CS 6170: Computational Topology, Spring 2019
Lecture 04

Topological Data Analysis for Data Scientists

Dr. Bei Wang

School of Computing
Scientific Computing and Imaging Institute (SCI)
University of Utah
www.sci.utah.edu/~beiwang
beiwang@sci.utah.edu

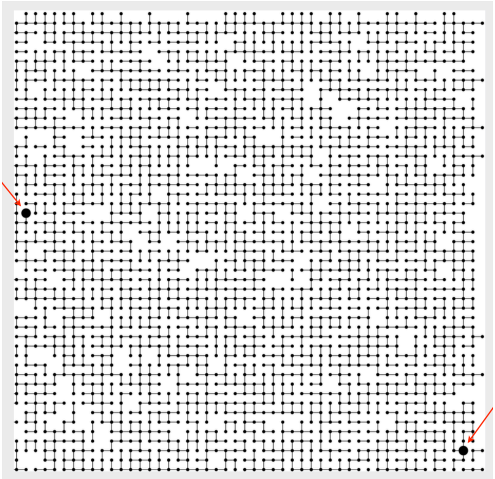
Jan 17, 2019

Union-Find (Disjoint Set Data Structure)

Book Chapter A.I.

Union and find for large network connectivity

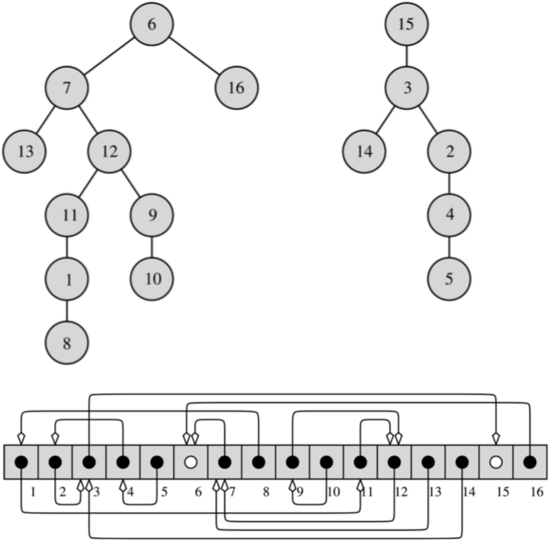
- *Union command*: connect two objects
- *Find query*: used to decide if there is a path connecting two objects.



https://www.cs.princeton.edu/~rs/AlgsDS07/01UnionFind.pdf

Store a tree in a linear array

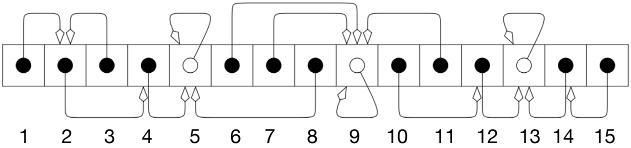
Store two trees in a single linear array using arbitrary ordering of the nodes.



Edelsbrunner and Harer (2010)[Page 7]

Store a tree in a linear array: quiz

Can you recover three trees (disjoint sets) from the following linear array?



https://www2.cs.duke.edu/courses/fall06/cps296.1/Lectures/sec-I-1.pdf

MakeSet

```
function MakeSet(x)
  if x is not already present:
    add x to the disjoint-set tree
    x.parent := x
    x.rank   := 0
    x.size   := 1
```

https://en.wikipedia.org/wiki/Disjoint-set_data_structure

Find

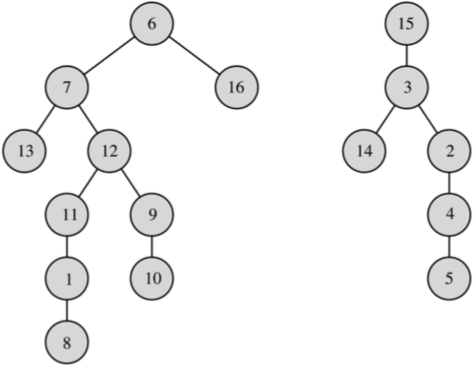
```
int FIND(i)
  if V[i].parent ≠ null then return FIND(V[i].parent)
                        else return i
  endif.
```

Edelsbrunner and Harer (2010)[Page 7]

Find

int $\textsc{Find}(i)$
  if $V[i].parent \neq$ null then return $\textsc{Find}(V[i].parent)$
                                    else return i
  endif.

Exercise: Find(9) = ?



Edelsbrunner and Harer (2010)[Page 7]

Find via path compression

```
int FIND(i)
  if V[i].parent ≠ i then
    return V[i].parent = FIND(V[i].parent)
  endif;
  return i.
```

Edelsbrunner and Harer (2010)[Page 8]

Union and **Union** by size

```
void UNION(i, j)
  x = FIND(i);  y = FIND(j);
  if x ≠ y then V[x].parent = y endif.
```

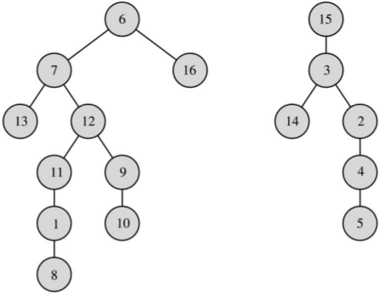
```
void UNION(i, j)
  x = FIND(i);  y = FIND(j);
  if x ≠ y then
    if V[x].size > V[y].size then x ↔ y endif;
    V[x].parent = y
  endif.
```

Edelsbrunner and Harer (2010)[Page 8]

Union by size

```
void UNION(i, j)
  x = FIND(i);  y = FIND(j);
  if x ≠ y then
    if V[x].size > V[y].size then x ↔ y endif;
    V[x].parent = y
  endif.
```

Exercise: Union(8, 4) = ?



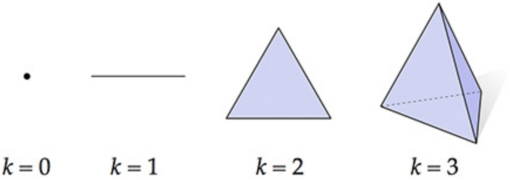
https://en.wikipedia.org/wiki/Disjoint-set_data_structure

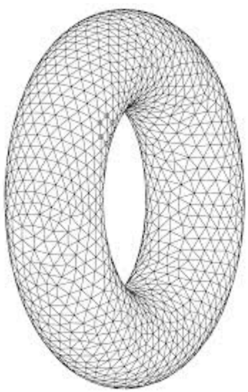
Simplicial Complexes

Book Chapter A.III

Simplex

0-simplex is a vertex, 1-simplex is an edge, 2-simplex is a triangle, 3-simplex is a tetrahedron, a 4-simplex is a 5-cell....





Simplices

Definition (Simplex)

A *k-simplex* is the convex hull of $k+1$ affinity independent points, $\sigma = \text{conv}\{u_0, u_1, ..., u_k\}$.

- Let $u_0, u_1, ..., u_k \in \mathbb{R}^d$
- A point $x = \sum_{I=0}^{k} \lambda_i u_i$ is an *affine combination* of the u_i if the $\lambda_i \in \mathbb{R}$ sum to 1. It is a *convex combination* if all $\lambda_i \geq 0$.
- The $k+1$ points are *affinely independent* iff the k-vectors $u_i - u_0$ are linearly independent (for $1 \leq i \leq k$).
- A *convex hull* is the smallest convex set that contains the points.
- A *convex hull* is the set of convex combinations.

Simplicial complexes

- A *face* τ of σ is the convex hull of a non-empty subset of the u_i, denoted as, $\tau \leq \sigma$.
- σ is the *coface* of τ.
- A face is *proper*, i.e., $\tau < \sigma$, if the subset is not the entire set.

Definition (Simplicial Complex)

A *simplicial complex* is a finite collection of simplicies K such that $\sigma \in K$ and $\tau \leq \sigma$ implies $\tau \in K$, and $\sigma, \sigma_0 \in K$ implies $\sigma \cap \sigma_0$ is either empty or a face of both.

Čech complexes and Vietoris-Rips complexes

Book Chapter A.III

Demo:

http://www.sci.utah.edu/~tsodergren/prob_net_vis_working/

References I

Edelsbrunner, H. and Harer, J. (2010). *Computational Topology: An Introduction*. American Mathematical Society, Providence, RI, USA.