Disjoint Set Representation

Andres Mendez-Vazquez

March 11, 2018

Contents

1 Introduction

In the areas of

- Maintaining partitions and equivalence classes.
- Graph connectivity under edge insertion.
- Minimum Spanning Tree
- Random maze construction.
- Connected components in a graph
- Etc.

We require to have an excellent representation of disjoint sets $\{S_1, S_2, ..., S_k\}$. For this, we need to support the operations of

- MakeSet(x) which makes a new set with only one member x.
- Union(x, y) as the names says it unites two sets which contain x and y.
- Find-Set(x) which returns a pointer to the set representative.

2 Example of Using the Operations

The following pieces of pseudo code are used to find the connected components elements, and they exemplify quite well the need for a disjoint set support.

Algorithm 1 Connected Components

Connected-Components(G)

1. for each vertex $v \in G.V$
2. Make-Set

```
CONNECTED-COMPONENTS(G)
1  for each vertex v ∈ G.V
2      MAKE-SET(v)
3  for each edge (u, v) ∈ G.E
4      if FIND-SET(u) ≠ FIND-SET(v)
5          UNION(u, v)
```

```
SAME-COMPONENT(u, v)
1  if FIND-SET(u) == FIND-SET(v)
2      return TRUE
3  else return FALSE
```

3 Representations

We have the following possible representations for a series elements $1, 2, ..., n$:

1. Using two arrays of size n:

(a) A set representative Set[x] that returns the representative.

(b) A set next next[x] that returns the next element in the set.

2. Weighted Lists - The two previous two arrays.

   (a) We add an extra array size[x] that returns the # items in the set.

Under these representations (Look at the slides for the implementations of each of the methods under the data structures), we have the following aggregated costs (Looks at the slides for the proofs):

1. Aggregate time $\Theta\left(n^2\right)$ then per operation is $\Theta\left(n\right)$.
2. Aggregate cost $O\left(m + n\log n\right)$ then per operation is $O\left(\log n\right)$.

We can do better through the use of a Forest of Up-Trees

- A data structure using a parent array such that p[x] returns the set representative.

Here, we go back to the worst aggregated analysis of the first representation. However, we can use an strategy based in self-adjusting

- Balanced Unions
- Path Compressions

The rest of the class is at the slides.

4 Theorems

Theorem. *21.1 Using the linked-list representation of disjoint sets and the weighted-Union heuristic, a sequence of m MakeSet, Union, and FindSet operations, n of which are MakeSet operations, takes $O\left(m + n\log n\right)$ time.*

Proof Because each Union operation unites two disjoint sets, we perform at most $n-1$ Union operations over all. We now bound the total time taken by these Union operations. We start by determining, for each object, an upper bound on the number of times the object's pointer back to its set object is updated. Consider a particular object x. We know that each time x's pointer was updated, x must have started in the smaller set. The first time x's pointer was updated, therefore, the resulting set must have had at least 2 members. Similarly, the next time x's pointer was updated, the resulting set must have had at least 4 members. Continuing on, we observe that for any $k \leq n$, after x's pointer has been updated $\lceil \log n \rceil$ times, the resulting set must have at least k members. Since the largest set has at most n members, each object's pointer is updated at most $\lceil \log n \rceil$ times over all the Union operations. Thus, the total time spent updating object pointers over all Union operations is $O\left(n\log n\right)$. We must also account

for updating the tail pointers and the list lengths, which take only $O(1)$ time per Union operation. The total time spent in all Union operations is thus $O(n \log n)$.

The time for the entire sequence of m operations follows easily. Each MakeSet and FindSet operation takes $O(1)$ time, and there are $O(m)$ of them. The total time for the entire sequence is thus $O(m + n \log n)$.