Single Source Shortest Path

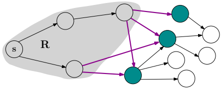
- Weighted Graphs and Shortest Paths
- Dijkstra Algorithm
- Proof of Correctness
- Runtime
  - Basic Implementation
  - Vertex-Centric Implementation
  - Heap Based Implementation

Imdad ullah Khan

Dijkstra Algorithm: Runtime

Algorithm Dijkstra's Algorithm for Shortest Paths from s to all vertices

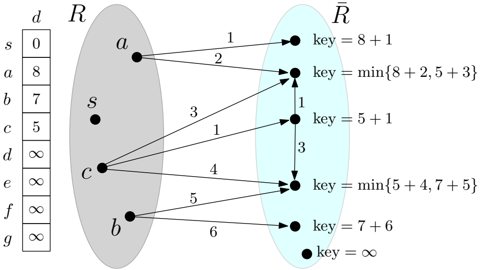
$d[1 \dots n] \leftarrow [\infty \dots \infty]$
$prev[1 \dots n] \leftarrow [null \ \dots \ null]$
$d[s] \leftarrow 0$
$R \leftarrow \{s\}$
while $R \neq V$ **do**
  Select $e = (u, v)$, $u \in R, v \notin R$, with minimum $d[u] + w(uv)$
  $R \leftarrow R \cup \{v\}$
  $d[v] \leftarrow d[u] + w(uv)$
  $prev[v] \leftarrow u$



- While loop runs for $O(n)$ iterations
- Find minimum score edge takes $O(m)$ times
- Total runtime $O(nm)$
- Repeatedly finding minimum is expensive

Dijkstra Algorithm - Vertex Centric

- Store information at vertices (target of many edges)
- Key at vertices is length of current best single edge extension
- Find closest vertex to s by key

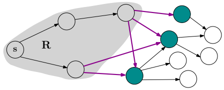


- Key is easy to update, just traverse neighbors of new vertex in R

Dijkstra Algorithm - Vertex Centric - Runtime

Algorithm Dijkstra's Algorithm for distances from s to all vertices

$d[1 \ldots n] \leftarrow [\infty \ldots \infty]$
$d[s] \leftarrow 0$
while $R \neq V$ **do**
  Select $v \in \overline{R}$ with minimum $d[v]$
  $R \leftarrow R \cup \{v\}$
  for each $z \in N(v) \cap \overline{R}$ **do**
    if $d[z] > d[v] + w(vz)$ **then**
      $d[z] \leftarrow d[v] + w(vz)$



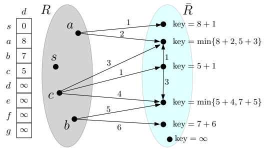
- While loop runs for $O(n)$ iterations
- Find minimum score **vertex** takes $O(n)$ time
- Need to update only neighbors of added vertex
- Total runtime $O(n \cdot n + m)$
- Better than last one, esp. for dense graphs
- Repeatedly finding minimum key is expensive

Dijkstra Algorithm - Heap Implementation

- Recall the Priority Queue ADT and its Heap implementation
- Operations:
  - $\mathcal{H} \leftarrow \textsc{initialize}()$ $\quad\triangleright\ O(n)$
  - $\textsc{insert}(\mathcal{H}, v, k)$ $\quad\triangleright\ O(\log n)$
  - $v \leftarrow \textsc{extract-min}(\mathcal{H})$ $\quad\triangleright\ O(\log n)$
  - $\textsc{delete}(\mathcal{H}, v)$ $\quad\triangleright\ O(\log n)$
  - $\textsc{decrease-key}(\mathcal{H}, v, k')$ $\quad\triangleright\ O(\log n)$

Dijkstra Algorithm - Heap Implementation

- Store information at vertices (target of many edges)
- Key at vertices is length of current best single edge extension
- Find closest vertex to s by key
- Key is easy to update, traverse neighbors of new vertex in R



- Store all vertices in $\overline{R}$ in a heap $\mathcal{H}$ with keys
- Initialize $\mathcal{H}$ with V, key of s is 0 for others ∞
- Save pointers (location in heap) to each vertex
- $v \leftarrow$ EXTRACT-MIN($\mathcal{H}$) to add to R
- Traverse $N(v)$ to update keys of neighbors of v in $\overline{R}$

Dijkstra Algorithm - Heap Implementation

Algorithm Dijkstra's Algorithm for distances from s to all vertices

$d[1 \dots n] \leftarrow [\infty \dots \infty]$
$d[s] \leftarrow 0$
$\mathcal{H} \leftarrow \textsc{initialize}(V, d)$ $\triangleright$ make a heap with all vertices and keys as $d[\cdot]$
while $R \neq V$ **do**
  $v \leftarrow \textsc{extract-min}(\mathcal{H})$
  for each $z \in N(v) \cap \overline{R}$ **do**
    if $d[z] > d[v] + w(vz)$ **then**
      $\textsc{decrease-key}(\mathcal{H}, z, d[v] + w(vz))$
  $R \leftarrow R \cup \{v\}$

Dijkstra Algorithm - Heap Implementation: Runtime

- In total there are n EXTRACT-MIN operations
- On extracting v, there are $O(deg(v))$ DECREASE-KEY operations
- Each EXTRACT-MIN takes $O(\log n)$ time
- Each DECREASE-KEY takes $O(\log n)$ time
- Total runtime $n \log n + m \log n = (n + m) \log n$