**Dijkstra Heap**

\[ X[v] = 0; \ A[v] = \infty; \ B[v] = \emptyset; \quad \forall \ v \in V \]

\[ v.\text{key} = \infty \quad \text{for all } v \in V \]

\[ v.\text{parent} = \emptyset \quad \text{for all } v \in V \]

\[ s.\text{key} = 0. \]

Heapify all \( v \in V \) \( \{ O(n \log n) \} \)

\[ \text{while (Heap is not empty)} \]

- Let \( w \) = vertex with min key

- Remove \( w \); \( X[w] = 1; \ A[w] = w.\text{key}; \) \( \{ O(\log n) \} \)

- \( B[w] = B[w.\text{parent}] + (w.\text{parent}, w) \)

- for \( u \in \text{Adj}[w] \) \& \( u \) not explored

  - Check if need to update \( u.\text{key} \)

  - If yes, remove & reinsert

**How many times does this loop run?**

\[ O(m) \rightarrow \]

\[ (\text{see next page}) \]

**How many times does this check happen over whole algorithm?**

\[ O(\log n) \]

**What is cost?**

\[ O(\log n) \]
Only need to update if

Only gets updated when w or v gets pulled into X. Only happens once for each edge.

Adding it all up:

\[ O(n) + O(n \log n) + O(n \log n) + O(m \log n) \]

\[ \Rightarrow O((n+m) \log n) \]

Much better than FOR loop approach which was \( O(nm) \)