CS325-HW3

Problem 1: Road Trip

a) Description: Loop through each hotel after the current hotel. Until we found that the distance of the hotel “Hn” minus the distance of the current hotel is greater than the maximum trip d. In order to ensure that we can walk as many miles as possible every day. So, choose previous hotel of “Hn”. Our code need loop is process.

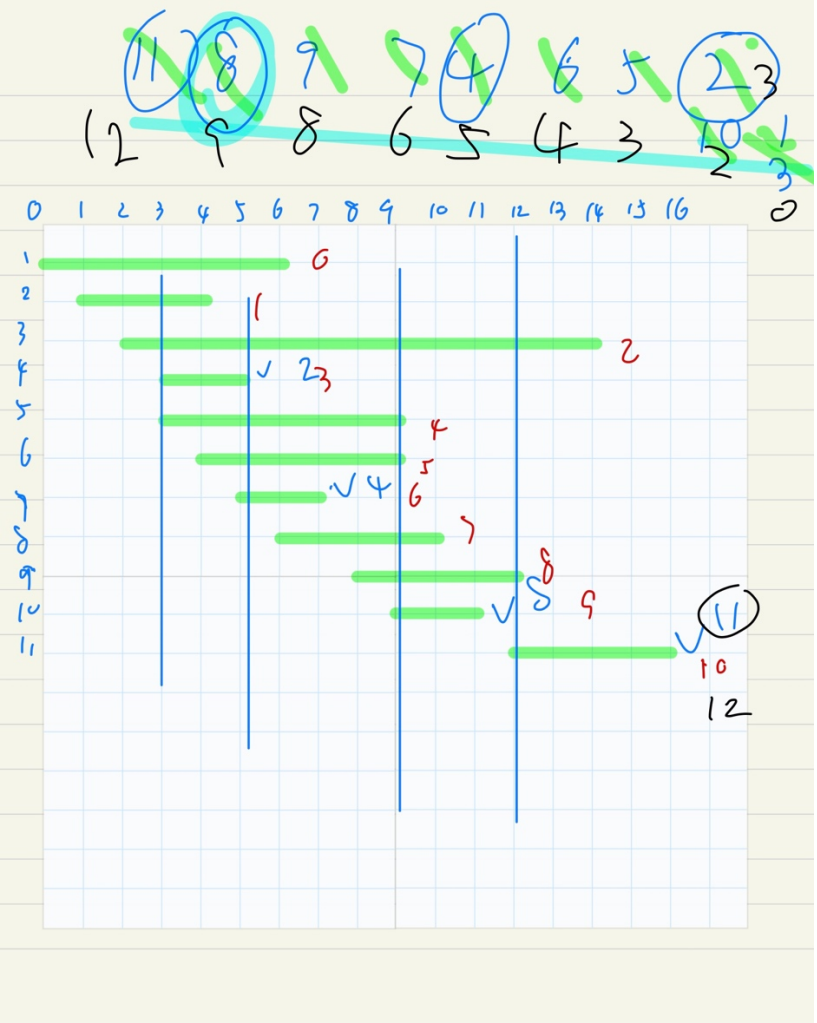
Pseudocode:

```
Hd <- {x1,x2...xn}      # the list to store distances from your start point to each hotel
n <- hotel number       # the number of hotel
D <- max distence can move
ans = {}                # store the answer
cur = 0                 # set current hotel index is 0
next = 1                # set next hotel index is 1

while 1:
    # When the distance of the next hotel minus the distance of the current hotel is greater than d
    if D > Hd[next] - Hd[cur]
        # then you can stay in the previous hotel of the next hotel
        ans{} <- next - 1
        # current hotel will become next-1
        cur = next - 1
        next += 1
        # if find the check hotal is greater than total number of hotel, just jump out
        if next > n:
            break
```

b) The worst-case scenario for a description is to check back once a time. That is, calculate the distance of each hotel up to twice. So, running time of the worst case is O(2n) = O(n). The theoretical running time should be theta(n)

Problem 2: CLRS 16-1-2 Activity Selection Last-to-Start Greedy Criteria



The Last-to-Start Greedy is same as the First-to-Finish Greedy. For the Last-to-Start Greedy, we need sort these activities by start time. Choosing the latest activity to start means that we will have enough time to schedule other activities before then.

The latest activity is selected first, so we need to look for the activity from the back. Because activities are already sorted by start time, the end time of the next activity is less than or equal to the start time of the current activity at each time the activity is satisfied, and the condition that the activity is relatively late to start. We then set the activity that meets the criteria to the current activity. With this loop, we can find the optimal solution.

Problem 3: Activity Selection Last-to-Start Implementation

Description: In this problem, there are not case number that need to be processed in the source file. So, I used the while loop and jumped out of the condition is when line index larger than the size of the row I read. Use for loops to continuously read the contents of the file. The data is added to the corresponding list. Then put these lists into the print_activity function. In this function, put them into a tuple, and sort this tuple by the start time. Then use the greedy algorithm to deal with these data.

Pseudocode:

```
printActivities(s , f, act_n):
    selected = []
    act_tuple <- tuple(act_n, s, f))
    act_tuple.sort(act_tuple[1])
    n = len(act_tuple)
    i = n-1
    selected <- act_tuple[i]
    num_activity = 1
    for j from i to 0:
        if act_tuple[j][2] <= act_tuple[i][1]:
            selected <- act_tuple[j]
            num_activity += 1
            i = j
    print("Maximum number of activities =", num_activity)
    selected = tuple(reversed(selected))
    for i in range(0, len(selected)):
        print(selected[i][0], end = " ")
```

```
printresult(filename):
    fe = open(filename)
    lines = fe.readlines()
    max_line = len(lines)
    n_act = lines[line_idx]
    while 1:
        set_num += 1
        s = []
        f = []
        act_idx = []
        line_idx += 1
        for _ from 0 to n_act:
            act_info = []
            act_info <- list(lines[line_idx])
            act_idx <- act_info[0]
            s[] <- act_info[1]
            f[] <- act_info[2]
            line_idx += 1
        print("Set", set_num)
        printActivities(s , f, act_idx)
        if line_idx < max_line:
            n_act = int(lines[line_idx])
        else:
            break

    fe.close()

printresult("act.txt")
```

I just use the sort function from the Python. The theoretical running time of my greedy algorithm is theta(n)