#1 Assignment - CMPT 405

Luiz Fernando Peres de Oliveira - 301288301 - lperesde@sfu.ca

September 16, 2018

#1 - Let C be the array containing all the possible coins {*1, 5, 10, 25, 100, 200* }. Let V be the total change value.

Algorithm:

Input: *C, V*
$d \leftarrow$ sort C such that $d_1 \geq d_2 \geq ... \geq d_n$
$res \leftarrow \emptyset$; $i \leftarrow 1$
while $V > 0$ **do**
  if $V \geq d_i$ **then**
    $n_{coins} \leftarrow \left\lfloor \frac{V}{d_i} \right\rfloor$
    $V \leftarrow V - (n_{coins} * d_i)$
    $res \leftarrow res \cup \{(d_i, n_{coins})\}$
  end if
  $i \leftarrow i + 1$
end while
return res

Intuition: $\forall (d_i, d_j) \in C$, $1 \leq i < j \leq n$, $d_i \geq 2 * d_j$, meaning that if I the algorithm chooses any d_j over any d_i, it will have to pick at least 2 times more coins for some value V that satisfies both d_i and d_j.

Proof. The only way to give more coins than the smallest possible number of coins for any change would be in a case where the algorithm chooses d_j over a d_i (see *Intuition*). Now, imagine that the algorithm chooses d_j over d_i, then we have two cases:
- Case 1. $d_i > V$:
If so, we are done because there are no possible ways of choosing d_i for value V.
- Case 2. $d_i \leq V$:
If that was the case, we would have an optimal set OPT such that $OPT_{i-1} \cup d_j \subseteq OPT$, which is not the case, once the iteration i will happen before the iteration j, causing the algorithm to choose d_i over d_j (and never the opposite) for any value V that satisfies both d_i and d_j. $\square$

#2 a)

Greedy approach to the fractional knapsack:
- n objects and a knapsack
- item i weighs $w_i > 0$ and has utility $u_i > 0$
- fill knapsack so as to **maximize** total utility/weight, not exceeding total capacity W

Algorithm approach:
- sort items in decreasing order of their utility-to-weight ratio u_i/w_i
- repeatedly add item with max ratio u_i/w_i. If not possible to add the whole object, add a fraction $\alpha \in (0, 1)$ of it, if possible.

Proof. Let K_{opt} be the optimal set of items in a knapsack and let K_j be the chosen items after an iteration j, $0 \leq j \leq n$.

Base case: K_0: K_0 is promising since the total number of chosen objects, in this case *none*, does not exceed total capacity W, there exists some optimal K_{opt} such that $K_0 \subseteq K_{opt} \subseteq K_0 \cup \{i_1, i_2, ..., i_n\}$.

Induction step: Assume K_{j-1}. Since K_{j-1} is promising for stage $j-1$, $K_{j-1} \subseteq K_{opt} \subseteq K_{j-1} \cup \{i_j, i_{j+1}, ..., i_n\}$. We want to show K_j. On a stage j we have two cases:

Case 1. i_j is rejected. Then $K_{j-1} \cup \{i_j\}$ or $K_{j-1} \cup \{i_j * \alpha\}$ (any fraction $\alpha \in (0, 1)$ of i_j) exceed the capacity W; thus, $K_{j-1} = K_j$. Since $K_{j-1} \subseteq K_{opt}$ and K_{opt} does not exceed the total capacity W, $i_j \notin K_{opt}$. So $K_j \subseteq K_{opt} \subseteq K_j \cup \{i_{j+1}, i_{j+2}, ..., i_n\}$.

Case 2. i_j or $i_j * \alpha$ is added to K_{j-1}. Let item i_{chosen} be i_j or $i_j * \alpha$ (whichever was added to K_{j-1}). Then $K_{j-1} \cup \{i_{chosen}\}$ does not exceed the total capacity W and we have $K_{j-1} \cup \{i_{chosen}\} = K_j$.

Case 2.1. $i_{chosen} \in K_{opt}$. Then we have $K_j \subseteq K_{opt} \subseteq K_j \cup \{i_{j+1}, i_{j+2}, ..., i_n\}$.

Case 2.2. $i_{chosen} \notin K_{opt}$. We show that there is another maximum set of utility-to-weight items K'_{opt} that witnesses the fact that K_j is promising. For example, consider an item i_{chosen} added to K_{opt}. This will exceed the capacity W and the knapsack will contain at least one item of $\{i_{j+1}, i_{j+2}, ..., i_n\}$.

Proof of claim: K_{opt} contains all elements of K_{j-1} and can be obtained from K_{j-1} by adding some items from the set $\{i_j, i_{j+1}, ..., i_n\}$. Adding i_j does not exceed capacity W, so the excess in $K_{opt} \cup \{i_{chosen}\}$ must contain some elements other items in $\{i_{j+1}, i_{j+2}, ..., i_n\}$.

□

#2 b)

item	utility	weight
1	2	1
2	1000	1000

$W = 1000$

$K_{opt} = \{i_2\}$(utility = 1000), $K_{greedy} = \{i_1\}$(utility = 2)

#3