

课程设计报告

课程名称： 《算法分析与设计》

设计题目：

院 系： 悉尼工商学院

设 计 者： 陈绍轩

学 号： 20170048

指导教师： 喻钢

设计时间： 2022.06.08

课程题目：磁带存储问题

个人小结：

在刚拿到磁带最优存储问题的题目时，由于题目要求与设计描述相当简略以及采用了数学表达的方式进行描述题目，以至于我在最初对拿到手的问题似乎有些毫无头绪，完全没有理解清楚问题以及其中所表达的含义，也对其数学表达问题的方式产生了些许的畏惧心理。然而在反复审阅题目，尝试去理解其中数学表达式的所代表的含义，以及查阅了一些相关资料后，我才逐渐理解了题目所想表达出的问题在于各个程序排列在不同位置时，其读取程序所需的时间不同，且为 tr 表达式，而问题在于需要通过排序使所有程序的读取时间之和（即 MRT 平均读取时间）最短，以及解决该问题的关键在于 tr 表达式是程序 i 及其之前所有程序的独立读取时间之和，各程序独立读取时间为其长度 li 乘以读取概率 pi ，因此排序越靠前的程序，在计算 MRT 时，被求和的次数就越多。

在通过学习分析以及理解题目后，我逐渐理解了解决该问题的思路，需要根据程序的 $li \cdot pi$ 值来对程序进行非递减排序，且该问题的解决符合贪心算法的思想。因为需要两个变量的乘积作为排序标准，由此我选择了使用二叉堆构成优先队列来以 $li \cdot pi$ 的值作为优先级进行依次出队进行堆排序。在编写这段算法的过程中，也加深了我对堆排序，优先队列概念的理解，尤其是优先队列与普通队列的区别，通过优先级来决定出队顺序，并且通过二叉堆来实现优先队列数据结构，以及二叉堆其中的一些操作比如堆的自底向上构建和下滤操作，通过学习温故了自底向上构建堆的操作，通过下滤重建堆的操作，以及其时间复杂度分别为 $O(n)$ 和 $O(\log n)$ 。由于需要进行的是非递减排序，所以我也温故了如何使用数组存储的方式通过大顶堆对数组里的元素进行就地排序。综上，在解决该问题的核心算法过程中，我温故到了二叉堆，优先队列以及堆排序的内容。

在编写其他函数，比如输出对程序的排序结果以及计算排序后的最小平均读取时间的算法中，我也加入了一些自己的想法，对一些问题进行了优化。在输出排序结果的过程中，我通过对结构体加入了 id 整型变量来记录程序的编号，使得在排序后可以对编号进行输出，从而查看所有程序对应的排序结果。而在编写计算最小平均读取时间的算法中，其表达式 $\sum \sum p_i k_{i,j}$ 需要进行两层的求和，如果采用两层循环则会导致该算法时间复杂度为 $O(n^2)$ ，因此我在一开始采用两层循环后，进行了些思考，根据表达式求和的特点，考虑到可以通过一层循环遍历，并且第 i 个程序会被读取 $n-i+1$ 次，因此在一次遍历中将其独立读取时间 $li \cdot pi$ 乘以 $n-i+1$ ，仅用一层循环进行计算，将该过程的时间复杂度减小为了 $O(n)$ 。

总得来说，此次项目操作帮助我通过实践编写算法解决问题，学习以及巩固到了一些所学的知识，并且实践进行了对问题的分析与思考解决，加强了我对一些知识点的深入理解，以及对问题的思考分析能力。

课程设计报告

1. 题目描述与设计要求

设有 n 个程序 $\{1, 2, \dots, n\}$ 要存放在长度为 L 的磁带上。程序 i 存放在磁带上的长度是 l_i ， $1 \leq i \leq n$ 。这 n 个程序的读取概率分别为 p_1, p_2, \dots, p_n ，且 $\sum p_i = 1$ ($i=1, 2, \dots, n$)。如果将这 n 个程序按 i_1, i_2, \dots, i_n 的次序存放，则读取程序所需的时间 $tr = c \sum p_{ik} l_{ik}$ ($k=1, 2, \dots, r$) (可假定 c 为 1)。这 n 个程序的平均读取时间为 $t(1) + t(2) + \dots + t(r)$ 。磁带最优存储问题要求确定这 n 个程序在磁带上的一个存储次序，使平均读取时间达到最小。

1.1 输入要求

输入第 1 行是正整数 n ，表示文件程序的个数。接下来的 n 行中，每行有 2 个正整数 len 和 pr ，分别表示程序存放在磁带上的长度和读取概率。实际上第 k 个程序的读取概率为 $pr_k / \sum pr_i$ 。对所有输入均假定 $c=1$

1.2 输出要求

对于每一个测试用例，第 1 行输出程序实现最小平均读取时间的存储顺序，第 2 行输出对应计算出的最小平均读取时间顺序。

2. 算法设计

2.1 问题分析

首先由问题可知程序 i 分别存在两个参数，其长度 l_i 以及读取概率 p_i ，且 $\sum p_i = 1$ 。为了方便输入，在问题参数输入时采用了正整数型读取概率 pr ，而实际读取概率 $p_i = pr_i / \sum pr_i$ 从而使得 $\sum p_i = 1$ 。对问题进行剖析后，需要理解的一点为对于磁带来说，其只提供数据的顺序访问，即在在录音带/磁带中，与 CD 不同，磁带上的第五首歌曲不能直接播放，而是必须得先遍历前四首歌曲的长度后才能播放第五首歌，这一点则体现在了读取各个程序所需时间 tr 的公式中。

其中 $tr = c \sum p_{ik} l_{ik}$ ，对所有输入均假定 $c=1$ ，则说明要读取第 i 个程序，则需要将前面 $i-1$ 个程序全部遍历一遍。这里需要区别两个概念，一个是单个程序的读取时间为 $p_i * l_i$ ，而在磁带中程序的读取时间需要考虑到前面 $i-1$ 个程序的读取时间，因此需要求和为 tr 。平均读取时间 MRT (Mean Retrieval Time) 则为所有程序的读取时间的加总，即 $MRT = \sum tr$ ，而不同的程序排序方式则会影响 MRT 的大小。

例 1：一个磁带上 有 3 首歌曲，其歌曲长度分别为 4、5、2 分钟，且读取概率均相等，即 $p_i = 1/3$ ，则其全排列个数为 $3! = 6$ ，根据程序读取时间 tr 公式以及平均读取时间 MRT 公式可得出不同歌曲排列下的平均读取时间如下表。

表 1 不同排列及其平均读取时间

Order	Mean Retrieval Time (MRT) formula	Result
2 5 4	$2*1/3+(2*1/3+5*1/3)+(2*1/3+5*1/3+4*1/3)$	673.33
2 4 5	$2*1/3+(2*1/3+4*1/3)+(2*1/3+4*1/3+5*1/3)$	639.66
5 2 4	$5*1/3+(5*1/3+2*1/3)+(5*1/3+2*1/3+4*1/3)$	774.33
5 4 2	$5*1/3+(5*1/3+4*1/3)+(5*1/3+4*1/3+2*1/3)$	841.66
4 2 5	$4*1/3+(4*1/3+2*1/3)+(4*1/3+2*1/3+5*1/3)$	707
4 5 2	$4*1/3+(4*1/3+5*1/3)+(4*1/3+5*1/3+2*1/3)$	808

由此可见其中 2、4、5 的排列 MRT 最小为 639.66，而其中的关键在于 p_i 与 l_i 的乘积，即单个程序独立的读取时间。根据 MRT 的计算公式可以看出，排序越靠前的程序其独立读取时间被累加的次数越多。如 5、2、4 的排序， $5*1/3$ 的读取时间在计算 MRT 的过程中会被累加 3 次， $2*1/3$ 会被累加 2 次，因此第 i 个程序，其独立读取时间 p_i*l_i 在计算 MRT 时会被累加 $n-i+1$ 次。

所以可知程序的独立读取时间 $pr*l_i$ （实际操作中用正整数 pr 代替 p_i 方便输入）越小的程序存储次序应该越靠前，使得其最先被读取，从而使得平均读取时间 MRT 越小。

磁带最优存储问题的关键在于以 pr 与 l_i 的乘积为优先级进行程序的存储排序，因此我以贪心法为算法思想，通过使用二叉堆来构造优先队列数据结构，对磁带程序以 $pr*l_i$ 作为优先级进行出队排序，采用二叉堆数据结构在于堆排序是一种非常高效的排序方式，且为就地 (in-place) 排序，辅助空间 (auxiliary space) 为 $O(1)$ ，因此时间与空间复杂度较低，而且程序的存储次序问题不需要考虑稳定性问题。

2.2 算法设计思路及求解过程

2.2.1 贪心算法分析

首先对磁带最优存储问题进行贪心问题分析：



图 1 贪心问题分析

贪心选择性质：磁带问题有一个最优解以贪心选择开始

- 命题：磁带问题有一个最优解以贪心选择开始，即包含最小的 $len \cdot pr$ ，即 t_0 。
- E 数组是将磁带上的程序以 $len \cdot pr$ 非递减排列得到的最优解， $E = \{t_0, t_1, t_2 \dots t_{n-1}\}$ 。

对于问题 $p = \bigcup \{t_i\}$ ，因为问题的解包含了所有的 tr ，如果只存在一个最优解 E ，那么序列的顺序是特定的，最优解 E 一定包含了 t_0 ，问题转换为证明最优解唯一。假设除 $E = \{t_0, t_1, \dots, t_j \dots t_i \dots t_{n-1}\}$ 之外还存在一个最优解 $E' = \{t_0, t_1, \dots, t_j \dots t_i \dots t_{n-1}\}$ ，那么访问的时间 $TE' \leq TE$ 。做差得：

$$TE' - TE = (t_j - t_i) + (t_j - t_i)(j - i - 1) + \{(t_j - t_i) + (t_j - t_i)(j - i)\} = 2(t_j - t_i)(j - i) + 1$$

由于 $t_j \geq t_i$ ，因此 $TE' \geq TE$ 。若 $TE' = TE$ ，二者等价于一个最优解。若 $TE' > TE$ ，与假设矛盾，所以说问题的最优解唯一。因为只存在一个最优解 E ，那么序列的顺序是特定的，最优解 E 一定包含了 t_0 。因此得结论：磁带最优存储问题有一个最优解以贪心选择开始，即以 t_0 开始。

最优子结构性质：子问题的最优解可以推导出原问题的最优解

若 A 是包含于 E 的一个最优解，则 $A' = A - \{E[0]\}$ 是子问题 E' 的最优解。

假设在 E' 中找到一个最优解 B' ，它的平均读取时间比 A' 更小，则将 $E[0]$ 加入 B' 中将产生 E 的一个最优解 $B = \{E[0]\} + A' - \{E[0]\}$ ，它的平均读取时间比 A 要更小，这与 A 的最优性相矛盾。所以每做一步贪心选择都能得到一个更小的，与原问题结构一致的子问题。因此得结论：磁带问题具有最优子结构性质。

最优子结构性质：子问题的最优解可以推导出原问题的最优解

设 $p = \bigcup \{t_i\}$ 为 n 个程序的集合， S_i 是集合 $\{i, i+1, \dots, n-1\}$ 中 tr 最小的程序的时间， A_0 是包含程序 0 的最优解，则 $A_0 = \bigcup \langle 0, n-1 \rangle (S_i)$ ，证明 $A_0 = \bigcup \langle 0, n-1 \rangle (S_i)$ 是最优解。根据数学归纳法：

- ② $|A|=1$ ，由定理 1，成立。
- ③ $|A|<k$ ，由定理 2，成立。
- ④ $|A|=k$ ，由定理 2， $A = \{0\} + A_1$ ， $A_1 = \{S_0\} + \bigcup \langle 0, n-1 \rangle (S_i)$

由假设归纳知 $A_0 = \bigcup \langle 0, n-1 \rangle (S_i)$

因此得结论： $A_0 = \bigcup \langle 0, n-1 \rangle (S_i)$ ，由局部最优可以得到整体最优。

2.2.2 算法求解过程

磁带最优存储问题根据 tr 计算公式，可知要使平均读取时间最小，即需要使 $pi \cdot li$ 最小的程序放在最前面，从而最先被读取，即本题可以通过贪心法，在每个步骤中，我们都会立即选择将独立读取时间最少的程序放在首位，以便逐个通过最优子结构建立起最终的优化解决方案。算法的求解过程为：

- 输入每个程序的长度与读取概率，并计算其乘积
- 构建以 $pi \cdot li$ 为优先级的二叉堆优先队列

- 通过优先队列出队进行堆排序，输出程序存储顺序
- 遍历求出 n 个程序的平均读取时间 MRT

算法首先定义程序(Program)结构体，结构体内包含整型变量程序 id、长度 len、正整数读取概率 pr，以及双精度浮点型变量程序独立读取时间 tr。其中 tr 等于 $\text{len} \times \text{pr}$ 。算法采用数组来对二叉堆优先队列元素进行存储，通过动态分配数组空间，利用 input 函数对空间内程序结构体的变量赋值，并在数组内就地进行二叉堆的建堆以及排序后的下滤重建，实现以 tr 为优先级的出队排序操作。其中 heapify 函数实现了二叉堆的下滤操作，在初始建堆的过程中自底向上进行以 tr 作为优先级构建大顶堆，即 tr 越大的程序越接近根结点，并在出队后重建大顶堆的过程中实现下滤。findOrderMRT 函数实现了堆排序的过程，在第一个循环语句中循环调用 heapify 进行自底向上的初始化建堆，而在第二个循环语句中，则是先使用 swap 函数将大顶堆堆顶的元素与存储在数组最后的一个元素交换，实现优先队列出队的操作，并循环对剩余的 i 个程序不断进行堆重建与出队操作，最终实现以 tr 为优先级的递增排序。以上述例 1 为例，各程序 tr 分别为 $4/3$ 、 $5/3$ 、 $2/3$ ，建堆与堆排过程如下所示：

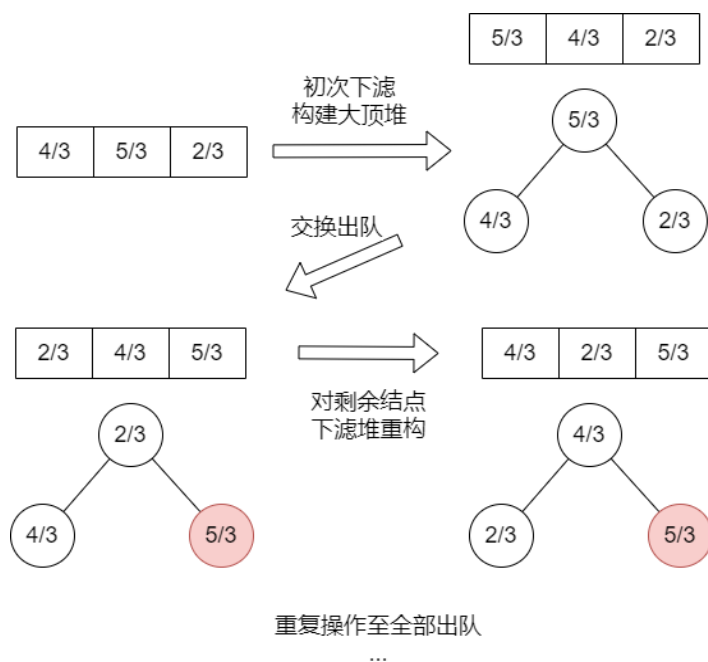


图 2 堆排序示例

算法随后通过 printOrder 函数对已就地排序的结构体内的 id 变量进行输出，输出排序后的程序的存储顺序，最后通过 minRetrievalTime 来计算最小平均读取时间。在 minRetrievalTime 函数中，我对该算法进行了优化，根据其求和的性质仅通过一层循环实现两层求和 $\sum \sum p_i k_{lik}$ ，由于遍历过程中，第 1 个程序会被读取 n 次，第 2 个程序会被读取 $n-1$ 次，因此第 i 个程序会被读取 $n-i+1$ 次，因此在遍历过程中，按照排序的遍历顺序依次对遍历程序乘上 $n-i+1$ ，从而实现从 $O(n^2)$ 时间复杂度降至 $O(n)$ 。

3. 算法实现

3.1 算法核心流程

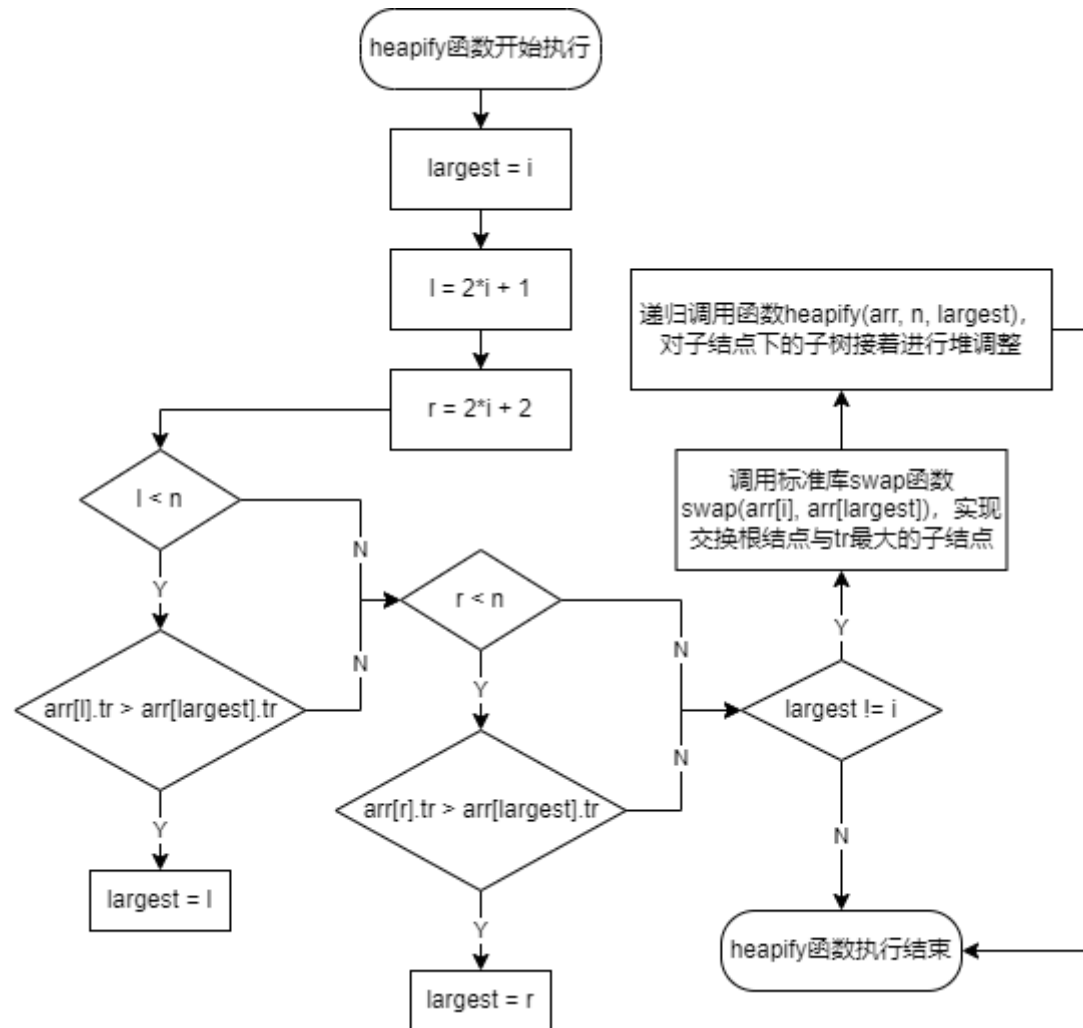


图 3 heapify 函数流程图

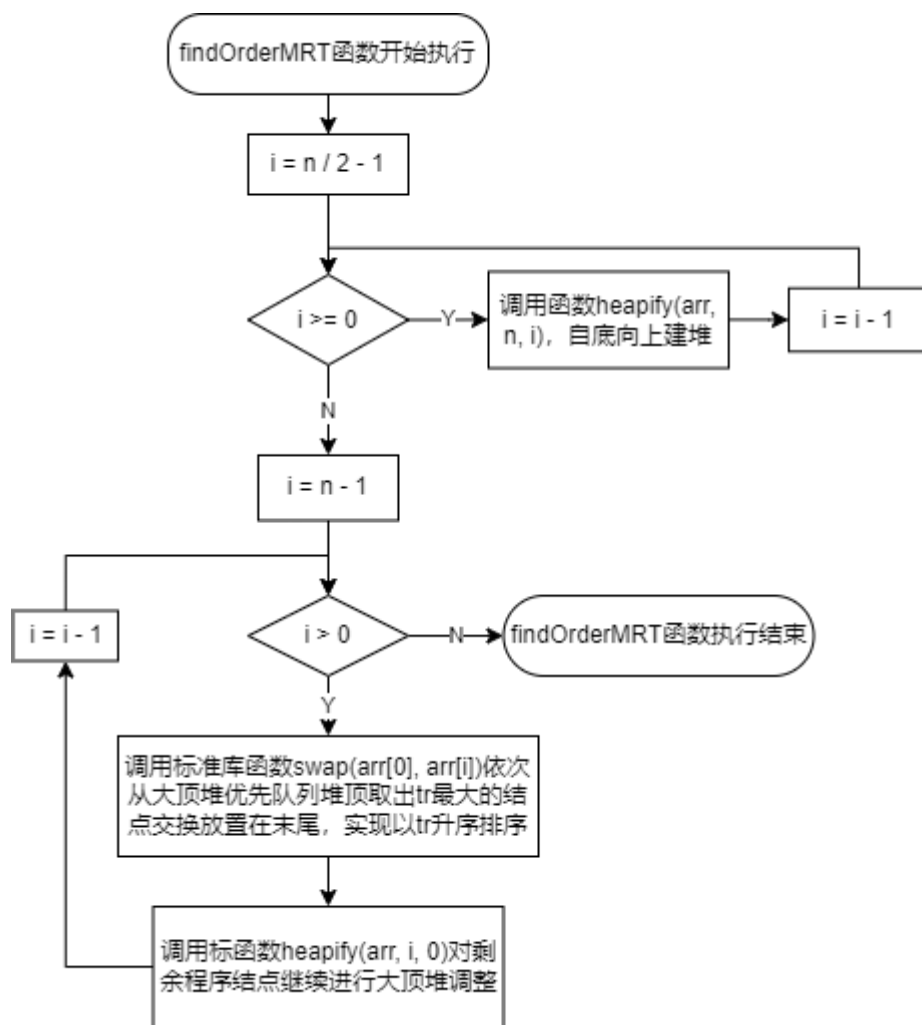


图 4 findOrderMRT 函数流程图

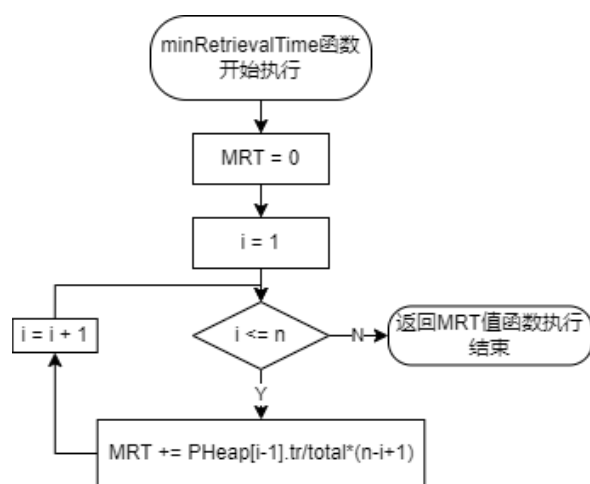


图 5 minRetrievalTime 函数流

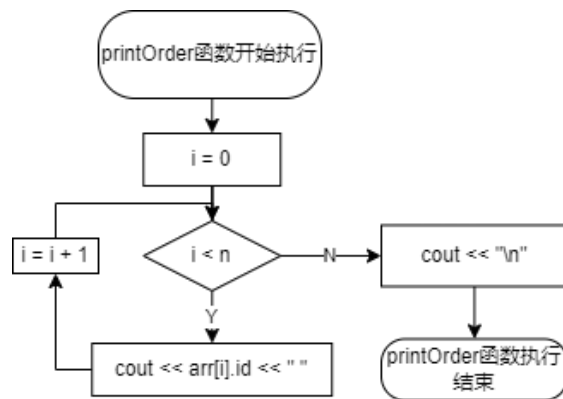


图 6 printOrder 函数流程图

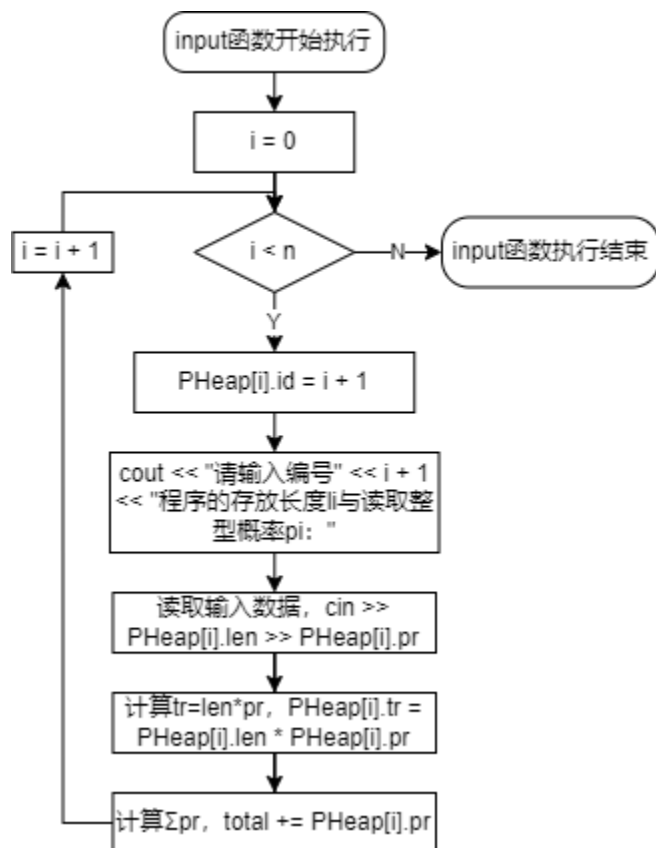


图 7 input 函数流程图

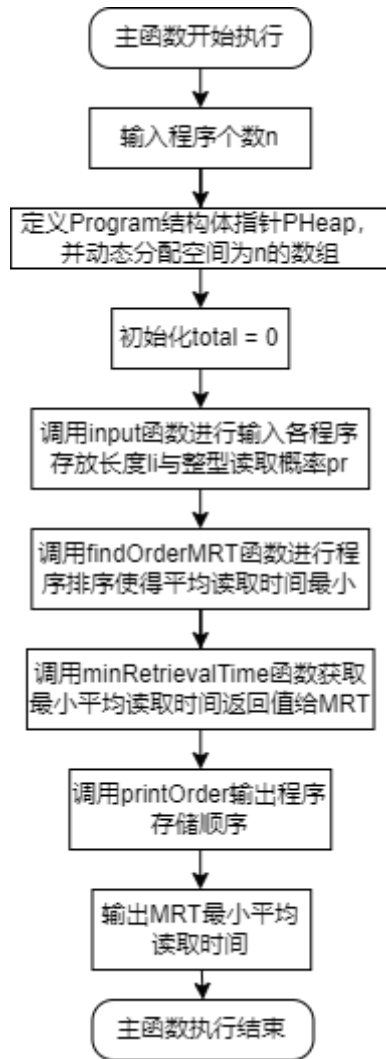


图 8 main 函数流程图

3.2 实现关键代码

```

#include <iostream>
using namespace std;

//定义每个程序结构体
struct Program
{
    int id;      //程序编号
    int len;    //长度li
    int pr;      //整型pr代表概率，实际概率pi=pr/Σpr
    double tr;  //tr=len*pr，且作为优先队列出队的条件，且单个程序读取时间为len*pr/Σpr，
    pi=pr/Σpr, Σpi=1
};
  
```

```

//比较根结点为i的子树各结点tr的大小，自底向上构造大顶堆，n为结点总数
void heapify(Program arr[], int n, int i)
{
    int largest = i; //默认根节点i的tr最大
    int l = 2 * i + 1; //左子结点2*i + 1
    int r = 2 * i + 2; //右子结点2*i + 2
    if (l < n && arr[l].tr > arr[largest].tr) //如果左子结点存在且左子结点的tr
比根结点的tr大
        largest = l; //记录该左子结点
    if (r < n && arr[r].tr > arr[largest].tr) //如果右子结点存在且右子结点的tr
比目前结点的tr大
        largest = r; //记录该右子结点
    if (largest != i) // 如果该子树不构成大顶堆
    {
        swap(arr[i], arr[largest]); //交换根结点与tr最大的子结点
        heapify(arr, n, largest); //对交换后的子结点下的子树接着进行堆调整，递归调
用heapify, largest为对应子树的根结点索引
    }
}

//通过二叉堆实现优先队列进行自定义排序
void findOrderMRT(Program arr[], int n)
{
    for (int i = n / 2 - 1; i >= 0; i--) //自底向上建堆，通过n/2-1获得最右边最后一个
非叶结点开始进行堆调整，依次至二叉堆根结点!! (或者是(index-1)/2, index=n-1)(关键!!
自底向上进行建堆的?)
        heapify(arr, n, i); //调用heapify, i为子树根结点

    for (int i = n - 1; i > 0; i--)
    {
        swap(arr[0], arr[i]); //依次从大顶堆优先队列堆顶取出tr最大的结点放置在末
尾，实现以tr升序排序
        heapify(arr, i, 0); //由于交换后二叉堆根结点不再为tr最大的结点，因此对剩
余程序继续进行大顶堆调整，其中需调整的子树根结点为0(即二叉堆根结点)，且结点数量i已减去取
出的结点个数
    }
}

//计算最小平均读取时间
double minRetrievalTime(int n, Program PHeap[], long long total)
{
    double MRT = 0;

```

```

        for (int i = 1; i <= n; i++)                //ΣΣpiklik
            MRT += PHeap[i - 1].tr / total * (n - i + 1);    //PHeap[i].tr/total为单
//输出程序的存储顺序
void printOrder(Program arr[], int n)
{
    for (int i = 0; i < n; ++i)
        cout << arr[i].id << " ";    //按顺序输出各程序的id
    cout << "\n";
}

//输入各程序存放长度li与整型读取概率
void input(int n, Program* PHeap, long long& total) //&引用传递total
{
    for (int i = 0; i < n; i++)
    {
        PHeap[i].id = i + 1; //存储程序id
        cout << "请输入编号" << i + 1 << "程序的存放长度li与读取整型概率pi: ";
        cin >> PHeap[i].len >> PHeap[i].pr; //输入各程序存放长度li与整型读取概率
        PHeap[i].tr = PHeap[i].len * PHeap[i].pr;    //计算tr=len*pr
        total += PHeap[i].pr;        //计算Σpr
    }
}

//主函数
int main()
{
    int n;    //程序个数
    cout << "请输入程序个数n: " << endl;
    cin >> n;
    Program* PHeap = (Program*)malloc(n * sizeof(Program)); //动态分配数组存储各
//程序结构体
    long long total = 0;    //初始化为0, 用于之后记录Σpr

    input(n, PHeap, total);    //输入各程序存放长度li与整型读取概率pr

    findOrderMRT(PHeap, n); //对程序进行排序使平均读取时间最小

    double MRT = minRetrievalTime(n, PHeap, total); //获取最小平均读取时间

```

```

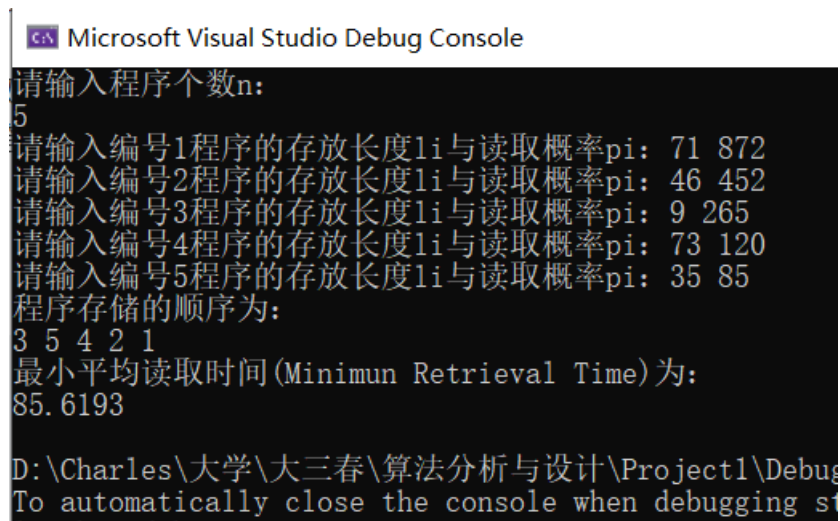
    cout << "程序存储的顺序为: " << endl;
    printOrder(PHeap, n);
    cout << "最小平均读取时间(Minimun Retrieval Time)为: " << endl;
    cout << MRT << endl;

    return 0;
}

```

4 算法运行效果

4.1 运行截屏

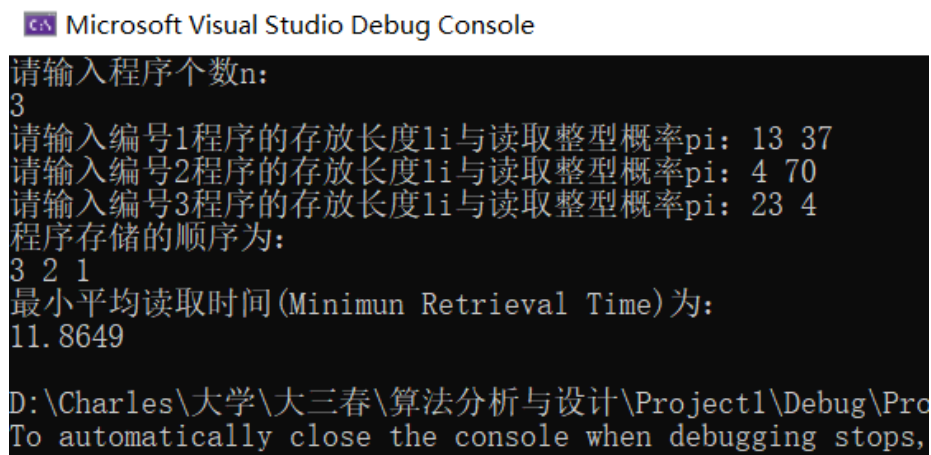


Microsoft Visual Studio Debug Console

```

请输入程序个数n:
5
请输入编号1程序的存放长度li与读取概率pi: 71 872
请输入编号2程序的存放长度li与读取概率pi: 46 452
请输入编号3程序的存放长度li与读取概率pi: 9 265
请输入编号4程序的存放长度li与读取概率pi: 73 120
请输入编号5程序的存放长度li与读取概率pi: 35 85
程序存储的顺序为:
3 5 4 2 1
最小平均读取时间(Minimun Retrieval Time)为:
85.6193
D:\Charles\大学\大三春\算法分析与设计\Project1\Debug
To automatically close the console when debugging stops,

```



Microsoft Visual Studio Debug Console

```

请输入程序个数n:
3
请输入编号1程序的存放长度li与读取整型概率pi: 13 37
请输入编号2程序的存放长度li与读取整型概率pi: 4 70
请输入编号3程序的存放长度li与读取整型概率pi: 23 4
程序存储的顺序为:
3 2 1
最小平均读取时间(Minimun Retrieval Time)为:
11.8649
D:\Charles\大学\大三春\算法分析与设计\Project1\Debug\Pro
To automatically close the console when debugging stops,

```

4.2 结果分析

(1) 时间复杂度为 $O(n\log n)$

该算法的核心为通过二叉堆实现优先队列,对所有程序按照其读取时间进行排序输出。初始化建堆从最后一个非叶子结点开始自底向上对各结点子树进行堆调整,其时间复杂度为 $O(n)$ 。而在排序并重建堆的过程中,每一次堆重建的下滤操作时间复杂度为 $O(\log n)$,而重建堆一共需要 $n-1$ 次循环,因此排序的时间复杂度为 $O(n\log n)$ 。最后在计算最小平均读取时间函数中进行了一次遍历所有程序时间复杂度为 $O(n)$ 。因此综上,该算法时间复杂度为 $O(n\log n)$ 。

(2) 空间复杂度 $O(n)$

该算法通过动态分配一维数组的方式来对二叉堆进行存储,其数组的大小为 n ,因此空间复杂度为 $O(n)$,而对于关键的二叉堆排序算法,由于使用数组进行的是就地(in-place)排序,不会需要多余的空间,因此其辅助空间(auxiliary space)为 $O(1)$ 。因此综上,该算法空间复杂度为 $O(n)$ 。