Edsger W. Dijkstra

The question of whether computers can think is like the question of whether submarines can swim.

Computer science is no more about computers than astronomy is about telescopes.

Elegance is not a dispensable luxury but a factor that decides between success and failure.

In their capacity as a tool, computers will be but a ripple on the surface of our culture. In their capacity as intellectual challenge, they are without precedent in the cultural history of mankind.

Object-oriented programming is an exceptionally bad idea which could only have originated in California.

**Shortest Paths in a Graph**

**Variations**

1. find the SP from some node s to all other nodes
2. find the SP from node s to node t

We can use 1 to solve 2, also there is no asymptotically faster algorithm for 2 than for 1.

**Dijkstra’s Algorithm**

**Shortest Path Problem**

Shortest path: inputs.
- Directed graph $G = (V, E)$.
- Source s, destination t.
- Length $l_e = length of edge e$.

Shortest path problem: find shortest directed path from s to t.
- length of path = sum of edge lengths in path

Dijkstra’s algorithm.
- Maintain a set of explored nodes $S$ for which we have determined the shortest path distance $d[u]$ from s to u.
- Initialize $S = \{s\}$, $d[s] = 0$.
- Repeatedly choose unexplored node v which minimizes $d'[v] = \min_{u \in S} d[u] + l_{uv}$, add v to S, and set $d[v] = d'[v]$.

**Cost of path a-2-3-5-t:** $0 + 9 + 23 + 2 + 16 = 50$.

- all pair SPs - can be solved by $|V|$ applications of 1, but can be solved faster.
Dijkstra’s Algorithm

Dijkstra’s algorithm:
- Maintain a set of explored nodes $S$ for which we have determined the shortest path distance $d(u)$ from $s$ to $u$.
- Initialize $S = \{s\}$, $d(s) = 0$.
- Repeatedly choose an unexplored node $v$ which minimizes $d'[v] = \min_{u \in S} d(u) + e_{uv}$, add $v$ to $S$, and set $d(v) = d'[v]$.

The algorithm computes shortest distances. How do we get the paths?
Producing shortest paths

- Maintain a set of explored nodes $S$ for which we have determined the shortest path distance $d[u]$ from $s$ to $u$.
- Initialize $S = \{s\}$, $d[s] = 0$.
- Repeatedly choose unexplored node $v$ which minimizes $d'[v] = \min_{x \in \{u \in S \}} d[u] + \ell_{uv}$.

The algorithm computes shortest distances. How do we get the paths?

Maintain an array $p[v]$ of predecessors

When $d[v]$ is set to $d'[v]$, set $p[v]$ to $u$

\[ d'[v] = \min_{x \in \{u \in S \}} d[u] + \ell_{uv}. \]

\[ d[v] = d'[v]. \]

Dijkstra's Algorithm: Proof of Correctness

Invariant: For each $u \in S$, Dijkstra's alg finds a shortest $s-u$ path.

Proof. (by induction on $|S|$)

Base case: $|S| = 1$ is trivial.

Induction step: Assume true for $|S| = k \geq 1$.

- Let $v$ be next node added to $S$, and let $u-v$ be the chosen edge.
- The shortest $s-u$ path plus $(u, v)$ is an $s-v$ path of length $d'(v)$.
- Consider any $s-v$ path $P$.
- Let $x-y$ be the first edge in $P$ that leaves $S$, and let $P'$ be the subpath to $x$.

\[ (P) \geq (P') \geq d[x] + \ell_{xy} \geq d'[x] \geq d'[v], \]

nonnegative weights
induction hypothesis
define of $d'(y)$
Dijkstra chose $v$ instead of $y$

Running Time

- Initialize $S = \{s\}$, $d[s] = 0$.
- Repeatedly choose unexplored node $v$ which minimizes

\[ d'[v] = \min_{x \in \{u \in S \}} d[u] + \ell_{uv}. \]


While loop is executed $|V| - 1$ times

- $O(|V|)$ extract_min operations
- $O(|E|)$ update_key operations

Therefore: $O(|V| \log |V| + |E| \log |V|) = O(|E| \log |V|)$

Priority Queue

- Data structure that maintains a set of elements.
- Each element has a key that denotes its priority.
- Support for adding, elements; selection of element with the highest priority, and modifying an element's priority.

Implementation: heap

Heaps

Heap: array representation of a complete binary tree
- Every level is completely filled except the bottom level, filled from left to right
- Can compute the index of parent and children, for 1-based arrays:
- $O(E)$ update_key operations

Min-Heap property: key at any node is smaller than key in its children
Heap operations

Heap operations:
- `extract_min` - return and delete the element with the smallest key (or more generally, `delete(i)` which deletes item at position `i`)
- `find_min`
- `insert(item)`
- `change_key(item)`

Support operations:
- `heapify_up(i)` - restore the heap after an insert operation
- `heapify_down(i)` - restore the heap after a delete operation