greedy and dp **May 15, 2025**

Homework 6

Spring 2025 *COS31: Algorithms*

1. **Activity Selection Revisted**
Not just any greedy approach to the activity-selection problem produces a maximum-size set of non-overlapping set of activities.
Note: In all the cases below, the most optimal MNOS is found by picking event with the earliest finish time.

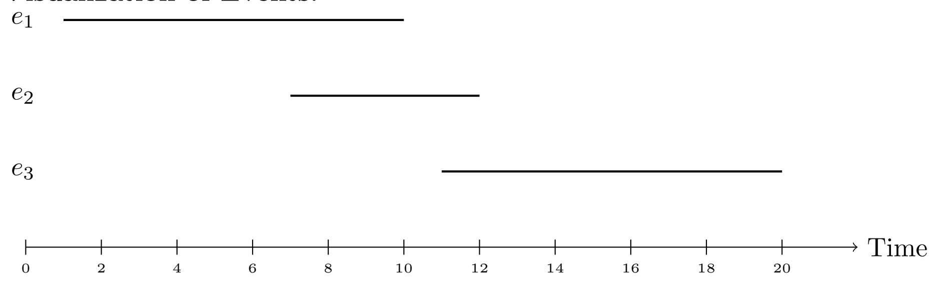
(a) Give a concrete counterexample to show that the approach of selecting the activity of least duration from among those that are compatible with previously selected activities does not work.

Answer

Suppose we have 3 events e_1, e_2, e_3, each with an id, start and finish time encoded as $e_i.id, e_i.s, e_i.f$ for $1 \leq i \leq 3$. The start and finish times are $e_1 = [1, 10], e_2 = [7, 12], e_3 = [11, 20]. e_2$ has the shortest duration and overlaps with the events e_1 and e_3. We also see that e_1 and e_3 don't overlap.

By choosing the event with the shortest duration as our greedy choice lemma, we see that $MNOS = \{e_2\}$, and $MNOS = |\{e_2\}| = 1$. However, we have most optimal MNOS as $|\{e_1, e_3\}| = 2$ since they are disjoint.

Visualization of Events:



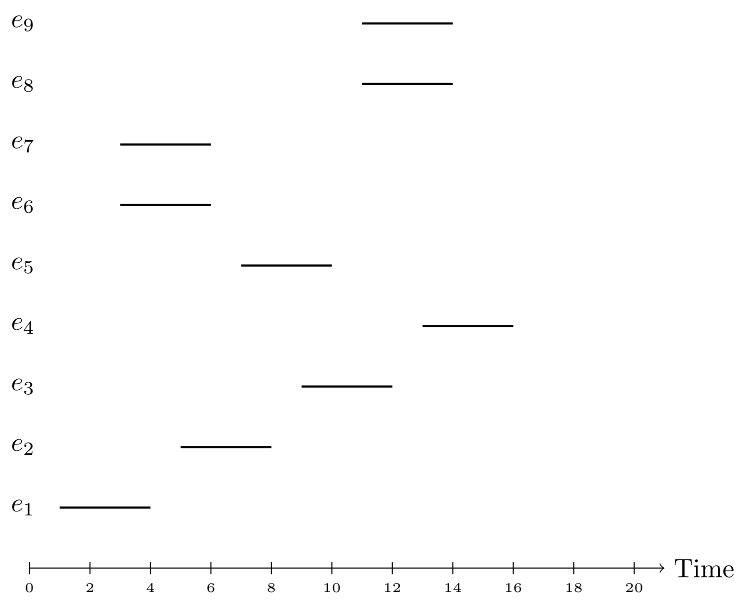
(b) Give a concrete counterexample to show that the approach of always selecting the compatible activity that overlaps the fewest other remaining activities does not work.

Answer

Suppose we have 4 events $e\{e_1, e_2, e_3, \ldots, e_9\}$, we have

- $e_1 = [1, 4]$
- $e_2 = [5, 8]$
- $e_3 = [9, 12]$
- $e_4 = [13, 16]$
- $e_5 = [7, 10]$

- $e_6 = [3, 6]$
- $e_7 = [3, 6]$
- $e_8 = [11, 14]$
- $e_9 = [11, 14]$



By the greedy lemma that selects the activity that overlaps the fewest other remaining activities, we might choose $e_5 = [7, 10]$ first. However, this blocks both $e_2 = [5, 8]$ and $e_3 = [9, 12]$. Then we can only pick $e_1 = [1, 4]$ before it, and $e_4 = [13, 16]$ after it, giving us $MNOS = |\{e_1, e_4, e_5\}| = 3$.
But the optimal solution is to pick $\{e_1 = [1, 4], e_2 = [5, 8], e_3 = [9, 12], e_4 = [13, 16]\}$, which are all compatible and yield $MNOS = 4$.
Thus, choosing the compatible activity that overlaps the fewest other activities does not always yield a maximum-size set.

(c) Give a concrete counterexample to show that the approach of always selecting the compatible remaining activity with the earliest start time does not work.

Answer

Suppose we have 3 events e_1, e_2, e_3 with start and finish times below:

- $e_1 = [1, 10]$
- $e_2 = [3, 5]$
- $e_3 = [6, 9]$

By greedy lemma, we first pick e_1, hence $MNOS = |\{e_1\}| = 1$ because e_1 overlaps with e_2 and e_3, but the optimal $MNOS = |\{e_2, e_3\}| = 2$ hence picking the earliest start time always doesn't yield the max-size set.

2. **Lodging**
Suppose we have $n \geq 1$ rooms $r[1 \ldots n]$, and $m \geq 1$ groups $g[1 \ldots m]$. Each room $r[i]$ has a distinct ID

$r[i].id$ and a capacity $r[i].cap$, which is the maximum number of people that the room accommodates. Each group g[i] has a distinct ID $g[i].id$ and a size $g[i].size$, which is the number of people in that group. We want to assign groups to rooms so that each group is assigned to at most one room, at most one group is assigned to each room, and if a group $g[i]$ is assigned to room $r[j]$, then $g[i].size \leq r[j].cap$. Thus, an assignment A is a set of pairs (g, r), where g is a group, r is a room, and the following conditions are satisfied: (i) If (g, r) and (g', r) are distinct pairs in A, then $g.id \neq g'.id$ and $r.id \neq r'.id$; and (ii) If $(g, r) \in A$, then $g.size \leq r.cap$.

Notice that if there are fewer rooms than groups or if rooms are not big enough, not all groups can be assigned rooms. Similarly, if there are too many rooms, some rooms might go empty (i.e., no groups are assigned to these rooms).

The merit of an assignment is the sum of the sizes of the groups that are assigned rooms. Give a greedy algorithm that prints an assignment of maximum merit. Be sure to rigorously state and prove the Greedy Choice Lemma that your algorithm is based on. State the time and space complexity of your algorithm

Answer

Greedy choice lemma: Given a set of rooms $R = \{r_1, r_2, \ldots, r_n\}$ and a set of groups $G = \{g_1, g_2, \ldots, g_m\}$, let $g^* \in G$ be the largest unassigned group (i.e., with the maximum size), and let $r^* \in R$ be the largest available room such that $g^*.size \leq r^*.cap$. Then there exists an optimal assignment in which the pair (g^*, r^*) is included, and for some room $r' \leq r^*$, we can find an optimal assignment (g', r') where $g' \leq g^*$.

Proof. Let S be the largest set with the maximum merit. Consider the following cases:

(a) If $(g^*, r^*) \notin S$, then we can define $S' = S \cup \{(g^*, r^*)\}$, yielding $|S'| > |S|$. This contradicts that S is the largest set with the maximum possible merit, and S' os the largest possible set with maximum merit.

(b) Suppose $(g', r^*) \in S$ and $(g^*, r^*) \notin S$. Since $g^*.size \leq r^*.cap$ by assumption, we can define

$$S' = (S \setminus \{(g', r^*)\}) \cup \{(g^*, r^*)\}$$

where $|S'| = |S|$. Therefore, S' is also a largest set with maximum merit.

(c) Suppose $(g^*, r') \in S$ and $(g^*, r^*) \notin S$. Since $r^* \leq r'$ in capacity, we can define

$$S' = (S \setminus \{(g^*, r')\}) \cup \{(g^*, r^*)\}$$

where $|S'| = |S|$. Therefore, S' is also a largest set with maximum merit.

(d) Suppose we have the pairs (g', r^*) and (g^*, r') in S. Then we can create the pairs (g^*, r^*) and (g', r'), and define

$$S' = (S \setminus \{(g', r^*), (g^*, r')\}) \cup \{(g^*, r^*), (g', r')\}$$

so that $|S'| = |S|$. Thus, S' is also a largest set with maximum merit.

□

```
Algorithm LodgingProblem
Function MaximumMerit ($r[1 \dots n], g[1 \dots m]$):
    Sort $r[1 \dots n]$ descending by $r[i].cap$
    Sort $g[1 \dots m]$ descending by $g[i].size$
    $ptr \leftarrow 1$                                  // Pointer to next unassigned group
    maxMerit ← 0

    for $i \leftarrow 1$ to $n$ do
        while $ptr \leq m$ and $(g[ptr].size > r[i].cap)$ do
            ptr++
        if $ptr \leq m$ then
            maxMerit ← maxMerit g[ptr].size
            ptr++
    return maxMerit
```

Time complexity: $O(nlogn + mlogm)$
Space complexity: $O(n + m)$

3. **Party Planning**
Professor Stewart is consulting for the president of a corporation that is planning a company party. The company has a hierarchical structure; that is, the supervisor relation forms a tree rooted at the president. The personnel office has ranked each employee with a conviviality rating, which is a real number. In order to make the party fun for all attendees, the president does not want both an employee and his or her immediate supervisor to attend. Professor Stewart is given the tree that describes the structure of the corporation, using the left-child, right-sibling representation (read my notes on the next page to understand/recall this representation and how to traverse a tree represented in this manner). Each node of the tree holds, in addition to the pointers, the name of an employee and that employee's conviviality ranking (in your algorithm, if you need to introduce a constant number of more fields in a node, you can assume that they have always been there for you to use them). Give a dynamic programming algorithm that prints out a guest list that maximizes the sum of the conviviality ratings of the guests. Your solution must include all of the steps of dynamic programming emphasized in the lecture.

Answer

Clever observation: For any parent node, we either include it in the guest list and exclude all its children, or exclude it and choose the best possible combination of its children. We maintain two lists: one if the president is included, and one if the president is excluded. We recursively compute the best option for each node.
Notation: Let x be a node in the tree. Define:

- $Include(x)$: Maximum conviviality of subtree rooted at x if x is invited.
- $Exclude(x)$: Maximum conviviality of subtree rooted at x if x is not invited.

Recurrence:

$$Include(x) = x.\texttt{val} + \sum_{\text{child } y \text{ of } x} Exclude(y)$$

$$Exclude(x) = \sum_{\text{child } y \text{ of } x} \max(Include(y), Exclude(y))$$

$$\text{MaxConviviality}(x) = \begin{cases} 0 & \text{if } x = NULL \\ x.\texttt{val} + \sum_{\text{child } y} \texttt{Exclude}(y), & \text{if } x \text{ is invited} \\ \sum_{\text{child } y} \max(\texttt{Include}(y), \texttt{Exclude}(y)), & \text{if } x \text{ is not invited} \end{cases}$$

$$\text{MaxConviviality}(x) = max(Include(x), Exclude(x))$$

$$\text{guest list} = \begin{cases} include\_root\_list & \text{if } Include(x) > Exclude(x) \\ exclude\_root\_list & \text{otherwise} \end{cases}$$

Algorithm Compute Max Conviviality and Guest List

```
Function FindGuests (root):
    $(inc, inc\_list, exc, exc\_list) \leftarrow$ ComputeConviviality(root)

    if $inc > exc$ then
        print the $inc\_list$
    else
        print the $exc\_list$

Function ComputeConviviality($x$: node):
    if $x = NULL$ then
        return $(0, [\,], 0, [\,])$                         // (max_value, guest_list)

    $include\_parent \leftarrow x.\texttt{val}$
    $include\_list \leftarrow [x.\texttt{id}]$
    $exclude\_parent \leftarrow 0$
    $exclude\_list \leftarrow [\,]$

    $child \leftarrow x.\texttt{lmc}$
    while $child \neq NULL$ do
        $(child\_inc,\ child\_inc\_list,\ child\_exc,\ child\_exc\_list) \leftarrow$ ComputeConviviality($child$)
        $include\_parent\ += child\_exc$
        $include\_list\ += child\_exc\_list$
        if $child\_inc > child\_exc$ then
            $exclude\_parent\ += child\_inc$
            $exclude\_list\ += child\_inc\_list$
        else
            $exclude\_parent\ += child\_exc$
            $exclude\_list\ += child\_exc\_list$
        $child \leftarrow child.\texttt{rs}$
    return $(include\_parent, include\_list, exclude\_parent, exclude\_list)$
```

Time complexity: $O(n)$ where n is the number of nodes in the tree.
Space complexity: $O(n)$ for recursion stack space.