1 Priority Queue Implementations

We will go through some prioriy queue implementations.

1.1 Unsorted List

Store n elements in an unsorted list.

Operations:

- `Insert`: $\mathcal{O}(1)$.
- `DecreaseKey`: $\mathcal{O}(1)$.
- `DecreaseMin`: $\mathcal{O}(1)$.

For Dijkstra, we would have $\mathcal{O}(|V|^2 + |E|)$.

1.2 Binary Heap

Store elements in a balanced binary tree with each element having smaller key value than its children.

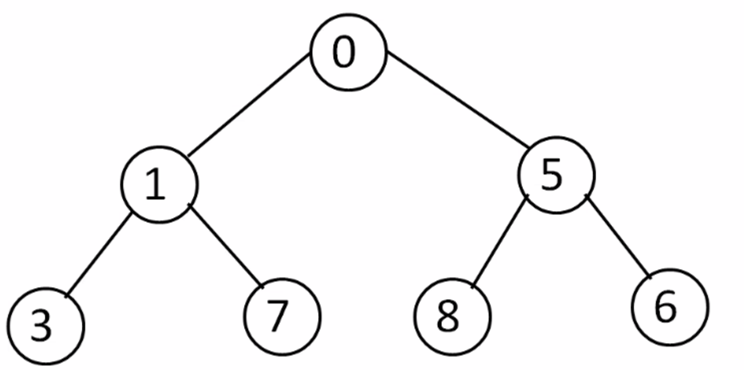

Figure: A binary heap.

The smallest key is at the top (`0`) and there are $\log n$ levels.

Operations:

- `Insert`: Add the key at the bottom, then bubble the new key up until it's in the right place. This is done in $\mathcal{O}(\log(n))$ time.
- `DecreaseKey`: We need to change the key. Then, we might need to bubble up the changed key until it's in the right place. This is done in $\mathcal{O}(\log(n))$ time.
- `DecreaseMin`: We remove and then return the root node. Then, we move the bottom-most node to the root. After this, we might need to continuously bubble down the root node until it's in the right place. This is done in $\mathcal{O}(\log(n))$ time.

For Dijkstra, we would have $\mathcal{O}(\log(|V|)(|V| + |E|))$.

1.3 d-ary Heap

This is like a binary heap, but each node has d children.

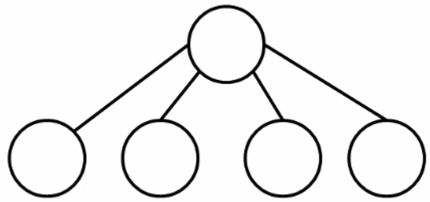

Figure: A 4-ary heap.

There are $\log(n)/\log(d)$ levels, so bubble up is faster. However, bubble down is slower since we need to compare more children.

Operations:

- `Insert`: This is done in $\mathcal{O}(\log(n)/\log(d))$ time.
- `DecreaseKey`: This is done in $\mathcal{O}(\log(n)/\log(d))$ time.
- `DecreaseMin`: This is done in $\mathcal{O}(d\log(n)/\log(d))$ time. This is because, for bubble down, we need to consider the d children.

For Dijkstra, we would have $\mathcal{O}\left(\frac{\log(|V|)(d|V|+|E|)}{\log(d)}\right)$.

1.4 Fibonacci Heap

This is an advanced data structure that uses amortization[1].

Operations:

- `Insert`: This is done in $\mathcal{O}(1)$ time.
- `DecreaseKey`: This is done in $\mathcal{O}(1)$ time.
- `DecreaseMin`: This is done in $\mathcal{O}(\log(n))$ time. This is because, for bubble down, we need to consider the d children.

For Dijkstra, we would have $\mathcal{O}(|V|\log(|V|)+|E|)$.

2 Negative Edge Weights

So far, we've talked about non-negative lengths. However, depending on what we're representing as lengths, we might have *negative* lengths. That being said, the problem statement is the same - find the path with the smallest sum of edge weight.

Right now, Dijkstra's algorithm doesn't actually work on negative edge values.

[1]So, you might spend more time on a particular operation, but the overall runtime will be "consistent."

2.1 Negative Weight Cycle

Definition 2.1

A **negative weight cycle** is a cycle where the total weight of edges is negative.

Remarks:

- If G has a negative weight cycle, then there are probably no shortest paths since we can go around the cycle over and over again.
- For an undirected graph G, a single negative weight edge gives a negative weight cycle by going back and forth on it. So, we usually don't talk about the negative edge weight in the context of an undirected graph.