const edge b) const{    //运算符重载
        return weight>b.weight;
```

```
    }
    bool operator>=(const edge b) const{
        return weight>=b.weight;
    }
    bool operator<(const edge b) const{
        return weight<b.weight;
    }
    bool operator<=(const edge b) const{
        return weight<=b.weight;
    }
    bool operator==(const edge b) const{
        return weight==b.weight;
    }
};

template <class T>
class myarray{
public:
    myarray(int capacity=10);
    void insert(T& newelement);
    int arraysize(){
        return size;
    }
    ~myarray(){
        delete [] element;
        size=0;
        length=0;
    }
    T* element;
private:
    int length;
    int size;
};

template <class T>
myarray<T>::myarray(int capacity){   //构造函数
    element = new T[capacity];
    length=capacity;
    size=0;
}

template <class T>
void myarray<T>::insert(T& newelement){   //在数组最后插入新元素
    if(size>=length){                     //若空间不够，重新进行动态分配
        length+=5000;
        T* temp = new T [length];
```

```
        for(int i=0;i<size;i++){
            temp[i]=element[i];
        }
        delete [] element;
        element=temp;
    }
    element[size++]=newelement;     //队尾插入
}

template<class T>
class minheap{
public:
    minheap(){ element=NULL; heapsize=0; length=0;}
    void push(const T thelement);
    void pop();
    bool empty(){ return heapsize==0;}
    T top(){ return element[1];}
private:
    T* element;
    int heapsize;
    int length;
};

template <class T>
void minheap<T>::push(const T thelement){
    if(heapsize>=length-1){                    //空间扩大
        length+=5000;
        T* temp = new T [length];
        for(int i=1;i<=heapsize;i++){
            temp[i]=element[i];
        }
        delete [] element;
        element=temp;
    }
    int cur = ++heapsize;
    while(cur!=1 && element[cur/2]>thelement){//根节点比插入的元素大，根节点下放
        element[cur]=element[cur/2];
        cur/=2;
    }
    element[cur]=thelement;
}

template<class T>
void minheap<T>::pop(){
    if(heapsize==0) return ;
    T thelement = element[heapsize--];    //删除根节点，把最后一个叶节点往上放
```

```
    int cur = 1;
    int child = 2;
    while(child<=heapsize){
        if(child<heapsize && element[child]>element[child+1]) child+=1;
        if(thelement<=element[child]) break;
        element[cur]=element[child];
        cur=child;
        child*=2;
    }
    element[cur]=thelement;
}

int check[500005]={0};

void prim(myarray<edge>* edg, int n, int e){
    long long ans=0;
    int edge_tree=0;
    minheap<edge> H;
    check[1]=1;
    for(int i=0;i<edg[1].arraysize();i++){          //把点 1 相连的边放入最小堆
        H.push(edg[1].element[i]);
    }
    while(edge_tree<n-1){
        int minn;
        int to;
        while(check[H.top().v]==1) H.pop();      //如果另一个点是被访问过的，就把
边 pop 掉
        minn = H.top().weight;
        to = H.top().v;                          //找到一个新的点且边权是剩下的边
里最小的
        H.pop();
        ans+=minn;
        check[to]=1;
        for(int i=0;i<edg[to].arraysize();i++){          //把新的点相邻的边如果
另一个点未被访问过则放入最小堆
            if(check[edg[to].element[i].v]==0) H.push(edg[to].element[i]);
        }
        edge_tree++;
    }
    printf("%ld",ans);
}

myarray<edge>* edg = new myarray<edge> [500005];

int main(){
    int n,e;
```

```
    scanf("%d%d",&n,&e);
    for(int i=1;i<=e;i++){
        int u,v,w;
        scanf("%d%d%d",&u,&v,&w);
        edge e1,e2;
        e1.weight=w;
        e2.weight=w;
        e1.v=v,e2.v=u;
        e1.u=u,e2.u=v;
        edg[u].insert(e1);                //无向图，一条边看作两条边
        edg[v].insert(e2);
    }
    prim(edg,n,e);
}
```

（2）kruskal，最小堆

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

struct edge{
    int u,v,weight;
    bool operator>(const edge b) const{    //运算符重载
        return weight>b.weight;
    }
    bool operator>=(const edge b) const{
        return weight>=b.weight;
    }
    bool operator<(const edge b) const{
        return weight<b.weight;
    }
    bool operator<=(const edge b) const{
        return weight<=b.weight;
    }
    bool operator==(const edge b) const{
        return weight==b.weight;
    }
};

template<class T>
class minheap{
public:
    minheap(){ element=NULL; heapsize=0; length=10;}
    void init(int n);
    void push(const T& thelement);
    void pop();
```

```
    bool empty(){ return heapsize==0;}
    T top(){ return element[1];}
private:
    T* element;
    int heapsize;
    int length;
};

template <class T>
void minheap<T>::init(int n){
    heapsize=n;
    length=2*n;
    element = new T [500005];
    for(int i=1;i<=heapsize;i++){
        cin>>element[i].u>>element[i].v>>element[i].weight;
    }
    for(int i=heapsize/2;i>=1;i--){             //检查每个根节点与子树的关系
        T temp = element[i];
        int child = 2*i;
        while(child<=heapsize){
            if(child<heapsize && element[child] > element[child+1]) child++;
            if(temp<element[child]) break;                                  //
找到最小的元素
            element[child/2]=element[child];
            child*=2;
        }
        element[child/2]=temp;
    }
}

template <class T>
void minheap<T>::push(const T& thelement){
    if(heapsize==length-1){                  //空间扩大
        length*=2;
        T* temp = new T [length];
        for(int i=1;i<=heapsize;i++){
            temp[i]=element[i];
        }
        element=temp;
    }
    int cur = ++heapsize;
    while(cur!=1 && element[cur/2]>thelement){//根节点比插入的元素大，根节点下放
        element[cur]=element[cur/2];
        cur/=2;
    }
    element[cur]=thelement;
```

```
    cout<<element[1]<<endl;
}

template<class T>
void minheap<T>::pop(){
    if(heapsize==0) return ;
    T thelement = element[heapsize--];    //删除根节点，把最后一个叶节点往上放
    int cur = 1;
    int child = 2;
    while(child<=heapsize){
        if(child<heapsize && element[child]>element[child+1]) child+=1;
        if(thelement<=element[child]) break;
        element[cur]=element[child];
        cur=child;
        child*=2;
    }
    element[cur]=thelement;
}

int father[500005]={0};

int find(int i){
    if(i==father[i]) return i;
    int t=find(father[i]);
    father[i]=t;
    return t;
}

template<class T>
void kruskal(minheap<T> H, int n){
    edge temp;
    unsigned long long ans=0;
    int edge_tree=0;
    while(!H.empty() && edge_tree<n-1){
        temp = H.top();
        H.pop();                          //pop 边权最小的边
        int t1=find(temp.u);
        int t2=find(temp.v);
        if(t1!=t2){                      //如果两个点不在同一个子树，则把这条边放入
生成树边集里，更新两个点之一的 father 值
            father[t1]=t2;
            ans+=temp.weight;
            edge_tree++;
        }
    }
    cout<<ans;
```

```
}

int main(){
    int n,e;
    cin>>n>>e;
    minheap<edge> H;
    H.init(e);
    for(int i=1;i<=n;i++){
        father[i]=i;
    }
    kruskal(H,n);
}
```

（3）kruskal 算法，快排

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

struct edge{
    int u,v,weight;
    bool operator>(const edge b) const{    //运算符重载
        return weight>b.weight;
    }
    bool operator>=(const edge b) const{
        return weight>=b.weight;
    }
    bool operator<(const edge b) const{
        return weight<b.weight;
    }
    bool operator<=(const edge b) const{
        return weight<=b.weight;
    }
    bool operator==(const edge b) const{
        return weight==b.weight;
    }
}edg[500005];

void quicksort(int leftend,int rightend){
    if(leftend>=rightend) return ;
    int leftcur=leftend;
    int rightcur=rightend+1;
    edge pivot=edg[leftend];
    while(1){
        do{
            leftcur++;
        }while(edg[leftcur]<pivot);
        do{
```

```
            rightcur--;
        }while(edg[rightcur]>pivot);
        if(leftcur>=rightcur) break;
        edge temp=edg[leftcur];
        edg[leftcur]=edg[rightcur];
        edg[rightcur]=temp;
    }
    edg[leftend]=edg[rightcur];
    edg[rightcur]=pivot;
    quicksort(leftend,rightcur-1);
    quicksort(rightcur+1,rightend);
}

void quicksort(int n){
    if(n<=0) return ;
    int max=-1;
    int index;
    for(int i=1;i<=n;i++){
        if(edg[i].weight>max){
            max=edg[i].weight;
            index=i;
        }
    }
    edge temp=edg[n];
    edg[n]=edg[index];
    edg[index]=temp;
    quicksort(1,n-1);
}

int father[5000005]={0};

int find(int i){
    if(i==father[i]) return i;
    int t = find(father[i]);  //路径压缩，节省时间
    father[i]=t;
    return t;
}

void kruskal(int n,int e){
    edge temp;
    unsigned long long ans=0;
    int edge_tree=0;
    for(int i=1;i<=e;i++){
        temp = edg[i];
        int t1=find(temp.u);
        int t2=find(temp.v);
```

```
        if(t1!=t2){                    //如果两个点不在同一个子树，则把这条边放入生成树边集里，更新两个点之一的 father 值
            father[t1]=t2;
            ans+=temp.weight;
            edge_tree++;
        }
        if(edge_tree==n-1) break;
    }
    if(edge_tree==n-1) printf("%lu",ans);
}

int main(){
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n,e;
    scanf("%d%d",&n,&e);
    for(int i=1;i<=e;i++){
        scanf("%d%d%d",&edg[i].u,&edg[i].v,&edg[i].weight);
    }
    for(int i=1;i<=n;i++){
        father[i]=i;
    }
    quicksort(e);              //按边权从小到大排序
    kruskal(n,e);
}
```