COMP90038 Algorithms and Complexity

Tutorial 9 Time/Space Tradeoffs and Hashing

Tutor: Chuang(Frank) Wang

1. Review

Time/Space tradeoffs

Fibonacci

1. Exponential time

Example: Fibonacci Numbers

To generate the nth number of sequence 1 1 2 3 5 8 13 21 34 55 ...

```
function FIB(n)
    if n = 0 then
        return 1
    if n = 1 then
        return 1
    return FIB(n − 1) + FIB(n − 2)
```

2.. Linear Time with the sacrifice of space

Fibonacci Numbers with Tabulation

We assume that, from the outset, all entries of the table F are 0.

```
function FIB(n)
    if n = 0 or n = 1 then
        return 1
    result ← F[n]
    if result = 0 then
        result ← FIB(n − 1) + FIB(n − 2)
        F[n] ← result
    return result
```

String Matching

1. Brute force - O(nm)

Brute Force String Matching

Pattern p: A string of m characters to search for.

Text t: A long string of n characters to search in.

We use i to run through the text, and j to run through the pattern.

```
for i ← 0 to n − m do
    j ← 0
    while j < m and p[j] = t[i + j] do
        j ← j + 1
    if j = m then
        return i
return −1
```

2. Horspool's String Search Algorithm

Horspool's String Search Algorithm

Building (calculating) the shift table is easy.

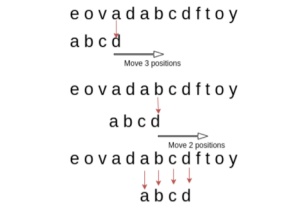
Let a be the size of the alphabet.

```
function FINDSHIFTS(P[·], m)          ▷ Pattern P has length m
    for i ← 0 to a do
        Shift[i] ← m
    for j ← 0 to m − 2 do
        Shift[P[j]] ← m − (j + 1)
```

Horspool's String Search Algorithm

```
function HORSPOOL(P[·], m, T[·], n)
    FINDSHIFTS(P, m)
    i ← m − 1
    while i < n do
        k ← 0
        while k < m and P[m − 1 − k] = T[i − k] do
            k ← k + 1
        if k = m then                          ▷ We have a match
            return i − m + 1                   ▷ Start of the match
        else
            i ← i + Shift[T[i]]                ▷ Slide the pattern along
    return −1
```

Letter	a	b	c	d	*
Value	3	2	1	4	4



Attribution: https://nearsoft.com/blog/the-boyer-moore-horspool-algorithm/

Unfortunately the worst-case behaviour of Horspool's algorithm is still O(m × n), like the brute-force method.

However, in practice, for example, when used on English texts, it is linear, and fast.

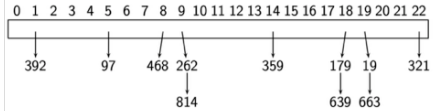
Hashing

1. Def: Key, hash function, hash address, hash table, collision, load factor etc..

2. Separate Chaining

Separate Chaining

Element k of the hash table is a list of keys with the hash value k.



This gives easy collision handling.

The load factor $\alpha = n/m$, where n is the number of items stored.

Number of probes in successful search $\approx (1+\alpha)/2$.

Number of probes in unsuccessful search $\approx \alpha$.

Separate Chaining Pros and Cons

Compared with sequential search, reduces the number of comparisons by a factor of m.

Good in a dynamic environment, when (number of) keys are hard to predict.

The chains can be ordered, or records may be "pulled up front" when accessed.

Deletion is easy.

However, separate chaining uses extra storage for links.

3. Open-Addressing:

linear probing

Linear Probing

In case of collision, try the next cell, then the next, and so on.

After the arrival of 19, 392 (1), 179 (18), 663 (19), 639 (18), 321 (22):

0	1	2	3	.....	16	17	18	19	20	21	22
	392			.....			179	19	663	639	321

Search proceeds in a similar fashion.

If we get to the end of the hash table, we wrap around.

For example, if key 20 now arrives, it will be placed in cell 0.

Linear Probing Pros and Cons

Space-efficient.

Worst-case performance miserable; must be careful not to let the load factor grow beyond 0.9.

Comparative behaviour, $m = 11113$, $n = 10000$, $\alpha = 0.9$:

- Linear probing: 5.5 probes on average (success)
- Binary search: 12.3 probes on average (success)
- Linear search: 5000 probes on average (success)

Clustering is a major problem: The collision handling strategy leads to clusters of contiguous cells being occupied.

Deletion is almost impossible.

double hashing

Double Hashing

To alleviate the clustering problem in linear probing, there are better ways of resolving collisions.

One is double hashing which uses a second hash function s to determine an offset to be used in probing for a free cell.

For example, we may choose $s(k) = 1 + k \bmod 97$.

By this we mean, if $h(k)$ is occupied, next try $h(k) + s(k)$, then $h(k) + 2s(k)$, and so on.

This is another reason why it is good to have m being a prime number. That way, using $h(k)$ as the offset, we will eventually find a free cell if there is one.

4. Rehashing

Rehashing

The standard approach to avoiding performance deterioration in hashing is to keep track of the load factor and to rehash when it reaches, say, 0.9.

Rehashing means allocating a larger hash table (typically about twice the current size), revisiting each item, calculating its hash address in the new table, and inserting it.

This "stop-the-world" operation will introduce long delays at unpredictable times, but it will happen relatively infrequently.

2. Tutorial Questions

Question 67

67. Use Horspool's algorithm to search for the pattern GORE in the string ALGORITHM.

Constructing match table:

Building (calculating) the shift table is easy.

Let a be the size of the alphabet.

```
function FindShifts(P[·], m)        ▷ Pattern P has length m
    for i ← 0 to a do
        Shift[i] ← m
    for j ← 0 to m − 2 do
        Shift[P[j]] ← m − (j + 1)
```

letter	G	O	R	E	*
value	3	2	1	4	4

- value = length – index – 1
- for the last letter:
  value = length if not already defined, otherwise leave
- for every other letter:
  value = length

the algorithm halts (after just two comparisons), reporting failure.

Question 68

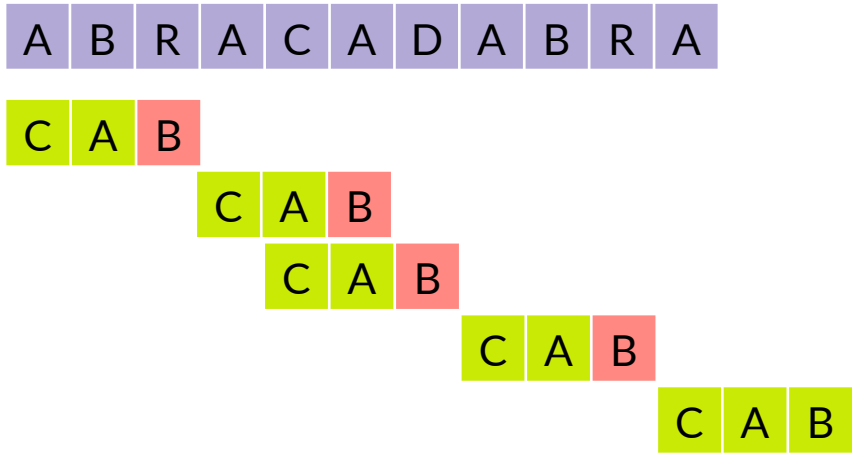
68. How many character comparisons does it take Horspool's algorithm to decided that CAB is not found in ABRACADABRA? How many to find that DRAC is not there?

Constructing match table:

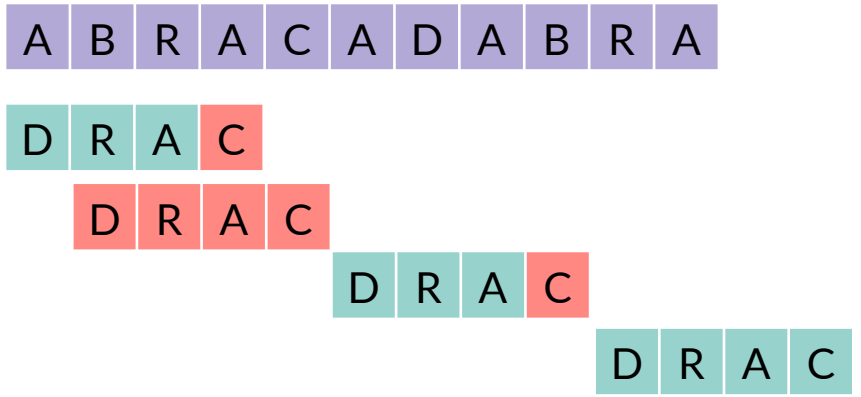
- value = length – index – 1
- for the last letter:
  value = length if not already defined, otherwise leave
- for every other letter:
  value = length

letter	C	A	B	*
value	2	1	3	3

letter	D	R	A	C	*
value	3	2	1	4	4



4



6

Question 69

69. How many character comparisons will be made by Horspool's algorithm in searching for each of the following patterns it the binary text of one million zeros?

(a) 01001

(b) 00010

(c) 01111

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	2	.	.	.	.	.	.	.	.	.	.	.	999,993	999,994	999,995	999,996	999,997	999,998	999,999

0	0	0	1	0

0	0	0	1	0	halts

0	1	0	0	1
0	1	2	3	4

D	0	1	*
V	1	3	5

(a) The pattern's last 1 will be compared against every single 0 in the text (except of course the first four), since the skip will be 1. So 999,996 comparisons.

0	0	0	1	0
0	1	2	3	4

D	0	1	*
V	2	1	5

(b) Here we will will make two comparisons between shifts, and each shift is of length 2. So the answer is again 999,996 = 1m - 3 - 1 comparisons. (see above)

0	1	1	1	1
0	1	2	3	4

D	0	1	*
V	4	1	5

(b) For the last pattern, the skip is 4. So with excluding the first 4 digits, we will make $\frac{1{,}000{,}000 - 4}{4} = 249,999$ comparisons.

Question 70

70. 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?

Answer: Let the text have n zeros and let the pattern be of length $\lceil n/2 \rceil$ and consist of a single 1 followed by zeros. Each skip will then be just a single position, and between skips, $\lceil n/2 \rceil$ comparisons are made. After the first $\lceil n/2 \rceil$ comparisons, we skip $\lfloor n/2 \rfloor$ times. Altogether we have $(1+\lfloor n/2 \rfloor)\lceil n/2 \rceil$ comparisons. If n is even, that is $(n^2+2n)/4$ comparisons. If n is odd, it is $(n^2+2n+1)/4$ comparisons.

Question 72

72. For the input 40, 60, 37, 84, 42, 18, 30, and hash function $h(K) = k \bmod 11$,

(a) construct the open hash table (separate chaining).

(b) find the largest number of key comparisons in a successful search in this table.

(c) find the average number of key comparisons in a successful search in this table.

0	1	2	3	4	5	6	7	8	9	10
				37	60		40	30	42	
							84			
							18			

(b) The largest number of probes in a successful search is 3.

(c) The average is $\frac{1+1+1+1+1+2+3}{7}$ = 1.43.

Question 73

73. For the input 40, 60, 37, 84, 42, 18, 30, and hash function $h(K) = k \bmod 11$,

(a) construct the closed hash table.

(b) find the largest number of key comparisons in a successful search in this table.

(c) find the average number of key comparisons in a successful search in this table.

0	1	2	3	4	5	6	7	8	9	10
30				37	60		40	84	42	18

(b) The largest number of probes in a successful search is 4.

(c) The average is $\frac{1+1+1+1+2+4+4}{7} = 2$

THE UNIVERSITY OF MELBOURNE

Next Week:

- *Dynamic Programming*

Thank you !