The lazy programmer (SPOJ)

——HaoYe 6238023

1. Question Review
2. Problem solved by greedy algorithm
3. Code Detail (Python)
4. Result

Question Review

A programmer is assigned N jobs, each job i has a corresponding completion time bi and a delivery time di. To speed up the completion of a job at the delivery time, the programmer has an acceleration package of ai for job i. For each additional $1 paid, the completion time per unit of ai for job i can be reduced.

Example: (ai, bi, di) 10 100 50

The job takes 100 units to complete, but needs to be delivered at the 50 unit time, while each additional $1 paid to the programmer for that job reduces the completion time by 10 units.

Goals：

Minimize the additional costs paid to programmers while ensuring that each job is completed on time for delivery.

Problem solved by greedy algorithm

By analyzing the lazy programmer problem, I can determine that it is similar to the "activity scheduling" problem. Each task has a deadline and a time needed to complete the task, so schedule the task wisely.

The problem requires **one more requirement**, namely that each task has an acceleration package to choose from, which in turn allows all jobs to be completed in the delivery time, with the least amount of extra cost paid under this premise.

The problem is solved by a greedy algorithm, i.e., the algorithm is guaranteed to iterate in such a way that each task is completed within the delivery time, with as little or as few extra-cost acceleration packages as possible.

In the process of solving this problem, each task is first sorted in incremental order by delivery time, so that when iterating over each task, each task is iterated over by delivery time.

At the same time, a priority queue is introduced into the problem solving process, i.e., the tasks present in the queue are sorted in descending order according to the size of the acceleration packets. In turn, each task can be completed at delivery time with minimum additional cost.

Code Detail -> Node Class

```
#Node Class
class Node:
    #Record three values
    def __init__(self, a, b, d):
        self.a = a   # Unit Acceleration Package
        self.b = b   # Time required for task completion
        self.d = d   # Time of task delivery
    #Comparison method, sorted by a from largest to smallest
    def __lt__(self, other):
           # When Node nodes are placed in the priority queue, they are sorted in descending order by accelerator pack size
           # so that the current task costs the least amount of extra money to complete at the time of delivery
        if self.a > other.a:
            return True
        return False
```

Code Detail -> Main Logic

```python
t = int(input())  # Number of test cases
while t != 0:
    t = t - 1
    arr = [] # List of stored tasks
    n = int(input()) # Number of tasks

    for i in range(n):
        s = input().split()
        arr.append(Node(int(s[0]), int(s[1]), int(s[2])))

    arr = sorted(arr, key=lambda x: x.d) #Sort by d first from smallest to largest
    now = 0   # Record current time
    q = PriorityQueue()  # Priority queue, for Node nodes stored by accelerated packet size

    dic = [0] * 11111
    # Statistical array that counts how much time each corresponding a has to be eliminated and is used to calculate
the cost
    # It can be seen as a list version of map, storing the number of acceleration packs of the corresponding size
    # that need to be purchased for a particular acceleration pack size
```

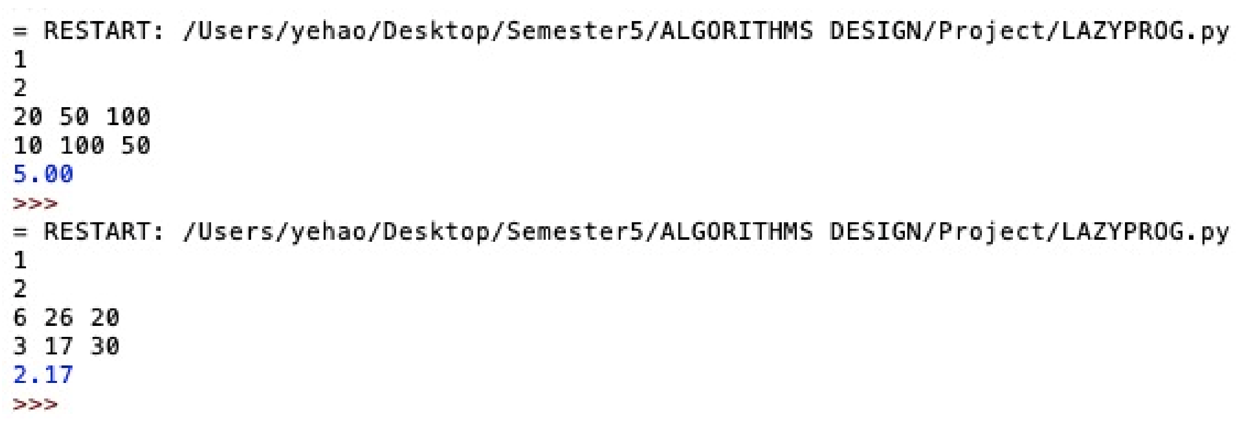
Code Detail -> Main Logic

```
    for i in arr:  # Iterate over each task

        q.put(i) # Adding the current task to the priority queue

        now += i.b  # Assuming that the current task can be completed, record the current time as now + the completion
time of the ith task, which is the current time when the next task can be run

        # The ith task will definitely be completed before its corresponding delivery time, so you can disregard the
ith task for now because it has been added to the priority queue, and it's a matter of how much to pay to get it done

        while now > i.d:  # If the current time is later than the delivery time of the ith task, i.e., the current task
cannot be completed within the specified time, then the accelerated package is bought.

            a = q.get() # Get the largest task from the priority queue that cannot be completed at the delivery time
for the current acceleration package

            if now - a.b <= i.d: # The i-th task finishes at least at i.d if now - i.d i.e. the difference between the
two times is less than the completion time of the task with the largest current acceleration package

                dic[a.a] += now - i.d # Buy the now-i.d time acceleration package first

                a.b -= (now - i.d)  #The completion time of the maximum task of the current acceleration package can be
reduced now - i.d now.

                q.put(a)  # Given that the largest task of the current acceleration package has not yet been completed,
add it to the priority queue and wait for the next iteration of the loop to see if it can be your turn

                now = i.d # Record the current time as the delivery time of the i-th task

            else: # Otherwise, now - i.d i.e. the difference between the two times is greater than the completion time
of the largest task of the current acceleration package

                now -= a.b  # Direct buyout of current assignment

                dic[a.a] += a.b # Record this task at this time to complete the time of the acceleration package it
```

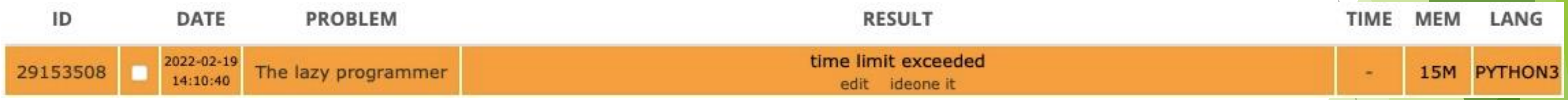
Code Detail -> Main Logic

```
    ans = 0
    for i in range(1, len(dic)):
        if dic[i] != 0: # If dic[i] is not 0, this means that the acceleration package corresponding to this value is
the one that needs to be purchased for all tasks
            ans += dic[i] / I #Do the math and add up this extra cost, which is dic[i] / i, i is the number of
acceleration packs you can buy for $1, and dic[i] is the total number of acceleration packs in units of i
    print('{:.2f}'.format(ans)) #Minimal additional fees to be paid for data
```

Result



```
= RESTART: /Users/yehao/Desktop/Semester5/ALGORITHMS DESIGN/Project/LAZYPROG.py
1
2
20 50 100
10 100 50
5.00
>>>
= RESTART: /Users/yehao/Desktop/Semester5/ALGORITHMS DESIGN/Project/LAZYPROG.py
1
2
6 26 20
3 17 30
2.17
>>>
```



ID		DATE	PROBLEM	RESULT	TIME	MEM	LANG
29153508		2022-02-19 14:10:40	The lazy programmer	time limit exceeded edit ideone it	-	15M	PYTHON3

Finally I checked this code twice and the result is correct, but when I submit it to SPOJ, it shows as Time limit exceeded.