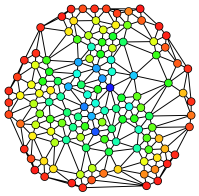
Adam Blank | Lecture 24 | Winter 2016

CSE 332

Data Abstractions

Graphs 5: Union Find



Disjoint Sets ADT

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
```

Data Structure?

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.

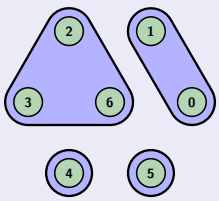
Let's start out easy...

Implementation 1: A List of LinkedLists

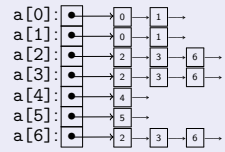
Data Structure

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)

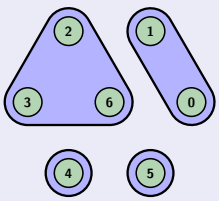
```
1 union(x, y) {
2    ...
3 }
```

Implementation 1.5: A List of LinkedLists

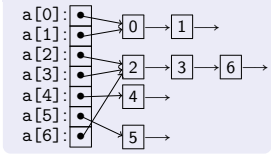
Data Structure

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 }
```

Implementation 1.5: A List of LinkedLists 5

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 }
```

Asymptotic Analysis

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

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 $1+2+\cdots+n-1=\frac{(n-1)n}{2}$. This is a total of $n-1$

Implementation 2: A List of LinkedLists Unioned-By-Weight 6

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 $\mathcal{O}(\min(\texttt{a[x].length},\texttt{a[y].length}))$

Amortized Analysis

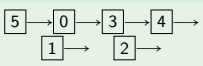
Consider any m `find`/`union` operations. The **worst** case is going to be that all the operations are all unions, but which unions?

Implementation 3: IMPLICIT Lists Unioned-By-Size 7

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)



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:

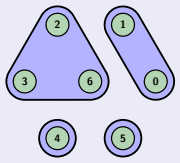
- pointers to get to the canonical member
- the size of the set

Implementation 3: IMPLICIT Lists Unioned-By-Size 8

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



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

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 }
```

Analyzing Implementation 3 9

-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 }
```

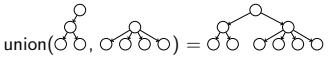
- Assume we only call each size/find once.
- Then, `union(x, y)` $\in \mathcal{O}(\texttt{find(x)}+\texttt{find(y)})$.
- So, we only need analyze `find(x)`.
- We claim that `find(x)` $\in \mathcal{O}(\lg n)$.
- To prove this, we will show the **height** of the tree resulting from some number of unions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

Analyzing Implementation 3 10

We claim:

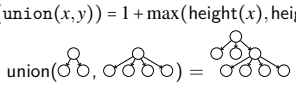
- If x, y are different heights,

$$\text{height}(\texttt{union}(x,y)) = \max(\text{height}(x), \text{height}(y))$$



- If x, y are the same height,

$$\text{height}(\texttt{union}(x,y)) = 1 + \max(\text{height}(x), \text{height}(y))$$



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.)

Implementation 4: Implicit Lists Unioned-By-Size with Path Compression 11

OLD `find(x)`

```
find(x) {
   while(a[x] >= 0) {
      x = a[x]
   }
   return x
}
```

NEW `find(x)`

```
find(x) {
   if (a[x] < 0) {
      return x
   }
   a[x] = find(a[x])
   return a[x]
}
```

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.

Iterated Logarithm 12

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

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

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

$2\text{STACK}(5) = 2^{65536}$ 13

$2^{65536} = 2003529930406846464979072351560255750447825475569751419\ldots$
$2650169737108940595563114530895061308809333481010382343429072$
$6318182294938211881266886950636476154702916504187191635158796$
$6347219442930927982084309104855990570159318959639524863372367$
$2030029169695921561087649488892540908059114570376752085002066$
$7156370236612635974714480711177481588091413574272096719015183$
$6282560618091458852699826141425030123391108273603843767876449$
$0432059603791244909057075603140350761625624760318637931264847$
$0374378295497561377098160461441330869211810248595915238019533$
$1030292162800160568670105651646750568038741529463842244845292$
$5373614425336143737290883037946012747249584148649159306472520$
$1515569392262818069165079638106413227530726714399815850881129$
$2628901134237782705567421080070065283963322155077831214288551$
$6755540733451072131124273995629827197691500548839052238043570$
$4584819795639315785351001899200002414196370681355984046403947$
$2194016069517690156119726982337890017641517190051133466306898$
$1402193834814354263873065395529696913880241581618595611006403$
$6211979610185953480278716720012260464249238511139340046435162$
$3867567078745259464670903886547743483217897012764455529409092$
$0219595857516229733335761595523948852975799540284719435299135$

$2\text{STACK}(5) = 2^{65536}$ 14

$4376370598692891375715374000198639433246489005254310662966916$
$5243419174691389632476560289415199775477703138064781342309596$
$1909606545913008901888875880847336259560654448885014473357060$
$5881709016210849971452956834406197969056546981363116205357936$
$9791403236328496233046421066136200220175787851857409162050489$
$7117818204001872829399434461862243280098373237649318147898481$
$1945271300744022076568091037620399920349202390662626449190916$
$7985461515778839060397720759279378852241294301017458086862263$
$3692847258514030396155585643303854506886522131148136384083847$
$7826379045960718687672850976347127198889068047824323039471865$
$0525660978150729861141430305816927924971409161059417185352275$
$8875044775922183011587807019755357222414000195481020056617735$
$8978149953232520858975346354700778669040642901676380816174055$
$0405117670093673202804549339027992491867306539931640720492238$
$4748152806191669009338057321208163507076343516698696250209690$
$2316285935007187419057916124153689751480826190484794657173660$
$1005892476655445840838334790544144817684255327207315586349347$
$6051374197795251903650321980201087647383686825310251833775339$
$0886142618480037400808223810407646887847164755294532694766170$
$0424461063311238021134588694532200116564076327023074292426051$

$2\text{STACK}(5) = 2^{65536}$ 15

$6340696503084422585596703927186946115851379338647569974856867$
$0079823960643934788508616492603049450617434123658283521448806$
$7266768418070837548622114082365798029612000274413244384324023$
$3125740354501935242877643088023285085588608996277445816468085$
$7875115807014743763867976955049991643998284357290415378143438$
$8473034842619033888414940313661398542576355771053355802066221$
$8557706008255128889333222643628198483861323957067619140963853$
$3832374343758830859233722284644287996245605476932428998432652$
$6773783731732880632107532112386806046747084280511664887090847$
$7029120816110491255559832236624486855665140268464120969498259$
$0565519216188104341226838996283071654868525536914850299539675$
$5039549383718534059000961874894739928804324963731657538036735$
$8671017578399481847179849824694806053208199606618343401247609$
$6639519778021441199752546704080604899344178256285092726523709$
$8986515394621930046073645079262129759176982938923670151709920$
$9153156781443979124847570623780460000991829332130688057004659$
$1458387208088016887445835557926258465124763087148566313528934$
$1661174906175266714926721761283308452739364692445828925713888$
$7783905630048248379983969202922221548614590237347822268252163$
$9957440801727144146179559226175083889020074169926238300282286$

2STACK$(5) = 2^{65536}$ 17

3688753719872913083173803391101612854741537737771595172808411
1627597186384924222802373441925469991983672192131287035585307
9669427134163910338827543186136434901009431974090473310144762
9986172542442335561223743571582593338280498624389249822278071
5951762757847109475119033482241412025182688713728193104253478
1961284401764795315050571107229743145699152234516431218486575
7578652819756484350895838472292353455946452121583165775147129
8708225909292655638836651120681943836904116252668710044560243
7042006637090019411855571604720446436969328500600469281405071
1906926139399390273553454556747031490388602202463994826050176
2431969305640666366626090207048887438898907498152865444381862
9173829010518208699368266186830391527326458128678280660133750
0009659336462514609172318031293034787742123467911845479131110
9897794648216922505629399956793483801699157439700537542134485
8745868560472867510654233418938390991105864655951136460610551
5683854121745980180713316361257307961116834386376766730735458
3494789788316330129240800836356825939157113130978030516441716
6825183465736759341980849589479409832925000863897785634946932
1247342610306271374507728615692259662857385790553324064184901
8451328284632709269753830867308409142247659474439973348130810

2STACK$(5) = 2^{65536}$ 18

7012560584236558953969030647496558414798131099715754204325639
5776070485100881578291408250777738559790129129407309462785944
5058594122731948127532251523248015034665190482289614066468903
0510251091623777044848623022948896671138055560795662073244937
3374027836767300203011615227008921843515652121379215748206859
3569207902145022771330999877294595969528170445821819560809658
1170279806266989120506156074232568684227130629500986442185347
0810407128917646906550836129916694778023822502789667843489199
4096573617045867862425540069425166939792926247145249454088584
2272615375526007190433632919637577750217600519580069384763578
9586878489536872122898557806826518192703632099480155874455575
1753127364714212955364940843855866152080121150790750685533444
8925869328385965301327204697069457154695935365857178889486233
3292465202735853188533370948455403336565356988172582528918056
6354883637437933484118455801683318276768346462919956055134700
3914787680864032262961664156066750815371064672310846196424753
7490553744805318226002710216400980584497526023035640038083472
0531499411729657367850664214008426964971032419191821212132069
3976914392336837470922826773870813223668008692470349158684099
1153098315412063566123187504305467536983230827966457417620806

2STACK$(5) = 2^{65536}$ 19

5835125982108076910196105222926387974504901925431190062056190
6577452416191913187533984049343976823310298465893318373015809
5925228292068208622303325852801192664963144413164427730032377
9227471233069641714994553226103547514563129066885434542686978
8447742981777493710117614651624183616680254815296335308490849
9430067636548061029400946937506098455885580439704859144495844
4507997849704558355068540874516331646411808312307970438984919
0506587586425810738422420591191941674182490452700288263983057
9500573417114870311871428341844991534567029152801044851451760
5530697144176136858238410278765932466268997841831962031226242
1177391477208004883578333569204533935953254564897028558589735
5057512351295365405028420810227852487766035742463666731486802
7948605244578267362623085297826505711462484659591421027812278
8941448163994973881884622768244851622051817076722169863265701
6543169197426512300417573299044735767253684579275436541282 65
5358185804684006936771860502007054724754840080553042495185449
5267247261347318174742180078574693465447136036975884118029408
0396167469462885406791721386012254195038197045384172680063988
2065632879283958270851091995883944829777564715202613287108952
6163417707151642899487953564854553553148754978134009964854498

2STACK$(5) = 2^{65536}$ 20

4620203201336835038542536031363676357521260470742531120923340
2837482949453104727418969287275572027615272268283376741393425
6526532830684699975970977500055608899326850250492128840682741
3988163154045649035077587168007405568572402175868543905322813
3770707415830756269628316955687424060527726485853050611356384
8519659189686495963355682169754376214307786659347304501648224
3296489127070989807667662567151726906205881554966638257382927
4182082278960684488222983394816670984039024283514306813767253
4601260072692629694686727507943461904399966189796119287505194
4235640264430327173734159128149605616835398818856948404534231
1424613559925272330064881627466723523751234311893442118885085
0793581638489944875447563316892138696755743027379537852625423
2902488104718193903722066689470220425883689584093999845356094
8869946833852579675161882159410981624918741813364726965123980
6775619479125579574464714278686240537505761042042671493660849
8023827468057598259133100691994190465190653117190892607794911
9217946407355129633864523035673345588033131970803654571847 91
5504326548995597058628882868666066180218822486021449999731221
6413817065348017551043840662441282280361664890425737764095632
6482825258407669045608439490325290526337532316509087681336614

2STACK$(5) = 2^{65536}$ 21

8008839555494223709673484007264270570116508907519615537018626
4797456381187856175457113400473810762763014953309735174180655
4791126609380343113785325328835335202493436597912934128485 49
7094682632907583019307266533778255931433111096384805394085928
3988907796210479847919686876539987477095912788727475874439806
7798249682782722009264499445593804146087706419418104407582698
0568803894965461658798390466058764534181028990719429302177451
9976104495043196841503455514044820928933378657363052830619990
0777487269229986082790531716918765788609089418170579934048902
1844155979109267686279659758395248392673488363474565168701616
6240642424241228961118010615682342539392180052483454723779219
9112285959141918774917938233400100781283265067102817813960291
2091472010094787875255126337288422235386949006792766451163475
8101193875319657242121476038284774774571704578610417385747911
3019085838778901523343430130052827970385803598151829296003056
8261209195094373732545417105638388704752895056396102984364136
0935641632589408137981511693338619797339821670761004607980096
0160248230969430438069566201232136501405495862506152825880330
2290838581247846931572032323360189946943764772672187937682643
1828382603564520699468630216048874528424363593558622333506235

2STACK$(5) = 2^{65536}$ 22

7749020024955273873458595640516083058305377073253397155262044
4705429573538361113677523169972740292941674204423248113875075
6313190782721888640533746942138421699288629404796353051505607
8812636620649723125757901959887304119562622734372890051656111
1094111745277965482790471250581999077498063821559376885546498
8229389854082913251290764783863224947810167534916934892881042
0301561028338614382737816094634133538357834076531432141715065
5877547820252454780657301342277470616744241968952613164274104
6954746214837562882997718041867850845469656191509086958742511
8443583730659095146098045124740941137389992782249298336779601
1015387096129749705566301637307202750734759922943792393824427
4211861582361613178863925530951171884212985083072382597291441
4225157940388301135908333165185823496722125962181250705811375
9495525022747274674369887131926670769299199084467161228738858
4575846227265733075373557282395161696417519867501268174542 93
2373829414382481437713986190671665757294580780482055951188168
7188075212971832636442155336787751274766940790117057509819575
0845635652173895441798750745238544552001335720333323798950743
9390531291821225525983379090946360202185353848854825062897 71
5616963860712382771725621313460549401770413581731931763370136

2STACK$(5) = 2^{65536}$ 23

915030491653394647637177664391207983474946273978221715020906701903024697621512785219561420708064616313732365178539762920920255002889620129701413796400380557349492690735351459612086747965477336929587736286356601437679640384307968641385634478013282612845891848985280480488441808216394239740143629034816654581144543664600324906187630395023564020445307482102413668951966442213392007574791286838051751506346625693919377402835120756662608298094918772878338521785227920457718469658552787904475621926639920084093020756739253637356283908298175779021532021064096173728359849406665214119818381088451545977289516457213189779790749194101314836854463961690460703010759681893374121757598816512700076126278916951040631585763753478742007022051070891251276239169510406315857637534787420070222051070891251276239169510406315857637534787420070222051070862680681585849985263146587808661680073326467683020639169720306489440562819540619068524200305346315662189132730906968735318164109451428803660599522024824888671155442910472192913424834643870536850864874909917881267056566538719104972182004237149274016446094345984539253670613221061653308566202118896823400575267548610147699368873820958455221157193247968688816085363161586288015039594941852948922707441082820716930338781808493620401825522227101098565344481720747075601924591559978180849362040182552222710109856534448172074707560192459155991502067019030246976215127852195614207080646163137323651785397629209202550028896201297014137964003805573494926907353514596120867479654773369295877362863566014376796403843079686413856344780132826128458918489852804804884418082163942397401436290348166545811445436646003249061876303950235640204453074821024136689519664422133920075747912868380517515063466256939193774028351207566626082989049187728783385217852279204577184696585527879044756219266399200840930207567392536373562839082981757790215320210640961737328359849406665214119818381088451545977289516457213189779790749194101314836854463961690460703010759681893374121757598816512700076126278916951040631585763753478742007022051070891251276239169510406315857637534787420070222051070862680681585849985263146587808661680073326467683020639169720306489440562819540619068524200305346315662189132730906968735318164109451428803660599522024824888671155442910472192913424834643870536850864874909917881267056566538719104972182004237149274016446094345984539253670613221061653308566202118896823400575267548610147699368873820958455221157193247968688816085363161586288015039594941852948922707441082820716930338781808493620401825522227101098565344481720747075601924591559978

2STACK$(5) = 2^{65536}$ 24

535630241812254732660933027103979680910649392727226830354104676325913552796838377050198552346212228584105571199217317179698043393177077507556270560478317798444476375602546370333692471142208155199736913719751632413027487121998634045482485245701185533426752647159783107312456634298052214554945615252724028915333543493412178620370072603152798707718724912344944771479095207347613854254853115527733010303424768358654960937223240071545181297326920810584240905577256458036814622344931897081388971432998313476177996797124537823107073791514738786921191875667003193212818968033226965944592862106074388274169194651622676325406650708810710303941788605648937698167341590259251946118236429456526693722031555047002135988462927580125277154220166299548631302491231102962792372389976641680349714122652793190763632613681414551637665655983978848938173308266877990196288693229659737995193162118721545528739417024366988559388879331674453336311954151840408828381519342123412282003095031334105070476015998798547252919066522247931971544033179483683737322082188577334162385644138070054191353024594391350255453188645479625226025176292837433046510236105758351455073944333961021622967546141578112719700173861149427950141125328062125477581051297208891415781127197001738611494279501411253280621254775810512972088