Dijkstra's Algorithm: Correctness

Why is this right? Getting the right loop invariant is hard.

- What does $d(v_i)$ correspond to if $v_i \notin U$?

Loop invariant. For all $k \geq 0$, after the kth interation of the loop:

- U consists of v_0 and the k vertices closest to v_0
- u is the kth closest vertex to v_0
- if $v \in U$, then $d(v)$ is the length of the shortest path from v_0 to v.
- if $v \notin U$, then $d(v)$ is the length of the shortest path from v_0 to v that has only vertices in U as intermediate points.
  - if there is no such path, then $d(v) = \infty$
- u is the most recently added vertex.

Once we get the right loop invariant, it's not hard to prove that it is maintained ... by induction (of course):

Basis: $P(0)$ is obvious.

Inductive step: ...

Eulerian Paths

Recall that $G(V, E)$ has an Eulerian path if it has a path that goes through every edge exactly once. It has an Eulerian cycle (or Eulerian circuit) if it has an Eulerian path that starts and ends at the same vertex.

How can we tell if a graph has an Eulerian path/circuit?

What's a necessary condition for a graph to have an Eulerian circuit?

Count the edges going into and out of each vertex:

- Each vertex must have even degree!

This condition turns out to be sufficient too.

Theorem: A connected (multi)graph has an Eulerian cycle iff each vertex has even degree.

Proof: The necessity is clear: In the Eulerian cycle, there must be an even number of edges that start or end with any vertex.

To see the condition is sufficient, we provide an algorithm for finding an Eulerian circuit in $G(V, E)$.

First step: Follow your nose to construct a cycle.

Second step: Remove the edges in the cycle from G. Let H be the subgraph that remains.

- every vertex in H has even degree
- H may not be connected; let $H_1, \ldots, H_k$ be its connected components.

Third step: Apply the algorithm recursively to $H_1, \ldots, H_k$, and then splice the pieces together.

Finding cycles

First, find an algorithm for finding a cycle:

Input: $G(V, E)$ [a list of vertices and edges]

procedure Pathgrow(V,E,v)
  [v is first vertex in cycle]
  $P \leftarrow ()$ [P is sequence of edges on cycle]
  $w \leftarrow v$ [w is last vertex in P]
  repeat until $I(w) - P = \emptyset$
    [$I(w)$ is the set of edges incident on w]
    Pick $e \in I(w) - P$
    $w \leftarrow$ other end of e
    $P \leftarrow P \cdot e$ [append e to P]
  endrepeat
  return P
endpro

Claim: If every vertex in V has even degree, then P will be a cycle

- Loop invariant: In the graph $G(V, E - P)$, if the first vertex (v) and last vertex (w) in P are different, they have odd degree; all the other vertices have even degree.

Finding Eulerian Paths

Input: $G(V, E)$ [a list of vertices and edges]

Algorithm ECycle:

```
procedure Euler(V',E',v')
    Pathgrow(V',E',v')
    if P is not Eulerian,
      delete the edges in P from E;
      let G_1(V_1, E_1), ..., G_n(V_n, E_n) be
        the resulting connected components
      let v_i be a vertex in V_i
      for i = 1 to n
        Euler(V_i, E_i, v_i)
        Attach C to P at v_i
      endfor
    C ← P
    return C
endpro
v ← any vertex in V
Euler(V,E,v)
```


Corollary: A connected multigraph has an Eulerian path (but not an Eulerian cycle) if it has exactly two vertices of odd degree.

Which of these graphs have Eulerian paths:

Hamiltonian Paths

Recall that $G(V, E)$ has a Hamiltonian path if it has a path that goes through every vertex exactly once. It has a Hamiltonian cycle (or Hamiltonian circuit) if it has a Hamiltonian path that starts and ends at the same vertex.

There is no known easy characterization or algorithm for checking if a graph has a Hamiltonian cycle/path.

Which of these graphs have a Hamiltonian cycle?

Searching Graphs

Suppose we want to process data associated with the vertices of a graph. This means we need a systematic way of searching the graph, so that we don't miss any vertices.

There are two standard methods.

- Breadth-first search
- Depth-first search

It's best to think of these on a tree:

Breadth-first search would visit the nodes in the following order:

$$1, 2, 3, \ldots, 10$$

Depth-first search would visit the nodes in the following order:

$$1, 2, 4, 5, 7, 8, 11, 3, 6, 9, 10$$

Breadth-First Search

Input $G(V, E)$ [a connected graph]
v [start vertex]

Algorithm Breadth-First Search
visit v
$V' \leftarrow \{v\}$ [V' is the vertices already visited]
Put v on Q [Q is a queue]
repeat while $Q \neq \emptyset$
$u \leftarrow head(Q)$ [$head(Q)$ is the first item on Q]
for $w \in A(u)$ [$A(u) = \{w | \{u, w\} \in E\}$]
if $w \notin V'$
then visit w
Put w on Q
$V' \leftarrow V' \cup \{w\}$
endif
endfor
Delete u from Q

If all edges have equal length, we can extend this algorithm to find the shortest path length from v to any other vertex:

- Store the path length with each node when you add it.
- Length(v) = 0.
- Length(w) = Length(u) + 1

Depth-First Search

Input $G(V, E)$ [a connected graph]
v [start vertex]

Algorithm Depth-First Search
visit v
$V' \leftarrow \{v\}$ [V' is the vertices already visited]
Put v on S [S is a stack]
$u \leftarrow v$
repeat while $S \neq \emptyset$
if $A(u) - V' \neq \emptyset$
then Choose $w \in A(u) - V'$
visit w
$V' = V' \cup \{w\}$
Put w on stack
$u \leftarrow w$
else $u \leftarrow top(S)$ [Pop the stack]
endif
endrepeat