COMP 2011: Data Structures

Lecture 11. Hashing

Dr. CAO Yixin

November, 2021

Review of Lecture 10

- Array implementation of heaps: $2i+1$ and $2i+2$ are children of node i.
- Priority queues

  Most commonly implemented with heaps.

  But conceptually different from heaps.
- In-place heapsort.

  $O(n \log n)$ time and $O(1)$ space.

  Not stable.
- Applications of heaps.

  Very useful because a heap is partially sorted.

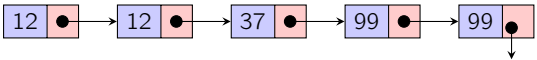
Motivations

I have a dream

that all the operations can be done in constant time.

A question

How to remove duplicated elements from a sorted linked list?

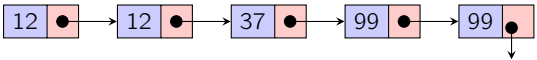


How about a sorted array?

12	12	12	37	37	99	99	99

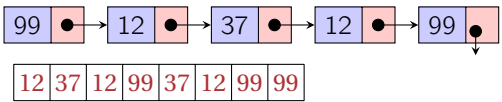
A question

How to remove duplicated elements from a sorted linked list?



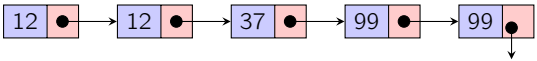
How about a sorted array?

What if they are not sorted?



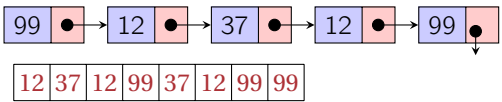
A question

How to remove duplicated elements from a sorted linked list?



How about a sorted array?

What if they are not sorted?



We can sort it first.
But sorting is costly and
should be avoided whenever possible.

Déjá vu

Let's relax the question:
You only need to detect whether there are duplicates.

Déjá vu

Let's relax the question:
You only need to detect whether there are duplicates.

```
boolean duplicateChar(String s) {
  boolean[] b = new boolean[256];
  for (int i = 0; i < s.length(); i++) {
    int c = s.charAt(i);
    if (b[c]) return true;
    else b[c] = true;
  }
  return false;
}
```

An example:
To detect duplicate characters from a string.

Does it apply to the first question? Why?

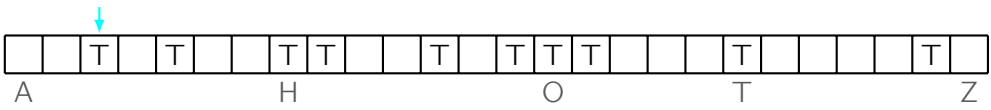
Are there duplicate characters?

茶孑孓睢麈汩姬祖甲霒已汨汆戍祇祇匸采鷙睢垚壿晝
巳毫禕口姬福萬祖鷙申毫由己理壺茶土曰袑芊徽壿汆
袑洗市戊夂妹麈芙魍冼嫩毳士口幣埋币福戌朘芊祇壿
萭匸祇采壿禕市壺戍畫晢日夂芙垚由妹晢甲戌戊微朘

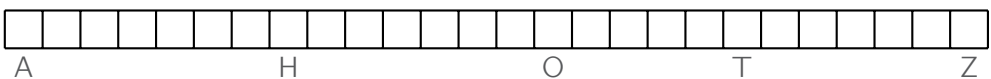
link 1 link 2

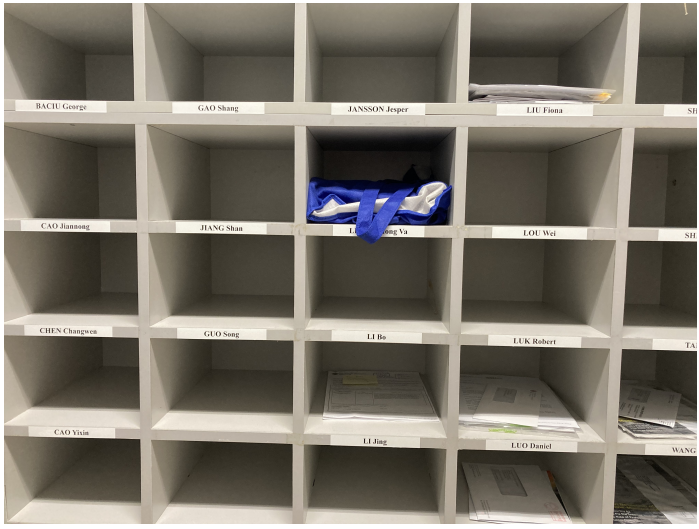
POLYTECHNIC

POLYTECHNIC



香港理工大學



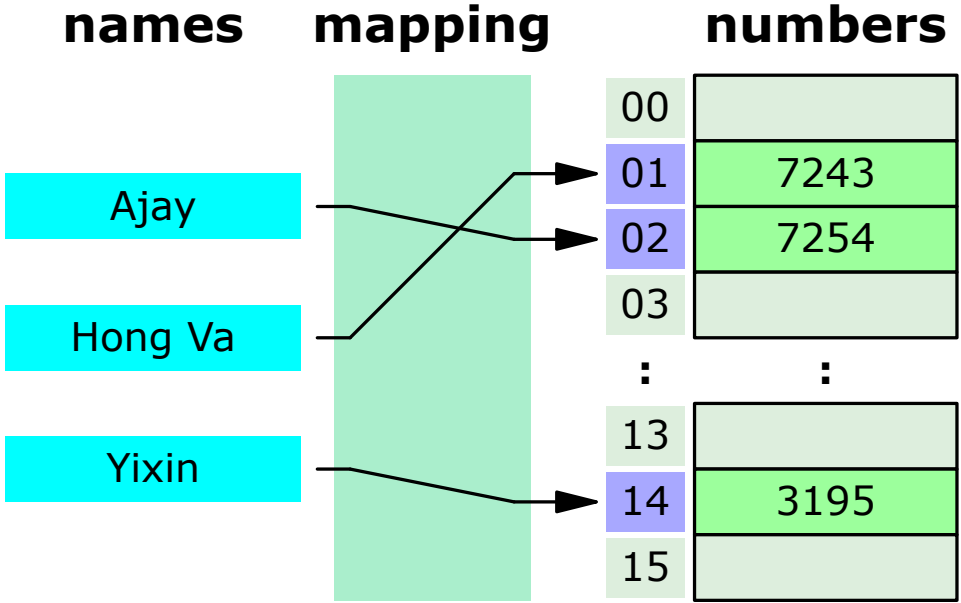


pigeonholes

pigeon holes

butterfly

butter fly


names
mapping
numbers
Ajay
Hong Va
Yixin
00
01
02
03
13
14
15
7243
7254
3195

Example 1: subjects

```
public static void main(String args[]){
  Hashtable<String, String> courses = new Hashtable<String,String>();
  courses.put("COMP1001", "PROBLEM SOLVING METHODOLOGY IN...");
  courses.put("COMP2011", "DATA STRUCTURES");
  courses.put("COMP2021", "OBJECT-ORIENTED PROGRAMMING");
  courses.put("COMP2121", "E-BUSINESS");
  courses.put("COMP2411", "DATABASE SYSTEMS");
  courses.put("COMP2421", "COMPUTER ORGANIZATION");
  courses.put("COMP3011", "DESIGN AND ANALYSIS OF ALGORITHMS");
  courses.put("COMP305",  "DATA STRUCTURES AND ALGORITHMS");
  // ...
  courses.put("COMP5517", "HUMAN COMPUTER INTERACTION");
  courses.put("COMP5538", "CUSTOMER RELATIONSHIP MANAGEMENT...");

  courses.put("COMP2011", "DATA STRUCTURES AND ALGORITHMS");

  System.out.println(courses.containsKey("COMP0000"));
  System.out.println(courses.get("COMP2011"));
}
```

How these data are stored?

Example 2: colors (🌍)

```
public static void main(String args[]){
  Map<String, Color> map = new HashMap<String, Color>();
  map.put("Aquamarine", new Color(0x7F, 0xFF, 0xD0));
  map.put("Black", new Color(0, 0, 0));
  map.put("white", new Color(255, 255, 255));
  map.put("Cyan", new Color(0x00, 0xFF, 0xFF));
  map.put("Dark␣blue", new Color(0x00, 0x00, 0x8B));
  map.put("Maroon", new Color(0x80, 0x00, 0x00));
  map.put("PolyU␣Red", new Color(160, 35, 55));
  map.put("PolyU␣Grey", new Color(88, 89, 91));
  map.put("Royal␣blue", new Color(0x00, 0x23, 0x66));
  map.put("Silver", new Color(0xC0, 0xC0, 0xC0));
  map.put("Violet", new Color(0x8F, 0x00, 0xFF));

  System.out.println("PolyU␣Red:␣" + map.get("PolyU␣Red"));
}
```

How these data are stored?

Example 3: domain names and IP addresses ()

```
public static void main(String args[]){
  Map<String, IP> map = new HashMap<String, IP>();
  map.put("www.polyu.edu.hk", new IP(158, 132, 48, 76));
  map.put("www.comp.polyu.edu.hk", new IP(158, 132, 20, 226))
  map.put("www.google.com", new IP(172, 217, 24, 68));
  map.put("www.youtube.com", new IP(216, 58, 200, 78));
  map.put("www.facebook.com", new IP(31, 13, 77, 35));
  map.put("www.amazon.com", new IP(13, 224, 154, 9));
  map.put("www.google.com.hk", new IP(216, 58, 199, 3));
  map.put("www.wikipedia.org", new IP(103, 102, 166, 224));
  map.put("www.zoom.us", new IP(3, 235, 71, 135));
  map.put("www.hktvmall.com", new IP(14, 198, 244, 42));
  map.put("www.cti.hk", new IP(14, 198, 244, 42));

  System.out.println("The␣IP␣address␣of␣wikipedia␣is␣" + map.
}
```

How these data are stored?

Unit	# Bytes	Approximate
KB	1024	$\mathbf{10}^3$
MB	1,048,576	$\mathbf{10}^6$
GB	1,073,741,824	$\mathbf{10}^9$
TB	1,099,511,627,776	$\mathbf{10}^{12}$
PB	1,125,899,906,842,624	$\mathbf{10}^{15}$

Motivation

- A naïve idea is to use an array of size _____.
- But the department is offering less than one hundred courses.
- Most of the space will be wasted: Less than $\frac{100}{10{,}000} = 1\%$ is nonempty.

- How about a sorted array with no empty slot?

- If this example above is not shocking enough, consider the following:

Either space-squandering or slow.

Motivation

- A naïve idea is to use an array of size _____.
- But the department is offering less than one hundred courses.
- Most of the space will be wasted: Less than $\frac{100}{10{,}000} = 1\%$ is nonempty.

- How about a sorted array with no empty slot?

- If this example above is not shocking enough, consider the following:

Suppose you're running a website.
You want to keep track of the visitors of your website.

- When a visitor is detected, check whether it's a returned user.
- Add a record (if new); update the existing record (otherwise).

For most people, this is *the definition* of big data.

Users statistics of a web site

- For simplicity, we assume each IP address corresponds to a user. ()
- There are around 2^{32} IP addresses in total.
- You can only expect a small fraction of them, say, 1 million, to visit your website.
- Obviously, using a big array (of size 2^{32}) is completely out of the question.
- Even worse for IPv6, which provides 2^{128} (approximately 3403×10^{38}) addresses.
- How about a (sorted) array?

This assumption is wrong, but useful.

Users statistics of a web site

- For simplicity, we assume each IP address corresponds to a user. ()
- There are around 2^{32} IP addresses in total.
- You can only expect a small fraction of them, say, 1 million, to visit your website.
- Obviously, using a big array (of size 2^{32}) is completely out of the question.
- Even worse for IPv6, which provides 2^{128} (approximately 3403×10^{38}) addresses.
- How about a (sorted) array?

Even Google!

Users statistics of a web site

- For simplicity, we assume each IP address corresponds to a user. ()
- There are around 2^{32} IP addresses in total.
- You can only expect a small fraction of them, say, 1 million, to visit your website.
- Obviously, using a big array (of size 2^{32}) is completely out of the question.
- Even worse for IPv6, which provides 2^{128} (approximately 3403×10^{38}) addresses.
- How about a (sorted) array?

What's the problem?

A simple solution

- Use an array of size 65536,
  each using a list to store addresses $a.b.*.*$ for different a and b.
- Visitors of a website tend to be from the same regional area, whose IP addresses are clustered for a and b.
- Use an array of size 65536,
  each using a list to store addresses $*.*.c.d$ for different c and d.
- It's still possible that some lists are very long, but less likely.
- Each list contains about $10^6/65536 \approx 16$.
- Both `check` and `insert` can be done in $O(16)$ steps, *on average*.

Recall that an IP address is a.b.c.d with $0 \le a, b, c, d \le 255$.

A big problem! What is it?

A simple solution

- Use an array of size 65536,
  each using a list to store addresses $a.b.*.*$ for different a and b.
- Visitors of a website tend to be from the same regional area, whose IP addresses are clustered for a and b.
- Use an array of size 65536,
  each using a list to store addresses $*.*.c.d$ for different c and d.
- It's still possible that some lists are very long, but less likely.
- Each list contains about $10^6/65536 \approx 16$.
- Both `check` and `insert` can be done in $O(16)$ steps, *on average*.

A simple solution

- Use an array of size 65536,
  each using a list to store addresses $a.b.*.*$ for different a and b.
- Visitors of a website tend to be from the same regional area, whose IP addresses are clustered for a and b.
- Use an array of size 65536,
  each using a list to store addresses $*.*.c.d$ for different c and d.
- It's still possible that some lists are very long, but less likely.
- Each list contains about $10^6/65536 \approx 16$.
- Both `check` and `insert` can be done in $O(16)$ steps, *on average*.

There is an even better idea.

Can you make 16 the worst case?

A toy example

```
public static int h(int ip) {
    return (ip & 0x7FFFFFFF) % 0x10000;
}

public static void main(String[] args) {
  int size = 10000;
  SecureRandom random = new SecureRandom();
  for (int i = 0; i < size; i++) {
    int ip = random.nextInt();
    System.out.println(display(ip)
        + " in the box " + h(ip));
  }
}
```

In Java, the % operator may return negative numbers.

In this toy example, we treat a (4-byte) integer as an IPv4 address.

Hash Tables

The problem

Input: n elements with keys from the range of $1..N$, with $N \gg n$.

Task: To store them in a structure of size M, where $N \gg M$, such that insertion, deletion, and search can be done in $O(1)$ average time.

Can we assume all elements to be integers? Why?

The problem

Input: n elements with keys from the range of $1..N$, with $N \gg n$.

Task: To store them in a structure of size M, where $N \gg M$, such that insertion, deletion, and search can be done in $O(1)$ average time.

- The *only* structure providing $O(1)$ access to every element is Array.
- So, we have no choice but to use an array.
- The real challenge is to use a small array ($M \ll N$).

Random access of arrays:
the ith entry of array A can be accessed in $O(1)$ time, by calculating the address of $A[i]$, which is i steps away from the starting address of A.

The problem

Input: n elements with keys from the range of $1..N$, with $N \gg n$.

Task: To store them in a structure of size M, where $N \gg M$, such that insertion, deletion, and search can be done in $O(1)$ average time.

- The *only* structure providing $O(1)$ access to every element is Array.
- So, we have no choice but to use an array.
- The real challenge is to use a small array ($M \ll N$).

Random access of arrays:
the ith entry of array A can be accessed in $O(1)$ time, by calculating the address of $A[i]$, which is i steps away from the starting address of A.

The problem

Input: n elements with keys from the range of $1..N$, with $N \gg n$.

Task: To store them in a structure of size M, where $N \gg M$, such that insertion, deletion, and search can be done in $O(1)$ average time.

Note that the relation between n and M isn't specified.

The ideal case

The ideal case

Suppose there exists a function $h : [1..N] \to [0..M-1]$ on the elements such that

1. h can be computed in $O(1)$ time; and
2. $h(k_1) \neq h(k_2)$ for different keys ($k_1 \neq k_2$).

To make them happen, we must have $M \geq n$.

The ideal case

The ideal case

Suppose there exists a function $h : [1..N] \to [0..M-1]$ on the elements such that

1. h can be computed in $O(1)$ time; and
2. $h(k_1) \neq h(k_2)$ for different keys ($k_1 \neq k_2$).

To make them happen, we must have $M \geq n$.

Then

- To insert element x with key k: $A[h(k)] \leftarrow x$.
- To search for element with key k: check if $A[h(k)]$ is empty.
- To delete the element with key k: set $A[h(k)] \leftarrow \texttt{null}$.

The ideal case

The ideal case

Suppose there exists a function $h : [1..N] \to [0..M-1]$ on the elements such that

1. h can be computed in $O(1)$ time; and
2. $h(k_1) \neq h(k_2)$ for different keys ($k_1 \neq k_2$).

To make them happen, we must have $M \geq n$.

Then

- To insert element x with key k: $A[h(k)] \leftarrow x$.
- To search for element with key k: check if $A[h(k)]$ is empty.
- To delete the element with key k: set $A[h(k)] \leftarrow \texttt{null}$.

The ideal case

The ideal case

Suppose there exists a function $h : [1..N] \to [0..M-1]$ on the elements such that

1. h can be computed in $O(1)$ time; and
2. $h(k_1) \neq h(k_2)$ for different keys ($k_1 \neq k_2$).

Then

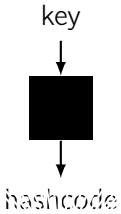
- To insert element x with key k: $A[h(k)] \leftarrow x$.
- To search for element with key k: check if $A[h(k)]$ is empty.
- To delete the element with key k: set $A[h(k)] \leftarrow \texttt{null}$.

Hashing

Hashing (🌍)

Meaning of the word: Merriam Webster

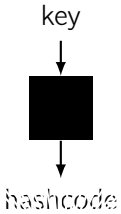
打到你呀媽都唔認得



Hashing (🌐)

Meaning of the word: Merriam Webster

打到你呀媽都唔認得



Examples:

- IP addresses (in the example website visitor monitor). Not good enough.
- File checksum (md5 sha256 sha512)
  JDK Get-FileHash Eclipse
- Shadowed password (🌐)

Hashing in Java

- All Java classes have method `hashCode()`, which returns a 32-bit `int`. Java API

- Must. If `x.equals(y)`, then `(x.hashCode() == y.hashCode())`.
- Highly desirable. If `!x.equals(y)`, then `(x.hashCode() != y.hashCode())`.

- Default implementation. converted from the internal address of the object.
- Customized implementations. `Integer`, `Long`, `Double`, `String`, `InetAddress`.

Hashing in Java

- All Java classes have method `hashCode()`, which returns a 32-bit `int`. Java API

- Must. If `x.equals(y)`, then `(x.hashCode() == y.hashCode())`.
- Highly desirable. If `!x.equals(y)`, then `(x.hashCode() != y.hashCode())`.

- Default implementation. converted from the internal address of the object.
- Customized implementations. `Integer`, `Long`, `Double`, `String`, `InetAddress`.

Implementations of hashCode (edited)

Double

```
int hashCode() {
  long bits = doubleToLongBit
  return (int)(bits ^ (bits >
}
```

Inet4Address

```
int hashCode() {
  return holder().getAddress(
}
```

Boolean

```
int hashCode() {
  return value?1231:1237;
}
```

Inet6Address

```
int hashCode() {
  if (ip != null) {
    int hash = 0;
    int i=0;
    while (i<INADDRSZ) {
      int j=0;
      int comp=0;
      while (j<4 && i<INADDRSZ) {
        comp=(comp<<8) + ip[i];
        j++;
        i++;
      }
      hash += comp;
    }
    return hash;
  }
```

The Implementation in `java.lang.String` (edited)

```
1  final class String {
2    private final char[] s;
3    private int hash = 0;
4    ...
5    int hashCode() {
6      int h = hash;
7      if (h != 0) return h;
8      for (int i = 0; i < length(); i++)
9        hash = s[i] + (31 * hash);
10     hash = h;
11     return hash;
12   }
13 }
```

What if `s.hashCode()` is 0?

For a string s of ℓ letters:

$$h(s) = s[0]\cdot 31^{\ell-1} + \cdots + s[\ell-2]\cdot 31^1 + s[\ell-1]\cdot 31^0.$$

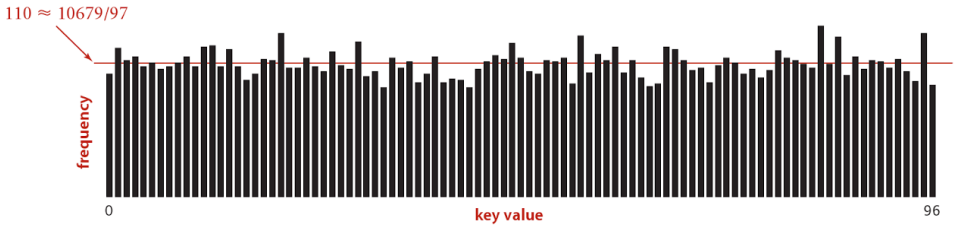
"call" ('c' = 99, 'a' = 97, 'l' = 108)

$$\begin{aligned} &99\cdot 31^3 + 97\cdot 31^2 + 108\cdot 31^1 + 108\cdot 31^0 \\ &= 108 + 31\cdot(108 + 31\cdot(97 + 31\cdot(99))) \\ &= 3045982 \end{aligned}$$

Prime numbers ()

- Prime number are very important for computing.
- All of 31, 1231, 1237 are prime.
- Which of the following are prime?

  2011, 2013, 2015, 2017, 2019, 2021, 2023, 2025, 2027, 2029



Hash value frequencies for words in *Tale of Two Cities* (10,679 keys, $M = 97$)

source: "Algorithms," Sedgewick and Wayne, P.463

A naïve recipe for `hashCode`

"Standard" recipe for user-defined types.

- Combine each significant field using the $31x + y$ rule.
- If a field is a primitive type, use wrapper type `hashCode()`.
- If a field is `null`, return 0.
- If a field is a reference type, use its `hashCode()`.
- If a field is an array, apply to each entry.

Very similar to the `String.hashCdoe()`: each field as a char.

- This recipe works reasonably well in practice; used in Java libraries.
- We usually use an array far smaller than 2^{32}, so `hashCode()` is not sufficient.
- You may need a better idea if the hash function is crucial for your application.

Compromises

The problem

Input: n elements with keys from the range of $1..N$, with $N \gg n$.

Task: To store them in a structure of size M, where $N \gg M$, such that insertion, deletion, and search can be done in $O(1)$ average time.

Although it sounds impossible, we've seen it happened somehow.

You've to make compromises; what're them?

Compromises

The problem

Input: n elements with keys from the range of $1..N$, with $N \gg n$.

Task: To store them in a structure of size M, where $N \gg M$, such that insertion, deletion, and search can be done in $O(1)$ average time.

Although it sounds impossible, we've seen it happened somehow.

You've to make compromises; what're them?

Disadvantages:

- operations `min`, `max`, `predecessor`, `successor` take $O(n)$ time;
- it cannot be sorted *internally*; and
- performance may degrade significantly when the table is too full (i.e., $n \approx M$).

Summary

- A hash table is based on an array.
- The range of key values is usually greater than the size of the array.
- A key value is hashed to an array index by a hash function.
- Hash table sizes should generally be prime numbers.

Week 13: Collision Handling and Conclusions