

Dijkstra's Algorithm



Carlos Moreno

cmoreno@uwaterloo.ca

EIT-4103

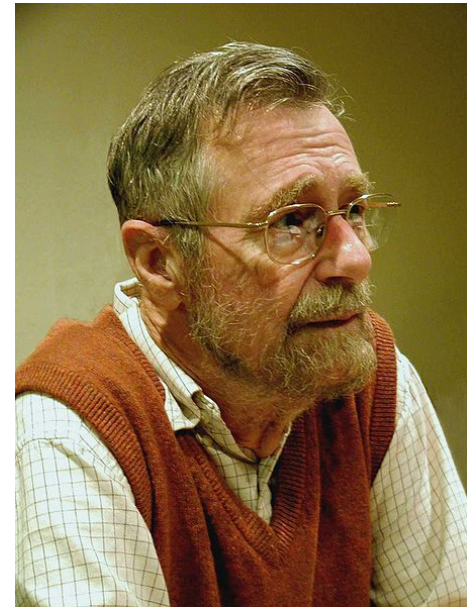


Image courtesy of wikipedia.org

<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 Dijkstra's algorithm).

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 Dijkstra'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.

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 Dijkstra'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 (e.g., lowest time) path first

Dijkstra's Algorithm

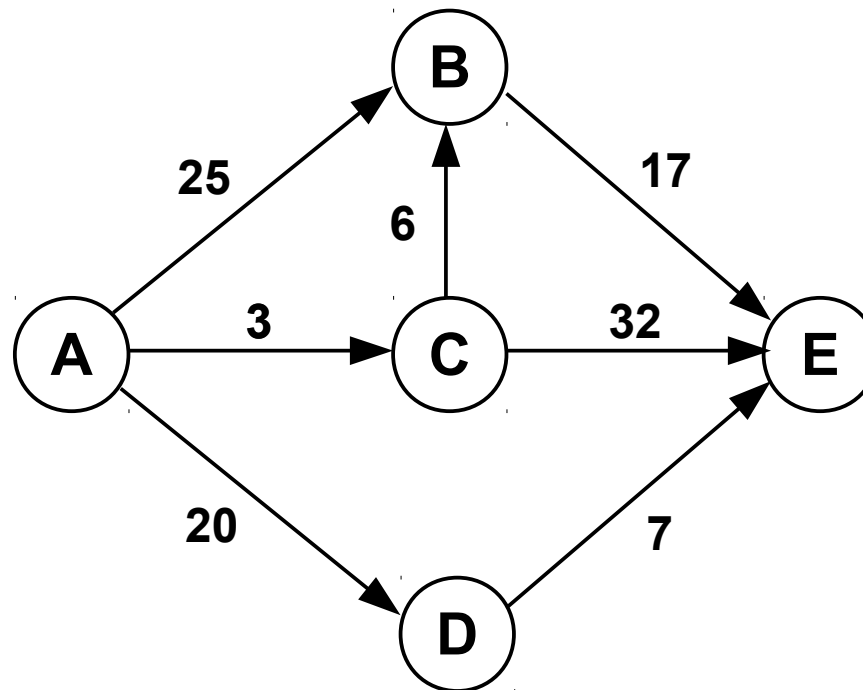
- Shortest path in a graph:
 - BTW ... Why do we say shortest path *first*?? Why not simply “we use the shortest path” ??

Dijkstra's Algorithm

- Shortest path in a graph:
 - BTW ... Why do we say shortest path *first*?? Why not simply “we use the shortest path” ??
 - For a network of computers, a given path between two nodes may be unavailable (one of the computers in the path may be temporarily down, or overloaded with excessive traffic, etc.).
 - So, we consider the shortest path as the first option, and not as the only option.

Dijkstra's Algorithm

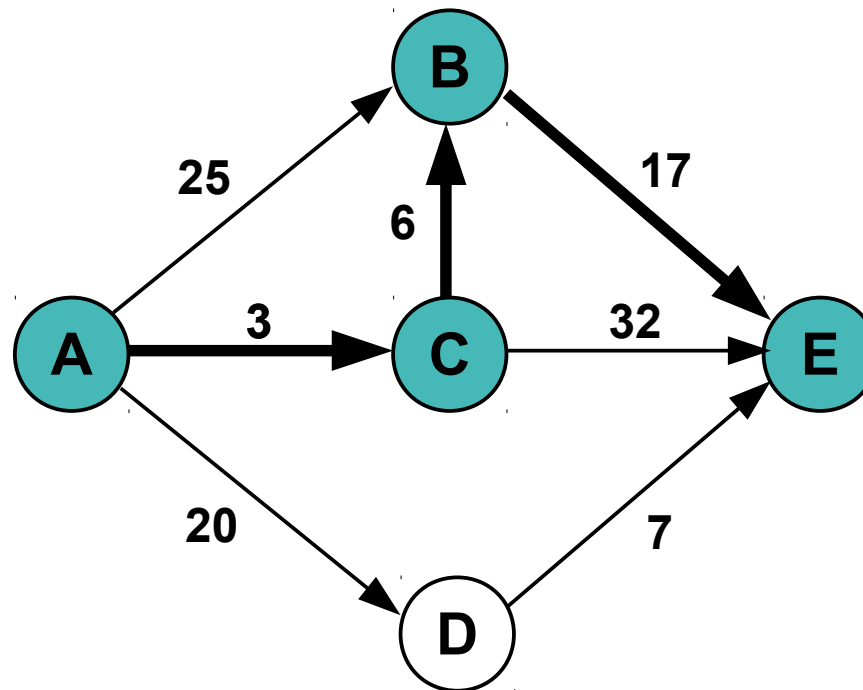
- Shortest path in a graph – example:



Shortest path from A to E ?

Dijkstra's Algorithm

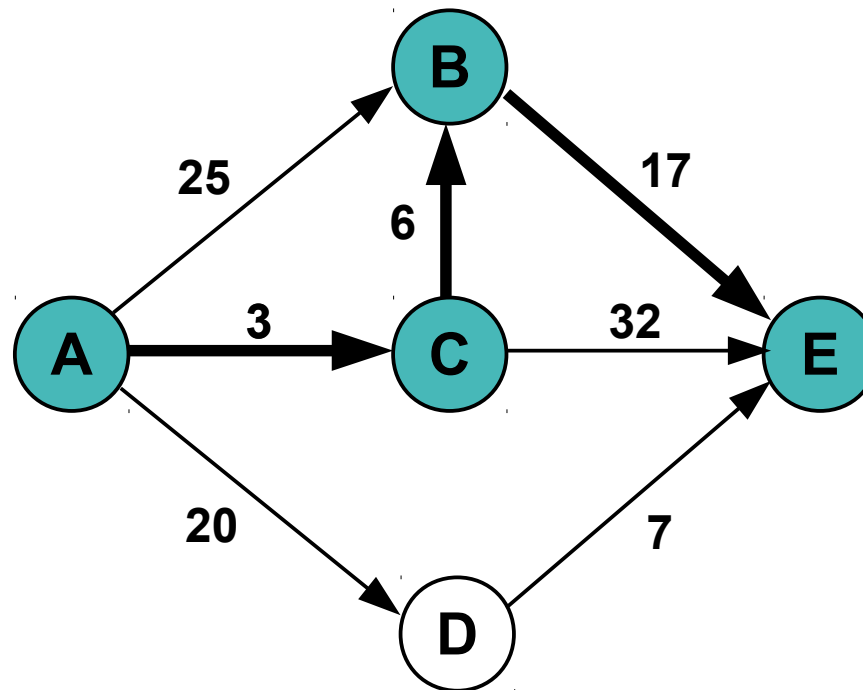
- Shortest path in a graph – example:



Shortest path from A to E ?

Dijkstra's Algorithm

- Shortest path in a graph – example:



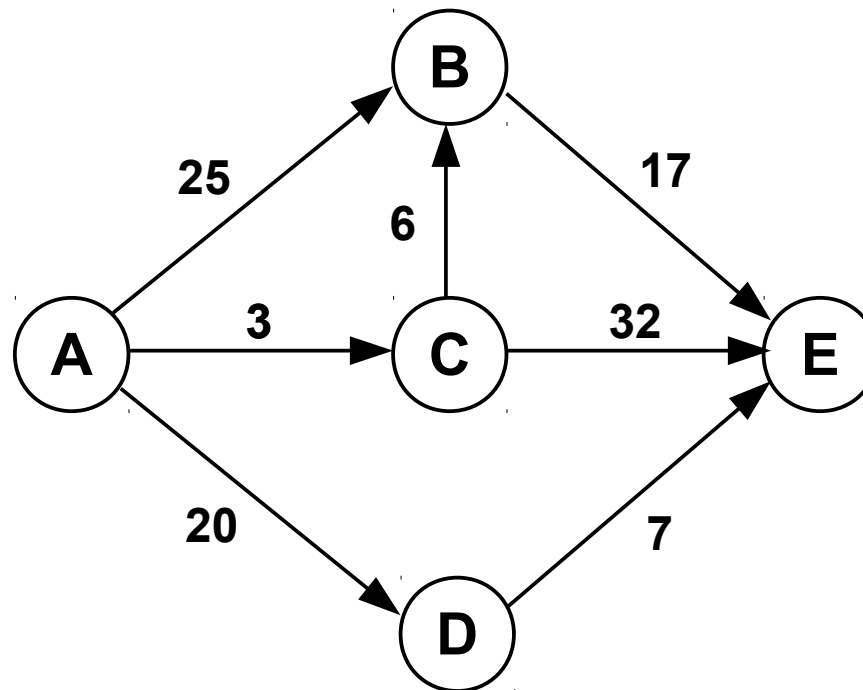
How do we know for sure ?

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:
 - Proof: Assume for a contradiction that this is not the case, and that the shortest path from v_1 to v_m is

$$\mathbf{P} = (v_1, \dots, v_{k-1}, v_k, c_1, \dots, c_i, v_k, v_{k+1}, \dots, v_m)$$

Then, the following is also a path from v_1 to v_m :

$$\mathbf{P}' = (v_1, \dots, v_{k-1}, v_k, v_{k+1}, \dots, v_m)$$

Since $\mathbf{P}'$ is a subset of $\mathbf{P}$, its length is lower, contradicting the assumption that $\mathbf{P}$ is the shortest path. Thus, our assumption that the shortest path can contain cycles must be false.

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

Dijkstra's Algorithm

- Dijkstra's Algorithm – preliminaries:
 - Proof: Assume for a contradiction that this is not the case, and that $\mathbf{P} = (v_1, v_2, \dots, v_m)$ is the shortest path from v_1 to v_m , yet the shortest path from v_1 to some vertex $v_k \in \mathbf{P}$ is $\mathbf{S}_k \neq (v_1, v_2, \dots, v_k)$

Then, the path $\mathbf{P}' = \mathbf{S}_k \parallel (v_{k+1}, \dots, v_m)$ is a path from v_1 to v_m , and its length is lower than the length of $\mathbf{P}$, contradicting the assumption that $\mathbf{P}$ is the shortest path from v_1 to v_m .

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 goes with observation 1, right?)

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 goes with observation 1, right?)
 - *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 goes with observation 1, right?)
 - *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.
 - (As we'll notice, the decision of when to mark a vertex is related to the corollary from observation 2)

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.

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*.

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”):

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

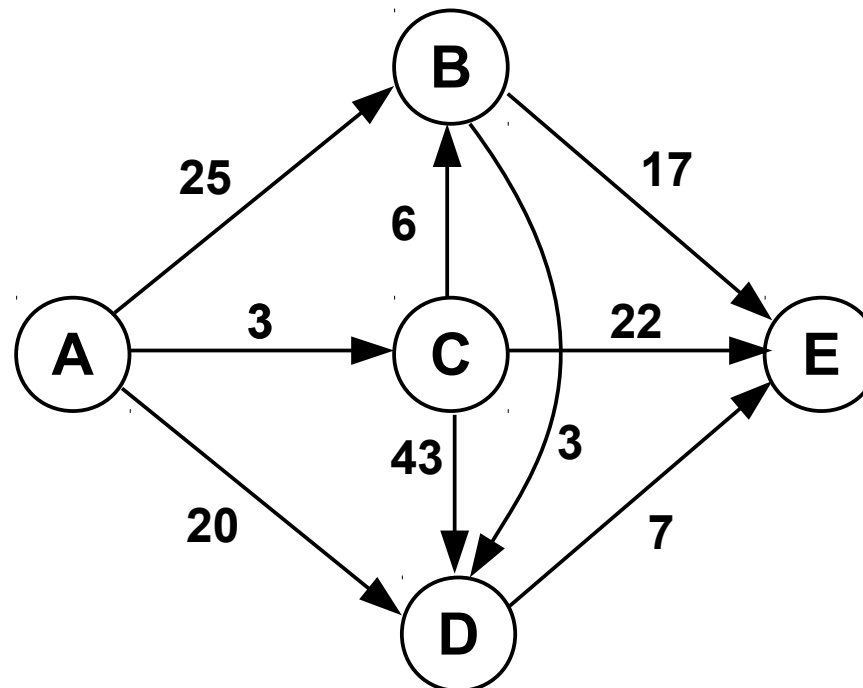
- Dijkstra's Algorithm:
 - Remarkably enough, that's it !! (well, sort of ...).

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.

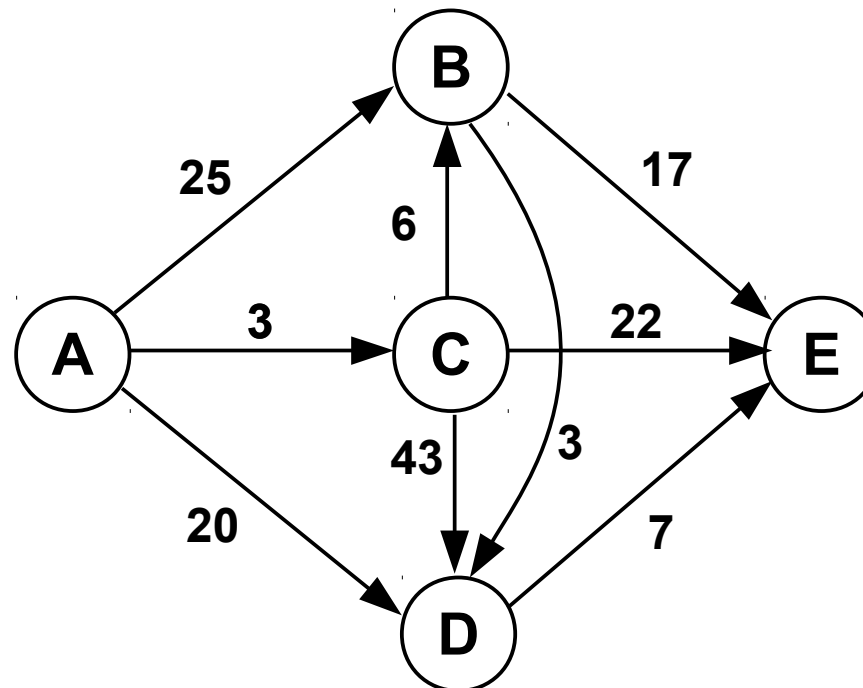
Dijkstra's Algorithm

- Dijkstra's Algorithm:
 - How about we give it a try! Maybe find the shortest path from A to E:



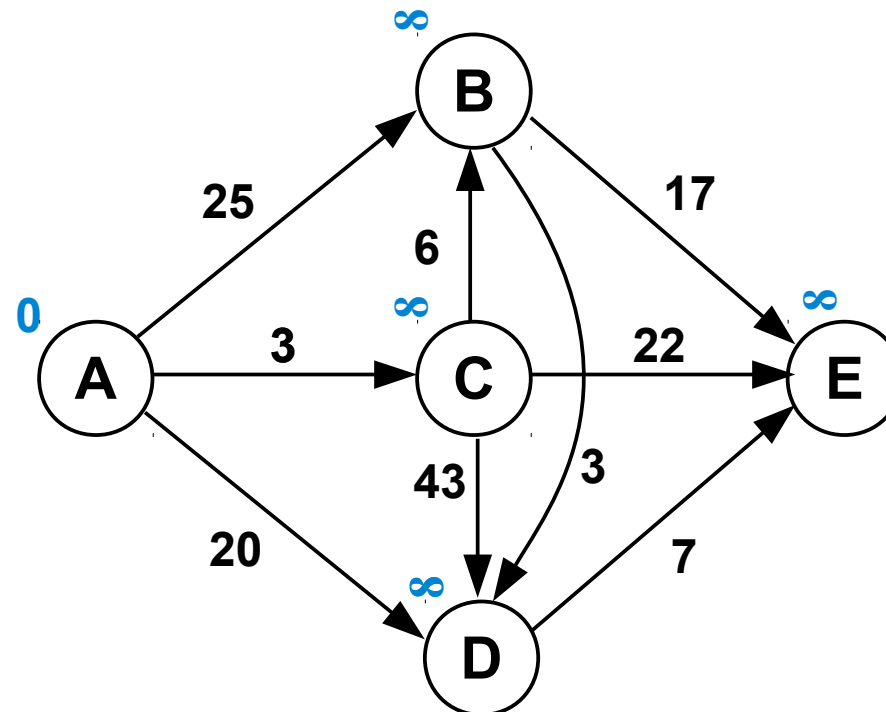
Dijkstra's Algorithm

- Dijkstra's Algorithm:
 - Convention: A vertex in green denotes visited. The number in blue denotes the associated distance. The condition *marked* is implicit by the presence of a non- ∞ distance.



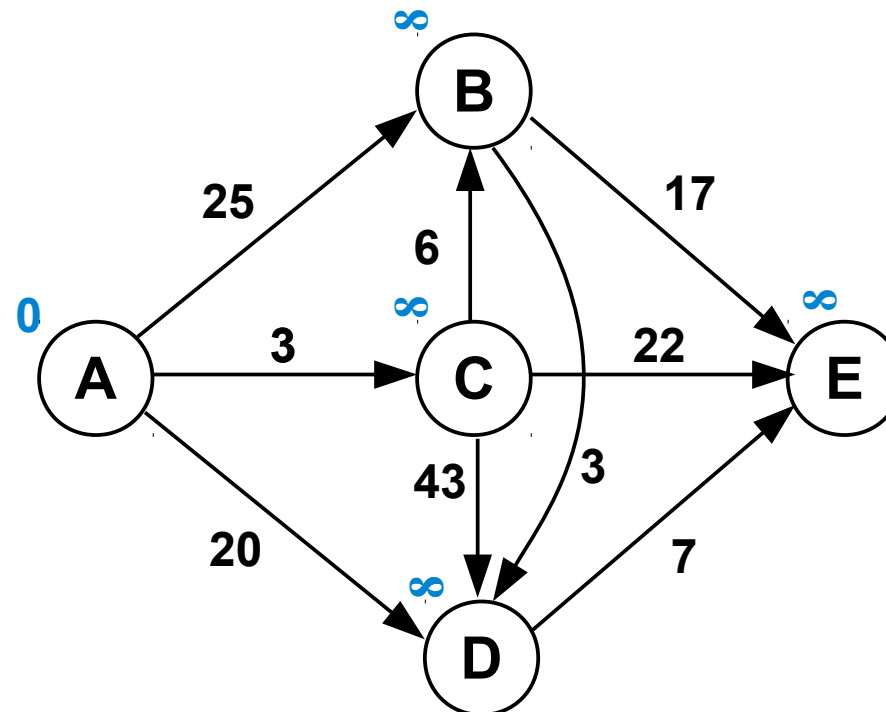
Dijkstra's Algorithm

- Dijkstra's Algorithm:
 - Initialization: All vertices unvisited, all with distance ∞ except for the starting vertex (with distance 0).



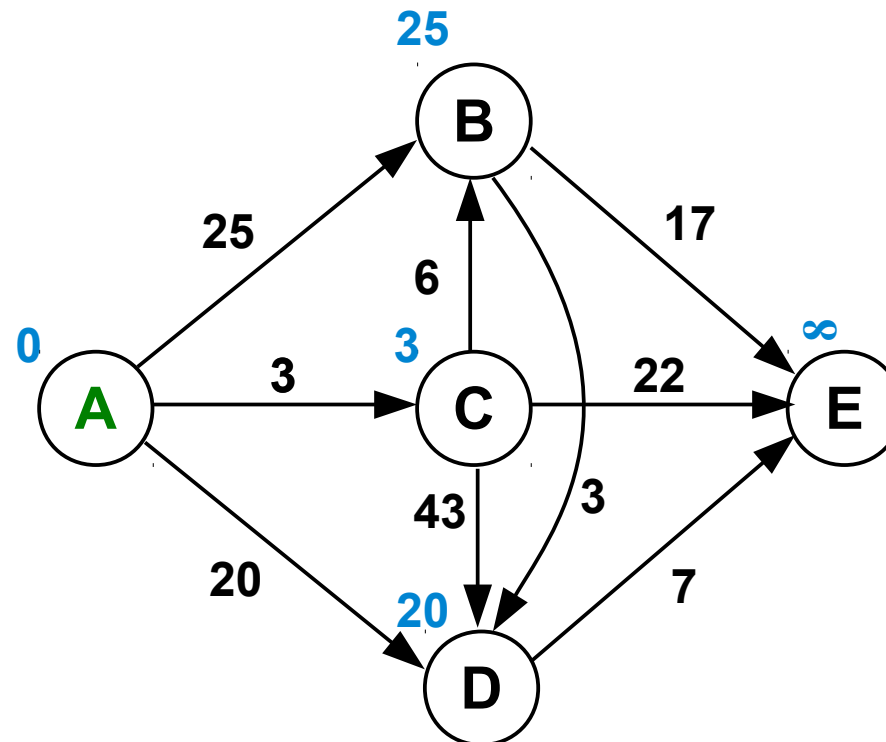
Dijkstra's Algorithm

- Dijkstra's Algorithm:
 - First iteration: there is only one non- ∞ distance, so we choose that one (A), set it to visited, and mark its neighbours:



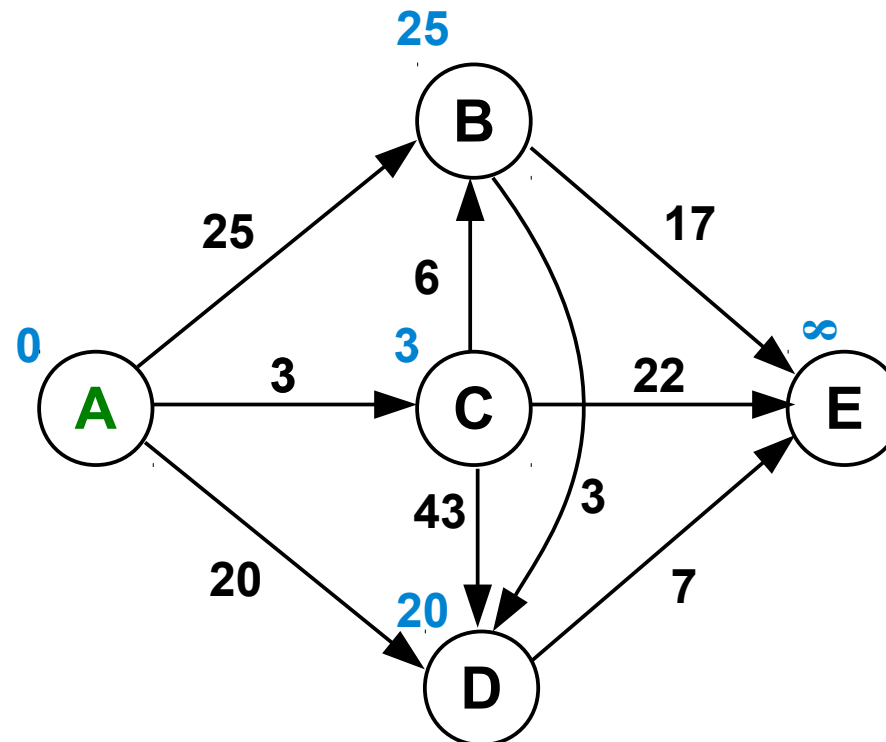
Dijkstra's Algorithm

- Dijkstra's Algorithm:
 - First iteration: there is only one non- ∞ distance, so we choose that one (A), set it to visited, and mark its neighbours:



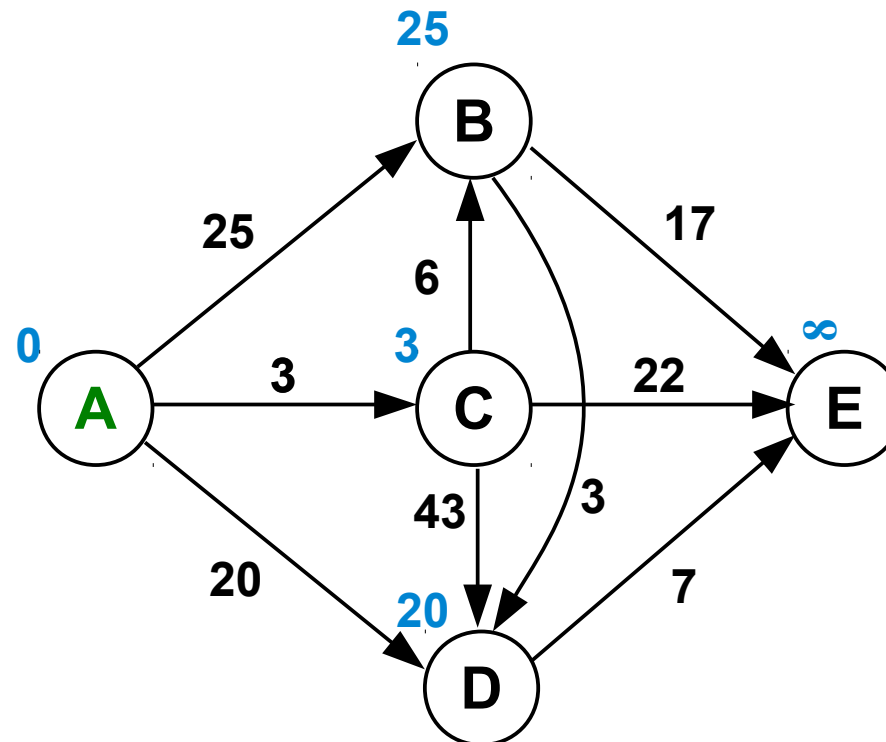
Dijkstra's Algorithm

- Dijkstra's Algorithm:
 - Second iteration: the smallest non- ∞ distance is 3 (vertex C), so we choose that one and mark it as visited:



Dijkstra's Algorithm

- Dijkstra's Algorithm:
 - Let's stop and try to think about (or *prove?*) why marking C as visited (meaning that we now have the shortest path to it) works:



Dijkstra's Algorithm

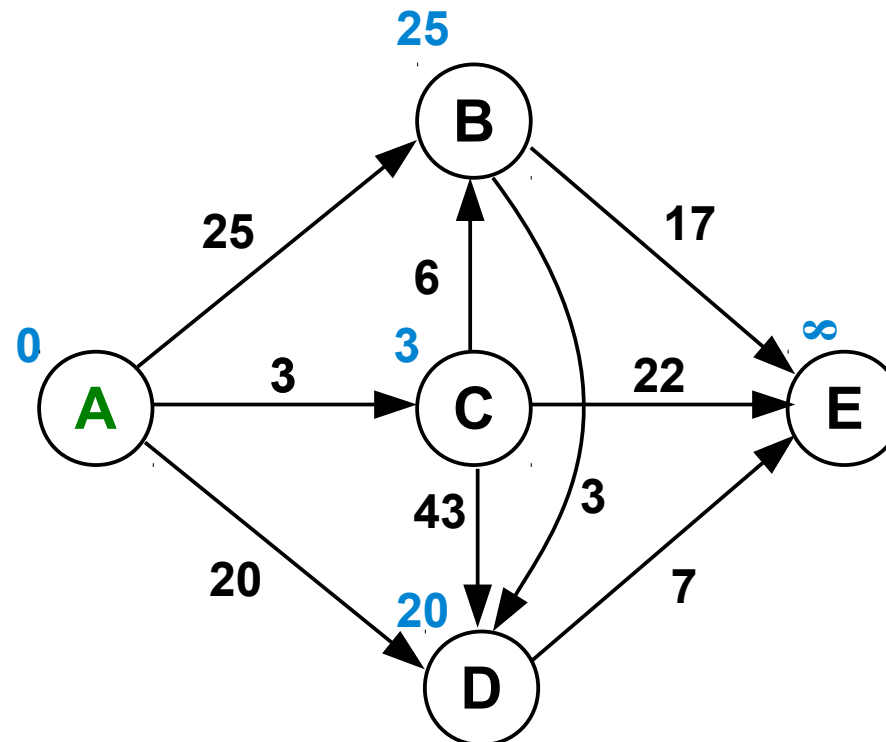
- Dijkstra's Algorithm:
 - One (perhaps obvious) observation: we can not simply say the same for all of the neighbours of the starting point — that is, it's not because it's a single edge going from A to C that it works (we have a counter-example for this, right?)

Dijkstra's Algorithm

- Dijkstra's Algorithm:
 - We could give an argument by contradiction: since 3 is the smallest weight starting from A, if we assume that the shortest path to C is not that edge, then it means that whatever shortest path we find will start with some of the other edges (let's call it edge x).
 - But the length of this other path would be the weight of x + something additional (positive, since all weights are positive) — but x 's weight is already larger than the path that we found to C, so the path starting through x can not be the shortest.

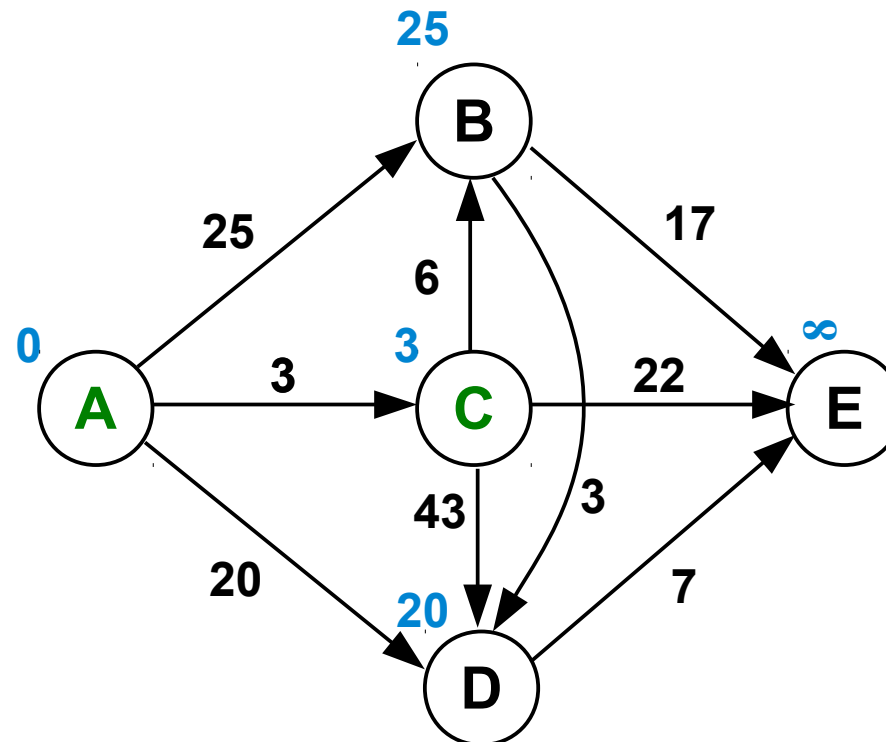
Dijkstra's Algorithm

- Dijkstra's Algorithm:
 - Back to vertex C — we set it as visited, and update the distance for each of its neighbours (B, D, E):



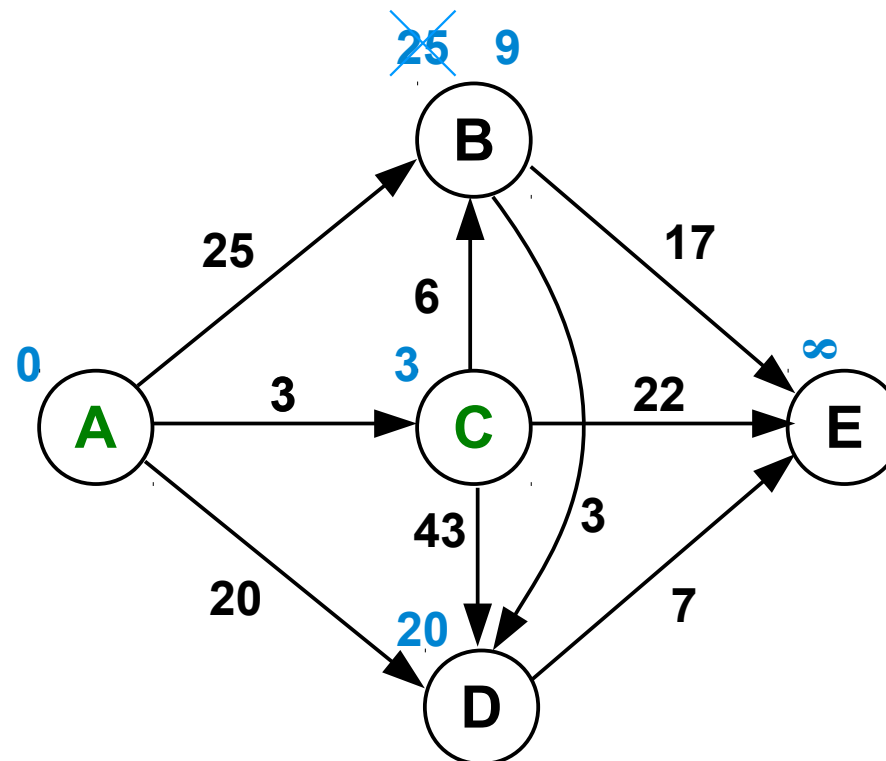
Dijkstra's Algorithm

- Dijkstra's Algorithm:
 - For B, the distance through C is $3 + 6 = 9 < 25$, so we overwrite this distance to B:



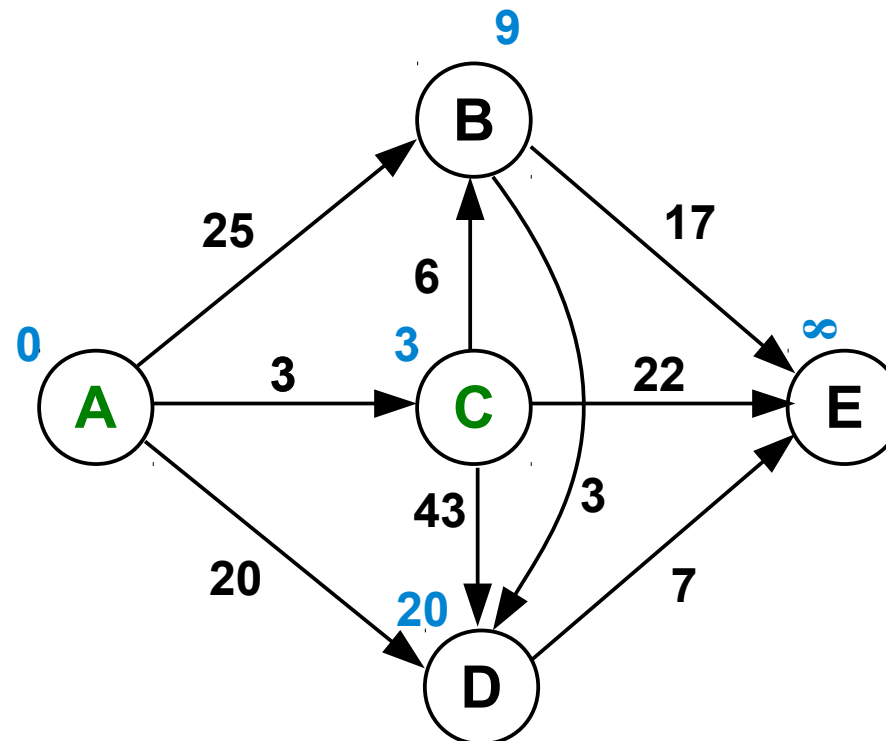
Dijkstra's Algorithm

- Dijkstra's Algorithm:
 - For B, the distance through C is $3 + 6 = 9 < 25$, so we overwrite this distance to B:



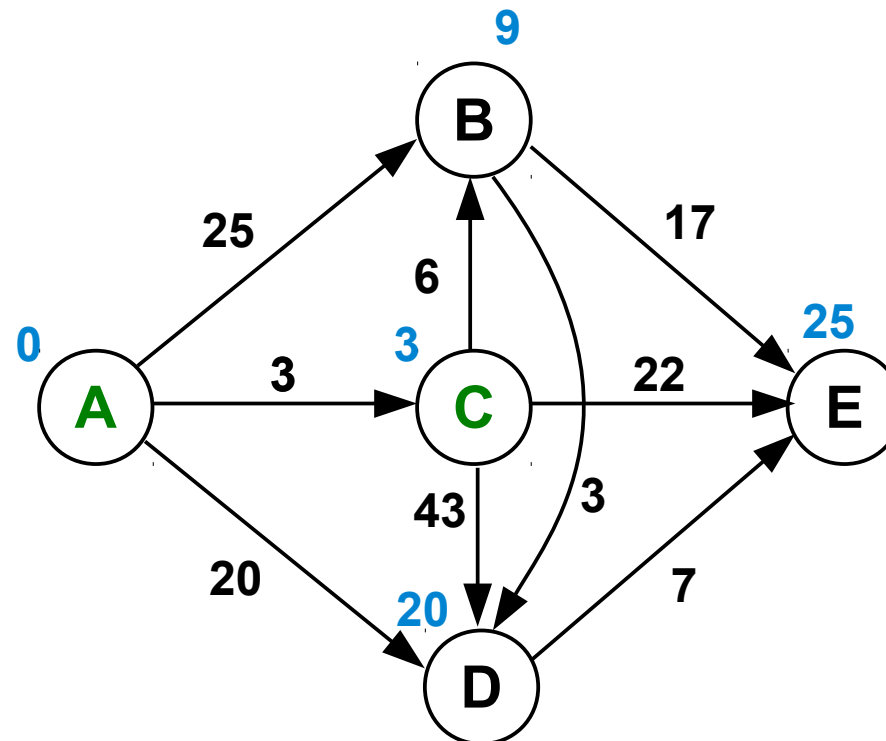
Dijkstra's Algorithm

- Dijkstra's Algorithm:
 - For D, there's no change — an existing path has length 20, which is less than the length of the path we're finding through C:



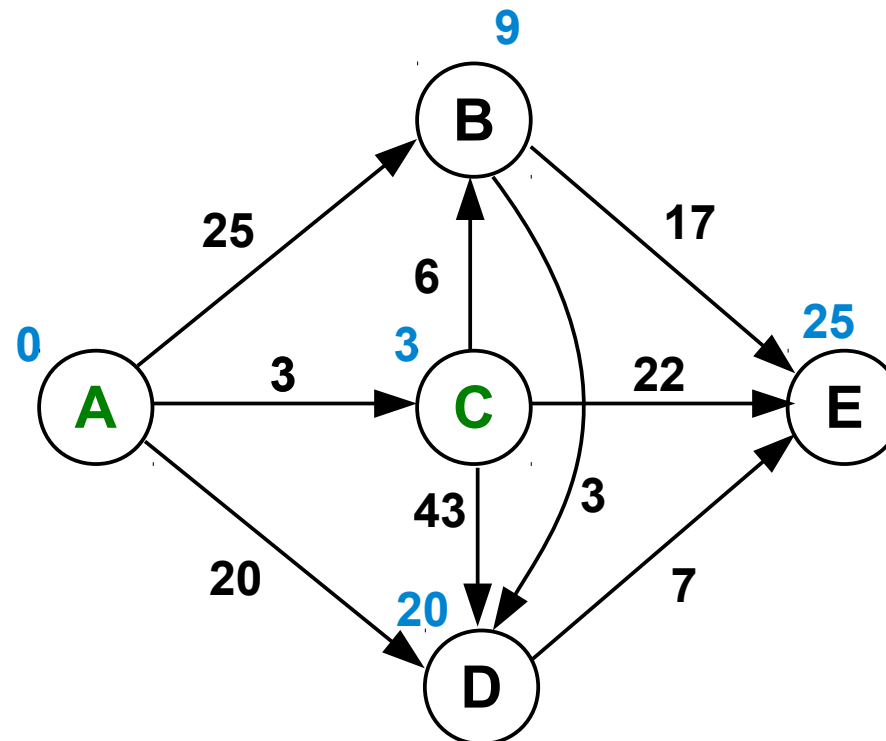
Dijkstra's Algorithm

- Dijkstra's Algorithm:
 - Finally, for E, since it is unmarked (distance ∞), we set its distance to $3 + 22$:



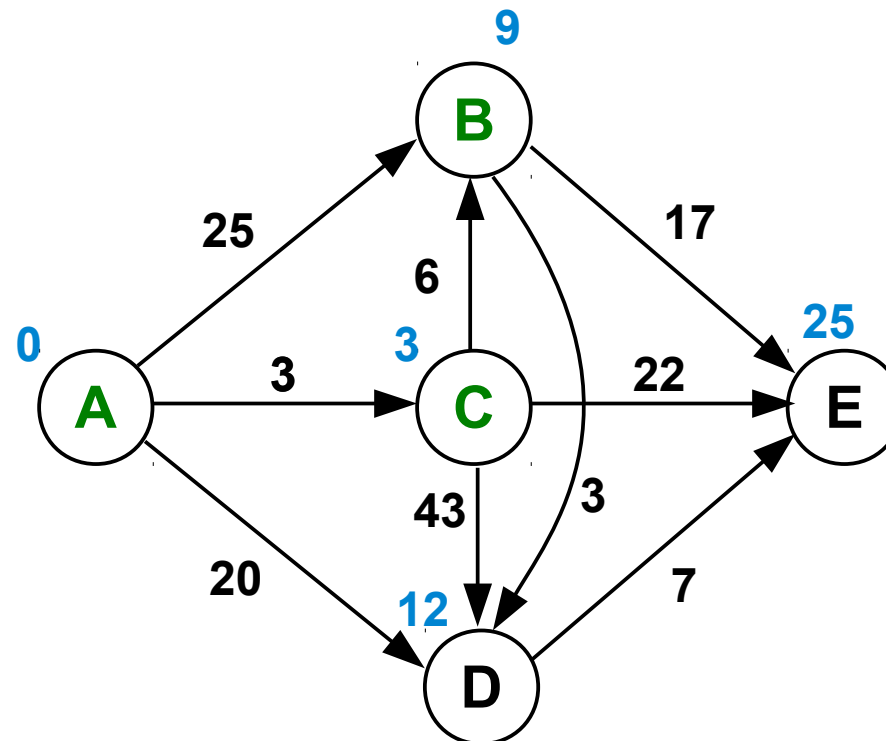
Dijkstra's Algorithm

- Dijkstra's Algorithm:
 - From the unvisited vertices (B, D, E), the one with smallest distance is B, so we select that one, set it as visited and update its neighbours' distances:



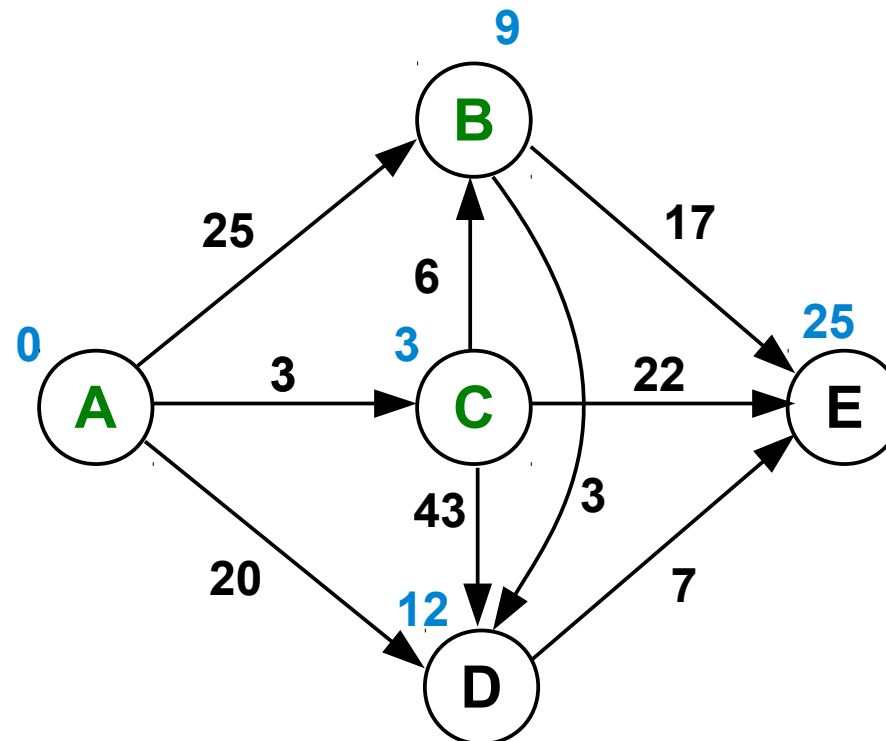
Dijkstra's Algorithm

- Dijkstra's Algorithm:
 - From the unvisited vertices (B, D, E), the one with smallest distance is B, so we select that one, set it as visited and update its neighbours' distances:



Dijkstra's Algorithm

- Dijkstra's Algorithm:
 - And again, should we stop to think about (and prove?) why this works in this more general case?

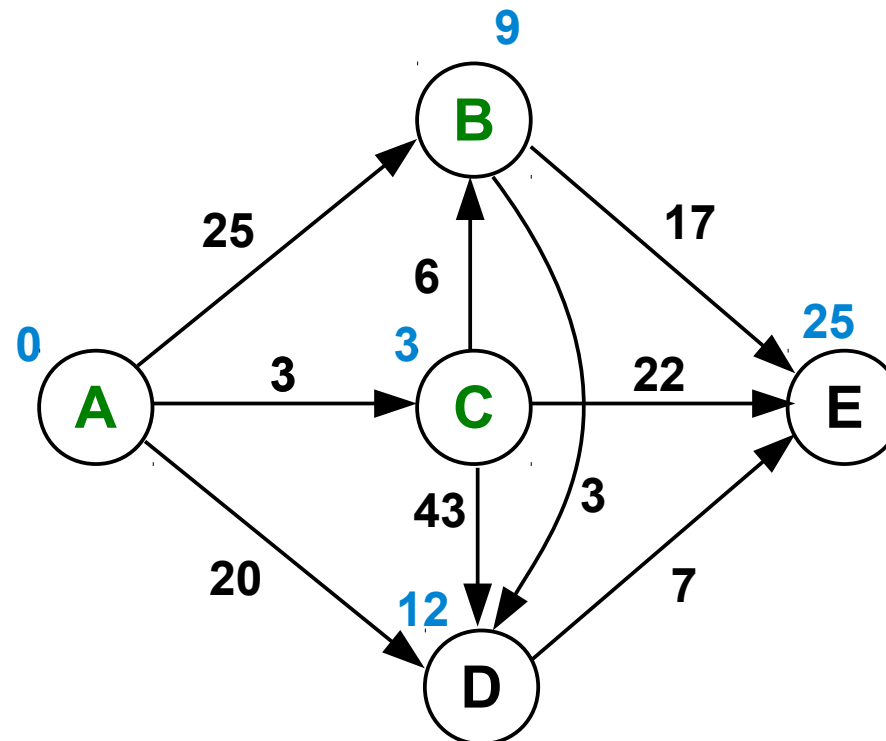


Dijkstra's Algorithm

- Dijkstra's Algorithm:
 - Let's prove it again by contradiction. Let v be the unvisited vertex with smallest distance (the one we're selecting), and let d be its distance.
 - Assume, for a contradiction, that there exists some other (shorter) path to v . Such path would have to go through some of the other unvisited vertices (then continue from there, and arrive at v)
 - But because v is the unvisited vertex with smallest distance, any such other path would have length $\ell = d' + \text{something}$, with $d' > d \Rightarrow \ell > d$ — contradicting the assumption that this other path is shorter than d .

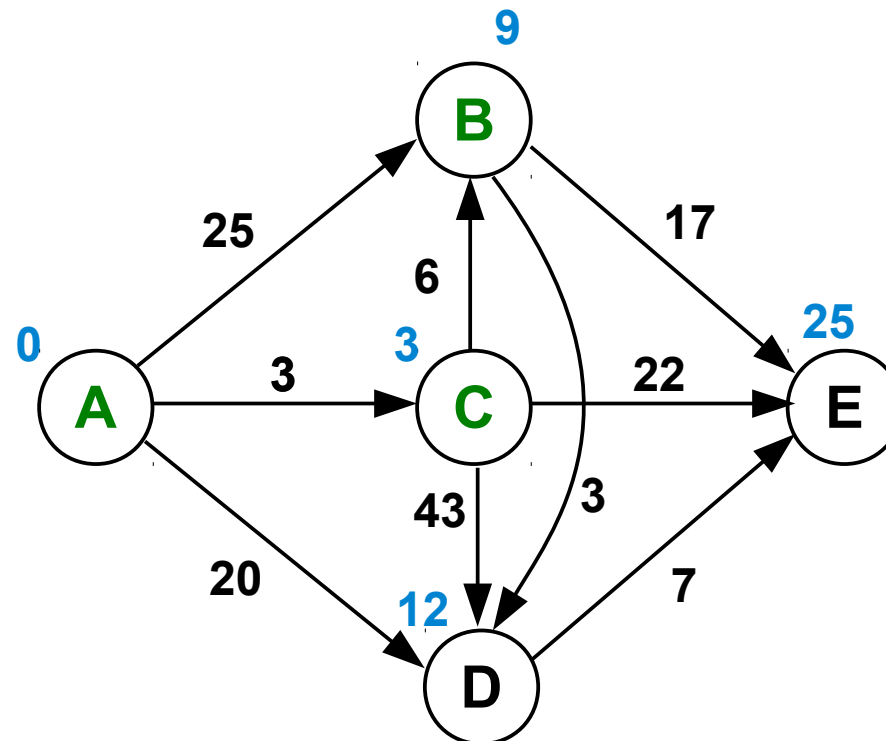
Dijkstra's Algorithm

- Dijkstra's Algorithm:
 - Back to vertex B — we just set it as visited and marked D's distance as 12 ($9 + 3 = 12 < 20$)



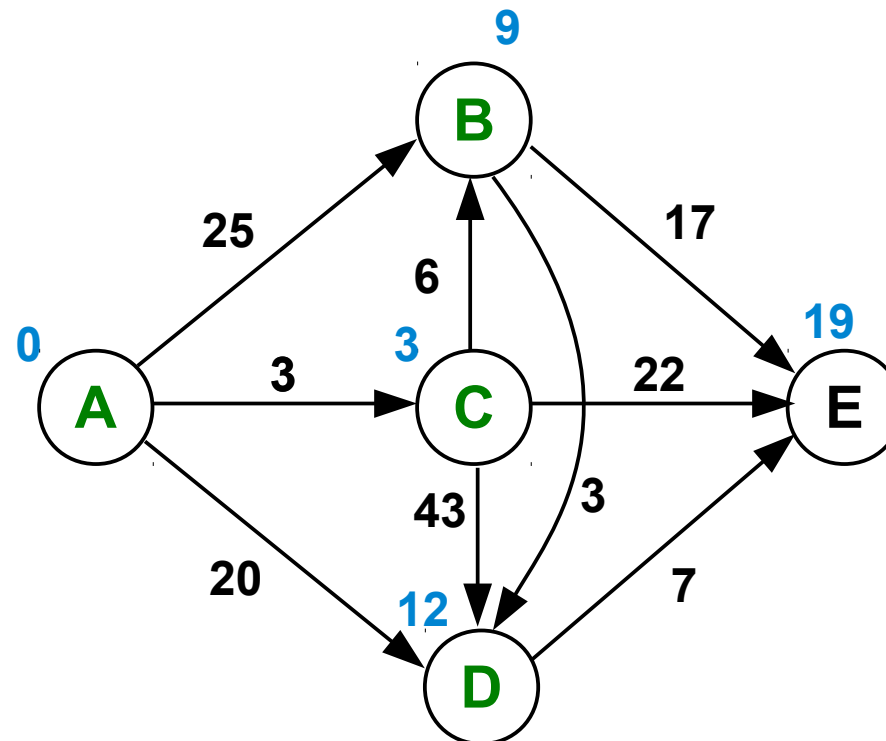
Dijkstra's Algorithm

- Dijkstra's Algorithm:
 - Then, select D, set it as visited and update its neighbour's distance:



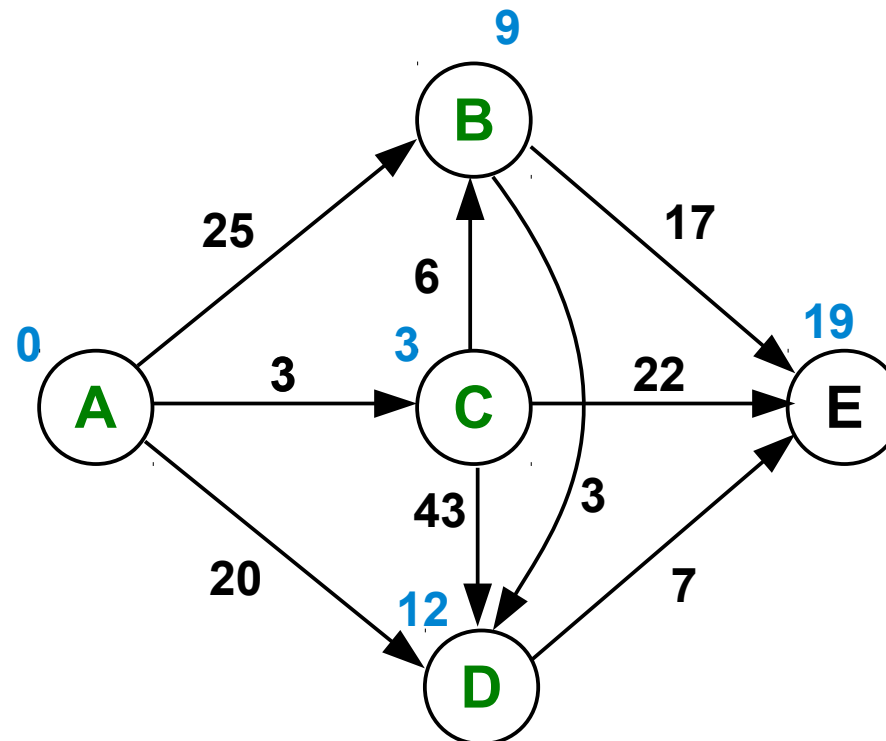
Dijkstra's Algorithm

- Dijkstra's Algorithm:
 - Then, select D, set it as visited and update its neighbour's distance:



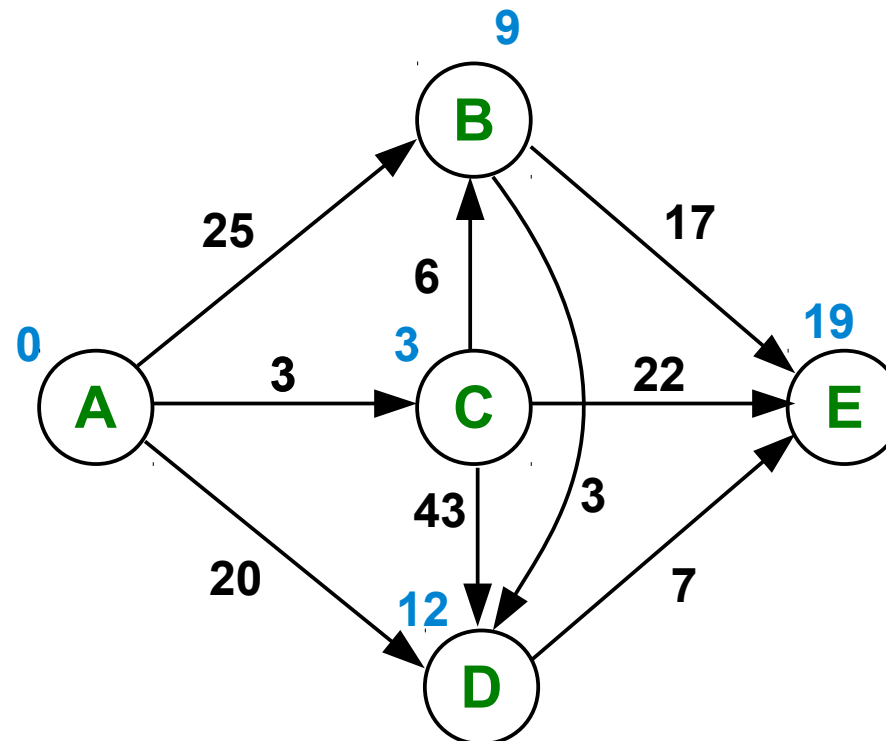
Dijkstra's Algorithm

- Dijkstra's Algorithm:
 - At the next iteration, select E, set it as visited, and then of course that means that execution completes:



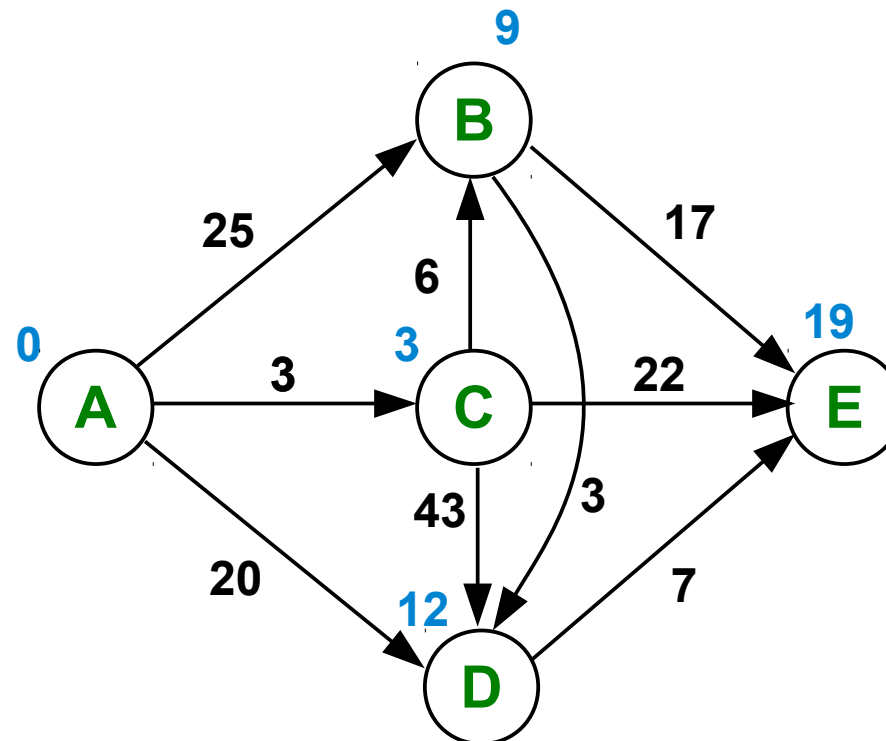
Dijkstra's Algorithm

- Dijkstra's Algorithm:
 - So, the shortest path has length 19, and it is anyone?



Dijkstra's Algorithm

- Dijkstra's Algorithm:
 - A – C – B – D – E ?? Really?? How do you know?

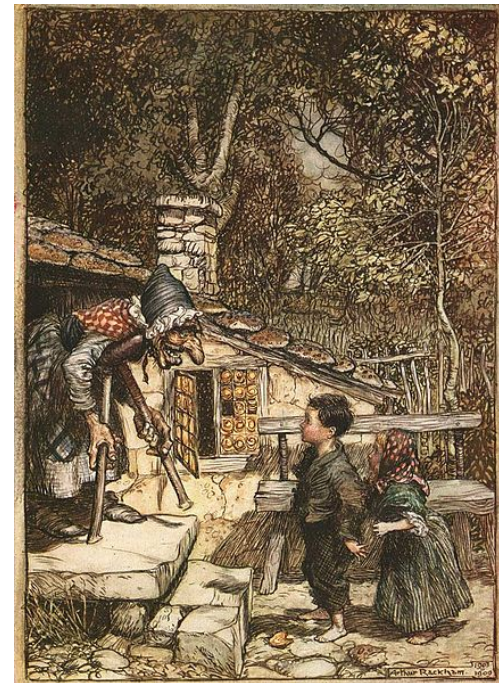


Dijkstra's Algorithm

- Dijkstra's Algorithm:
 - So, what's missing in the description of the algorithm?

Dijkstra's Algorithm

- Dijkstra's Algorithm:
 - So, what's missing in the description of the algorithm? (Hint: you really should do better than Hansel and Gretel — you don't want the birds to eat your shortest path!)

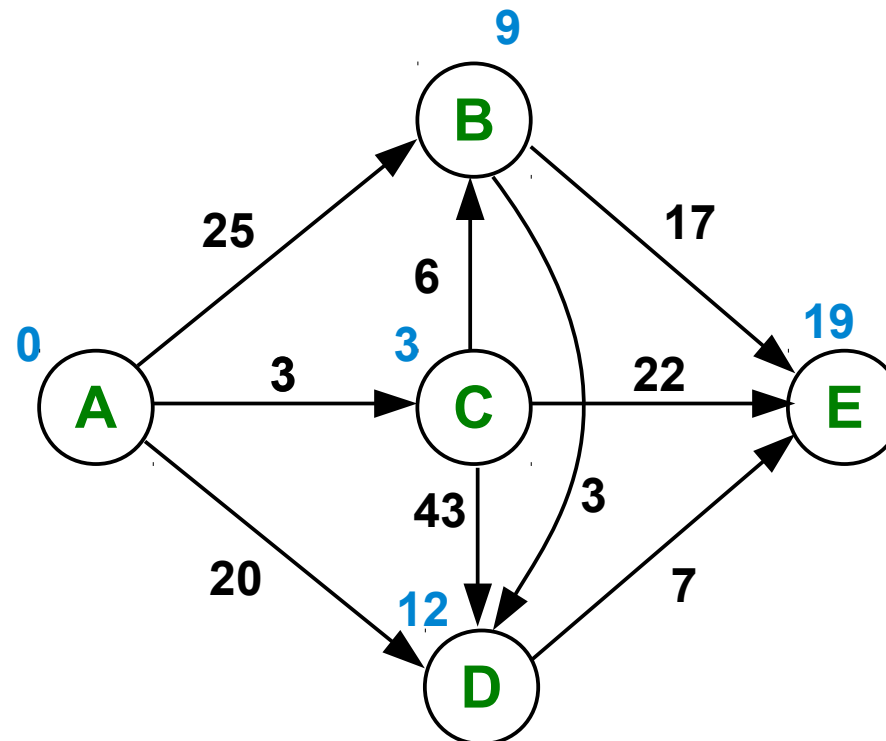


Dijkstra's Algorithm

- Dijkstra's Algorithm:
 - The problem is that we're going through the shortest path, but not remembering it — so, we need the marks (the “bread crumbs”) that will allow us to trace back the path from the target vertex back to the starting one.
 - Every time that we mark a vertex, we set a pointer (associated to each vertex) pointing to the vertex where the path comes from.
 - When overwriting a distance, we of course overwrite the pointer as well !!

Dijkstra's Algorithm

- Dijkstra's Algorithm:
 - In our example, E will point back to D, which points back to B, then back to C, then back to A, so we have our shortest path !



Summary

- During today's class:
 - We presented and discussed the problem of shortest path in a directed weighted graph.
 - Talked about some of its applications.
 - We introduced Dijkstra's shortest path algorithm
 - Discussed how and why it works
 - Went through an example of execution.
 - Noticed that we have to do better than Hansel and Gretel to avoid the birds eating our bread crumbs marking the shortest path !!