Shortest Paths in a Graph

Slides by Carl Kingsford

Feb. 5, 2014

Based on/Reading: Chapter 4.5 of Kleinberg & Tardos
Shortest Paths in a Weighted, Directed Graph

Given a directed graph $G$ with lengths $\ell_e > 0$ on each edge $e$:

Goal: Find the shortest path from a given node $s$ to every other node in the graph.
Shortest Paths

**Shortest Paths.** Given directed graph \( G \) with \( n \) nodes, and non-negative lengths on each edge, find the \( n \) shortest paths from a given node \( s \) to each \( v_i \).

- Dijkstra’s algorithm (1959) solves this problem.
- If we have an undirected graph, we can replace each undirected edge by 2 directed edges:

  ![Diagram](image)

- If all the edge lengths are \( = 1 \), how can we solve this?
Shortest Paths. Given directed graph \( G \) with \( n \) nodes, and non-negative lengths on each edge, find the \( n \) shortest paths from a given node \( s \) to each \( v_i \).

- Dijkstra’s algorithm (1959) solves this problem.
- If we have an undirected graph, we can replace each undirected edge by 2 directed edges:

\[
\text{\begin{tikzpicture}[baseline=-0.1cm]
  \node[circle, draw] (A) at (0,0) {};
  \node[circle, draw] (B) at (1,0) {};
  \node[circle, draw] (C) at (2,0) {};
  \draw (A) -- node[pos=0.5] {k} (B);
\end{tikzpicture}} \quad \Rightarrow \quad \text{\begin{tikzpicture}[baseline=-0.1cm]
  \node[circle, draw] (A) at (0,0) {};
  \node[circle, draw] (B) at (1,0) {};
  \node[circle, draw] (C) at (2,0) {};
  \draw (A) -- node[pos=0.5] {k} (B);
  \draw (B) to [out=0,in=180] node[pos=0.5] {k} (C);
\end{tikzpicture}}
\]

- If all the edge lengths are \( = 1 \), how can we solve this? \( \text{BFS} \)
General Tree Growing

Dijkstra’s algorithm is just a special case of tree growing:

- Let $T$ be the current tree $T$, and
- Maintain a list of frontier edges: the set of edges of $G$ that have one endpoint in $T$ and one endpoint not in $T$:
- Repeatedly choose a frontier edge (somehow) and add it to $T$. 
Tree Growing

TreeGrowing(graph G, vertex v, func nextEdge):
    T = (v, ∅)
    S = set of edges incident to v
    While S is not empty:
        e = nextEdge(G, S)
        T = T + e // add edge e to T
        S = updateFrontier(G, S, e)
    return T

- The function nextEdge(G, S) returns a frontier edge from S.
- updateFrontier(G, S, e) returns the new frontier after we add edge e to T.
nextEdge for Shortest Path

- Let $u$ be some node that we’ve already visited (it will be in $S$).

- Let $d(u)$ be the length of the $s-u$ path found for node $u \in S$.

- $\text{nextEdge}$: return the frontier edge $(u, v)$ for which $d(u) + \text{length}(u, v)$ is minimized.

- The “$d(u)$” term is the difference from Prim’s algorithm.
Example

\[ d[s] = 0; \quad d[u] = 1 \]
(green gives frontier)

\[ d[w] = 2 \]
**Proof of Correctness**

**Theorem.** Let $T$ be the set of nodes explored at some point during the algorithm. For each $u \in T$, the path to $u$ found by Dijkstra’s algorithm is the shortest.

**Proof.** By induction on the size of $T$. **Base case:** When $|T| = 1$, the only node in $T$ is $s$, for which we’ve obviously found the shortest path.

**Induction Hypothesis:** Assume theorem is true when $|T| \leq k$.

Let $v$ be the $(k + 1)^{st}$ node added using edge $(u, v)$.

Let $P_v$ be the path chosen by Dijkstra’s to $v$ and let $P$ be any other path from $s$ to $v$.

Then we have the situation on the next slide.
Proof, cont.

The path to \( v \) chosen by Dijkstra’s is of length \( \leq \) the alternative blue path.
Theorem. There is some optimal set of shortest paths from source \( s \) such that their union forms a tree.

Proof. Dijkstra’s algorithm is correct and produces a tree.
Implementation of Dijkstra

1: for u ∈ V do dist[u] ← ∞
2: H ← MAKE_HEAP()
3: u ← s # (s is an arbitrary start vertex)
4: while u ≠ null do
5:   for v ∈ NEIGHBORS(u) do
6:     # If the distance is smaller than before, we have to update
7:     if dist[u] + d(u,v) < dist[v] then
8:       dist[v] ← dist[u] + d(u,v)
9:     if v ∉ H then
10:        INSERT(H, v, dist[v]) # Sift up for new key
11:     else
12:        REDUCE_KEY(H, v, dist[v]) # Sift up for new key
13:     parent[v] ← u
14:   u ← DELETE_MIN(H)
15: return parent
Running time of Dijkstra’s Algorithm

Same as Prim’s MST algorithm:

- Every edge is processed in the `for` loop at most once.

- In response to that processing, we may either
  1. do nothing; \( O(1) \),
  2. insert a item into the heap of at most \(|V|\) items; \( O(\log |V|) \), or
  3. reduce the key of an item in a heap of at most \(|V|\) items; \( O(\log |V|) \)

- Total time is therefore: \( O(|E| \log |V|) \).