Introduction to Graph Algorithms

This Class

Contact and Office Hours

Office Hours

- Monday – Thursday, 10 am – 11 am
  or on appointment.
- Room 107, Hebeler Hall

Email

- `arne.leitert@cwu.edu`

Course Requirements

Lab Assignments	50 %
Quizzes (including Final)	50 %

Dates will be posted later on canvas.

Quizzes

- closed book
- 5 questions, each 20 %
- List of potential questions given in advance
- Usual case: Given a graph, run a certain algorithm on it.

Lab Assignments

- (mostly) weekly
- implementing algorithms or similar in Java
- labs: mostly independent work

Feedback

Feedback

- Class is not perfect.
- Your (constructive) feedback is appreciated (and needed) to improve the class.
- Via any way you like: directly, email, evaluations at end of quarter, via other professors

What You Should Know

Array Based Lists

Add(e) (amortised) $\mathcal{O}(1)$

- Adds an element e to the end.

AddAt(e, i) $\mathcal{O}(n)$

- Adds an element e at index i. Other elements are shifted.

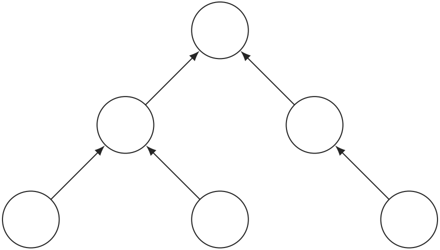
Get(i) / Set(i, e) $\mathcal{O}(1)$

- Reads or overrides the element at index i.

Find(e) $\mathcal{O}(n)$

- Finds the first element equal to e.

Trees



Enqueue $\mathcal{O}(\log n)$

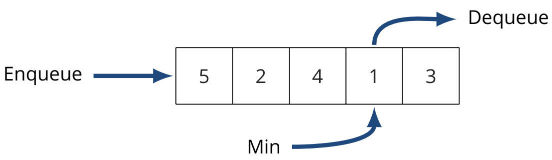
- Adds an element to the queue.

Dequeue $\mathcal{O}(\log n)$

- Removes the *smallest* element in the queue.

Min $\mathcal{O}(1)$

- Return the *smallest* element in the queue without removing it.



Sorting

Array Based Sorting

- In $\mathcal{O}(n \log n)$ time.
- Requires $\mathcal{O}(\log n)$ additional space.
- Not stable.

5	2_a	4	6_a	3	1	6_b	2_b

Input

1	2_b	2_a	3	4	5	6_a	6_b

Output

Hash Tables

Insert($\boldsymbol{k}, \boldsymbol{v}$) $\mathcal{O}(1)$

- Inserts a key-value pair (k, v).

Delete($\boldsymbol{k}$) $\mathcal{O}(1)$

- Deletes a value with the given key k.

Find($\boldsymbol{k}$) $\mathcal{O}(1)$

- Finds a value with the given key k.

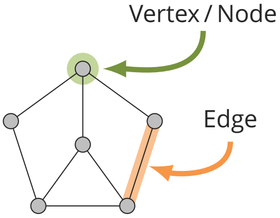
Note: Elements cannot be sorted within the table.

Graphs

Graph

Graph

A *graph* $G = (V, E)$ is a set V of vertices connected by an edge set E.



If not explicitly defined differently, we let $n = |V|$ and $m = |E|$.

Variations

Multi-Graph: Multiple edges between two vertices.

Directed: Edges have a direction.

Weighted: Vertices and/or edges have weights.

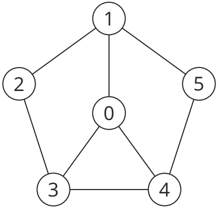
Simple: No multiple edges, no loops.

Simple Undirected Graph

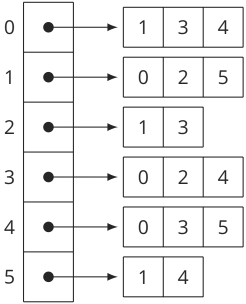
A *simple undirected graph* $G = (V, E)$ is a set V of vertices connected by an edge set $E \subseteq \big\{ \{u, v\} \mid u, v \in V, u \neq v \big\}$. An edge $\{u, v\}$ is usually written as uv.

Implementation: Adjacency List

For each vertex, there is an array storing “pointers” to all neighbours.
(Usually, the vertex index is sufficient.)



G



Example Problem

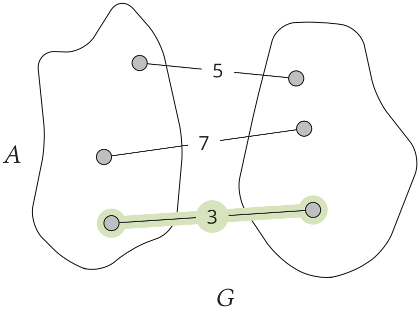
Example Problem

Let $G = (V, E, \omega)$ be an undirected weighted graph and let A be a non-empty proper subset of V. The *separation* $\operatorname{sep}(A)$ is defined as $\operatorname{sep}(A) = \min\{\, \omega(ab) \mid a \in A, b \in V \setminus A \,\}$. Given a graph G, find a subset A with maximum possible separation.

Understanding the Problem

Separation

Let $A \subset V$ be non-empty. The *separation* $\mathrm{sep}(A)$ is defined as $\mathrm{sep}(A) = \min\{\, \omega(ab) \mid a \in A, b \in V \setminus A \,\}$.



Problem

- Find the set A for which $\mathrm{sep}(A)$ is maximal.

Finding a First Solution

Naive Approach

- Test all subsets A
- Problem: Too many subsets.

Observation

- A solution is defined by an edge.

Finding a First Solution

Better Approach

- For each edge e, check if there is a set A with $\mathrm{sep}(A) = \omega(e)$.
- How do we check this?

Observation

- If $\mathrm{sep}(A) = \omega(e)$, then, for each edge uv with $\omega(uv) < \omega(e)$, $u \in A$ if and only if $v \in A$, i. e., both are in A or bot are not in A.

Lemma

There is a set A with $\mathrm{sep}(A) = \omega(uv)$ if and only if there is no path from u to v where each edge has a lower weight than uv.

Finding a First Solution

Algorithm Idea

- For each edge uv, check if there is path from u to v only using edges with less weight than uv.
- If there is no such path, store uv as potential solution.
- Out of all these edges uv, pick the one with the largest weight.

Runtime

- $\mathcal{O}\left(m^2\right)$

This is an acceptable solution. However, can we do better?

Improve a Solution

A defines a partition of the graph and $\mathrm{sep}(A)$ is represented by the smallest edge connecting bots sets. What else do we know about this edge?

Improve a Solution

A defines a partition of the graph and $\text{sep}(A)$ is represented by the smallest edge connecting bots sets. What else do we know about this edge?

Lemma

If e is in $\{ ab \mid a \in A, b \in V \setminus A \}$ and $\omega(e) = \text{sep}(A)$, then there is a minimum spanning tree containing e.

New Algorithm

- Compute a minimum spanning tree T.
- Find the edge e of T with the largest weight.
- Runtime (using Prim's algorithm): $\mathcal{O}(m \log n)$

Question

- Do we really need a MST?

Improve a Solution

Minimum Spanning Tree

- The tree with the smallest sum of edges.

Observation

- We only want the largest edge e of an MST.
- If we remove all edges e' from G with $\omega(e) \leq \omega(e')$, G is disconnected.

Can we find e without finding the MST first?

Improve a Solution

Minimum Spanning Tree

- The tree with the smallest sum of edges.

Observation

- We only want the largest edge e of an MST.
- If we remove all edges e' from G with $\omega(e) \leq \omega(e')$, G is disconnected.

Can we find e without finding the MST first?

- Related problem: Minimum Bottleneck Spanning Tree
- Can be implemented in (expected) linear time.
- Uses quick-select algorithm as basic strategy.
- Not easy to implement correctly.