Administrative Details

Reminders

• No lab this week

• Final exam
  • Monday, December 17 at 9:30 in TPL 203
  • Covers everything, with strong emphasis on post-midterm
  • Study guide, sample exam will be posted on handouts page
Topics Covered

- Vectors (and arrays)
- Complexity (big O)
- Recursion + Induction
- Searching
- Sorting
- Linked Lists (SLL & DLL)
- Stacks
- Queues
- Iterators
- Bitwise operations
- Comparables/Comparators
- Ordered Structures
- Binary Trees
- Priority Queues
- Heaps
- Binary Search Trees
- Graphs
- Maps/Hashtables
Last Time

- Graph applications (more in Ch 16)
  - Prim’s algorithm for MCST
  - Dijkstra’s Algorithm for shortest paths
    - Single source
Today’s Outline

• Finish Dijkstra’s algorithm

• Maps
  • Revisit Naïve implementation from Lab 2
  • structure5.Hashtable (finally)
    • Hash functions
    • “Load factor”
    • Collisions and how to handle them

• You should also read Ch 15 for more info
Single Source Shortest Paths

The Input: A graph $G$ such that each edge has a positive cost and a starting vertex $v$.

The Output: For each vertex $u \neq v$ reachable from $v$, a shortest path $P_u$ from $v$ to $u$.

Graph can be directed or undirected

The method: Dijkstra’s Algorithm: Grow a tree
Finding the Best Vertex to Add to $T_k$

Not all edges are displayed

Question: Can we find the next closest vertex to $s$?
What’s a Good Greedy Choice?

Idea: Pick edge e from u in $T_k$ to v in $G-T_k$ that minimizes the length of the tree path from s up to—and through—e.

Now add v and e to $T_k$ to get tree $T_{k+1}$.

Now $T_{k+1}$ is a tree consisting of shortest paths from s to the k vertices closest to s! [Proof?] Repeat!
Some Notation Reminders

- \( l(e) \): length (weight) of edge \( e \)
- \( d(u,v) \): distance from \( u \) to \( v \)
  - Length of shortest path from \( u \) to \( v \)

- The priority queue stores an estimate of the distance from \( s \) to \( w \) by storing, for some edge \((v,w)\), \( d(s,v) + l(v,w) \)
  - The estimate is always an upper bound on \( d(s,w) \)
Dijkstra: What Do We Return?

- As we find a new vertex $v$ to add to the tree $T$ from some $u$ in $T$, add info to a PQ and a Map.
- Precisely:
  - Use a PQ of association $(X, Y)$ `edgInfo` where
    - $X$ is $d(s, v) + l(v, w)$
    - $Y$ is the edge $e = (v, w)$
  - Add all edges from $v$ to $w$, $w$ not in $T$, to the PQ
  - Add the key/value pair $(v, u)$ to the Map
- So the map entry with key $v$ tells us the vertex $u$ that precedes $v$ on shortest path from $s$ to $v$
Dijkstra’s Algorithm

\[\text{Dijkstra}(G, s) \quad \text{// } l(e) \text{ is the length of edge } e\]

let \(T \leftarrow \{s\}, \emptyset\) and \(PQ\) be an empty priority queue

for each neighbor \(v\) of \(s\), add edge \((s, v)\) to \(PQ\) with priority \(l(e)\)

while \(T\) doesn’t have all vertices of \(G\) and \(PQ\) is non-empty

repeat

\[e \leftarrow PQ.\text{removeMin}() \quad \text{// skip edges with both ends in } T\]

until \(PQ\) is empty or \(e=(u, v)\) for \(u \in T, v \notin T\)

if \(e=(u, v)\) for \(u \in T, v \notin T\)

add \(e\) (and \(v\)) to \(T\)

for each neighbor \(w\) of \(v\)

add edge \((v, w)\) to \(PQ\) with weight/key \(d(s, v) + l(v, w)\)
Dijkstra: Space Complexity

- **Graph**: $O(|V| + |E|)$
  - Each vertex and edge uses a constant amount of space
- **Priority Queue**: $O(|E|)$
  - Each edge takes up constant amount of space
- **Map**: $O(|V|)$
- **Result**: $O(|V| + |E|)$
  - Optimal in Big-O sense!
Dijkstra: Time Complexity

Assume Map ops are $O(1)$ time

Across all iterations of outer while loop

- Edges are added to and removed from the priority queue
  - But any edge is added (and removed) at most once!
  - Total PQ operation cost is $O(|E| \log |E|)$ time
    - Which is $O(|E| \log |V|)$ time
  - All other operations take constant time
- Thus time complexity is $O(|E| \log |V|)$
Final Topic: Maps and Hashing
Map Interface

Methods for Map<K, V>

• `int size()` - returns number of entries in map
• `boolean isEmpty()` - true iff there are no entries
• `boolean containsKey(K key)` - true iff key exists in map
• `boolean containsValue(V val)` - true iff val exists at least once in map
• `V get(K key)` - get value associated with key
• `V put(K key, V val)` - insert mapping from key to val, returns value replaced (old value) or null
• `V remove(K key)` - remove mapping from key to val
• `void clear()` - remove all entries from map
Other methods for Map<K,V>: 

• void putAll(Map<K,V> other) - puts all key-value pairs from Map other in map 
• Set<K> keySet() - return set of keys in map 
• Set<Association<K,V>> entrySet() - return set of key-value pairs from map 
• Structure<V> valueSet() - return set of values 
• boolean equals() - used to compare two maps 
• int hashCode() - returns hash code associated with values in map (stay tuned...)
public class Dictionary {

    public static void main(String args[]) {
        Map<String, String> dict = new Hashtable<String, String>();
        ...
        dict.put(word, def);
        ...
        System.out.println("Def: " + dict.get(word));
    }
}

What’s missing from the Map API that a dictionary needs?

    successor(key), predecessor(key)

Maps do NOT preserve order!
Simple Implementation: MapList

- Uses a SinglyLinkedList of Associations as underlying data structure
  - Think back to Lab 2, but a List instead of a Vector
- How would we implement `get(K key)`?
- How would we implement `put(K key, V val)`?
public class MapList<K, V> implements Map<K, V> {

    // instance variable to store all key-value pairs
    SinglyLinkedList<Association<K, V>> data;

    public V put (K key, V value) {
        Association<K, V> temp =
            new Association<K, V> (key, value);
        // Association equals() just compares keys
        Association<K, V> result = data.remove(temp);

        data.addFirst(temp);
        if (result == null)
            return null;
        else
            return result.getValue();
    }
}
Simple Map Implementation

• What is MapList’s running time for:
  • `containsKey(K key)`?
  • `containsValue(V val)`?

• Bottom line: not $O(1)$!
Search/Locate Revisited

- How long does it take to search for objects in Vectors and Lists?
  - $O(n)$ on average
- How about in BSTs?
  - $O(\log n)$
- Can this be improved?
  - Hash tables can locate objects in really quickly!
    - (we will cover two reasons that $O(1)$ performance is a fuzzy claim)
Example from Bailey

“We head to a local appliance store to pick up a new freezer. When we arrive, the clerk asks us for the last two digits of our home telephone number! Only then does the clerk ask for our last name. Armed with that information, the clerk walks directly to a bin in a warehouse of hundreds of appliances and comes back with the freezer in tow.”

• Thoughts?
  • What is Key? What is Value?
  • Are names evenly distributed?
  • Are the last 2 phone digits evenly distributed?
Hashing in a Nutshell

• Assign objects to “bins” based on key
• When searching for object, go directly to appropriate bin (and ignore the rest)
• If there are multiple objects in bin, then search for the correct one

• Important Insight: Hashing works best when objects are evenly distributed among bins
  • Phone numbers are randomly assigned, last names are not (there were a lot of Smiths in Smithsville!)
Implementing a HashTable

• How can we represent bins?
• Slots in array (or Vector, but arrays are faster)
  • Initial size of array is a prime number
• How do we find a key’s bin number?
  • We use a hash function that converts keys into integers
  • In Java, all Objects have public int hashCode()
    • Hashing function is one way: key → fingerprint
    • Hashing function is deterministic
hashCode() rules

The general contract of hashCode is:

• Whenever it is invoked on the same object more than once during an execution of a Java application, the hashCode method must consistently return the same integer, provided no information used in equals comparisons on the object is modified. This integer need not remain consistent from one execution of an application to another execution of the same application.

• If two objects are equal according to the equals(Object) method, then calling the hashCode method on each of the two objects must produce the same integer result.

• It is not required that if two objects are unequal according to the equals(java.lang.Object) method, then calling the hashCode method on each of the two objects must produce distinct integer results. However, the programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hash tables.

https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html#hashCode()
Implementing HashTable

• How do we add Associations to the array?
  • array[o.hashCode() % array.length] = o; ?
    • What’s "aaaaaa".hashCode() ?

• Collisions make life hard

• Two approaches
  • Open Addressing
    • Linear or Quadratic Probing
    • Double Hashing
  • External chaining