02-613 Week 1

Algorithms and Advanced Data Structures

Aidan Jan

August 29, 2025

Logistics

Dates

- Midterms (7pm - 9pm):
  - Wed. Oct. 1
  - Wed. Nov. 12
- Recitations:
  - Wed. 5pm (Grad)
  - Thurs. 3pm (UG)
  - Fri. 1pm (UG)

Grading

There are two grading schemes:

- Scheme 1:
  - 20% Homework
  - 5% Recitation Participation
  - 75% Exams
- Scheme 2:
  - 100% Exams

Grading is done by module, so you can pick which scheme for each module.

Homework

- 1 Homework assignment per week
- 4-5 Oral homework in semester
- Collaboration is allowed, but no sharing answers.
- No generative AI.

Graphs

- An undirected graph is defined as $G = (V, E)$, with V being the vertices (a.k.a. Nodes), and E being the edges. Vertices are a set of objects and Edges are a set of connections between objects.
  - $V = \{v_1, v_2, \ldots, v_n\}$
  - $E = \{e_1, e_2, \ldots, e_n\}$. $e \in E \ : \ e = \{u, v\}, u, v \in V$
- In an undirected graph, all edges are bidirectional. Suppose, $e_1 = \{u, v\}$, $e_2 = \{v, u\}$. In an undirected graph, $e_1 = e_2$.
- In a directed graph, edges are one-way. In this case, $e_1 \neq e_2$.

Subgraphs

Let $H = (V_H, E_H)$, a subgraph of G. This implies that $V_H \subseteq V$ and $E_H \subseteq E$. Additionally, it is required that $\forall e \in E, e = \{u, v\}$, and $u, v \in V_H$.

Connected Graphs

A *connected graph* is a graph where every vertex can take a path consisting of one or more edges to every other vertex.

- A *maximal connected subgraph* is the largest connected subgraph with the most vertices.

Cycles

A *cycle* in graph $G = (V, E)$ is a sequence or distinct vertecies $v_1, \ldots, v_k \in V$ such that $\{v_i, v_{i+1}\} \in E$ and $\{v_k, v_1\} \in E$.

- A *tree* is a graph that is connected and has no cycles.

Minimum Spanning Trees (MST)

- Given a graph, find a set of edges that connects all of the nodes which minimizes the total cost.
- For example, low-cost wiring of a computer network. Minimize the distance of wires between all the computers.
- Formally, given an undirected graph G with edge costs $d(e) = d(u, v) > 0$, find a subgraph T that connects all nodes of G that minimizes $\text{Cost}(T) \sum_{e \in T} d(e)$

Greedy MST Algorithms

There are three main algorithms used to find the MST:

- **Prim's**: Choose lowest edge on the cut and add it to the graph, repeat until all the nodes are connected.
- **Reverse-Delete**: Start with full graph, and remove edges in decreasing order of weights unless it splits the graph.
- **Kruskal's**: Start with all the nodes, and add edges in increasing order, unless it creates a cycle.

Prim's Algorithm

One solution is Prim's algorithm.

- Given $G = (V, E)$, $|V| = n$, and $|E| = m$, select an arbitrary node. Make that the start of T.
- While there exists at least one node not in T, add the lowest edge from something in T to something not in T.
  - Since there are n nodes in the graph, this loop will execute exactly $n - 1$ times.

Theorem: MST Cut Property

Let $S \subset V$, $|S| \geq 1$, $|S| < |V|$. Every MST contains the edge $e = \{u, v\}$ with $v \in S$, $u \in V \backslash S$ or in a weight. A pair $(S, V \backslash S)$ is a cut of the graph.

- Suppose T is the MST, and $e \notin T$ for cut S, but $e' \in T$. In essence, e has a lower cost than e', but the tree contains e' over e. If $d(e) \neq d(e')$, and $d(e) \leq d(e')$, then T is not a minimum spanning tree, and the weight can be lowered by replacing e' with e. Therefore, by contradiction, the assumption that $e \in T$ in every MST must be true.

Proof of Prim's Algorithm

First we want to check correctness.

- The subgraph should contain all nodes
- There should be no cycles
- The subgraph is connected

Let T be a subgraph over all the nodes. Since every step adds 1 node for $n - 1$ steps, plus the initial node, the subtree will contain all the nodes. Since there are exactly $n - 1$ edges, this implies that there are no cycles. Using the MST Cut Property, this must be the minimum spanning tree, since each edge added per step is on a cut of the graph, and therefore must be in the MST.

Reverse Delete Algorithm

The Reverse Delete Algorithm also produces an MST. It goes as follows:

- Start with the full graph
- Remove edges in decreasing order of weights unless it splits the graph

MST Cycle Property

A different property of MSTs allow us to develop the Reverse Delete Algorithm, and that is the Cycle property. It states:

- Let C be a cycle in G.
- Let $e = (u, v) \in C$ be the edge of the cycle with the maximum weight.
- e can not be in the MST.

Proof of the Reverse Delete Algorithm

- First, we want to show the product is a tree. This is trivial; a tree is a graph such that there exists only one path between any two nodes, therefore any edge removed from a tree would disconnect it. If an edge is not removed by the algorithm, it means that the graph would be disconnected. Therefore, the product must be a tree.
- Next, to show the product is the MST, consider a given edge removed from the tree. If the edge was removed, two things are required: (1) the edge is present in a cycle, therefore removing it will not disconnect the graph, and (2), the edge in question must have the largest weight in said cycle, since edges are removed in reverse order of weights.
  - By the MST cycle property, the edge removed is definitely not in the MST. Since edges are removed until there are no cycles, and all remaining edges are required for the graph to be connected, it must be the MST.

Kruskal's Algorithm

This algorithm goes as follows:

- Start with all the vertices and none of the edges of the graph.
- Add edges in increasing order of weights, unless it creates a cycle.

Proof of Kruskal's Algorithm

- First, consider the edges added to the algorithm. If the edge is added, it must not create a cycle. Therefore, the product must be a tree which connects all nodes.
- When an edge is added, it must not create a cycle. If it is not added, it must be the maximum in the cycle it forms, and by the cycle property the rejected edge is not in the MST.
- Therefore, if the graph only consists of edges that are not the maximum of the cycles, the product must be a MST.

Storing a Graph

How do we store a graph? There are generally three ways:

- Adjacency matrix
- List of edges represented as tuples
- Branched linked list with an abstract data type (a class/struct)
  - An abstract data type is simply a description of the data's structure. It says nothing about how it is stored in a computer.
  - For example, a queue is an abstract data type. However, the concept can be implemented in a computer using an array, even though an array is not a queue.

Storing a graph for Kruskal's Algorithm

Importantly, we want the structure to allow for

- adding and removing edges quickly
- easily searchable to determine which group (in graph or out of graph) a given edge is in
- easily combine two groups

Technically, we want a

- Find(i), where i is an object. Returns the group i is in.
- Union(a, b), which puts items from groups a and b together.
- Constructor, which instantiates the object.