2022-2023《数据结构》大作业报告

学号：211220127

姓名：顾嘉宇

院系：计算机科学与技术

一、第一题 小蓝鲸的冒险第一季

1. 解题思路：

本题即求解从第二行开始每次输入数据之后，这些数字之中从大到小第[x/M](x 即为已经输入的数字数量，M 为题目给与)的数字。这道题很明显是一道卡时间复杂度的题，如果每次输入就排序一次，那么时间复杂度为 O(n^2logn)，这种解法必然超时。所以本道题我采取最大堆和最小堆来分别存储输入的数据。

核心思路：每次输入数据后，先将数据放入最大堆，然后取出堆顶数据。此时算出[x/M]的大小，由于[x/M]每次输入之后最多比[(x-1)/M]大 1，所以最小堆的大小要么比[x/M]小 1，要么相同。若最小堆大小小于[x/M]，则直接将最大堆堆顶数据移入最小堆；若最小堆大小和[x/M]相同，则将最大堆堆顶元素移入最小堆之后再将最小堆堆顶元素移入最大堆。

时间复杂度:O(nlogn)

2. 核心代码+注释：

全局变量：

```
int maxHeap[10000000]; //以数组的方式实现最大堆
int minHeap[10000000]; //以数组的方式实现最小堆
unsigned long long maxHeapSize = 0; //记录最大堆的容量
unsigned long long minHeapSize = 0; //记录最小堆的容量
```

最大堆和最小堆实现函数（具体实现就不在此列出）：

```
//最大堆下滤
void FilterDownMax(int start, int end)
//最大堆上滤
void FilterUpMax(int start)
//最大堆插入元素
void InsertMax(int val)
//最大堆移除堆顶
void RemoveMax()
//最小堆下滤
void FilterDownMin(int start, int EndOfHeap)
//最小堆上滤
void FilterUpMin(int start)
//移除最小堆堆顶元素
void RemoveMin()
```

```
//最小堆插入元素
void InsertMin(int x)
```

思路实现：

```
for (int i = 0; i < N; ++i) {
        scanf("%d", &input);
        InsertMax(input); //将输入元素存入最大堆
        int cap; //计算当前的[x/m]值大小
        if ((i + 1) % M == 0)
            cap = (i + 1) / M;
        else
            cap = (i + 1) / M + 1;
        //如果最小堆尺寸小于[x/m]，直接将最大堆堆顶元素插入最小堆，并移除最大堆堆顶元素
        if (minHeapSize < cap) {
            InsertMin(maxHeap[0]);
            RemoveMax();
        }
        //由于最小堆大小每轮最多加 1，则最小堆尺寸一定等于[x/m]或([x/M]-1)
        //所以在移除最大堆堆顶元素后，再将最小堆堆顶元素插入最大堆并移除最小堆堆顶元素
        else {
            InsertMin(maxHeap[0]);
            RemoveMax();
            InsertMax(minHeap[0]);
            RemoveMin();
        }
        //输出最小堆堆顶元素即为答案
        printf("%d ", minHeap[0]);
    }
```

3、OJ 运行结果：

#19500	#42. 小蓝鲸异世界冒险 - 第一季	211220127	100	460ms	7492kb	C++	3.6kb	2023-01-05 22:59:31

Test #1:	score: 10	Accepted	time: 3ms	memory: 3560kb
Test #2:	score: 10	Accepted	time: 3ms	memory: 3376kb
Test #3:	score: 10	Accepted	time: 0ms	memory: 3600kb
Test #4:	score: 10	Accepted	time: 3ms	memory: 3484kb
Test #5:	score: 10	Accepted	time: 3ms	memory: 3620kb
Test #6:	score: 10	Accepted	time: 4ms	memory: 3572kb
Test #7:	score: 10	Accepted	time: 10ms	memory: 3636kb
Test #8:	score: 10	Accepted	time: 46ms	memory: 3984kb
Test #9:	score: 10	Accepted	time: 385ms	memory: 7492kb
Test #10:	score: 10	Accepted	time: 3ms	memory: 3620kb

二、第二题 小蓝鲸的冒险第二季

1. 解题思路：本题即为求有向图中两个点到某一点的最短距离，且重复路径仅计算一次。本题的思路十分明确，为三次 dijkstra 算法求 src1,src2,dest 到各点的距离。难点为数据用何种形式保存。由于本题数量级为 1e5，所以不采取邻接矩阵保存，而是采用链式前向星来保存每条有向边。在 dijkstra 时采用堆优化的方式来选取权值最小的边，从而降低时间复杂度。

本题时间复杂度为 O((n+m)logn)。

2. 核心代码+注释：

全局变量

```
//存储输入时的每条边的起点，终点和权值
struct temp {
    int start; int end;
    long long time;
};
//构造 Edge 来存储边
struct Edge {
    int to, next, v;
} a[300000];
int visit[300000];//判断该城市是否被访问
int N;//城市数量
//链式前向星的辅助数组和数值
int h[300000];
int cnt;
long long dijk[3][300000];//分别对应 src1,src2,dest 到各点的最短距离
```

最小堆代码：

```
struct Pair {
    long long time;
    int src;
};
//用 pair 初始化最小堆，使得每个堆元素中分别存储城市编号和相隔时间
Pair minHeap[300000];
//记录最小堆大小
int minHeapSize = 0;
//最小堆上滤（基于 time 大小，下同）
void FilterDown(int start, int EndOfHeap)
//最小堆下滤
void FilterUp(int start)
//移除最小堆堆顶元素
void Remove()
//插入最小堆
void Insert(int time, int src)
```

核心 dijkstra 代码：

```
void dij(int src, long long* dijk) {
    for (int i = 0; i < N; i++) {
        //初始化当前的 dijk 数组和 visit 数组
        dijk[i] = 1e18;
        visit[i] = 0;
    }
    dijk[src] = 0;//初始点到当前点距离为 0
    Insert(0, src);//将当前点插入最小堆
    while (minHeapSize != 0) {
        int x = minHeap[0].src; //取出最小堆堆顶的 src
        Remove();//移除最小堆堆顶元素
        if (visit[x]) continue;//如果当前 src 被访问过,继续下一轮循环
        visit[x] = 1;//当前结点被访问过
        for (int i = h[x]; i; i = a[i].next)//遍历与当前结点相连的边
            //如果该节点被访问过，并且经过该点而到达最终点的权值总和小于原来的
权值
            //则更改到达最终点的权值，并将其保存到最小堆中
            if (visit[x] && dijk[x] + a[i].v < dijk[a[i].to]) {
                dijk[a[i].to] = dijk[x] + a[i].v;
                Insert(dijk[a[i].to], a[i].to);
            }
    }
}
```

核心流程：

```
    //存储邻接表
    for (int i = 0; i < n; ++i)
        add(tmp[i].start, tmp[i].end, tmp[i].time);
    // 得出 src1, src2 到各点的最短距离
    dij(src1, dijk[0]); dij(src2, dijk[1]);
    //清零辅助数组和数值
    for (int i = 0; i < m; i++) h[i] = 0;
    cnt = 0;
    //存储逆邻接表
    for (int i = 0; i < n; ++i)
        add(tmp[i].end, tmp[i].start, tmp[i].time);
    //得出 dest 到各点的最短距离
    dij(dest, dijk[2]);
    long long ans = 1e18;
    //遍历所有节点得出最小值
    for (int i = 0; i < m; i++)
        ans = min(ans, dijk[0][i] + dijk[1][i] + dijk[2][i]);
    //输出答案
    if (ans == 1e18) cout << -1;
    else cout << ans;
```

3、OJ 运行结果

#19530	#43. 小蓝鲸异世界冒险 - 第二季	211220127	100	642ms	14628kb	C++	3.8kb

Test #1:	score: 10	Accepted	time: 0ms	memory: 3432kb
Test #2:	score: 10	Accepted	time: 1ms	memory: 3496kb
Test #3:	score: 10	Accepted	time: 0ms	memory: 3456kb
Test #4:	score: 10	Accepted	time: 4ms	memory: 3420kb
Test #5:	score: 10	Accepted	time: 4ms	memory: 3244kb
Test #6:	score: 10	Accepted	time: 31ms	memory: 4264kb
Test #7:	score: 10	Accepted	time: 64ms	memory: 5812kb
Test #8:	score: 10	Accepted	time: 126ms	memory: 8348kb
Test #9:	score: 10	Accepted	time: 145ms	memory: 11960kb
Test #10:	score: 10	Accepted	time: 267ms	memory: 14628kb

三、第三题 小蓝鲸的冒险第三季

1. 解题思路：

本题即维护一个存储系统，分别保存人员的 ID，姓名和力量，并分别实现插入人员，删除人员，和查询人员三个不同功能。其中删除人员分为指定 ID 删除和 ID 范围内删除。查询分别 ID 查询，姓名查询，力量范围查询和规定 ID 范围的力量范围查询。

采用三个数组分别维护 ID，姓名和力量，并通过不同的函数来实现对应的功能。本题关键点为输出时若有多个人员信息，则应保持输出 ID 从小到大，所以在输入行为指令时应判断 ID 的大小并将之插入到对应的位置。

时间复杂度 O(mn)

2. 核心代码+注释：

全局变量：

```
int strength[100200]; //存储冒险者力量
int id[100200];       //存储冒险者 ID
string name[100200];   //存储冒险者姓名
int adsize = 0;  //记录当前冒险者的数量
```

功能实现函数（具体实现在 home3.cpp 中）：

```
//插入冒险家
void Insert(int Id, string Name, int Strength)
//删除指定冒险家 ID
void Delete(int ID)
//删除 ID >= ID1 并且 ID <= ID2 的冒险家
void Delete(int ID1, int ID2)
//查询指定 ID 的冒险家
void Query(int ID)
//查询指定姓名的冒险家
void Query(string Name)
//查询 strength cmp(cmp 为指令所给的符号，为=,!=,>=,>,<=,<) value 的冒险家
void Query(int value, string cmp)
//查询 ID1<=ID<=ID2，并且 strength cmp(cmp 为指令所给的符号，=,!=,>=,>,<=,<)
value 的冒险家
void Query(int ID1, int ID2, int value, string cmp)
```

插入时的判断：

```
void Insert(int Id, string Name, int Strength)
{
    id[adsize] = Id;
    name[adsize] = Name;
    strength[adsize] = Strength;
    adsize++;
    //如果插入元素比原先队列最大的元素要大，则查询插入位置
    if (id[adsize - 1] < id[adsize - 2]) {
        //存储冒险家对应的 ID,name，strength
```

```
        int tempid = id[adsize - 1];
        string tempname = name[adsize - 1];
        int tempstrength = strength[adsize - 1];
        int index = 0;
        for (int i = 0; i < adsize - 1; ++i) {
            //查找到位置
            if (tempid < id[i]) {
                index = i ;
                //将该位置即其后的元素全部后移一位
                for (int i = adsize - 1; i > index; --i) {
                    id[i] = id[i - 1];
                    name[i] = name[i - 1];
                    strength[i] = strength[i - 1];
                }
                //插入
                id[index] = tempid;
                name[index] = tempname;
                strength[index] = tempstrength;
                break;
            }
        }
    }
}
```

获取指令的判断(包含 Query 内容的简要说明)：

```
void getCommand() {
    //收取第一个输入的英语单词
    string cmd;
    cin >> cmd;
    if (cmd == "INSERT") { ... }
    else if (cmd == "DELETE") { ... }
    else if (cmd == "QUERY") { ... }
}
```

```
    else if (cmd == "QUERY") {
        string a[10], b;
        int count = 0;
        //获取该行输入的所有元素
        while (cin >> b) { ... }
        //count = 1,说明为查找ID
        if (count == 1) { ... }
        //输入的第一个单词为name,说明查找姓名
        else if (a[0] == "name") { ... }
        //输入的第一个单词为strength,说明查找strength范围
        else if (a[0] == "strength") { ... }
        //若非上列所属情况，则为查找ID1<=ID<=ID2,且strength符合一定范围的冒险者
        else { ... }
    }
}
```

主体流程：

```
int main()
{
    int m,n;
    cin >> m >> n;
    int ID, value;
    string name;
    for (int i = 0; i < m; ++i) {
        cin >>ID >> name >> value;
        Insert(ID, name, value);//保存插入冒险家
    }
    for (int i = 0; i < n; ++i)
        getCommand();//获取指令并解析
    return 0;
}
```

3、OJ 运行结果：

#19557	#44. 小蓝鲸异世界冒险 - 第三季	211220127	100	1333ms	10972kb	C++	7.8kb	2023-01-06 15:02:05

Test #1:	score: 10	Accepted	time: 2ms	memory: 6600kb
Test #2:	score: 10	Accepted	time: 2ms	memory: 6656kb
Test #3:	score: 10	Accepted	time: 71ms	memory: 7372kb
Test #4:	score: 10	Accepted	time: 89ms	memory: 7352kb
Test #5:	score: 10	Accepted	time: 114ms	memory: 7324kb
Test #6:	score: 10	Accepted	time: 111ms	memory: 10972kb
Test #7:	score: 10	Accepted	time: 112ms	memory: 7472kb
Test #8:	score: 10	Accepted	time: 86ms	memory: 7464kb
Test #9:	score: 10	Accepted	time: 337ms	memory: 7384kb
Test #10:	score: 10	Accepted	time: 409ms	memory: 7428kb

四、第四题 小蓝鲸的冒险第四季

1、解题思路:

本题即求有向图中长度为 3~7 的环的个数。采用深度优先搜索。由于遍历所有结点采用 dfs 会导致时间复杂度过高。所以将 dfs 分解为 rdfs 和 dfs，即由于长度为 7 的环中距离七点最远的距离不超过 3，所以用 vis1[MAXN]和 vis2[MAXN]数组来保存对应结点，同时由于城市 ID 数值会十分大，所以采用 hash 表来对城市 ID 进行处理。同时为防止相同环重复计算，使得初始搜索点的值始终为环中的最小值

DFS 的时间复杂度无法具体求出。

2、核心代码:

全局变量:

```
#define maxN 200000  //定义数组尺寸
#define mod 199999;  //哈希 mod 值
int Vout[maxN][600]; //经过哈希处理的结点所相连的点
int sizeVout[maxN];  //经过哈希处理的结点出度
int Vin[maxN][600];  //经过哈希处理的结点所相连的点
int sizeVin[maxN];   //经过哈希处理的结点的入度
bool vis[maxN];      //判断哈希处理的结点是否被访问过
int rPath2[maxN][8]; //反向存储两层路径
int rPath2Size[maxN];//存储结点对应路径的 size
int vis1[maxN];      //标记反向距离不超过 3 的所有结点
int vis2[maxN];      //标记走两步可到达 head 的结点
int answer = 0;      //答案
int Hash[maxN];      //hash 表
struct EDGE {
    int start, end;
}; //存储边
```

哈希表的处理:

```
//哈希值写入
int writeHash(int x)
//哈希值读取
int getHash(int x)
```

核心代码（rdfs 和 dfs）:

```
//以逆邻接表来求得 rPath2 和 vis2
void rdfs3(int head, int cur, int depth)
{
    //求得 cur 对应的 Hash 值，并使得该 hashCur 已被访问
    int hashCur = getHash(cur);
    vis[hashCur] = true;
    //遍历 hashCur 的入边
    for (int i = 0; i < sizeVin[hashCur]; ++i)
    {
        //求得入边点并将之进行 Hash 处理
        int ver = Vin[hashCur][i];
```

```
        int hashVer = getHash(ver);
        //如果访问过或者当前值小于 head 值，此放弃此轮循环
        if (ver < head || vis[hashVer]) continue;
        //将当前 head 值保存到 vis1[hashVer]中，表示该节点可以到达 head
        vis1[hashVer] = head;
        //如果深度为 2 则深入处理
        if (depth == 2)
        {
            //如果 vis2[hashVer]保存的不是 head 结点，则将 rPath2[hash]清零
            if (vis2[hashVer] != head)
            {
                for (int i = 0; i < rPath2Size[hashVer]; ++i)
                    rPath2[hashVer][i] = 0;
                rPath2Size[hashVer] = 0;
            }
            //令 vis2[hashVer] = head，并将当前的 cur 存储到 hashVer 所对应的
rPath2 路径中
            vis2[hashVer] = head;
            rPath2[hashVer][rPath2Size[hashVer]] = cur;
            rPath2Size[hashVer]++;
        }
        //若深度小于 3，则继续搜索
        if (depth < 3)
            rdfs3(head, ver, depth + 1);
    }
    //回溯，置 vis[hashCur]为未访问的状态
    vis[hashCur] = false;
}
//以邻接表进行搜索
void dfs5(int head, int cur, int depth)
{
    //将 Cur 值进行 hash 处理
    int hashCur = getHash(cur);
    vis[hashCur] = true;
    //遍历 hashCur 所有的出边
    for (int i = 0; i < sizeVout[hashCur]; ++i)
    {
        //获取当前的出边结点并将之进行 hash 处理
        int ver = Vout[hashCur][i];
        int hashVer = getHash(ver);
        //如果访问过或者当前值小于 head 值，此放弃此轮循环
        if (ver < head || vis[hashVer]) continue;
        //若深度大于 3，且此时不能访问到 head，则放弃此轮循环
        if (depth > 3 && vis1[hashVer] != head) continue;
```

```
        // 环的判定条件：ver 结点走两步可到达 head
        if (vis2[hashVer] == head)
        {   //遍历 hashVer 对应的 rPath
            for (int j = 0; j < rPath2Size[hashVer]; ++j)
            {
                //得到 hashVer 路径上所经过的点，并将之进行 hash 处理
                //若已经访问，则放弃此轮循环，若非，则答案加一
                int temp = rPath2[hashVer][j];
                int hashTemp = getHash(temp);
                if (vis[hashTemp]) continue;
                ++answer;
            }
        }
        if (depth < 5)   // 继续搜索
            dfs5(head, ver, depth + 1);
    }
    vis[hashCur] = false;
}
```

主体代码：

```
//保存邻接表和逆邻接表
    for (int i = 0; i < m; ++i) {
        int tempStart = writeHash(edge[i].start);
        int tempEnd = writeHash(edge[i].end);
        Vout[tempStart][sizeVout[tempStart]] = edge[i].end;
        Vin[tempEnd][sizeVin[tempEnd]] = edge[i].start;
        ++sizeVout[tempStart];
        ++sizeVin[tempEnd];
    }
    for (int i = 0; i < maxN; ++i)
    {
        if (sizeVout[i] && sizeVin[i])  // 只有出入度不为 0 的结点才会构成环
        {
            rdfs3(Hash[i], Hash[i], 1);
            dfs5(Hash[i], Hash[i], 1);
        }
    }
```

3、OJ 运行结果：

#19584	#45. 小蓝鲸异世界冒险 - 第四季	211220127	100	1138ms	406936kb	C++	4.8kb	2023-01-06 16:37:50

Test #1:	score: 10	Accepted	time: 0ms	memory: 4088kb
Test #2:	score: 10	Accepted	time: 6ms	memory: 4324kb
Test #3:	score: 10	Accepted	time: 8ms	memory: 23936kb
Test #4:	score: 10	Accepted	time: 25ms	memory: 61108kb
Test #5:	score: 10	Accepted	time: 33ms	memory: 50292kb
Test #6:	score: 10	Accepted	time: 40ms	memory: 51616kb
Test #7:	score: 10	Accepted	time: 45ms	memory: 47660kb
Test #8:	score: 10	Accepted	time: 57ms	memory: 49864kb
Test #9:	score: 10	Accepted	time: 63ms	memory: 52056kb
Test #10:	score: 10	Accepted	time: 861ms	memory: 406936kb