CS2040S

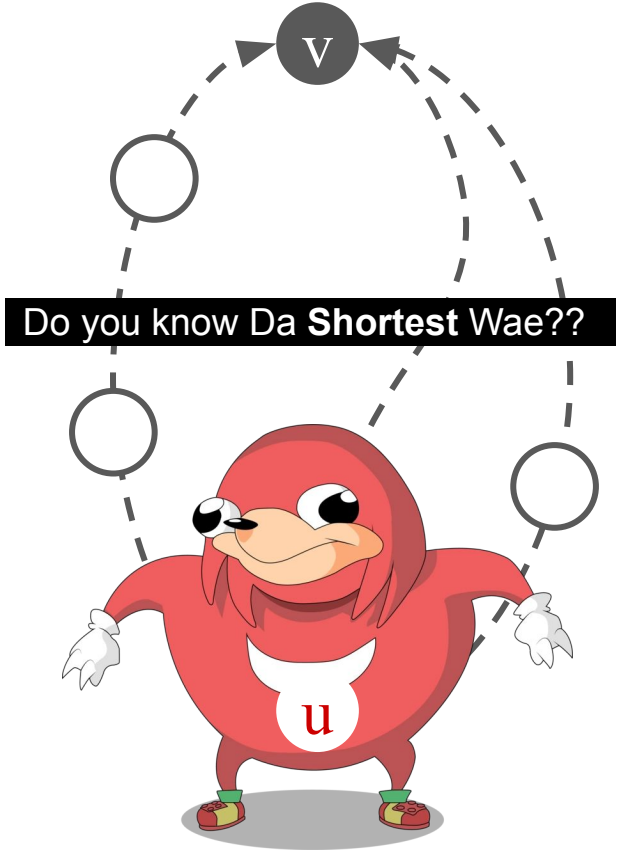
Recitation 7
AY22/23S2

Shortest Path Problems

Given a graph with weighted/unweighted edges,

what is $\delta(u, v)$, the **shortest path** from a vertex u to another vertex v?

Shortest means the path with lowest "length".



Shortest Path Problems

The definition of “length” varies from problem to problem:

- Sum of all edge weights in the path *Most common*
- Number of edges in the path. Applies for unweighted graphs or graphs with all edge weight being equal (uniformly-weighted)
- Sum of all *vertex weights* in the path
- *Product* of all edge weights in the path
    - Only applicable if all weights are positive

Shortest Path Problems

In general, 2 main types of shortest path problems:

Single Source Shortest Path (SSSP)

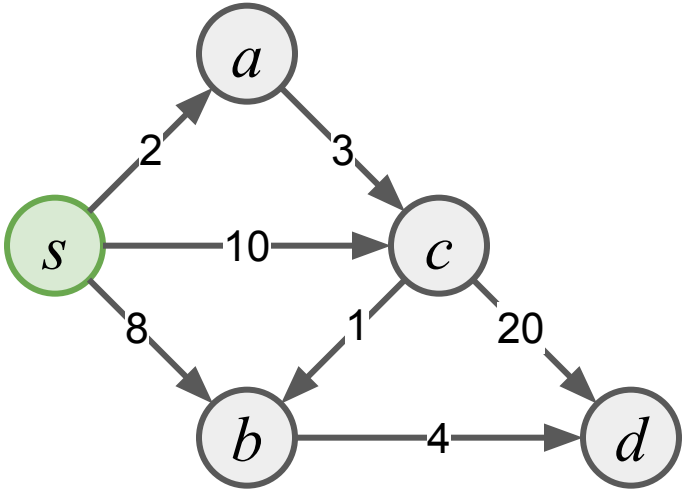
- Shortest path from a *single* source vertex s, to *every other vertex* in the graph

All Pairs Shortest Path (APSP)

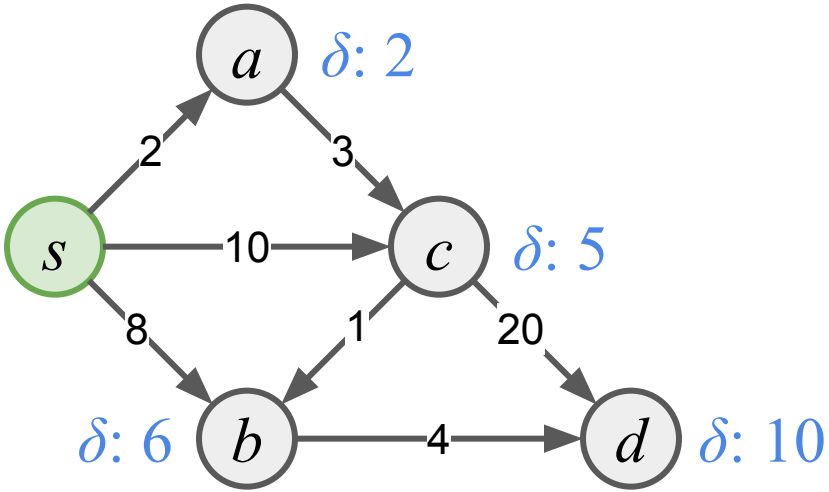
- Shortest path between *every pair* of vertices in the graph
- Just run SSSP on each of the V vertices in graph!

SSSP example

What are the SSSP lengths with s as the source vertex?



SSSP example



Recitation 7

Goals

- Model problems using graphs
- Transform graphs to utilize existing algorithms as blackboxes
- Identify and formulate shortest paths problems
- Appreciate the representational power of graphs

Graph modelling

When modelling a problem using a graph, you have to first answer these questions:

- What do the *vertices* represent?
- What do the *edges* represent?
  - Are they *directed*/*undirected*
    - What do the directions represent?
  - Are they *weighted*/*non-weighted*?
    - What do the edge weights represent?

Graph modelling

Once those questions are answered, then the the next round of questions to ask are:

- What is the graph problem we want to solve?
- What DS shall we use to encode our graph?
  - Adjacency list
  - Adjacency matrix

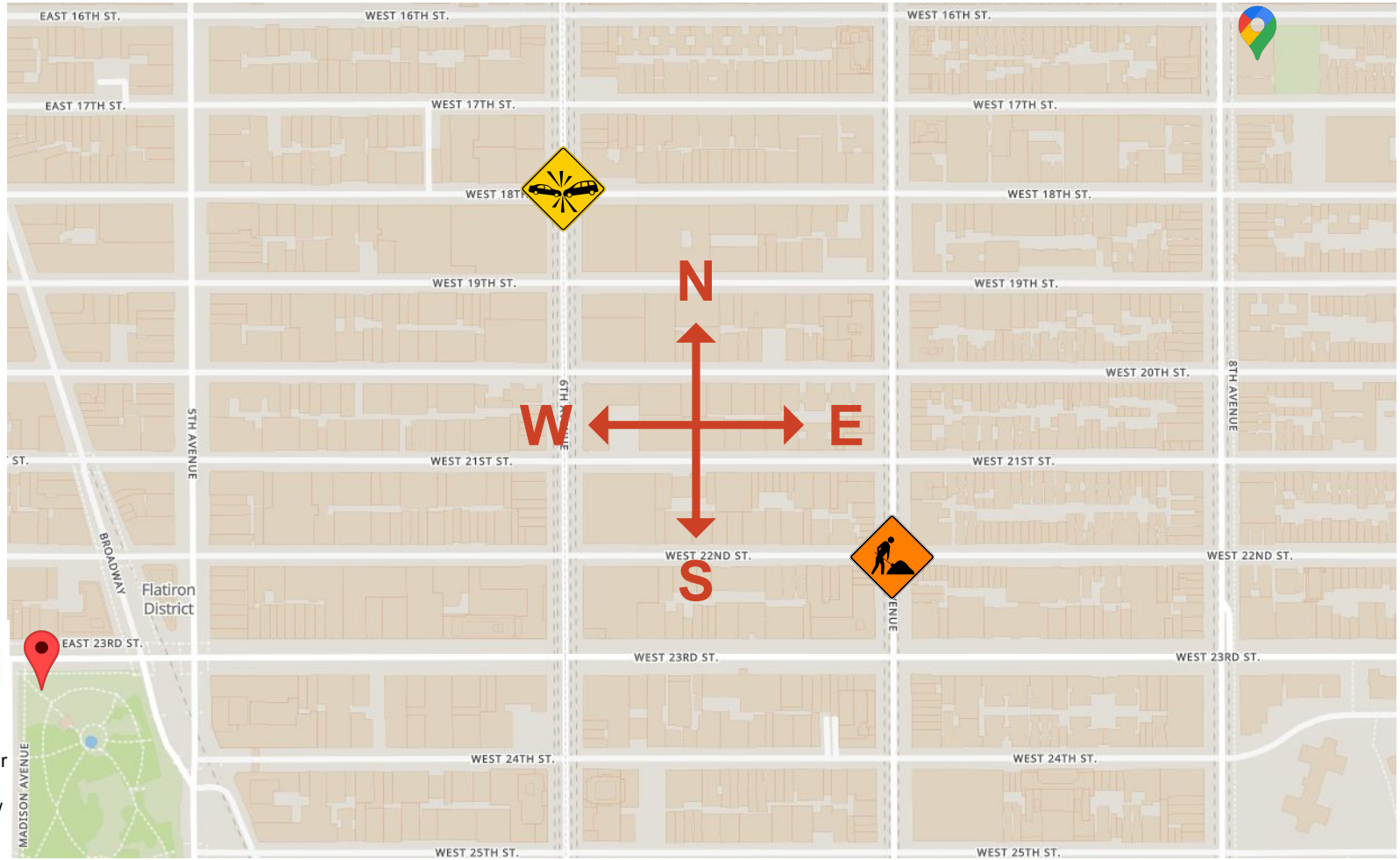

Problem 1
Empire *State* of Delivery

Blackbox

For this question, we shall assume we have SSSP algorithms at our disposal and can use them as *blackboxes*.

Problem

- You are given a manhattan grid
- Some intersections may be closed off due to accidents/roadworks
- The length of a path is measured by the number of right turns it contains
- You are not allowed to make any left turns
- Find the shortest path from source to destination on the grid



Source:

SHAKE SHACK

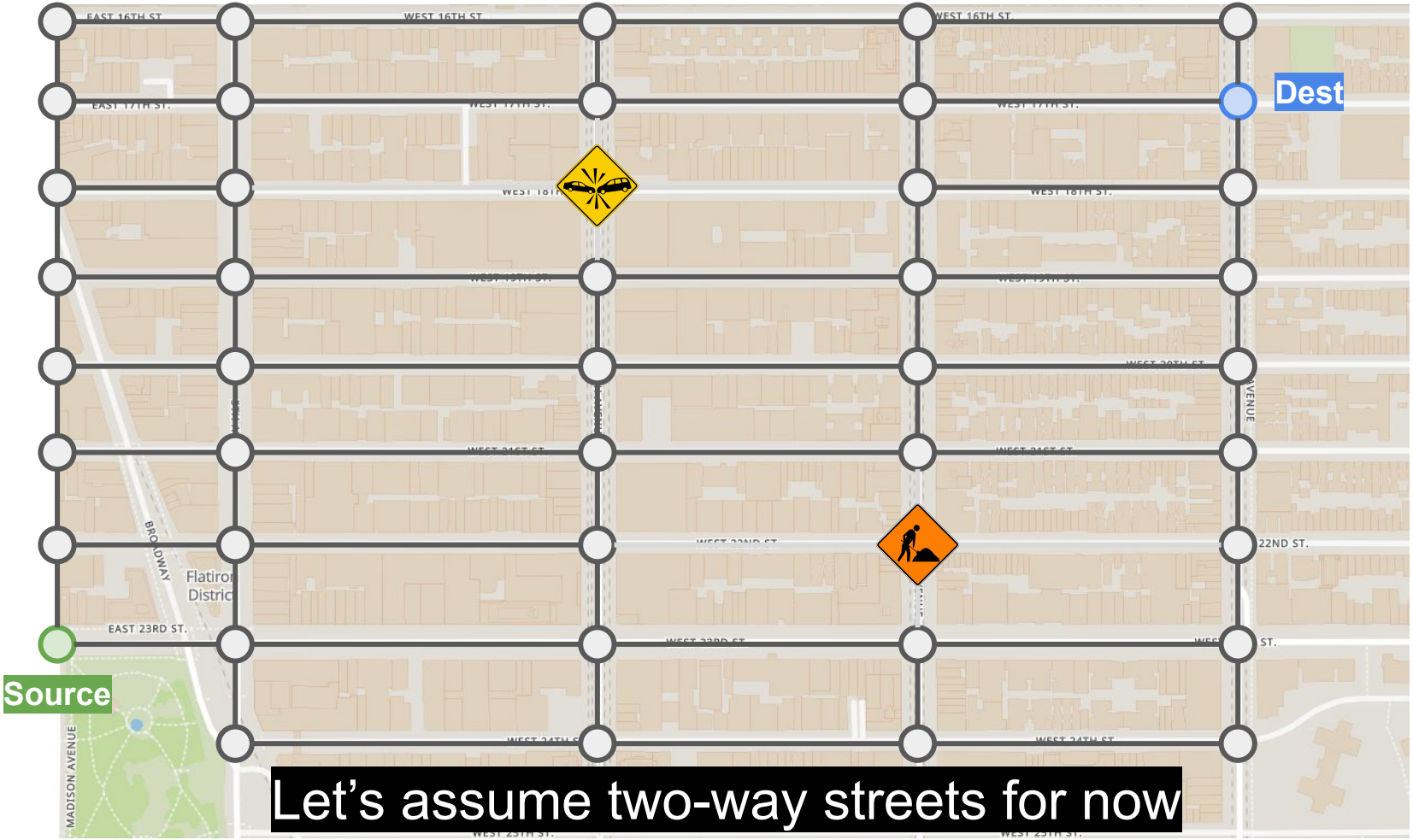
Southeast corner of Madison Square Park (near Madison Avenue &, E 23rd St, New York, NY 10010

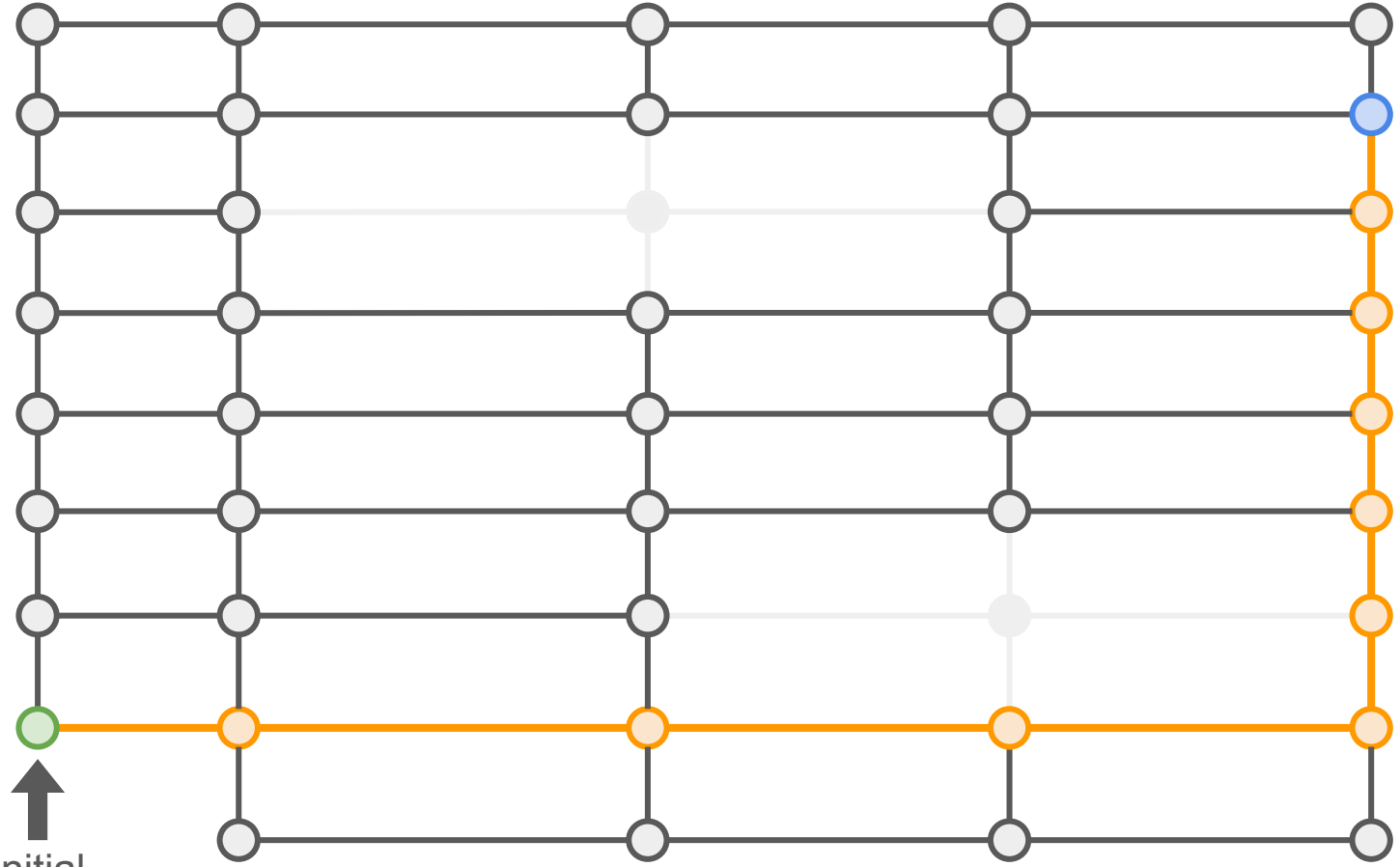
Dest:

Google

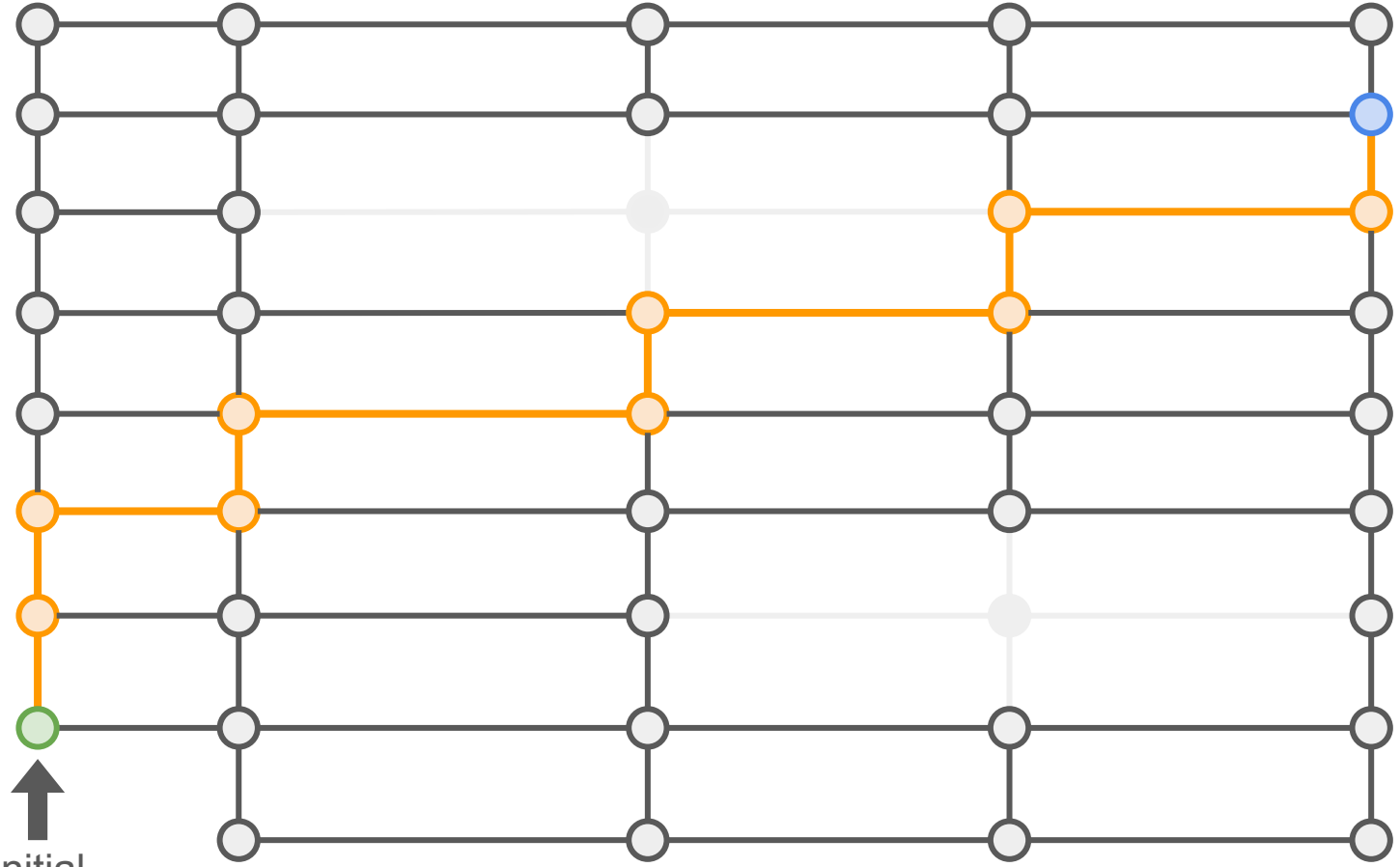
111 8th Ave, New York, NY 10011

Map from: https://www.maptiler.com/

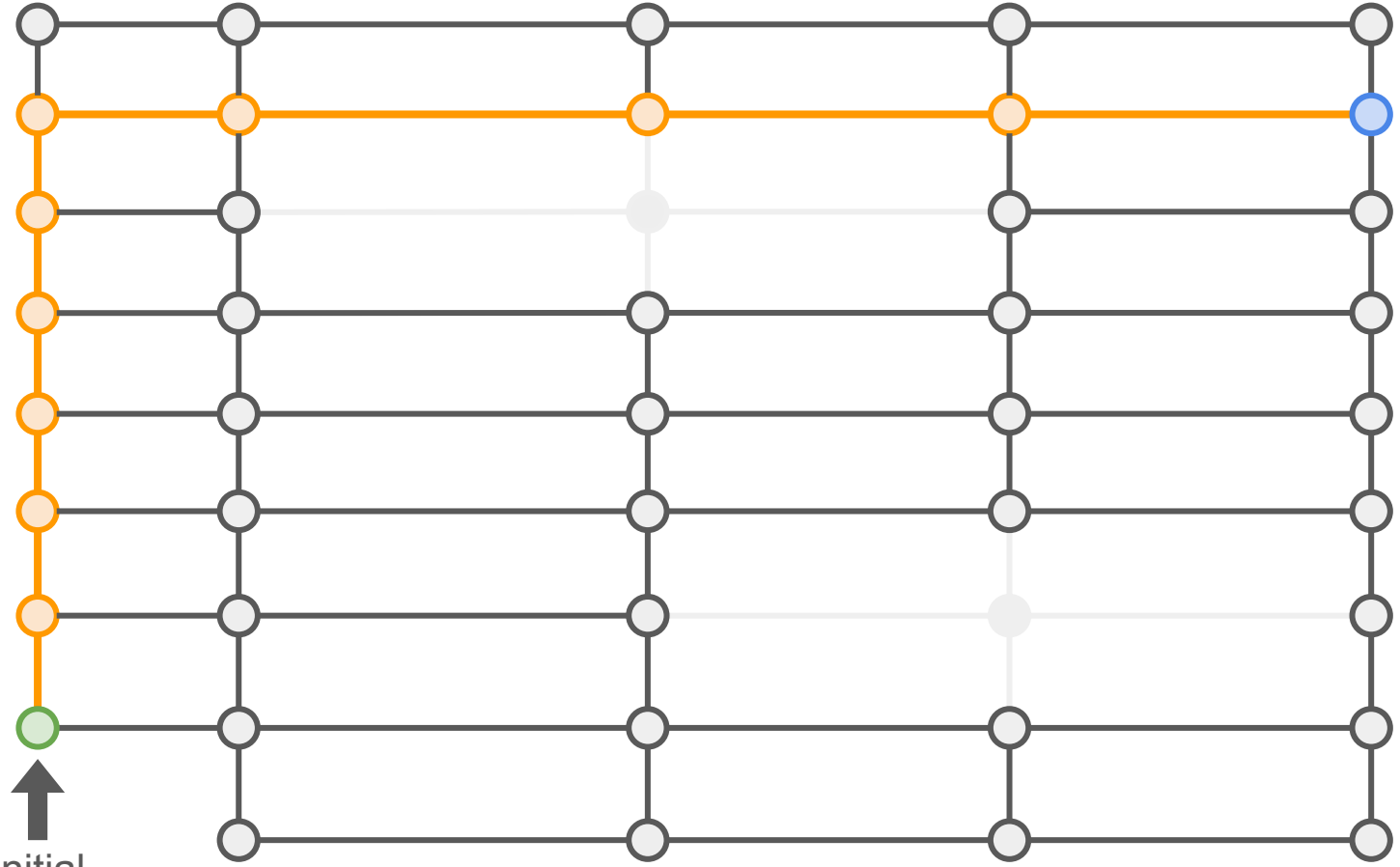

Dest
Source
Let’s assume two-way streets for now



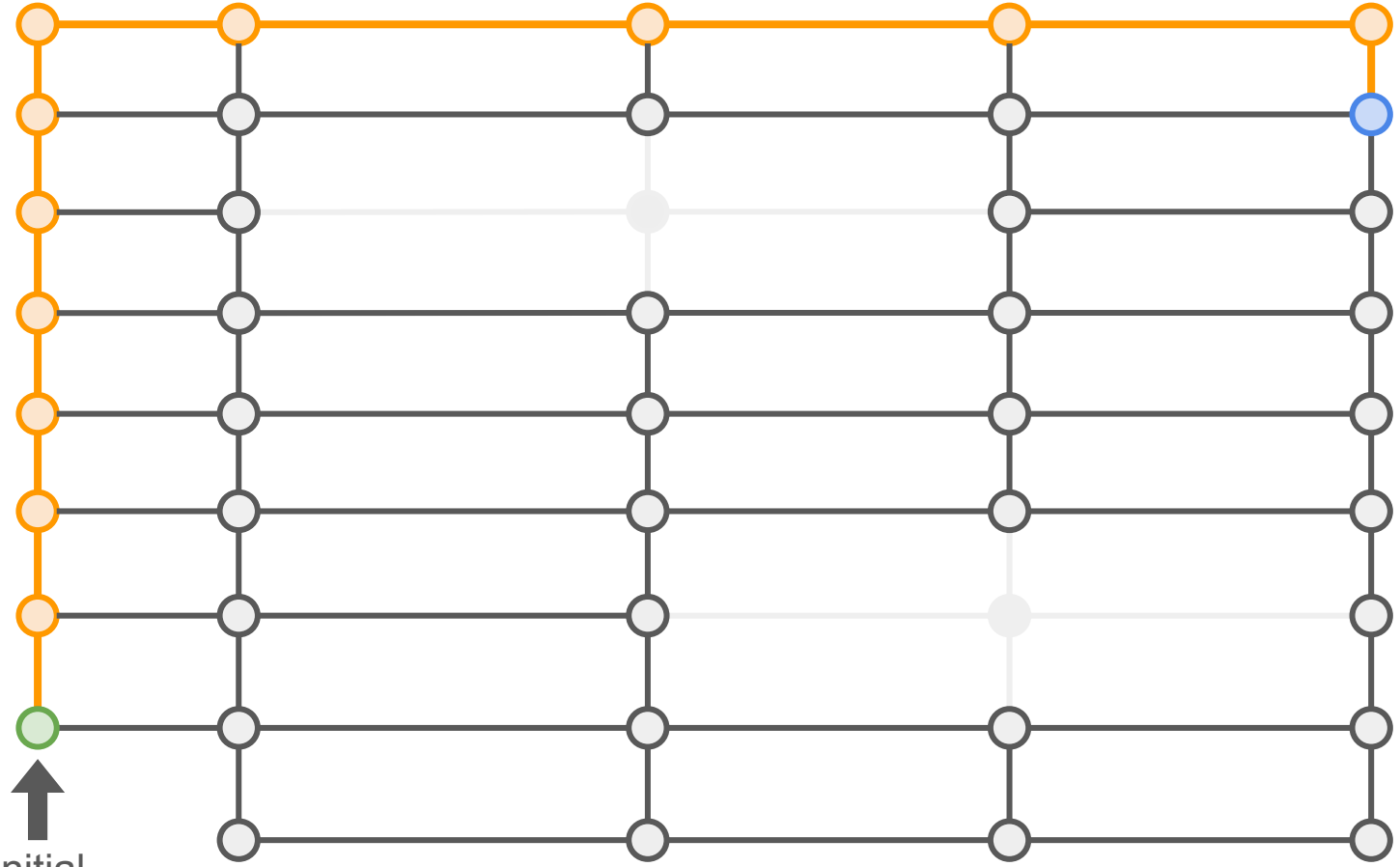
Is this a valid path?



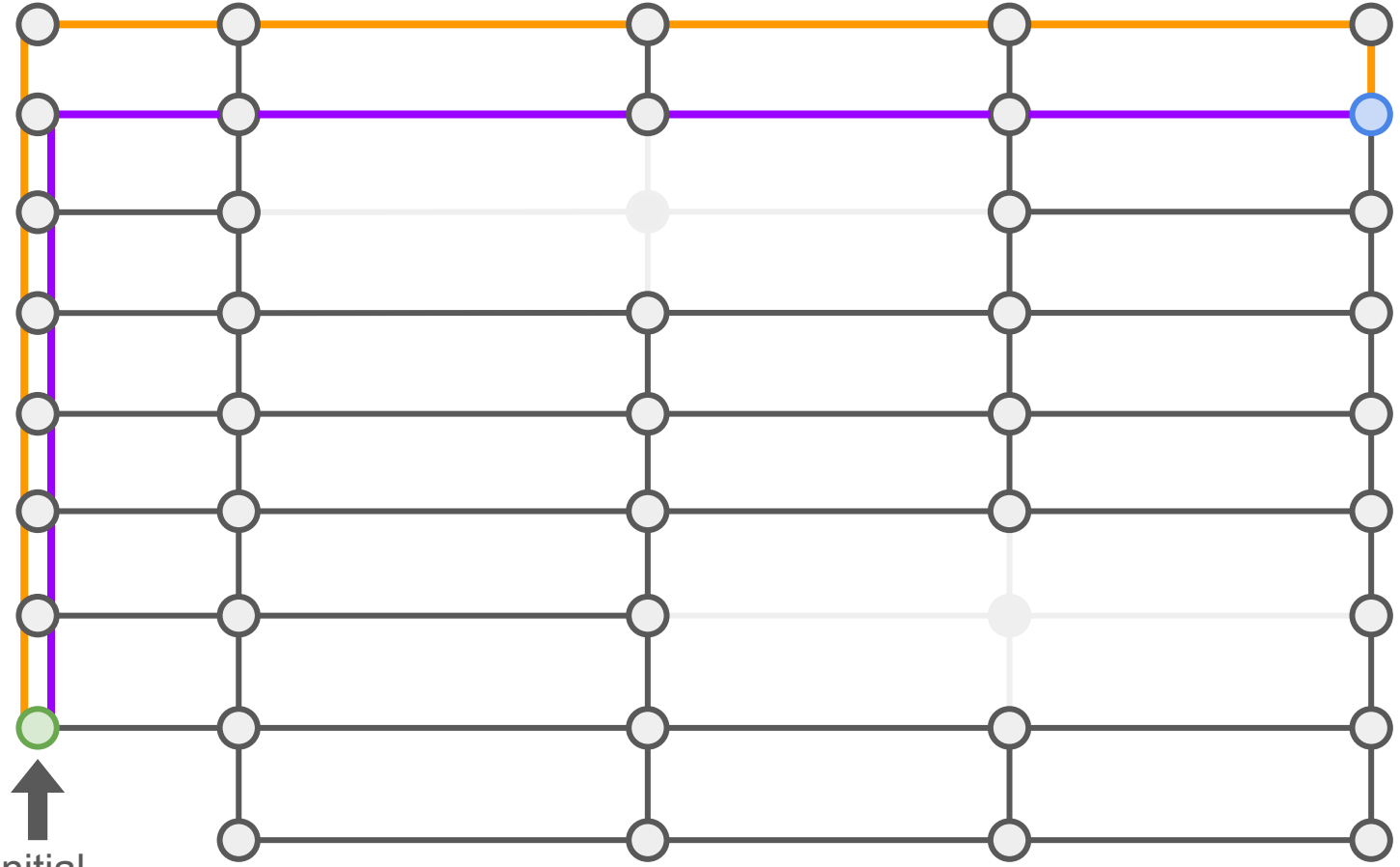
What about this?



What about this?



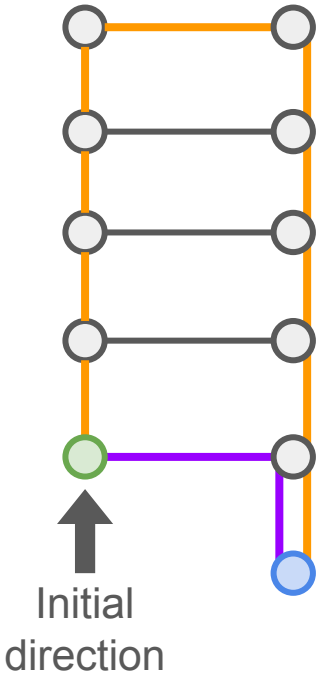
What about this?



So which is a better path?

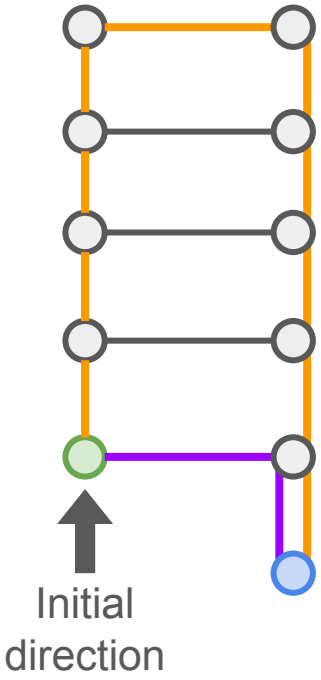
Test yourself!

Which is a better path in this scenario?



Test yourself!

Which is a better path in this scenario?

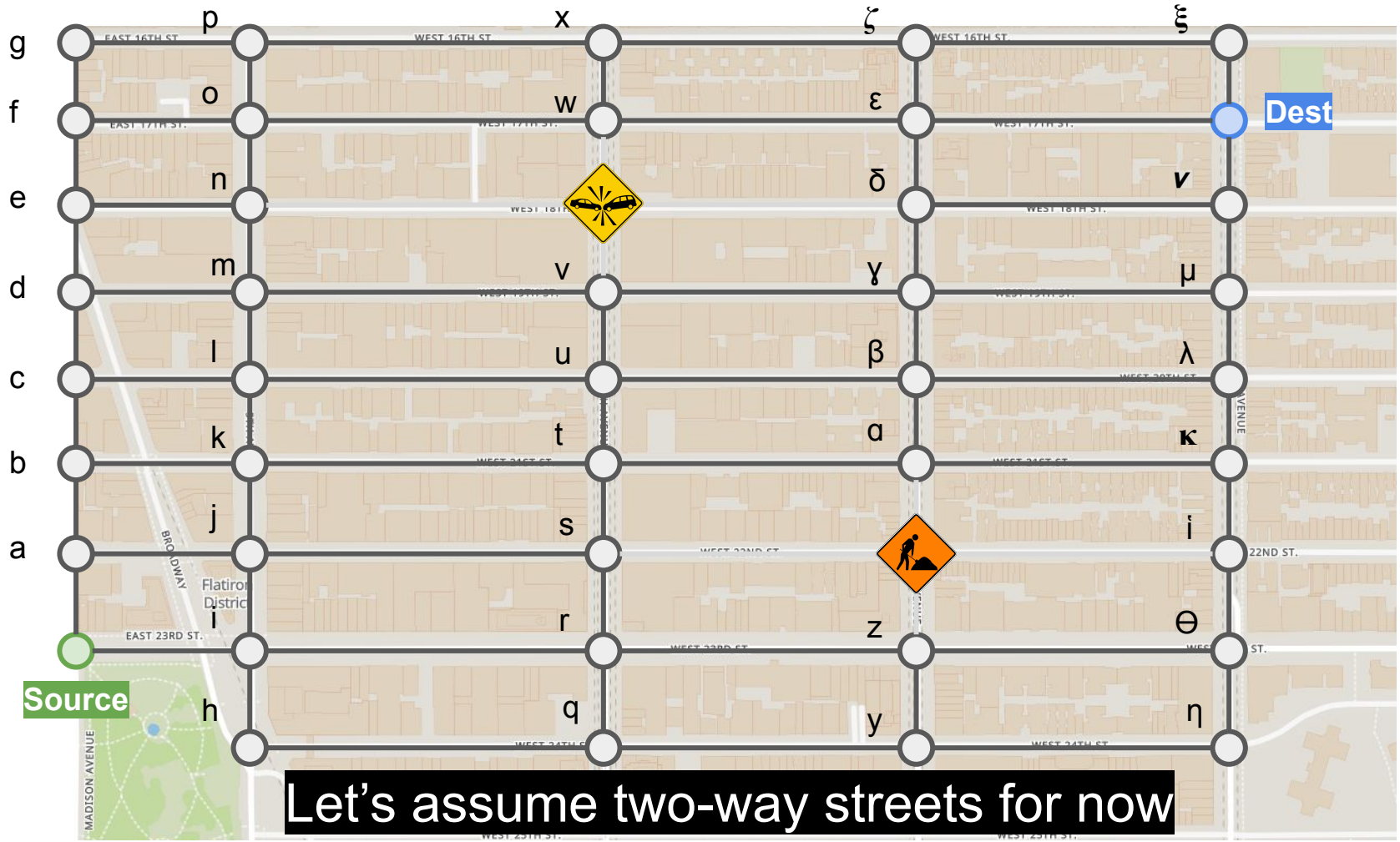


Answer:

Both incur 2 right turns so both are *equally* as good! Recall that we disregard total distance.

Input graph

Is the “raw” input graph good enough?



Let’s assume two-way streets for now

Naive attempt

Something isn't quite right: The edge between vertices are now *conditional* on their states!

This is no good because when we draw an edge between vertices, we expect that edge to *always be there*! Our graph *hardly* helped us at all!

We are clearly not taking advantage of the representational power of graphs here!

Guiding question

What are the problem states in this question?

Guiding question

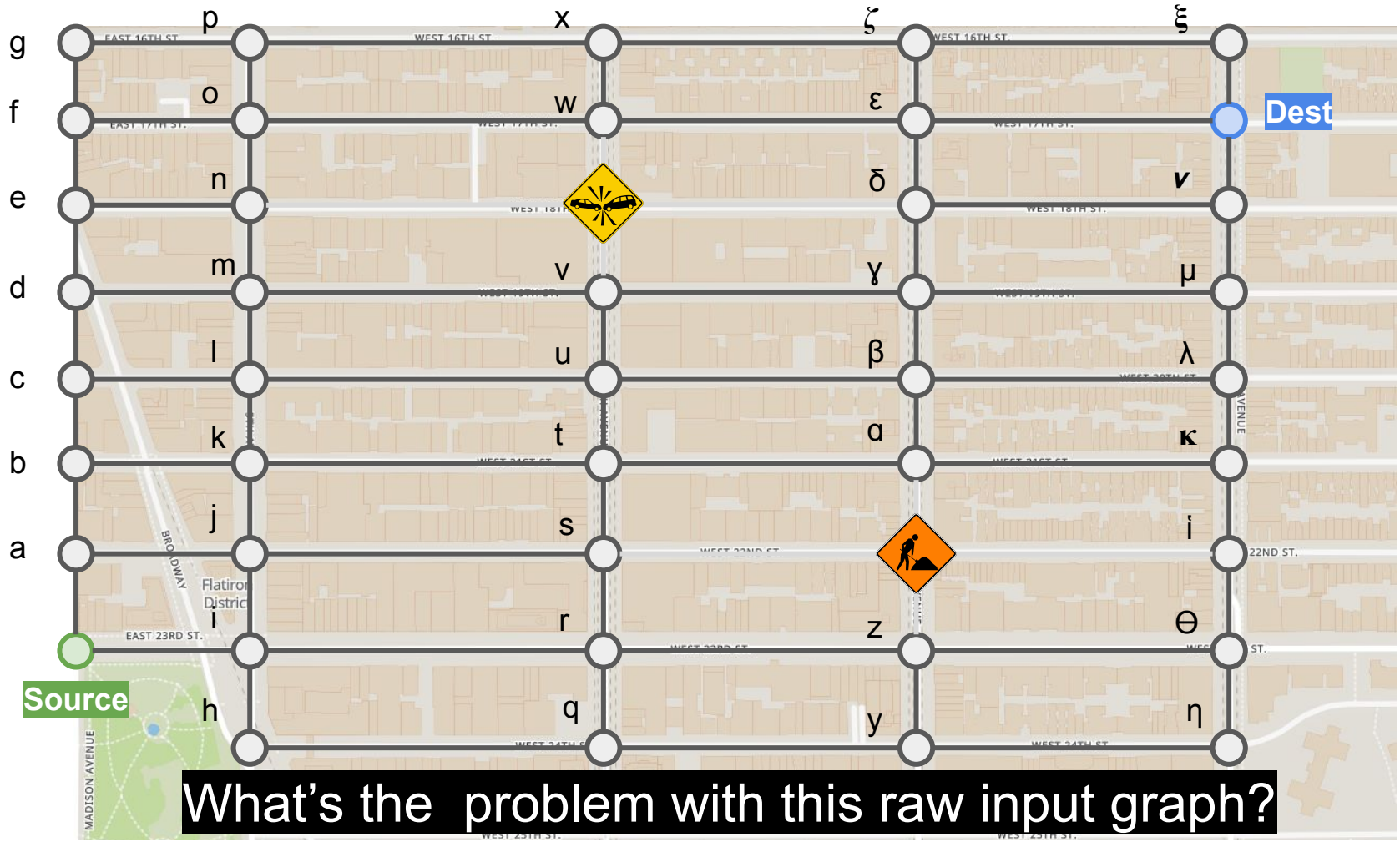
What are the problem states in this question?

Answer:

1. Current location
2. Current direction

Guiding question

What is the problem with the “raw” input graph?



What's the problem with this raw input graph?

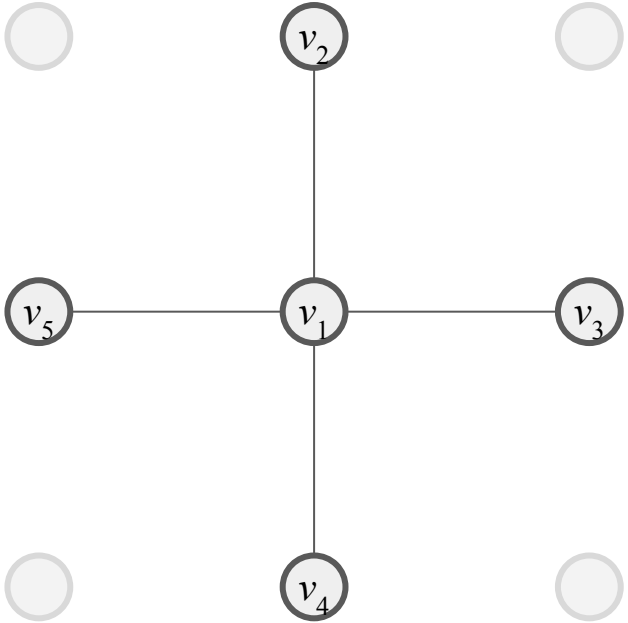
Guiding question

What is the problem with the “raw” input graph?

Answer: It’s able to capture current location but not current direction!

Guiding question

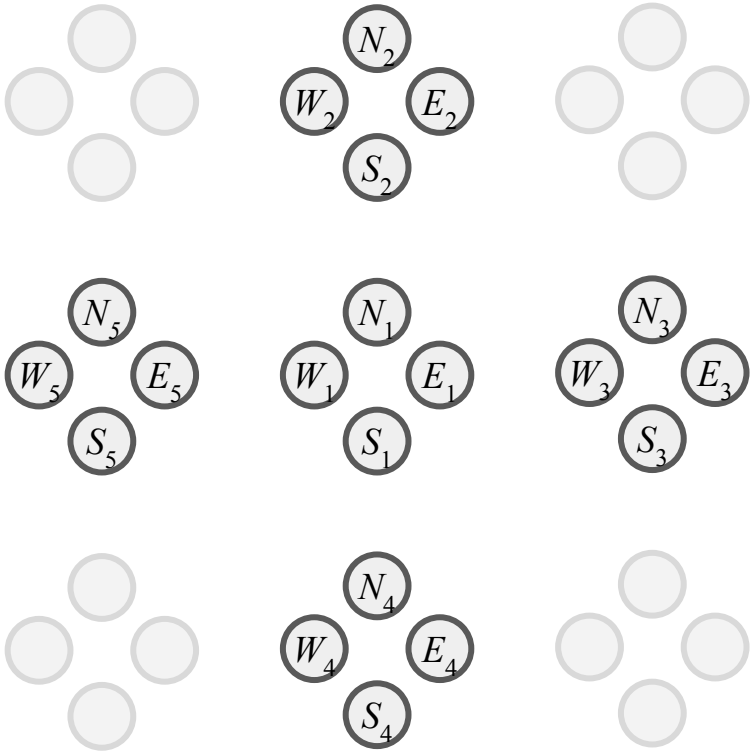
How can we transform the vertices to represent every state fully?



Guiding question

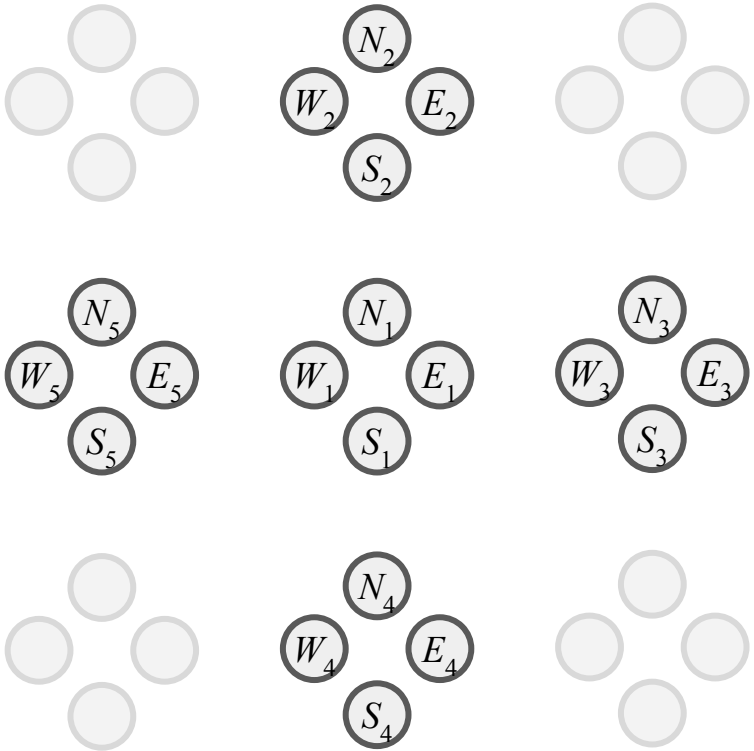
How can we transform the vertices to represent every state fully?

Answer: Use 4 vertices to represent an intersection, where each captures one of the 4 directions of the vehicle when it reaches that intersection.



Guiding question

How can we capture all possible state transitions using edges?

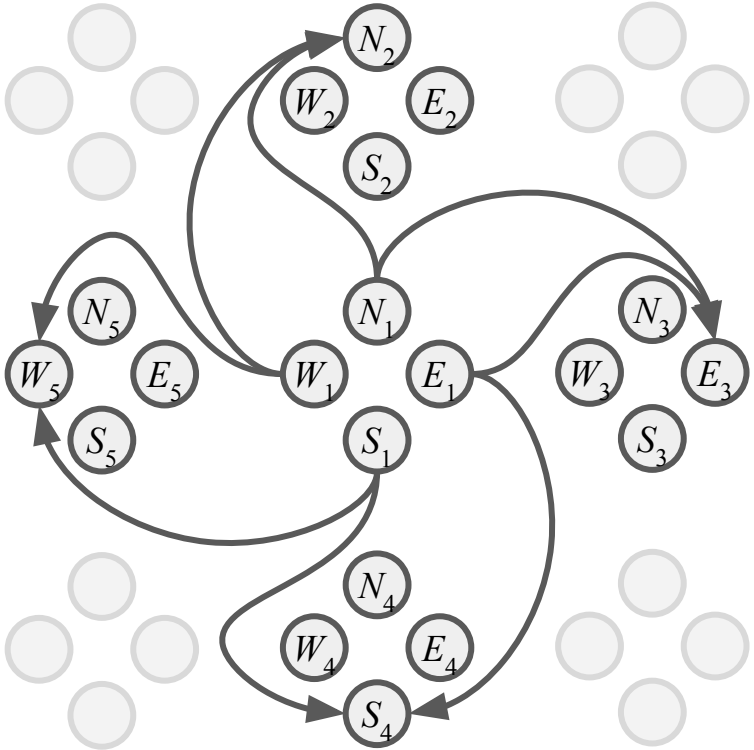


Guiding question

How can we capture all possible state transitions using edges?

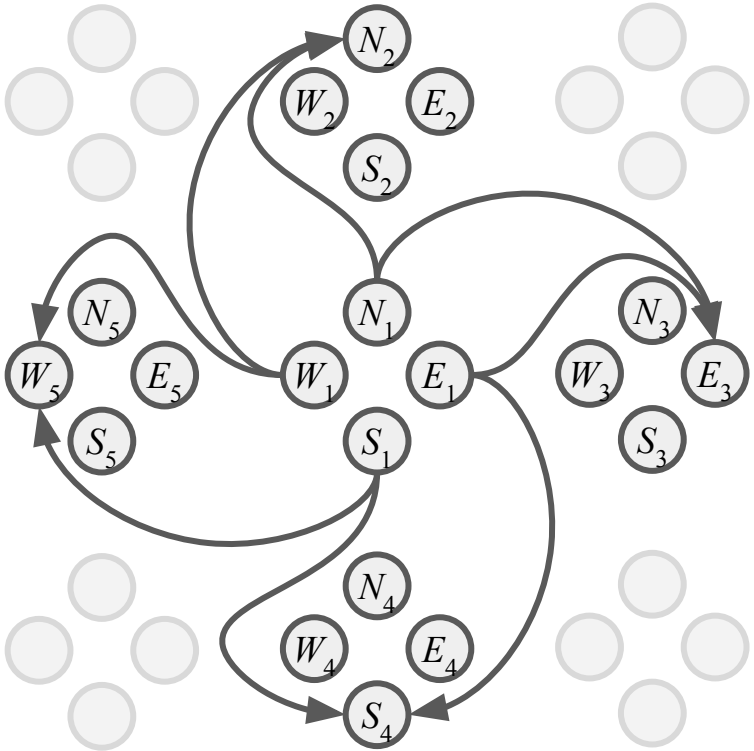
Answer:

Connect each vertex with outgoing edges to the *only two* valid states in the next step: going straight and turning right.



Guiding question

What should the edge weights be?

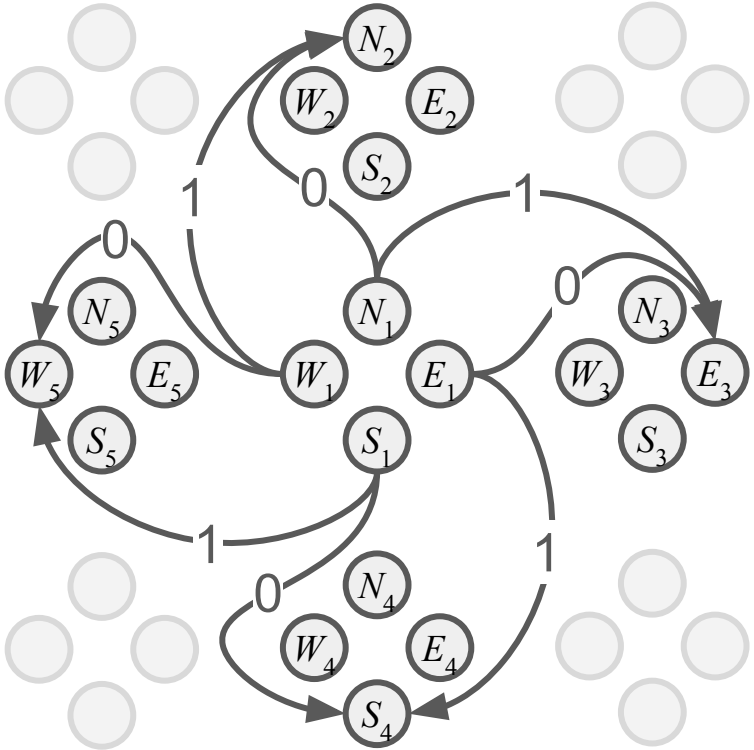


Guiding question

What should the edge weights be?

Answer:

0 for going straight and 1 for turning right.



Solution

After constructing such a graph that captures *all permitted state transitions* between intersections,

we can just run SSSP on the source node and everything will work out.

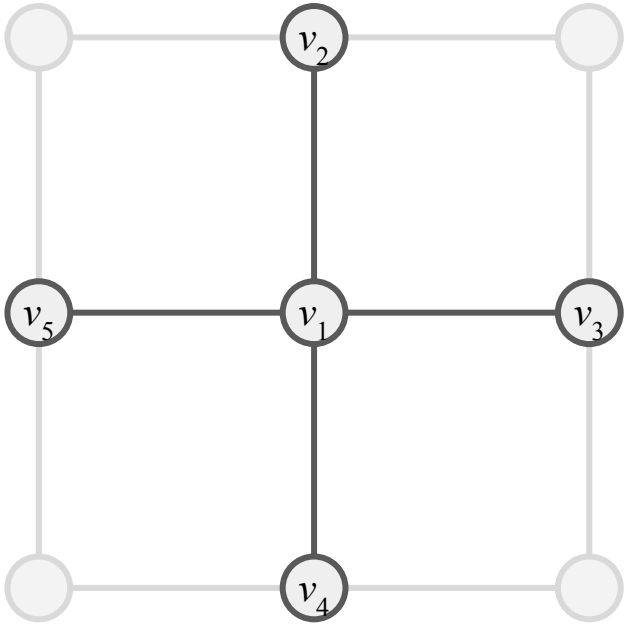
Moral of the story

To exploit the power of graphical representations,

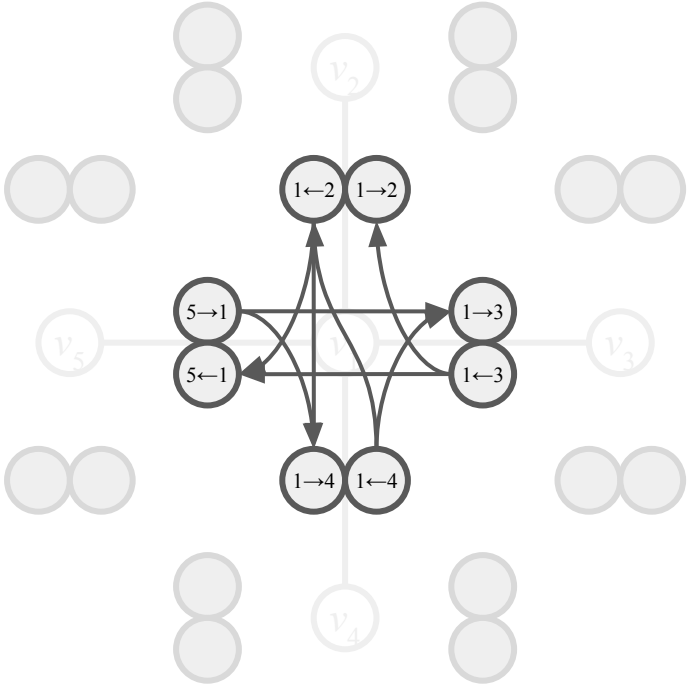
we should use vertices to fully capture **all problem states** and edges to fully capture **all possible state-transitions** in the problem.

Interesting solutions

Modelling roads as vertices



One vertex for each lane along the road



Modelling roads as vertices

The trouble now with dealing with location is that intersections are between two vertices. We might have to insert dummy node(s) (if intersection or destination is along the perimeter of the grid).

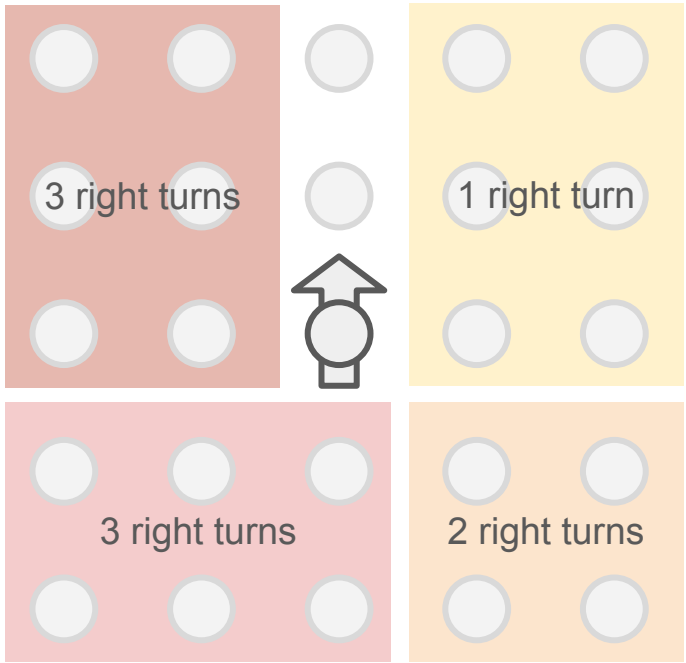
Furthermore we now need to pick the minimum out of the shortest paths to all 8 vertices surrounding an intersection (since we can reach an intersection in that many directions).

This modelling choice is an interesting proposal nevertheless! It’s just a wee bit unwieldy in the context of this problem.

Non-graph solution

If we assume the map is a perfect grid and all intersections are **always accessible** then the number of right turns simply depend on which quadrant the destination lie on :)

Realize each intersection just need to be given a coordinate to determine which quadrant they lie in w.r.t. the starting point/direction.



Problem 2: EuroTrip 2077

Problem description

- Manon has a map of roads connecting European cities
- Each road is either “odd” or “even” parity
- A road can only be used on the days of the month with the same parity, except December where this rule doesn’t apply
- Assumptions:
  - Even/odd days always alternate as date progresses
  - Each road takes 1 day to travel
  - There is at least one one valid route to the destination
- Goal: Get Manon from Paris to Moscow in the shortest time

Input graph

Manon

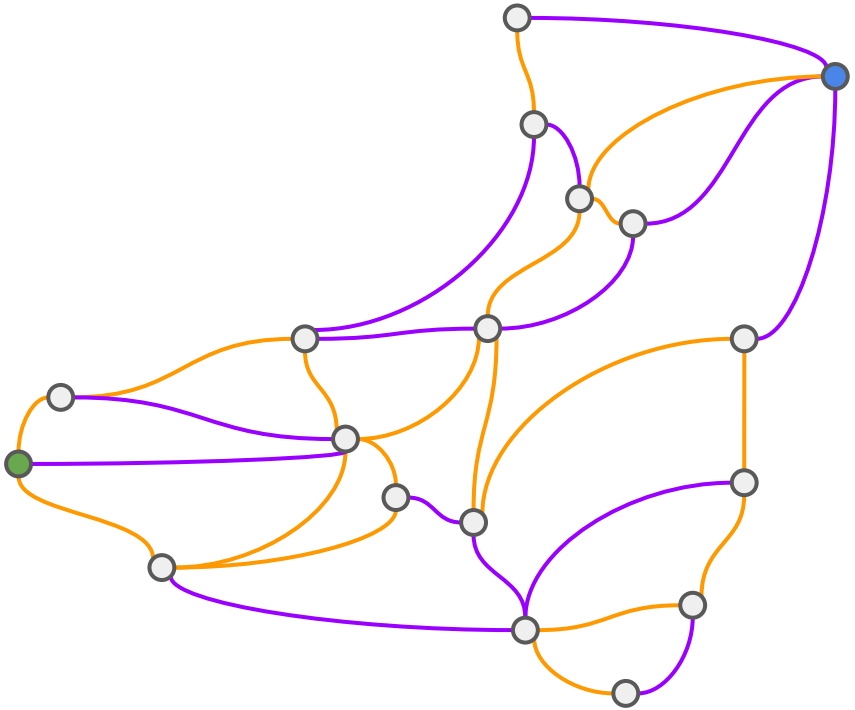
Input graph


Tallinn
Moscow
Riga
Vilnius
Minsk
Berlin
Warsaw
Kyiv
Brussels
Prague
Paris
Chișinău
Vienna
Budapest
Bern
Bucharest
Belgrade
Sofia

Input graph

Even

Odd



Problem 2.a.

Suppose Manon's trip is planned for the month of December when the odd-even scheme is not in effect (i.e., you can use all the roads every day).

With the goal of finding a driving route from Paris to Moscow that is the *fastest*, model this as a graph problem and solve it using a single *single* graph algorithm you have learnt in class so far.

Why does the algorithm work in this problem?

Guiding question

How is the notion of *fastest* captured in the graph here?

Guiding question

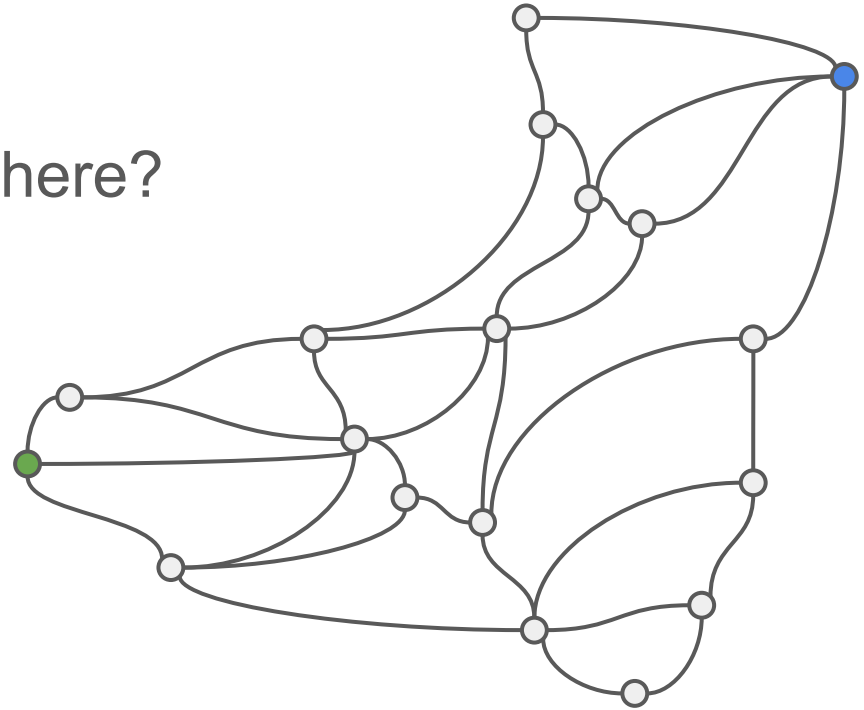
How is the notion of *fastest* captured in the graph here?

Answer: The path with the least number of edges.

Solution

Just run BFS!

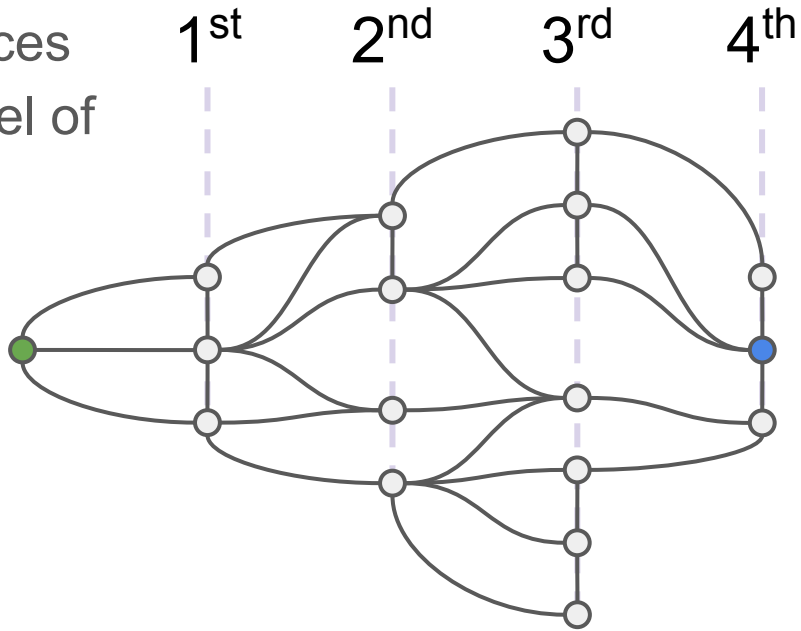
Why does it work here?



Why BFS works

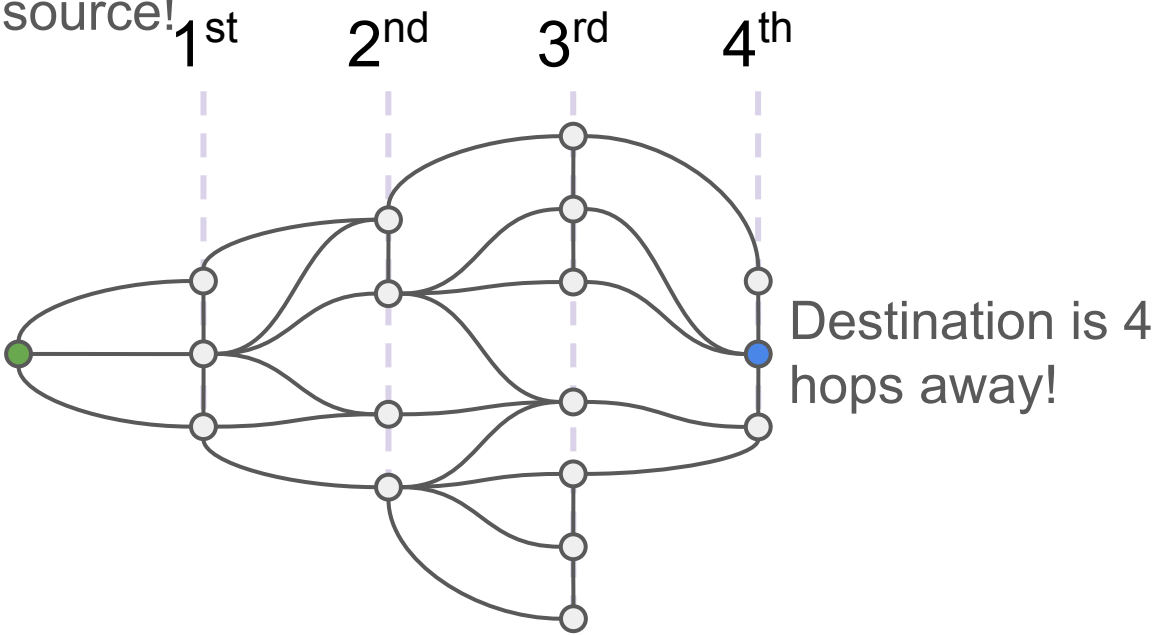
Recall that BFS conducts *level-order* traversal.

Rearranging the vertices according to each level of BFS traversal:



Why BFS works

For a node, its level of visitation during BFS is its *minimum* number of "hops" away from the source!



Test yourself!

How do you recover the path?

Test yourself!

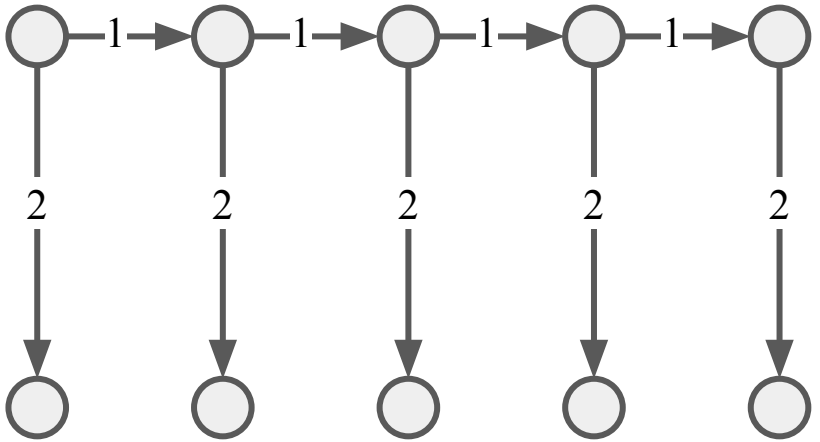
How do you recover the path?

Answer: When a vertex is being visited, store the *previous* vertex from whence it came.

Test yourself!

What if some roads take 1 day to traverse while others takes 2?

Consider the directed graph below with only edge weights 1 and 2, how can we transform the graph such that we can just run BFS on it to obtain the SSSP?



Test yourself!

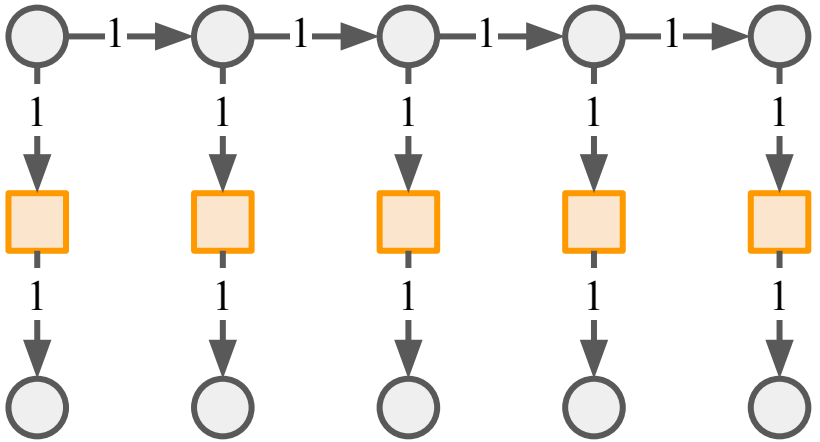
What if some roads take 1 day to traverse while others takes 2?

Consider the directed graph below with only edge weights 1 and 2, how can we transform the graph such that we can just run BFS on it to obtain the SSSP?

Answer:

Insert a dummy vertex between edges of weight 2 so as to split it into 2 edges of weight 1 each!

This of course incurs more space..



Problem 2.b.

Suppose now Manon will have to take into account the even-odd scheme because her planned trip will *not* overlap with any days of December. Suppose also that she will have to leave from Paris on an *even* day and she will *not* be spending overnight in any city.

By constructing a new graph, show how to find a driving route from Paris to Moscow that is the *fastest*, using a *single* graph algorithm you have learnt in class so far.

Guiding question

What are the problem states that the graph should capture?

Guiding question

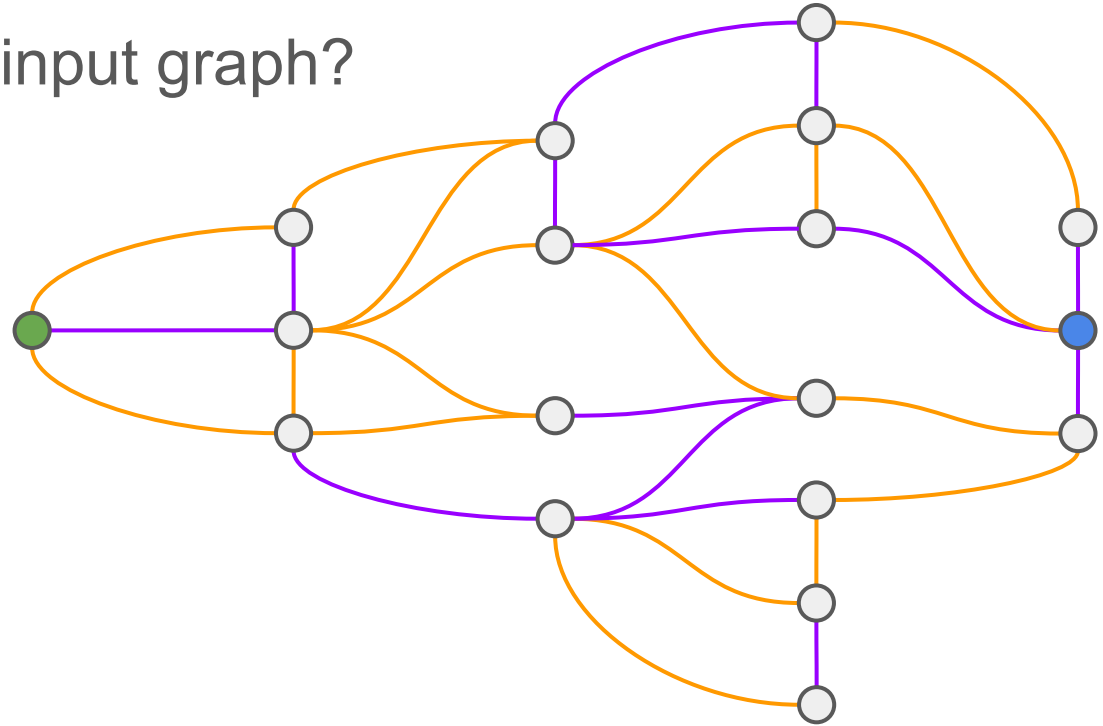
What are the problem states that the graph should capture?

Answer:

- Current location (city)
- Current day parity (odd/even)

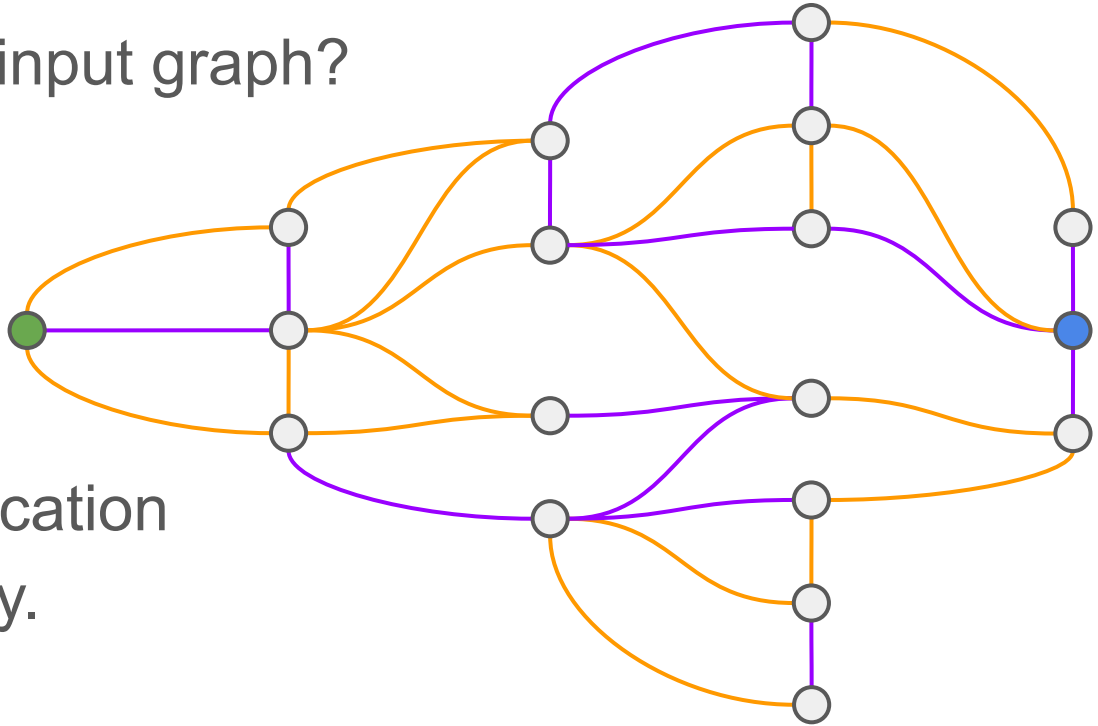
Guiding question

Should we use this raw input graph?



Guiding question

Should we use this raw input graph?

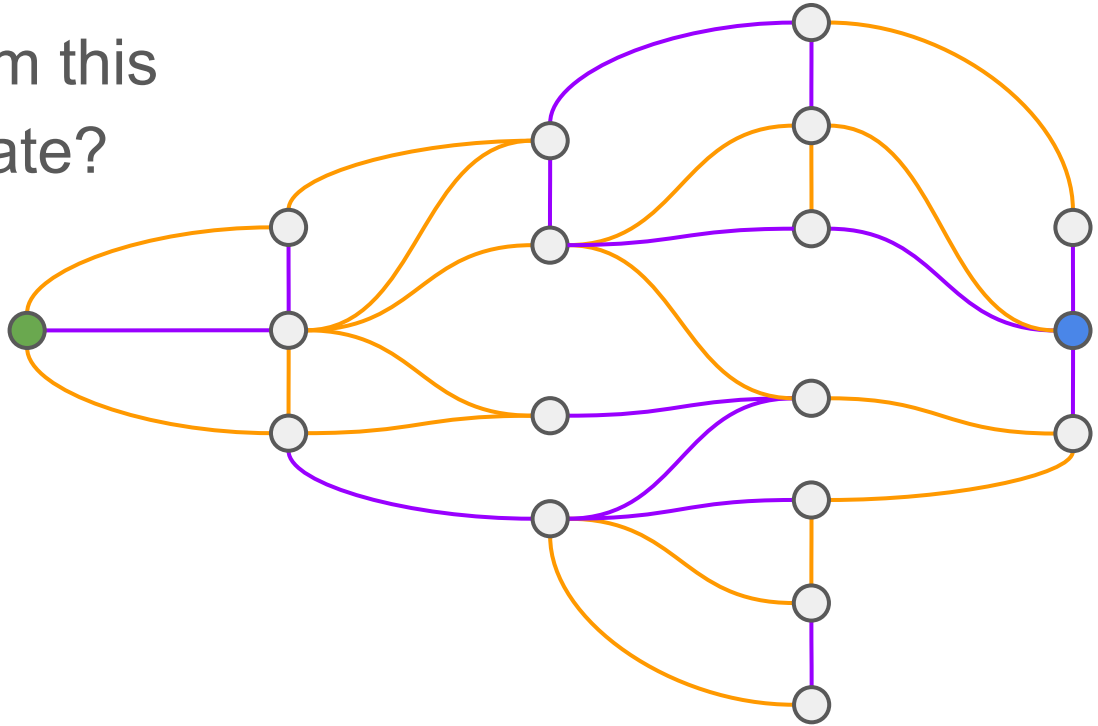


Answer:

No it captures current location but not current day parity.

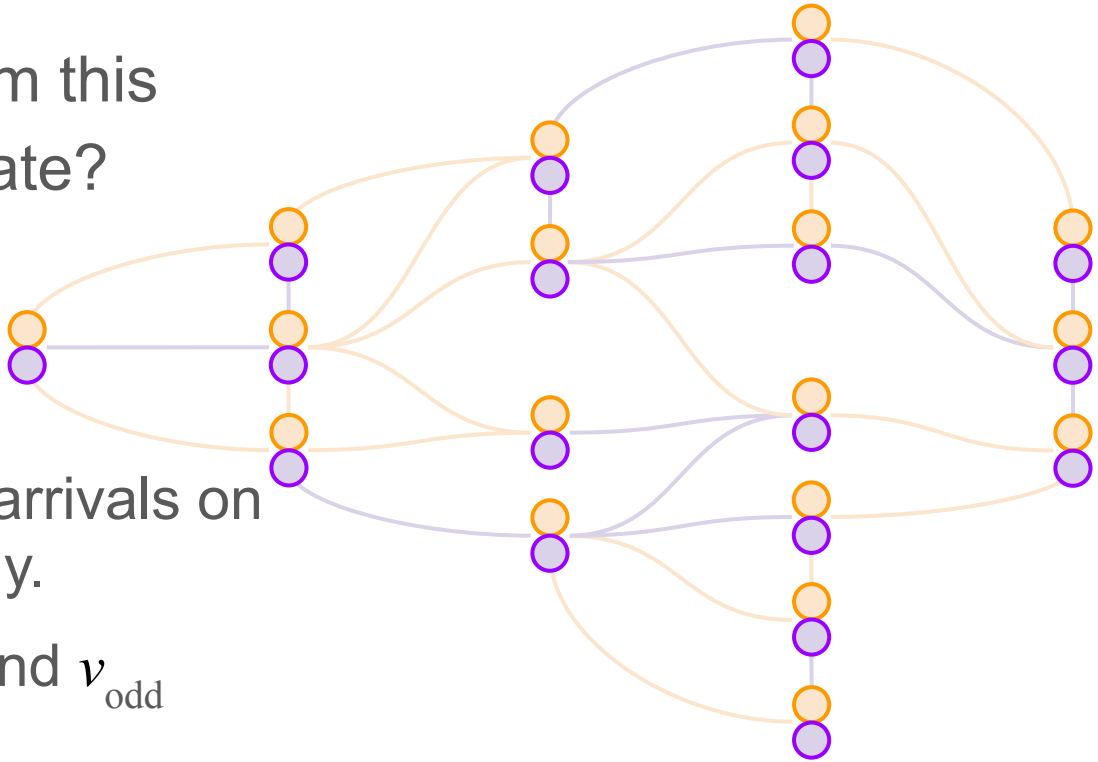
Guiding question

How should we transform this graph to fully capture state?



Guiding question

How should we transform this graph to fully capture state?



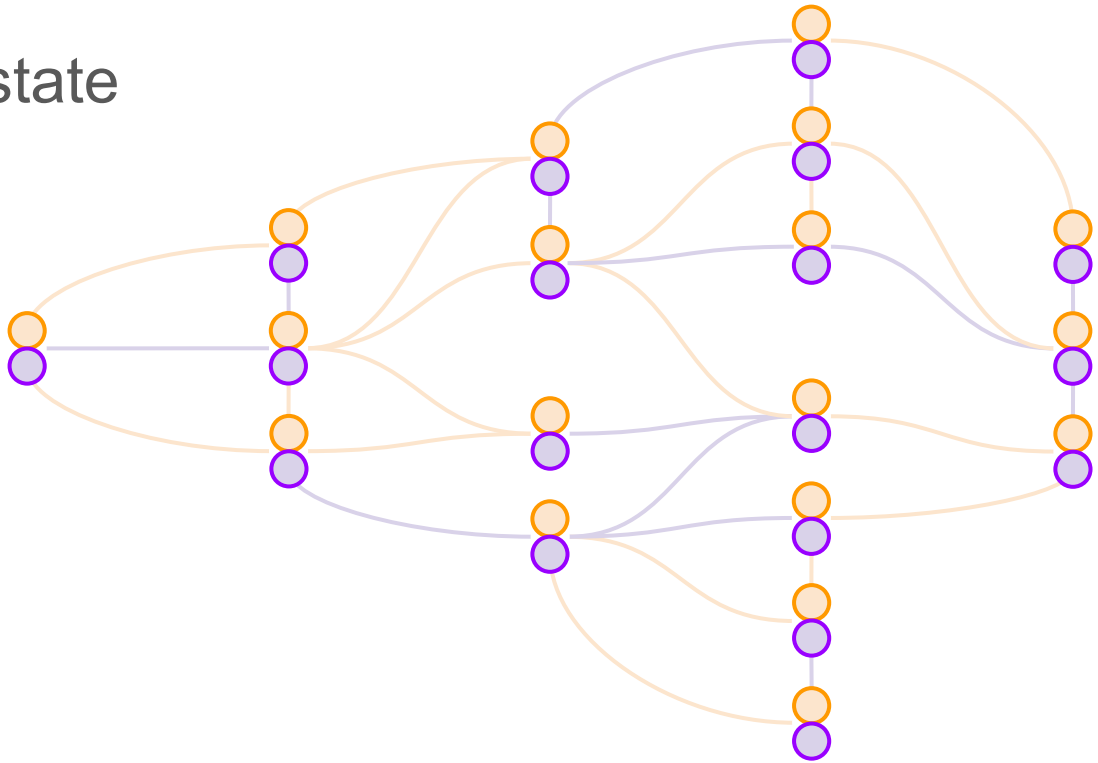
Answer:

Split each city vertex into arrivals on odd/even days respectively.

E.g. split vertex v in v_{even} and v_{odd}

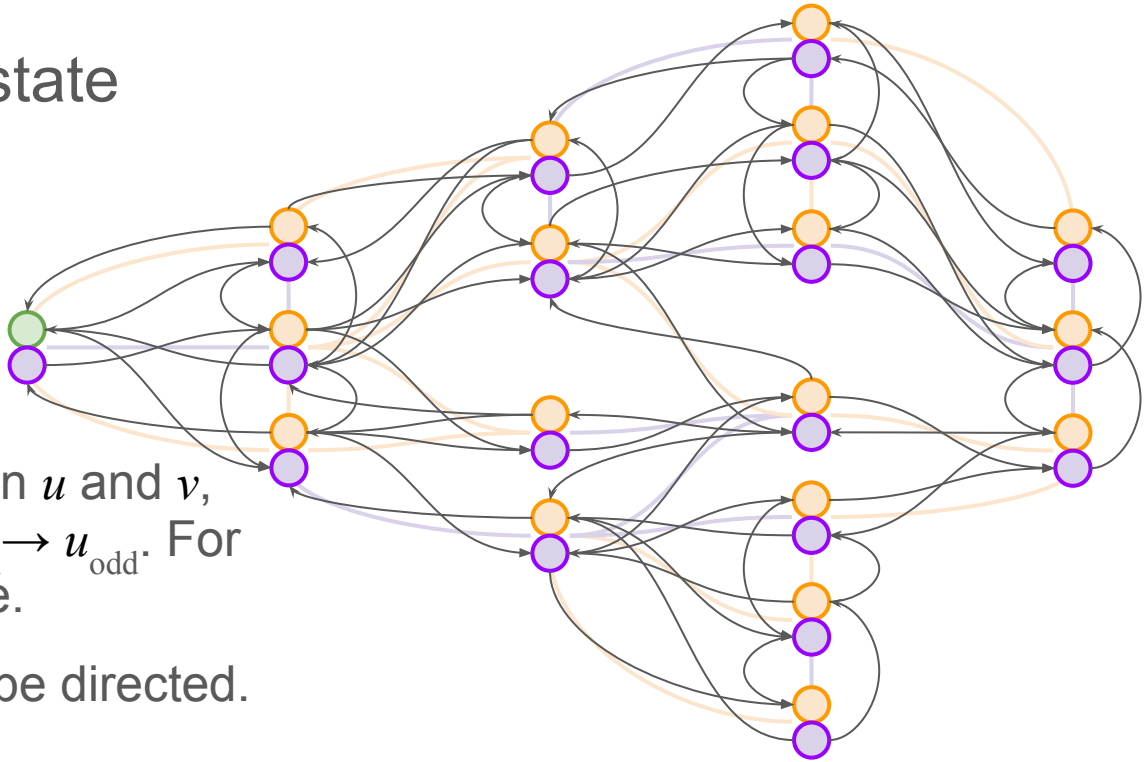
Guiding question

How should we handle state transitions now?



Guiding question

How should we handle state transitions now?



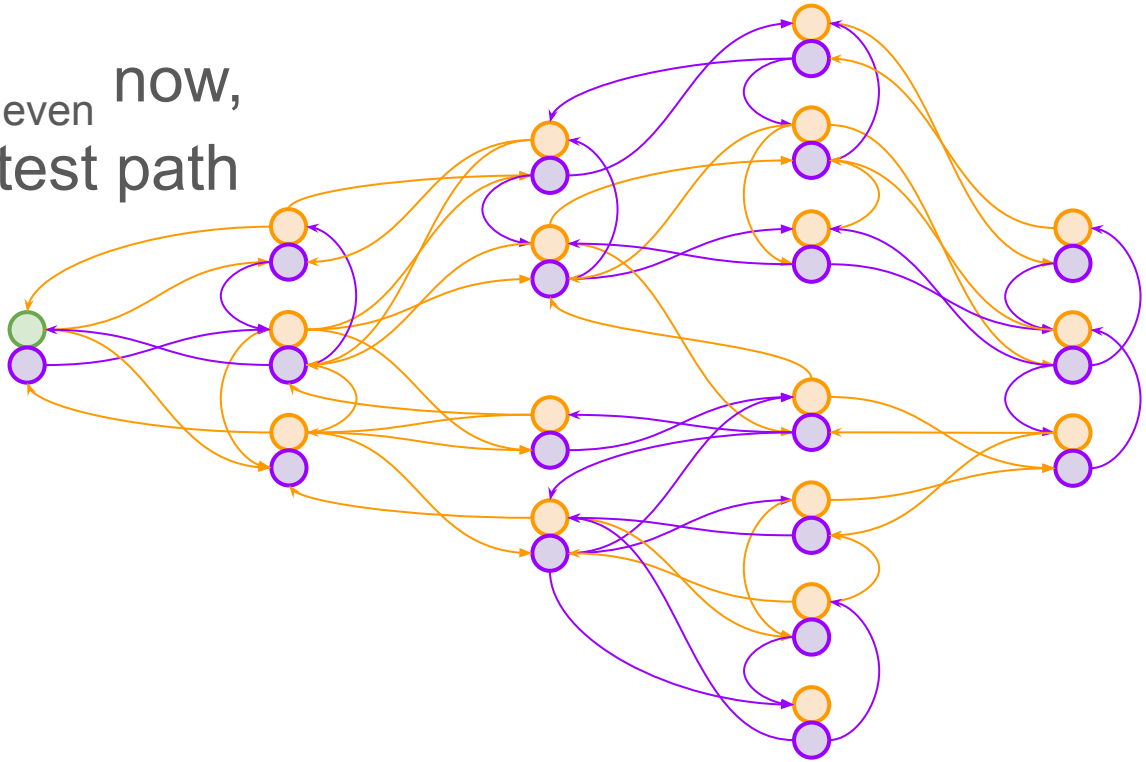
Answer:

For *even* roads serving between u and v, we connect $u_{\text{even}} \rightarrow v_{\text{odd}}$ and $v_{\text{even}} \rightarrow u_{\text{odd}}$. For odd roads, the opposite is done.

Note that the edges must now be directed.

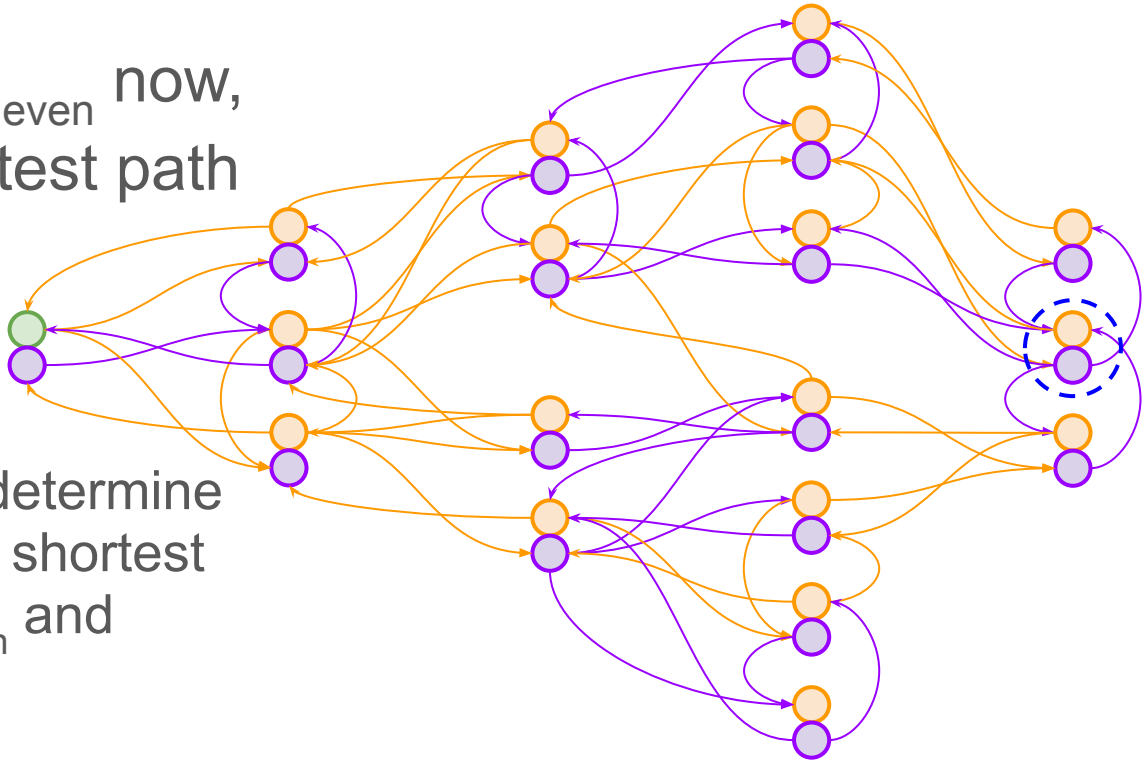
Guiding question

Running BFS from $\text{paris}_{\text{even}}$ now, how do we find the shortest path to Moscow?



Guiding question

Running BFS from $paris_{even}$ now, how do we find the shortest path to Moscow?



Answer:

One final step is needed to determine the shorter one between the shortest paths to vertices $Moscow_{even}$ and $Moscow_{odd}$.

Problem 2.c.

What if Manon has a choice of whether to leave on an odd or an even day and also whether or not to stay overnight at a city?

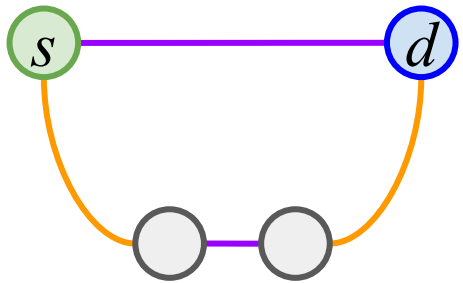
Guiding question

Is it possible that a shorter path can be found if Manon decides to stay overnight at a city?

Guiding question

Is it possible that a shorter path can be found if Manon decides to stay overnight at a city?

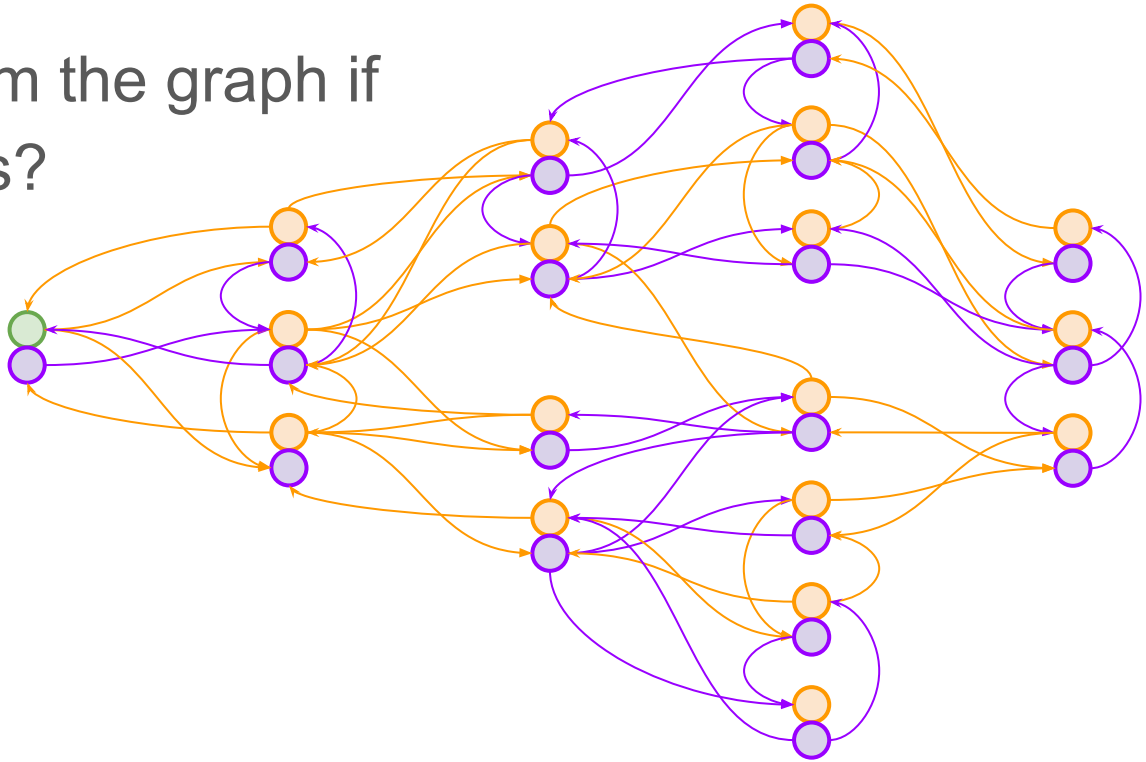
Answer: Yes, consider the scenario on the right where we leave s on an even day and destination is d.



If we stay overnight at s, we can reach d in 2 days, otherwise we would need 3 days.

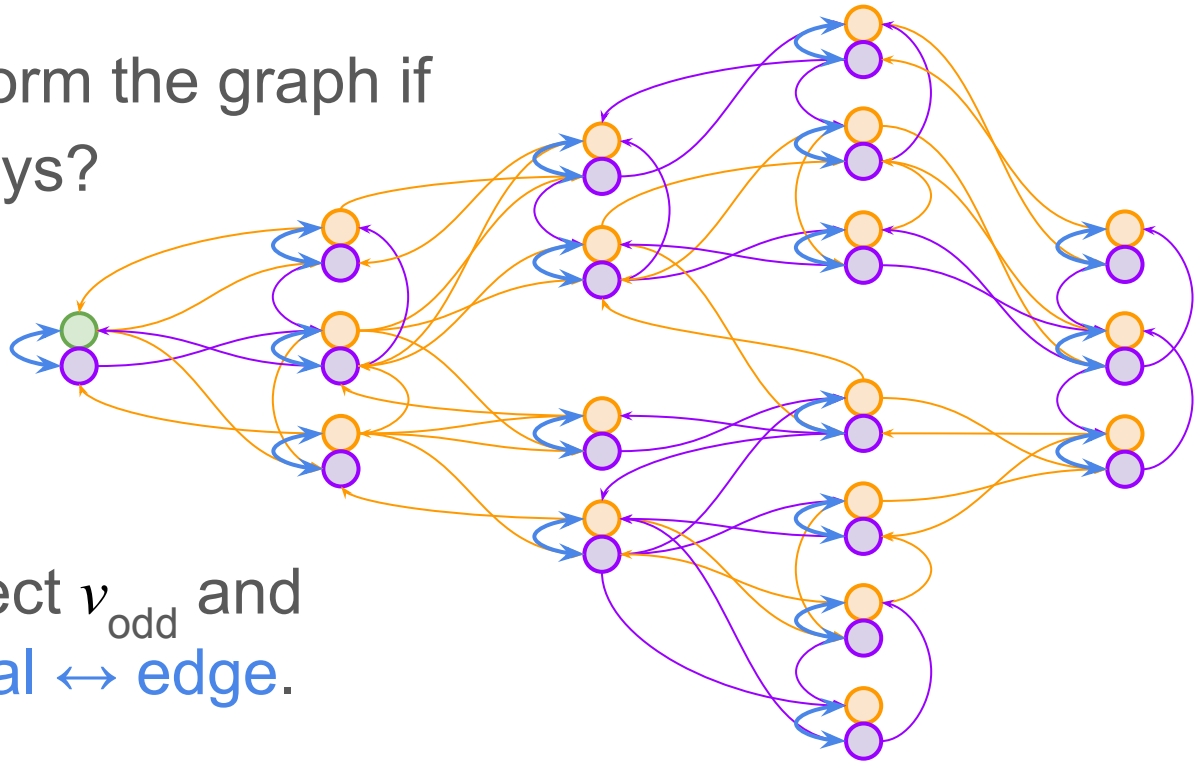
Guiding question

How should we transform the graph if we allow overnight stays?



Guiding question

How should we transform the graph if we allow overnight stays?



Answer:

For every city v, connect v_{odd} and v_{even} with a bidirectional $\leftrightarrow$ edge.

Guiding question

Without running BFS twice (once from $\text{Paris}_{\text{even}}$ and once from $\text{Paris}_{\text{odd}}$), how can we determine in a single BFS which is the better day to leave?

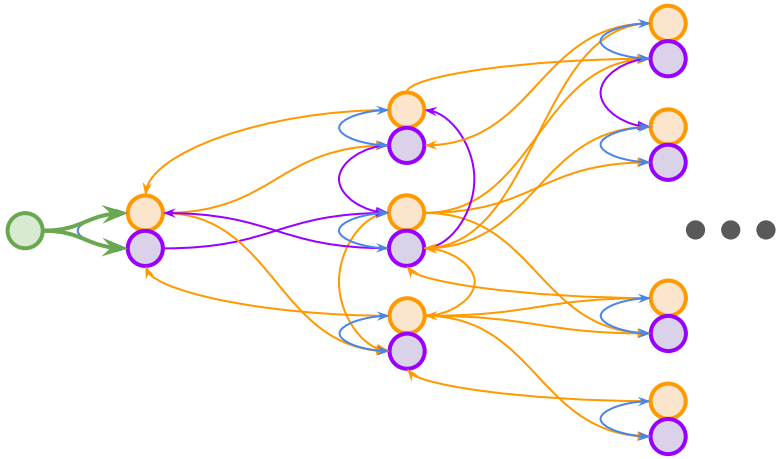
Guiding question

Without running BFS twice (once from $\text{Paris}_{\text{even}}$ and once from $\text{Paris}_{\text{odd}}$), how can we determine in a single BFS which is the better day to leave?

Answer:

Connect $\text{Paris}_{\text{even}}$ and $\text{Paris}_{\text{odd}}$ to a *dummy source* vertex from which we will run BFS.

Deduct off the dummy edge in the final step after backtracking path.



Test yourself!

Is the solution output by your algorithm unique? I.e. Is it possible for other paths to exist that are just as fast?

Test yourself!

Is the solution output by your algorithm unique? I.e. Is it possible for other paths to exist that are just as fast?

Answer: The solution is *not* unique!