CS3 Theory Task 1

Anthony Cade Streich

August 29, 2024

Question 1.15

Give a sequence of input pairs that causes the weighted quick-union algorithm with path compression by halving to produce a path of length 4.

Begin with a sequence of unique integers from the range of [0,7]. Execute the following union commands to build an algorithm with a path length of 4:

- 1. union(0,1);
- 2. union(2,3);
- 3. union(4,5);
- 4. union(6,7);
- 5. union(1,2); // This union joins sequences (0,1) and (2,3).
- 6. union(5,6); // This union joins sequences (4,5) and (6,7).
- 7. union(0,4); // This union joins the two trees by unionizing (0,1)-(2,3) on "top" of (4,5)-(6,7), resulting in a path length of 4.

Question 1.22

Modify Program 1.4 (Weighted Quick Union with Path Compression by Halving) to generate random pairs of integers between 0 and N-1 instead of reading them from standard input, and to loop until N-1 union operations have been performed. Run your program for N = 10^3, 10^4, 10^5, and 10^6 and print out the total number of edges generated for each value of N.

Code

```
#include <iostream>
#include <random>

#define N 1000000

int main(int argc, char *argv[ ])
{
    int i, j, p, q, id[N], size[N];
    std::default_random_engine randomNumGenerator;

    for (i = 0; i < N; i++) // Initializing each array.
    {
        id[i] = i;
        size[i] = 1;
    }

    for (int z = 0; z < (N - 1); z++)
    {
        p = (randomNumGenerator() % N); // Assigning p a random value.
        q = (randomNumGenerator() % N); // Assigning q a random value.

        for (i = p; i != id[i]; i = id[i])
        {
            id[i] = id[id[i]]; // Halves the length of the path to root.
        }
        for (j = q; j != id[q]; j = id[q])
        {
            id[j] = id[id[j]];
        }

        if (i == j) {continue;}

        if (size[i] < size[j])
        {
            id[i] = j;
            size[j] += size[i];
        } else {
```

```
            id[j] = i;
            size[j] += size[i];
        }

        printf(" %d %d\n", p, q);
    }

    int* resultArray = new int[N]; // Must be declared on heap. Program will result in stack overflow when element size approaches 10^6.

    int resultArraySize = 0;

    for (int x = 0; x < N; x++)
    {
        bool hasGroup = false; // Needed to track if ea individual element belongs to a tree/group.

        for (int y = 0; y < resultArraySize; y++) // Looping through each pre-existing group.
        {
            if (id[x] == resultArray[y]) // Comparing current group with pre-existing groups.
            {
                hasGroup = true;
                break;
            }
         }

        if (hasGroup == true) {break;}

        resultArray[resultArraySize] = id[x];
        resultArraySize++;
    }

    std::cout << "\nNumber of edges: << (N - resultArraySize) << std::endl;
    delete[ ] resultArray;
// End of program.
}
```

Results

- 10^3 : 993 edges.
- 10^4 : 9999 edges.
- 10^5 : 99998 edges.
- 10^6 : 999998 edges.

Question 1.21

Show that Property 1.3 (Weighted Quick Union M lg N edge processing) holds for the algorithm described in Exercise 1.20 (Weighted Quick Union).

Exercise 1.20 - Quick Union Solution

Modify Program 1.3 (Weighted Quick Union) to use the height of the trees (longest path from any node to the root), instead of the weight, to decide whether to set (id[i] = j) or (id[j] = i). Run empirical studies to compare this variant with Program 1.3.

Program 1.3 (Modified Weighted Quick Union)

// This program has been modified to generate random union commands so that it is easier to empirically compare different algorithms.

```
#include <iostream>
#include <random>

#define N 1000000
```