PSO7

Quadratic Hashing, Union Find, BST

You are in the role of a hacker trying to break down a hash table. The information collected so far indicates the hash table uses Quadratic Probing with $h(k, i) = (k + i^2) \bmod m$ for collision management and its current capacity is $m = 9$. The current state of the table is:

0	1	2	3	4	5	6	7	8
17	28	20			5	32		19

The system is nearly overloaded and will collapse if the next item inserted causes at least 4 probes. As an attacker you are considering inserting the following keys: 16, 35 and 10. Which (if any) of these values would bring the system down if inserted next? Explain your answer.

Quadratic probing:

i = i'th collision

Trying 16

0	1	2	3	4	5	6	7	8
17	28	20			5	32		19

$h(16,0) = 16 + 0^2 \bmod 9 = 7$

No collision

Trying 35

0	1	2	3	4	5	6	7	8
17	28	20			5	32		19

$h(35,0) = 35 + 0^2 \bmod 9 =$

Trying 35

0	1	2	3	4	5	6	7	8
17	28	20			5	32		19

h(35,0) = 35 + 0^2 mod 9 = 8 **Collision**

h(35,1) = 35 + 1 mod 9 =

Trying 35

0	1	2	3	4	5	6	7	8
17	28	20			5	32		19

$h(35,0) = 35 + 0^2 \bmod 9 = 8$ **Collision**

$h(35,1) = 35 + 1^2 \bmod 9 = 0$ **Collision**

$h(35,2) = 35 + 2^2 \bmod 9 =$

Trying 35

0	1	2	3	4	5	6	7	8
17	28	20			5	32		19

$h(35,0) = 35 + 0^2 \bmod 9 = 8$ **Collision**

$h(35,1) = 35 + 1^2 \bmod 9 = 0$ **Collision**

$h(35,2) = 35 + 2^2 \bmod 9 = 3$ **No Collision**

Trying 10

0	1	2	3	4	5	6	7	8
17	28	20			5	32		19

$h(10,0) = 10 + 0^2 \bmod 9 =$

Trying 10

0	1	2	3	4	5	6	7	8
17	28	20			5	32		19

h(10,0) = 10 + 0^2 mod 9 = 1 **Collision**

h(10,1) = 10 + 1^2 mod 9 =

Trying 10

0	1	2	3	4	5	6	7	8
17	28	20			5	32		19

$h(10,0) = 10 + 0^2 \bmod 9 = 1$ **Collision**

$h(10,1) = 10 + 1^2 \bmod 9 = 2$ **Collision**

$h(10,2) = 10 + 2^2 \bmod 9 =$

Trying 10

0	1	2	3	4	5	6	7	8
17	28	20			5	32		19

$h(10,0) = 10 + 0^2 \bmod 9 = 1$ **Collision**

$h(10,1) = 10 + 1^2 \bmod 9 = 2$ **Collision**

$h(10,2) = 10 + 2^2 \bmod 9 = 5$ **Collision**

$h(10,3) = 10 + 3^2 \bmod 9 =$

Trying 10

0	1	2	3	4	5	6	7	8
17	28	20			5	32		19

$h(10,0) = 10 + 0^2 \bmod 9 = 1$ **Collision**

$h(10,1) = 10 + 1^2 \bmod 9 = 2$ **Collision**

$h(10,2) = 10 + 2^2 \bmod 9 = 5$ **Collision**

$h(10,3) = 10 + 3^2 \bmod 9 = 1$ **Collision**

Question 2

(1) What is the asymptotic performance of inserting n items with keys sorted in a descending order into an initially empty binary search tree?

(2) Is the operation of deletion "commutative" in the sense that deleting x and then y from a binary search tree leaves the same tree as deleting y and then x? Argue why it is or give a counterexample.

(3) Your friend thinks he has discovered a remarkable property of binary search trees. Suppose that the search for key k in a binary search tree ends up in a leaf. Consider three sets: A, the keys to the left of the search path; B, the keys on the search path; and C, the keys to the right of the search path. Your friend claims that any three keys $a \in A$, $b \in B$, and $c \in C$ must satisfy $a \leq b \leq c$. Give a simple counterexample to his claim.

```
Insert(root,x):
    If root == null: return x
    If (x <= root.val): insert(root.left,x)
    If (x > root.val): insert(root.right,x)
```

https://justin-zhang.com/teaching/cs251Old/S25/pso6Noted.pdf

Question 2

(1) What is the asymptotic performance of inserting n items with keys sorted in a descending order into an initially empty binary search tree?

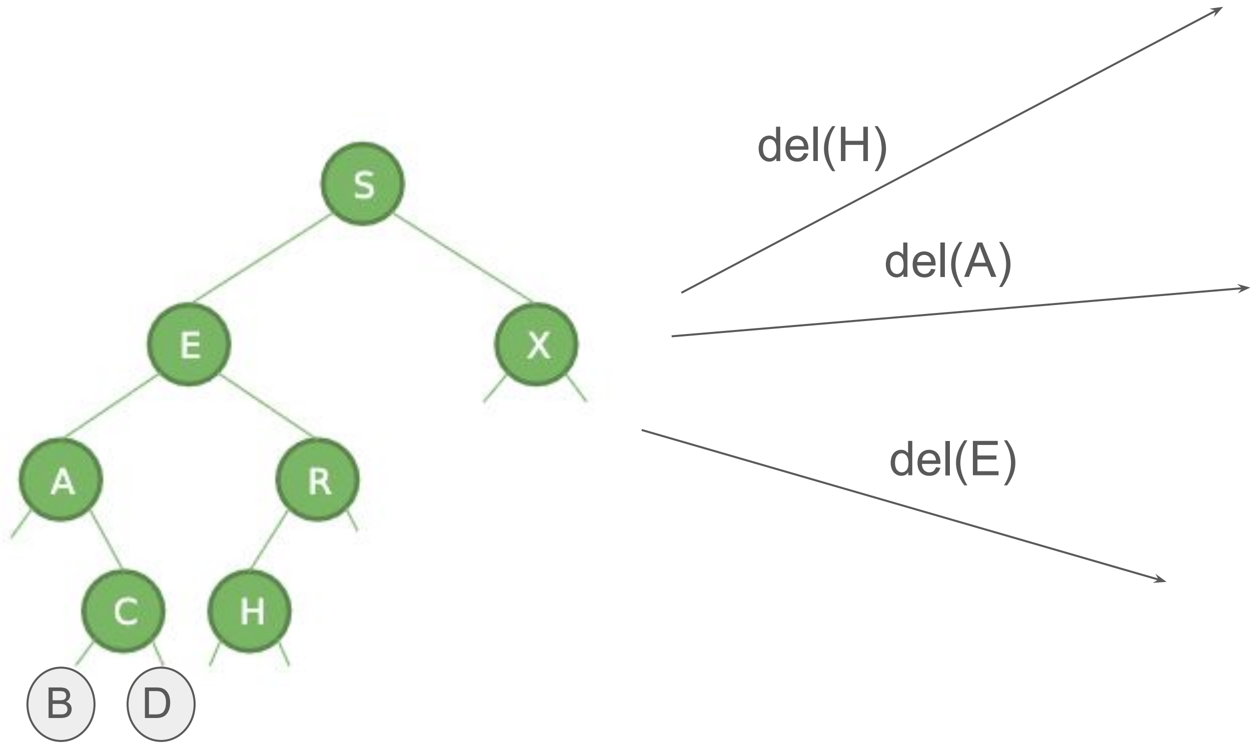
(2) Is the operation of deletion "commutative" in the sense that deleting x and then y from a binary search tree leaves the same tree as deleting y and then x? Argue why it is or give a counterexample.

(3) Your friend thinks he has discovered a remarkable property of binary search trees. Suppose that the search for key k in a binary search tree ends up in a leaf. Consider three sets: A, the keys to the left of the search path; B, the keys on the search path; and C, the keys to the right of the search path. Your friend claims that any three keys $a \in A$, $b \in B$, and $c \in C$ must satisfy $a \leq b \leq c$. Give a simple counterexample to his claim.

How does deletion work?

Deletion in a BST: Depends on # children

Basically, want to delete while keeping order



Question 2

(1) What is the asymptotic performance of inserting n items with keys sorted in a descending order into an initially empty binary search tree?

(2) Is the operation of deletion "commutative" in the sense that deleting x and then y from a binary search tree leaves the same tree as deleting y and then x? Argue why it is or give a counterexample.

(3) Your friend thinks he has discovered a remarkable property of binary search trees. Suppose that the search for key k in a binary search tree ends up in a leaf. Consider three sets: A, the keys to the left of the search path; B, the keys on the search path; and C, the keys to the right of the search path. Your friend claims that any three keys $a \in A$, $b \in B$, and $c \in C$ must satisfy $a \leq b \leq c$. Give a simple counterexample to his claim.

Assume 1 child deletion swaps with **successor**

Delete A, then B	A
	 / \\
	B D
	   /
	  C

Delete B, then A	A
	 / \\
	B D
	   /
	  C

Question 2

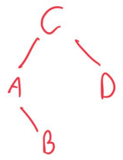
(1) What is the asymptotic performance of inserting n items with keys sorted in a descending order into an initially empty binary search tree?

(2) Is the operation of deletion "commutative" in the sense that deleting x and then y from a binary search tree leaves the same tree as deleting y and then x? Argue why it is or give a counterexample.

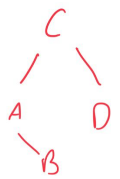
(3) Your friend thinks he has discovered a remarkable property of binary search trees. Suppose that the search for key k in a binary search tree ends up in a leaf. Consider three sets: A, the keys to the left of the search path; B, the keys on the search path; and C, the keys to the right of the search path. Your friend claims that any three keys $a \in A$, $b \in B$, and $c \in C$ must satisfy $a \le b \le c$. Give a simple counterexample to his claim.

If 1 child deletion swaps with **predecessor**

delete C, then delete D



delete D, then delete C



Question 2

(3) Your friend thinks he has discovered a remarkable property of binary search trees. Suppose that the search for key k in a binary search tree ends up in a leaf. Consider three sets: A, the keys to the left of the search path; B, the keys on the search path; and C, the keys to the right of the search path. Your friend claims that any three keys $a \in A$, $b \in B$, and $c \in C$ must satisfy $a \leq b \leq c$. Give a simple counterexample to his claim.

(Union find)

1. Suppose that we implemented the Union-Find data structure with **quick-find**. The current state of the data-structure is defined in the following table.

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

List each disjoint set.

What is Quick Find?

2. What does the table of the union-find data structure look like after running the following two unions: `Union`$(5, 4)$, `Union`$(0, 7)$?

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

`Union(5,4)`

`Union(0,7)`

Question 3

(Union find)

1. Suppose that we implemented Union-Find data structure with quick-union. The current state of the data-structure is defined in the following table.

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

List each disjoint set along with its canonical element (Hint: It may help to draw the corresponding trees).

What is quick union?

How do the trees look?

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

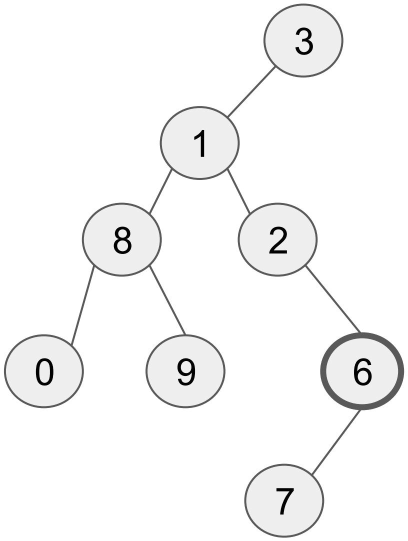
2. Suppose we optimize our construction by implementing path compression and union-by-weight. We then run Union(6, 5). What is updated state of the Union–Find data structure? (Note: Refer to the table in part (a) for the initial state of the union-find data structure.)

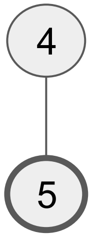
Path compression:

Union by weight:

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

2. Suppose we optimize our construction by implementing path compression and union-by-weight. We then run Union(6, 5). What is updated state of the Union–Find data structure? (Note: Refer to the table in part (a) for the initial state of the union-find data structure.)

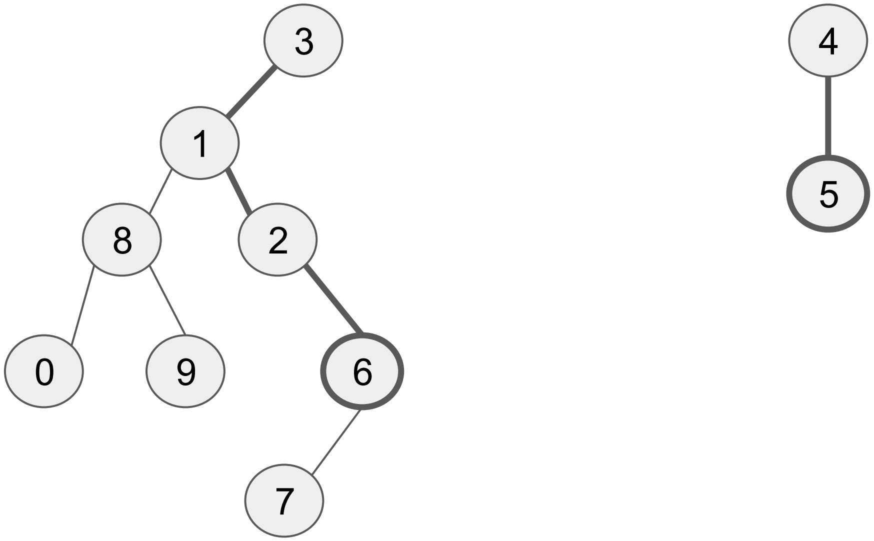




Union(5,6)

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

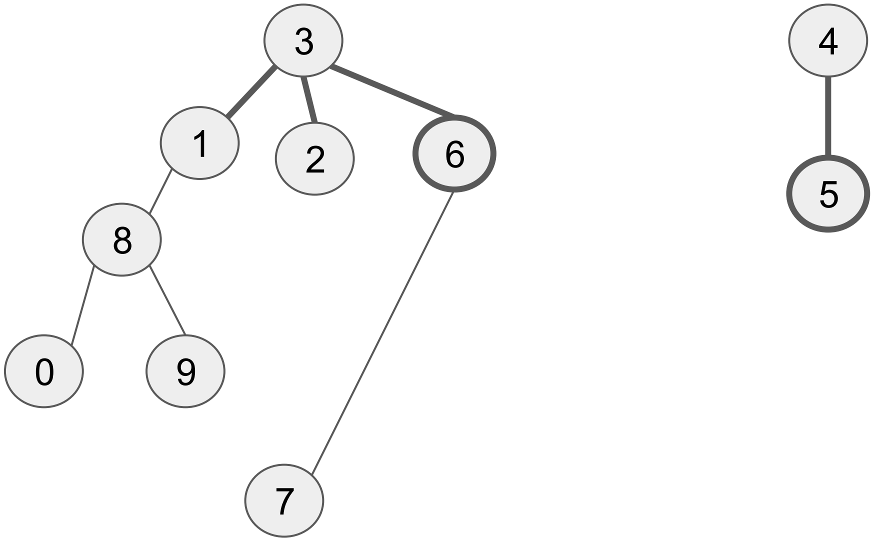
2. Suppose we optimize our construction by implementing path compression and union-by-weight. We then run Union(6, 5). What is updated state of the Union–Find data structure? (Note: Refer to the table in part (a) for the initial state of the union-find data structure.)



Union(5,6)
Step 1: find their roots by traversing up the tree

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

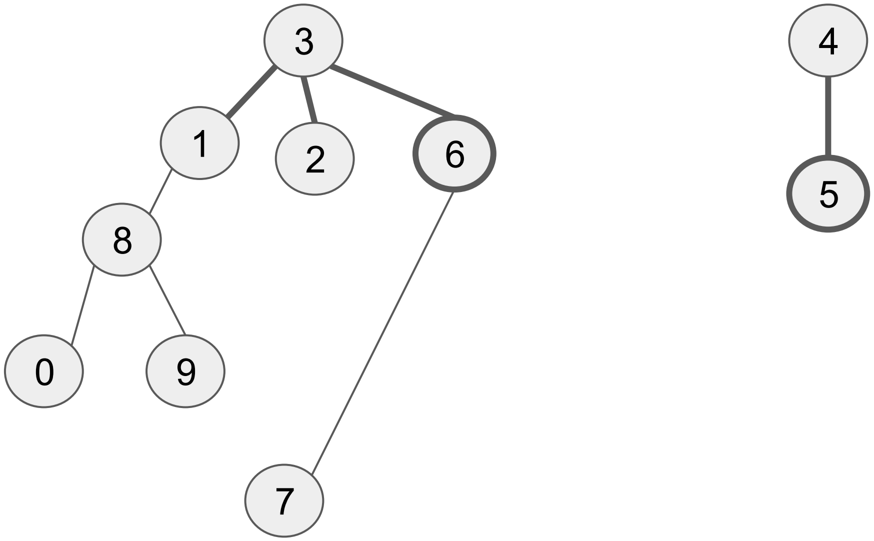
2. Suppose we optimize our construction by implementing path compression and union-by-weight. We then run Union(6, 5). What is updated state of the Union–Find data structure? (Note: Refer to the table in part (a) for the initial state of the union-find data structure.)



Union(5,6)
Step 1: find their roots by traversing up the tree
Path compress

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

2. Suppose we optimize our construction by implementing path compression and union-by-weight. We then run Union(6, 5). What is updated state of the Union–Find data structure? (Note: Refer to the table in part (a) for the initial state of the union-find data structure.)

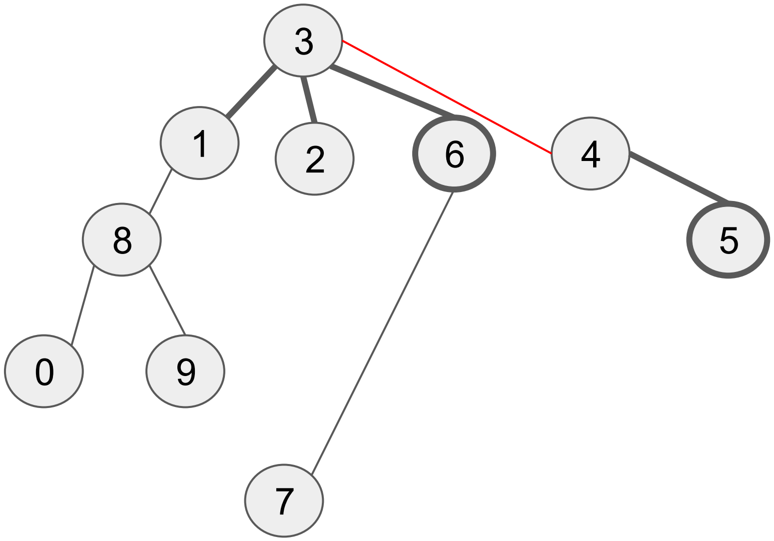


Union(5,6)
Step 1: find their roots by traversing up the tree
Path compress

Step 2: connect roots
Union by weight
(minimize suffering!)

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

2. Suppose we optimize our construction by implementing path compression and union-by-weight. We then run Union(6, 5). What is updated state of the Union–Find data structure? (Note: Refer to the table in part (a) for the initial state of the union-find data structure.)

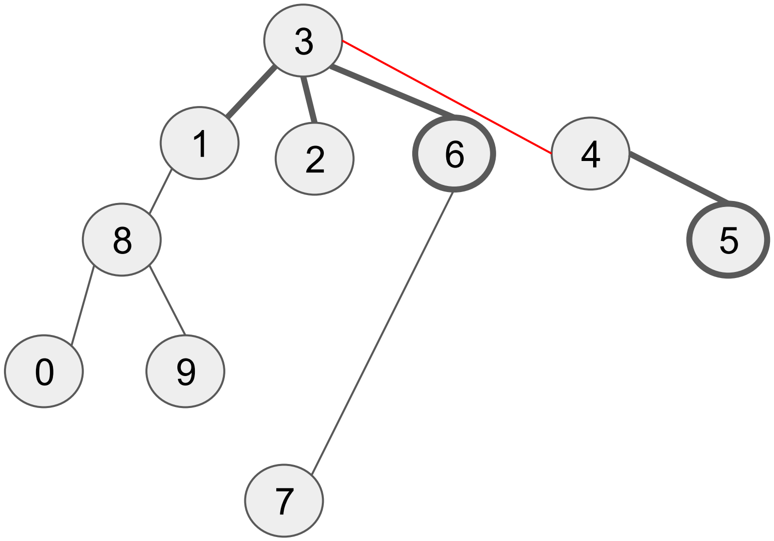


Union(5,6)
Step 1: find their roots by traversing up the tree
Path compress

Step 2: connect roots
Union by weight
(minimize suffering!)

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

2. Suppose we optimize our construction by implementing path compression and union-by-weight. We then run Union(6, 5). What is updated state of the Union–Find data structure? (Note: Refer to the table in part (a) for the initial state of the union-find data structure.)



Union(5,6)
Step 1: find their roots by traversing up the tree
Path compress

Step 2: connect roots
Union by weight
(minimize suffering!)
Step 3: Update the table

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