Dijkstra's Algorithm

Carlos Moreno
cmoreno@uwaterloo.ca
EIT-4103

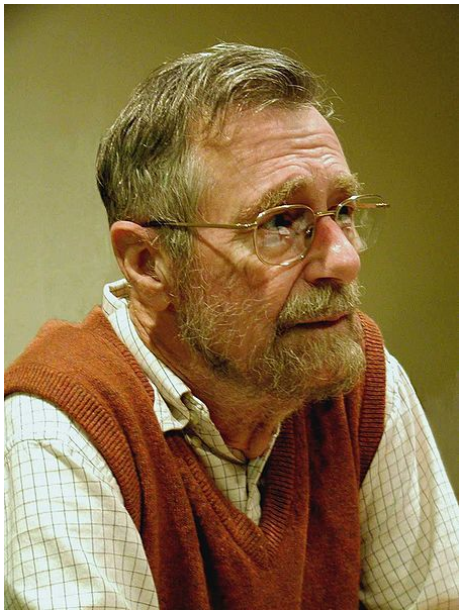


Image courtesy of wikipedia.org

https://ece.uwaterloo.ca/~cmoreno/ece250

These slides, the course material, and course web site are based on work by Douglas W. Harder

Dijkstra's Algorithm

Standard reminder to set phones to silent/vibrate mode, please!

Dijkstra's Algorithm

- During today's class we'll:
  - Discuss the problem of shortest path in a graph, and its applications.
  - Look at the naive solution (exhaustive search) and its run time.
  - Discuss Dijkstra's Algorithm:
    - Preliminaries – definitions / notation
    - Description of the algorithm
      - How and why it works

Dijkstra's Algorithm

- Shortest path in a graph:
  - In a directed weighted graph (possibly with cycles) with positive weights:
  - Given two vertices, v_s and v_d (source and destination) determine the path from v_s to v_d with lowest length
    - We recall that this corresponds to lowest sum of the weights of the edges in the path.

Dijkstra's Algorithm

- Shortest path in a graph:
  - Typical example: driving directions — given the graph representing streets and intersections, highways, cities, etc., we want to determine the best route.

Dijkstra's Algorithm

- Shortest path in a graph:
  - Typical example: driving directions — given the graph representing streets and intersections, highways, cities, etc., we want to determine the best route.
    - Careful with the notions of “best” vs. “shortest”

Dijkstra's Algorithm

- Shortest path in a graph:
  - Typical example: driving directions — given the graph representing streets and intersections, highways, cities, etc., we want to determine the best route.
    - Careful with the notions of “best” vs. “shortest”
    - The most typical goal in this example is getting the *fastest* route, not the *shortest* in actual (geographic) distance.

Dijkstra's Algorithm

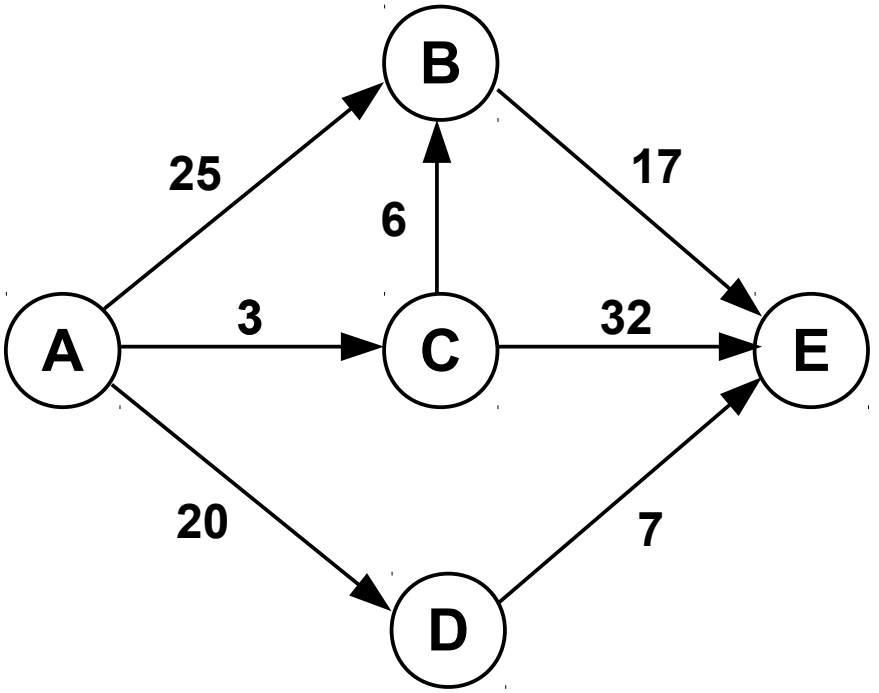
- Shortest path in a graph:
  - Typical example: driving directions — given the graph representing streets and intersections, highways, cities, etc., we want to determine the best route.
    - Careful with the notions of “best” vs. “shortest”
    - The most typical goal in this example is getting the *fastest* route, not the *shortest* in actual (geographic) distance.
    - This is not an issue in graph terminology — the weights in the graph may represent estimated time, distance, or some other cost — we always talk about “shortest path” when referring to the graph.

Dijkstra's Algorithm

- Shortest path in a graph:
  - Another example of use — routing protocols in networking systems (in particular, the RFC 2328 standard, part of the building blocks of the Internet, defines OSPF protocol, using Dikjstra's algorithm).
    - The Internet can be represented as a graph where nodes are computers or in general “network nodes”, and edges represent a direct connection.
    - For a message to get from a point to another, the message has to be passed from computer to computer; a path has to be found:
      - We want to try the shortest (lowest time) path first (why do we say shortest path *first* — why not simply: we use the shortest path?)

Dijkstra's Algorithm

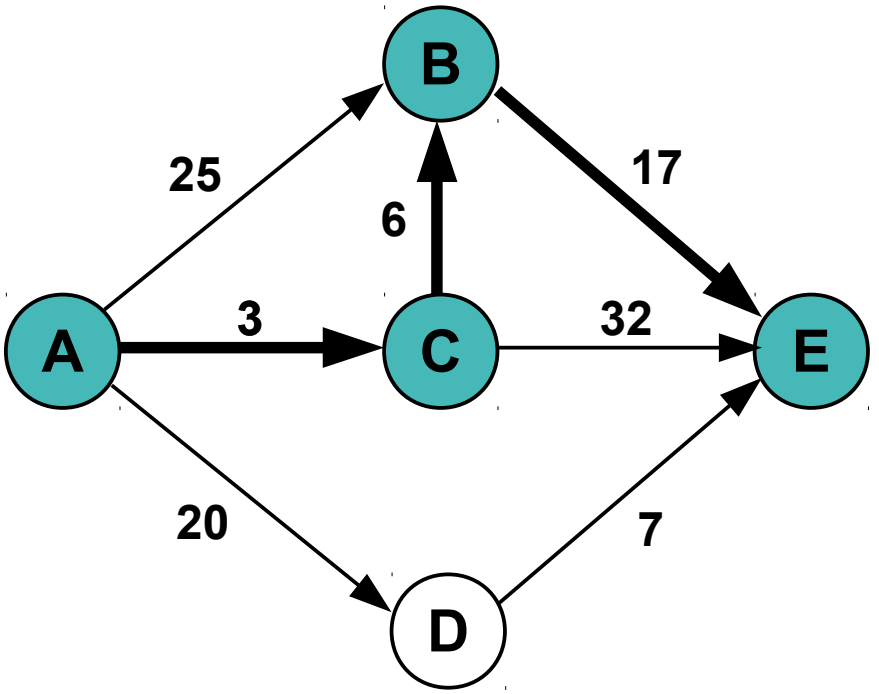
- Shortest path in a graph – example:



Shortest path from A to E ?

Dijkstra's Algorithm

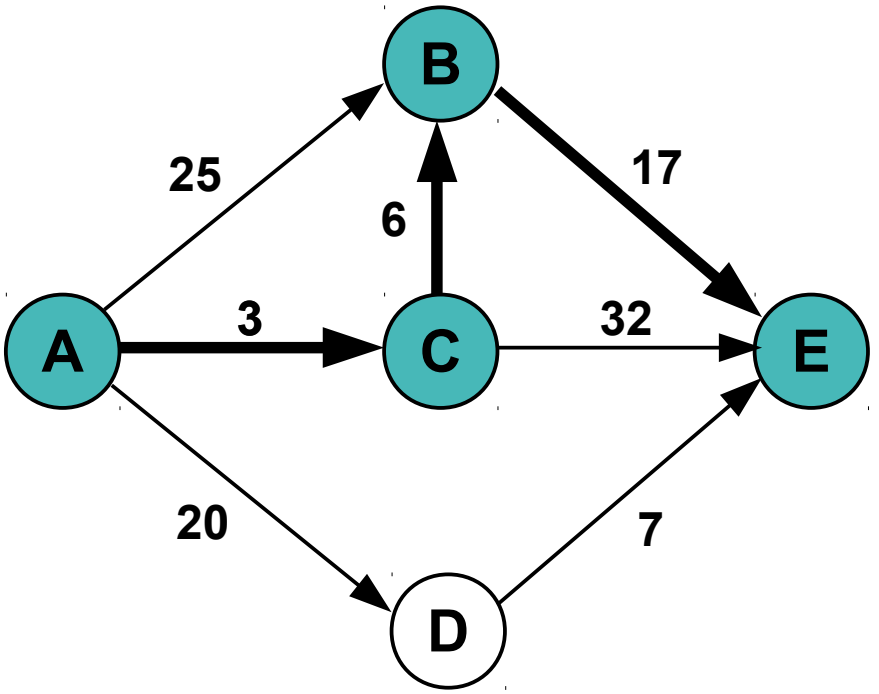
- Shortest path in a graph – example:



Shortest path from A to E ?

Dijkstra's Algorithm

- Shortest path in a graph – example:



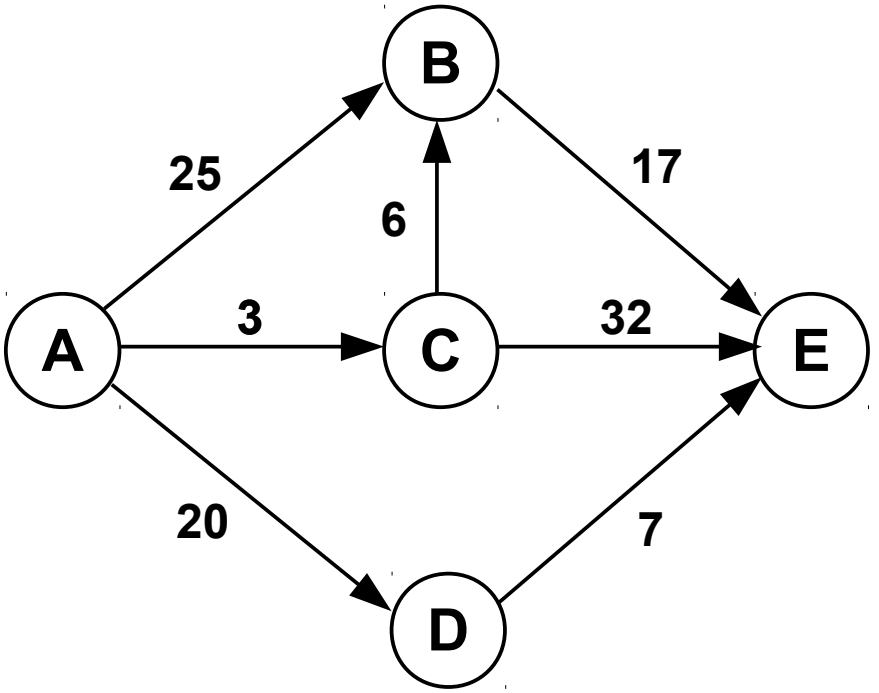
How do we know for sure ?

Dijkstra's Algorithm

- Shortest path in a graph – example:

  Possible paths:

  - A-B-E
  - A-C-E
  - A-C-B-E
  - A-D-E



Dijkstra's Algorithm

- Shortest path in a graph – example:
  - Of course, this worked well for a 5-vertices graph.
  - Asymptotically, for large values of *n* (*n* being the number of vertices in the graph), the number of possible paths is .... anyone?

Dijkstra's Algorithm

- Shortest path in a graph – example:
  - Of course, this worked well for a 5-vertices graph.
  - Asymptotically, for large values of *n* (*n* being the number of vertices in the graph), the number of possible paths is .... anyone?
  - There's an issue to consider:
    - Since the graph can contain cycles, then the number of possible paths is unbounded (can cycle any arbitrary number of times before resuming our way to the target vertex)

Dijkstra's Algorithm

- Shortest path in a graph – example:
  - If we assume no cycles (or in any case, restrict the count of possible paths to simple paths — thus, containing no cycles), then .... anyone?

Dijkstra's Algorithm

- Shortest path in a graph – example:
  - If we assume no cycles (or in any case, restrict the count of possible paths to simple paths — thus, containing no cycles), then .... anyone?
  - The actual math is a little heavy (just a little, but enough that I will omit it) — but intuitively, it goes with $n!$ (more specifically, with $(n-2)!$)
    - A worst-case has to consider every vertex adjacent to every other vertex; in this case, the number of paths is really the number of permutations of the $(n-2)$ remaining vertices.

Dijkstra's Algorithm

- Shortest path in a graph – example:
  - That didn't look that bad, right? The slight complication comes from the fact that there are also paths formed by fewer vertices, including all possible permutations of every subset of vertices.

Dijkstra's Algorithm

- Having seen how catastrophically slow things could be, let's look at Dijkstra's remarkable (and remarkably efficient) idea...

Dijkstra's Algorithm

- Dijkstra's Algorithm – preliminaries:
  - Observation 1:

    A shortest path to a vertex never passes twice through the same vertex.

Dijkstra's Algorithm

- Dijkstra's Algorithm – preliminaries:
  - Observation 1:

    A shortest path to a vertex never passes twice through the same vertex.
  - Proof: (this is quite obvious to see, and quite trivial to prove, right?)

Dijkstra's Algorithm

- Dijkstra's Algorithm – preliminaries:
  - Observation 1:

    A shortest path to a vertex never passes twice through the same vertex.
  - Proof: (this is quite obvious to see, and quite trivial to prove, right?)

Dijkstra's Algorithm

- Dijkstra's Algorithm – preliminaries:
  - Observation 1:

    A shortest path to a vertex never passes twice through the same vertex.
    - Thus, once we decide that we know the shortest path to some vertex (some intermediate vertex), we know we won't visit that vertex again.

Dijkstra's Algorithm

- Dijkstra's Algorithm – preliminaries:
  - Observation 2:

    If $\boldsymbol{P} = (v_1, v_2, \cdots, v_m)$ is the shortest path from v_1 to v_m, then the shortest path from v_1 to each of the points $v_k \in \boldsymbol{P}$ is the corresponding initial portion of $\boldsymbol{P}$ — that is, $(v_1, v_2, \cdots v_k)$

Dijkstra's Algorithm

- Dijkstra's Algorithm – preliminaries:
  - Observation 2:

    If $\boldsymbol{P} = (v_1, v_2, \cdots, v_m)$ is the shortest path from v_1 to v_m, then the shortest path from v_1 to each of the points $v_k \in \boldsymbol{P}$ is the corresponding initial portion of $\boldsymbol{P}$ — that is, $(v_1, v_2, \cdots v_k)$
  - Proof: (this one is also quite intuitive to see and quite easy to prove as well, right?)

Dijkstra's Algorithm

- Dijkstra's Algorithm – preliminaries:
  - Observation 2:

    If $\boldsymbol{P} = (v_1, v_2, \cdots, v_m)$ is the shortest path from v_1 to v_m, then the shortest path from v_1 to each of the points $v_k \in \boldsymbol{P}$ is the corresponding initial portion of $\boldsymbol{P}$ — that is, $(v_1, v_2, \cdots v_k)$
  - Proof: (this one is also quite intuitive to see and quite easy to prove as well, right?)

Dijkstra's Algorithm

- Dijkstra's Algorithm – preliminaries:
  - Observation 2 — corollary:
    - Once we determine the shortest path to a vertex v, then the paths that continue from v to each of its adjacent vertices (its “neighbours”) could be the shortest path to each of those neighbour vertices.

Dijkstra's Algorithm

- Dijkstra's Algorithm – definitions / notation:
  - Having noticed that couple of details, let's agree on some definitions and notational conventions...

Dijkstra's Algorithm

- Dijkstra's Algorithm – definitions / notation:
  - *Visited* vertex: a vertex for which we have determined the shortest path to it. Once we set a vertex as visited, that is final, and we won't visit that vertex again.

Dijkstra's Algorithm

- Dijkstra's Algorithm – definitions / notation:
  - *Visited* vertex: a vertex for which we have determined the shortest path to it. Once we set a vertex as visited, that is final, and we won't visit that vertex again.
    - (This is related to which observation?)

Dijkstra's Algorithm

- Dijkstra's Algorithm – definitions / notation:
  - *Visited* vertex: a vertex for which we have determined the shortest path to it. Once we set a vertex as visited, that is final, and we won't visit that vertex again.
    - (This is related to which observation?)
  - *Marked* vertex: a vertex for which a path to it has been found — we mark that path as a candidate for shortest path to that vertex.

Dijkstra's Algorithm

- Dijkstra's Algorithm – definitions / notation:
  - *Visited* vertex: a vertex for which we have determined the shortest path to it. Once we set a vertex as visited, that is final, and we won't visit that vertex again.
    - (This is related to which observation?)
  - *Marked* vertex: a vertex for which a path to it has been found — we mark that path as a candidate for shortest path to that vertex.
    - (This one is related to observation 2, right?)

Dijkstra's Algorithm

- Dijkstra's Algorithm:
  - We're now ready to present the algorithm...

Dijkstra's Algorithm

- Dijkstra's Algorithm:
  - Initialization:
    - Each vertex has a “distance” associated to it, representing the length of the path to it (the sum of the weights) — set that distance to some large value (e.g., ∞), except for the starting vertex, whose distance is initialized to 0.
    - Initialize every vertex to *unvisited*.

Dijkstra's Algorithm

- Dijkstra's Algorithm:
  - At each iteration:
    - Select the unvisited vertex with smallest non-∞ distance, denoted v_{min}. Set it as *visited*.
    - Mark each of the vertices adjacent to v_{min} (its "neighbours"):
      - If a neighbour was not marked, set its distance to v_{min}'s distance + the weight of the edge going to that neighbour.
      - If it was marked, overwrite its distance if the result is smaller than its current distance.

Dijkstra's Algorithm

- Dijkstra's Algorithm:
  - Remarkably enough, that's it !! (well, sort of ...).

Dijkstra's Algorithm

- Dijkstra's Algorithm:
  - Remarkably enough, that's it !! (well, sort of ...).
    - We should add, of course, that the algorithm ends when we *visit* the target vertex.

Dijkstra's Algorithm

- Dijkstra's Algorithm:
  - How about we give it a try! Maybe find the shortest path from A to E: (hopefully, you'll notice that there is a little problem with the algorithm as described ... We'll talk about this in class ...)