CS213/293 Data Structure and Algorithms 2025

Lecture 9: UnionFind for disjoint sets

Instructor: Ashutosh Gupta

IITB India

Compile date: 2025-09-14

Topic 9.1

Disjoint sets

Example: equality reasoning

Example 9.1

Is the following equality constraints satisfiable?

$t_1 = t_8 \wedge t_7 = t_2 \wedge t_7 = t_1 \wedge t_6 = t_7 \wedge t_9 = t_3 \wedge t_5 = t_4 \wedge t_4 = t_3 \wedge t_7 = t_5 \wedge t_1 \neq t_4$

In equality reasoning, we maintain the sets of equivalent terms.

After processing $t_1 = t_8$, we have $\{t_1, t_8\}$.
After processing $t_7 = t_2$, we have $\{t_1, t_8\}$ and $\{t_2, t_7\}$.
After processing $t_7 = t_1$, we have $\{t_1, t_2, t_7, t_8\}$.
After processing $t_6 = t_7$, we have $\{t_1, t_2, t_6, t_7, t_8\}$.
After processing $t_3 = t_9$, we have $\{t_1, t_2, t_6, t_7, t_8\}$ and $\{t_3, t_9\}$.
After processing $t_5 = t_4$, we have $\{t_1, t_2, t_6, t_7, t_8\}, \{t_3, t_9\}$, and $\{t_4, t_5\}$.
After processing $t_4 = t_3$, we have $\{t_1, t_2, t_6, t_7, t_8\}$ and $\{t_3, t_4, t_5, t_9\}$.
After processing $t_7 = t_5$, we have $\{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9\}$.

Since t_1 and t_4 are equivalent, the final dis-equality implies that the constraints are unsatisfiable.

UnionFind data structure

The data structure that is used for the disjoint sets is called ”UnionFind”.

Interface of UnionFind

UnionFind supports four interface methods.

- `UnionFind<T> s` : allocates empty disjoint sets object s
- `s.makeSet(x)` : creates a new set that contains *x*.
- `s.union(x,y)` : merges the sets that contain *x* and *y* and returns representative of union
- `s.findSet(x)` : returns representative element of the set containing *x*.

Repeated calls to `s.findSet(x)` return the same element if no other calls are made in between.

Exercise 9.1

What other methods should be in the interface?

Commentary: Answer: The interface does not let you enumerate the sets. We cannot ask for the sizes. One can think of the extensions depending on the applications.

Equality reasoning via UnionFind

Algorithm 9.1: EqualityReasoning(Set of equalities E, Set of disequalities D)

```
UnionFind s;
for  term t occurring in $E \cup D$ do
    s.makeSet(t)
for  $t = t' \in E$ do
    if  $s.findSet(t) \neq s.findSet(t')$ then
        s.union(t,t')
for  $t \neq t' \in D$ do
    if  $s.findSet(t) = s.findSet(t')$ then
        return Unsatisfiable;
return Satisfiable;
```

Implementing UnionFind

If we can have an efficient implementation of `findSet` and `union` then we may implement equality reasoning efficiently.

What is the best can we do?

We are interested in the amortized cost of the two operations since in practical use they are called several times.

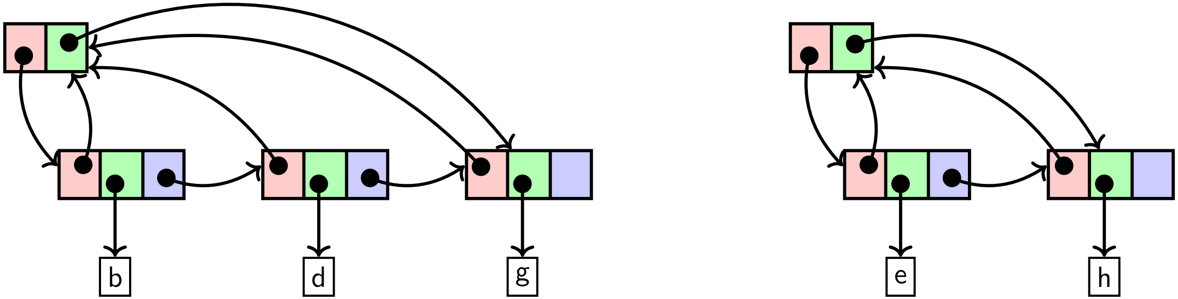
We will design an almost linear data structure for unionFind in terms of the number of calls to the operations.

Topic 9.2

UnionFind via Linked list

Disjoint sets representation via linked list

We need a node with two pointers to serve as the header of the set. It points to the head and tail.



Implementing findSet and makeSet

Exercise 9.2

a. Give implementation of makeSet?

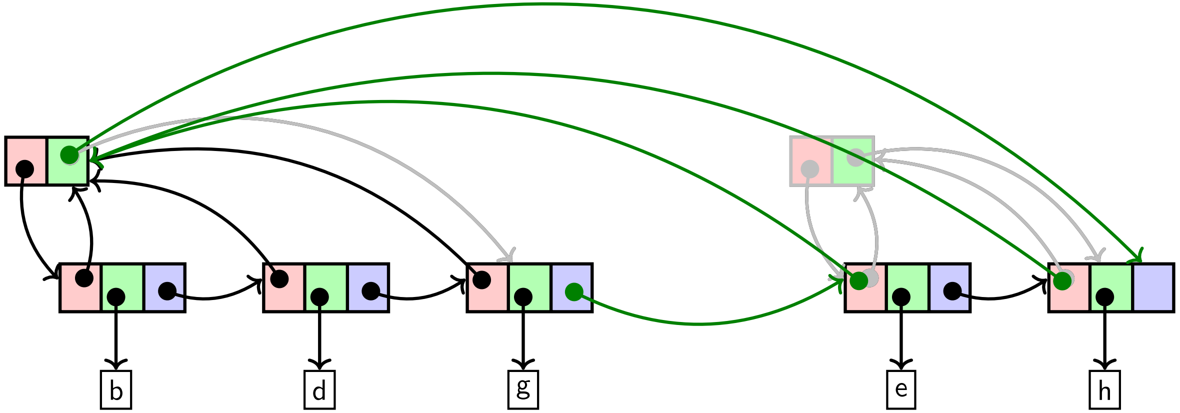
b. Give implementation of findSet?

Commentary: a. makeSet allocates a header node and a linked list, and links the nodes as suggested in the previous slide. b. findSet(node) { return `node->header->head;` }

Implementing union for UnionFind

Exercise 9.3

On calling *union*(*node*(*d*), *node*(*e*)), the gray edges are removed and green edges are added.



Exercise 9.4

Write code for the above transformation.

Running time of union

Theorem 9.1

For n elements, there is a sequence of n union calls such that the total time of the calls is $\Omega(n^2)$.

Proof.

Consider the following sequence of $n-1$ calls after creating nodes x_1,..., x_n.

$$union(x_2, x_1),\ union(x_3, x_2),\ union(x_4, x_3),\ \dots,\ union(x_n, x_{n-1})$$

At ith call, the union will update pointers towards headers in i nodes.

Therefore, the total run time is $\sum_{i=1}^{n-1} \Omega(i)$, which is $\Omega(n^2)$. □

Can we do better?... Yes.

We will consider **three ideas** to make the running time better.

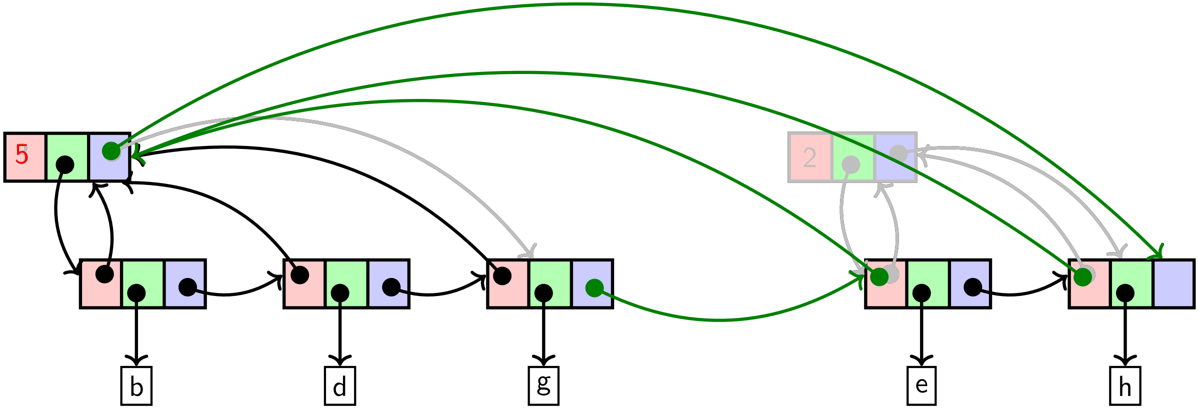
Topic 9.3

Idea 1: weighted union heuristics

weighted union heuristics

Update only the shorter list.

Add a third field in the header node to store the length of the array. Exchange the parameters in *union*(x, y) if the set containing y has a longer length.



Running time with weighted union

Theorem 9.2

Each time a header pointer is updated in a node x, x joins a set with at least double length.

Proof.

Let us suppose x is in a set of size k and y is in a set of size k'.

During call $union(x, y)$, the header pointer of x is updated if $k' \geq k$.

After the union, the total size of the set will be $k + k' \geq 2k$. $\square$

Running time with weighted union(2)

Theorem 9.3

Let us suppose there are n elements in all disjoint sets. The total running time of any sequence of n unions is $O(n \log n)$

Proof.

For each node, the header pointer cannot be updated more than $\log n$ times.

Therefore, the total run time is $O(n \log n)$. □

Can we do better?.... **Yes**

Topic 9.4

Idea 2: UnionFind via forest

Each set is a tree

To avoid long traversals along the linked lists, we may represent sets via trees.

The root of the tree represents the set.

Each node has only two fields: parent and size.

Forest implementation of UnionFind

Algorithm 9.2: MakeSet(x)

```
x.parent = x;
x.size = 1;
```

Algorithm 9.3: FindSet(x)

```
if $x.parent \neq x$ then  return FindSet( x.parent ) ;
return x.parent;
```

Exercise 9.5

Is FindSet tail-recursive?

Forest implementation of UnionFind(2)

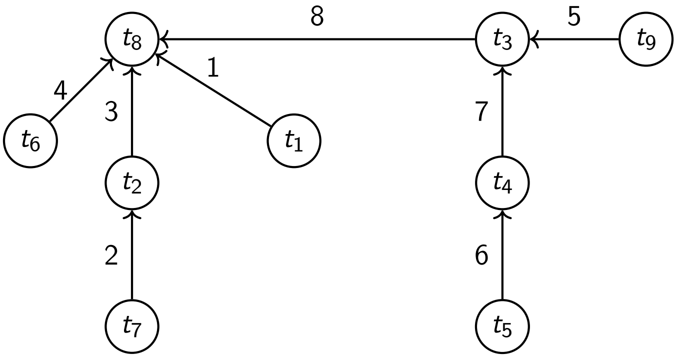
Algorithm 9.4: UNION(x, y)

```
x := FINDSET( x );
y := FINDSET( y );
if $x.size < y.size$ then SWAP(x,y) ;
y.parent := x;
x.size := x.size + y.size
```

Example: unionFind for equality reasoning via forest

Example 9.2

Consider: $\underbrace{t_1 = t_8}_{1} \wedge \underbrace{t_7 = t_2}_{2} \wedge \underbrace{t_7 = t_1}_{3} \wedge \underbrace{t_6 = t_7}_{4} \wedge \underbrace{t_9 = t_3}_{5} \wedge \underbrace{t_5 = t_4}_{6} \wedge \underbrace{t_4 = t_3}_{7} \wedge \underbrace{t_7 = t_5}_{8} \wedge \underbrace{t_1 \neq t_4}_{9}$



Running time of UnionFind via forest

Theorem 9.4

The running time is still $O(n \log n)$.

Can we do better?.... **Yes**

Exercise 9.6

Show that if the height of a tree is h, then it has at least 2^{h-1} nodes.

Commentary: Assume all trees with height h have at least 2^{h-1} nodes. Consider $union(x, y)$, let y be the smaller tree with height h. Then, the number of nodes is 2^{h-1}. The tree T_x containing x will have at least 2^{h-1} nodes. Let us suppose the height of $T_x \leq h + 1$. If the height of the tree is $h + 1$ after the union, then the number of nodes is more than or equal to 2^h. If the height of $T_x \geq h + 1$, there is no height change and only more nodes are added.

Topic 9.5

Idea 3: Path compression

Path compression

Let us directly link a node to its representation, each time we visit a node during the run of FindSet.

Algorithm 9.5: FindSet(x)

```
if $x.parent \neq x$ then  x.parent := FindSet( x.parent ) ;
return x.parent;
```

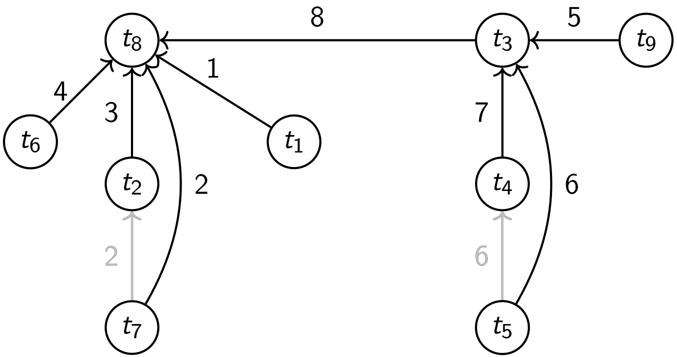
Exercise 9.7

Is FindSet tail-recursive?

Example: unionFind for equality reasoning with path compression

Example 9.3

Consider: $\underbrace{t_1 = t_8}_{1} \wedge \underbrace{t_7 = t_2}_{2} \wedge \underbrace{t_7 = t_1}_{3} \wedge \underbrace{t_6 = t_7}_{4} \wedge \underbrace{t_9 = t_3}_{5} \wedge \underbrace{t_5 = t_4}_{6} \wedge \underbrace{t_4 = t_3}_{7} \wedge \underbrace{t_7 = t_5}_{8} \wedge \underbrace{t_1 \neq t_4}_{9}$



Running time of UnionFind with all three ideas

The running time is $O(n\alpha(n))$, where α is a very slow-growing function.

For any practical $n < 10^{80}$, $\alpha(n) \leq 4$.

The final data structure is almost linear.

The proof of the above complexity is involved. Please read the textbook for the proof.

Commentary: Watch the following interview by Tarjan, who proved the bound. `https://www.youtube.com/watch?v=Hhk8ANKWGJA`

Topic 9.6

Tutorial problems

Exercise: undo problem

Exercise 9.8

Give an algorithm to undo the last union operation assuming there is path compression or not?

Exercise: tight bounds

Exercise 9.9

Show that complexity bounds in theorem 9.3 and theorem 9.4 are tight.

Exercise: printSet

Exercise 9.10

Give an implementation of printSet(x) function in UnionFind (with path compression) that prints the set containing x. You may add one field in each node and must not alter the asymptotic running times of the other operations.

Topic 9.7

Problems

True or False

Exercise 9.11

Mark the following statements True / False and also provide justification.

1. The worst-case running time of UnionFind with path compression is linear.

Exercise: average running time of union find (Endsem 2024)

Exercise 9.12

Prove that the average (not worst case) total time complexity for the linked-list based union find algorithm is $O(n \log n)$ for n union operations for the data structure that initially contains n disjoint elements, even if unweighted unions are used.

Topic 9.8

Extra slides: proof of work

Proof of work

Should we be content if an algorithm says that the formula is unsatisfiable?

We must demand “why?”.

Can we generate proof of unsatisfiability using the UnionFind data structure?

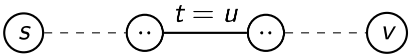
Proof generation in UnionFind (without path compression)

Proof generation from UnionFind data structure for an unsatisfiable input.

The proof is constructed bottom up.

1. There must be a dis-equality $s \neq v$ that was violated.
   We need to find the proof for $s = v$.

2. Find the latest edge in the path between s and v. Let us say it is due to input literal $t = u$.



Recursively, find the proof of $s = t$ and $u = v$.

We stitch the proofs as follows.

$$\frac{\dfrac{\cdots}{s = t} \quad t = u \quad \dfrac{\cdots}{u = v}}{s = v}$$

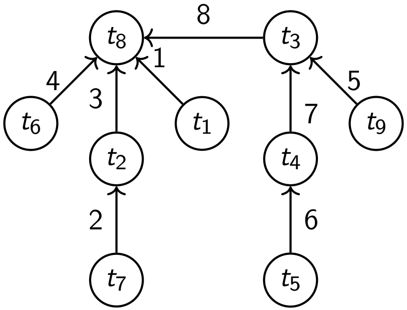
For improved algorithm: R. Nieuwenhuis and A. Oliveras. Proof-producing congruence closure. RTA'05, LNCS 3467

Commentary: We may need to apply the symmetry rule to get the equality in the right order.

Example: UnionFind proof generation

Example 9.4

Consider: $\underbrace{t_1 = t_8}_{1} \wedge \underbrace{t_7 = t_2}_{2} \wedge \underbrace{t_7 = t_1}_{3} \wedge \underbrace{t_6 = t_7}_{4} \wedge \underbrace{t_9 = t_3}_{5} \wedge \underbrace{t_5 = t_4}_{6} \wedge \underbrace{t_4 = t_3}_{7} \wedge \underbrace{t_7 = t_5}_{8} \wedge \underbrace{t_1 \neq t_4}_{9}$



$$\dfrac{t_1 \neq t_4 \qquad \dfrac{\dfrac{t_7 = t_1}{t_1 = t_7} \qquad t_7 = t_5 \qquad t_5 = t_4}{t_1 = t_4}}{\perp}$$

1. $t_1 \neq t_4$ is violated.
2. 8 is the latest edge in the path between t_1 and t_4
3. 8 is due to $t_7 = t_5$
4. Look for proof of $t_1 = t_7$ and $t_5 = t_4$
5. 3 is the latest edge between t_1 and t_7, which is due to $t_7 = t_1$.
6. Similarly, $t_5 = t_4$ is edge 6

End of Lecture 9