Example of how to use Algorithm2e

Robert Woodward

August 8, 2017

Below we illustrate the formatting as pseudo code of some sample of simple algorithms. The goal is not to entice you to use LaTeX for formatting your algorithms as currently the best possible formatting tool for algorithms. Please carefully check the source files and learn how to use this style. Importantly:

- Always state your input
- State the output if any
- Always number your lines for quick referral.
- Always declare and initialize your local variables
- Always use \gets for assignments
- Always end with "return" even when not returning any values
- Use common functions and operands such as Union, PowerSet, etc. as often as needed, unless you are asked to define them.

Algorithm 1 will find the maximum element in a finite sequence (Slide 14 in Class Slides).

Algorithm 1: Training Algorithms

Input: $x_s = \{s_1, s_2, y\}_{i=1}^{n}$ from source domain,
$x_t = \{s_1, s_2, y\}_{i=1}^{m}$ from target domain, $n \gg m$

Output: $f_t(s_1, s_2) \mapsto y$

1 Train $f_s(s_1, s_2) \mapsto y$
2 **for** *each batch* **do**
3 x'_t: sample from x_t
4 x'_s: sample k batches from x_s
5 Use $x'_s \cup x'_t$ to finetune model $f_s(s_1, s_2) \mapsto y$ on domain t

Algorithm 2 is a greedy change-making algorithm (Slide 19 in Class Slides).

Algorithm 3 and Algorithm 4 will find the first duplicate element in a sequence of integers.

Algorithm 2: CHANGE Makes change using the smallest number of coins

Input: A set $C = \{c_1, c_2, \ldots, c_r\}$ of denominations of coins, where $c_i > c_2 > \ldots > c_r$ and a positive number n

Output: A list of coins $d_1, d_2, \ldots, d_k$, such that $\sum_{i=1}^{k} d_i = n$ and k is minimized

```
1 C ← ∅
2 for i ← 1 to r do
3 |   while n ≥ c_i do
4 |   |   C ← C ∪ {c_i}
5 |   |   n ← n − c_i
6 return C
```

Algorithm 3: FINDDUPLICATE

Input: A sequence of integers $\langle a_1, a_2, \ldots, a_n \rangle$

Output: The index of first location witht he same value as in a previous location in the sequence

```
1  location ← 0
2  i ← 2
3  while i ≤ n and location = 0 do
4  |   j ← 1
5  |   while j < i and location = 0 do
6  |   |   if a_i = a_j then
7  |   |   |   location ← i
8  |   |   else
9  |   |   |   j ← j + 1
10 |   i ← i + 1
11 return location
```

Algorithm 4: FINDDUPLICATE2

Input: A sequence of integers $\langle a_1, a_2, \ldots, a_n \rangle$

Output: The index of first location witht he same value as in a previous location in the sequence

```
1 location ← 0
2 i ← 2
3 while i ≤ n ∧ location = 0 do
4 |   j ← 1
5 |   while j < i ∧ location = 0 do
6 |   |   if a_i = a_j then location ← i
7 |   |   else j ← j + 1
8 |   i ← i + 1
9 return location
```