Algorithms

Dario L

July 20, 2014

Preface

Mathematics possesses not only truth, but supreme beauty,a beauty cold and austere, like that of a sculpture, and capable of stern perfection, such as only great art can show.
–Bertrand Russell

Contents

1
Introduction

Steps for developing a usable algorithm

- Model the Problem
- Find an algorithm to solve it.
- Fast Enough? Fits in memory?
- If not, figure out why.
- Find a way to address the problem.
- Iterate until satisfied

The scientific method

Mathematical analysis

2 Union-Find

2.1 Dynamic Connectivity

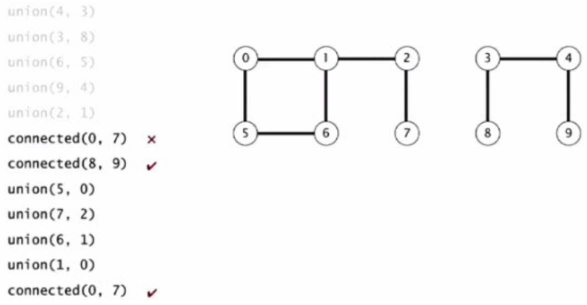
2.1.1 Applications involve manipulating objects of all types

- Pixels in a digital photo
- Computers in a network
- Friends in a social network.
- Transistors in a computer chip
- Variable name in Fortran program
- Metallic sites in a composite system

Given a set of N objects

Union command: connect two objects

Find/connected query: is there a path connecting the two objects?



```
union(4, 3)
union(3, 8)
union(6, 5)
union(9, 4)
union(2, 1)
connected(0, 7)   ×
connected(8, 9)   ✓
union(5, 0)
union(7, 2)
union(6, 1)
union(1, 0)
connected(0, 7)   ✓
```

2.1.2 Implementing the operations

Find Query Check if two objects are in the same component.

Union Command Replace components containing two objects with their union.

For example if you have [0][1 4 5][2 3 6 7] where each [X...X] represents the connectec components if you use the operation **union(2,5)** you will have [0][1 2 3 4 5 6 7]

2.1.3 Union-find data type (API)

Goal Design efficient data structure for union-find.

- Number of objects N can be huge.
- Number of operations M can be huge.
- Find queries and union commands may be intermixed.

Public Class UF

UF(int N)	initialize union-find data structure with N objects(0 to N-1)
void union(int p, int q)	add connection between p and q
boolean connected(int p, int q)	are p and q in the same component ?
int find(int p)	component identifier for p(0 to N-1)
int count()	number of components

2.2 Quick Find

Data Structure

- Integer array id[] of size N.
- Interpretation: p and q are connected if they have the same id.

Find Check if p and q have the same id. **Union** to merge componenets containing p and q, change all entries whose id equals to id[p] to id[q]

```
public class QuickFind
{
        private int[] id;

        public QuickFind(int N)
        {
                id= new int[N];
                for ( int i = 0; i < N; i++)
                        id[i] = i;
        }

        public boolean find(int p, int q)
        {
                return id[p] == id[q];
        }

        public void unite(int p, int q)
        {
                int pid = id[p];
                for (int i=0; i< id.length; i++)
                        if (id[i] == pid) id[i] = id[q];
        }
}
```

Cost Model Number of array acesses for read or write.

Algorithm	initialize	union	find
Quick-Find	N	N	1

ex. takes N^2 array acesses to process a sequence of N union commands on N objects.

2.3 Quick Union

```
public class QuickUnionUF
{
  private int[] id;

  public QuickUnionUF(int N)
  {
    id = new int[N];
    for (int i = 0; i < N; i++) id[i] = i;
  }

  private int root(int i)
  {
    while(i != id[i]) i = id[i];
    return i;
  }

  public bolean connected(int p, int q)
  {
    return root(p) == root(q);
  }

  public void union(int p, int q)
  {
    int i = root(p);
    int j = root(q);
    int[i] = j;
  }
}
```

Algorithm	initialize	union	find
Quick-Find	N	N	1
Quick-Union	N	N	N

Quick-Find defect

- Union too expensive (N array acesses).
- Trees are flat, but too expensive to keep them flat.

Quick-Union defect

- Trees can get tall.
- Find too expensive (could be N array accesses).

2.3.1 improvements