I111E Algorithms & Data Structures

Answer to the second report

School of Information Science
Ryuhei Uehara & Giovanni Viglietta
uehara@jaist.ac.jp & johnny@jaist.ac.jp

2019-11-28

All materials are available at
http://www.jaist.ac.jp/~uehara/couse/2019/i111e

Problem 1 When we compare two strings, their ordering is defined as follows:

$$\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, \ldots,$$

where ϵ represents the empty string of length 0. This is not the same as the "usual" ordering in your English dictionary. Define the "length-preferred" lexicographical ordering and the "usual" lexicographical ordering. Why might we want to use this length-preferred ordering rather than the usual one?

Let $x=x_0x_1\cdots x_{n-1}$ and $y=y_0y_1\cdots y_{m-1}$ be two strings to be compared. We first observe that x=y if and only if n=m and $x_i=y_i$ for all i=0,1,⋯, n-1. We here define $\varepsilon<0<1$ for the sake of notational convention.

Definition of "length-preferred" lex. ordering

1. When $n \neq m$, $x<y$ if $n<m$ or $x>y$ if $n>m$.
2. When n=m, x<y if and only if $x_i<y_i$ and $x_{i'}=y_{i'}$ for *some* $0 \leqq i<n$ and *all* $0 \leqq i'<i$.

Problem 1 When we compare two strings, their ordering is defined as follows:

$$\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, \ldots,$$

where ϵ represents the empty string of length 0. This is not the same as the "usual" ordering in your English dictionary. Define the "length-preferred" lexicographical ordering and the "usual" lexicographical ordering. Why might we want to use this length-preferred ordering rather than the usual one?

Definition of "usual" lex. ordering

1. In any case, $x<y$ if and only if $x_i<y_i$ and $x_{i'}=y_{i'}$ for *some* $0 \leqq i<n$ and *all* $0 \leqq i'<i$.

Why we use "length-preferred" in computer?

Enumerate all strings in "usual" lex. ordering:

ε, 0, 00, 000, 0000, 00000, 000000, 0000000, …

"1" has no finite index!! How inconvenient!!

Problem 2 In quick sort, there are cases where a bad choice of a pivot makes the algorithm run slower. Give concrete examples of arrays and poorly chosen pivots that make quick sort have the worst possible running time.

- A pivot does not work if it divides into two unbalanced arrays.
- Example:
  - Pivot is “the first element in the array”
  - Input is “an array in order”;
    - E.g.

1	2	3	5	8	13	21	34	55

In this case, quick sort runs in $O(n^2)$ time⋯.

Problem 3 Let us consider the following shuffle problem, which is the *reverse* of sorting:

Input: An array $a[0], \ldots, a[n-1]$.

Output: The array $a[0], \ldots, a[n-1]$, where the items are randomly shuffled.

That is, we want an algorithm that randomly re-orders an array of n items in such a way that each possible ordering appears with uniform probability. Assume that we can use a function `random`(k) that returns any integer i with $0 \leq i < k$ with probability $1/k$. Then give an *efficient* algorithm to solve the shuffle problem.

- Naïve algorithm:

```
for i=0,1,2,···,n-1 do
   r= random(n-i);
   output the r-th "not yet output items" in a[];
   mark the output item in step 3 by "output";
end.
```

- How do we mark? → use an extra array
- How can we find the r-th item in a[]? → use O(n) time.

In total, the running time is $O(n^2)$

Problem 3 Let us consider the following shuffle problem, which is the *reverse* of sorting:

Input: An array $a[0], \ldots, a[n-1]$.

Output: The array $a[0], \ldots, a[n-1]$, where the items are randomly shuffled.

That is, we want an algorithm that randomly re-orders an array of n items in such a way that each possible ordering appears with uniform probability. Assume that we can use a function `random`(k) that returns any integer i with $0 \leq i < k$ with probability $1/k$. Then give an *efficient* algorithm to solve the shuffle problem.

- Smart algorithm (known as Fisher-Yates algorithm):

```
for i=0,1,2,···,n-1 do
  r= random(n-i);
  output a[r];
  a[r] = a[n-i-1];
end.
```

We used similar idea in
Bubble sort,
Heap sort,

- We always keep “not yet output items” in a[0]···a[n-i-1]
- We break the array, but it is quite simple and linear time algorithm!