E0-245: ASP

Lecture 10: Complexity of Algorithms, Data Structures and Libraries: Part 2

Dipanjan Gope

Module 1: OOPs and DS

JAVA and C++ mainly: (others C# and Objective-C)

- Object oriented programing: Classes and objects
- Inheritance, polymorphism, abstract class, const
- Templates and generics
- **Data structures**
- **Standard library, JCF, STL**
- Complexity analysis
- Multithreading and synchronization
- Good programming styles

ADT Dictionaries

- Associative array/map
- Functions: insert, delete, searchByKey
- Implementation: hash-table (others also possible)

ADT Dictionaries

- Key: e.g. name & Value: e.g. age
- **Hash-function**

  - f(key) = index

  - deterministic index

  - unique index

Hash Code

- Hash code:

  - Key to integer

- Popular hash code:

  - Integer cast, component sum, polynomial accumulation

- Example: Dictionary of 2 letter English words

  - https://www.youtube.com/watch?v=UPo-M8bzRrc Prof. Shewchuk

Hash Code: 26^2

location=26*int(letter1-'a') + int(letter2-'a')

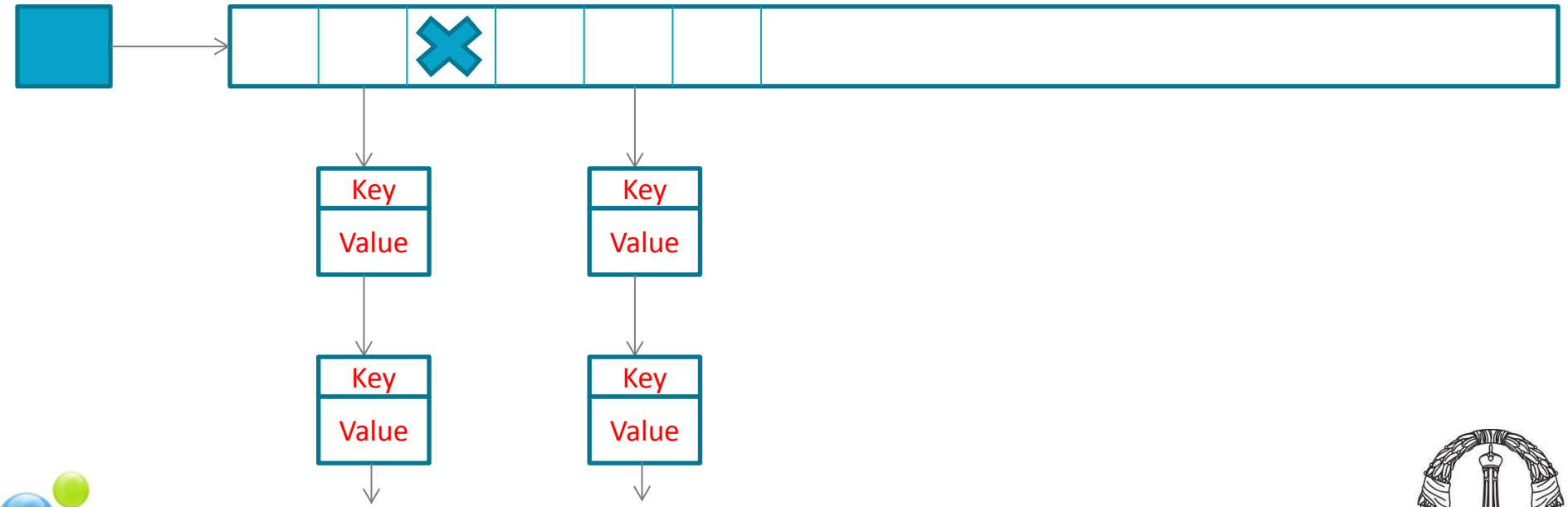
Compression Map

- Integer to Index (0…N-1)
- n = total number of keys
- N = bucket size slightly greater than n
- h(hashcode) = hashcode mod N

- Collision: h(hashcode1) = h(hashcode2)

Collisions

- Avoidance: chaining
- Linked list of (key, value) at an index



Hash Function

- Hash function = Hash Code + Compression Map
- Good hash function reduces the number of collisions

- Load factor = n/N

  - what is the significance of high and low load factors?

Example

- Circuit simulation:

  - netlist to matrix: node-name to node-indices

ADT Trees

- Representation of hierarchical data
- Functions: Traversal
- Implementation: linked-lists and others

Tree

- Nodes + Edges (no loops)
- Only 1 path from one node to another
- Rooted tree: select a node as root

Definitions

- Leaf: Node with 0 children
- Parent: First node in path to root
- Siblings: Node with same parent
- Ancestor: All nodes in path to root (including itself)
- Path: Sequence of connected edges
- Height: path length to root
- Depth: path length to deepest descendant

Type of trees

- k-d tree: 2-d tree = binary tree
- Balanced/unbalanced

Tree Data Structure

```
class Tree
{
        TreeNode* root;
        int height;
}
```

```
class TreeNode
{
        TreeNodeDataType data;
        TreeNode* parent;
        TreeNode* leftChild;
        TreeNode* next-sibling;
}
```

Tree Traversal

- In-order (only binary)

```
public void inorder
{
        if(leftChild) leftChild.inorder();
        this.visit();
        if(rightChild) rightChild.inorder();
}
```

- Pre-order

```
public void preorder
{
        this.visit();
        if(firstChild) firstChild.preoder();
        if(nextSibling) nextSibling.preorder();
}
```

- Post-order

```
public void postorder
{
        if(firstChild) firstChild.preoder();
        this.visit();
        if(nextSibling) nextSibling.preorder();
}
```

Example

- Given a 3D distribution of points, find the closest 100 points to a given point

  - what kind of tree should we use?

ADT Graphs

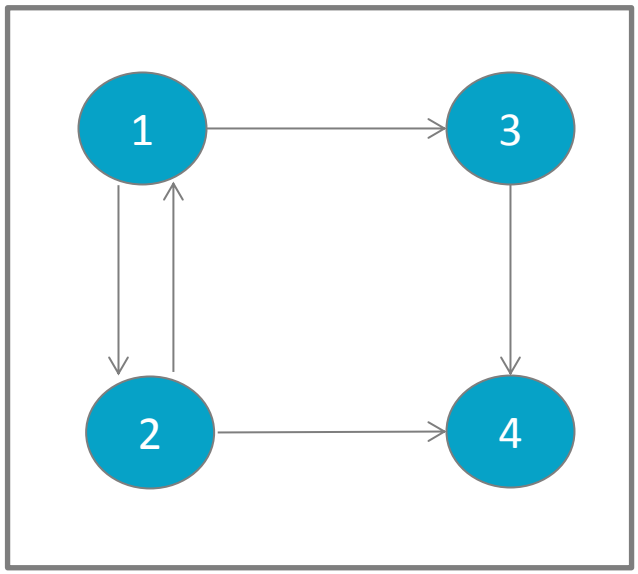
- Set of nodes and edges
- Functions: Traversal
- Implementation: Adjacency matrix or adjacency list

Graph

- Set of vertices/nodes and edges (V,E)

- Types of graphs:
  - directed
  - un-directed

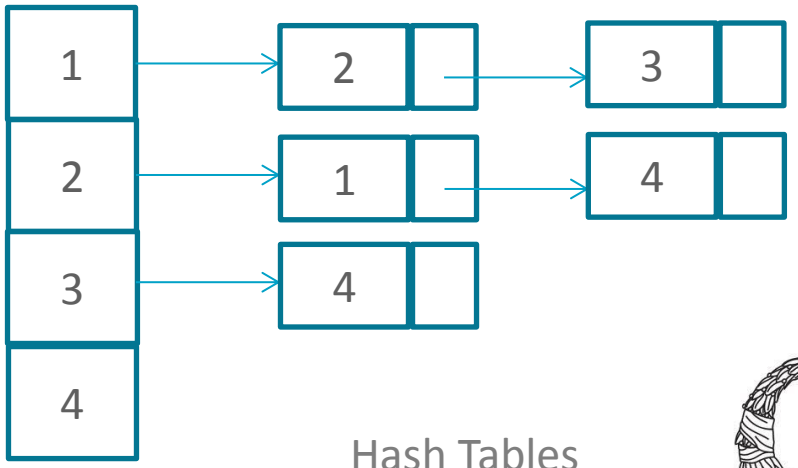
Graph Implementations



Adjacency Matrix

$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Adjacency List



Hash Tables

Definitions

- Degree of a node: number of edges

- In-degree: number of edges towards node
- Out-degree: number of edges away from node

Traversal

- Depth First Search (DFS)
- Breadth First Search (BFS)

```
public void dfs(V node)
{
    node.visited = true;
    for each node w in adjacency list of node
    {
        if(!w.visited)
        {
            w.visited = true;
            dfs(w);
        }
    }
}
```

```
public void bfs(V node)
{
    node.visited = true;
    q = new Queue(node);
    q = enqueue(node);
    while q not empty
    {
        v = q.dequeue();
        for each vertex w in adjacency of v
        {
            if(!w.visited)
            {
                w.visited = true;
                q.enqueue(w);
            }
        }
    }
}
```

Example

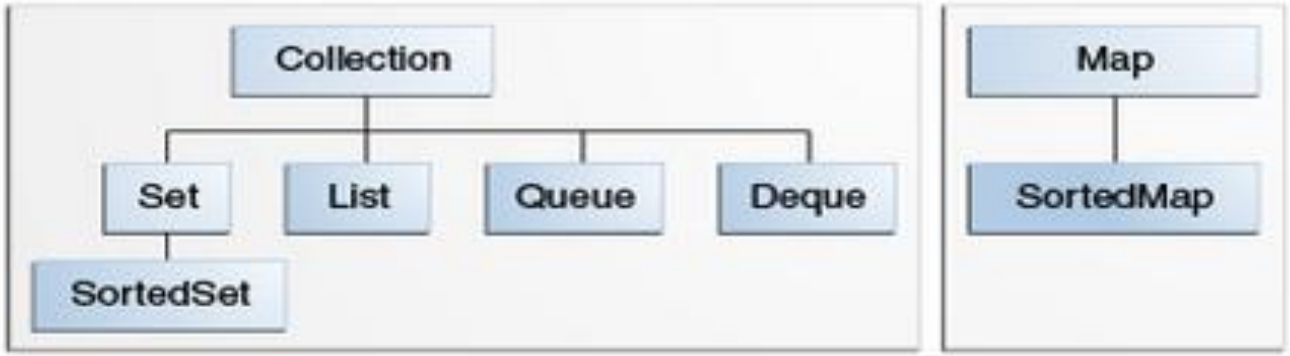
- Find the number of independent loops in a triangular mesh

Libraries in Java, C++

JAVA: http://www.tutorialspoint.com/java/java_collection_algorithms.htm
C++: http://www.tutorialspoint.com/cplusplus/cpp_stl_tutorial.htm

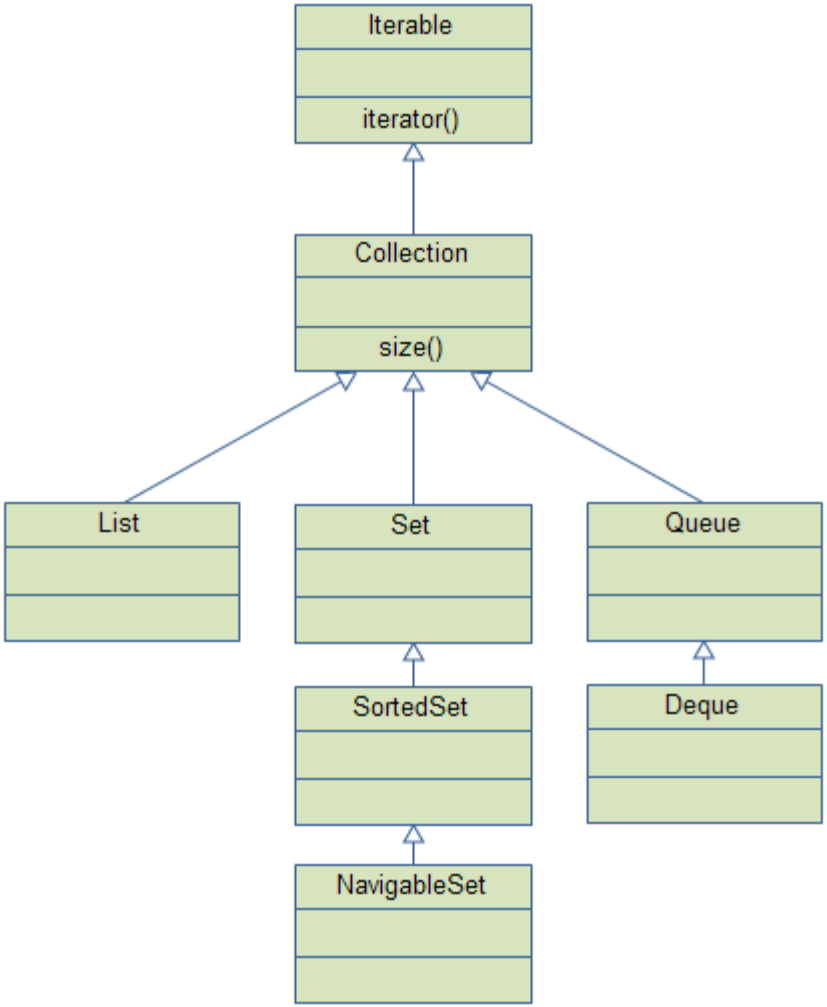
JAVA Collection Framework

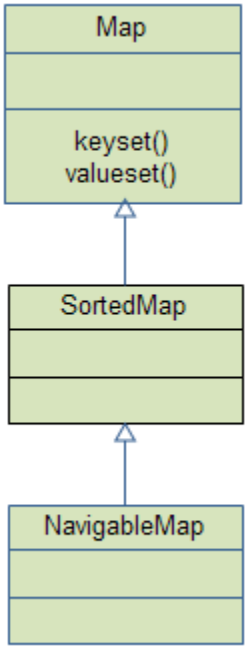
- Interfaces
- Implementations
- Algorithms



http://docs.oracle.com/javase/tutorial/collections/interfaces/index.html

JAVA Collection Framework





http://tutorials.jenkov.com/java-collections/overview.html

Iterators

```
public interface Iterable<T> {
  public Iterator<T> iterator();
}
```

```
List list = new ArrayList();

for(Object o : list){
    //do something o;
}
```

```
LinkedList ll = new LinkedList();
```

```
Iterator li = ll.iterator();
System.out.print("List sorted in reverse: ");
while(li.hasNext()){
   System.out.print(li.next() + " ");
}
```

http://tutorials.jenkov.com/java-collections/iterable.html

Using Generics

```
Collection<String> stringCollection = new HashSet<String>();
```

```
Collection<String> stringCollection = new HashSet<String>();

for(String stringElement : stringCollection) {
  //do something with each stringElement
}
```

http://tutorials.jenkov.com/java-collections/generic-collections.html

Algorithms

```
List list = new ArrayList();

//add elements to the list

Collections.sort(list);
```

```
public interface Comparable<T> {
  int compareTo(T o);
}
```

Example

```
import java.util.*;

public class AlgorithmsDemo {

   public static void main(String args[]) {
      // Create and initialize linked list
      LinkedList ll = new LinkedList();
      ll.add(new Integer(-8));
      ll.add(new Integer(20));
      ll.add(new Integer(-20));
      ll.add(new Integer(8));

      // Create a reverse order comparator
      Comparator r = Collections.reverseOrder();
      // Sort list by using the comparator
      Collections.sort(ll, r);
      // Get iterator
      Iterator li = ll.iterator();
      System.out.print("List sorted in reverse: ");
      while(li.hasNext()){
         System.out.print(li.next() + " ");
      }
      System.out.println();
      Collections.shuffle(ll);
      // display randomized list
      li = ll.iterator();
      System.out.print("List shuffled: ");
      while(li.hasNext()){
         System.out.print(li.next() + " ");
      }
      System.out.println();
      System.out.println("Minimum: " + Collections.min(ll));
      System.out.println("Maximum: " + Collections.max(ll));
   }
}
```

http://www.tutorialspoint.com/java/java_collection_algorithms.htm