Assignment 4

Algorithms: Correctness and Complexity

2014 Solution

Q0 [20] MARS

You are on the operations team of the CSA's Mars Ascend and Return Samples mission. The rover has identified hundreds of samples (set S) that could be returned. Each sample $x \in S$ has a real weight $w(x)$ and a real value $v(x)$. A subset of S is optimal if it maximizes the total value while the total weight is under or equal to $w_{\max}$.

(a) [10] Give a greedy algorithm for this problem. Your algorithm need not find an optimal subset, but it should do a reasonable job at selecting a reasonable subset.

> **Solution:** There is always at least one, which is the empty set, so our invariant does not need to say "if there is a solution". Since we are not trying to make the solution optimal, the invariant need not (indeed should not) say "there is an optimal solution ... ". I decided to order the samples by density, where the density of sample x is $v(x)/w(x)$

```
var $C := \emptyset$ // $C$ collects the solution. At the end $C$ is the answer
var $r := w_{\max}$ // $r$ is the remaining weight.
inv: there is a solution that includes all items in $C$ and only items in $C \cup S$.
inv $r = w_{\max} - \sum_{x \in C} w(x)$
while $S \neq \emptyset$
    val $x \mid x$ is the (or a) densest item of $S$.
    if $w(x) \leq r$ then
        // Keep $x$
        $C := C \cup \{x\}$
        $r := r - w(x)$
    end if
end while
```

(b) [10] Give counter-examples showing that each of the three most obvious ways of ranking the samples can lead your algorithm to find suboptimal subsets.

Solution: I'll write each sample as v/w where v is the value and w is the weight

- Highest value first. Suppose we have a weight limit of 6. The samples in order of value are 5/5, 4/3, 4/3. Picking the most valuable first prevents the optimal solution of value 8.
- Lowest weight first. Suppose we have 1/1, 4/2, 4/2 and a weight limit of 3, lowest weight first gives 1/1 and 4/2 with value 5, but the best solution has value 8.
- Highest density first. Again density is value over weight. Suppose we have 5/3, 3/2, 3/2 and a weight limit of 4. The highest density first strategy picks 5/3 only, whereas a value of 6 can be obtained.

Q1 [10] You work for ZipTrip.com, the travel site for people in a hurry. They guarantee to find the quickest sequence of flights, from location x to location y, such that the first flight leaves on or after time t.

Their database consists of a set of flights, where each flight is associated with a starting airport, a destination airport, a boarding time, and a landing time.[1] Your algorithm needs to find the set of flights that gets a passenger from airport x at time t to airport y the soonest. Layovers where the passenger must change plane must be at least 30 minutes between landing and boarding time, 60 minutes if the passenger must change terminals, or 120 minutes if the passenger must pass through immigration and customs.. Find an efficient algorithm to solve this problem. Keep in mind that ZipTrip.com will be processing many such queries on the same data base, so some preprocessing of the data base is reasonable to do.

Solution: I would approach this using Dijkstra's algorithm. In order to do that, we make a graph in which each node represents an arrival or a departure of a flight. Let's say for each flight f, $d(f)$ is its departure node and $a(f)$ is its arrival node. Then put in edges so that for each flight f there is an edge from $d(f)$ to $a(f)$. The weight of these edges is the length of the flight. Say that f connects with g the airport of $a(f)$ is the same as the airport of $d(g)$ and the time of $d(g)$ is sufficiently long. Now we add an edge from $a(f)$ to $d(g)$ for each pair of connecting flights. The weight of such an edge is the time

[1]In reality a flight might have stops.. However we can treat such a flight as several simple flights for our purposes.

difference between the arrival and the departure. As new information comes in about flights,

Having built this graph, we can now process each customer query using Dijkstra's algorithm.

[In practice, we would likely use A*, which is an optimized version of Dijkstra's that uses heuristic information. However we did not cover A*, so if you used plain Dijkstra's that is fine.]

Q2 [20] As mentioned in class, the greedy algorithm for making change fails for certain mixes of denominations, for example $\{1, 5, 10, 25\}$. Develop a dynamic programming solution for the following problem. Input: a natural number t and a set S of coins, each having a natural number value. Output: A minimal sized subset of coins that have total value t.

(a) [**5**] Develop a recursive algorithm to determine whether there is or is not a subset of S that has total value t.

(b) [**5**] Develop an efficient top-down (memoizing) algorithm to solve the problem in part (a)

(c) [**5**] Develop an efficient bottom-up algorithm to solve the problem of part (a).

(d) [**5**] Modify your solution to either part (b) or (c) to output a minimal sized subset of coins that adds up to t, if there is one.

Solution:

(a) First we will put S into a sequence s without repetition, so that we can refer to subsets of it using a single integer: $s\{0, ..i\}$ is the subset of S consisting of the first i elements of s.

The question that the algorithm needs to answer is the size of the smallest subset of $s\,\{0, ..i\}$ that has total value of g. Let $w(x)$ be the value of coin x. Define

$$\begin{aligned} val(C) &= \sum_{x \in C} w(x) \\ mss(i, g) &= \min_{C \subseteq s\{0,..i\} \mid g = val(C)} |C| \end{aligned}$$

That is, $mss(i, g)$ is the size of a minimally sized subset of $s\,\{0, ..i\}$ whose total value is g, assuming there is one. We'll take the minimum of the empty set to be ∞, so if there is no subset of $s\,\{0, ..i\}$ whose total value is g, the value of $mss(i, g)$ is ∞.

```
proc $minimalSubsetSize(i, g) : \mathbb{N}$
    pre $0 \leq i \leq s.\text{length}$ and $0 \leq g \leq t$
    post $result = mss(s, i, g)$
    if $i = 0$ then
        if $g = 0$ then return 0 else return $\infty$ end if
    else
        var $best := \infty$
        val $x := s(i - 1)$
        // Consider keeping $x$
        if $w(x) \leq g$ then $best := 1 + best \min mss(i - 1, g - w(x))$ end if
        // Consider discarding $x$
        $best := best \min mss(i - 1, g)$
        return $best$
    end if
end $minimalSubsetSize$
```

Now the answer is given by $minimalSubsetSize(s.\text{length}, t)$.

(b) Converting the algorithm to part (a) is just a matter of introducing a cost table C that maps (i, g) pairs to either $mss(i, g)$ or nil.

```
var $C : \{0, .., s.\text{length}\} \times \{0, .., t\} \rightarrow \mathbb{N} \cup \text{nil}$
// data invariant $\forall i, g \cdot C(i, g) = \text{nil} \vee C(i, g) = mss(i, g)$
```

Now the algorithm is given by

```
proc $minimalSubsetSize(i, g) : \mathbb{N}$
    pre $0 \leq i \leq s.\text{length}$ and $0 \leq g \leq t$
    post $result = mss(s, i, g) = C(i, g)$
    if $C(i, g) = \text{nil} \wedge i = 0$ then
        if $g = 0$ then $C(i, g) := 0$ else $C(i, g) := \infty$ end if
    elsif $C(i, g) = \text{nil}$ then
        var $best := \infty$
        val $x := s(i - 1)$
        // Consider keeping $x$
        if $w(x) \leq g$ then $best := 1 + best \min mss(i - 1, g - w(x))$ end if
        // Consider discarding $x$
        $best := best \min mss(i - 1, g)$
        $C(i, g) := best$
    end if
    return $C(i, g)$
end $minimalSubsetSize$
```

(c) There are a number of orders we could fill the table in with. Here is one.

```
var $C : \{0, .., s.\text{length}\} \times \{0, .., t\} \to \mathbb{N}$
$C(0, 0) := 0$
for $g \leftarrow [1, .., t]$ do $C(0, g) := \infty$ end for
for $i \leftarrow [1, .., s.\text{length}]$ do
    for $g \leftarrow [0, .., t]$ do
        var $\mathit{best} := \infty$
        val $x := s(i - 1)$
        // Consider keeping $x$
        if $w(x) \le g$ then $\mathit{best} := 1 + \mathit{best} \min C(i - 1, g - w(x))$ end if
        // Consider discarding $x$
        $\mathit{best} := \mathit{best} \min C(i - 1, g)$
        $C(i, g) := \mathit{best}$
    end for
end for
```

At the end of this process the answer is in $C(s.\text{length}, t)$.

(d) Provided $C(s.\text{length}, t) \neq \infty$ we can work our way back through the table to determine the set.

```
output var $F : \mathit{Set}\langle \mathit{Coin} \rangle := \emptyset$
var $g := t$
var $i := s.\text{length}$
while $g \neq 0$ do
    if $C(i, g) = C(i - 1, g)$ then
        // Discard $C(i)$
        $i := i - 1$
    else
        $F := F \cup \{s(i - 1)\}$
        $i := i - 1$
        $g := g - w(s(i - 1))$
    end if
end while
```