COMP20007 Workshop Week 11

LAB	**Preparation:** - *have draft papers and pen ready* - ready to work with assignment 2 Counting & Radix sort: Questions 11.1-11.3 Horspool's Algorithm: Questions 11.4-11.6 Assignment 2: Q&A Assignment 2 Revision on demands: complexity, recurrences, master theorem

Counting Sort

Simple Distribution Sort = Counting Sort

Conditions:

- keys are integers in a small range (small in comparison with n), for example: array of positive integers, each ≤ 2:

```
input array: {0,1,2,0,0,1,2,1,1,0,0,0}
```

freq(0) =6 freq(1)= 4 freq(2)=2

Sorted array: { 0,0,0,0,0,0, 1,1,1,1, 2, 2 }

keys 0 start from index 0

keys 1 starts from index **6**

6 = freq(0)
= freq(**<**1)

keys 2 starts from index 10

10 = freq(0 & 1)
= freq(**<2**)

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

Input: `A[0..n-1]` where
`0 ≤ A[i] ≤ k`, and
`k` is small (`k ≪ n`)

Output: `B[0..n-1]` which is the sorted version of `A[]`

Step 1: build array `C[0..k]` such that `C[i]`= starting index of keys `i`, by:

- First, `C[i+1]`← freq(**`i`**)
- Then accumulate:

```
for i := 1 to k do
    C[i] := C[i-1] + C[i]
```

Step 2: scan `A[]` again and copy to `B`.
For `A[j]` :

```
x := A[j]
B[  C[x] ] := x
C[x]= C[x]+1
```

`A[0..11]= {2,0,1,0,3,1,2,1,1,0,0,0}`
`k= 3`

`C[]= table of frequencies`

idx	0	1	2	3	4=k+1
		5	4	2	1

`C →`

0	5	9	11	

`A[]= {2,0,1,0,3,1,2,1,1,0,0,0}`

`B:`

- complexity=? is stable? is in-place?
- What if: `min ≤ A[i] ≤ max & max-min is small`

Counting Sort

- Can be applied when A[i] in range min..max, where k= max-min+1 is small
- Time complexity: $\Theta(n+k)$, or $\Theta(n)$ if k could be considered as a small constant
- In-place: NO additional memory: $\Theta(n+k)$, or $\Theta(n)$ if k small
- Stable: YES

Bucket Sort:

- counting sort is a special case of bucket sort, where k is the number of buckets
- Bucket sort
  - gather keys into K≤k buckets
  - sort each bucket using a stable auxiliary sort
  - concatenate the sorted buckets

Radix Sort

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

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

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

{1,22,17,167,26,19,28,173,…}

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

→ {01, 16, 11, A7, 1A, 13, 1C, AD…} σ= {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
- Concatenate (join) buckets in increasing order of symbols

Complexity: n * string_size

Q11.1 - 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.

Q11.2 - 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.

Q11.3: 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?

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

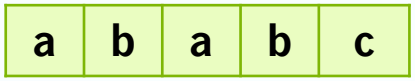
Output:

- 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):
  - shift pattern left to right on the text, 1 position each time
  - compare pattern with text from left to right
- Horspool’s: also O(mn) but practically fast:
  - shift pattern left to right on the text, *at least* 1 position each time
  - compare pattern with text **from right to left**

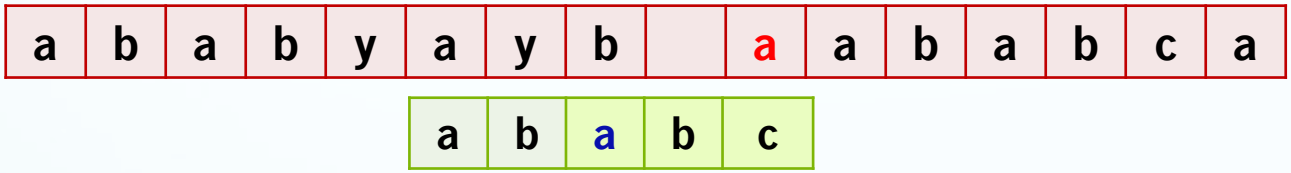
How to run Horspool's *manually*

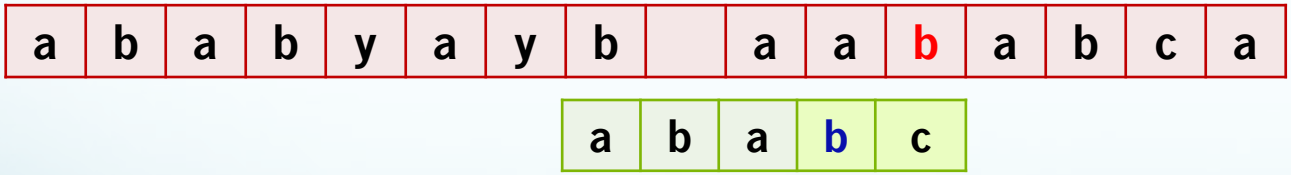


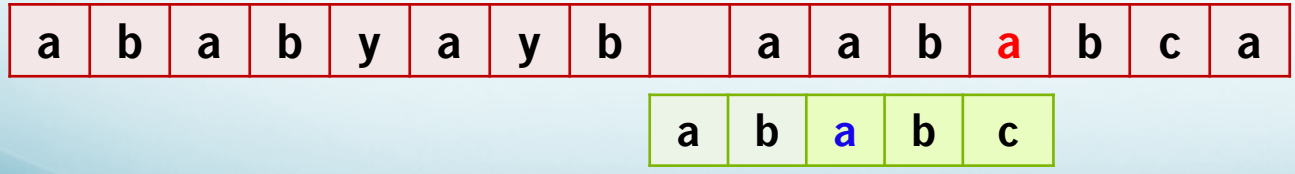
SHIFT m because
y not in P

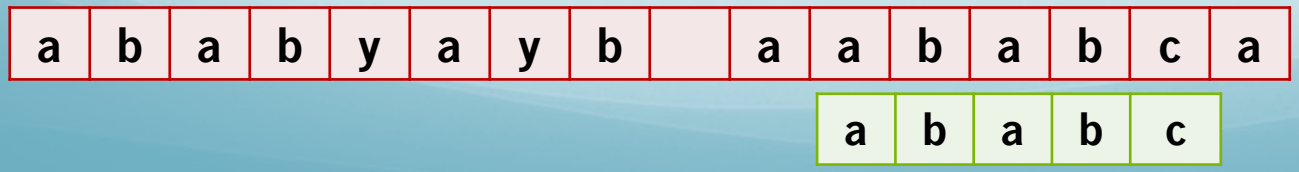
mismatch found at the first comparison
no matter where mismatch happens, the shift is totally decided by the rightmost examined char of T, **y**

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









Horspool's Algorithm Review

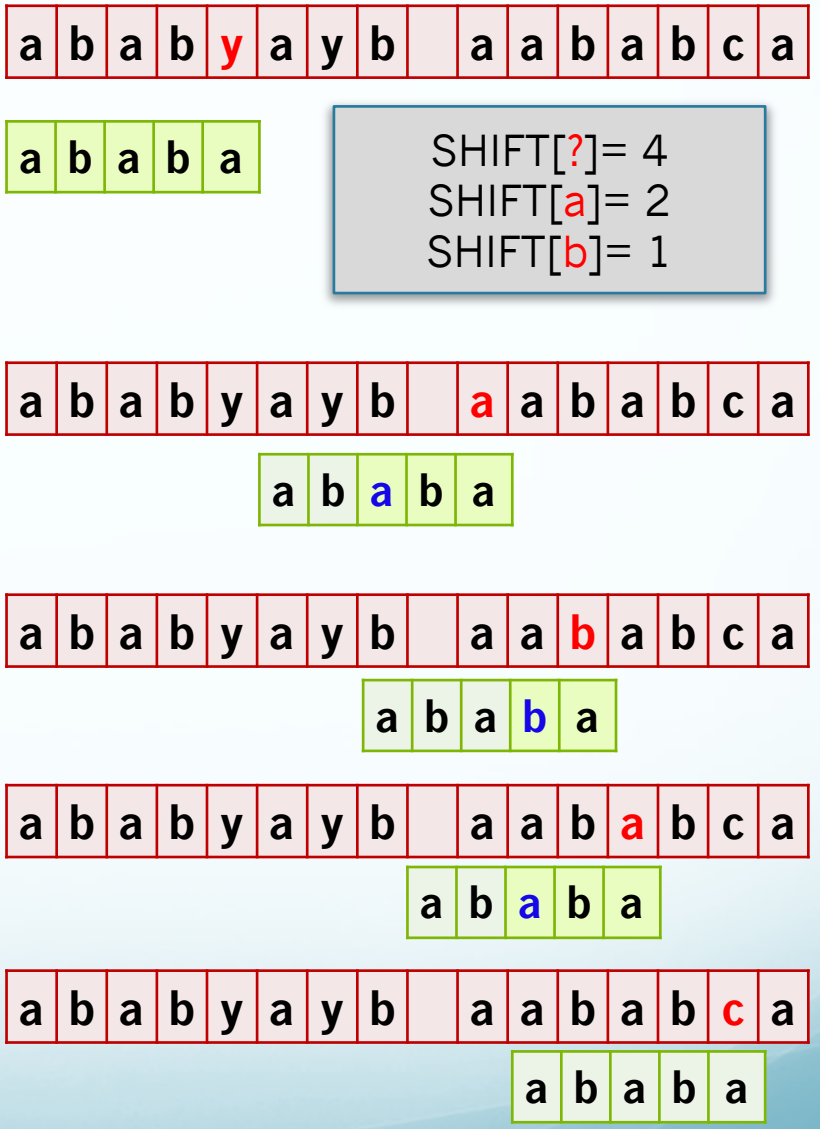
The task: Searching for a pattern `P` (such as "ababa" that has length m=5) in a text `T` (such as "ababyayb aababca", having length n=16).

***Stage 1**: build `SHIFT[x]` for all `x`, by:*

1. set `SHIFT[x]= m` for all `x`, then
2. for each `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= T[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 a 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

Q11.4: Use Horspool's algorithm to search for the pattern `GORE` in the string `ALGORITHM`.

Q11.5: 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

Q11.6 - 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?

Assignment 2: Q&A (Part 1, Part 2, Part 3)

Lab: Assignment 2

- Make sure that you understand the tasks of A2, know what to do, ask questions if in doubt.
- Do assignment 2, further questions, and/or
- Review complexity, recurrences, and other parts.

?