设计

想法

我创建了一个 “自定义类型”, 自定义类型中, 保存有: 索引(i), 开始时间(s), 结束时间(f), 然后我定义了一个比较的方法, 因为heapq需要用到这个, 这个就有点像C++中的重载运算符(<)之类的, 然后我还定义了对象的打印的方法, 类似于C++中重载(<<)

定义了一个函数`is_compatible`, 用来判断是否兼容的

然后定义了一个辅助函数, 传入已经打包好的`act_arr`, 将`act_arr`堆化, 其中, (剩余的)最晚结束的事件永远在堆顶 先将最晚开始的那个元素, 放到传入到result中, 然后进入循环, 循环的终止条件是: 堆空. 循环不变式: 我们可以保证, result中的所有元素都是 “优化子结构”(看后面的证明), 然后我们取出来result中, 最近一次加入的元素last, 让last与堆顶i进行比较, 如果(last, i)兼容, 那么就将i加入到result中(优化子结构扩大), 否则, 抛弃i

代码实现

```python
import heapq
import random


class activity:
    def __init__(self, s: int, f: int, i: int) -> None:
        self.s = s
        self.f = f
        self.i = i  # 这个i表示标号

    def __lt__(self, other) -> bool:
        return self.s > other.s

    def __str__(self) -> str:
        return f"i: {self.i}\ts: {self.s}\tf: {self.f}"


def is_compatible(a: activity, b: activity) -> bool:
    if a.s <= b.s and a.f <= b.s:
        # a在b之前, 并且a与b相容
        return True
    if b.s <= a.s and b.f <= a.s:
        return True
    return False


def solution_utils(act_arr: list) -> list:
    result = []
    heapq.heapify(act_arr)  # 建堆
    last = heapq.heappop(act_arr)
    result.append(last)
    while act_arr:
        i = heapq.heappop(act_arr)
        last = result[-1]
        if is_compatible(last, i):
            result.append(i)
    result.reverse()
    return result
```

```python
def solution(s_arr: list, f_arr: list) -> list:
    act_len: int = len(s_arr)
    act_arr: list = [
        activity(s, t, i) for (s, t, i) in zip(s_arr, f_arr, range(act_len))
    ]
    random.shuffle(act_arr)
    result = solution_utils(act_arr)
    return result


if __name__ == "__main__":
    s_arr: list = [3, 1, 5, 2, 5, 3, 8, 6, 8, 12]
    f_arr: list = [6, 4, 7, 5, 9, 8, 11, 10, 12, 14]
    result = solution(s_arr, f_arr)
    for i in result:
        print(i)
    print("hello world")
```

证明

优化子结构

引理**1 (base case)**

某个优化解包含 "活动n"(其中, "活动n"是最晚开始的活动)

1. 平凡的: 如果一个优化解直接包含 "活动n"
2. 构造: 如果一个优化解并不直接包含 "活动n", 那么有两种情况:
   - "活动k"(优化解中的最后一个元素) 与 "活动n"不兼容: (构造法) 那么从优化解中去掉 "活动k", 并加上 "活动n", 这并不影响活动的数量, 因此这个操作是合法的
   - "活动k"(优化解中的最优一个元素) 与 "活动n"兼容: 这个是伪命题, 因为可以加上 "活动n", 成为一个 "更优" 的解, 因此该情况下并不是一个 "优化解"

引理**2** (递推)

设 $S = \{1, 2, ..., n\}$ 是 n 个活动集合, $[s_i, f_i]$ 是活动的起始和终止时间, 且$s_1 \le s_2 \le ... \le s_n$, 设$A$ 是S的 "活动安排问题" 的一个 "优化解", 且 包含 "活动n", 则 $A' = A - \{n\}$ 是 $S' = \{i \in S \mid s_i \le s_n\}$ 的 "活动安排问题" 的 "优化解"

- 显然, A' 中的活动是 "相容" 的
- 即证明: A' 是 S'问题中, 最大的

(反证法) 若 $\exists B' \land |B| > |A|$, 则有 $B = B' \cup \{n\} \Rightarrow |B| = |B'| + 1 > |A'| + 1 = |A|$ 然而, 这与 A 是 "优化解" 的假设相悖

故: "活动选择问题" 具有 "优化子结构"

贪心选择性

设 $S = \{1, 2, ..., n\}$ 是 n 个 "活动" 的集合, 其中 $s_1 \le s_2 \le ... \le s_n$, let l_i 是集合 $S_i = \left\{j \in S \mid f_j \le s_{l_i + 1}\right\}$ 中 具有 "最晚" 开始时间 $s_{\{l_i\}}$ 的活动, 设 A 是 S 包含 "活动n" 的 "优化解", 则$A = \cup_{i=1}^{k} \{l_i\}$

(归纳法)

- 当 $|A| = 1$ 时, 由 "引理1", 命题成立
- 假设 $|A| < k$时, 命题(即 A 是 "优化解")成立 (归纳假设)

- 当 $|A| = k$时, 由 "引理2", $A = A_{n-1} \cup \{n\}$, 其中 A_{n-1} 是 $S_{n-1} = \left\{j \in S \mid f_j \le s_n\right\}$ 的 "优化解", 由假设, A_{n-1} 是 "优化解", 并且 "引理1" 指出, $\exists A_n$ 包含 "活动n", 故 $A_n = A_{n-1} \cup \{n\}$ 是一个 "优化解", 得证