CS 367 - Introduction to Data Structures
Week 7, 2017

Program 3 released, due 10pm Thursday, August 8th
Homework 6 released, complete as soon as possible. Due by 10 pm Sunday, August 6th

Last Week

Graphs: undirected, directed, weighted, operation, representations: adjacency-matrix, adjacency-list, traversals, applications of BFS/DFS, more terminology

This Week

Read: Hashing

Finish Graphs

- topological ordering
- Dijkstra's algorithm

Hashing

- terminology
- designing a good hash function
- choosing table size
- expanding a hash table
- handling collisions
- expanding a hash table
- handling collisions

Java Support for Hashing

Tree Map vs. Hash Map

Sorting Intro

Basic Sorts

- bubble sort
- insertion sort
- selection sort

Better Sorts

- heap sort
- merge sort

Next Week

Read: continue *Sorting*

Better Sorts

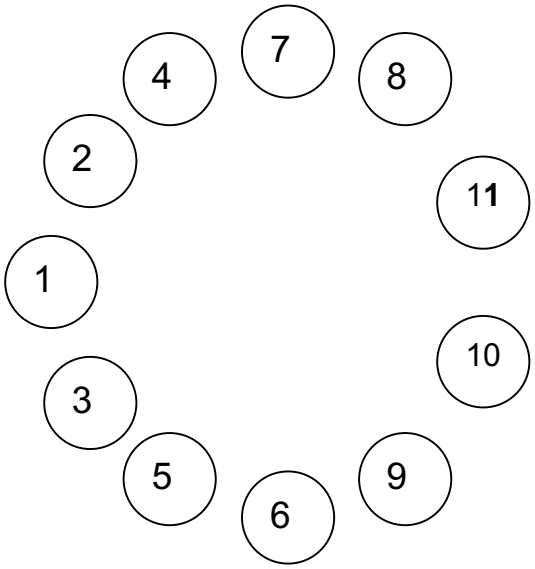
- heap sort
- merge sort
- quick sort

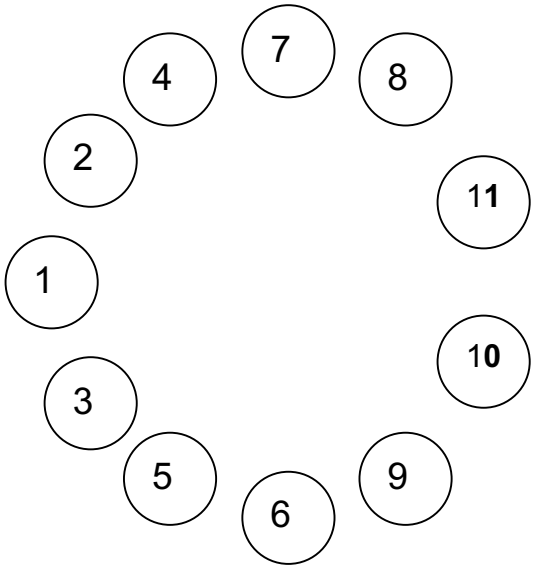
Course Evaluations

Midterm 2 Exam Review

Topological Ordering

1. get bread
2. get jelly
3. get peanut butter
4. get butter knife
5. open jelly
6. open peanut butter
7. take bread slice 1
8. take bread slice 2
9. use knife to spread jelly on bread slice
10. use knife to spread peanut butter on bread slice
11. put slices together with spread sides facing each other



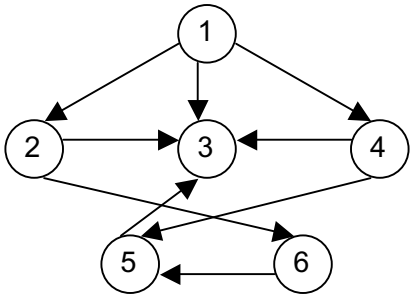


IDEA:

Topological Ordering Algorithm

Iterative Algorithm (see readings for recursive algo)

Example



Dijkstra's Algorithm

-
-
-
-
-
-

Psuedo Code

```
for each vertex V
   initialize V's visited mark to false
   initialize V's total weight to "infinity"
   initialize V's predecessor to null

set start vertex's total weight to 0

create new priority queue pq
pq.insert( [start vertex total weight,start vertex] )

while !pq.isEmpty()
    [C's total weight,C] = pq.removeMin()
   set C's visited mark to true

   for each unvisited successor S adjacent to C
      if S's total weight can be reduced
         S's total weight = C's total weight + edge weight from C to S
         update S's predecessor to C
         pq.insert( [S's total weight,S] )
         (if S already in pq we'll just update S's total weight)
```

Dijkstra's Practice

Iteration	Priority Queue (just list smallest to largest)
0	
1	
2	
3	
4	
5	
6	
7	

Vertex	Visited	Total Weight	Predecessor
A			
B			
C			
D			
E			
F			
H			
G			

Reconstruct shortest path from A to F

Hashing

Goal:

Concept:

hash table

table size (TS)

load factor (LF)

key

hash function

Ideal Hashing

Assume

- store 150 students records
- table is an array of student records
- null is sentinel value meaning element is unused
- key is the student id number, a 5 digit integer

  11000, 11001, 11002, … 11048, 11049, … 11148, 11149

→ What would be a good hash function to use on the ID number?

```
int hash(K key) {


}
```

Trivial Hash Function:

Perfect Hash Function:

```
void insert(K key, D data) {

D    lookup(K key)         {

void delete(K key)         {
```

The UW uses 10 digit ID numbers: 9012345789 9012345432 9023456789

→ Is a perfect hash function possible for these id numbers?

→ Would the last 3 digits of the ID work as above?

Collision:

Key Issues:

-
-
-

Designing a Hash Function

Good Hash Functions:

1.

2.

3.

4.

Java Hash Function Steps:

1.

2.

Techniques for Generating Hash Codes

Integer Key 90123456789

```
123 * 11      +     456 * 121     +     789 * 1
```

Extraction

Weighting

Folding

Handling `String` Keys

Handling `Double` Keys

Choosing the Table Size

Table Size and Collisions

Assume 100 items with random keys in the range 0 – 9999 are being stored in a hashtable.
Also assume the hash function is simply %tablesize.

→ How likely would a collision occur if the table had 10000 elements? 1000? 100?

Table Size and Distribution

Assume 50 items are stored in a hashtable.
Also assume the hashCode function returns multiples of some value x.
For example, if x = 20 then hashCode returns 20, 40, 60, 80, 100, ...

→ How likely would a collision occur if the table had 60 elements? 50? 37?

Resizing the Hash Table

Naïve Expand

30		17	88	

Rehashing

12.

13.

Complexity

Collision Handling using Open Addressing

Open Addressing

Linear Probing

166
359
263

440		266	124	246			337			351

Collision Handling using Open Addressing

Quadratic Probing

166
359
263

440		266	124	246			337			351

Double Hashing

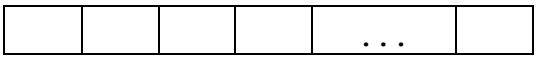
probe sequences assuming H_k is index 0:

Step size	Table size 10	Table size 11
2		
5		

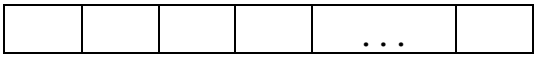
Collision Handling using Buckets

Buckets

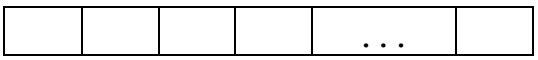
Array Buckets



"Chained" Buckets



Tree Buckets



Java API Support for Hashing

`hashCode` method

- method of Object class
- returns an int
- default hash code is BAD - computed from object's memory address

Guidelines for overriding `hashCode`:

`Hashtable<K,V>` and `HashMap<K,V>` class

- in java.util package
- implement `Map<K,V>` interface
  `K`
  `V`
  operations:

- constructors allow you to set
  initial capacity (default = 16 for HashMap, 11 for HashTable)
  load factor (default = 0.75)
- handles collisions with chained buckets
- HashMap only:
- Hashtable only:

TreeMap vs HashMap

	TreeMap	HashMap

Sorting

Problem

Solution

Complexity

In-Place Sorts

Basic In-Place Comparison Sorts

Bubble Sort

Idea

Psuedocode

```
int passes = A.length-1;

for (int i = 0; i < passes                  ; i++) {


   for (int j = A.length-1; j > i; j--) {
      if (A[j] < A[j-1]) {
         swap(A[j], A[j-1]);

      }
   }
}
```

Analysis

	best case	worst case
kind of array		
# comparisons		
# swaps		
total		

Insertion Sort

Idea

Psuedocode (linear insertion)

```
for (int i = 1; i < A.length; i++) {
   int temp = A[i];

   int j;
   for (j = i-1; j >= 0 && A[j] > temp; j--)
      A[j+1] = A[j];

   A[j+1] = temp;
}
```

Analysis

	best case	worst case
kind of array		
# comparisons		
# shifts		
total		

Selection Sort

Idea

Psuedocode

```
int passes = A.length-1;
for (int i = 0; i < passes; i++) {
   int minIndex = i;

   for (int j = i+1; j < A.length; j++) {
      if (A[j] < A[minIndex])
         minIndex = j;
   }

   swap(A[minIndex],A[i]);
}
```

Analysis

	best case	worst case
kind of array		
# comparisons		
# swaps		
total		