习题实验 4

本次习题实验主要涉及贪心算法。请用 LaTeX 编辑所有解答。提交文件格式为 PDF。

姓名: xxx
学号: xxxxxxxx

题目 4-1. 选择学术报告 [50 分]
王同学参加了一个学术会议，组织者在第一天安排了 n 个学术报告，对每一个学术报告 i 都标明了开始 s_i 和结束时间 f_i。这些学术活动被安排到不同的会议室，参会人员在同一时间只能选择听一个报告。王同学想在这些学术报告里面尽可能多的听学术报告。由于如果确定参加一个报告，必须完整地听完，因此王同学选择的报告里面不能有相互冲突的报告，也就是对于两个相互没有冲突的报告 i 和 j，要么 i 的开始时间大于等于 j 的结束时间，要么 j 的开始时间大于等于 i 的结束时间。比如有如下一些报告：(1, 4), (3, 5), (0, 6), (5, 7), (3, 8), (5, 9), (6, 10), (8, 11), (8, 12), (2, 13), (12, 14)，那么王同学最多选择 4 个没有冲突的报告，它们分别是 (1, 4), (5, 7), (8, 11), (12, 14)。

(a) 简述贪心算法求解以上问题的过程，并分析其时间复杂度。

解答：本问题可以用贪心算法解决。具体过程如下：
1. 首先将所有学术报告按照结束时间 f_i 从小到大排序。
2. 依次选择当前能参加且结束时间最早的报告，将其加入结果集合。
3. 每次选择后，更新当前时间为该报告的结束时间，继续选择下一个不冲突的报告。
4. 重复上述过程，直到所有报告都考察完毕。

这种贪心策略保证了每次选择后，留给后续报告的时间最多，从而使得最终选择的报告数量最多。

时间复杂度分析:
排序需要 $O(n \log n)$，遍历一次为 $O(n)$，总复杂度为 $O(n \log n)$。

(b) 给出以上按贪心算法求解的代码。

解答:

```
def activity_selection(activities):
    # activities: [(start, finish), ...]
    # 按结束时间排序
    activities.sort(key=lambda x: x[1])
    n = len(activities)
    res = []
    last_end = -float('inf')
    for s, f in activities:
        if s >= last_end:
            res.append((s, f))
            last_end = f
    return res

# 示例
activities = [(1, 4), (3, 5), (0, 6), (5, 7), (3, 8), (5, 9), (6, 10), (8, 11), (8,
selected = activity_selection(activities)
print("最多可选的报告数: ", len(selected))
print("选择的报告: ", selected)
```

题目 4-2. 单源最短路径问题 [50 分]
给定一个有向图 G，图中的边的权重值非负，要求找出从出发点 s 到图中其它各个节点的最短路径。

(a) 以上问题可以用 Dijkstra 算法求解，请描述其过程，并给出时间复杂度分析。

解答：Dijkstra 算法用于解决带非负权重的有向图的单源最短路径问题。其基本过程如下：

1. 初始化所有节点的最短距离为无穷大，起点 s 的距离为 0。
2. 使用优先队列（小根堆）维护当前未确定最短路径的节点及其距离。
3. 每次从队列中取出距离最小的节点 u，并对其所有邻居节点 v 进行松弛操作：如果通过 u 到 v 的距离更短，则更新 v 的距离并将其加入队列。
4. 重复上述过程，直到所有节点的最短路径都被确定。

时间复杂度分析：

使用优先队列时，复杂度为 $O((n+m)\log n)$，其中 n 为节点数，m 为边数。

(b) 请给出一个利用优先队列的 Dijkstra 算法代码实现。

解答：

```
import heapq

def dijkstra(graph, start):
    # graph: {u: [(v, w), ...], ...}
    # start: 起点
    dist = {node: float('inf') for node in graph}
    dist[start] = 0
    pq = [(0, start)]
    while pq:
        d, u = heapq.heappop(pq)
        if d > dist[u]:
            continue
        for v, w in graph[u]:
            if dist[v] > dist[u] + w:
                dist[v] = dist[u] + w
                heapq.heappush(pq, (dist[v], v))
    return dist

# 示例
```

```
graph = {
    'A': [('B', 1), ('C', 4)],
    'B': [('C', 2), ('D', 5)],
    'C': [('D', 1)],
    'D': []
}
start = 'A'
distances = dijkstra(graph, start)
print("从A出发到各点的最短距离：", distances)
```