Lab 1 Report Template

1. GivImplement algorithms with MinHeap for job scheduling, assuming job with a lower key (expected running time) has higher priority.

1. add a time keeper, show the running status evolution of the jobs
2. add job scheduler, to add a job into the MinHeap.
3. add a trouble-maker, to randomly pick an element in the MinHelp and add a random value to its key.

要求：

(1) 每 3 秒增加一个任务（包括完成时间 key 值/名称等）
(2) 完成任务的时间为 1-10 的随机数
(3) 开始执行任务时，删除该任务；完成该任务后，可执行下一个任务
(4) 每 5 秒调整第 k 个任务，k 为 1-n 的随机数
(5) 每次删除任务前，每次调整任务前/后，按顺序打印当前 Heap 中的所有节点（包括名称/key 值等）
(6) 增/删/改任务的执行时间忽略
(7) 指定第 N 秒时可以停止（如 N=200）

A. Implementation of the MinHeap：

```python
class MinHeap(object):
    """最小堆"""
    def __init__(self):
        self.data = []  # 创建堆
        self.count = len(self.data)  # 元素数量

    def __str__(self):
        return str(self.data)

    def size(self):
        return self.count

    def is_empty(self):
        return self.count == 0

    def insert(self, item):
        # 插入元素入堆
        self.data.append(item)
        self.count += 1
        self.shift_up(self.count)

    def shift_up(self, count):
        # 将插入的元素放到合适位置，保持最小堆
        while count > 1 and self.data[(count//2)-1][0] > self.data[count-1][0]:
            self.data[(count//2)-1], self.data[count-1] = self.data[count-1], self.data[(count//2)-1]
            count //= 2
```

```python
    def extract_min(self):
        # 出堆
        if self.count > 0:
            ret = self.data[0]
            self.data[0], self.data[self.count-1] = self.data[self.count-1], self.data[0]
            self.data.pop()
            self.count -= 1
            self.shift_down(1)
            return ret

    def shift_down(self, count):
        # 将堆的索引位置元素向下移动到合适位置，保持最小堆
        while 2 * count <= self.count:
            # 证明有孩子
            j = 2 * count
            if j + 1 <= self.count:
                # 证明有右孩子
                if self.data[j][0] < self.data[j-1][0]:
                    j += 1
            if self.data[count-1][0] <= self.data[j-1][0]:
                # 堆的索引位置已经小于两个孩子节点，不需要交换了
                break
            self.data[count-1], self.data[j-1] = self.data[j-1], self.data[count-1]
            count = j

    def change(self, count, value):
        self.data[count-1][0] = value
        if count > 1 and self.data[(count//2)-1][0] > self.data[count-1][0]:
            self.shift_up(count)
        else:
            self.shift_down(count)
```

The ***insert*** will insert a new element into the MinHeap and shift it.

The ***extract_min*** will remove the root of the MinHeap and shift it.

The ***change*** will change the key value of specific node and shift the MinHeap.

The ***shift_up*** and ***shift_down*** could be used to shift the MinHeap form a specific node with different direction.

B. time keeper, job scheduler and trouble-maker:

```
min_heap = MinHeap()
mark = 1
expect_time = 1
# time keeper
for time_keeper in range(0, 100):
    # job scheduler
    if time_keeper % 3 == 0:
        print(str(time_keeper)+":Add:"+str(min_heap)+"\n-->", end=" ")
        min_heap.insert([random.randint(1, 10), "work" + str(mark)])
        print(str(min_heap))
        mark += 1
    # trouble-maker
    if not min_heap.is_empty() and time_keeper % 5 == 0:
        print(str(time_keeper) + ":Change:" + str(min_heap) + "\n-->", end=" ")
        k = random.randint(1, min_heap.count)
        min_heap.change(k, random.randint(1, 10))
        print(str(min_heap))
    # job scheduler
    if not min_heap.is_empty() and time_keeper >= expect_time:
        print(str(time_keeper) + ":Run:" + str(min_heap) + "\n-->", end=" ")
        current_work = min_heap.extract_min()
        expect_time += current_work[0]
        print(str(min_heap))
```

The time keeper increase from 0 to 100.

Since while the time keeper equal to 100, the length of the MinHeap will be too large to show in this document, I simply use all of the output before time keeper become larger than 30 as the output.

In first line of every turn of the output, the number show the value of time keeper and the word after the ":" is the operation will be operated on the MinHeap.

In second line of every turn of the output, the MinHeap without option at that second in front of the "-->". The changed MinHeap is showed after the "-->" .

The output:

```
0:Add:
[]-->[[2, 'work1']]
0:Change:
[[2, 'work1']]-->[[1, 'work1']]
1:Run:
[[1, 'work1']]-->[]
3:Add:
[]-->[[1, 'work2']]
3:Run:
[[1, 'work2']]-->[]
6:Add:
[]-->[[9, 'work3']]
6:Run:
[[9, 'work3']]-->[]
9:Add:
[]-->[[2, 'work4']]
10:Change:
[[2, 'work4']]-->[[7, 'work4']]
12:Add:
[[7, 'work4']]-->[[7, 'work4'], [7, 'work5']]
12:Run:
[[7, 'work4'], [7, 'work5']]-->[[7, 'work5']]
15:Add:
[[7, 'work5']]-->[[4, 'work6'], [7, 'work5']]
15:Change:
[[4, 'work6'], [7, 'work5']]-->[[4, 'work6'], [7, 'work5']]
18:Add:
[[4, 'work6'], [7, 'work5']]-->[[4, 'work6'], [7, 'work5'], [6, 'work7']]
19:Run:
[[4, 'work6'], [7, 'work5'], [6, 'work7']]-->[[6, 'work7'], [7, 'work5']]
20:Change:
[[6, 'work7'], [7, 'work5']]-->[[6, 'work7'], [7, 'work5']]
21:Add:
[[6, 'work7'], [7, 'work5']]-->[[6, 'work7'], [7, 'work5'], [10, 'work8']]
23:Run:
[[6, 'work7'], [7, 'work5'], [10, 'work8']]-->[[7, 'work5'], [10, 'work8']]
24:Add:
[[7, 'work5'], [10, 'work8']]-->[[4, 'work9'], [10, 'work8'], [7, 'work5']]
25:Change:
[[4, 'work9'], [10, 'work8'], [7, 'work5']]-->[[1, 'work5'], [10, 'work8'], [4, 'work9']]
27:Add:
[[1, 'work5'], [10, 'work8'], [4, 'work9']]-->[[1, 'work5'], [6, 'work10'], [4, 'work9'], [10, 'work8']]
29:Run:
[[1, 'work5'], [6, 'work10'], [4, 'work9'], [10, 'work8']]-->[[4, 'work9'], [6, 'work10'], [10, 'work8']]
30:Add:
[[4, 'work9'], [6, 'work10'], [10, 'work8']]-->[[4, 'work9'], [6, 'work10'], [10, 'work8'], [7, 'work11']]
30:Change:
[[4, 'work9'], [6, 'work10'], [10, 'work8'], [7, 'work11']]-->[[4, 'work9'], [6, 'work10'], [10, 'work8'], [7, 'work11']]
30:Run:
[[4, 'work9'], [6, 'work10'], [10, 'work8'], [7, 'work11']]-->[[6, 'work10'], [7, 'work11'], [10, 'work8']]
```