Yinyanghu's Blog

Miscellaneous Gallery of Life

[Solution] CLRS: Problem 16-4

by *Yinyanghu* on **March 20, 2014**

Tagged as: Algorithm, CLRS, Solution, Problem, Greedy Algorithm, Matroid.

Problem

Consider the following algorithm for the problem from *Section 16.5* of scheduling unit-time tasks with deadlines and penalties. Let all n time slots be initially empty, where time slot i is the unit-length slot of time that finishes at time i. We consider the tasks in order of monotonically decreasing penalty. When considering task a_j, if there exists a time slot at or before a_j's deadline d_j that is still empty, assign a_j to the latest such slot, filling it. If there is no such slot, assign task a_j to the latest of the as yet unfilled slots.

a. Argue that this algorithm always gives an optimal answer.

b. Use the fast disjoint-set forest presented in *Section 21.3* to implement the algorithm efficiently. Assume that the set of input tasks has already been sorted into monotonically decreasing order by penalty. Analyze the running time of your implementation.

Solution

Question (a)

This algorithm always gives an optimal answer, since it is actually an implementation of the algorithm scheme introduced in section 16.5. In this algorithm, we assign each task to a non-penalty latest empty time slot. If there is no such slot, we assign this task into an empty slot as late as possible. So this schedule is the extreme one among all the possible optimal schedules. If a task cannot be assigned to an available slot in this schedule, it also cannot be in all other possible schedules.

Question (b)

It is a perfect question. We could improve the running time in this case, since we only concentrate on a specific extreme schedule instead of check all possible schedules by the *lemma 16.12*. According to the algorithm, we assume that the set of input tasks has already been sorted into monotonically decreasing order by penalty, so bottleneck is the running time of finding the empty time slot.

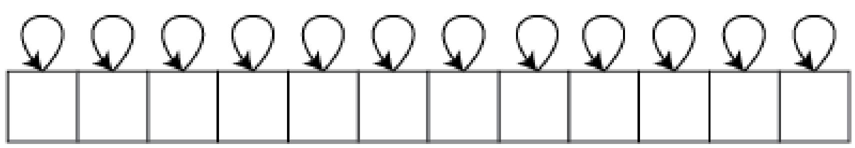
Now, we could reduce original problem into the following problem,

> Given an initially empty array A, each time we want to insert a key into the position k of array, if A_k is empty, we insert the key and done. Otherwise, we have to find the largest i such that $i < k$ and A_i is empty. And then insert the key to A_i. If no such i exists, return -1.

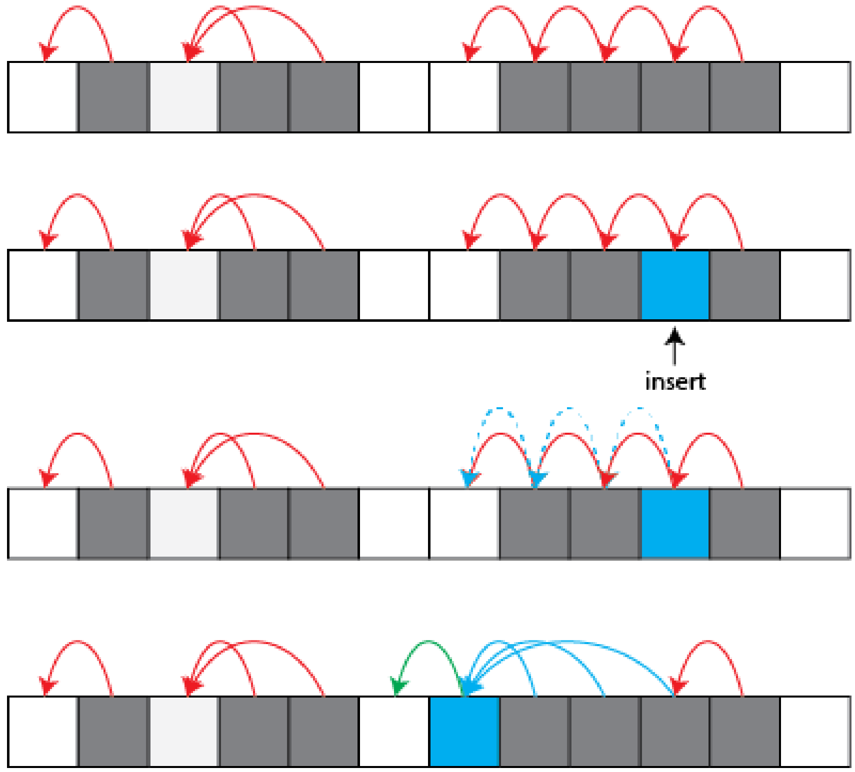
The problem also gives us a hint: **Disjoint-Set** (introduced in *Section 21.3*)

Let $next(x)$ denote the next empty position from position x.

- Initially, let $\forall i, next(i) = i$, i.e. the whole array is empty.



- If we want to insert a key into the position x, then we repeatedly find the position $x = next(x)$ until A_x is empty, i.e. $x == next(x)$, insert the key and let $next(x) = x - 1$. **OR** we cannot find an empty position, i.e. $x == -1$.



So, it is easy to see that this structure is exactly the same as disjoint-set. Therefore, we could use the ~~**union by rank** and~~ **path compression** techniques to improve the running time of each finding operation to $\cancel{\mathcal{O}(\alpha(n))}$ $\mathcal{O}(\log n)$, where n is the length of array ~~and α is the **inverse Ackermann function**~~.

Therefore, back to original problem, the running time of the algorithm is $\cancel{\mathcal{O}(n \cdot \alpha(n))}$ $\mathcal{O}(n \log n)$ using this data structure.

As you see, I made some mistakes on the analysis of this algorithm. But I believe that there exists a $\mathcal{O}(n \cdot \alpha(n))$ algorithm for this problem. I would append the better solution as long as I get it.

Reference

- [Book] Introduction to Algorithms (Third Edition) - *Cormen, Leiserson, Rivest & Stein*

7 Comments **Yinyanghu's Blog** 1 Login

Recommend Share Sort by Best

Join the discussion…

Roy • 9 months ago
hi, bro! i wonder what tools you use to edit the blog. i looks really great!
• Reply • Share ›

矢澤にこ • a year ago
How to prove the running time of each finding operation is O(logn)?
• Reply • Share ›

amadeupname • a year ago
YOU ARE WRONG AND LAME!
• Reply • Share ›

Anonymous • a year ago
There is a O(n) solution to this problem, but I can't find the data structure to achieve this complexity
• Reply • Share ›

yinyanghu Mod → Anonymous • a year ago
Probably. But I don't know currently.
• Reply • Share ›

Anonymous → yinyanghu • a year ago
To achieve O(n a(n)). I think you need to use path compression and union by rank
• Reply • Share ›

yinyanghu Mod → Anonymous • a year ago
Exactly! But I don't know how to do union by rank for this problem.
• Reply • Share ›

this website is available at Github.