**Ruolin Li**
**31764160**

# CPSC 320 2020S2: Tutorial 5

## 1 Safety First

UBC Computer Science is developing a plan for its $n$ employees to return to campus for a limited shift each day, while respecting social distancing. There are $n$ employees, numbered 1 to $n$ for convenience.

A daily schedule has been developed, with employee $i$ working for the same shift $[s_i, e_i]$ each day, where $s_i$ is the start time and $e_i$ is the end time of the shift. Assume that no shift starts earlier than 06:00am or ends later than 10:00pm. Assume also, for convenience, that employers are numbered in nondecreasing order of shift end time. That is, for all pairs $i, j$ of employees, if $i < j$ then $e_i \leq e_j$.

**Example:** Here $n = 4$ and the shifts are listed in nondecreasing order of end time. (The list would also be in nondecreasing order of end time if employees 2 and 3 were switched.)

```
Employee 1: 08:00 to 10:00
Employee 2: 06:00 to 12:00
Employee 3: 10:00 to 12:00
Employee 4: 12:00 to 13:00
```

The Department Head is required to form a safety committee $C \subseteq [1..n]$, which will monitor social distancing guidelines. For every employee $i \notin C$ (that is, $i$ is *not* on the committee), $i$'s shift must overlap at least partially with the shift of some committee member $j \in C$, in which case we say that $i$ is *covered* by $j$. Shifts overlap even if they just "meet" at their start or end times; for example, the shifts of employees 3 and 4 above overlap.

The following algorithm aims to find a safety committee that has as few people on it as possible, while ensuring that all employees are covered. The algorithm takes as input an ordered list $L$ of the shifts $[s_i, e_i]$ of the employers.

**procedure** CREATE-COMMITTEE($L$) ▷ $L$ is a list of $n$ shifts $[s_i, e_i], 1 \leq i \leq n$
  $C \leftarrow \emptyset$; $E \leftarrow \{1, 2, ..., n\}$
  **while** $E$ is not empty **do**
    let $i$ be the employee in $E$ with the earliest starting time
    let $S$ be the set of employees whose shifts overlap that of employee $i$
    let $j$ be the employee in $S$ with the latest ending time (break ties arbitrarily)
    add $j$ to $C$
    remove from $E$ all employees covered by $j$ (including both $i$ and $j$)
  **return** $C$

Show that this algorithm is not correct.

## 2 Knapsack with Structured Weights: Part 2

**Let's continue the problem we saw in Tutorial 4:**

The *Knapsack Problem* is a classic optimization problem in computer science. The most common version is as follows: consider a set of $n$ items, numbered $1, ..., n$. Each item $i$ has a positive weight $w_i > 0$ and a positive value $v_i > 0$. Furthermore, we're given an overall weight capacity $C$. The problem is to select a subset $S \subseteq \{1, ..., n\}$ that maximizes the total value of $S$, but keeps the total weight below $C$. Formally, we want to maximize $\sum_{i \in S} v_i$ subject to $\sum_{i \in S} w_i \leq C$.

The knapsack problem is known to be hard: there are no algorithms known that find the optimal solution to all instances in better than worst-case exponential time. However, for this tutorial question, we'll consider a variation of the problem. In particular, we'll set the capacity as $C = 1$, and restrict that all weights are either 1/2, 1/4, 1/8, or 1/16.

**Recall the two lemmas that we proved last time:**

- There is always an optimal solution that uses an even number of 1/16-weight items.
- If an optimal solution uses $c$ items of weight 1/16, it uses $c$ of the highest value 1/16-weight objects.

With the preceding two lemmas, we start to see a hint of how our algorithm will work. Suppose we sort the 1/16-weight items in decreasing order of value, and then pair up the items in sequence (i.e. the 1st and 2nd items form a pair, then the 3rd and 4th items, then the 5th and 6th, etc.).

1. Prove that there is an optimal solution that respects these pairs (i.e., for each pair, it either uses both items or neither of them.).

Now, we reach the key idea of our greedy algorithm. Given an instance of the problem, create a new instance as follows: Sort the 1/16-weight items in decreasing order of value, and pair them up (as in the previous question). Now, for each pair (call them items $i$ and $j$), delete the items from the instance and replace them with a new item whose weight is $w_i + w_j$ and whose value is $v_i + v_j$.

This creates a new instance of the problem, except there are no 1/16-weight items anymore. We want to prove that an optimal solution to the new problem instance (without 1/16-weight items) corresponds to an optimal solution to the original problem instance. This is an "if and only if"-type theorem, so we will prove this in two parts.

2. Prove that for any optimal solution to the original instance, you can get an equally good solution to the new instance.
3. Prove that for any optimal solution to the new instance, there's an equally good solution to the original instance.

Given an original problem instance, we've seen how to eliminate the 1/16-weight items to create an equivalent problem without 1/16-weight instances. But once we have that new problem instance, the exact same reasoning allows us to create an equivalent problem instance without 1/8-weight items! Once the 1/8-weight items are gone, we can eliminate the 1/4-weight items. Then, we solve the problem simply by picking the top two 1/2-weight items.

This is the efficient greedy algorithm for this problem, and we've already proven that it is optimal using an exchange arguments approach!

2020/07/22.
Ruolin Li 31764160

Tutorial 5

1. Step 1: $C=\{\}$ $E=\{1,2,3,\dots,n\}$.
$i=2$
$S=\{1,2,3,4\}$
$j=4$
$C=\{4\}$.
$E=\{1\}$.

The optimal solution: $\{3\}$. $\{2\}$. they overlap with all the time slots. But the result is $\{1,4\}$ which is not correct!

Step2: $C=\{4\}$ $E=\{1\}$.
$i=1$
$S=\{1\}$. overlap with itself
$j=1$
$C=\{1,4\}$.
$E=\{\}$.

A counterexample is enough.

2. ① $w_1$ | $v_1$
⋮
⑪ $w_n$ | $v_n$

$\sum w_i \le C$. While Maximize $\sum v_i$

1) weight value
1/16 1
1/16 2
1/16 3
1/16 4
1/16 5
1/16 6

There's always an optimal solution using an even number of 1/16-weight items, and these items are the first c items in the sorted list.

Addition is commutative and associative.
c is even, optimal solution will use the first c/2 pairs
So we don't need to split the pairs.

2) $\text{weight}_{new}$ $\text{value}_{new}$
1/8 3
1/8 7
1/8 11

no 1/16 items.
replace the original item with new value and weight
total weight and value remain the same when we add them up and replace the original one.
So we have a new solution has the same $\sum v_i$, $\sum w_i$
The optimal solution to the new instance cannot be worse than the original one.

3) Suppose we have the optimal solution to the new instance
→ 1/8 - weight items are derived from pairs of 1/16 weights in the original problem
→ we can decompose any solution to the new instance into the 1/16 items.