A Race against Time

(Week of Code 36)

By,
Anup V T
01FE15BEC032

Problem Statement:

A relay race is being organised in a school for middle school students by two high school students, Mason and Madison. Mason starts with the baton and Madison receives the baton at the final destination. There are middle school students in between Mason and Madison, and each of their heights is given. Mason's height, too, is given. Initially, the baton is with Mason and it is passed to the destination in a manner similar to a relay race.

1) At any moment, the current baton carrier has an option to hand over the baton to the student at the current position or to continue to the next position. However, if the student at the given position is taller than the current baton carrier, it is mandatory to hand over the baton because it is a safer option according to Mason.

2) It takes one second to move between consecutive positions.

3) Whenever the baton is handed over, there is a time and price associated with it.

4) The time taken, in seconds, is the absolute difference between the heights of the current baton carrier and the student to whom the baton is handed.

5) The student to whom the baton is passed charges a given price.

Note: Price charged can be negative too.

The baton must be sent to Madison in the minimum possible sum of time and price. Complete the function Solve which takes the number of middle school students, Mason's height, and heights and prices charged by middle school students as input, and return the minimum possible sum of time and price required for the baton to reach Madison.

Also, note that no handover takes place once the baton has reached the destination.

Concepts used: Dynamic Programming and Stacks.

Dynamic Programming (DP):

In computer science, mathematics, management science, economics and bioinformatics, dynamic programming (DP) (also known as dynamic optimization) is a method for solving a complex problem by breaking it down into a collection of simpler subproblems, solving each of those subproblems just once, and storing their solutions. The next time the same subproblem occurs, instead of recomputing its solution, one simply looks up the previously computed solution, thereby saving computation time at the expense of a (hopefully) modest expenditure in storage space. (Each of the subproblem solutions is indexed in some way, typically based on the values of its input parameters, so as to facilitate its lookup.) The technique of storing solutions to subproblems instead of recomputing them is called "memoization". The intuition behind dynamic programming is that we trade space for time, i.e. to say that instead of calculating all the states taking a lot of time but no space, we take up space to store the results of all the sub-problems to save time later.

Formula of DP for given problem:

$Dp[i]=\min_{i<j<=k}(dp[j] + (j – i) + abs(height[j]-height[i]) + price[j])$

Maintain a dp state for each index, which denotes the minimum sum of price and height if the activity was to start at that index. At index 'i', lets denote the index with the next greater height as 'k'.

Solution in Python:

```
#!/bin/python

import sys

#Reading the input

n = int(raw_input())

h = int(raw_input())

hlist = [h] + [int(t) for t in raw_input().strip().split()]

clist = [0] + [int(t) for t in raw_input().strip().split()]
```

```
st = []
best = [1000000000000] * n
best_dip = [1000000000000] * n
for i in xrange(n - 1, -1, -1):
   h, c = hlist[i], clist[i]
   while st:
      cur = st[-1]
      if h < hlist[cur]:
         best[i] = min(best[i], clist[i] + best[cur] + (hlist[cur] - h))
         break
      t = min(best[cur], best_dip[cur]) + (h - hlist[cur])
      best[i] = min(best[i], clist[i] + t)
      best_dip[i] = min(best_dip[i], t)
      st.pop()
   if not st:
      best[i] = min(best[i], clist[i])
   st.append(i)
print(best[0] + n)
```

References:

https://www.codechef.com/wiki/tutorial-dynamic-programming

https://www.hackerearth.com/practice/algorithms/dynamic-programming

https://www.geeksforgeeks.org/dynamic-programming/#basicProblems

https://web.stanford.edu/class/cs97si/04-dynamic-programming.pdf