CS 225

#31: Disjoint Sets

March 29, 2021 · *G Carl Evans*

Disjoint Sets

Let **R** be an equivalence relation. We represent R as disjoint sets

- Each element exists in exactly one set.
- Every set is an equitant representation.
  - Mathematically: 4 ∈ $[0]_R$ → 8 ∈ $[0]_R$
  - Programmatically: find(4) == find(8)

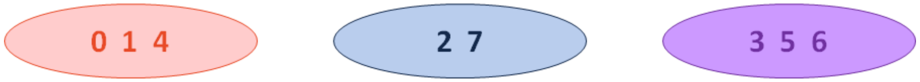
Building Disjoint Sets:

- Maintain a collection S = {s_0, s_1, ... s_k}
- Each set has a representative member

  void makeSet(const T & t);

  void union(const T & k1, const T & k2);

  T & find(const T & k);



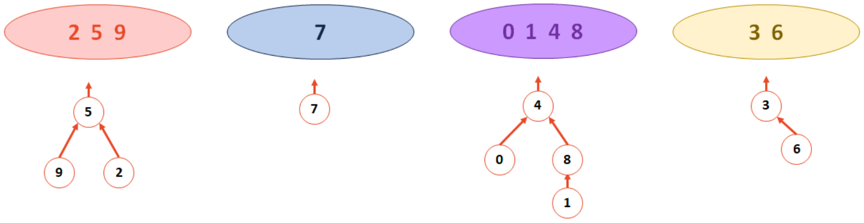
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Operation: find(k)

Operation: union(k1, k2)

Implementation #2:

- Continue to use an array where the index is the key
- The value of the array is:
  - **-1**, if we have found the representative element
  - **The index of the parent**, if we haven't found the rep. element



4	8	5	-1	-1	-1	3	-1	4	5
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Implementation – DisjointSets::find

DisjointSets.cpp (partial)

```
int DisjointSets::find(int i) {
  if ( s[i] < 0 ) { return i; }
  else { return _find( s[i] ); }
}
```

What is the running time of **find**?

What is the ideal UpTree?

Implementation – DisjointSets::union

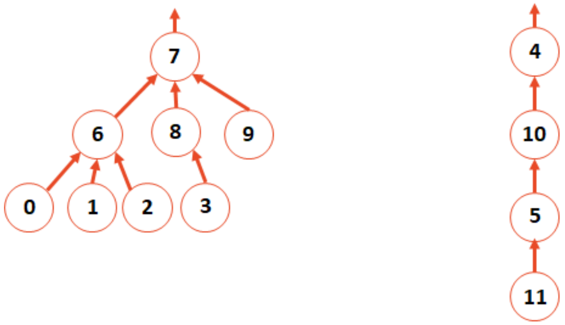
DisjointSets.cpp (partial)

```
void DisjointSets::union(int r1, int r2) {


}
```

How do we want to union the two UpTrees?

Building a Smart Union Function



The implementation of this visual model is the following:

6	6	6	8	-1	10	7	-1	7	7	4	5
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]

What are possible strategies to employ when building a "smart union"?

Smart Union Strategy #1: ________________
Idea: Keep the height of the tree as small as possible!

Metadata at Root:

After `union( 4, 7 )`:

6	**6**	**6**	**8**		**10**	**7**		**7**	**7**	**4**	**5**
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]

Smart Union Strategy #2: ________________
Idea: Minimize the number of nodes that increase in height.
(Observe that the tree we union have all their nodes gain in height.)

Metadata at Root:

After `union( 4, 7 )`:

6	**6**	**6**	**8**		**10**	**7**		**7**	**7**	**4**	**5**
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]

Smart Union Implementation:

`DisjointSets.cpp (partial)`

```
void DisjointSets::unionBySize(int root1, int root2) {
  int newSize = arr_[root1] + arr_[root2];

  if ( arr_[root1] < arr_[root2] ) {
    arr_[root2] = root1;  arr_[root1] = newSize;
  } else {
    arr_[root1] = root2;  arr_[root2] = newSize;
  }
}
```

How do we improve this?

`DisjointSets.cpp (partial)`

```
int DisjointSets::find(int i) {
  if ( arr_[i] < 0 ) { return i; }
  else { return _find( arr_[i] ); }
}
```

`DisjointSets.cpp (partial)`

```
void DisjointSets::unionBySize(int root1, int root2) {
  int newSize = arr_[root1] + arr_[root2];

  // If arr_[root1] is less than (more negative), it is the
  // larger set; we union the smaller set, root2, with root1.
  if ( arr_[root1] < arr_[root2] ) {
    arr_[root2] = root1;
    arr_[root1] = newSize;
  }
  // Otherwise, do the opposite:
  else {
    arr_[root1] = root2;
    arr_[root2] = newSize;
  }
}
```

Running Time:

- Worst case running time of find(k)**:**
- Worst case running time of union(r1, r2), given roots:
- New function: "Iterated Log":
  log*(n) :=
- Overall running time:
  - A total of **m** union/find operation runs in:

CS 225 – Things To Be Doing:

1. mp_mosaics due today.
2. Exam on Friday practice on PrairieLearn now
3. Daily POTDs are ongoing!