Algorithmn HW1

5140379032 JIN YI FAN

$1 \prec \log\log n \prec logn \prec \sqrt{n} \prec n^{3/4} \prec n \prec n\log n \prec n^2 \prec 2^n \prec n! \prec 2^{n^2}$

Problem 1: Union-find with specific canonical element

- Use Weighted-Union instead of the naive Union-Find;
- Add a new array $largest[]$ to store the biggest element included in the components containing the target element;
- Modify the value in $largest[]$ when unioning two trees with roots a and b:

```
1: aL ← largest[a]
2: bL ← largest[b]
3: if aL > bL then
4:     largest[b] ← aL
5: else
6:     largest[a] ← bL
7: end if
```

- $find(i)$ function simply return the element $largest[i]$.

Problem 2: Successor with delete

- $remove(x)$:
  We can add a new array $exist[]$ to store if the element is still in.

```
1: exist[x] ← false
2: if x − 1 is removed then
3:     union(x, x − 1)
4: end if
5: if x + 1 is removed then
6:     union(x, x + 1)
7: end if
```

- $successor(x)$:
  We can use the $find()$ function in Problem 1.

```
1: if exist[x] is true then
2:     return x
3: else
```

4: **return** $find(x) + 1$

5: **end if**

Problem 3: Union-by-height

Inplementation:

```
#include<vector>
using namespace std;
class HeightUnion{
        vector<int>id;
        vector<int>height;
public:
        HeightUnion(int N);
        int root(int p);
        void Union(int p, int q);
        bool Connected(int p, int q);
};

HeightUnion::HeightUnion(int N){
        for (int i = 0; i < N; i++){
                id.push_back(i);
                height.push_back(0);
        }
}
int HeightUnion::root(int p){
        int pPar = id[p];
        while (p != pPar){
                pPar = id[pPar];
                p = pPar;
        }
        return p;
}
void HeightUnion::Union(int p, int q){
        int rp = root(p);
        int rq = root(q);
        if (rp==rq) return;
        if (height[rp] > height[rq]) {
                id[rq] = rp;
        }
        else{
                id[rp] = rq;
                if (height[rp] == height[rq]) height[rq]++;
        }
}
bool HeightUnion::Connected(int p, int q){
        return root(p) == root(q);
}
```

Proof:

As we can see, the height of the tree unioned will change iff these two offspring trees are equally high and height will +1 in this case, otherwise it will remain the same.

So $Height(N) \leq$ (worst case)the height is updated at every *union* operation, that is, these N nodes have done $Height(N)$ symmetrical 2-to-1 merging to 1 nodes, which is obviously $\log_2(N)$.