ADw8

Name : **Peter Garamvoelgyi** Student ID: **2018280070**

CLRS 16.1-2. Suppose that instead of always selecting the first activity to finish, we instead select the last activity to start that is compatible with all previously selected activities. Describe how this approach is a greedy algorithm, and prove that it yields an optimal solution. (pg. 422)

First, let us notice that changes in the algorithm have no effect on the properties of *optimal substructure* and *overlapping subproblems*, i.e. these still hold.

It is easy to see that this algorithm is indeed a greedy one: In every step, we make a single greedy choice based on the starting time. To prove the correctness of this alternative greedy algorithm, we need to prove the *greedy-choice property*, i.e. **for any input, there is at least one global optimal solution that contains the greedy choice**.

Let $a_1, a_2, \dots, a_n$ denote the activities, sorted based on their starting time s_i.
Let A be a global optimal solution for an arbitrary input. There are two cases:

1. $a_n \in A$, i.e. A contains the greedy choice, the property is fulfilled. ✓
2. $a_n \notin A$, i.e. A does not contain the greedy choice.

   Let $a_k \in A$ denote the activity within A with the last start time, i.e.

   $$\forall\, a_i \in A, i \neq k\colon s_i < s_k$$

   As A is a valid solution, the following also holds (no overlaps):

   $$\forall\, a_i \in A, i \neq k\colon f_i \leq s_k$$

   Let us construct a new solution by replacing a_k with the greedy choice a_n

   $$A' = (A - \{a_k\}) \cup \{a_n\}$$

   – A' is optimal:

   $$|A'| = |A| - 1 + 1 = |A|$$

   – A' is valid:

   $$\left.\begin{array}{r}\forall\, a_i \in A, i \neq k\colon f_i \leq s_k \\ s_k < s_n\end{array}\right\} \implies \forall\, a_i \in A', i \neq n\colon f_i < s_n$$

   A' is a global optimal solution containing the greedy choice. ✓

We have proven the greedy-choice property holds in all cases.

CLRS 16-2. Scheduling to minimize average completion time. Give an algorithm that schedules the tasks so as to minimize the average completion time. Each task must run non-preemptively, that is, once task a_i starts, it must run continuously for p_i units of time. Prove that your algorithm minimizes the average completion time, and state the running time of your algorithm. (pg. 447)

For a scheduling

$$A_{1,n} = (i_1, i_2, \dots, i_n) \quad \sim \quad a_{i_1}, \dots, a_{i_n}$$

… where $(i_1, i_2, \dots, i_n)$ is some permutation of $\{1,2, \dots, n\}$, the completion times are:

$$c_{i_k} = \sum_{j=1}^{k} p_{i_j}$$

… and the overall cost is

$$\text{cost}(A_{1,n}) = \frac{1}{n}\sum_{l=1}^{n} c_l = \frac{1}{n}\sum_{k=1}^{n}\sum_{j=1}^{k} p_{i_j}$$

First, let us prove that this problem has the *optimal substructure property*:

For an optimal scheduling $A_{1,n} = (i_1, \dots, i_x, \dots, i_y, \dots, i_n)$

… the scheduling $A_{x,y} = (i_x, \dots, i_y)$ is also optimal for all $1 \le x < y \le n$.

The cost of any sub-scheduling can be expressed as

$$\text{cost}(A_{x,y}) = \frac{1}{y-x+1}\sum_{k=x}^{y}\sum_{j=x}^{k} p_{i_j}$$

… and the overall cost is

$$\text{cost}(A_{1,n}) = \frac{1}{n}\left(\sum_{k=1}^{x-1}\sum_{j=1}^{k} p_{i_j} + (y-x+1)\cdot \text{cost}(A_{x,y}) + \sum_{k=y+1}^{n}\sum_{j=y+1}^{k} p_{i_j}\right)$$

Suppose $A_{x,y}$ is sub-optimal, i.e. there is an alternative scheduling

$$A'_{x,y} = \left(i'_x, \dots, i'_y\right)$$

… such that

$$\text{cost}\left(A'_{x,y}\right) < \text{cost}\left(A_{x,y}\right)$$

We can use the *cut-and-paste* technique and use this sub-solution to construct a new global solution:

$$A_{1,n} = \left(i_1, \dots, i_x, \dots, i_y, \dots, i_n\right) \quad \rightarrow \quad A'_{1,n} = \left(i_1, \dots, \boldsymbol{i'_x}, \dots, \boldsymbol{i'_y}, \dots, i_n\right)$$

It is easy to see that we just use a new permutation for the indices affected; thus, the next solution is still valid. The cost becomes

$$\begin{aligned}
\text{cost}\left(A'_{1,n}\right) &= \frac{1}{n}\left(\sum_{k=1}^{x-1}\sum_{j=1}^{k} p_{i_j} + (y-x+1)\cdot\text{cost}\left(A'_{x,y}\right) + \sum_{k=y+1}^{n}\sum_{j=y+1}^{k} p_{i_j}\right) \\
&= \frac{1}{n}\left(n\cdot\text{cost}\left(A_{1,n}\right) + (y-x+1)\cdot\left(\text{cost}\left(A'_{x,y}\right) - \text{cost}\left(A_{x,y}\right)\right)\right) \\
&= \text{cost}\left(A_{1,n}\right) + \frac{y-x+1}{n}\left(\text{cost}\left(A'_{x,y}\right) - \text{cost}\left(A_{x,y}\right)\right)
\end{aligned}$$

We know that

$$\left.\begin{array}{c}\text{cost}\left(A'_{x,y}\right) < \text{cost}\left(A_{x,y}\right) \\ x < y\end{array}\right\} \quad \Rightarrow \quad \frac{y-x+1}{n}\left(\text{cost}\left(A'_{x,y}\right) - \text{cost}\left(A_{x,y}\right)\right) < 0$$

… thus

$$\text{cost}\left(A'_{1,n}\right) < \text{cost}\left(A_{1,n}\right)$$

… which contradicts with our original assumption that $A_{1,n}$ is optimal. This concludes the proof that the scheduling problem exhibits the optimal substructure property.

Let us define our candidate greedy choice as choosing the task with the shortest processing time from all unfinished tasks. To prove the correctness of this alternative greedy algorithm, we need to prove the *greedy-choice property*, i.e. **for any input, there is at least one global optimal solution that contains the greedy choice**.

Let $A_{1,n} = (i_1, i_2, \dots, i_n)$ denote a global optimal solution. There are two cases to consider:

1. The first task has the shortest execution time, i.e.

$$\forall\, w \in \{2, \dots, n\}\colon\ p_{i_w} \geq p_{i_1}$$

   This means that the solution contains the greedy choice. ✓

2. The first task is not the one with the shortest execution time, i.e.

$$\exists\, w \in \{2, \dots, n\}\colon\ p_{i_w} < p_{i_1}$$

   Let us construct a new solution by replacing i_1 with i_w:

$$A_{1,n} = (i_1, \dots, i_w, \dots, i_n) \quad \rightarrow \quad A'_{1,n} = (i_w, \dots, i_1, \dots, i_n)$$

   – $A'_{1,n}$ is valid.

     After swapping two tasks
     … $(i_w, \dots, i_1, \dots, i_n)$ is still a permutation of $\{1,2, \dots, n\}$.

   – $A'_{1,n}$ is optimal.

   Let us decompose $\text{cost}(A_{1,n})$ into four components:

$$\text{cost}(A_{1,n}) = \frac{1}{n}\sum_{k=1}^{n}\sum_{j=1}^{k} p_{i_j} = \frac{1}{n}\sum \begin{cases} p_{i_1} \\ \sum_{k=2}^{w-1}\left(p_{i_1} + \sum_{j=2}^{k} p_{i_j}\right) \\ p_{i_1} + \sum_{j=2}^{w-1} p_{i_j} + p_{i_w} \\ \sum_{k=w+1}^{n}\left(p_{i_1} + \sum_{j=2}^{w-1} p_{i_j} + p_{i_w} + \sum_{j=w+1}^{k} p_{i_j}\right) \end{cases}$$

Similarly, $\text{cost}(A'_{1,n})$ is

$$\text{cost}(A'_{1,n}) = \frac{1}{n}\sum \begin{cases} p_{i_w} \\ \sum_{k=2}^{w-1}\left(p_{i_w} + \sum_{j=2}^{k} p_{i_j}\right) \\ p_w + \sum_{j=2}^{w-1} p_{i_j} + p_{i_1} \\ \sum_{k=w+1}^{n}\left(p_{i_w} + \sum_{j=2}^{w-1} p_{i_j} + p_{i_1} + \sum_{j=w+1}^{k} p_{i_j}\right) \end{cases}$$

The relationship between the two is given by

$$\begin{aligned} \text{cost}(A'_{1,n}) &= \text{cost}(A_{1,n}) - p_{i_1} + p_{i_w} - \sum_{k=2}^{w-1}\left(p_{i_1} + \sum_{j=2}^{k} p_{i_j}\right) + \sum_{k=2}^{w-1}\left(p_{i_w} + \sum_{j=2}^{k} p_{i_j}\right) \\ &= \text{cost}(A_{1,n}) + (p_{i_w} - p_{i_1}) + \sum_{k=2}^{w-1}(p_{i_w} - p_{i_1}) \\ &= \text{cost}(A_{1,n}) + (w-1)(p_{i_w} - p_{i_1}) \end{aligned}$$

… i.e.

$$p_{i_w} < p_{i_1} \quad \Longrightarrow \quad \text{cost}(A'_{1,n}) < \text{cost}(A_{1,n})$$

… which is a contradiction, as $A_{1,n}$ is optimal. Thus, our assumption was false:

$$\nexists\, w \in \{2, \dots, n\}\colon\ p_{i_w} < p_{i_1}$$

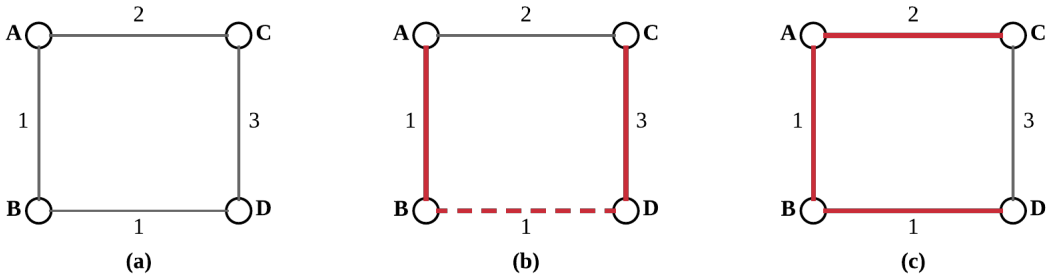
… i.e. p_{i_1} is the greedy choice. ✓

This concludes the proof that our algorithm fulfills the greedy-choice property. The algorithm is:

Input: $ps = \{p_1, p_2, \dots, p_n\}$
Output: A permutation of ps with minimum average completion time.
SCHEDULE(ps):
1 **return** SORT(ps)

CLRS 23.2-8. … Either argue that the algorithm correctly computes a minimum spanning tree of G, or provide an example for which the algorithm fails. (pg. 637).

We will show an example for which the proposed algorithm fails.



Let us look at the graph on figure (a). Let us say that the algorithm partitions G into

$$V_1 = \{A, B\} \quad V_2 = \{C, D\} \quad (|V_1| = |V_2|)$$

E_1 and E_2 contain the corresponding edges.

$$E_1 = \{e_{AB}\} \quad E_2 = \{e_{CD}\}$$

As $|E_1| = |E_2| = 1$, the algorithm can trivially solve the two subproblems. As $w(e_{BD}) < w(e_{AC})$, the algorithm will choose e_{BD} in to unite the two minimum spanning trees into a single spanning tree. This process in shown on figure (b). The total cost of the resulting spanning tree is

$$\text{cost}(T_{(b)}) = 1 + 1 + 3 = 5$$

However, this is not the minimum spanning tree for G; figure (c) shows a better one.

$$\text{cost}(T_{(c)}) = 1 + 1 + 2 = 4 < \text{cost}(T_{(b)})$$

If the subgraphs are connected, it is easy to see that the algorithm will indeed deliver a spanning tree. However, as the counter-example above shows, it is not necessarily the minimum spanning tree.

Corresponding to this example, here are two formulations of the optimal substructure property for MST; the first one is wrong, the second one is correct:

1. If $T = (V, E_T)$ is a MST in $G = (V, E)$, then any subset of vertices $V' \subset V$ together with the corresponding edges from E_T also forms an MST in $G' = (V', E)$. ✗
2. If $T = (V, E_T)$ is a MST in $G = (V, E)$, then by removing any edge from T, the resulting two smaller trees will be MSTs in their corresponding subgraphs. ✓