COMP20007 Workshop Week 11

LAB	**Preparation:** - *have draft papers and pen ready* - open `a2_spec.pdf, wokshop11.pdf` (from `LMS`), and - ready to work with assignment 2 Counting & Radix sort: Problems T1, T2, T3 Horspool's Algorithm: Problems T4, T5, T6 Assignment 2 Revision on demands: assignment 2

Counting Sort

Simple Distribution Sort = Counting Sort

Conditions:

- keys are integers in a small range (small in comparison with n), for example:
  - array of 1000 positive integers, each ≤ 200
  - array of 1000 integers between -100 and +200 inclusively
- Note 1: normally a key is just a part of a data record
- Note 2: The sorted version of the array `{0,1,2,0,0,1,2,1,1,0,0,0}` is

```
{ 0,0,0,0,0,0,    1,1,1,1,    2, 2 }
```

starts from index 0

starts from index 6 = num of items <1

starts from index 10 = num of items <2

Counting Sort for sorting array A[0..n-1]

- Build array `C[0..k]` such that `C[i]`= number of keys that < `i`, by:
  - First, make `C[i]`← frequency of key **`i-1`** (for `i ∈ 1..k`)
  - Then: `for i ← 1 to k do  C[i] ← C[i-1] + C[i]`
- Note: `C[i]=` starting index of key value `i` in the *sorted* array
- Using another array `B[0..n-1]`, scan `A[]` again and copy to `B` using:

```
j= A[i];                  // here C syntax used for convenience
B[  C[j]++ ] = k;         // don't do that in your pseudocode
```

Notes:

- Time complexity: $\Theta(n+k)$, or $\Theta(n)$ if k small
- Not inplace, additional memory: $\Theta(n+k)$, or $\Theta(n)$ if k small

Radix Sort

Applied when all keys can be represented as same-size strings over a small alphabet A. Examples:

{1, 12, 7, 10, 6, 9, 8, 3}

→ {0001, 1100, 0111, 1010, 0110, 1001, 1000, 0011} A= {0,1}

{1,22,17,167,26,19,28,173,…}

→ {001, 022, 017, 167, 026, 019, 028, 173,…} A= {0,1,…,9}

→ {01, 16, 11, A7, 1A, 13, 1C, AD…} A= {0,1,…,9,A,B,..F}

Radix Sort:

From rightmost to leftmost symbols of strings:

- Put items into buckets defined by the value of that symbol (counting sort)
- Concatenate (join) buckets in increasing order of the sorted symbols

Complexity: n |A|

Problem 1 - Counting Sort: Use counting sort to sort the following array of characters:

[a, b, a, a, c, d, a, a, f, c, b]

How much space is required if the array has n characters and our alphabet has k possible letters.

Problem 2 - Radix Sort: Use radix sort to sort the following strings:

`abc bab cba ccc bbb aac abb bac bcc cab aba`

As a reminder radix sort works on strings of length k by doing k passes of some other (stable) sorting algorithm, each pass sorting by the next most significant element in the string. For example in this case you would first sort by the 3rd character, then the 2nd character and then the 1st character.

Problem 3: Which property is required to use counting sort to sort an array of tuples by only the first element, leaving the original order for tuples with the same first element. For example the input may be:

(8, campbell), (6, tal), (3, keir), . . . (6, gus), (0, nick), (8, tom)

Discuss how you would ensure that counting sort satisfies this property. Can you achieve this using only arrays? How about using auxiliarry linked data structures?

Check your answer: Problem 2

Radix Sort: Use radix sort to sort the following strings:

`abc bab cba ccc bbb aac abb bac bcc cab aba`

- First, sort (to buckets) by the last letters (and keep stability!), then join the buckets:

→ `cba aba | bab bbb abb cab | abc ccc aac bac bcc`

→ `cba aba bab bbb abb cab abc ccc aac bac bcc`

Next, do the same for the middle letter:

→ `bab cab aac bac | cba aba bbb abb abc | ccc bcc`

→ `bab cab aac bac cba aba bbb abb abc ccc bcc`

Last, do the same for the first letter:

→ `aac aba abb abc | bab bac bbb bcc | cab cba ccc`

→ `aac aba abb abc bab bac bbb bcc cab cba ccc`

Notes on distribution sort

- Not using key comparisons (as opposed to others such as insertion, quick, merge, heap)
- *O(n), but relies on some constraints on data*
- not in-place
- *additional memory needed: O(n+k)*
- can be easily made stable
- not quite practical.

String Searching

Input:

- A (normally long) text T[0..n-1]. Example: T= “SHE SELLS SEA SHELLS”, with n=20
- A (normally short) text pattern P[0..m-1]. Example: P=“HELL”, m=4.

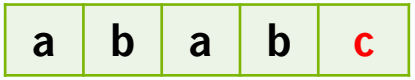
Output:

- index i such that T[i..i+m-1]=P[0..m-1], or NOTFOUND

Algorithms:

- Naïve: brute force, complexity O(nm) (max= (n-m+1)*m character comparison)
- Horspool’s: also O(mn) but practically fast
- Others:

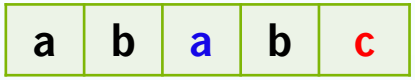
How to run Horspool's *manually*



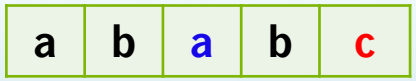
SHIFT m because
y not in P

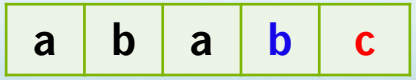
1 comp until mismatch
no matter where mismatch happens, the shift is totally decided by the rightmost examined char of T y

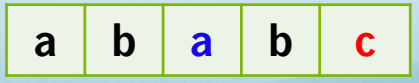
Shift until having the fisrt match of character on P with that rightmost y
(here, no match found)



SHIFT 2 = distance from last a to c







Horspool's Algorithm Review

The task: Seaching for a pattern `P` (such as "HELL" that has length m=5) in a text `T` (such as "SHE SELLS SEA SHELLS", having length n=20).

The Algorithm:

Stage 1: build `SHIFT[x]` for every possible character `x`, by:

1. first, set `SHIFT[x]= m` for all `x`, then
2. for each character `x` in `P`, *except for the last one*: `SHIFT(x)`= distance from the last appearance of `x` to the end of `P`

Stage 2: searching, by first set `i=m-1`, then

1. set `c= P[i]`, align `P` with `T` so that `P[m-1]` aligned with `T[i];`
2. compare characters *backwardly* from the last character of `P` until the start or until finding the first mismatch:
3. if no mismatch found: return solution which is `i-m+1`
4. otherwise, set `i= i+ SHIFT[c]`, back to step 1

Horspool's Algorithm

Problem 4: Use Horspool's algorithm to search for the pattern `GORE` in the string `ALGORITHM`.

Problem 5: How many character comparisons will be made by Hor-spool's algorithm in searching for each of the following patterns it the binary text of one million zeros?

(a) 01001 (b) 00010 (c) 01111

Problem 6 - Horspool's Worst-Case Time Complextity: Using Horspool's method to search in a text of length n for a pattern of length m, what does a worst-case example look like?

Problem 7 – Review: Solve the following recurrence relations. Give both a closed form expression in terms of n and a Big-Theta bound.

a) $T(n)= T(n/2)+1,\ T(1)= 1$ *[using 2 methods]*

b) $T(n)= T(n-1) + n/5,\ T(0)= 0$

c) $T(n)= 3T(n-1) + 1,\ T(1)=1$

d) $T(n)= 2T(n/3) +1,\ T(1)= 1$

R2: Recurrences T(n)= ? T(<n) + $f(n)$ and T(1)=c

- Apply the Master Theorem if possible (ie. if having a, b, and $\boldsymbol{\Theta}(n^d)$ or $O(n^d)$)
- Otherwise, using substitution to expand until T(1)

Note that we can easily make mistakes with substitution. *Do it step by step using draft paper, don't rush.*

Exercises: Solve the following recurrence relations. Give both a closed form expression in terms of n and a Big-Theta bound.

a) $T(n)= T(n/2)+1,\ T(1)= 1$ [using 2 methods]

b) $T(n)= T(n-1) + n/5,\ T(0)= 0$

c) $T(n)= 3T(n-1) + 1,\ T(1)=1$

d) $T(n)= 2T(n/3) +1,\ T(1)= 1$

Other exercises: review exercises and solution for Workshop Week 3, Workshop Week 8 (master theorem)

Lab: Assignment 2 Q&A

- Do assignment 2, and/or
- Review complexity, recurrences, and other parts.

?