CSE 326: Data Structures

Topic #11: Disjoint Set ADT (1)

Ashish Sabharwal
Autumn, 2003

Today's Outline

- Admin
  - Project 3 will be out Thursday
    Introduction in tomorrow's section
  - Midterm statistics
    Sample solutions
  - B-Tree clarification
- **Disjoint Set ADT**
  - **Union-Find implementation**

2

Get ready for Project 3!

- Find a partner and send me an email!
  - someone you haven't worked with yet
- Save your work from project 2
  - your team ID may change
  - may not be able to access old shared directory

3

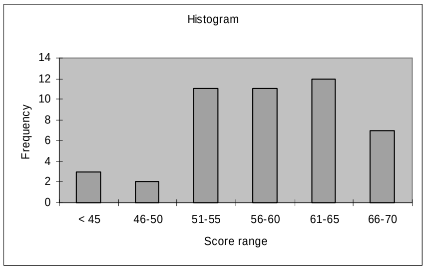
Midterm Statistics

Total points	70
Max	70
Min	31
Average	57.8
Median	57

Good job!

4

Histogram of Midterm Scores



Class size = 46

5

B-Tree Clarification

Homework 2, problem 1, parts c and d

- Sample solutions split internal node differently than what we did in class
- M is # *pointers*, not # keys

6

Our Last Data Structure!

- Implementation of the Disjoint Set ADT
- Uses Union-Find Algorithm

Hmm… so what will we do next?

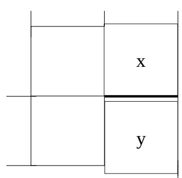
- Sorting algorithms, Graph based algorithms, …
- Use data structures learned to make these efficient!

Motivation: What's a Good Maze?

Maze Construction Algorithm

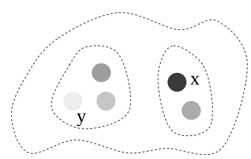
- Given:
  - A collection of rooms **V**
  - Walls/doors between the rooms (initially no doors) **E**
- We want to build a collection of walls to knock down, **E'** ⊆ **E**, such that one unique path connects every two rooms

```
While edges remain in E {
   (x, y) = RemoveRandomWall()
   if( x and y are not
            connected so far ) {
        Add (x, y) to E'
        Mark x and y as connected
   }
}
```

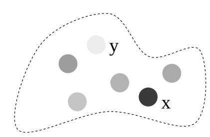


`connected` ≡ have a direct or indirect path

The Problem, Formally



- "If **x** and **y** have not yet been connected"
  - Are two elements in the same set?



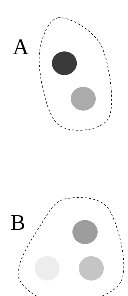
- "Mark **x** and **y** as connected"
  - Form the *union* of two sets

Disjoint Set ADT

<u>Data</u>: elements (no priority, not necessarily comparable)

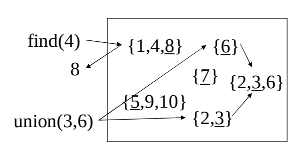
<u>Operations</u>

1. Find(x)
   - Returns set identifier
   - Find(x) = Find(y) iff x and y are in the same set
2. Union(A, B)
   - Arguments are set identifiers
3. MakeNewSet(item)
   - Create a new set containing only *item*



Disjoint Set: Properties

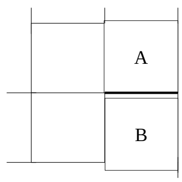
- Equivalence property
  - Every element of a DS belongs to exactly one set
- *Dynamic* equivalence property
  - The set of an element can change after execution of a union



Note: Underlined elements are set IDs

Modified Maze Construction Algorithm

```
While edges remain in E
  (A, B) = RemoveRandomWall()
  if( Find(A) != Find(B) )
     E' = E' U (A, B)
     Union( Find(A), Find(B) )
```



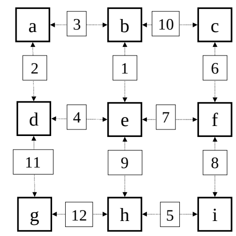
13

Maze Construction Example

a	b	c
d	e	f
g	h	i

Construct this maze!

Initially (the identifier of each set is underlined):
{a}{b}{c}{d}{e}{f}{g}{h}{i}



Order of edges in blue

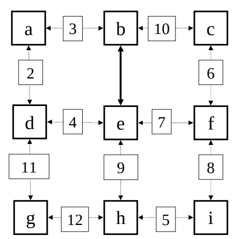
14

Example, continued

{a}{b}{c}{d}{e}{f}{g}{h}{i}

find(b) $\Rightarrow$ b
find(e) $\Rightarrow$ e
find(b) $\neq$ find(e) so:
  add 1 to **E'**
  union(b, e)

Result:



Order of edges in blue

15

Implementing the DS ADT

- *n* elements, *m* finds, ≤ *n*-1 unions

can there be more unions?

- Target complexity: $\Theta(m+n)$
  i.e. $\Theta(1)$ amortized

- $\Theta(1)$ worst-case for find as well as union would be great, but…

  Known result: both find and union *cannot* be done in worst-case $\Theta(1)$ time

16

Attempt #1

- Hash elements to a hashtable
- Store set identifier for each element as data

runtime for find:

runtime for union:

runtime for m finds, n-1 unions:

17

Attempt #2

- Hash elements to a hashtable
- Store set identifier for each element as data
- *Link* all elements in the same set together

runtime for find:

runtime for union:

runtime for m finds, n-1 unions:

18

Attempt #3

- Hash elements to a hashtable
- Store set identifier for each element as data
- *Link* all elements in the same set together
- Always update identifiers of *smaller* set

runtime for find:

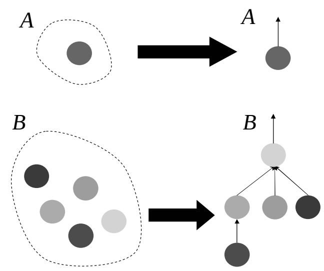
runtime for union:

runtime for m finds, n-1 unions:

[Read section 8.2]

19

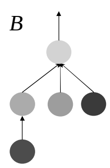
DS ADT Tree Representation



- Maintain a forest of up-trees
- Each set is a tree
- What's a natural set identifier?

20

Find Implementation

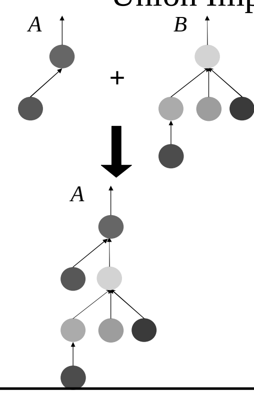


Find(*x*)
- Traverse parents of *x* to the root

Runtime:

21

Union Implementation

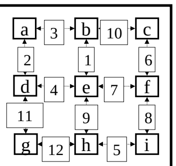


Union(*A*, *B*)
- Join the two trees
- Since *A* and *B* are already the roots of a tree, this is easy!

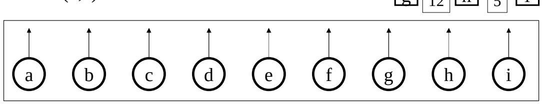
Runtime:

22

More of the Example



union(b,e)

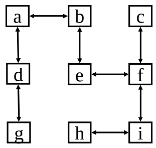


23

(extra space)

24

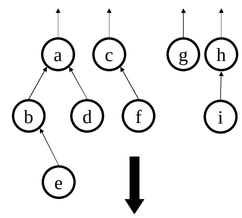
The Final Maze



Ooh… scary!
Such a hard maze!

Nifty storage trick

- A forest of up-trees can easily be stored in an array
- Use *hashtable* to map node names to array indices



	0 (a)	1 (b)	2 (c)	3 (d)	4 (e)	5 (f)	6 (g)	7 (h)	8 (i)
up-index:	-1	0	-1	0	1	2	-1	-1	7

Implementation

```
int Find(Object x) {
  int xID = hTable[x];

  while(up[xID] != -1) {
    xID = up[xID];
  }

  return xID;
}
```

```
void Union(int x, int y) {
  up[y] = x;
}
```

runtime for Union():

runtime for Find():

runtime for m Finds and n-1 Unions:

Now this doesn't look good L

Can we do better? *Yes!*

1. Improve union so that *find* only takes $\Theta(\log n)$
   - Union-by-size
   - Reduces complexity to $\Theta(m \log n + n)$
2. Improve find so that it becomes even better!
   - Path compression
   - Reduces complexity to almost $\Theta(m + n)$

To Do

- Find partner for Project 3
  – Send me email
- Read Chapter 8