# NPTEL Questions — Week 4

CP — July-Nov 2021

## Week 4 — Disjoint Set Union

1. (2 points) Recall the problem War discussed in class, reproduced below for your reference:

   A war is being led between two countries, A and B. As a loyal citizen of C, you decide to help your country's espionage by attending the peace-talks taking place these days (incognito, of course). There are n people at the talks (not including you), but you do not know which person belongs to which country. You can see people talking to each other, and through observing their behaviour during their occasional one-to-one conversations, you can guess if they are friends or enemies. In fact what your country would need to know is whether certain pairs of people are from the same country, or they are enemies. You may receive such questions from C's government even during the peace-talks, and you have to give replies on the basis of your observations so far. Fortunately, nobody talks to you, as nobody pays attention to your humble appearance.

   Now, more formally, consider a black box with the following operations:

   setFriends(x,y) shows that x and y are from the same country setEnemies(x,y) shows that x and y are from different countries areFriends(x,y) returns true if you are sure that x and y are friends areEnemies(x,y) returns true if you are sure that x and y are enemies The first two operations should signal an error if they contradict your former knowledge.

   The two relations 'friends' (denoted by $\times$) and 'enemies' (denoted by $\dagger$) have the following properties:

   $\times$ is an equivalence relation, i.e.

   1. If $x \times y$ and $y \times z$ then $x \times z$ (The friends of my friends are my friends as well.)

   2. If $x \times y$ then $y \times x$ (Friendship is mutual.)

   3. $x \times x$ (Everyone is a friend of himself.)

   $\dagger$ is symmetric and irreflexive

   1. If $x \dagger y$ then $y \dagger x$ (Hatred is mutual.)

   2. Not $x \dagger x$ (Nobody is an enemy of himself.)

   Also:

   1. If $x \dagger y$ and $y \dagger z$ then $x \times z$ (A common enemy makes two people friends.)

   2. If $x \times y$ and $y \dagger z$ then $x \dagger z$ (An enemy of a friend is an enemy.)

   Operations setFriends(x,y) and setEnemies(x,y) must preserve these properties.

   Goal. For every "areFriends" and "areEnemies" operation write '0' (meaning no) or '1' (meaning yes) to the output. Also for every "setFriends" or "setEnemies" operation which contradicts with previous knowledge, output a '-1' to the output; note that such an operation should produce no other effect and execution should continue. A successful "setFriends" or "setEnemies" gives no output.

   Suppose we represent the situation above as an instance of DSU as follows. Every country is an element and the sets are such that two countries belong to the same set if and only if they are friends. Further, for every leader, we maintain pointers to any other leader elements that they are enemies with.

   (a) Consider a situation with three distinct sets X, Y, and Z with leaders x, y, and z. If z is enemies with x and y, have all the friendships been accounted for?

      A. Yes

      **B. No**

      Explanation. No, all friendships have not been accounted for. If z is enemies with x and y, then x and y must be friends (because z is a common enemy). However, since the sets X and Y are given to be disjoint, there is no friendship between x and y. Therefore, all friendships have not been accounted for. In fact, this argument shows that when everything is processed correctly, the leader elements need to keep at most one pointer to another enemy.

   (b) Suppose x and y are enemies and we are asked to make enemies between a and b, where a belongs to the same set as x and b belongs to the same set as y. Is this a contradiction?

A. Yes

**B. No**

Explanation. There is no contradiction. If x and y are enemies, then you can deduce that all elements in set(x) are enemies with all elements in set(y). In particular, let a be an element in set(x) and let b be an element of set(y). Since y is an enemy of x (a friend of a), y is an enemy of a. Since a is now an enemy of y (who is a friend of b), a is also an enemy of b. Since enmity is mutual, we have that b is an enemy of a as well. This holds for all pairs (a,b). Therefore, there is no contradiction when we are asked to make enemies of a and b.

2. (3 points) Path Compression refers to the operation after a *FIND* through which all nodes along the path from the current vertex to the root of the tree are made direct children of the root.

   Union by Rank (by depth) refers to the operation after a *UNION* through which the the root with greater tree depth (longest path to child) is made the parent of the one with the lesser depth.

   These are some of the optimisations that are used to make the implementation of DSUs more efficient. The following questions explore the two techniques.

   (a) Suppose you have a DSU data structure that has already been created. There are a total of $N$ vertices in the entire data structure. You have to process $M$ queries of the *FIND* type and none of *UNION*. What is the worst case time complexity for this *without* Path Compression?

   A. $O(Mlog(N))$

   B. $O(N^2M)$

   C. $O(N+M)$

   **D.** $O(NM)$

   Explanation: The longest child to root path may be of length $O(N)$. Since there is no path compression, the longest path could be queries $O(M)$ times. Hence $O(NM)$ time complexity.

   (b) Suppose you have a DSU data structure that has already been created. There are a total of $N$ vertices in the entire data structure. You have to process $M$ queries of the *FIND* type and none of *UNION*. What is the worst case time complexity for this *with* Path Compression?

   A. $O(Mlog(N))$

   B. $O(N^2M)$

   **C.** $O(N+M)$

   D. $O(NM)$

   Explanation: Each time we call the *FIND* operation on a child at depth $k$, we spend $O(k)$ steps to find the root, however at the same time a path of length $k$ shrinks (due to Path Compression). Hence there can only be $O(N)$ of such traversal steps. After that we have direct edges between any vertex and a root.

   (c) Suppose you have a DSU data structure consisting of two trees $T_1$ and $T_2$, of max depths (longest path from root to a child in the tree) $k_1$ and $k_2$ respectively. You now perform a *UNION* operation between vertices $v_1$ at depth $d_1$ from $T_1$ and $v_2$ at depth $d_2$ from $T_2$. What is the minimum depth of the newly formed tree that you can *guarantee*?

   A. $min(k_1, k_2)$

   B. $max(k_1, k_2)$

   **C.** $max(k_1, k_2) + 1$

   D. $min(d_1 + k_2, d_2 + k_1) + 1$

   Explanation: We make root with the greater depth the parent. This ensures that the depth of the tree does not exceed $1 + max(k_1, k_2)$. If $k_1 = k_2$ then the depth of the resultant tree will increase by 1, in all other cases it will be equal to the depth of the tree with greater depth.

3. (1 point) Suppose you have a set of N people who all do not know each other, and you have to process a sequence of queries of the following kinds:

**Type 1:** `MakeFriends(x,y)` - x and y become friends with each other

**Type 2:** `AreFriends(x,y)` - output 1 if x and y are friends, and output 0 otherwise

In particular, for every pair of people, we need to maintain a state indicating whether they are friends or not, and update this state based on queries of type 1, and report the state for queries of type 2.

Having learned about DSU this week, you decide to store the set of N people as N singleton sets to begin with. For each query of type 1, you perform a union operation, and for each query of type 2, you perform a find operation.

What can you say about this algorithm?

- **A. It may not always produce an accurate answer.**
- B. It works with $O(lg*n)$, $O(lg*n)$ amortized response time to both queries.
- C. It works with $O(lg*n)$, $O(lg*n)$ response time to both queries, but it can be done in $O(1)$,$O(1)$ time with a different approach.
- D. It works with $O(n)$, $O(n)$ worst-case response time to both queries.

Suppose we have the sequence: MakeFriends(x,y) followed by MakeFriends(y,z) and AreFriends(x,z). With the DSU approach, the response to the last query will be 1, while the correct answer is 0.

4. (4 points) Given a language with $N$ words. You want to create a particular message of length $m$ from this language. However each word in the language has an associated cost, and the cost of the message is the sum of costs of all words in the message.

   You find out that some words in this language have the same meaning (but they may have different costs). You want to minimise the cost of your message, and for this you can perform the following operation any number of times

   - Replace a word in the message with another word that has the same meaning.

   Every day you discover a new pair of words that share the same meaning. Your goal is to find out the minimum cost of the message at the end of the $i^{th}$ day.

   You have figured that maintaining a DSU data structure will be the best way to solve this problem.

   (a) On day 1 you discover that the words “dsu” (having cost 3) and “uf” (having cost 7) have the same meaning. Which word would you keep as the parent of the other?

   - **A. “dsu”**
   - B. “uf”

   Explanation: For each group of words that has the same meaning, maintain separate trees. It also makes more sense to have the root of the tree as the word that has the minimum cost.

   (b) On the $i^{th}$ day you wonder how the message *find it* could be sent economically? What strategy should you use?

   - A. For each word in the message, go to the tree where the word is located and return the immediate ancestor of that word.
   - **B. For each word in the message, go to the tree where the word is located. Keep going to the ancestor of the current node till you reach the root of the tree. Return the word at the root.**
   - C. For each word in the message, go to the tree where the word is located. Keep going to the child of the current node till you reach a leaf of the tree. Return the word at that leaf.
   - D. For each word in the message, go to the tree where the word is located. Find all the leaves in this tree. Return the word at the leaf that has the minimum cost.

   Explanation: The most logical and optimal way to maintain the DSU structure is to have the most economic/ cheapest word as the root of any tree. This is because we are interested only in the cheapest word that shares the same meaning.

(c) On the $j^{th}$ day, you discover that the words *find* and *discover* have the same meaning. However, you take a look at your DSU data structure and find that they are part of two different trees. How do you modify your DSU data structure on the $j^{th}$ day?

A. Make the cheaper of the two words as the parent of the more expensive word.

B. Make the more expensive of the two words as the parent of the cheaper word.

**C. Find the roots of the respective trees. Make the cheaper root the parent of the more expensive root.**

D. Find the roots of the respective trees. Make the more expensive root the parent of the cheaper root.

Explanation: This is the Union operation of DSU. The merged tree should have its root as the word with the least cost among all the words in the entire tree. The roots of the two trees before merging were their respective minimum cost words, so the minimum of these two should be the minimum of the entire, new merged tree.

(d) At the end of the $7^{th}$ day you decide that you need to urgently send the message *Find me I is lost.* So far, you have learnt that the following words mean the same, in your given language:

1. *find* - *search*
2. *lost* - *stranded*
3. *discover* - *search*
4. *discover* - *see*
5. *me* - *I*
6. *are* - *is*
7. *he* - *them*

The words in the language and their costs are as follows *find* - 12, *me* - 6, *I* - 8, *is* - 10, *lost* - 7, *search* - 13, *stranded* - 12, *discover* - 3, *see* - 1, *are* - 4, *he* - 4, *them* - 7

What is the minimum cost of the message that you should send?

**A. 24**

B. 20

C. 26

D. 29

Explanation: *See me me are lost.* is the message you should send.