2023秋《数据结构与算法》大作业报告

院系： 技术科学实验班（智能科学与技术）

学号： 221900180

姓名： 田永铭

OJ(id)： 221900180

一、小蓝鲸的奇妙冒险-第一季

1. 解题思路:

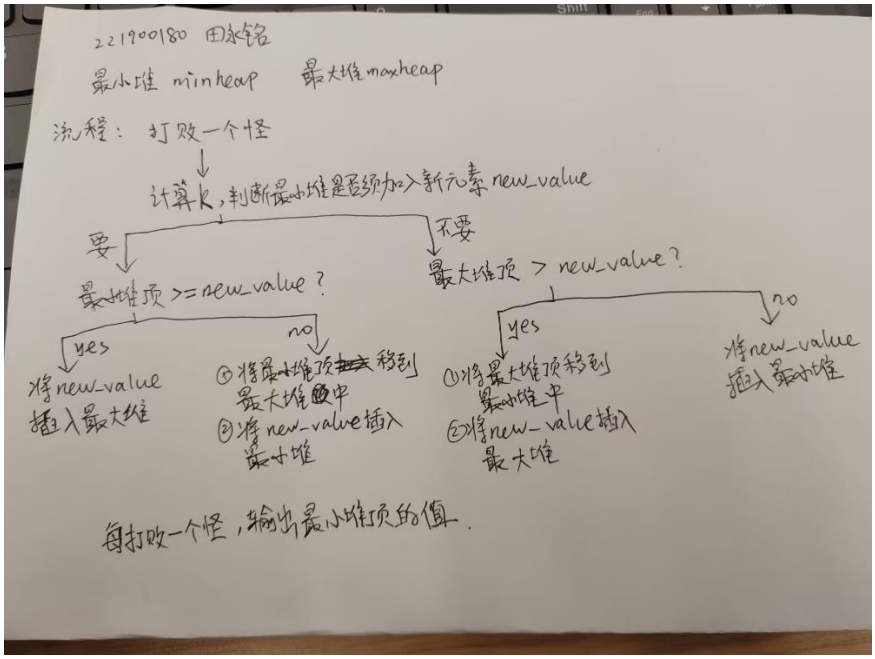
（1）**思路1：利用有偏的快速排序**

这种思路来源于本学期OJ作业第6次《第k大的数》。题目可抽象为不断地加入数据，寻找已加入数据中第k（k可由题目要求简单地算出）大的数字并且输出。可以维护一个数组arr，每次“打怪”，都将该小怪的战力值加进arr，并对arr进行有偏的快速排序，找到arr中第k大的数字，进行输出。但是这种思路效率并不够，不具体展开。

（2）**思路2：利用最大堆和最小堆**

由于快速排序的效率都不足以通过此题，所以要寻找效率更高的做法，考虑能否实现每次打印第k大的数字效率为O（1）？于是想到了堆。此题可以维护两个堆，一个最大堆，一个最小堆。需要使得最小堆顶为第k（k可由题目要求简单地算出）大的数字，所以需要保持最小堆里面有k个元素（已获得的战力值最大的k个），最大堆里面存剩下的已获得的战力值。这样的好处是，很方便从最大堆选取最大的元素加进最小堆，此操作只需要O（1） 。

以下为算法流程图:



2. 核心代码+注释

```
   void move_min_to_max(int added_val) {//从最小堆取到最大堆并将新值加入最小堆
    int save;
    min_heap.remove(save);
    max_heap.insert(save);
    min_heap.insert(added_val);
```

```
}

void move_max_to_min(int added_val) {//从最大堆取到最小堆并将新值加入最大堆
    int save;
    max_heap.remove(save);
    min_heap.insert(save);
    max_heap.insert(added_val);
}

int func(int a, int b) {//计算 k 值
    if (a % b == 0) return a / b;
    else return a / b + 1;
}

int main() {
    int m, n;
    scanf("%d%d", &m, &n);
    for (int i = 0;i < n;i++)
    {
        scanf("%d", &a[i]);
    }

    int min_size = 0;//最小堆元素个数
    for (int i = 0;i < n;i++)
    {
        int mintop = min_heap.top();//最小堆顶
        int maxtop = max_heap.top();//最大堆顶
        int num = func(i + 1, m);

        if (num == min_size)//k 值没变
        {
            if (mintop >= a[i]) max_heap.insert(a[i]);//最小堆顶比新值大，将新值加入最大堆
            else move_min_to_max(a[i]);//反之，将最大顶堆加入最小堆，将新值加入最大堆
        }
        else if (num > min_size)
        {
            min_size++;
            if (maxtop > a[i]) move_max_to_min(a[i]);//最大堆顶元素比新值大，将其加入最小堆，新值加入最大堆
            else min_heap.insert(a[i]);//反之，将新值加入最小堆
        }
```

```
        cout << min_heap.top() << " ";
    }

    return 0;
}
```

其中，最小堆和最大堆的实现完全参考姜远老师课程PPT Lesson5，此处略去。

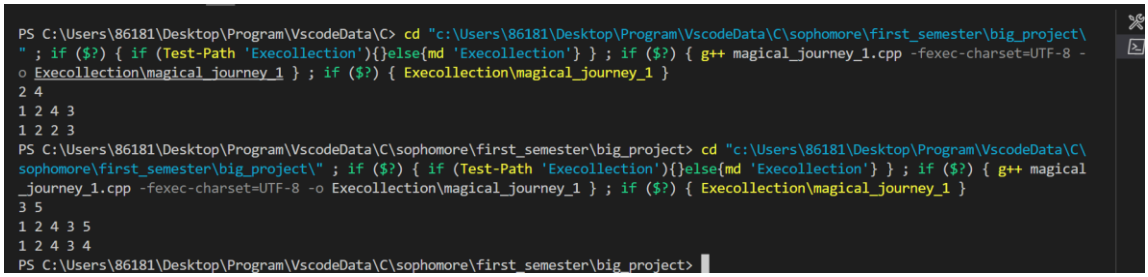
3. 程序（算法）的分析

时间复杂度：O（$n\log_2 n$） 解释：每打败一个怪，堆调整需要O（$\log_2 n$），取答案只要O（1），一共n个怪

空间复杂度：O（n） 解释：即堆中最大存储的怪的个数，是O（n）

4. OJ 运行结果截图

本地测试样例通过截图：



OJ通过截图：

#43747	#56. 小蓝鲸的奇妙冒险 - 第一季	221900180	100	170ms	11276kb	C++	5.4kb	2023-12-19 14:51:09

5.总结：

快速排序虽然很快，但是有时候不能满足需求。堆是一种效率很高的数据结构，在处理“最大”“最小”等等问题的时候非常值得我们利用。算法已经非常完善，目前无可改进地方。

二、小蓝鲸的奇妙冒险-第二季

1. 解题思路:

这是典型的图论题目。

（1）建图，抽象数据

考虑到顶点的数目很大，邻接矩阵所需空间开不下来，所以用邻接表表示图。

（2）转换

该题目实际是在寻找最短的路径。需要寻找这样一个节点，使得从两个起点到它的路径长的和 + 从终点到它的路径长 最小。

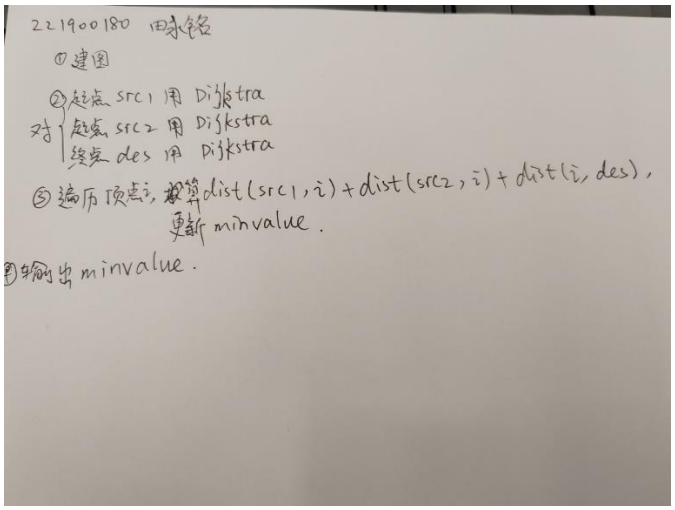
（3）算法

使用迪杰斯特拉算法，起点分别用一次，终点用一次，一共三次。对中间结点遍历，更新最短路径值，最后输出。

（4）优化

朴素迪杰斯特拉算法效率不高，需要利用堆来优化。

以下为算法流程图:



2. 核心代码+注释

```
void Dijkstra(Graph G, int start, int index) {//优化过的 Dijkstra 算法
    node vertex[m];
    bool visited[m];

    for (int i = 0; i < m; i++)
    {
        vertex[i].num = i;
        vertex[i].dis = maxValue;
        visited[i] = false;
    }
```

```
    vertex[start].dis = 0;
    q.insert(vertex[start]);
    while (q.size() != 0)
    {
        node min_vertex = q.top();//每次取得时候利用堆来优化时间，而
不是随便顺序遍历再拿最小的
        q.remove();
        int u = min_vertex.num;
        if (visited[u]) continue;
        visited[u] = true;
        Edge *ptr = G.NodeTable[u].adj;//G 为图，图的实现参考课程
PPT
        while (ptr != nullptr)//Dijkstra 算法更新过程
        {
            if (!visited[ptr->dest] && vertex[ptr->dest].dis >
vertex[u].dis + G.getWeight(vertex[u].num, ptr->dest))
            {
                vertex[ptr->dest].dis = vertex[u].dis +
G.getWeight(vertex[u].num, ptr->dest);
                q.insert(vertex[ptr->dest]);
            }
            ptr = ptr->link;
        }
    }
    //分三个出发点储存路径长
    if (index == 1) for (int i = 0;i < m;i++) dist_1[i] =
vertex[i].dis;
    else if (index == 2) for (int i = 0;i < m;i++) dist_2[i] =
vertex[i].dis;
    else if (index == 3) for (int i = 0;i < m;i++) dist_3[i] =
vertex[i].dis;
}

int main() {
    cin >> m >> n >> src1 >> src2 >> dest;
    int f, t, c;
    //建图
    for (int i = 0;i < n;i++)
    {
        scanf("%d%d%d", &f, &t, &c);
        graph.insert(f, t, c);
        graph_2.insert(t, f, c);
    }
    //使用三次 Dijkstra
    Dijkstra(graph, src1, 1);
```

```
    Dijkstra(graph, src2, 2);
    Dijkstra(graph_2, dest, 3);
    int min_path = 1433223;
    for (int i = 0;i < m;i++)
    {
        min_path = min(min_path, dist_1[i] + dist_2[i] +
dist_3[i]);//更新最小路径值
    }
    if (min_path >= 1433223)
    {
        cout << -1;
        return 0;
    }
    else cout << min_path;
    return 0;
}
```

其中，图和最小堆的实现完全参考姜远老师课程PPT，此处略去。

3. 程序（算法）的分析

设m表示点数，n表示边数

(1) 时间复杂度：O（$n\log_2 m$）

解释：最复杂的部分来自于Dijkstra（优化过的）算法

①寻找路径最短的点：

在朴素dijkstra算法中，效率最低的是遍历找距离最小的点（$O(m^2)$）

用堆优化后每次循环时间复杂度为O(m)

②加入集合：计算量为m次

③更新距离：由于采用堆的方式，所以在“用t更新其他点的距离”这步也会采用堆。

而在堆当中修改一个数的时间复杂度是logm，修改n次，故这步计算量为nlogm。

故堆优化版时间复杂度为O（$n\log_2 m$）。

(2)空间复杂度：O(m+n)　解释：即存储图的邻接表所用空间

4. OJ 运行结果截图

本地测试样例通过截图:

```
PS C:\Users\86181\Desktop\Program\VscodeData\C> cd "c:\Users\86181\Desktop\Program\VscodeData\C\sophomore\first_semester\big_project\" ; if ($?) { if
(Test-Path 'Execollection'){}else{md 'Execollection'} } ; if ($?) { g++ magical_journey_2.cpp -fexec-charset=UTF-8 -o Execollection\magical_journey_2
} ; if ($?) { Execollection\magical_journey_2 }
6 9 0 1 5
0 2 2
0 5 6
1 0 3
1 4 5
2 1 1
2 3 3
2 3 4
3 4 2
4 5 1
9
PS C:\Users\86181\Desktop\Program\VscodeData\C\sophomore\first_semester\big_project>
```

OJ通过截图:

#44595	#57. 小蓝鲸的奇妙冒险 - 第二季	221900180	100	338ms	27856kb	C++	5.7kb	2023-12-19 20:06:36

5.总结:

Dijkstra算法虽然好用，但是朴素Dijkstra的算法的复杂度效率有待提高，在每次取最小边的时候，完全可以利用堆来优化。

而此题的转化思想也很重要。可以找终点到各个点的距离，代替找每个顶点到重点的距离，这样少用了很多次Dijkstra算法。

该算法已较为完善，暂无可优化的地方。

三、小蓝鲸的奇妙冒险-第三季

1. 解题思路:

（1）方法1：利用数组暴力解决

开1000005位（略大于最大的id值）的数组，每个数组内存一个结构体，结构体存储冒险家成员的id、名字、战力值、该id的成员是否存在（valid == true表示存在，valid == false表示不存在（可能是没加入，可能是被删除了））。

对于insert操作：将对应id的valid改为true，进行信息填入。

对于delete操作：将对应id的valid改为false。

对于Query操作：从id==0一直顺序遍历到id==1000005，寻找valid并且符合要求的冒险家输出其信息。

此外：需要通过sstream和scanf和printf来优化读入和输出时间。

（2）方法2：利用平衡二叉树来解决

与方法1不同在，始终利用一棵平衡二叉搜索树储存每个冒险家的信息，这样效率更高。

读入和输出处理同方法1。

2. 核心代码+注释

（1）数组版本

```
struct People//结构体定义
{
    string name;
    int strength;
    bool valid = false;
};
People people[1000005];
```

```
int main() {
    int m, n;
    int in_id;
    string in_name = "";
    int in_strength;
    string read;

    scanf("%d%d", &m, &n);
    //读入部分处理
    for (int i = 0;i < m;i++)
    {
        scanf("%d", &in_id);
        cin >> in_name;
```

```
        scanf("%d", &in_strength);
        people[in_id].name = in_name;
        people[in_id].strength = in_strength;
        people[in_id].valid = true;
    }
    for (int j = 0;j <= n;j++)
    {
        getline(cin, read);
        stringstream ss(read);
        string a, b, c, d, e, f;
        ss >> a >> b >> c >> d >> e >> f;
        switch (a[0])
        {
        case 'I'://插入
        {
            int x = stoi(b);
            people[x].name = c;
            people[x].strength = stoi(d);
            people[x].valid = true;
            break;
        }
        case 'D'://删除
        {
            if (c.empty()) people[stoi(b)].valid = false;
            else
            {
                int x = stoi(b), y = stoi(c);
                for (int i = x;i <= y;i++) people[i].valid =
false;
            }
            break;
        }
        case 'Q'://询问,过长，略去
```

(2) AVL树版本

```
struct People//树节点定义
{
    int height;
    int id;
    string name = "";
    int strength;
    People *left;
    People *right;
```

```
    People(int in_id, string in_name, int in_strength) {
        id = in_id;
        name = in_name;
        strength = in_strength;
        left = nullptr;
        right = nullptr;
    }
};

People *insert(People *root, int in_id, string in_name, int
in_strength) {//插入
    if (root == nullptr)
    {
        root = new People(in_id, in_name, in_strength);
        return root;
    }
    if (in_id < root->id)
    {
        root->left = insert(root->left, in_id, in_name,
in_strength);
        root->height = max(root->left->height + 1, root->height);
        return adjust(root);
    }
    else if (in_id > root->id)
    {
        root->right = insert(root->right, in_id, in_name,
in_strength);
        root->height = max(root->right->height + 1,
root->height);
        return adjust(root);
    }
    root->strength = in_strength;
    root->name = in_name;
    return root;
}

People *del(People *root, int in_id) {//删除
    if (!root) return nullptr;
    if (in_id < root->id) root->left = del(root->left, in_id);
    else if (in_id > root->id) root->right = del(root->right,
in_id);
    else
    {
        if (!root->left && !root->right)
        {
```

```
            delete root;
            return nullptr;
        }
        else if (root->left && !root->right)
        {
            People *leftchild = root->left;
            delete root;
            root = leftchild;
        }
        else if (!root->left && root->right)
        {
            People *rightchild = root->right;
            delete root;
            root = rightchild;
        }
        else
        {
            People *to_del = root->left;
            while (to_del->right) to_del = to_del->right;
            root->id = to_del->id;
            root->name = to_del->name;
            root->strength = to_del->strength;
            root->left = del(root->left, to_del->id);
        }
    }
    if (root) root->height = cal_height(root);
    return adjust(root);
}
```

其中，调整平衡（左旋和右旋）的操作参考课程PPT，以及计算高度算法非核心，略去。

3. 程序（算法）的分析

设操作次数是位n,冒险家期望数目为m。

(1)数组版本

时间复杂度：O（1000005n）

解释：n次查找，每次查找最复杂的情况是数组从头找到尾

空间复杂度：1000005

（2）AVL树版本：

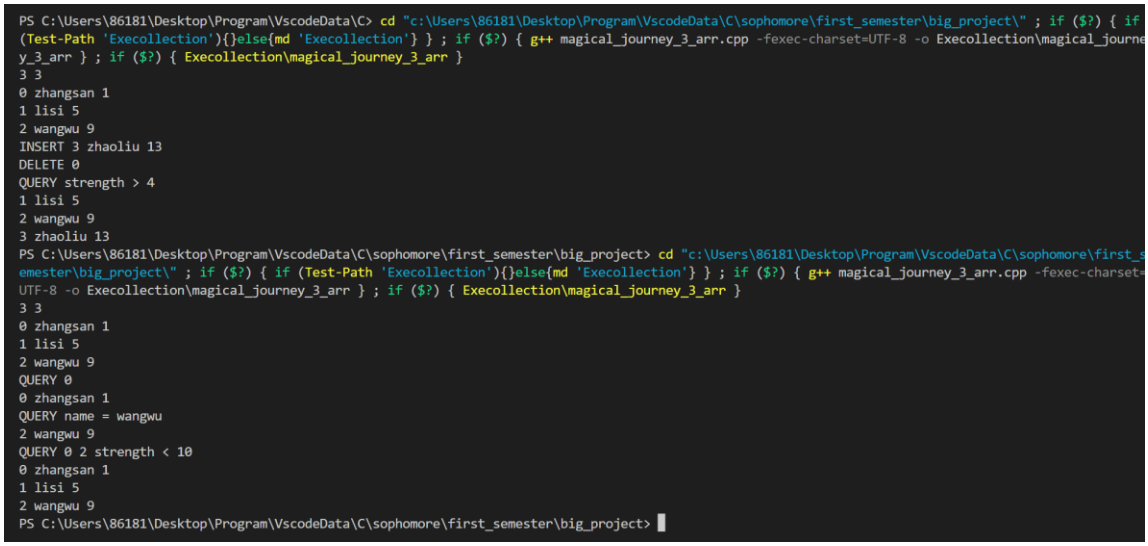
时间复杂度：O（$n\log_2 m$）

解释：n次查找，每次期望在m个树节点中找，花$\log_2 m$。

空间复杂度：O（m）

4. OJ 运行结果截图

本地测试样例通过截图:



OJ通过截图:

#47666	#58. 小蓝鲸的奇妙冒险 - 第三季	221900180	100	1277ms	42704kb	C++	20.2kb	2023-12-22 15:59:22

5.总结:

欲穷千里目，更上一层楼。虽然数组做法能够过这一题，但是效率还能提高，而且这样学不到更多知识。所以我还尝试了二叉搜索树和平衡二叉搜索树的写法，尽管有PPT可以参考，但是自己亲手实现还是会有很多bug，水平也提高了不少。