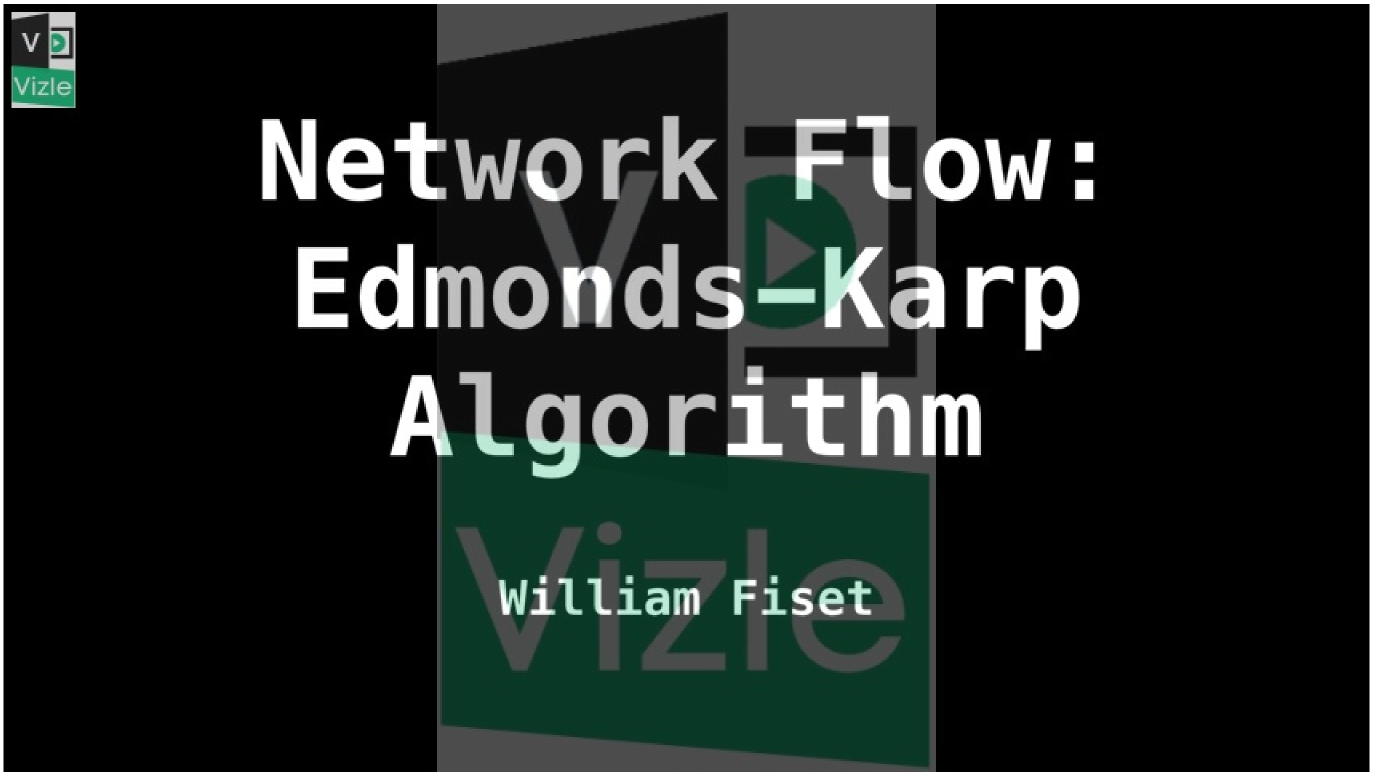
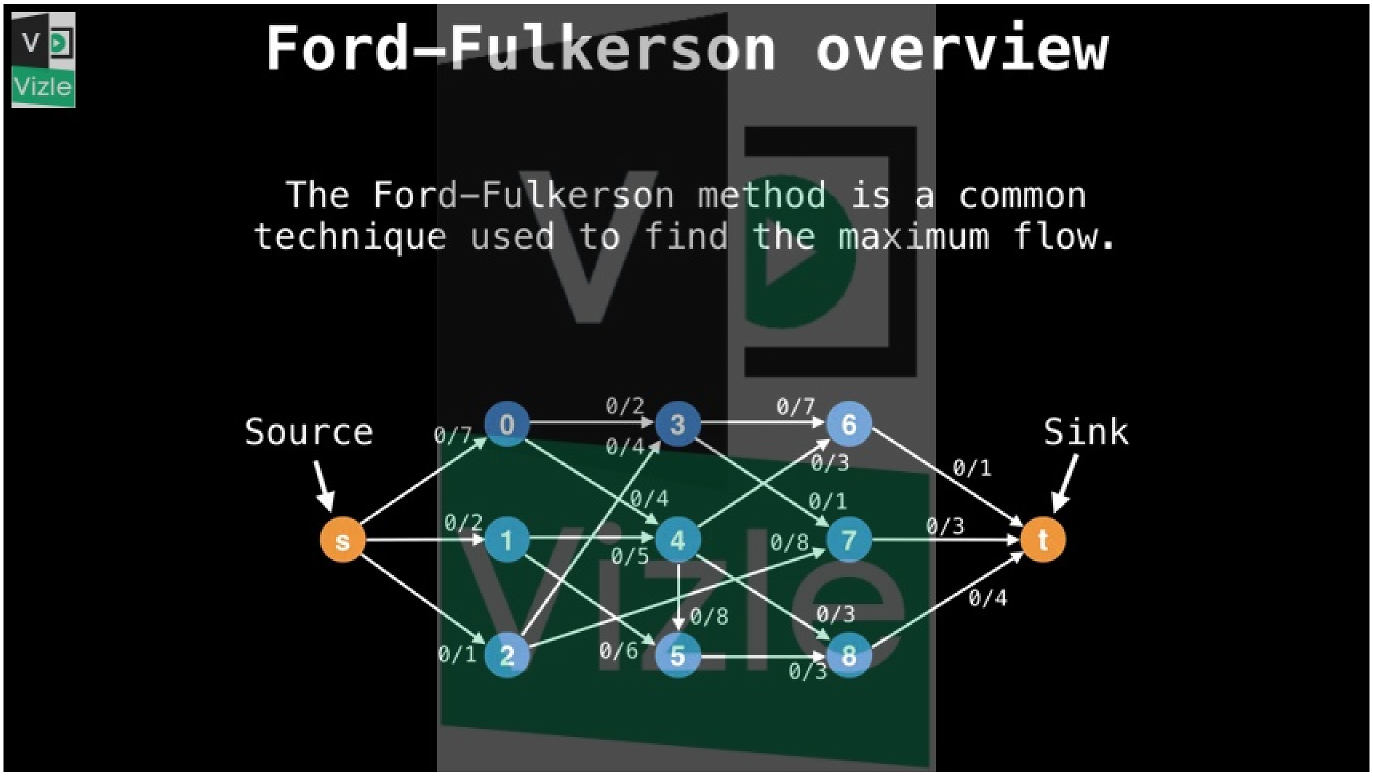
[Music] hello and welcome my name is William and

today we're going to probe even further into network flow we're going to be

talking about a specific implementation



of the ford-fulkerson method which is the edmonds-karp algorithm edmonds-karp is another maxim flow algorithm which uses a different technique to find augmenting paths through the flow graph before we get started let me give you a refresher on what we're trying to do we

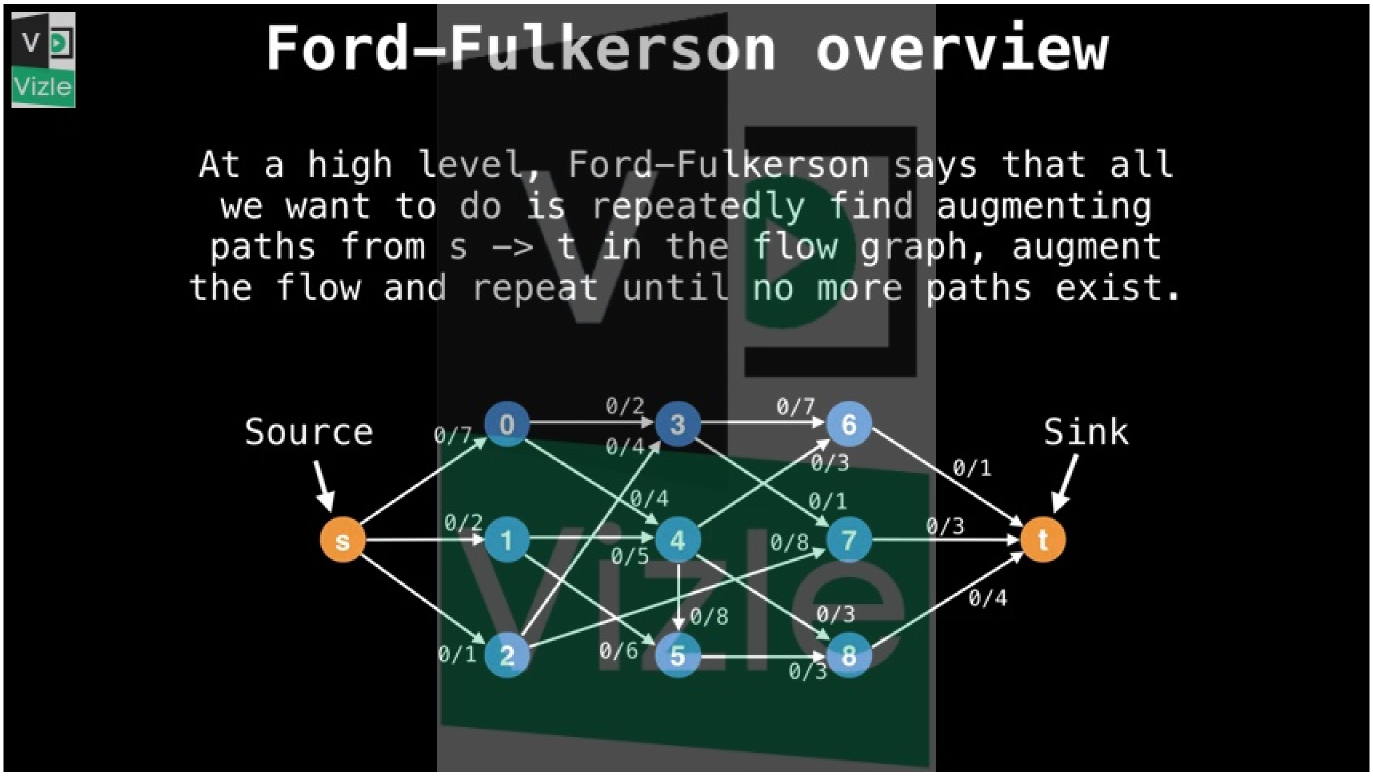


are trying to find the maxim flow on a flow graph because we know that finding

the maxim flow is really useful for finding bipartite matching and also to

solve a whole host of problems so far we've looked at one other technique to

find the maxim flow which is to use

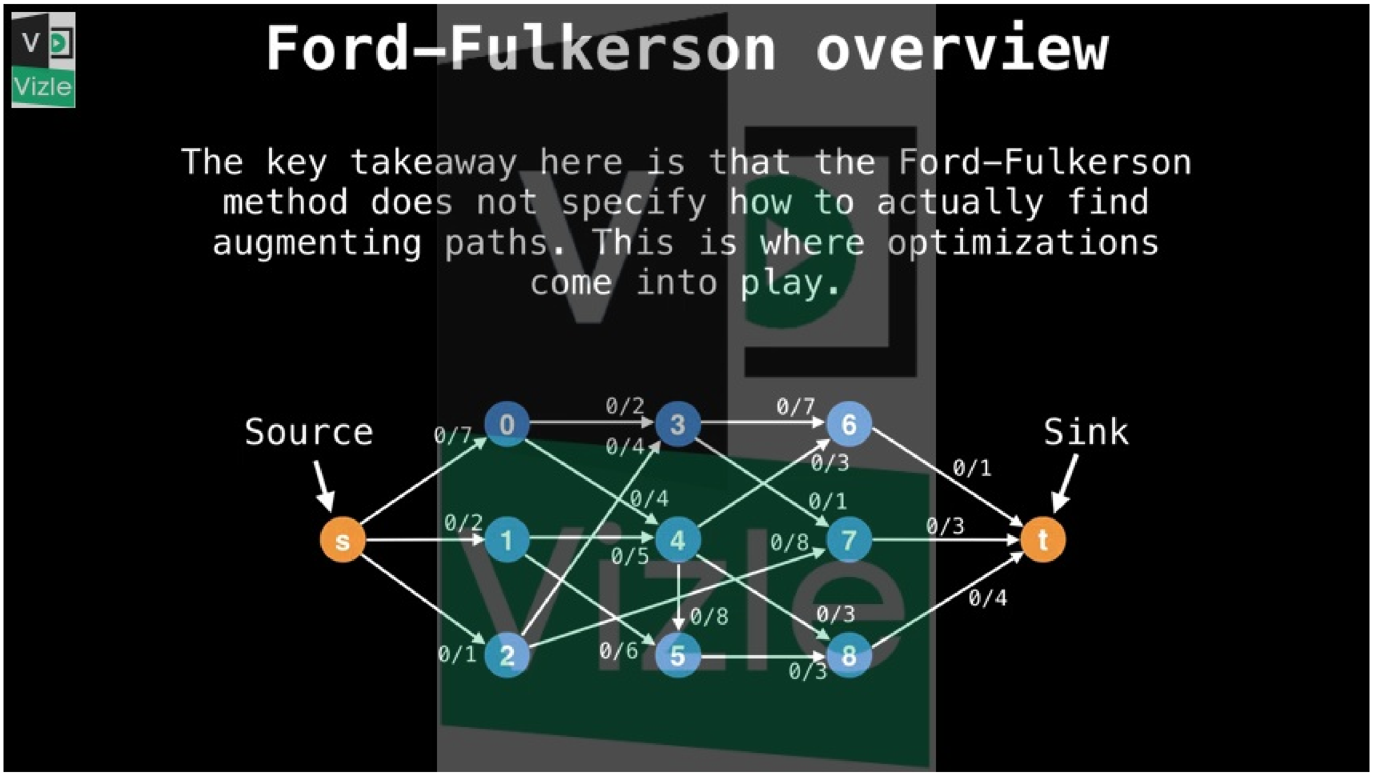


the ford-fulkerson method with a depth-first search at a high level it

says that all we want to do is repeatedly find augmenting paths from

the source to the sink augment the flow and then repeat this process until no

more paths exist the key takeaway here is that the ford-fulkerson method does

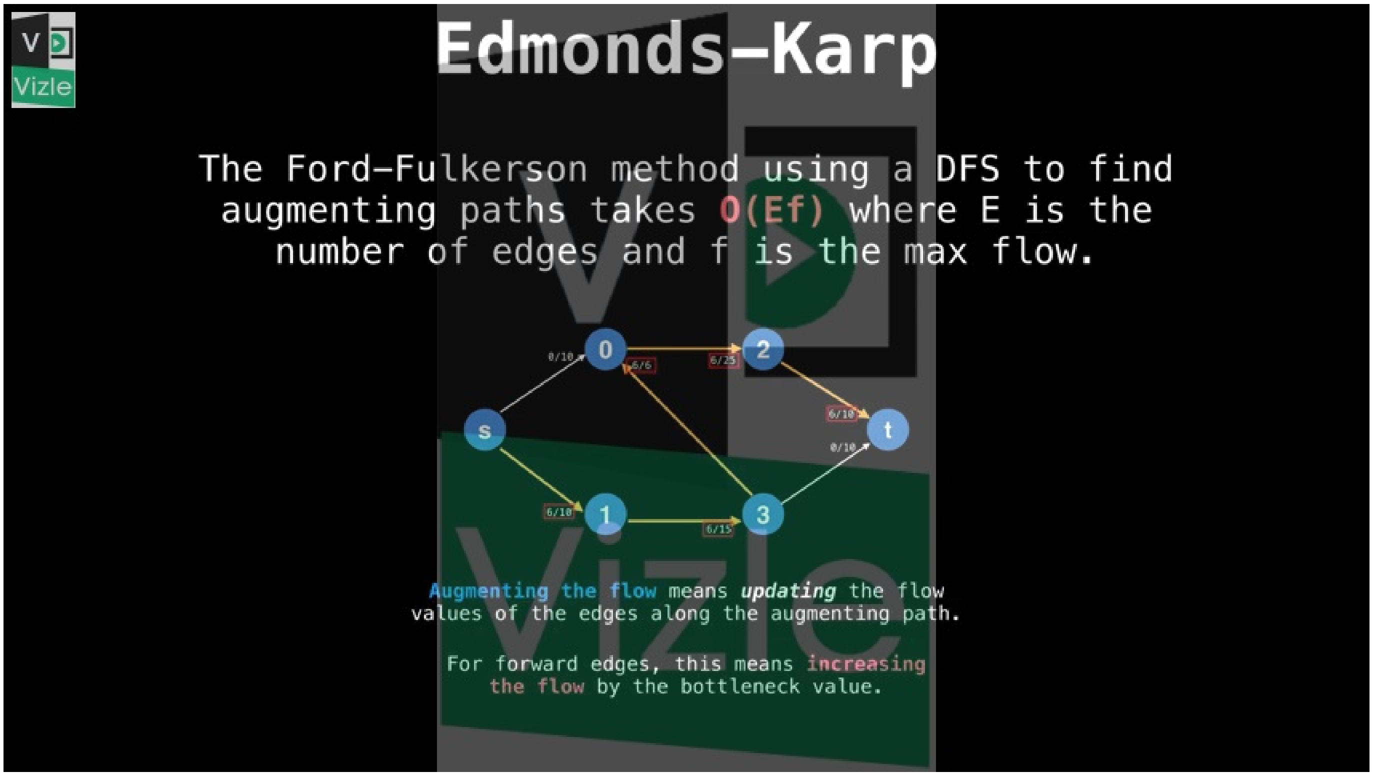


not specify how to actually find these augmenting paths so this is where we can

optimize the algorithm a few videos ago we saw that the ford-fulkerson method

can be implemented with a depth-first search to find the maxim flow however

the pitfall with that technique was that

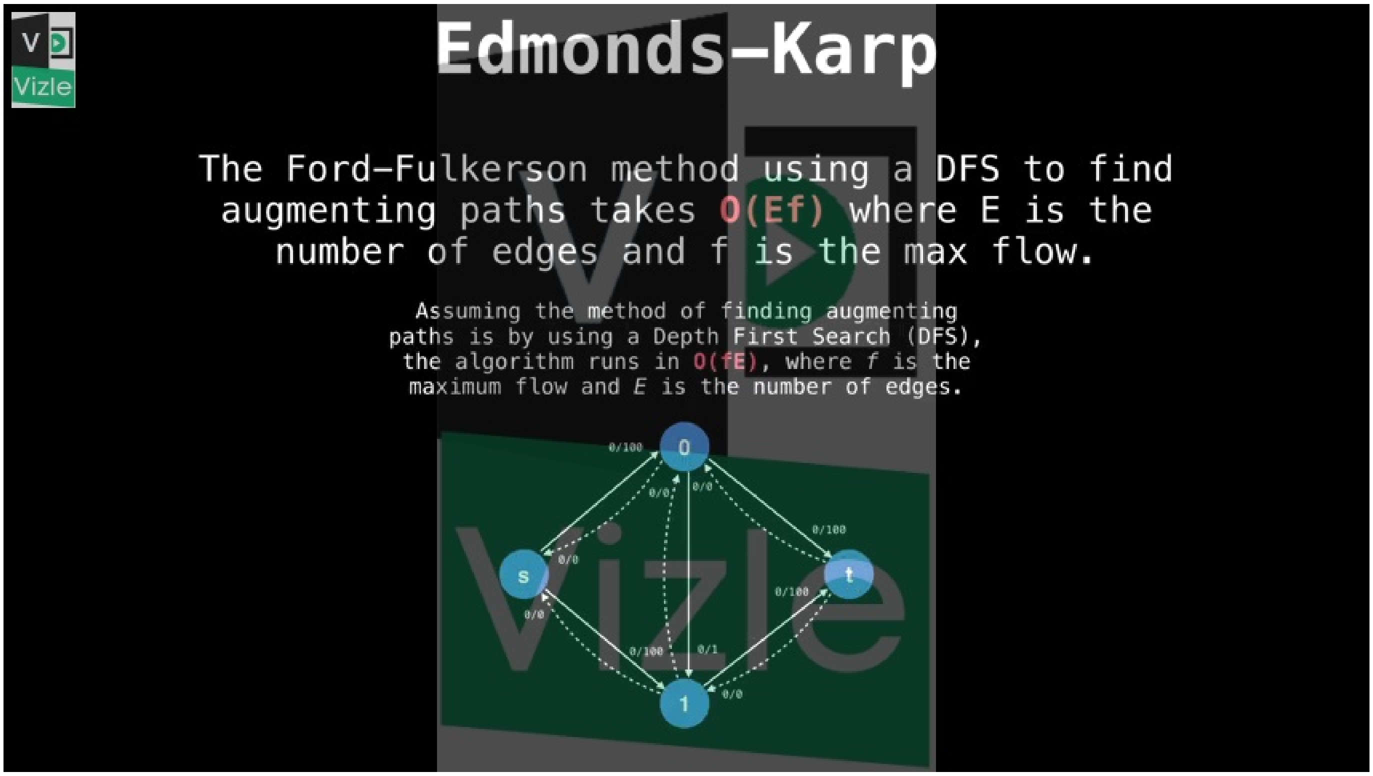


the time complexity depended on the capacity values of the edges in the

graph this is because the depth first search picks edges to traverse in such a

way that we might only ever be able to push one unit of flow in each iteration

this is really bad and can kill the time



complexity even though it's highly unlikely to happen in practice but it's

absolutely something we want to avoid should it happen right now the time

complexity afford Folker with a depth-first search is Big O of e

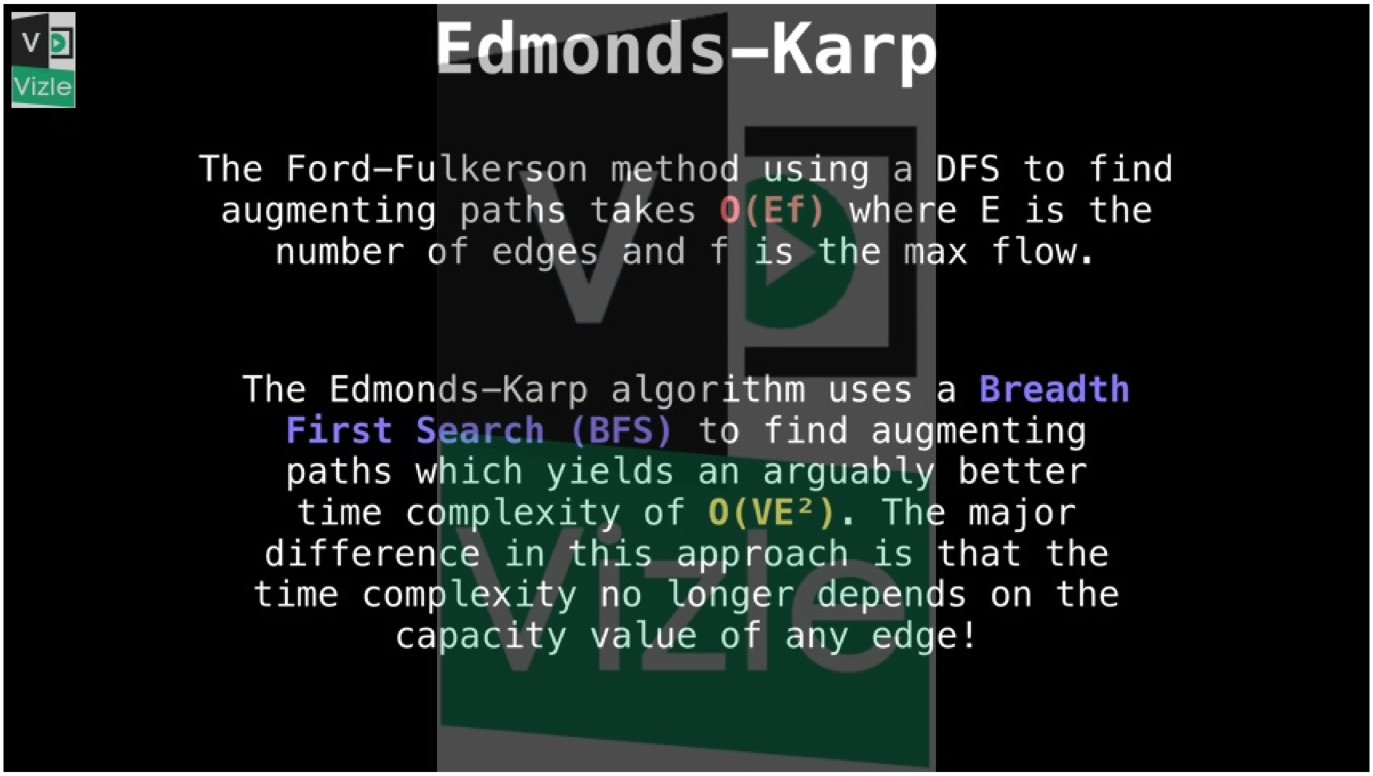
times F where E is the nber of edges and F is the maxim flow

the idea behind edmonds-karp says that instead of using a depth-first search to

find augmenting paths we should use a breadth-first search instead to get a

better time complexity Big O of V times E squared

may not look like a better time



complexity but it actually is what's different is that the time complexity

while it might not look great does not depend on the capacity value of any edge

in the flow graph which is crucial we call such an algorithm that doesn't

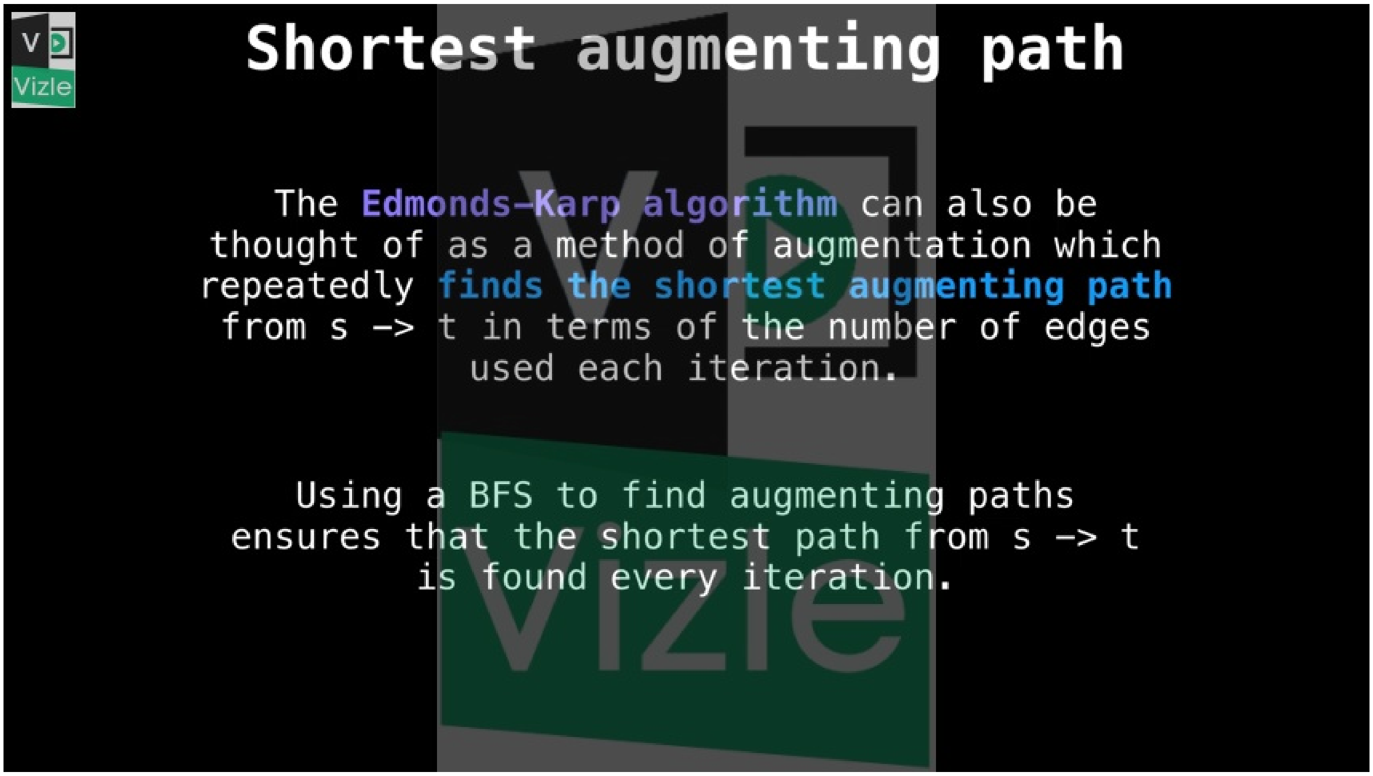
depend on the actual input values a strongly polynomial algorithm and that's

exactly what edmonds-karp is and why it was so revolutionary at the time

edmonds-karp can also be thought of as an algorithm which finds the shortest

augmenting path from s to t that is in terms of the nber of edges used in

each iteration using a breadth-first

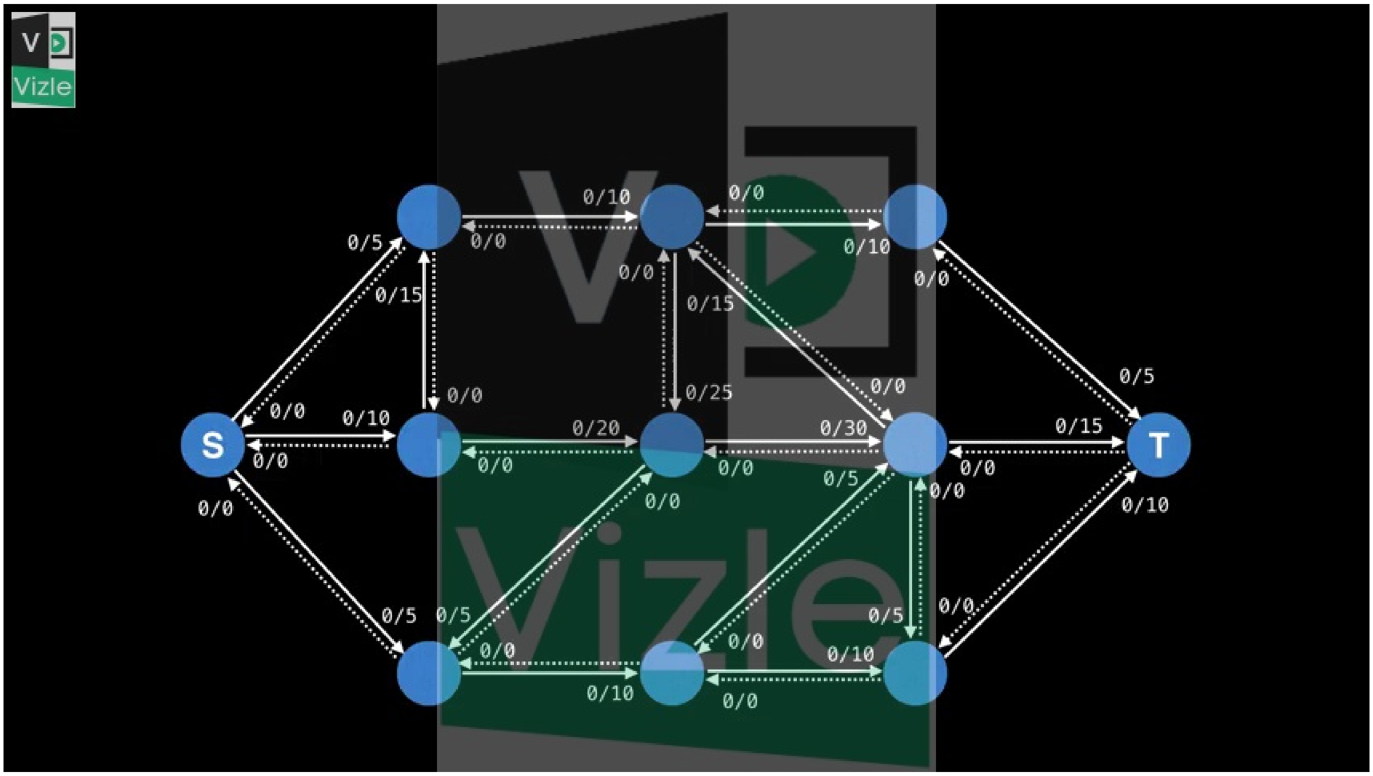


search during edmonds-karp ensures that we find the shortest path this is a

consequence of each edge being unweighted when I say unweighted I mean

that as long as the edge has a positive capacity we don't distinguish it between

one edge being any better or worse than any other edge now let's look at why we



might care about using edmonds-karp suppose we have this flow graph and we

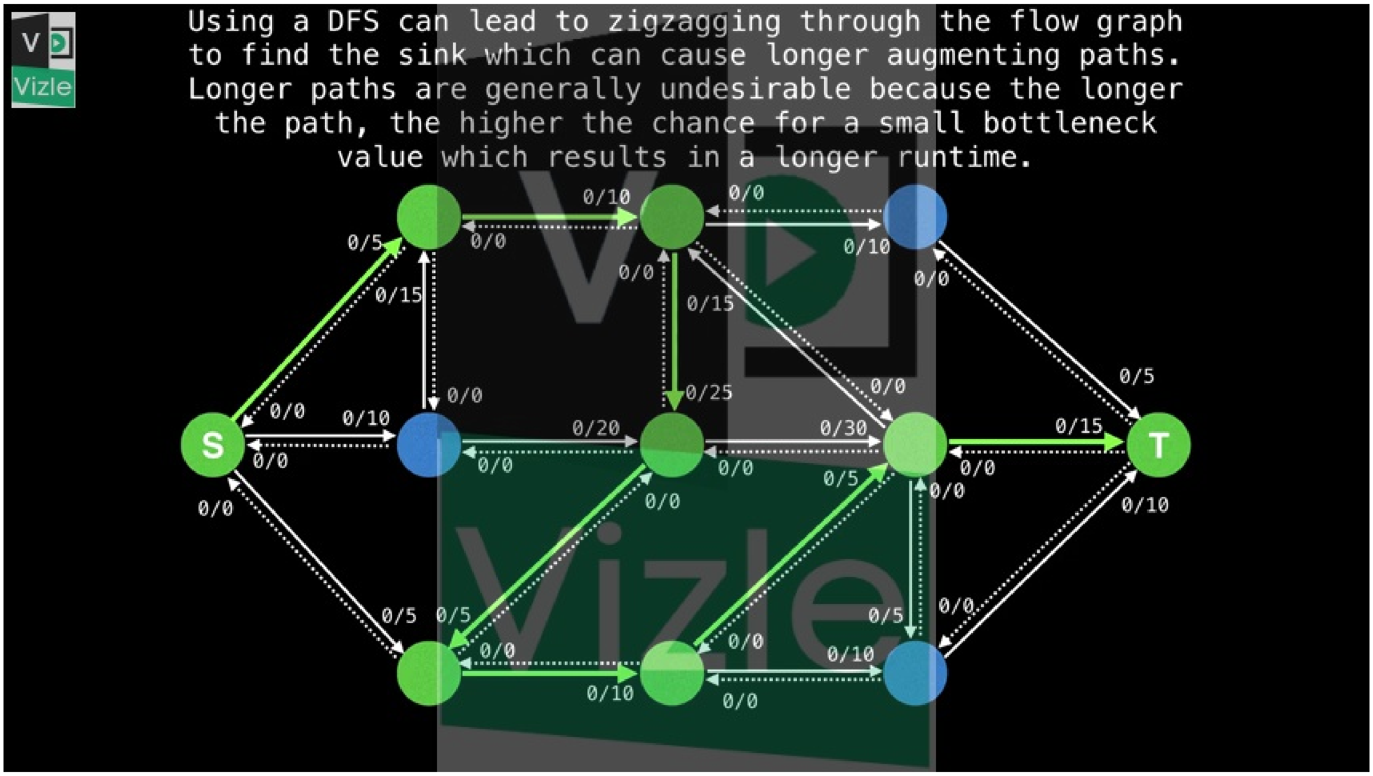
want to find what the maxim flow is if we're using a depth-first search

we might do something like this start at the source and do a random depth-first

search for words so after a love is exactly the flow

graph we are able to find the sink as we just saw a depth-first search has the

chance to cause long augmenting paths and longer paths are generally

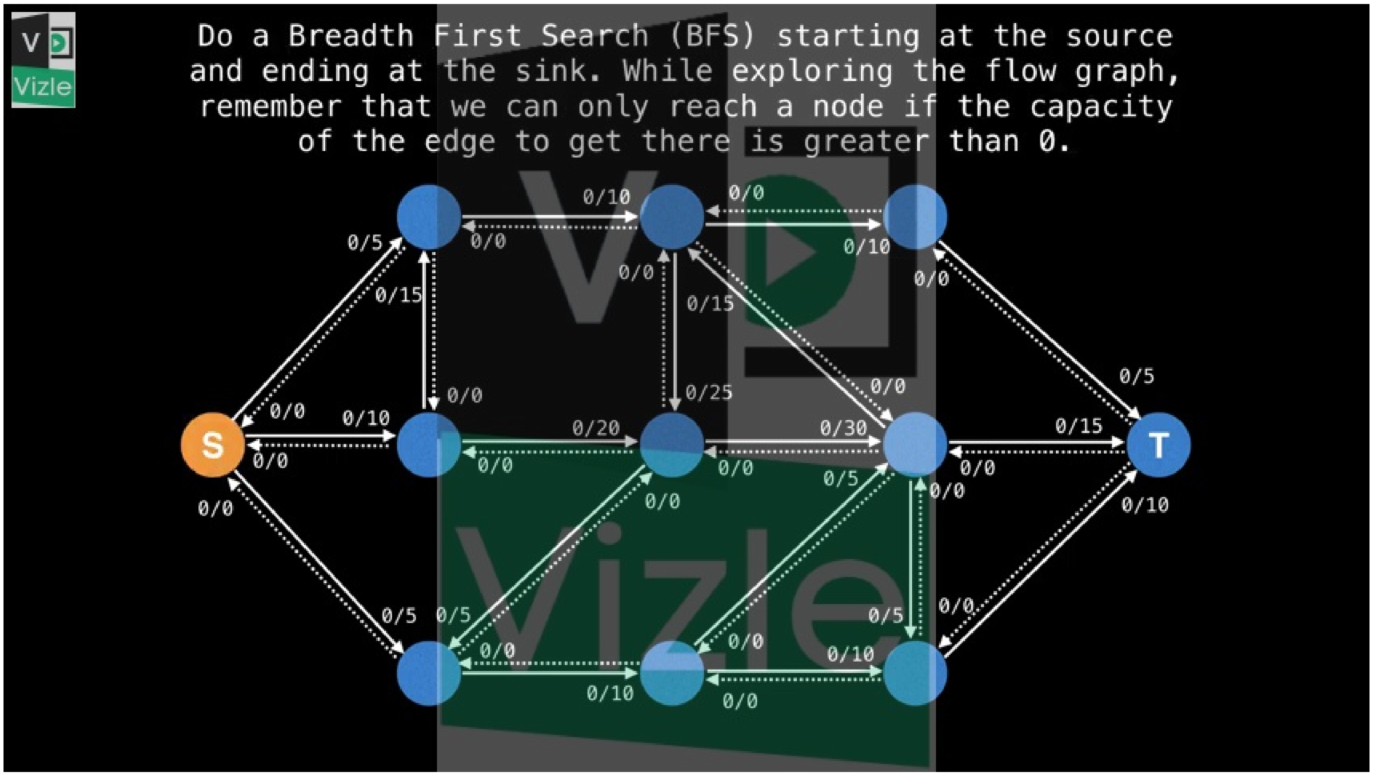


undesirable because the longer the path the higher the chance for a small bowel

mech value which results in a longer run time finding the shortest path from s to

T again in terms of nber of edges is a great approach to avoid the depth first

search worst case scenario and reduce the length of augmenting paths to find

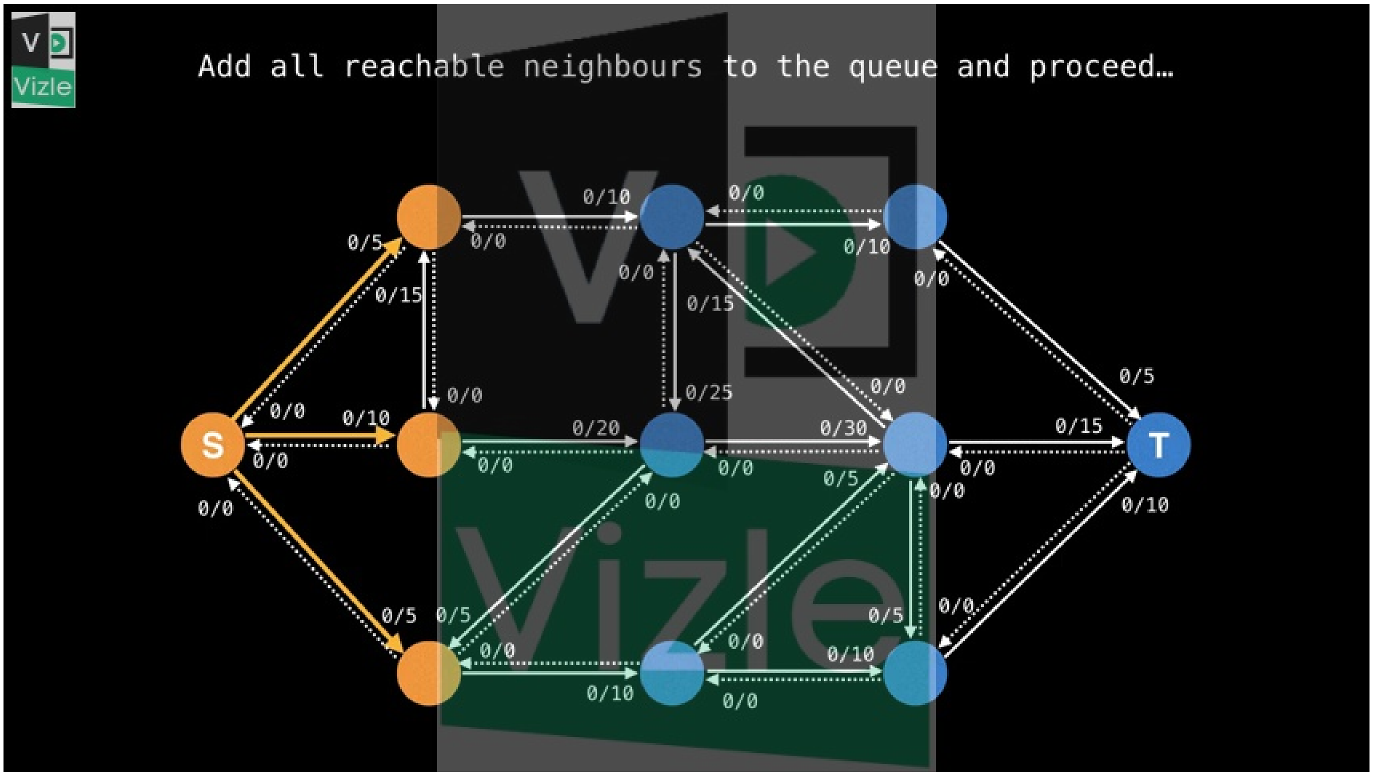


the shortest path from s to T do a breadth-first search starting at the

source and ending at the sink while exploring the flow graph remember that

we can only take an edge if the remaining capacity of that edge is

greater than zero in this example all edges outwards from s have a remaining



capacity greater than zero so we can add all the neighbors to the queue when

we're doing the breadth-first search step and then we keep going forwards so

add all reachable neighbors to the queue and continue and now the breadth-first

search has reached the sink so we can stop in the real algorithm we would stop

as soon as any of the edges reach the sink but just for symmetry I show three

edges here entering the sink while in a reality we would stop as soon as one of

them reaches the sink if we asse that the bottom edge made it to the sink

first and we retrace the path we get the following augmenting path but we didn't

just find any augmenting path we found a shortest length augmenting path so to

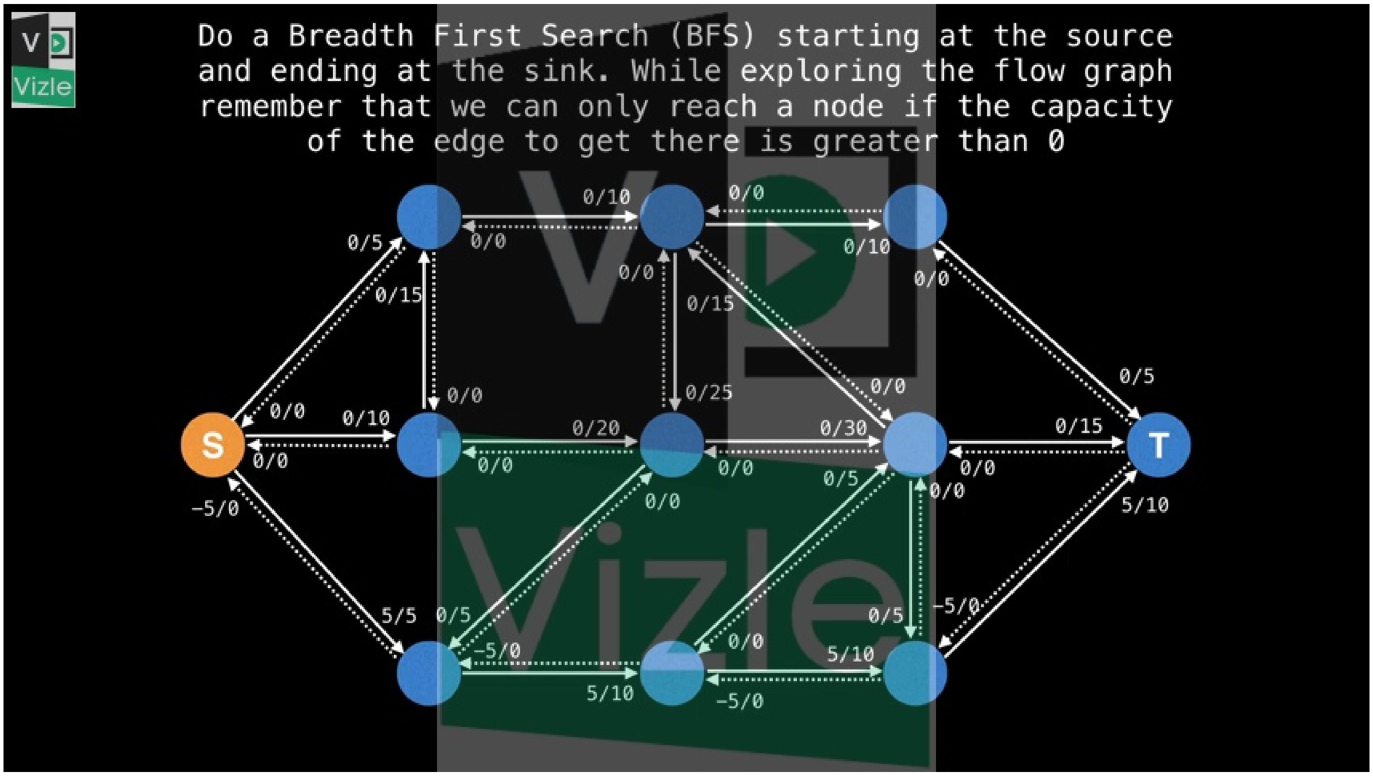
augment the flow do the usual find the bottleneck value by finding the smallest

remaining capacity of all the edges along the path then augment the flow

values along the path that by the bottleneck so that was the first path

however we're not done yet let's continue finding paths until the entire

graph is saturated recall that while exploring the flow graph we can only reach a node if the remaining capacity of the edge to get to that node

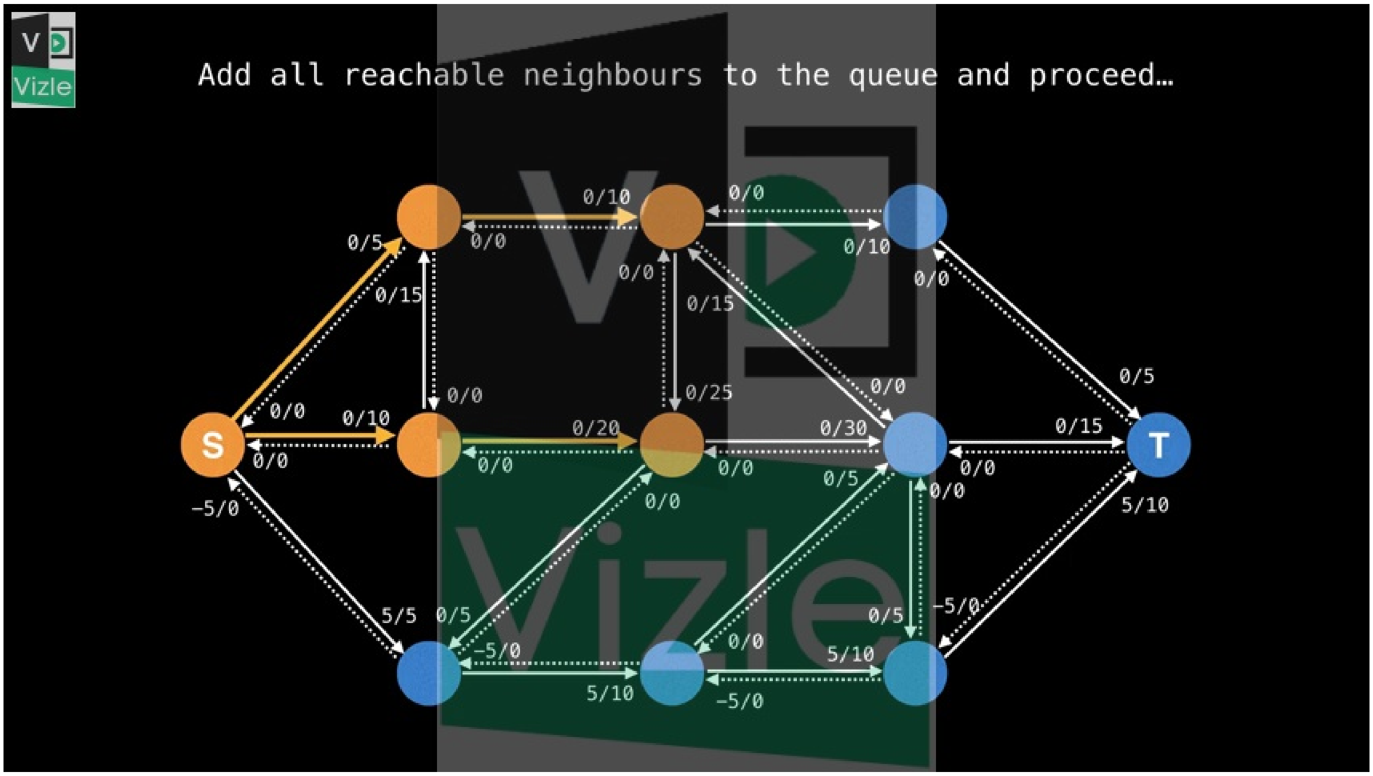


is greater than zero for instance all the reachable neighbors of the source

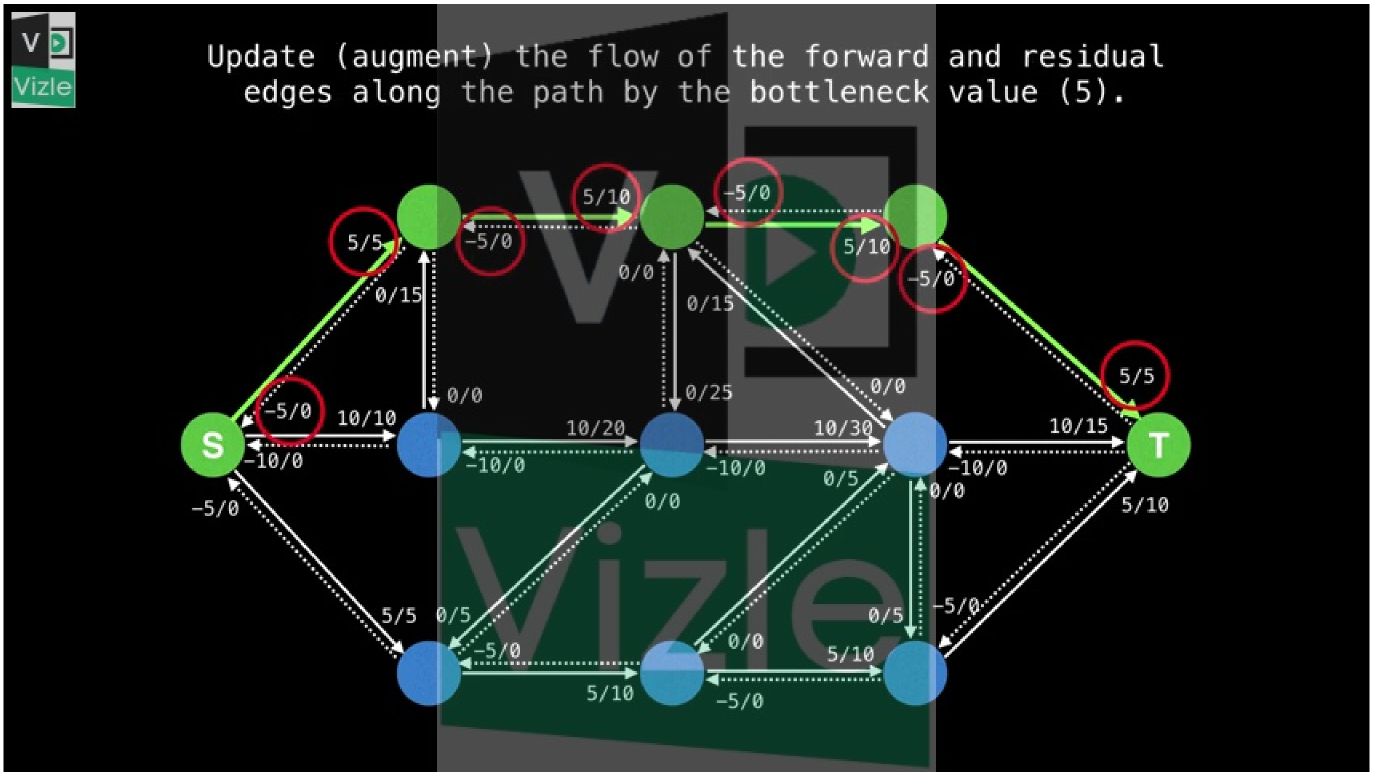
node in this case does not include the bottom-left node because the edge from

the source to the bottom-left node has a remaining capacity of zero all right

keep exploring until the sink is reached



and now we've reached the sink once more so find the bottleneck value along this path then use the bottleneck value to update the flow along the augmenting path don't forget to update the residual edges and we're still not done because there still exists another augmenting path so now there only exists one edge outwards from the source with a capacity greater than zero so it's the only edge we can take so we follow it there's also only one edge to follow from the second node because the other edges have a remaining capacity of zero and now the breadth-first search has reached the sink we can trace back the edges that were used we can find the bottleneck by finding the minim capacity along the path and also augment the flow and now

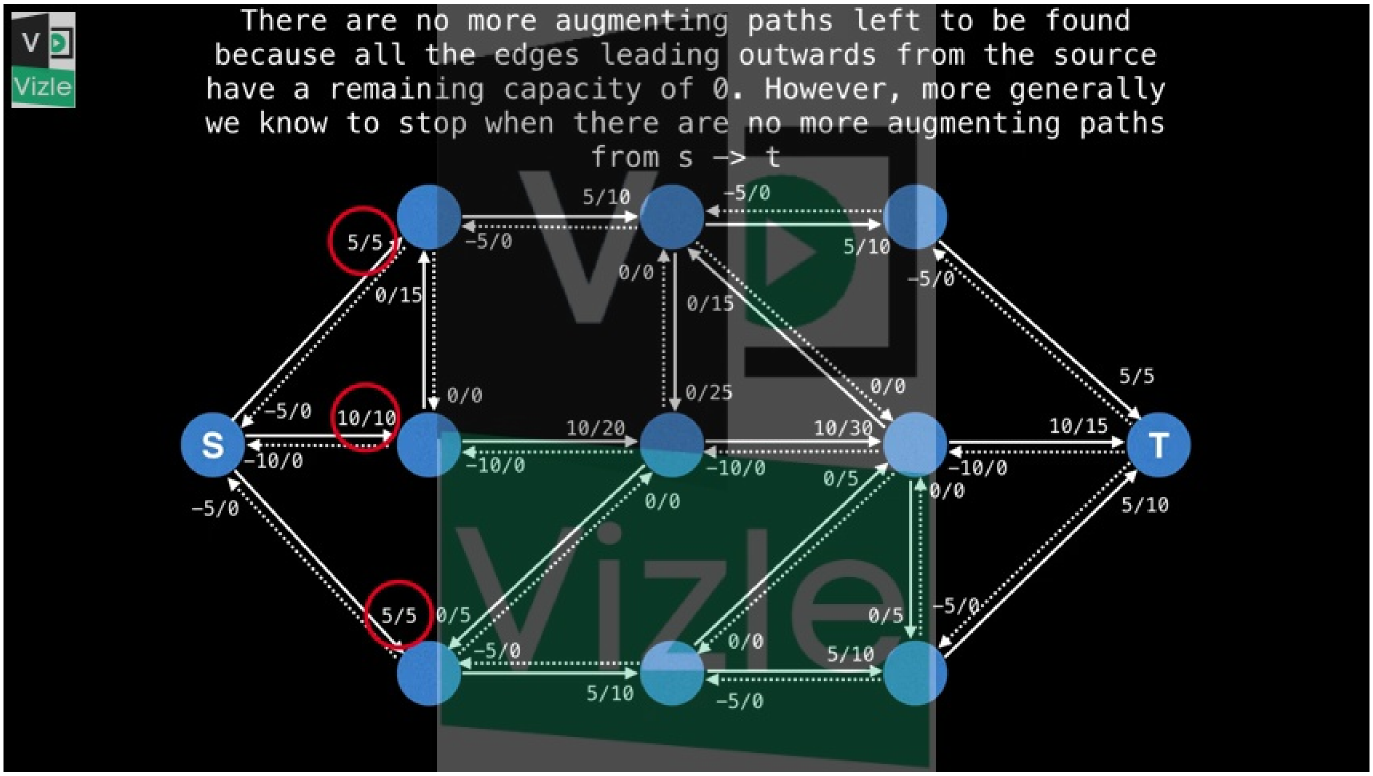


you can see that there are no more augmenting paths left to be found

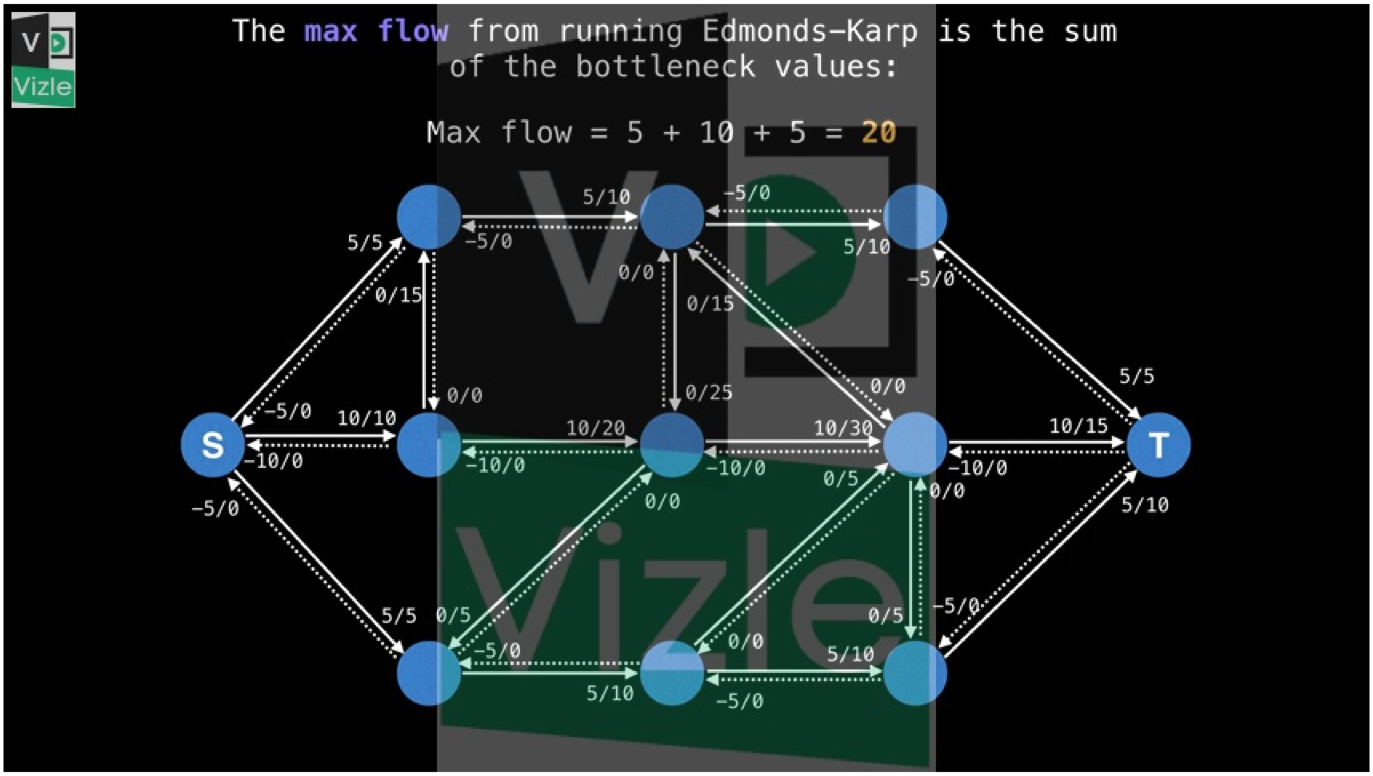
because all the edges leading outwards from the source have a remaining

capacity of the zero however more generally we know to stop edmonds-karp

when there are no more augmenting paths from s to t because we know we cannot



increase the flow any more if this is the case the maxim flow we get from running edmonds-karp is the s of the bottleneck values if you recall in the first iteration we were able to push 5 units of flow in the second iteration 10 units and in the last iteration 5 units for a total of 20 units of flow another

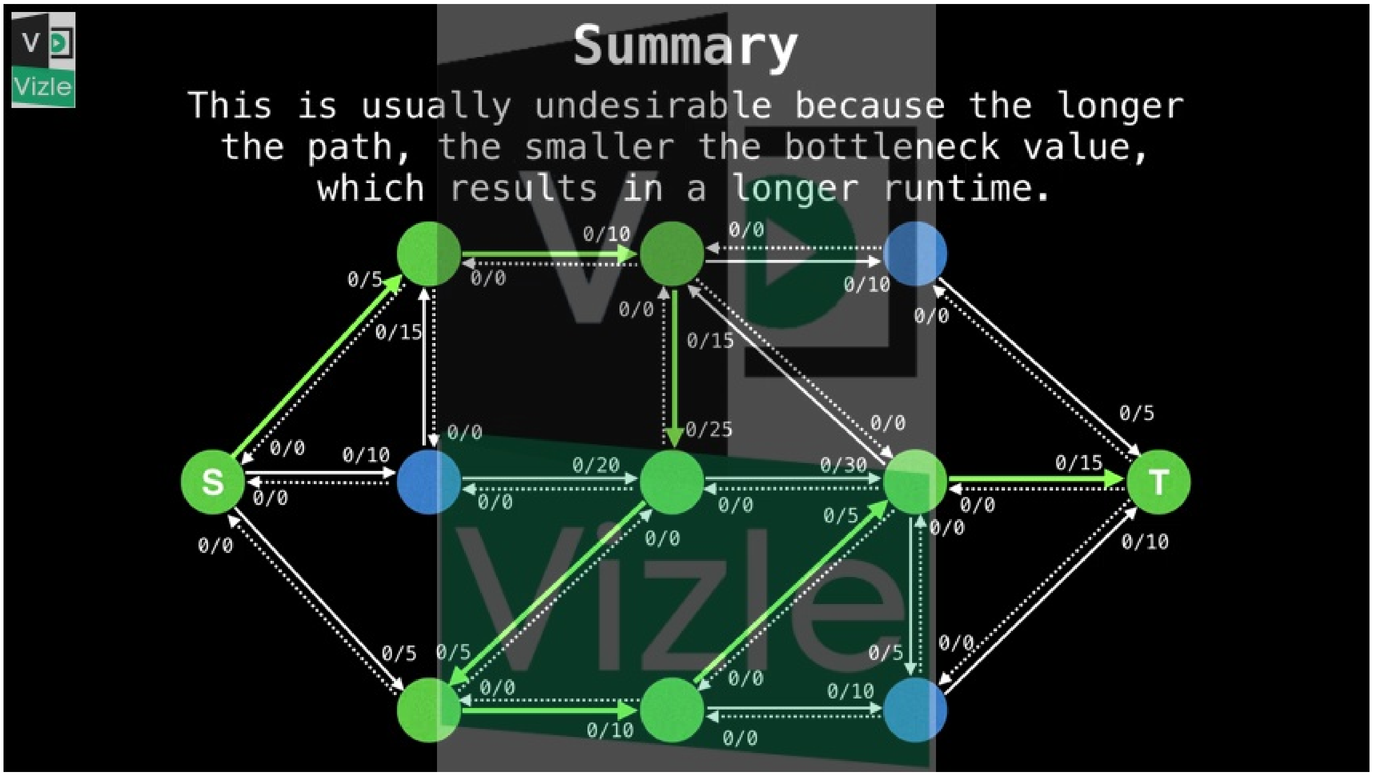


way to find the maxim flow is the s the capacity values going into the sink

which I have circled in red in smary this is what we learned using

a depth-first search on a flow graph can sometimes find a long windy path from

the source to the sink this is usually



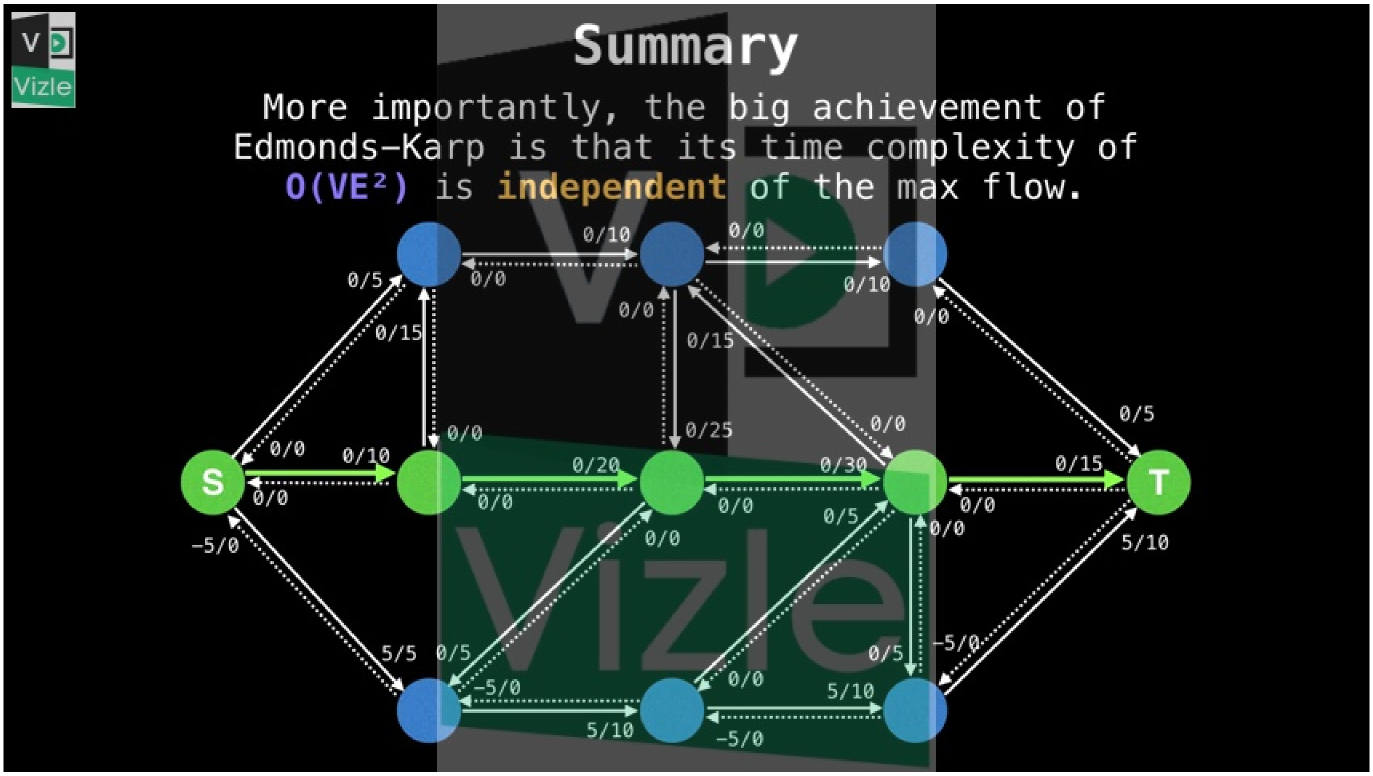
undesirable because the longer the path the smaller the bottleneck value and the

longer the runtime edmonds-karp tries to resolve this problem by finding the

shortest length augmenting paths from the source to the sink using a

breadth-first search however more importantly the big achievement of

edmonds-karp is that its time complexity



of Big O of V times e squared is independent of the max flow so it

doesn't depend on the capacity values of the flow graph and that's edmonds-karp

in a nutshell thank you for watching next video we'll

cover some source code for now please like this video if you learned something

and subscribe for more mathematics and


Next Video: Edmonds–Karp Source Code
Vizle


This PDF is generated automatically by Vizle.
Slides created only for a few minutes of your Video.
For the full PDF, please Login to Vizle.
https://vizle.offnote.co (Login via Google, top-right)
Stay connected with us:
Join us on Facebook, Discord, Quora, Telegram.