13 A: External Algorithms II; Disjoint Sets; Java API Support

CS1102S: Data Structures and Algorithms

Martin Henz

April 15, 2009

1. External Sorting
2. Disjoint Sets
3. Java API Support for Data Structures
4. Puzzlers

1. External Sorting
    - Model for External Sorting
    - The Simple Algorithm
    - Multiway Merge

2. Disjoint Sets

3. Java API Support for Data Structures

4. Puzzlers

Tapes as Storage

Similar to disks

Access time many orders of magnitude slower than main memory

Additional characteristics

Large amounts of data can be read *sequentially* quite efficiently

Access of previous locations

is *extremely* slow, as it requires re-winding the tape!

External Sorting

Main idea

Use tapes sequentially, and read one block from each input tape tape

Merge blocks

Sort the blocks
Use merge procedure from mergesort to merge

The Simple Algorithm: Overview

Four tapes

Two input tapes; two output tapes

Read and write runs

Read runs from input tape, sort them and write alternatively to output tapes

Continue, writing larger runs

Read two runs from each “output” tape, and merge them on the fly, writing alternatively to “input” tapes

Continue

until one tape has all sorted data

Multiway Merge

Why only four tapes?

If we have more than four tapes, we can take advantage of them by using *multiway merge*

How finding the smallest element during merge?

Priority queue!

Each iteration of inner loop

deleteMin to find smallest element
insert new element from tape from which element was deleted

Polyphase Merge and Replacement Selection

Polyphase merge: main idea

Make use of fewer tapes, by re-using tapes for reading and writing
Leading to tape organization using *k*th order Fibonacci numbers

Replacement selection: main idea

Make use of input tape as output tape, reusing the tapes “on the fly”

Equivalence Relations

Definition

An *equivalence relation* is a relation R that satisfies three properties:

1. (Reflexive) aRa, for all $a \in S$.
2. (Symmetric) aRb if and only if bRa.
3. (Transitive) aRb and bRc implies aRc.

Examples

- Electrical connectivity (metal wires between points)
- Cities belonging to same country

The Dynamic Equivalence Problem

Initial setup

Collection of N disjoint sets, each with one element

Operations

- $\mathit{find}(a)$: return the set of which x is element
- $\mathit{union}(a, b)$: merge the sets to which a and b belong, so that $\mathit{find}(a) = \mathit{find}(b)$

Strategies

Fast Find, Slow Union

Use array *repres* to store equivalence class for each element

- $\mathit{find}(a)$: return $\mathit{repres}[a]$
- $\mathit{union}(a, b)$: if $\mathit{repres}[x] = \mathit{repres}[b]$ then set $\mathit{repres}[x]$ to $\mathit{repres}[a]$

Fast Union, Reasonable Find

Union/find data structure

Basic Data Structure

Idea

Maintain *forest* corresponding to equivalence relation

Union

Merge trees

Find

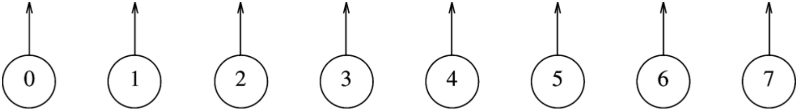
Return root of tree

Observe

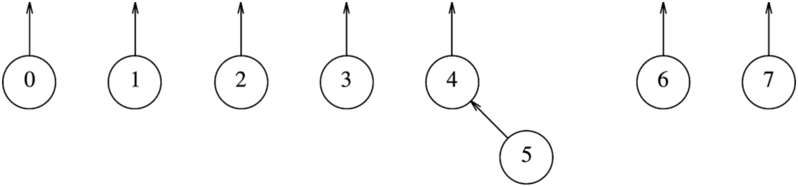
Only upward direction needed!

Example

Initial setup:

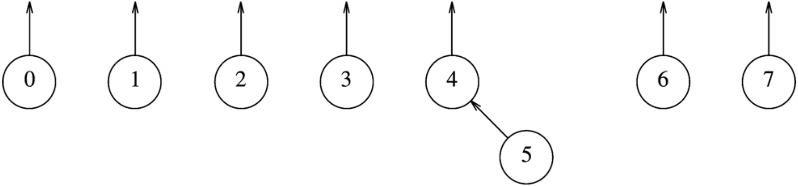


After *union*$(4, 5)$

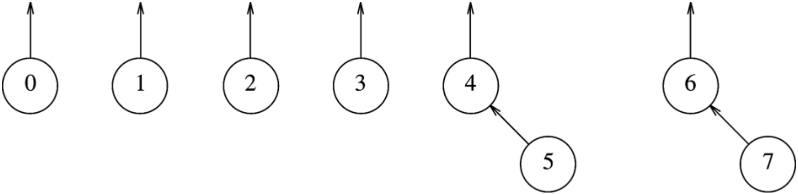


Example

After *union*$(4, 5)$

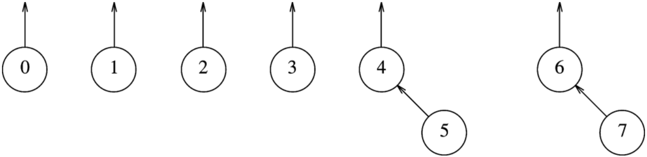


After *union*$(6, 7)$

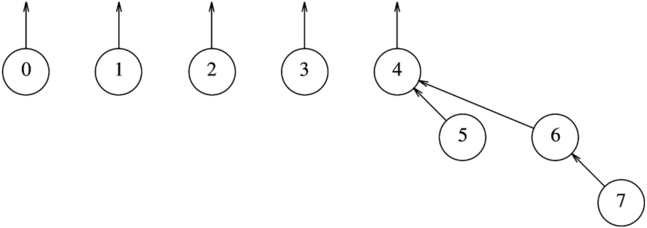


Example

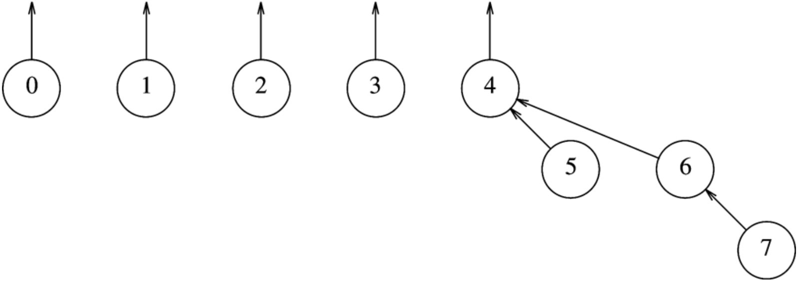
After *union*$(6, 7)$



After *union*$(4, 6)$



Representation



Idea

Remember parent node only; mark root with -1

–1	–1	–1	–1	–1	4	4	6
0	1	2	3	4	5	6	7

Variants

Problem

How to choose root for union? Bad choice can lead to long paths

Union-by-size

Always make the smaller tree a subtree of the larger tree

Analysis

When depth increases, the tree is smaller than the other side. Thus, after union, it is at least twice as large.

Height

less than or equal to $\log N$

Variants

Union-by-height

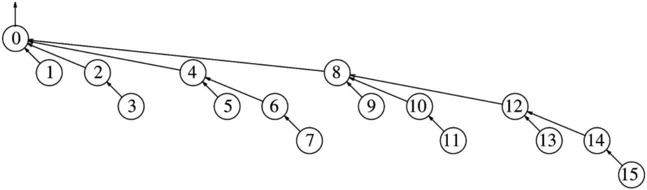
Always make the shorter tree a subtree of the higher tree

Height

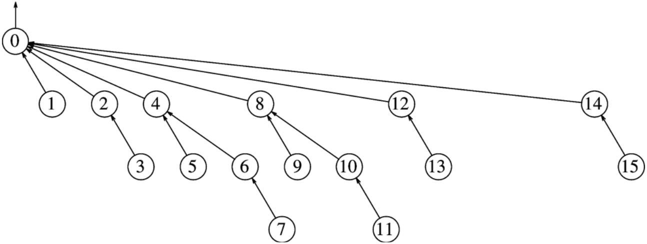
As with union-by-size: $O(\log N)$

Path Compression

During *find* make every node point to root



after *find*(14)



A Very Slowly Growing Function

Definition

$\log^* N$ is the number of times log needs to be applied to N until $N \leq 1$.

Examples

- $\log^* 2 = 1$
- $\log^* 4 = 2$
- $\log^* 16 = 3$
- $\log^* 65536 = 4$
- ...

Runtime

Consider variant

Union-by-height combined with path compression

Theorem

The running time of M *unions* and *finds* is $O(M \log^* N)$.

1. External Sorting
2. Disjoint Sets
3. Java API Support for Data Structures
   - Collections, Lists, Iterators
   - Trees
   - Hashing
   - PriorityQueue
   - Sorting
4. Puzzlers

The Top-level Collection Interface

```
public interface Collection<Any>
        extends Iterable<Any>
{
    int size();
    boolean isEmpty();
    void clear();
    boolean contains(Any x);
    boolean add(Any x);         // sic
    boolean remove(Any x);      // sic
    java.util.Iterator<Any> iterator();
}
```

The List Interface in Collection API

```
public interface List<Any>
       extends Collection<Any>
{
  Any get(int idx);
  Any set(int idx, Any newVal);
  void add(int idx, Any x);
  void remove(int idx);

  ListIterator<Any> listIterator(int pos);
}
```

ArrayList and LinkedList

```
public class ArrayList<Any>
        implements List<Any> {...}
public class LinkedList<Any>
        implements List<Any> {...}
```

Iterators

```
public interface Iterator<Any> {
  boolean hasNext( );
  Any next( );
  void remove( );
}
```

ListIterators

```
public interface ListIterator<Any>
       extends Iterator<Any>
{
  boolean hasPrevious();
  Any previous();
  void add(Any x);
  void set(Any newVal);
}
```

TreeSet

- Implements Collection
- Guarantees $O(\log N)$ time for add, remove and contains

AbstractMap<K,V>

Basic operations

- V get(K key): Returns the value to which the specified key is mapped.
- V put(K key, V value): Associates the specified value with the specified key in this map.

Other operations

containsKey(key), containsValue(val), remove(key)

TreeMap

- Extends AbstractMap
- Guarantees $O(\log N)$ time for put, get, containsKey, containsValue, remove

HashMap

- Extends AbstractMap
- Uses separate chaining with rehashing
- Rehashing is governed by initial capacity and load factor, set in constructor

HashSet

- Implements Collection using HashMap

PriorityQueue

- Implements Collection
- Efficient implementation of heap data structure
- Operation names:
    - deleteMin is called “poll”
    - insert is called “add”

Sorting

- Generic sorting supported by class Collections
- Uses mergesort in order to minimize number of comparisons
- Sorting of built-in numerical types supported by class Arrays
- Uses efficient implementation of quicksort, to take advantage of tight inner loop.

1. External Sorting
2. Disjoint Sets
3. Java API Support for Data Structures
4. **Puzzlers**

Last Puzzler: Package Deal

```
package click;
public class CodeTalk {
  public void doIt() {
    printMessage();
  }
  void printMessage() {
    System.out.println(”Click”);
} }
```

Last Puzzler: Package Deal

```
package hack;
import click.CodeTalk;
public class TypeIt {
  private static class ClickIt extends CodeTalk {
    void printMessage() {
      System.out.println("Hack");
  } }
  public static void main(String[] args) {
    ClickIt clickit = new ClickIt();
    clickit.doIt();
} }
```

What does `clickit.doIt()` print? “Click” or “Hack”?

Java’s Access Modifiers

public : available wherever the class is available
private : only available within the class
protected : available in subclasses and within same package
none : available within the same package

Access Modifiers Govern Inheritance

Overriding only available methods

A method can be overridden only when it is available according to the modifier rules.

Package visibility

Method printMessage can only be overridden within package click.

Result

"Click" is printed.

This Week and Beyond

- Thursday tutorial: Assignment 9
- Friday lecture: CS1102S summary, outlook; questions?
- Next week: Reading week, consultation by appointment
- 27/4 and 28/4: no consultation
- 29/4, 5pm: Final