A $O(n)$ UNT ALGORITHM FOR THE UNION-FIND (USTCONN) PROBLEM

Oscar Riveros

March 6, 2017

oscar.riveros@peqnp.com
http://twitter.com/maxtuno
http://klout.com/maxtuno
http://independent.academia.edu/oarr
http://github.com/maxtuno/On_UNT_Union_Find_Algorithm

1 THE UNION-FIND PROBLEM

The general setting for the union-find problem is that we are maintaining a collection of disjoint sets $\{S_0, S_1 \dots S_{k-1}\}$ over some universe, with the following operations:

$\texttt{union}(\texttt{x}, \texttt{y})$: replace the set $\{x\}$ is in (let's call it S) and the set $\{y\}$ is in (let's call it S') with the single set $S \bigcup S'$.

$\texttt{connected}(\texttt{x}, \texttt{y})$: test if exist a subset S with x and $y \in S$.

2 DATA REPRESENTATION

Suppose there is an arbitrary but finite number of elements $\{x_0, x_1 \dots x_{k-1}\}$ then can represent the entire structure with, $\{1, 2, 4, \dots, 2^{(k-1)}\}$ and the subsets how $\{x_{i_0}, x_{i_1} \dots x_{i_l}\} = \sum_{i \in \{i_0, \dots, i_l\}} 2^i$.

Example 1. Let be $U = \{P, NP, =, \neq\}$ then the subset $\{P, =, NP\}$ is writting like $2^0 + 2^1 + 2^3 = 11$, and $\{P, \neq, NP\}$is writting like $2^0 + 2^1 + 2^4 = 19$.

3 ALGORITHM DESIGN & IMPLEMENTATION

With all generality can only work with integers forget the particular given set, then implementation is as follows:

3.0.1 PYTHON IMPLEMENTATION

```
class USTCONN:
    def __init__(self):
        self.universe = []

    def __str__(self):
        paths = ''
        for item in self.universe:
            path = self.nary(item)
            if path:
                paths += '{}, '.format(path)
        return paths

    def nary(self, n):
        s = ''
        while n:
            s += str(n % 2)
            n //= 2
        return [idx for idx in range(len(s)) if s[idx] == '1']

    def union(self, a, b):
        element = (1 << a) | (1 << b)
        idx = 0
        while idx != len(self.universe):
            if self.universe[idx] & element:
                element |= self.universe[idx]
                del self.universe[idx]
                idx -= 1
            idx += 1
        self.universe.append(element)

    def connected(self, a, b):
        for item in self.universe:
            if (item & (1 << a)) and (item & (1 << b)):
                return item
        return 0

    def make_set(self, a):
        self.union(a, a)

    def find(self, a):
        return self.connected(a, a)
```

4 EXPLANATION

4.1 UNION

1. There is the simple list for storage the elements and fusion them and delete

   ```
   universe = []
   ```

2. The definition of function, this receives two arguments that are the indices of the elements of the original set.

   ```
   def union(a, b):
   ```

3. This is equivalent to $2^a + 2^b$ but it is more efficient for that is a bit shifing and not an exponencicion

   ```
   element = (1 << a) | (1 << b)
   ```

4. It is to merge elements if necessary, the loop is executed $|U|$ times, that is equal to number of subsets or paths.

   ```
   idx = 0
   while idx != len(self.universe):
   ```

5. If there are elements in common the merges and eliminates the excess and find a next if not exit the loop, this operation is executed each time.

   ```
   if self.universe[idx] & element:
       element |= self.universe[idx]
       del self.universe[idx]
       idx -= 1
   idx += 1
   ```

6. Finally append the new element.

   ```
   universe[len(universe)] = element
   ```

4.2 CONNECTED

The explanation its the same, the diference is when it finds a match, it returns the subset, if not it returns 0 ($\emptyset$).

5 COMPLEXITY

Suppose the base set has n elements, the minimal amount of information for representing this on a Turing Machine (binary) its the sequence:

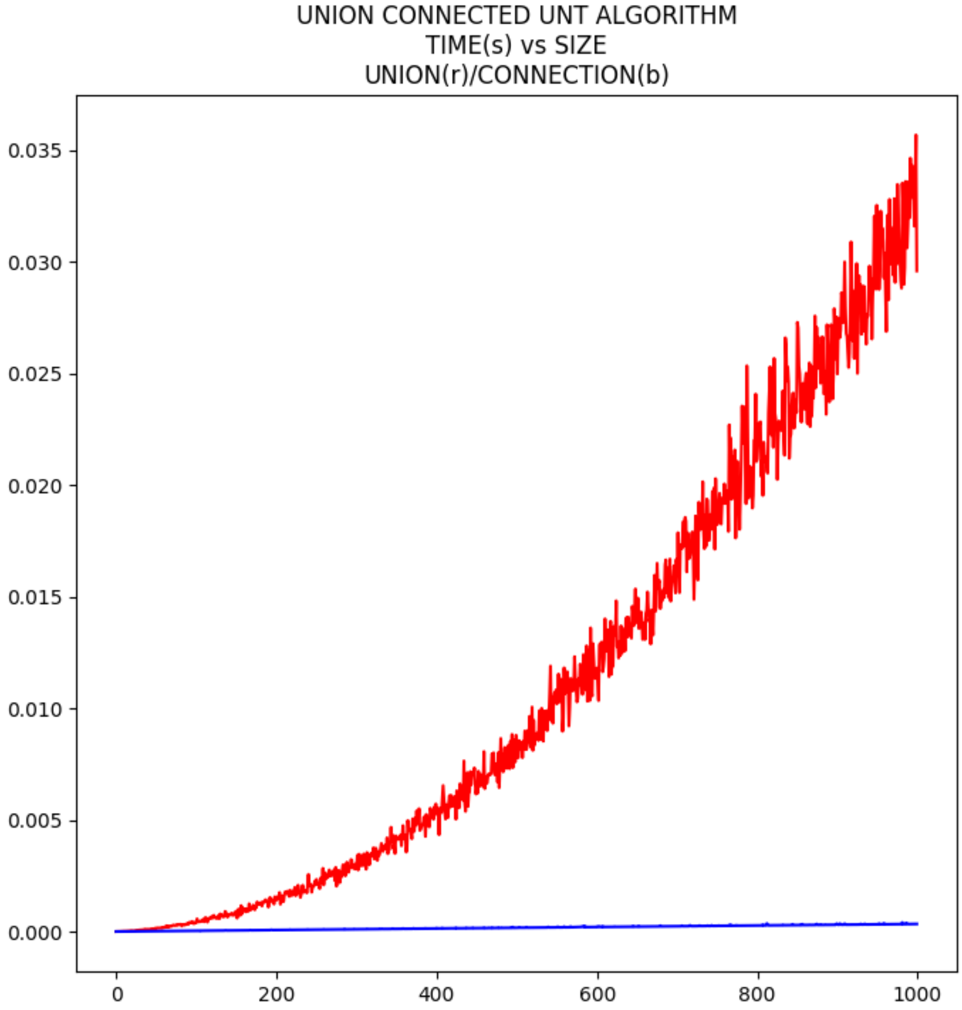
$$\texttt{tape} = \{\underbrace{00\ldots001}_{\texttt{0}}\_\underbrace{00\ldots010}_{\texttt{1}}\_\underbrace{00\ldots010}_{\texttt{2}}\_\cdots\_\underbrace{10\ldots000}_{\texttt{n}}\_\}$$

then on the working tape, on the a-th and b-th bit can be on n steps. with the following turing machine.

```
Turing Machine:
OR GATE (010, 101)
tape: 010101
1)010[1]01
2)01[1]_01 (0,1=1)
3)011_[0]1
4)0[1]1__1 (0,1=1)
5)011__[1]
6)[1]11___ (0,1=1)
=> O(n)
```

For the & gate its the same, and for representing sets, not needed do (1<<a) | (1<<b) if already take for n (3) 001, 010, 100 = 1,2,4, then input tape is for m elements: $\texttt{tape} = \{\underbrace{00\ldots001}_{\texttt{0}}\_\underbrace{00\ldots010}_{\texttt{1}}\_\underbrace{00\ldots010}_{\texttt{2}}\_\cdots\_\underbrace{10\ldots000}_{\texttt{n}}\_\} = n + n^2$ this take n for OR + n times AND = $n + n^2$, this is $O(n)$ (same of input size).

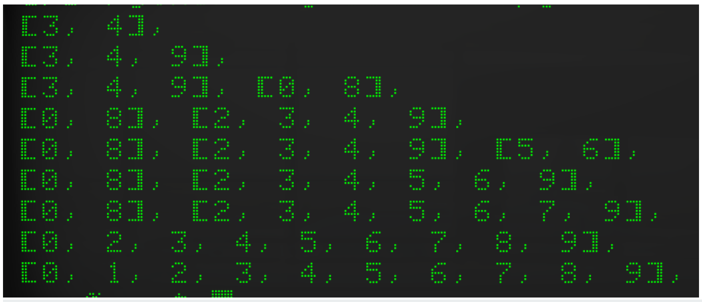
6 RUNNING TIME

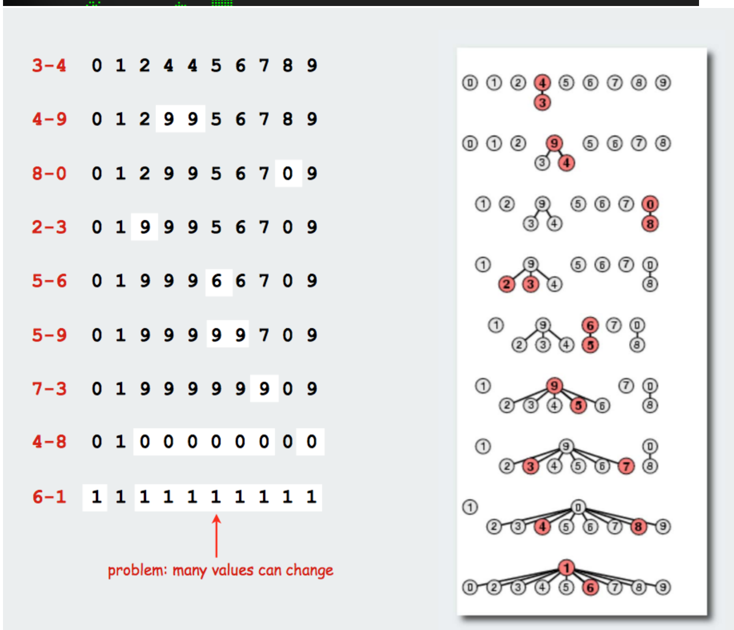


7 EXAMPLE FROM STANDARD PUBLICATIONS

```
if __name__ == '__main__':
    ustconn = USTCONN()

    ustconn.union(3, 4)
    print(ustconn)
    ustconn.union(4, 9)
    print(ustconn)
    ustconn.union(8, 0)
    print(ustconn)
    ustconn.union(2, 3)
    print(ustconn)
    ustconn.union(5, 6)
    print(ustconn)
    ustconn.union(5, 9)
    print(ustconn)
    ustconn.union(7, 3)
    print(ustconn)
    ustconn.union(4, 8)
    print(ustconn)
    ustconn.union(6, 1)
    print(ustconn)
```





8 DISCLAMER

All content is original and of my authorship, there are no partners or affiliations, only used basic concepts of the theory of computational complexity, and standard definitions, giving references is impossible, are part of the general culture. the UNT is acronim for the UNIVERSAL NUMBER THEORY, look at my other papers.

9 LICENCE