Disjoint sets

Prof. Rhadamés Carmona

Content

- Motivation
- Data structures
- Operations
- Path compression
- Complexity
- Problems
- Homework

Motivation

- There are problems where join operations, and checking if two elements belong to the same set are widely used.
- For these problems, an appropriate data structure must be used for testing membership and for joining operations.
- Sometimes a set can represent an equivalence class.
- Disjoint Sets data structure: called Disjoin sets, union-find, merge-find set.

Data structure

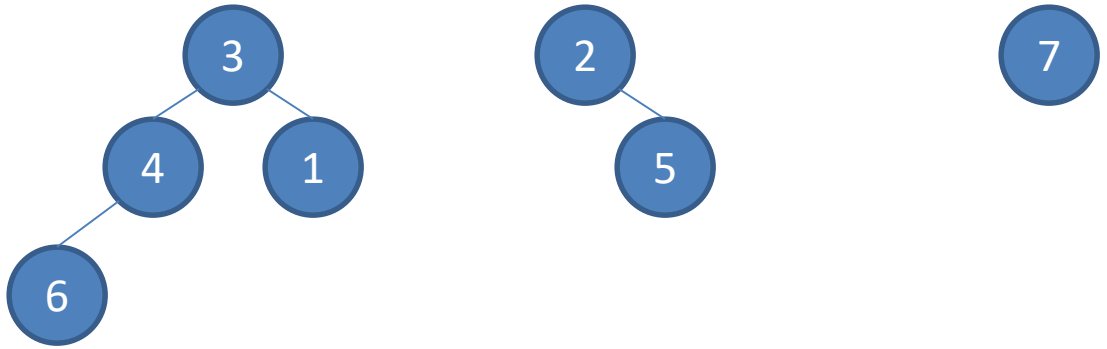
- Each set is represented as a general tree.
- All sets are stored as a forest.
- The union is simply joining two trees.
- Checking if two items belong to the same set (membership) is reduced to checking if they are in the same tree.

Data structure

- To know if 2 elements are in the same tree, just verify if they have a common ancestor. To reduce checks, only the root of the tree to which they belong is checked (it must be the same).
- For this operation, it is more useful for nodes to have a pointer to the parent, and not to pointers to their children.

Data structure

- It is sufficient to store, for every element of the tree, the position of its father. We will ignore the real value of the node, and focus only on its position in the array.



father	3	0	0	3	2	4	0
0	1	2	3	4	5	6	7

Operations

- Find(x): Returns the root of the tree to which x belongs.
- Union-find: This operation finds the root of the trees where both elements are located. If the root matches it returns true. Otherwise false. The “u” parameter indicates whether the sets should be merged.

Operations

```
bool union-find(int x, int y, bool u)
{
        int i=x, j=y;
        while(father[i]>0) i=father[i];
        while(father[j]>0) j=father[j];
        if (u && (i!=j)) father[j]=i;
        return i==j;
}
```

Operations

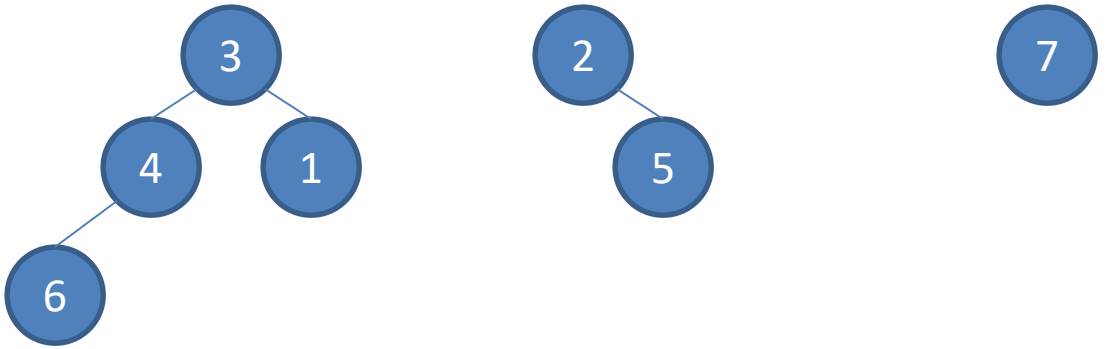
- The implementation is very simple, although its behavior in the worst case is very bad.
- The complexity depends on the height of the tree. To maintain complexity at all the idea is to always keep the tree with minimum height.
- When two trees come in, one is left as root, and the other lowers a level. To minimize the height of the tree, it seems logical that the tree with the largest number of descendants should be rooted.

Operations

- For this purpose, information on the number of descendants of each root node (size-1) should always be maintained.
- To avoid using another array, instead of using 0 for root nodes, a negative number whose absolute value is equal to the number of descendants of this node is placed

Operations

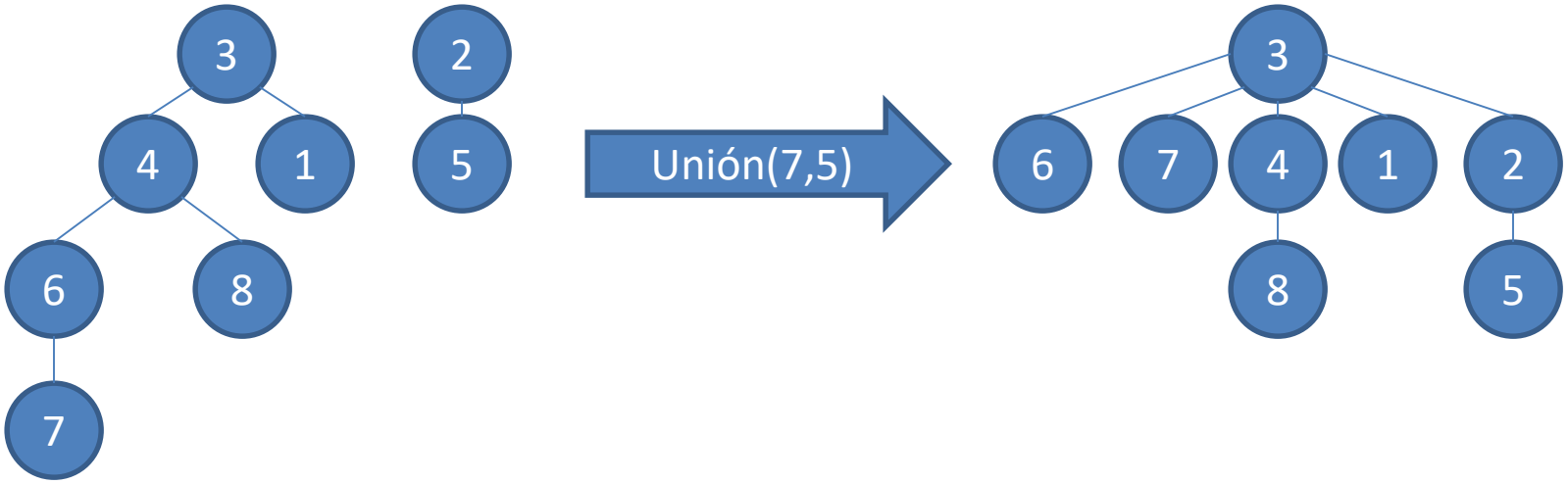
- Instead of using 0 for root nodes, a negative number whose absolute value is equal to the number of descendants of this node is placed.



father	3	-1	-3	3	2	4	0
0	1	2	3	4	5	6	7

Operations

- Other optimization: Each time a join membership operation is made, all visited nodes are placed as direct children of the respective tree root (path compression)



Operations

```
bool union-find(int x, int y, bool u) {  // with path comp.
  int i=x, j=y, t;
  while(father[i]>0) i= father[i]; // find root of x
  while(father[j]>0) j= father[j]; // find root of y
  while(father[x]>0) {t=x; x=father[x]; father[t]=i;}  // path c.
  while(father[y]>0) {t=y; y=father[y]; father[t]=j;}  // path c.
   if (u && (i!=j))  {
          if (father[j] < father[i])   // more negative➔larger
           {   fagher[j]+=fagther[i]-1; father[i]=j;    }
          else
           {   father[i]+=father[j]-1; father[j]=i;    }
   }
  return i==j;
}
```

Complexity

- Hopcroft and Ullman demonstrated that performing a union-find operation runs in time proportional to log*(n) – iterated logarithm of n (how many times do you have to apply logarithm to n to get to 1?).
- As log*(n) grows so slowly, in practice it is considered an operation of O(1)

Problems

- Kruskal algorithm to find the minimum spanning tree of a graph.
- Obtain connected components of a graph.
- Network connectivity.
- Equivalence of finite-state automatons.
- Game (Go, Hex).
- Maintain lists of duplicate copies of web pages.
- Detect cycle in undirected graph
  https://www.geeksforgeeks.org/union-find/

Problems

- Count connected components: Traverse all vertices and count how many nodes have a negative or 0 index (constant time)
- Size of each connected component: find the "non-positive" vertices (roots), get module, and sum 1 .
- Some problems from UVA: 459, 599, 793 (Network Connections), 10158(war), 10178 (counting Faces), 10685 (Nature), 11503 (Virtual Friends), 11987.

Problems

- Implement the disjoint set class. Include constructor, find-root, and union-find.
- Implement 2 problems alone, and one problem in a team.

Problems

These problems have the “ disjoin sets” tag in spoj

- 14834 https://www.spoj.com/problems/FOXLINGS/
- 15267 https://www.spoj.com/problems/FRNDCIRC/
- 18284 https://www.spoj.com/problems/IITWPC4I/
- 2731 https://www.spoj.com/problems/INVENT/
- 26193 https://www.spoj.com/problems/LOSTNSURVIVED/
- 3932 https://www.spoj.com/problems/MCIRGAME/
- 726 https://www.spoj.com/problems/PRO/
- 1707 https://www.spoj.com/problems/RELINETS/
- 71 https://www.spoj.com/problems/TREE1/

But, some of them are not really related to disjoint sets, or they need to be combined with other algorithm or data structure.

Minimal Task for next week

Per student: this week solve two problems by yourself

- 15267 The easiest one (just to start) https://www.spoj.com/problems/FRNDCIRC/
- 26193 A bit challenging https://www.spoj.com/problems/LOSTNSURVIVED/

For the second problem, I can provide a help in the last slide. But do not look at the help before trying yourself.

Minimal Task for next week

Team1: Prem and Hans (select 1)

https://www.spoj.com/problems/RELINETS/

https://www.spoj.com/problems/SOCNETC/

Minimal Task for next week

Team2: Thao & Anthon (select 1)

https://www.spoj.com/problems/FOXLINGS/

https://www.spoj.com/problems/CHAIN/

Help

DO NOT LOOK AT THE NEXT SLICES if you are trying to figure out the solution of

https://www.spoj.com/problems/LOSTNSURVIVED/

LOSTNSURVIVED

- 26193 https://www.spoj.com/problems/LOSTNSURVIVED/
- Use union find.
- We need to group the “roots” indexes for each connected component size. Keep the data structure sorted by connected component size. Update it during the union-find function.
- Then, we simply print the larger size minus the shortest size.