



University of
Pittsburgh

Algorithms and Data Structures 1

CS 0445



Fall 2022

Sherif Khattab

ksm73@pitt.edu

Announcements

- Upcoming Deadlines
 - Homework 11: This Friday 12/9 @ 11:59 pm
 - Lab 11: next Monday 12/12
 - Lab 12 and Homework 12: Monday 12/19
 - Assignment 5 is now for extra credit ONLY
 - We have 4 programming assignments
 - the lowest is dropped
 - each worth 13.3%
 - Assignment 3: Friday 12/16 @ 11:59 pm
 - Assignment 4: Friday 12/16 @ 11:59 pm

Bonus Opportunities

- Bonus Lab due on 12/19
- Bonus Homework due on 12/19
- Bonus Assignment due on 12/19
- 1 bonus point for entire class when OMETs response rate $\geq 80\%$
 - Currently at 23%
 - Deadline is Sunday 12/11

Final Exam

- Same format as midterm
- Non-cumulative
- Date, time and location on PeopleSoft
 - Thursday 12/15 8-9:50 am (coffee served!)
- Same classroom as lectures
- Study guide and practice test to be posted soon

Previous Lecture ...

- Hashing!
 - what makes a good hash function
 - Horner's method + modular hashing
 - Handling collisions
 - Open addressing
 - Linear probing

This Lecture ...

- Hashing!
 - Handling collisions
 - Open addressing
 - Double hashing
 - Closed addressing
- String matching

Muddiest Points

- **Q: why do we have iterable interface and iterator interface. As only iterator works here**
- Iterator interface is used to implement iterators
- Iterable interface is used to implement containers that have iterators
 - allows us to use the for-each loop structure

```
IterableLinkedList<Integer> list = new .....  
for(Integer x : list){  
    //do something with x  
}
```

Muddiest Points

- **Q: Can we please get more in class tophat questions? It would be a very helpful way to boost our grades.**
- Sure. Let's have a couple today and next lecture!

Double hashing

- After a collision, instead of attempting to place the key x in $i+1 \bmod m$, look at $i+h_2(x) \bmod m$
 - $h_2()$ is a second, different hash function
 - Should still follow the same general rules as $h()$ to be considered good, but needs to be different from $h()$
 - $h(x) == h(y)$ AND $h_2(x) == h_2(y)$ should be very unlikely
 - Hence, it should be unlikely for two keys to use the same increment

Double hashing

- $h(x) = x \bmod 11$
- $h_2(x) = (x \bmod 7) + 1$
- Insert 14, 17, 25, 37, 34, 16, 26

0	1	2	3	4	5	6	7	8	9	10
	34		14	37	16	17		25		26

- Why could we not use $h_2(x) = x \bmod 7$?
 - Try to insert 2401

A few extra rules for h2()

- Second hash function cannot map a value to 0
- You should try all indices once before trying one twice
- Were either of these issues for linear probing?

As $\alpha \rightarrow 1...$

- Meaning n approaches $m...$
- Both linear probing and double hashing degrade to $\Theta(n)$
 - How?
 - Multiple collisions will occur in both schemes
 - Consider inserts and misses...
 - Both continue until an empty index is found
 - With few indices available, close to m probes will need to be performed
 - $\Theta(m)$
 - n is approaching m , so this turns out to be $\Theta(n)$

Horner's method

```
public long horners_hash(String key, int n) {  
    long h = 0;  
    for (int j = 0; j < n; j++)  
        h = (R * h + key.charAt(j)) % m;  
    return h;  
}
```

`horners_hash("abcd", 4) =`

○ $'a' * R^3 + 'b' * R^2 + 'c' * R + 'd' \% m$

○ **$h = 'a' \% m$**

○ **$h = h * R + 'b' \% m$**

○ $= ('a' \% m) * R + 'b' \% m$

○ **$h = h * R + 'c' \% m$**

○ $= (('a' \% m) * R + 'b' \% m) * R + 'c' \% m$

○ **$h = h * R + 'd' \% m$**

○ $= (((('a' \% m) * R + 'b' \% m) * R + 'c' \% m) * R + 'd'$

Open addressing issues

- Must keep a portion of the table empty to maintain respectable performance
 - For linear hashing $\frac{1}{2}$ is a good rule of thumb
 - Can go higher with double hashing
- What do we do when the hash table is more than half full?
 - resizing!
 - How?

Closed addressing

- i.e., if a pigeon's hole is taken, it lives with a roommate
- Most commonly done with **separate chaining**
 - Create **a linked-list** of keys at each index in the table
 - Similar to Assignment 2!
 - array of linked lists

Closed addressing

- Performance depends on chain length
 - Which is determined by the load factor $\alpha = n/m$ and the quality of the hash function
 - With a good hash function, on average, n/m keys per chain
- In closed addressing, number of keys $n >$ table size m
 - not possible with open addressing

In general...

- Closed-addressing hash tables are fast and efficient for many applications
- Where would open addressing be preferable?
 - Strict memory limits
 - Lack of dynamic memory allocation
 - needed to allocating nodes in the linked lists in separate chaining

String Matching

- Have a pattern string p of length m
- Have a text string t of length n
- Can we find an index i of string t such that each of the m characters in the substring of t starting at i matches each character in p
 - Example: can we find the pattern "fox" in the text "the quick brown fox jumps over the lazy dog"?
 - Yes! At index 16 of the text string!

Simple approach

- BRUTE FORCE
 - Start at the beginning of both pattern and text
 - Compare characters left to right
 - Mismatch?
 - Start again at the 2nd character of the text and the beginning of the pattern...

Brute force code

```
public static int bf_search(String pat, String txt) {  
    int m = pat.length();  
    int n = txt.length();  
    for (int i = 0; i <= n - m; i++) {  
        int j;  
        for (j = 0; j < m; j++) {  
            if (txt.charAt(i + j) != pat.charAt(j))  
                break;  
        }  
        if (j == m)  
            return i; // found at offset i  
    }  
    return n; // not found  
}
```

Brute force Algorithm

i:	0							
text:	A	B	A	B	A	B	A	C
pattern:	A	B	A	B	A	C		
j:	0							

```

public static int bf_search(String pat, String txt)
{
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i <= n - m && j < m; i++) {
        if (txt.charAt(i) == pat.charAt(j))
            j++;
        else { i -= j; j = 0; }
    }
    if (j == m)
        return i - m; // found at offset i
    else return n; // not found
}

```

i:	0	1						
text:	A	B	A	B	A	B	A	C
pattern:	A	B	A	B	A	C		
j:	0	1						

```

public static int bf_search(String pat, String txt)
{
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i <= n - m && j < m; i++) {
        if (txt.charAt(i) == pat.charAt(j))
            j++;
        else { i -= j; j = 0; }
    }
    if (j == m)
        return i - m; // found at offset i
    else return n; // not found
}

```

i:		1	2					
text:	A	B	A	B	A	B	A	C
pattern:	A	B	A	B	A	C		
j:		1	2					

```

public static int bf_search(String pat, String txt)
{
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i <= n - m && j < m; i++) {
        if (txt.charAt(i) == pat.charAt(j))
            j++;
        else { i -= j; j = 0; }
    }
    if (j == m)
        return i - m; // found at offset i
    else return n; // not found
}

```


i:			2	3				
text:	A	B	A	B	A	B	A	C
pattern:	A	B	A	B	A	C		
j:			2	3				

```

public static int bf_search(String pat, String txt)
{
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i <= n - m && j < m; i++) {
        if (txt.charAt(i) == pat.charAt(j))
            j++;
        else { i -= j; j = 0; }
    }
    if (j == m)
        return i - m; // found at offset i
    else return n; // not found
}

```

i:				3	4			
text:	A	B	A	B	A	B	A	C
pattern:	A	B	A	B	A	C		
j:				3	4			

```

public static int bf_search(String pat, String txt)
{
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i <= n - m && j < m; i++) {
        if (txt.charAt(i) == pat.charAt(j))
            j++;
        else { i -= j; j = 0; }
    }
    if (j == m)
        return i - m; // found at offset i
    else return n; // not found
}

```

i:					4	5		
text:	A	B	A	B	A	B	A	C
pattern:	A	B	A	B	A	C		
j:					4	5		

```

public static int bf_search(String pat, String txt)
{
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i <= n - m && j < m; i++) {
        if (txt.charAt(i) == pat.charAt(j))
            j++;
        else { i -= j; j = 0; }
    }
    if (j == m)
        return i - m; // found at offset i
    else return n; // not found
}

```

i:	0					5		
text:	A	B	A	B	A	B	A	C
pattern:	A	B	A	B	A	C		
j:	0					5		

```

public static int bf_search(String pat, String txt)
{
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i <= n - m && j < m; i++) {
        if (txt.charAt(i) == pat.charAt(j))
            j++;
        else { i -= j; j = 0; }
    }
    if (j == m)
        return i - m; // found at offset i
    else return n; // not found
}

```

i:	0	1						
text:	A	B	A	B	A	B	A	C
pattern:	A	B	A	B	A	C		
j:	0							

```

public static int bf_search(String pat, String txt)
{
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i <= n - m && j < m; i++) {
        if (txt.charAt(i) == pat.charAt(j))
            j++;
        else { i -= j; j = 0; }
    }
    if (j == m)
        return i - m; // found at offset i
    else return n; // not found
}

```

i:		1						
text:	A	B	A	B	A	B	A	C
pattern:	A	B	A	B	A	C		
j:	0							

```

public static int bf_search(String pat, String txt)
{
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i <= n - m && j < m; i++) {
        if (txt.charAt(i) == pat.charAt(j))
            j++;
        else { i -= j; j = 0; }
    }
    if (j == m)
        return i - m; // found at offset i
    else return n; // not found
}

```

i:		1	2					
text:	A	B	A	B	A	B	A	C
pattern:	A	B	A	B	A	C		
j:	0							

```

public static int bf_search(String pat, String txt)
{
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i <= n - m && j < m; i++) {
        if (txt.charAt(i) == pat.charAt(j))
            j++;
        else { i -= j; j = 0; }
    }
    if (j == m)
        return i - m; // found at offset i
    else return n; // not found
}

```

i:			2	3				
text:	A	B	A	B	A	B	A	C
pattern:	A	B	A	B	A	C		
j:	0	1						

```

public static int bf_search(String pat, String txt)
{
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i <= n - m && j < m; i++) {
        if (txt.charAt(i) == pat.charAt(j))
            j++;
        else { i -= j; j = 0; }
    }
    if (j == m)
        return i - m; // found at offset i
    else return n; // not found
}

```


i:				3	4			
text:	A	B	A	B	A	B	A	C
pattern:	A	B	A	B	A	C		
j:		1	2					

```

public static int bf_search(String pat, String txt)
{
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i <= n - m && j < m; i++) {
        if (txt.charAt(i) == pat.charAt(j))
            j++;
        else { i -= j; j = 0; }
    }
    if (j == m)
        return i - m; // found at offset i
    else return n; // not found
}

```

i:					4	5		
text:	A	B	A	B	A	B	A	C
pattern:	A	B	A	B	A	C		
j:			2	3				

```

public static int bf_search(String pat, String txt)
{
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i <= n - m && j < m; i++) {
        if (txt.charAt(i) == pat.charAt(j))
            j++;
        else { i -= j; j = 0; }
    }
    if (j == m)
        return i - m; // found at offset i
    else return n; // not found
}

```

i:						5	6	
text:	A	B	A	B	A	B	A	C
pattern:	A	B	A	B	A	C		
j:				3	4			

```

public static int bf_search(String pat, String txt)
{
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i <= n - m && j < m; i++) {
        if (txt.charAt(i) == pat.charAt(j))
            j++;
        else { i -= j; j = 0; }
    }
    if (j == m)
        return i - m; // found at offset i
    else return n; // not found
}

```

i:							6	7
text:	A	B	A	B	A	B	A	C
pattern:	A	B	A	B	A	C		
j:					4	5		

```

public static int bf_search(String pat, String txt)
{
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i < n && j < m; i++) {
        if (txt.charAt(i) == pat.charAt(j))
            j++;
        else { i -= j; j = 0; }
    }
    if (j == m)
        return i - m; // found at offset i
    else return n; // not found
}


```

i:								7	8
text:	A	B	A	B	A	B	A	C	
pattern:	A	B	A	B	A	C			
j:						5	6		

```

public static int bf_search(String pat, String txt)
{
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i < n && j < m; i++) {
        if (txt.charAt(i) == pat.charAt(j))
            j++;
        else { i -= j; j = 0; }
    }
    if (j == m)
        return i - m; // found at offset i
    else return n; // not found
}

```

i:								8
text:	A	B	A	B	A	B	A	C
pattern:	A	B	A	B	A	C		
j:							6	

```

public static int bf_search(String pat, String txt)
{
    int j, m = pat.length();
    int i, n = txt.length();
    for (i = 0, j = 0; i < n && j < m; i++) {
        if (txt.charAt(i) == pat.charAt(j))
            j++;
        else { i -= j; j = 0; }
    }
    if (j == m)
        return i - m; // found at offset i
    else return n; // not found
}

```