Lemma 3 (Path Relaxation Property)

If

\[ P: s = x_0, x_1, x_2, \ldots, x_k \]

is a min. weight \( s \rightarrow x_k \) path, and

if the edges of \( P \) are relaxed in the order:

\[ (x_0, x_1), (x_1, x_2), (x_2, x_3), \ldots, (x_{k-1}, x_k) \]

then \( d[x_k] = \delta(s, x_k) \). This is true regardless of any other relaxing that may take place, even if interspersed with above seq.
Proof: Lemma 24.15 p. 673

Induction on \( k = \# \text{ edges in a shortest path} \).

\[ \text{Bellman-Ford}(G, s) \]

1.) \text{Initialize} \((G, s)\)
2.) for \( i = 1 \) to \(|V| - 1\)
3.) for each \((x, y) \in E\)
4.) \text{Relax} \((x, y)\)
5.) for all \((x, y) \in E\)
6.) if \( d[y] > d[x] + w(x, y)\)
7.) return false
8.) return true
runtime

\[ n = |V|, \ m = |E| \]

- Initialize: \( \Theta(n) \)

- \( \text{Relax}(i, j) \) costs \( \Theta(1) \)

  - Each edge is relaxed \( n-1 \) times in total 2-4 times:
    - cost \( \Theta((n-1)m) \)

  - Total cost \( \Theta(m) \)

  total cost: \( \Theta(n) + \Theta((n-1)m) + \Theta(m) \)

  \[ = \Theta(nm) \]
Dijkstra\((G,\pi)\) Pre: edge weights \(\pi\)

1. Initialize \((G, s)\)
2. \(S = \emptyset\) // finished vertices
3. \(Q = V\) // a min-p.d. keys = d-values.
4. \(\text{while } Q \neq \emptyset\)
5. \(x = \text{ExtractMin}(Q)\)
6. \(S = S \cup \{x\}\)
7. for all \(y \in \text{Adj}(x)\)
8. \(\text{Relax}(x, y)\)

If \(Q\) is implemented as a min-heap, must re-write \(\text{Relax}(x, y)\).
\text{Relax}(x,y) : p := y \text{ if } d[y] \leq d[x] + w(x,y)

1.) if \( d[y] \geq d[x] + w(x,y) \)
2.) \text{DecreaseKey} \( (y, d[x] + w(x,y)) \)
3.) \( y = x \)

Runtime: \( n = |V|, m = |E| \).

- Initialize: \( \Theta(n) \)
- line 3 (BuildHeap): \( \Theta(n) \)
- each call to ExtractMin: \( \Theta(\log n) \)
- all executions of line 5:
  - cost \( \Theta(n \log n) \).
- Relax containing DecreaseKey
  which costs \( \Theta(\log n) \).
total cost of relaxation is $\Theta (m \log n)$.

Total cost: $\Theta(n) + \Theta(n \log n) + \Theta(m \log n)$

= $\Theta((n+m) \log n)$.

Example:

Diagram:

Problem:

Trick:
12.1 Binary Search Trees (BST)

<table>
<thead>
<tr>
<th>node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent</td>
</tr>
<tr>
<td>Key</td>
</tr>
<tr>
<td>Satellite data</td>
</tr>
<tr>
<td>left</td>
</tr>
</tbody>
</table>

BST Properties

For all nodes x

- If y is in left subtree of x, then key[y] ≤ key[x]
- If y is in right subtree of x, then key[y] ≥ key[x]
root[T] = nil indicated empty tree.
Runtime of search: $O(h)$

Where $h = \text{height}(T)$

Note:

$$1 \leq h \leq n-1$$

Algorithm:

- Search $O(h)$
- Insert $O(h)$
- Delete $O(h)$
- Tree Walks:
  - InOrder
  - PreOrder
  - PostOrder
  $O(n)$
3.1 Red-Black Trees (RBT)

A RBT is a BST satisfying:

RBT properties
1.) Each node is red or black
2.) Root is black
3.) Each leaf (nil children) is black
4.) Every red node has 2 black children
5.) For any node x, every path from root to leaf has same # of black nodes.
Note: the # black nodes from x to a leaf (not counting x) is called black height of x: bh(x).

Ex.
\begin{align*}
\text{node} & \quad \text{b.h.} \\
8 & \quad 3 \\
5, 11, 3 & \quad 2 \\
1, 2, 4, 6, 7, 9, 10, 12 & \quad 1 \\
\text{all nil leaves} & \quad 0
\end{align*}

\underline{Note:}

\[ bh(x) = 0 \quad \text{if and only if} \quad \text{height}(x) = 0 \]
\[ \text{if} \quad x \quad \text{is a leaf.} \]
A RBT with $n$ internal (i.e. non-nil) nodes and height $h$ satisfies

$$h \leq 2 \lfloor \log (n+1) \rfloor$$

Note: $h \geq \lfloor \log n \rfloor$ so for RBTs, $h = \Theta(\log n)$

Dict. op runs in time $\Theta(\log n)$.