- Dijkstra Algorithm

  \[\text{want: compute shortest paths from } S \text{ to other vertices in ascending order of distance}\]

  \[\text{initially only know } \text{dis}(S) = 0\]

  \[\text{first step: try to find a vertex closest to } S.\]

  \[\text{observation: the closest point must be a neighbor of } S.\]

  \[\text{next step: neighbors of all vertices that we have computed before}.\]

- maintain a set \( W \)

  \[\text{Property: 1. know the shortest path from } S \text{ to any } u \in W\]

  \[2. \text{distance to any } u \in W \text{ no larger than distance to any } v \notin W\]

- ACG: find a vertex \( u \) \((u \notin W)\), add \( u \to W \).

  \[u \text{ should be the one with min distance to } S\]
among all \( u \notin W \).

Claim: if \( u \) is the one with min distance to \( S \) for \( u \notin W \), then the shortest path from \( S \) to \( u \) only uses points in \( W \).

Proof: assume the shortest path is not entirely in \( W \) 

- Implementing Dijkstra's algorithm
  - maintain \( W \) (set of vertices with known shortest path)
  - maintain \( \text{dist}[v] \)
    
    \[
    \begin{align*}
    \text{for } v \in W & \quad \text{dist}[v] = \text{length of shortest path} \\
    \text{for } v \notin W & \quad \text{dist}[v] = \begin{cases} 
    \text{length of shortest path} & \text{to } v \text{ where all vertices are in } W \\
    \end{cases}
    \end{align*}
    \]
  
  - every iteration: find \( u \notin W \) with smallest \( \text{dist}[u] \)
    add \( u \) to \( W \), update \( \text{dist}[u] \)

- negative edge length
$3 \times 2 = 6$

$\log 3 + \log 2 = \log 6$

$\log 3 + \log 2 + \log \frac{1}{7} = \log \frac{6}{7} < 0$