L27: Disjoint Sets + More MST

Alex Steiger
CompSci 201: Spring 2024
4/22/2024

1

Logistics, coming up

- Today, Monday, 4/22
  - Project P6: Route (last project) due
- Extra credit! 3 surveys for 0.5% final grade each:
  - Official course evals (>70% completion)
  - End-of-semester survey (individual completion)
  - AiiCE survey (>70% completion)
  - Due 4/27 @ midnight
- Next week on Tuesday, 4/30
  - Final exam, 9 am-12pm
  - Required, comprehensive

2

Final Exam Policy Reminder

- Final exam composed of 3 parts:
  - F1, F2, F3 corresponding to 3 midterms M1, M2, M3.
  - Final Exam Grade: F1 + F2 + F3
  - Midterm Exam i (=1,2,3) Grade: Max(Fi,Mi)
- The four exam grades compose 11% of overall course grade each
  - Due to replacement policy, the final may compose up to 44% of your course overall (replace all 3 midterm grades)
- May bring three 8.5"x11" double-sided reference sheets
- Any questions on MSTs, disjoint sets, later material **are extra credit** on final exam grade (expect a few!)

3

Final Grade Estimates

- By this weekend, ***all*** grades should be on Canvas
  - (Aiming to get most up on Thursday, ideally all)
- Will provide a final grade ***estimate*** with a 0% on final
- Will announce ASAP when these are ready

4

Today's Agenda

1. Review Minimum Spanning Tree (MST) problem and Kruskal's Algorithm
2. Investigate efficient disjoint sets / union find data structure
3. (Time-permitting) ***Euclidean*** Minimum Spanning Trees

5

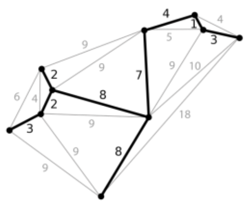
Minimum Spanning Tree (MST) and Greedy Graph Algorithms

6

Minimum Spanning Tree (MST) Problem

- Given N nodes and M edges, each with a weight/cost…
- Find a set of edges that connect *all* the nodes with minimum total cost (will be a tree)

Weighted undirected graph with:
- Edges labeled with weights/costs
- Minimum spanning tree highlighted



7

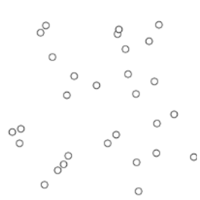
Greedy Optimization Again: Kruskal's Algorithm

- Initialize?
  - All nodes in ***disjoint sets***
- Partial solution?
  - Forest of spanning trees in disjoint sets
- Greedy step?
  - Choose the cheapest / least weight edge that connects two disjoint sets / trees, connect them.

8

Visualizing Kruskal's Algorithm

In the visualization:
- Edges between all pairs of vertices
- Weights are implicit by distances
- Algorithm greedily grows by cheapest edge that connects disjoint sets/trees.



9

Kruskal's Algorithm in *Pseudocode*

Input: N node, M edges, M edge weights
- Initialize MST as empty set
- Let S be a collection of N ***disjoint sets***, one per node
- While S has more than 1 set:
  - Let (u, v) be the minimum cost remaining edge
  - *Find* which sets u and v are in. If different sets:
    - *Union* the sets together
    - Add (u, v) to MST
- Return MST

10

Kruskal's Algorithm Runtime?

Input: N node, M edges, M edge weights
- Initialize MST as empty set
- Let S be a collection of N ***disjoint sets***, one per node
- While S has more than 1 set:
  - Let (u, v) be the minimum cost remaining edge
  - *Find* which sets u and v are in. If different sets:
    - *Union* the sets together
    - Add (u, v) to MST
- Return MST

Looping over (worst case) all M edges

Remove from binary heap, O(log(M))

Overall: O(M(log(M)+C)) where C is time for Union/Find

11

Disjoint Sets and Union-Find

DIYDisjointSets implementation viewable here:
coursework.cs.duke.edu/cs-201-spring-24/diydisjointsets

12

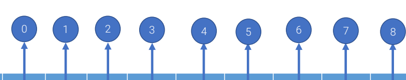
Union-Find Data Structure

- AKA Disjoint-Set Data Structure
- Start with N distinct (disjoint) sets
  - consider them labeled by integers: 0, 1, …
- *Union* two sets: create set containing both
  - label with one of the numbers
- *Find* the set containing a number
  - Initially self, but changes after unions

13

Disjoint-Set Forest Implementation

- Each set will be represented by a parent "tree": Instead of child pointers, nodes have a parent "pointer".
- Everything starts as its own tree: a single node

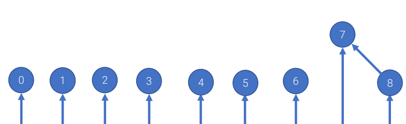


parent									
itemID	0	1	2	3	4	5	6	7	8

14

Disjoint-Set Forest Union

- Union(7,8)
- Make root parent[8] point to root parent[7]

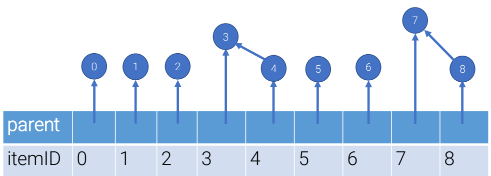


parent									
itemID	0	1	2	3	4	5	6	7	8

15

Disjoint-Set Forest Union

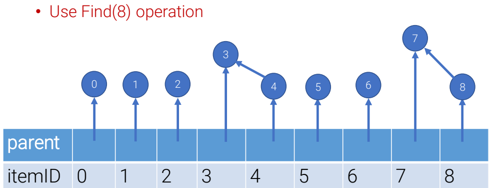
- Union(3,4)
- Make root parent[4] point to root parent[3]



16

Disjoint-Set Forest Union

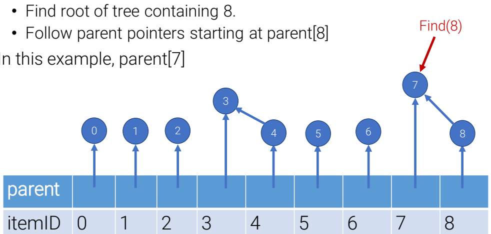
- Union(3,8)
- parent[8] is not the root anymore—Need to find its root first
  - Use Find(8) operation



17

Disjoint-Set Forest Find

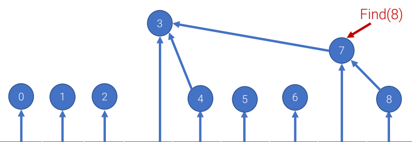
- Find(8):
  - Find root of tree containing 8.
  - Follow parent pointers starting at parent[8]
- In this example, parent[7]



18

Disjoint-Set Forest Find

- Back to Union(3,8)
  - Set root of parent[8], which is Find(8) = parent[7], to root parent[3]

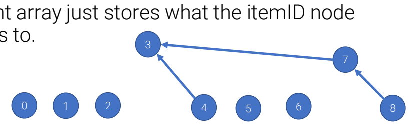


parent									
itemID	0	1	2	3	4	5	6	7	8

19

Disjoint-Set Forest Array Representation

- The “nodes” and “pointers” are just conceptual – can represent with a simple array, like binary heap.
- Parent array just stores what the itemID node points to.



parent	0	1	2	3	3	5	6	3	7
itemID	0	1	2	3	4	5	6	7	8

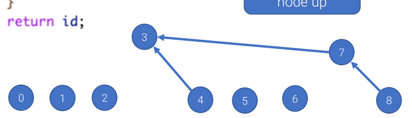
20

Disjoint-Set Forest Find

```
public int find(int id) {
    while (id != parent[id]) {
        id = parent[id];
    }
    return id;
}
```

root is just when parent[i] = i

Else go to next “node up”



parent	0	1	2	3	3	5	6	3	7
itemID	0	1	2	3	4	5	6	7	8

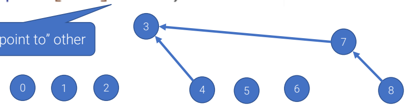
21

Disjoint-Set Forest Union Revisited

```java
public void union(int set1, int set2) {
    int root1 = find(set1);
    int root2 = find(set2);
    parent[root2] = root1;
```

roots from initial set1 and initial set2 "nodes"

Make one "point to" other



parent	0	1	2	3	3	5	6	3	7
itemID	0	1	2	3	4	5	6	7	8

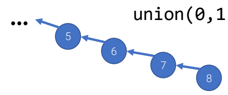
22

Worst-Case Runtime Complexity?

```java
public void union(int set1, int set2) {
    int root1 = find(set1);
    int root2 = find(set2);
    parent[root2] = root1;
```

What if we…
`union(7,8)`
`union(6,7)`
`union(5,6)`
…
`union(0,1)`

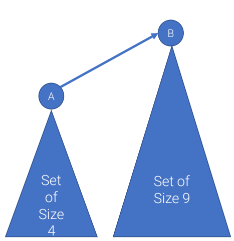
Now `find(8)` would have linear runtime complexity!!



parent	0	0	1	2	3	4	5	6	7
itemID	0	1	2	3	4	5	6	7	8

23

Optimization 1: Union by Size

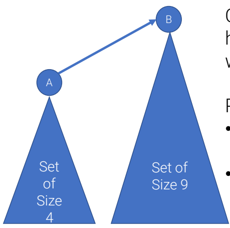


Be careful in how you union. Always make the "root" for the set with *fewer* elements point to the "root" for the set with *more* elements.

Sufficient for worst case logarithmic efficiency.

24

Optimization 1: Union by Size



Claim. Each element to root path has length at most O(log(N)) with union by size optimization.

Proof.
- Consider an element a, initially a set of size 1.
- Each time the path length increases, the size of the set must at least double.
- Can happen at most O(log(N)) times with N initial sets.

25

Optimization 1: Union by Size

```
public void union(int set1, int set2) {
    int root1 = find(set1);
    int root2 = find(set2);
    if (root1 == root2) { return; }
    if (setSizes[root1] < setSizes[root2]) {
        parent[root1] = root2;
        setSizes[root2] += setSizes[root1];
    }
    else {
        parent[root2] = root1;
        setSizes[root1] += setSizes[root2];
    }
    size--;
}
```

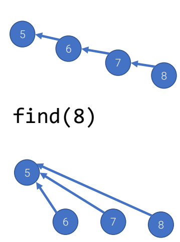
If already in same set, nothing to do.

Make the smaller set "point to" the bigger set.

26

Lazy Path Compression

- **Lazy path compression**: When ever you traverse a path in **find**, connect all the pointers to the top.
- Sufficient for **amortized logarithmic** runtime complexity for union/find operations.



27

Disjoint Set Forest Path Compression

```java
public int find(int id) {
    int idCopy = id;
    while (id != parent[id]) {
        id = parent[id];
    }
    int root = id;
    id = idCopy;
    while(id != parent[id]) {
        parent[idCopy] = root;
        id = parent[id];
        idCopy = id;
    }
    return id;
}
```

Get the root as before

Traverse path again, assigning everything to the root

28

Optimized Runtime Complexity

- Optimizations considered separately:
  - Union by size: *Worst-case* logarithmic
  - Path compression: *Amortized* logarithmic
- Considered together…?
  - Worst-case logarithmic, and *amortized inverse Ackermann function* $\boldsymbol{\alpha(n)}$
  - $\alpha(n) < 5$ for $n < 2^{2^{2^{2^{16}}}} = 2^{2^{65536}}$
  - *Number of atoms in observable universe only* $\sim\mathbf{10^{80}}$
  - Practically constant for any n you can write down

29

Remember Kruskal's Algorithm Runtime?

Input: N node, M edges, M edge weights

- Let MST to an empty set
- Let S be a collection of N **disjoint sets**, one per node
- While S has more than 1 set:
  - Let (u, v) be the minimum cost remaining edge
  - **Find** which sets u and v are in. If different sets:
    - **Union** the sets
    - Add (u, v) to MST
- Return MST

Looping over (worst case) all M edges

Remove from binary heap, O(log(M))

O(M(log(M)+C) = O(M log M) because C < log(M) for our optimized union find

30

L27-WOTO1-DisjointSets-Sp24

Hi, Alexander. When you submit this form, the owner will see your name and email address.

* Required

1

NetID *

solutions

2

Suppose we initialize a disjoint sets data structure with 10 sets (numbered 0 through 9), then do the following operations:

union(0, 1)
find(1)
union(2, 3)
union(0, 4)
union(4, 5)
union(1, 5)

How many disjoint sets remain / what is the size of the data structure at this point? *

- ◯ 3
- ◯ 4
- ◯ 5
- ◉ 6

3

Consider the following array representation of a disjoint sets data structure. **What would be returned by find(5)?** *

parent	4	4	2	2	4	1	6	7	8	9
itemID	0	1	2	3	4	5	6	7	8	9

- ◯ 1

- [x] 4
- [ ] 5
- [ ] None of the above

4

Consider the same array representation of a disjoint sets data structure as the previous problem. **How many sets have a single element?** *

parent	4	4	2	2	4	1	6	7	8	9
itemID	0	1	2	3	4	5	6	7	8	9

- [ ] 2
- [ ] 3
- [x] 4
- [ ] 5

5

Consider the same array representation of a disjoint sets data structure. Suppose we **union(3,5)**. Which of the following updates would be performed under union by size optimization? *

parent	4	4	2	2	4	1	6	7	8	9
itemID	0	1	2	3	4	5	6	7	8	9

- (selected) Change the itemID 2 parent value to 4
- Change the itemID 3 parent value to 4
- Change the itemID 3 parent value to 5
- Change the itemID 5 parent value to 2
- Change the itemID 5 parent value to 3

6

Select all that are true of the amortized runtime complexity of union/find operations on a disjoint sets forest data structure with union by size and path compression optimizations. *

- [x] Constant for n up to trillions
- [x] Constant for n up to the number of grains of sand on earth
- [x] Constant for n up to the number of seconds that have elapsed since the big bang
- [x] Constant for n up to the number of stars in the known/observed universe
- [ ] Constant in the limit as n --> infinity

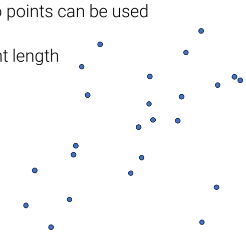
Extensions of MST

- We described $O(M \log N)$-time for "offline" case
  - Entire input is given upfront (like all algs. in 201)
  - $O(M\alpha(N))$ is possible, avoids sorting [Chazelle '99]
- *Dynamic* MST: How quickly can an MST be ***updated*** as the result of:
  - Insertion of a new edge?
  - Deletion of an edge?
- Need to recompute the entire MST from scratch?
  - No! O(M+N) time suffices via BFS/DFS
  - With advanced data structures, O(log N) possible

32

Euclidean MST

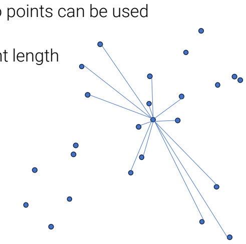
- Given N points in the plane:
  - Find spanning tree T of the points
  - *Any* segment between two points can be used
    - # of edges is $O(N^2)$
    - Edge weight = segment length



33

Euclidean MST

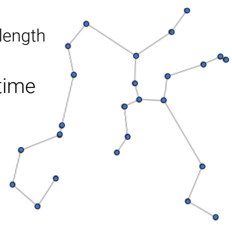
- Given N points in the plane:
  - Find spanning tree T of the points
  - *Any* segment between two points can be used
    - # of edges is $O(N^2)$
    - Edge weight = segment length



34

Euclidean MST

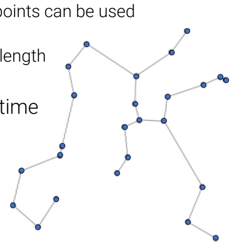
- Given N points in the plane:
  - Find spanning tree T of the points
  - *Any* segment between two points can be used
    - # of edges is $O(N^2)$
    - Edge weight = segment length
- $O(M \log N) = O(N^2 \log N)$ time
- Does geometry help?
  - Do we need to consider every possible edge?



35

Euclidean MST

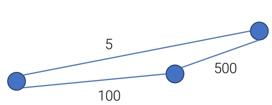
- Given N points in the plane:
  - Find spanning tree T of the points
  - *Any* segment between two points can be used
    - # of edges is $O(N^2)$
    - Edge weight = segment length
- $O(M \log M) = O(N^2 \log N)$ time
  - $O(N \log N)$ possible!
- Does geometry help?
  - Do we need to consider every possible edge?



36

Geometric Structure

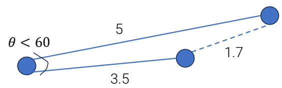
- Intuition: The corresponding graph is not arbitrary
  - Edges cannot be just anything --- they are exactly their distance measured in the plane (ex. below is absurd)



37

Geometric Structure

- Intuition: The corresponding graph is not arbitrary
  - Edges cannot be just anything --- they are exactly their distance measured in the plane



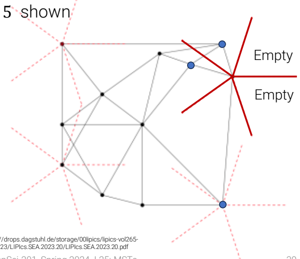
- Example of useful structure:
  - Any two incident edges must make ≥60° angle
    - If <60° angle, opposite edge is shorter than one of the incident edges; use it instead

38

Yao Graph

- *k-Yao Graph*: For each point, include only segments to closest neighbor in each of ***k*** slices
  - Example with $k = 5$ shown

Theorem: The 6-Yao Graph contains the Euclidean MST (each slice is 60°)



https://drops.dagstuhl.de/storage/00lipics/lipics-vol265-sea2023/LIPIcs.SEA.2023.20/LIPIcs.SEA.2023.20.pdf

39

Putting It All Together

- Any Euclidean MST (EMST) makes angles >60°
- The 6-(slice) Yao Graph contains the EMST and has only $O(N)$ edges
- Improved algorithm:
  - Compute the Yao Graph in $O(N \log N)$ time [Chang et al. '90]
  - Run Kruskal's on the graph in only $O(N \log N)$ time
- Much faster than $O(N^2 \log N)$!

40

Punchline

- Realistic settings have additional constraints
- Sometimes can be exploited to give better solutions than those for more general settings

- Take Alex's class on Applied Computational Geometry!
  - CS 290, Fall 2024

41