**ADA hw4 – 2** B08902065 資工二 洪易

Cite:

B08902013 張永達 B08902127 林歆凱 B08902063 陳羿穎 B08902075 林耘平

3. Lonely Christmas

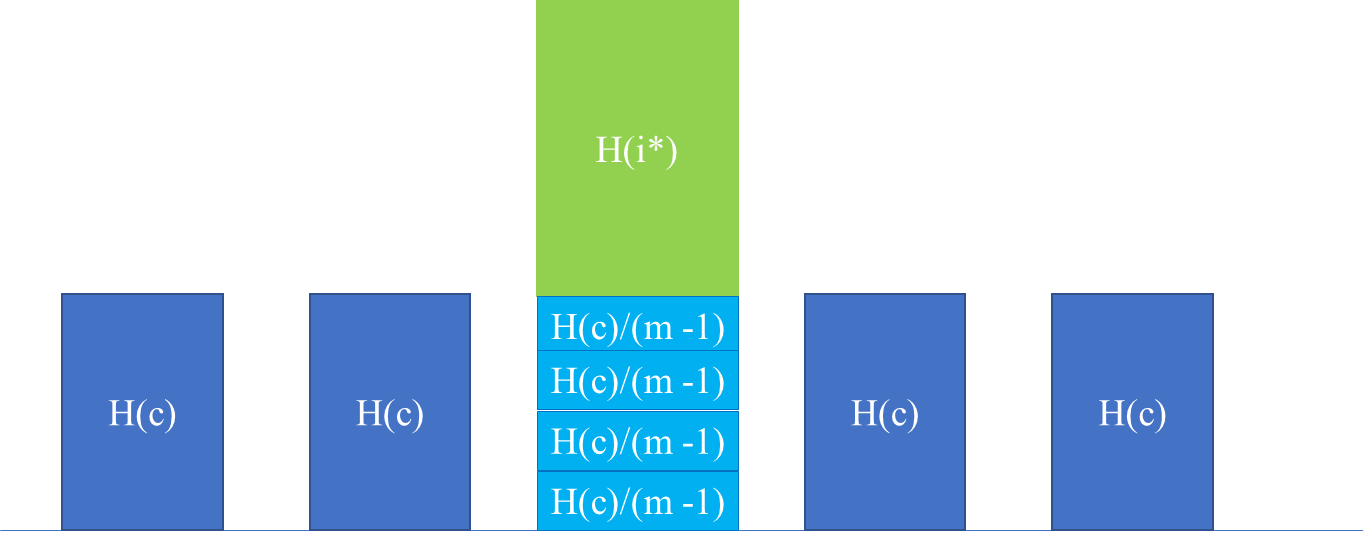
(a)

To TA:

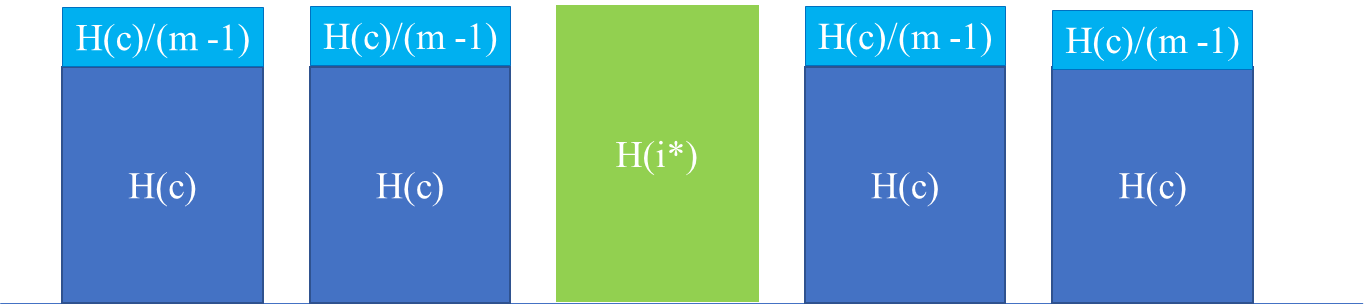
I first think of an interesting solution, but I was convinced by peers that sol\_2 is better, I’d like to make sure if sol\_1 is also an acceptable solution? Could you please grade sol\_2 and also tell me whether sol\_1 is OK? Thanks.

Sol\_1:

We can see that there will exist i\* that decide Hmax. It’s trivial that the worst case (max (H­max / OPT(I))) happens when h(i\*) is placed while current height of all towers is same and it’s placed last. Let’s call the current height before i\* is placed h(c). Therefore, Hmax = h(i\*) + h(c). However, OPT(I) = max ( h(i\*), h(c) \* m / (m - 1) ) as the following figure shown. As a result, while and h(i\*) = h(c), we can see that OPT(I) = 2 \* Hmax in the worst case. Otherwise, 2 \* Hmax < OPT(I), and thus it’s a *2-approximation* algorithm. The following two graphs show the worst case in the algorithm of Hmax and OPT(I), I take m = 5 for instance.



Hmax = h(i\*) + h(c)

OPT(I) = max (h(i\*), h(c) \* m / (m - 1))

When m , and h(i\*) = h(c), the maximum (Hmax / OPT(I)) occurs, which is 2.

Sol\_2:

I split the height of tower i\* is into two part, Hmax = h(i\*) + (Hmax - h(i\*)). It’s trivial to say that h(i\*) Moreover, (Hmax – h(i\*)) OPT(I), Thus Hmax . It’s a *2-approximation* algorithm.

(b)

I consider two cases. I\* in B’ and i\* not in B’.

First, if I\* in B’, then it’s arranged by brute force algorithm and be placed in optimal way. Therefore, the Hmax = OPT(I) in this case.

Otherwise, if I\* is not in B’. I split the height of tower i\* is into two part, Hmax = h(i\*) + (Hmax - h(i\*)). It’s trivial to say that h(i\*) Moreover, (Hmax – h(i\*)) OPT(I), Thus Hmax .

In both cases, Hmax , it’s a *-approximation* algorithm.

(c)

Same as (b). Let’s consider two cases, i\* in B and i\* not in B.

First, if I\* in B’, which means that h’(i) . We can see that h’(i) > , and Hmax, thus, for each tower, the maximum boxes there can be placed is at most . Moreover, for each box, the maximum error between h(i) and h’(i) is Therefore, the maximum error between Hmax come from h’ and Hmax come from h is . Thus, as Hmax from h’ , Hmax from h .

Second, if I\* not in B’. We split the height of tower i\* is into two part, Hmax = h(i\*) + (Hmax - h(i\*)). It’s trivial to say that h(i\*) Moreover, (Hmax – h(i\*)) OPT(I) , else i\* won’t be placed on this tower. As a result, Hmax = h(i\*) + (Hmax - h(i\*))

According to two cases, Hmax . QED.

(d - 1)

If h’(i) . It’s trivial that , and Thus .

Else, if h’(i) > V, return false, no .

Therefore, , QED.

(d - 2)

(d - 3)

Let’s split the process into two steps.

First, constructing U. As there are at most boxes in a tower and kind of boxes. Construct U takes O () = O (1) time.

Second, fill in the dynamic programming table. As we need to calculate all possibility of , it equals to , as (n\_initi + 1) (n\_initi means the ith entry of ). Accordingly, the total time complexity of this algorithm is O (1) + .

(e)

I apply binary search on V and use function Partial\_Rounded (I, V) to find a algorithm. Search start from (0, ), check what is the maximum V that ORACLE (I’, V) won’t return false, and then put all box that’s not in B into the towers in greedy algorithm. Finally, I find out the answer.

For the time complexity, as and I use binary search in the algorithm, we can achieve an algorithm of , which is polynomial.