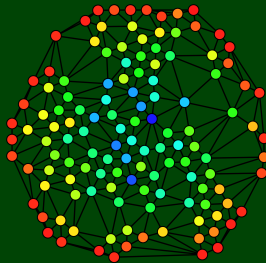# CSE 332

Data Abstractions

Graphs 5: Union Find

A **disjoint sets** data structure keeps track of multiple sets which do not share any elements. Here's the ADT:

UnionFind ADT

find(**x**)	Returns a number representing the set that **x** is in.
union(**x**, **y**)	Updates the sets so whatever sets **x** and **y** were in are now considered the same sets.

Example

```
 1  list = [1, 2, 3, 4, 5, 6];
 2  UF uf = new UF(list); // State: {1}, {2}, {3}, {4}, {5}, {6}
 3  uf.find(1);           // Returns 1
 4  uf.find(2);           // Returns 2
 5  uf.union(1, 2);       // State: {1, 2}, {3}, {4}, {5}, {6}
 6  uf.find(1);           // Returns 1
 7  uf.find(2);           // Returns 1
 8  uf.union(3, 5);       // State: {1, 2}, {3, 5}, {4}, {6}
 9  uf.union(1, 3);       // State: {1, 2, 3, 5}, {4}, {6}
10  uf.find(3);           // Returns 1
11  uf.find(6);           // Returns 6
```

This lecture is an excuse to walk through the genesis of a data structure. We know what we want to get, and we'd like to **create (and analyze) an efficient data structure** that meets our requirements.

To define a UnionFind data structure, we need three things:

- The idea of the data structure
- An implementation of union
- An implementation of find

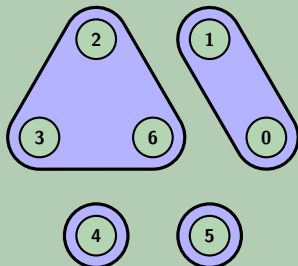**For the duration of the lecture, we will assume we can identify each item with a number from 1 to n.**

Let's start out easy. . .

Data Structure

Type: List<LinkedList<Integer>>
Idea: A mapping from **id** → a list of **id**s in the same set
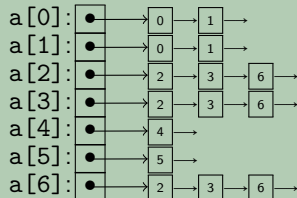
Pictorial View

Data Structure

find(x)

```
1  find(x) {
2      return a[x].front;
3  }
```
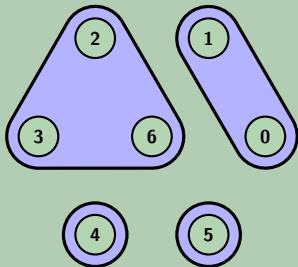
union(x, y)

```
1  union(x, y) {
2      ...
3  }
```

Data Structure

Type: List<LinkedList<Integer>>
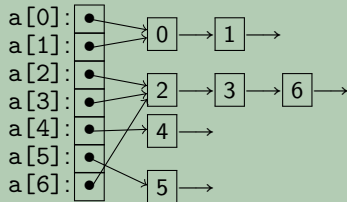**Idea**: A mapping from **id** → a list of **id**s in the same set

Pictorial View

Data Structure

find(x)

```
1  find(x) {
2      return a[x].front;
3  }
```

union(x, y)

```
1  union(x, y) {
2      curr = a[x].head;
3      a[y].tail.next = curr;
4      while (curr != null && curr.next != null) {
5          a[curr.data] = a[y].head
6          curr = curr.next;
7      }
8  }
```

Data Structure

Type: List<LinkedList<Integer>>
Idea: A mapping from **id** $\rightarrow$ a list of **id**s in the same set

union(x, y)

```
1  union(x, y) {
2    curr = a[x].head;
3    a[y].tail.next = curr;
4    while (curr != null && curr.next != null) {
5      a[curr.data] = a[y].head
6      curr = curr.next;
7    }
8  }
```

find(x)

```
1  find(x) {
2    return a[x].front;
3  }
```

Asymptotic Analysis

- find(x) is $\mathcal{O}(1)$
- union(x, y) is $\mathcal{O}(a[x].length)$

Data Structure

Type: List<LinkedList<Integer>>
Idea: A mapping from **id** → a list of **id**s in the same set

find(x)

```
1  find(x) {
2      return a[x].front;
3  }
```

union(x, y)

```
1  union(x, y) {
2      curr = a[x].head;
3      a[y].tail.next = curr;
4      while (curr != null && curr.next != null) {
5          a[curr.data] = a[y].head
6          curr = curr.next;
7      }
8  }
```

Amortized Analysis

Consider any m find/union operations. The **worst** case is going to be that all the operations are all unions, but which unions?

Always union(LARGEST, y), because we have to traverse the first one.

This ends up being $1 + 2 + \cdots + n - 1 = \dfrac{(n-1)n}{2}$. This is a total of $n - 1$ operations. So, the amortized cost of a union is:

$$\frac{(n-1)n}{2(n-1)} = \frac{n}{2}$$

This is horrible! How could this be better?

Data Structure

Type: List<LinkedList<Integer>>
Idea: A mapping from **id** → a list of **id**s in the same set

OLD union(x, y)

```
union(x, y) {
   curr = a[x].head;
   a[y].tail.next = curr;
   while (curr != null && curr.next != null) {
      a[curr.data] = a[y].head;
      curr = curr.next;
   }
}
```

NEW union(x, y)

```
union(x, y) {
   if (a[x].length > a[y].length) {
      x, y = swap(x, y)
   }

   curr = a[x].head;
   a[y].tail.next = curr;
   while (curr != null && curr.next != null) {
      a[curr.data] = a[y].head
      curr = curr.next;
   }
}
```

Asymptotic Analysis

- find(x) is $\mathcal{O}(1)$
- union(x, y) is $\mathcal{O}(\min(a[x].length, a[y].length))$

Data Structure

Type: List<LinkedList<Integer>>
Idea: A mapping from **id** $\rightarrow$ a list of **id**s in the same set

Amortized Analysis

Consider any m find/union operations. The **worst** case is going to be that all the operations are all unions, but which unions?

> Keep the sets as balanced as possible. This will get us the largest gurantee possible, as quickly as possible

n sets of $1 \xrightarrow{\text{union}} n/2$ sets of $2 \xrightarrow{\text{union}} \ldots n/2^i$ sets of $2^i \xrightarrow{\text{union}} \ldots$
So, now we compute:

$$\sum_{i=0}^{\lg n} \left(\frac{n}{2^{i+1}}\right) \text{cost}(\text{union}(2^i, 2^i)) = \sum_{i=0}^{\lg n} \left(\frac{n}{2^{i+1}}\right) 2^i = \sum_{i=0}^{\lg n} \frac{n}{2} = \frac{n}{2} \lg(n)$$
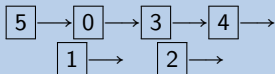
Since we remove one "gap" each time we union, there are $n-1$ union operations total.

So, the amortized cost of a union is: $\dfrac{\frac{n}{2}\lg(n)}{n-1} \approx \lg(n)$.

We started with a **list of linked lists**. Then, we realized that we could use **references to the same linked list** to save memory.

We can do even better. The idea is to use an "implicit list".

Example (Explicit List)

5 → 0 → 3 → 4 →

1 → 2 →

Example (Implicit List)

3	-1	-1	4	-1	0
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]
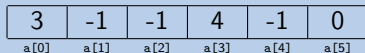
If you've already taken CSE 351, you've seen this idea already! When implementing `malloc`, you store a **free list**. You can save a lot of memory (which in `malloc` is important. . .) by using the unused **data fields** to store the **pointers**.

Using An Implicit List

We need to store:

- pointers to get to the canonical member
- the size of the set

Data Structure

Type: An array
Idea: Each index has either the value of the "next" thing in its set or a negative number representing the size of the set

Pictorial View

Implementation

```
1  init(x) { a[x] = −1 }
2  find(x) {
3      while(a[x] >= 0) {
4          x = a[x]
5      }
6      return x
7  }
8
9  size(x) { return −a[find(x)] }
10
11 union(x, y) {
12     if (size(x) > size(y)) {
13         x, y = swap(x, y)
14     }
15
16     // Now, we have: size(x) <= size(y)
17     a[find(x)] = find(y)
18
19     // Update the size
20     a[find(y)] = size(x) + size(y)
21 }
```
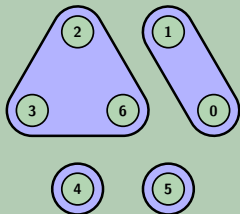
Data Structure

-2	0	6	2	-1	-1	-3
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]

- "Non-canonicals" store "pointers"
- "Canonicals" store −size

-2	0	6	2	-1	-1	-3
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]

Implementation

```
1  init(x) { a[x] = −1 }
2  find(x) {
3      while(a[x] >= 0) {
4          x = a[x]
5      }
6      return x
7  }
8
9  size(x) { return −a[find(x)] }
10
11 union(x, y) {
12     if (size(x) > size(y)) {
13         x, y = swap(x, y)
14     }
15
16     // Now, we have: size(x) <= size(y)
17     a[find(x)] = find(y)
18
19     // Update the size
20     a[find(y)] = size(x) + size(y)
21 }
```

- Assume we only call each size/find once.
- Then, union(x, y) ∈ $\mathcal{O}(\mathrm{find}(x) + \mathrm{find}(y))$.
- So, we only need analyze find(x).
- We claim that find(x) ∈ $\mathcal{O}(\lg n)$.
- To prove this, we will show the **height** of the tree resulting from some number of unions is $\mathcal{O}(\lg n)$
- (Sound familiar?)

We claim:

- If x, y are different heights,
$$\text{height}(\texttt{union}(x,y)) = \max(\text{height}(x), \text{height}(y))$$

$$\texttt{union}(\overset{\bigcirc}{\bigcirc}\,\bigcirc,\ \bigcirc\overset{\bigcirc}{\bigcirc}\overset{\bigcirc}{\bigcirc}\bigcirc) = \overset{\bigcirc}{\bigcirc}\,\bigcirc\ \ \bigcirc\overset{\bigcirc}{\bigcirc}\overset{\bigcirc}{\bigcirc}\bigcirc$$

- If x, y are the same height,
$$\text{height}(\texttt{union}(x,y)) = 1 + \max(\text{height}(x), \text{height}(y))$$

$$\texttt{union}(\overset{\bigcirc}{\bigcirc}\,\bigcirc,\ \bigcirc\overset{\bigcirc}{\bigcirc}\overset{\bigcirc}{\bigcirc}\bigcirc) = \overset{\bigcirc}{\underset{\bigcirc\ \bigcirc\overset{\bigcirc}{\bigcirc}\overset{\bigcirc}{\bigcirc}\bigcirc}{}}$$

Let $f(h)$ be the number of (distinct) nodes necessary to make a tree of height h. Then, $f(1) = 1$ and $f(h) = 2f(h-1)$. So, $f(h) = 2^{h-1}$.

Then, what is the largest tree we can make with n nodes?

$$n = 2^{h-1} \rightarrow \lg(n) = h - 1$$

So, $h \in \mathcal{O}(\lg n)$.

(Notice this is nearly identical to the proof for AVL Trees, except with a slightly different recurrence.)

OLD find(x)

```
1  find(x) {
2      while(a[x] >= 0) {
3          x = a[x]
4      }
5      return x
6  }
```

NEW find(x)

```
1  find(x) {
2      if (a[x] < 0) {
3          return x
4      }
5      a[x] = find(a[x])
6      return a[x]
7  }
```

In Words: Once we've **found** a node... save it.

Amortized Analysis of m find Operations?

Consider what we know:

- We know the worst case height of a tree is $\lg(n)$.
- We know it's difficult to make a tree of large height.
- We know that as soon as we access a path in a tree, it flattens the whole path

This **feels** like it should be better than $\lg(n)$, and it is.

We can use facts to show this, but its outside the scope of this lecture. Instead, we'll just talk about two bounds.

Upper Bound 1: $\texttt{find}(x)$ is amortized $\mathcal{O}(\lg^*(n))$

Let $2\text{STACK}(n) = 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}\!\!\searrow_n$

- $2\text{STACK}(0) = 1$
- $2\text{STACK}(1) = 2$
- $2\text{STACK}(2) = 2^2 = 4$
- $2\text{STACK}(3) = 2^{\left(2^2\right)} = 16$
- $2\text{STACK}(4) = 2^{\left(2^{\left(2^2\right)}\right)} = 2^{16} = 65536$
- $2\text{STACK}(5) = 2^{65536}$

2^{65536} = 20035299304068464649790723515602557504478254755695751419...
26501697371089405955631145308950613088093334810103823434290 72
63181822949382118812668869506364761547029165041871916351587 96
63472194429309279820843091048559905701593189596395248633723 67
20300291696959215610876494888925409080591145703767520850020 66
71563702366126359747144807111774815880914135742720967190151 83
62825606180914588526998261412503012339110827360384376787644 9
04320596037912449090570756031403507616256247603186379312648 47
03743782954975613770981604614413308692118102485959152380195 33
10302921628001605686701056516467505680387415294638422448452 92
53736144253361437372908830379460127472495841486491593064725 20
15155693922628180691650796381064132275307267143998158508811 29
26289011342377827055674210800700652839633221550778312142885 51
67555407334510721311242739956298271976915005488390522380435 70
45848197956393157835510018992000024141963706813559840464039 47
21940160695176901561197269823378900176415171900511334663068 98
14021938348143542638730653955296969138802415816185956110064 03
62119796101859534802787167200122604642492385111393400464351 62
38675670787452594646709038865477434832178970127644555294090 92
02195958575162297333357615955239488529757995402847194352991 35

```
4376370598692891375715374000198639433246489005254310662966916
5243419174691389632476560289415199775477703138064781342309596
1909606545913008901888875880847336259560654448885014473357060
5881709016210849971452956834406197969056546981363116205357936
9791403236328496233046421066136200220175878785185740916205048 9
7117818204001872829399434461862243280098373237649318147898481
1945271300744022076568091037620399920349202390662626449190916
7985461515778839060397720759279378852241294301017458086862263
3692847258514030396155585643303854506886522131148136384083847
7826379045960718687672850976347127198889068047824323039471865
0525660978150729861141430305816927924971409161059417185352275
8875044775922183011587807019755357222414000195481020056617735
8978149953232520858975346354700778669040642901676380816174055
0405117670093673202804549339027992491867306539931640720492238
4748152806191669009338057321208163507076343516698696250209690
2316285935007187419057916124153689751480826190484794657173660
1005892476655445840838334790544144817684255327207315586349347
6051374197795251903650321980201087647383686825310251833775339
0886142618480037400808223810407646887847164755294532694766170
0424461063311238021134588694532200116564076327023074292426051
```

6340696503084422585596703927186946115851379338647569974856867
0079823960604393478850861649260304945061743412365828352144806
7266768418070837548622114082365798029612000274413244384324023
3125740354501935242877643088023285085588608996277445816468085
7875115807014743763867976955049991643998284357290415378143438
8473034842619033888414940313661398542576355771053355802066221
8557706008255128889333222643628198483861323957067619140963853
3832374343758830859233722284644287996245605476932428998432652
6773783731732880632107532112386806046747084280511664887090847
7029120816110491255559832236624486855665140268464120969498259
0565519216188104341226838996283071654868525536914850299539675
5039549383718534059000961874894739928804324963731657538036735
8671017578399481847179849824694806053208199606618343401247609
6639519778021441199752546704080608499344178256285092726523709
8986515394621930046073645079262129759176982938923670151709920
9153156781443979124847570623780460000991829332130688057004659
1458387208088016887445835557926258465124763087148566313528934
1661174906175266714926721761283308452739364692445828925713888
7783905630048248379983969202922221548614590237347822268252163
9957440801727144146179559226617508388902007416992623830028286

17214453142574944015066139463169197629181506579745526236191224848063890033669074365989226349564114665503062965960199720636202603521917776740668777463549375318899587866282125469797102065747232721372918144666659421872003474508942830911535189271114287108376159222380276605327823351661555149369375778466670145717971901227117812780450240026384758788339396817962950690798817121690686929538248529830023476068454114178139110648560236549754227497231007615131870024053910510913817843721791422528587432098524957878034683703337818421444017138688124249984418618129271198533315382567321870421530631197748535214670955334626336610864667332292409879849256691109516143618601548909740241913509623043612196128165950518666022030715613684732364660868905014263913906515063908199378852318365059897299125404479443425166774299659811849233151555272883274028352688442408752811283289980625912673699546247341543333500147231430612750390307397135252069338173843322950701049061867539433130784798015655130384758155685236218010419650255596181934986315912330360964619059902361126811960234418433633345949276319461017166529138237171823942992162725384617760656945422978770713831988170369645886898118632109769003557358846244648357062914530527571012788720279
65

```
3688753719872913083173803391101612854741537737771595172808411
1627597186384924222802373441925469991983672192131287035585307
9669427134163910338827543186136434901009431974090473310144762
9986172542442335561223743571582593338280498624389249822278071
5951762757847109475119033482241412025182688713728193104253478
1961284401764795315050571107229743145699152234516431218486575
7578652819756484350895838472292353455946452121583165775147129
8708225909292655638836651120681943836904116252668710044560243
7042006637090019411855571604720446436969328500600469281405071
1906926139399390273553454556747031490388602202463994826050176
2431969305640666366626090207048887438898907498152865444381862
9173829010518208699363826618683039152732645812867828066013375
0009659336462514609172318031293034787742123467911845479131110
9897794648216922505629399956793483801699157439700537542134485
8745868560472867510654233418938390991105864655951136460610551
5683854121745980180713316361257307961116834386376766730735458
3494789788316330129240800836356825939157113130978030516441716
6825183465736759341980849589479409832925000863897785634946932
1247342610306271374507728615692259662857385790553324064184901
8451328284632709269753830867308409142247659474439973348130810
```

```
70125605842365558953969030647496558414798131099715754204325639
57760704851008815782914082507777385597901291294073094627859444
50585941227319481275322515232480150346651904822896140664668903
05102510916237770448486230229488966711380555607956620732449373
37402783676730020301161522700892184315652121379215748206859
35692079021450227713309998772945959695281704458218195608096581
17027980626698912050615607423256868422713062950098644218534708
10407128917646906550836129916694778023822502789667843489199
40965736170458678624255400694251669397929624714524945408858422
72615375526007190433632919637577750217600519580069384763578
95868784895368721228985578068265181927036320994801558744555751
75312736471421295536494084385586615208012115079075068553344489
25869328385965301327204697069457154695935365857178889486233329
24652027358531885333709484554033365653569881725825289180566
35488363743793348411845580168331827676834646291995605513470039
14787680864032262961664156066750815371064672310846196424753749
05537448053182260027102164009805844975260230356400380834720531
49994117296573678506642140084269649710324191918212121320693976
91439233683747092282677387081322366800869247034915868409911530
98315412063566123187504305467536983230827966457417620806
```

```
58351259821080769101961052229263879745049019254311900620561906
57745241619191318753398404934397682331029846589331837301580959
25228292068208622303325852801192664963144413164427730032377922
74712330696417149945532261035475145631290668854345426869788447
74298177749371011761465162418361668025481529633530849094930067
63654806102940094693750609845588558043970485914449584445079978
49704558355068540874516331646411808312307970438984919050658758
64258107384224205911919416741824904527002882639830579500573417
11487031187142834184499153456702915280104485145176055306971441
76136858238410278765932466268997841831962031226242117739147720
80048835783335692045339359532545648970285585897355057512351295
36540502842081022785248776603574246366673148680279486052445782
67362623085297826505711462484659591421027812278894144816399497
38818846227682448516220518170767221698632657016543169197426512
30041757329904473537672536845792754365412826553581858046840069
36771860502007054724754840080553042495185449526724726134731817
47421800785746934654471360369758841180294080396167469462885406
79172138601225419503819704538417268006398820655632879283958270
85109199588394482977756471520261328710895261634177071516428994
87953564854553553148754978134009964854498
```

```
46202032013368350385425360313636776357521260470742531120923340
28374829494531047274189692872755720276152722682833767417393425
65265328306846999759709775000556088993268502504921288406827741
39881631540456490350775871680074055685724021758685439053228133
77070741583075626962831695556874240605277264858530506113563848
51965918968649596335568216975437621430778665934730450164822432
96489127070989807667662567151726906205881554966638257382927418
20822789606844882229833948166709840390242835143068137672534601
26007269262969468672750794346190439996618979611928750519442356
40264430327173734159128149605616835398818856948404534231142461
35599252723300648816274667235237512343118934421188850850793581
63848994487544756331689213869675574302737953785262542329024881
04718193903722066689470220425883689584093999845356094886994683
38525796751618821594109816249187418133647269651239806775619479
12557957446471427868624053750576104204267149366084980238274680
57598259133100691994190465190653117190892607794911921794640735
51296338645230356733455880333131970803654571847915504326548995
59705862888286866606618021882248602144999973122164138170653480
17551043840662441282280361664890425737764095632648282525840766
904560843949032529052633753231650908768133614
```

```
80088395554942237096734840072642705701165089075196155370186626
47974563811878561754571134004738107627630149533097351741806554
79112660938034311378532532883533352024934365979129341284854970
94682632907583019307266533778255931433111096384805394085928398
89077962104798479196868765399874770959127887274758744398067798
24968278272200926449944559380414608770641941810440758269805688
03894965461658798390466058764534181028990719429302177451997610
44950431968415034555140448209289333786573630528306199900777487
26922998608279053171691876578860908941817057993404890218441559
79109267686279659758395248392673488363474565168701616624064242
42424122896111801061568234253939218005248345472377921991122859
59141918774917938233400100781283265067102817813960291209147201
00947878752551263372884222353869490067927664511634758101193875
31965724212147603828477477457170457861041738574791130190858387
78901523343430130052827970385803598151829296003056826120919509
43737325454171056383887047528950563961029843641360935641632589
40813798151169333861979733982167076100460798009601602482309694
30438069566201232136501405495862506152825880330229083858124784
69315720323233601899469437647726721879376826431828382603564520
69946863021604887452842436359355862233350623
5
```

```
77490200249552738734585956405160830583053770732533971552620441
47054295735383611136775231699727402929416742044232481138750750
63131907827218886405337469421384216992886294047963530515056070
88126366206497231257579019598873041195626227343728900516561111
10941117452777965482790471250581999077498063821559376885546498
82293898540829132512907647838632249478101675349169348928810420
03015610283386143827378160946341335383578340765314321417150655
58775478202524547806573013422774706167442419689526131642741046
96954746214837562882997718041867850845469656191509086958742511
84435837306590951460980451247409411373899927822492983367796011
01538709612974970556630163730720275073475992294379239382442742
11861582361613178863925530951171884212985083072382597291441
42251579403883011359083331651858234967221259621812507058113759
49552502274727467436988713192667076929919908446716122873885845
75846227265733307537355728239516169641751986750126817454293237
38294143824814377139861906716657572945807804820559511881687188
07521297183263644215533678775127476694079011705750981957508456
35652173895441798750745238544552001335720333323798950743939053
12918212255259833790909463630202185353848548250628977156169638
60712382771725621313460549401770413581731931763370136
```

```
915030491653394647637177664391207983474946273978221715020906701
903024697621512785219561420708064616313732365178539762920920
2550028896201297014137964003805573494926907353514596120867479
65477336929587736286356601437679640384307968641385634478013282
61284589184898528048048844180821639423974014362903481665458
1144543664600324906187630395023564020445307482102413668951966
44221339200757479128683805175150634662569391937740283512075666
26082989049187728783385217852279204577184696585527879044756296
19266399200840930207567392536373562839082981757790215320210640
96173732835984940666521411981838108845154597728951645721318977
97907491941013148368544639616904607030107596818933741217575988
1651270007612627891695104063158576375347874200702220510708912
57612361658026806815858499852631465878086616800733264676830206
391697203064894405628195406190685242003053463156621891327309069
68735318164109451428803660599522024824888671155442910472192913
424834643870536850864874909917881267056566538719104972182004237
149274016446094345984539253670613221061653308566202118896823400
575267548610147699368873820958455221157192347968688160853631615
86288015039594941852948922707441082820716930338781808493620401
825522227101098565344481720747075601924591599
```

5356302418122547326609330271039796809106493927272268303541046
7632591355279683837705019855234621222858410557119921731717969
8043393177077507556270560478317798444476375602546370333692471
1422081551997369137197516324130274871219986340454824852457011
8553342675264715978310731245663429805221455494156252724028915
3333543493412178620370072603152798707718724912344944771479095
2073476138542548531155277330103034247683586549609372232400715
4518129732692081058424090557725645803681462234493189708138897
1432998313476177996797124537823107037391514738786921191875667
0031932128189680332269659445928621060743882741691946516226763
2540665070881071030394178860564893769816734159025925194611823
6429456526669372203155504700213598846292758012527715422016299
5486313032491231102962792372389976641680349714122652793190763
6326136814145516376656559839788489381733082668779901962886932
2965973799519316211872154552873941702436698855938887933167445
3336311954151840408828381519342123412282003095031334105070476
0159987985472529190665222479319715440331794836837373220821885
7733416238564413807005419135302459439135025545318864547962522
6025176292837433046510236105758351455073944333961021622967546
1415781127197001738611494279501411253280621254775810512972088

129677307591973829734414452566887708553245708889583209938234321027182241147637327913575686154212528496579033350931527769255058456440105521926445053120737562877449981636463328358161403301758139673594273276904489203618803867549557518068900585329272014939235005252584514670698262854825788326739873522045722823929020714482221988558710289699193587307427781515975762076402395124386020203259659625021257834995771008562638611823381331850901468657706401067627861758377277289589274603940393033727187385053691295712671506689668849388088514294360996201296675907922508227531381284985152690293170026313632894209579757795932763553116206667534886513173238724387480635133145126448899675898288129254800764251865864902411112730135719718138160258317850693224400799865663537154408845486639318170839573578079905973083909488180406093595919090747396090441015051632174968141210076571917748376735575100073361692238653742907945780320004233745280756615304292901449578062963413838355178359976470885134900485697369796523869584599459559209070905895689145114141268450546211794502661175016692826025095077077821195043261783223562437601776799362796099368975191394965033335850715541843645685261667424368892037103749532842592713161053783498074073915863381796

```
10116137124237614267225417320559592027821293257259471464172249
77321316381845326555279604270541871496236585252458648933254145
06264233788565146467060429856478196846159366328895429978072254
22647904006160197519750074605451500602918066382714970161109879
51336633771378434416194053121445291855180136575558667615019373
02969193207612000925506508158327550849934076879725236998702356
79310268041367457189566414318526790547171699629903630155456450
90044802789055701968328313630718997699153166679208958768572290
60009154729196363816735966739599757103260155719202373485805211
28117458610065152598883843114511894880552129145775699146577530
04138471712457796504817585639507289533753975582208777506072339
44558789590571915673<6
```

This number has 19729 digits. . .

$\lg^*(n)$ is the **inverse function** of 2STACK.

So, basically:

$$\lg^*(n) \le 5$$

But it gets better. . .

Upper Bound 2: find(x) is amortized $\mathcal{O}(\alpha(n))$

The Ackermann function grows even more quickly than 2STACK.

It turns out $\alpha(n)$, the **inverse Ackermann function** is also an upper bound. . .

Interestingly, **it is also a lower bound** for the disjoint data structures problem! We can't do better than the algorithm we came up with! (Just like with sorting!)