

## 数据结构与算法 II 作业 (10.13)

中国人民大学 信息学院 崔冠宇 2018202147

助教学长/学姐好，我之前的实验报告本来已完成，但是看到的提交要求中没有实验报告，于是就没交，我将之前完成的实验报告和现在的作业一起补交，如果能更改之前的成绩，烦请您更改一下，谢谢！

注：代码实现均为 C++17 标准，请使用新款编译器！

**P90, T6.4-3** 对于一个按升序排列的包含  $n$  个元素的有序数组  $A$  来说，HEAPSORT 的时间复杂度是多少？如果  $A$  是降序呢？

解：首先给出 HEAPSORT( $A$ ) 函数的伪码：

```
1 HEAPSORT(A):
2     BUILD-MAX-HEAP(A)
3     for i = length[A] downto 2
4         swap(A[1], A[i])
5         heap-size[A] = heap-size[A] - 1
6         MAX-HEAPIFY(A, 1)
```

分两种情况分析（二者的区别只在于建堆，其他基本相同）：

1. 升序情况：因为我们要构建的是大顶堆，故在升序时 BUILD-MAX-HEAP 的复杂度是  $\Theta(n)$  的，而后面运行了  $\Theta(n)$  次循环，每次循环中调用的 MAX-HEAPIFY 都是  $\Theta(\log n)$  的，所以总时间复杂度

$$T(n) = \Theta(n) + \Theta(n) \cdot \Theta(\log n) = \Theta(n \log n)$$

2. 降序情况：尽管我们要构建的是大顶堆，在降序时 BUILD-MAX-HEAP 的复杂度仍然是  $\Theta(n)$ （在它内部调用的 MAX-HEAPIFY 是  $\Theta(1)$  的，但因为有一个  $\Theta(n)$  的循环，所以是  $\Theta(n)$  的），而后面运行了  $\Theta(n)$  次循环，每次循环中调用的 MAX-HEAPIFY 都是  $\Theta(\log n)$  的，所以总时间复杂度

$$T(n) = \Theta(n) + \Theta(n) \cdot \Theta(\log n) = \Theta(n \log n)$$

综上，在两种情况下，时间复杂度均为  $T(n) = \Theta(n \log n)$ 。

**P97, T7.1-2** 当数组  $A[p..r]$  中的元素都相同时，PARTITION 返回的  $q$  值是什么？修改 PARTITION，使得当数组  $A[p..r]$  中所有元素的值都相同时， $q = \lfloor (p+r)/2 \rfloor$ 。

解：首先给出 PARTITION( $A, p, r$ ) 的伪代码：

```
1 PARTITION(A, p, r):
2     x = A[r] //x为主元
3     i = p - 1
```

```

4   for j = p to r - 1
5       if A[j] <= x
6           i = i + 1
7           swap(A[i], A[j])
8   swap(A[i + 1], A[r])
9   return i + 1

```

当所有元素都相同时，显然返回的是  $q = r$ 。要想让  $\text{PARTITION}(A, p, r)$  返回的  $q = \lfloor (p+r)/2 \rfloor$ ，可以设置一个标志 (FLAG)，遇到相等的情况将标志取反，例如：

```

1 PARTITION(A, p, r):
2     x = A[r] //x为主元
3     i = p - 1
4     flag = true
5     for j = p to r - 1
6         if A[j] < x
7             i = i + 1
8             swap(A[i], A[j])
9         else if A[j] == x
10            i = i + 1
11            if flag
12                swap(A[i], A[j])
13            flag = not flag
14     swap(A[i + 1], A[r])
15     return i + 1

```

**上机题 1** 用最小堆实现最小优先队列，并写出 HEAP-MINIMUM、HEAP-EXTRACT-MIN、HEAP-DECREASE-KEY 以及 MIN-HEAP-INSERT 的伪代码。

**解：**将老师讲义中关于大顶堆的代码全部调整为小顶堆的即可。由于全部代码过长，此处仅给出部分功能伪代码：

```

1 HEAP-MINIMUM(A):
2     return A[1]

```

显然算法时间复杂度是  $\Theta(1)$  的。

```

1 HEAP-EXTRACT-MIN(A):
2     if heap-size[A] < 1
3         error "heap underflow"
4     min = A[1]
5     A[1] = A[heap-size[A]]

```

```

6     heap-size[A] = heap-size[A] - 1
7     MIN-HEAPIFY(A, 1)
8     return min

```

由于维持堆的性质的 MIN-HEAPIFY 的复杂度是  $O(\log n)$  的，所以该算法也是  $O(\log n)$  的。

```

1 HEAP-DECREASE-KEY(A, i, key):
2     if key > A[i]
3         error "new key is larger than current key"
4     A[i] = key
5     while i > 1 and A[PARENT(i)] > A[i]
6         swap(A[i], A[PARENT(i)])
7         i = PARENT(i)

```

由于 PARENT 最多执行次数与堆的高度有关，所以该算法是  $O(\log n)$  的。

```

1 MIN-HEAP-INSERT(A, key)
2     heap-size[A] = heap-size[A] + 1
3     A[heap-size[A]] = +INFINITY
4     HEAP-DECREASE-KEY(A, heap-size[A], key)

```

由于 HEAP-DECREASE-KEY 是  $O(\log n)$  的，所以该算法也是  $O(\log n)$  的。

**程序运行截图：**代码参见 T6.5-3.cpp。以下是我用最小堆实现的最小优先队列的运行截图，其中主要测试了最小优先队列的插入和弹出，同时附带堆的情况：

```
CuiGuanyu@localhost:~  
生成随机序列：  
68 98 75 25 82 79 89  
测试插入节点：  
插入 68  
堆情况：68  
插入 98  
堆情况：68 98  
插入 75  
堆情况：68 98 75  
插入 25  
堆情况：25 68 75 98  
插入 82  
堆情况：25 68 75 98 82  
插入 79  
堆情况：25 68 75 98 82 79  
插入 89  
堆情况：25 68 75 98 82 79 89  
测试弹出最小值：  
弹出 25  
已弹出情况：25  
弹出 68  
已弹出情况：25 68  
弹出 75  
已弹出情况：25 68 75  
弹出 79  
已弹出情况：25 68 75 79  
弹出 82  
已弹出情况：25 68 75 79 82  
弹出 89  
已弹出情况：25 68 75 79 82 89  
弹出 98  
已弹出情况：25 68 75 79 82 89 98
```

**上机题 2** 用优先队列实现先进先出队列以及栈。

**解：**以最小优先队列为例。因为最小优先队列弹出的总是关键字最小的元素，所以要使用最小优先队列实现先入先出的队列，只需要让入队的节点的**关键字递增**即可；要实现后入先出的栈，只需要让入栈的节点的**关键字递减**即可。

**程序运行截图：**代码参见 T6.5-7.cpp。以下是我用最小优先队列实现的队列和栈的运行截图，其中主要测试了队列的入队出队以及栈的入栈和弹出：

```
CuiGuanyu@localhost:~
-----队列-----
生成随机序列：
97 43 23 34 97
测试入队：
插入 97
插入 43
插入 23
插入 34
插入 97
测试出队：
弹出 97
弹出 43
弹出 23
弹出 34
弹出 97
已弹出情况：97 43 23 34 97
-----栈-----
生成随机序列：
10 96 54 94 22
测试入栈：
插入 10
插入 96
插入 54
插入 94
插入 22
测试出栈：
弹出 22
弹出 94
弹出 54
弹出 96
弹出 10
已弹出情况：22 94 54 96 10
```

附录——最小堆的定义及实现: **heap.h**:

```
1 #ifndef HEAP_H
2 #define HEAP_H
3
4 #include <vector>
5 #include <optional>
6 #include <utility>
7 #include <cmath>
8 #include <iostream>
```

```

9
10 // 最小堆
11 // K: 关键字类型, 要求 K 可比
12 // V: 值类型, 可空
13 template <typename K, typename V = int>
14 class MinimumHeap
15 {
16     public:
17         // 构造与析构
18         MinimumHeap()
19         {
20             _heapSize = 0;
21             _isHeap = false;
22             _isSorted = false;
23         }
24         ~MinimumHeap(){}
25         // 给内部数组赋值
26         void assignArray(const std::vector<std::pair<K, std::optional<V>>> & arr);
27         // 返回内部数组
28         std::vector<std::pair<K, std::optional<V>>> getArray();
29         // 对应 MIN-HEAPIFY
30         // 假定i下标的左右子树均为最小堆, 调整i使得满足最小堆
31         void heapify(long long i);
32         // 对应 BUILD-MIN-HEAP
33         // 从无序数组建立最小堆
34         void buildHeap();
35         // 对应 HEAP-SORT
36         // 排序, 但会破坏堆
37         void sort();
38         // 对应 MINIMUM-HEAP
39         // 仅返回最小值
40         std::pair<K, std::optional<V>> minimum();
41         // 对应 HEAP-EXTRACT-MIN
42         // 弹出最小值节点
43         std::pair<K, std::optional<V>> extractMinimum();
44         // 对应 HEAP-DECREASE-KEY
45         // 减小某键

```

```
46     void decreaseKey(long long i, K newKey);
47     // 对应 MIN-HEAP-INSERT
48     // 插入一个节点
49     void insert(K key, std::optional<V> value = std::nullopt);
50     // 打印
51     void print();
52 private:
53     // 内部数组
54     std::vector<std::pair<K, std::optional<V> >> _arr;
55     // 其中堆占的大小
56     long long _heapSize;
57     // 是否是堆
58     bool _isHeap;
59     // 数组是否有序
60     bool _isSorted;
61 };
62
63 // 将外部数组赋值给堆
64 //
65 template <typename K, typename V>
66 void MinimumHeap<K, V>::assignArray(const std::vector<std::pair<K, std::optional<V>
    >>> & arr)
67 {
68     _arr = arr;
69     _heapSize = 0;
70     _isHeap = false;
71     _isSorted = false;
72 }
73
74 // 返回数组
75 //
76 template <typename K, typename V>
77 std::vector<std::pair<K, std::optional<V>>> MinimumHeap<K, V>::getArray()
78 {
79     return _arr;
80 }
81
```

```

82 // 假定左右两堆已成最小堆，调整i
83 // 保持堆的性质
84 //  $T(n)=O(\log n)$ 
85 template <typename K, typename V>
86 void MinimumHeap<K, V>::heapify(long long i)
87 {
88     assert(i >= 0 && i < _arr.size());
89     // 最小值下标
90     long long smallest = i;
91     // 左子树的下标
92     long long lIndex = 2 * i + 1;
93     // 右子树的下标
94     long long rIndex = 2 * i + 2;
95     // 左子树在范围内且比i小
96     if(lIndex <= _heapSize - 1 && _arr[lIndex].first < _arr[i].first)
97     {
98         smallest = lIndex;
99     }
100     // 右子树在范围内且比刚才更小的还小
101     if(rIndex <= _heapSize - 1 && _arr[rIndex].first < _arr[smallest].first)
102     {
103         smallest = rIndex;
104     }
105     // 交换并调整
106     if(smallest != i)
107     {
108         std::swap(_arr[i], _arr[smallest]);
109         heapify(smallest);
110     }
111     // _isSorted = false;
112 }
113
114 // 建立小顶堆
115 //  $T(n)=O(n)$ 
116 template <typename K, typename V>
117 void MinimumHeap<K, V>::buildHeap()
118 {

```



```
119     // 是堆，直接返回
120     if(!_isHeap)
121     {
122         _isSorted = false;
123         return;
124     }
125     // 从后往前建立
126     _heapSize = _arr.size();
127     for(long long int i = _arr.size() / 2 - 1; i >= 0; i--)
128     {
129         heapify(i);
130     }
131     // 修改状态
132     _isHeap = true;
133     _isSorted = false;
134 }
135
136 // 堆排序
137 //  $T(n) = O(n \log n)$ 
138 template <typename K, typename V>
139 void MinimumHeap<K, V>::sort()
140 {
141     // 是排好序的就不用排了
142     if(_isSorted)
143     {
144         _isHeap = false;
145         return;
146     }
147     // 建堆
148     buildHeap();
149     for(long long int i = _arr.size() - 1; i >= 1; i--)
150     {
151         // 交换
152         std::swap(_arr[0], _arr[i]);
153         _heapSize--;
154         // 调整
155         heapify(0);
```

```
156     }
157     _isHeap = false;
158     _isSorted = true;
159 }
160
161 // 返回最小值
162 //  $T(n)=O(1)$ 
163 template <typename K, typename V>
164 std::pair<K, std::optional<V>> MinimumHeap<K, V>::minimum()
165 {
166     // 对于堆来说，最小值在最上
167     if(_isHeap)
168     {
169         return _arr[0];
170     }
171     // 对于已排好的数组来说，最小值在最后
172     if(!_isSorted)
173     {
174         return _arr[_arr.size() - 1];
175     }
176 }
177
178 // 释出最小值节点
179 //  $T(n)=O(\log n)$ 
180 template <typename K, typename V>
181 std::pair<K, std::optional<V>> MinimumHeap<K, V>::extractMinimum()
182 {
183     // 保证是堆
184     assert(_isHeap && _heapSize >= 1);
185     // 最小值
186     std::pair<K, std::optional<V>> min = _arr[0];
187     // 交换
188     _arr[0] = _arr[_heapSize - 1];
189     // 减小堆的规模
190     _heapSize--;
191     // 调整
192     heapify(0);
```

```

193     return min;
194 }
195
196 // 减少键
197 //  $T(n)=O(\log n)$ 
198 template <typename K, typename V>
199 void MinimumHeap<K, V>::decreaseKey(long long i, K newKey)
200 {
201     assert(i >= 0 && i < _arr.size());
202     // 保证新键比旧键小
203     assert(newKey < _arr[i].first);
204     _arr[i].first = newKey;
205     long long index = i;
206     while(index > 0 && _arr[(index - 1) / 2].first > _arr[index].first)
207     {
208         std::swap(_arr[(index - 1) / 2], _arr[index]);
209         index = (index - 1) / 2;
210     }
211 }
212
213 // 插入节点
214 //  $T(n)=O(\log n)$ 
215 template <typename K, typename V>
216 void MinimumHeap<K, V>::insert(K key, std::optional<V> value)
217 {
218     // 在堆最后插入一个节点，值为该类型的最大值
219     _arr.insert(_arr.begin() + _heapSize, std::make_pair(std::numeric_limits<K>::
max(), value));
220     // 堆扩容
221     _heapSize++;
222     // 减小键到合适大小
223     decreaseKey(_heapSize - 1, key);
224 }
225
226 // 打印堆
227 //  $T(n)=O(n)$ 
228 template <typename K, typename V>

```

```

229 void MinimumHeap<K, V>::print()
230 {
231     for(auto i : _arr)
232     {
233         std::cout << i.first << " ";
234     }
235 }
236 #endif

```

附录——用最小堆实现最小优先队列: **priorityqueue.h**:

```

1 #ifndef PRIORITYQUEUE_H
2 #define PRIORITYQUEUE_H
3
4 #include "heap.h"
5
6 template <typename K, typename V = int>
7 class MinimumPriorityQueue
8 {
9     public:
10         // 构造与析构
11         MinimumPriorityQueue()
12         {
13             _heap.buildHeap();
14         }
15         ~MinimumPriorityQueue(){}
16         // 支持的操作
17         // 直接用最小堆实现
18         std::pair<K, std::optional<V>> minimum()
19         {
20             return _heap.minimum();
21         }
22         std::pair<K, std::optional<V>> extractMinimum()
23         {
24             return _heap.extractMinimum();
25         }
26         void decreaseKey(long long i, K newKey)
27         {

```

```

28         _heap.decreaseKey(i, newKey);
29     }
30     void insert(K key, std::optional<V> value = std::nullopt)
31     {
32         _heap.insert(key, value);
33     }
34     void print()
35     {
36         _heap.print();
37     }
38     private:
39         MinimumHeap<K, V> _heap;
40 };
41
42 #endif

```

附录——T6.5-3 测试: **T6.5-3.cpp**:

```

1 #include <random>
2 #include <vector>
3 #include <iostream>
4
5 #include "priorityqueue.h"
6
7 // T6.5-3 测试
8 // (优先队列)
9 int main(int argc, char * argv[])
10 {
11     // 测试节点数
12     const int testTimes = 7;
13     // 将用于为随机数引擎获得种子
14     std::random_device rd;
15     // 以播种标准 mersenne_twister_engine
16     std::mt19937 gen(rd());
17     // 随机数分布生成器 [a, b]
18     std::uniform_int_distribution<> dis(0, 100);
19
20     std::cout << "生成随机序列:" << std::endl;

```

```

21     std::vector<int> seq;
22     // 生成随机数序列
23     for(int i = 0; i < testTimes; i++)
24     {
25         int randVal = dis(gen);
26         seq.push_back(randVal);
27     }
28     std::copy(seq.begin(), seq.end(), std::ostream_iterator<int>(std::cout, " "));
29     std::cout << std::endl;
30
31     // 优先队列
32     MinimumPriorityQueue<int> q;
33
34     std::cout << "测试插入节点:" << std::endl;
35     for(int i = 0; i < testTimes; i++)
36     {
37         std::cout << "插入 " << seq[i] << std::endl;
38         q.insert(seq[i]);
39         std::cout << "堆情况: ";
40         q.print();
41         std::cout << std::endl;
42     }
43     std::cout << "测试弹出最小值:" << std::endl;
44     std::vector<int> outSeq;
45     for(int i = 0; i < testTimes; i++)
46     {
47         int min = q.extractMinimum().first;
48         std::cout << "弹出 " << min << std::endl;
49         outSeq.push_back(min);
50         std::cout << "已弹出情况: ";
51         std::copy(outSeq.begin(), outSeq.end(), std::ostream_iterator<int>(std::
cout, " "));
52         std::cout << std::endl;
53     }
54     return 0;
55 }

```

```
1 #include <random>
2 #include <vector>
3 #include <iostream>
4
5 #include "queue.h"
6 #include "stack.h"
7
8 // T6.5-7 测试
9 int main(int argc, char * argv[])
10 {
11     // 测试节点数
12     const int testTimes = 5;
13     // 将用于为随机数引擎获得种子
14     std::random_device rd;
15     // 以播种标准 mersenne_twister_engine
16     std::mt19937 gen(rd());
17     // 随机数分布生成器 [a, b]
18     std::uniform_int_distribution<> dis(0, 100);
19
20     std::cout << "-----队列-----" << std::endl;
21     std::cout << "生成随机序列:" << std::endl;
22     std::vector<int> seq;
23     // 生成随机数序列
24     for(int i = 0; i < testTimes; i++)
25     {
26         int randVal = dis(gen);
27         seq.push_back(randVal);
28     }
29     std::copy(seq.begin(), seq.end(), std::ostream_iterator<int>(std::cout, " "));
30     std::cout << std::endl;
31
32     // 队列
33     Queue<int> q;
34     std::cout << "测试入队:" << std::endl;
35     for(int i = 0; i < testTimes; i++)
36     {
37         std::cout << "插入 " << seq[i] << std::endl;
```

```
38         q.enqueue(seq[i]);
39     }
40     std::cout << "测试出队:" << std::endl;
41     std::vector<int> outSeq;
42     for(int i = 0; i < testTimes; i++)
43     {
44         int min = q.dequeue();
45         std::cout << "弹出 " << min << std::endl;
46         outSeq.push_back(min);
47     }
48     std::cout << "已弹出情况: ";
49     std::copy(outSeq.begin(), outSeq.end(), std::ostream_iterator<int>(std::cout, "
    "));
50     std::cout << std::endl;
51
52
53     std::cout << "-----栈-----" << std::endl;
54     std::cout << "生成随机序列:" << std::endl;
55     seq.clear();
56     // 生成随机数序列
57     for(int i = 0; i < testTimes; i++)
58     {
59         int randVal = dis(gen);
60         seq.push_back(randVal);
61     }
62     std::copy(seq.begin(), seq.end(), std::ostream_iterator<int>(std::cout, " "));
63     std::cout << std::endl;
64
65     // 栈
66     Stack<int> s;
67     std::cout << "测试入栈:" << std::endl;
68     for(int i = 0; i < testTimes; i++)
69     {
70         std::cout << "插入 " << seq[i] << std::endl;
71         s.push(seq[i]);
72     }
73     std::cout << "测试出栈:" << std::endl;
```



```
74     outSeq.clear();
75     for(int i = 0; i < testTimes; i++)
76     {
77         int min = s.pop();
78         std::cout << "弹出 " << min << std::endl;
79         outSeq.push_back(min);
80     }
81     std::cout << "已弹出情况: ";
82     std::copy(outSeq.begin(), outSeq.end(), std::ostream_iterator<int>(std::cout, "
    "));
83     std::cout << std::endl;
84     return 0;
85 }
```