CS2040S

Recitation 8

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!

Blackbox

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

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

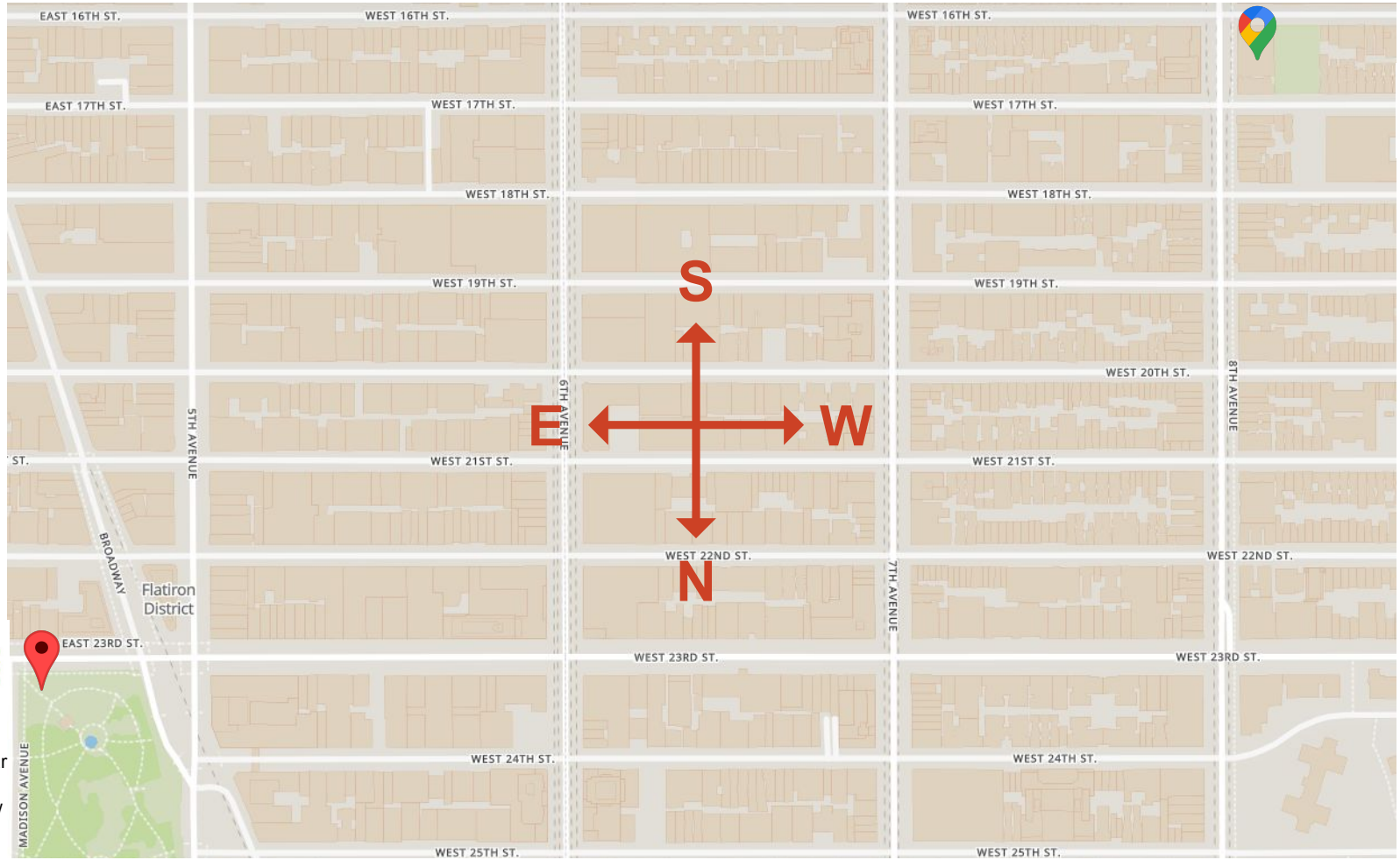
Goals

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


Problem 1
Empire *State* of Delivery

Problem

- You are given a manhattan grid
- 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

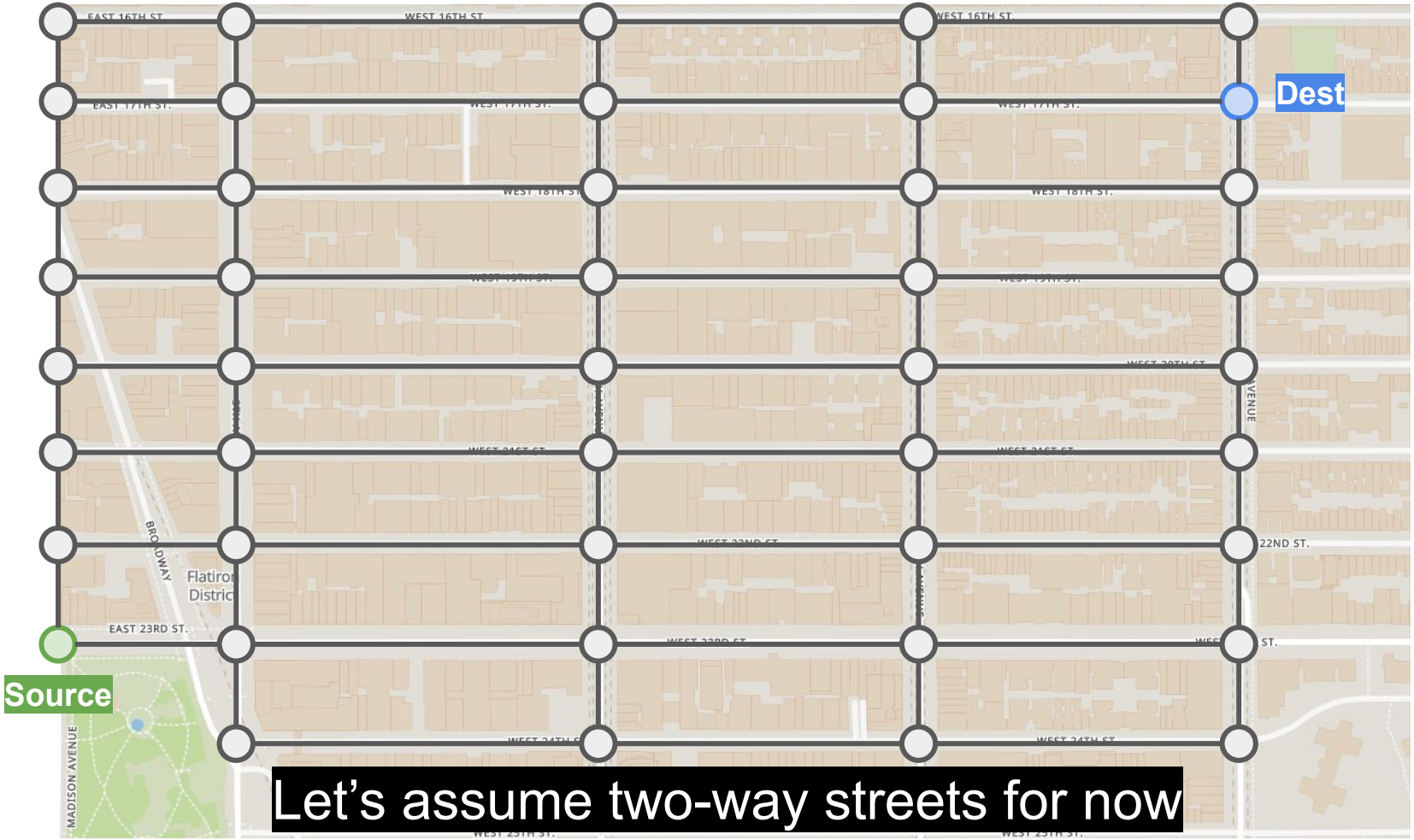
Southeast corner of Madison Square Park (near Madison Avenue &, E 23rd St, New York, NY 10010

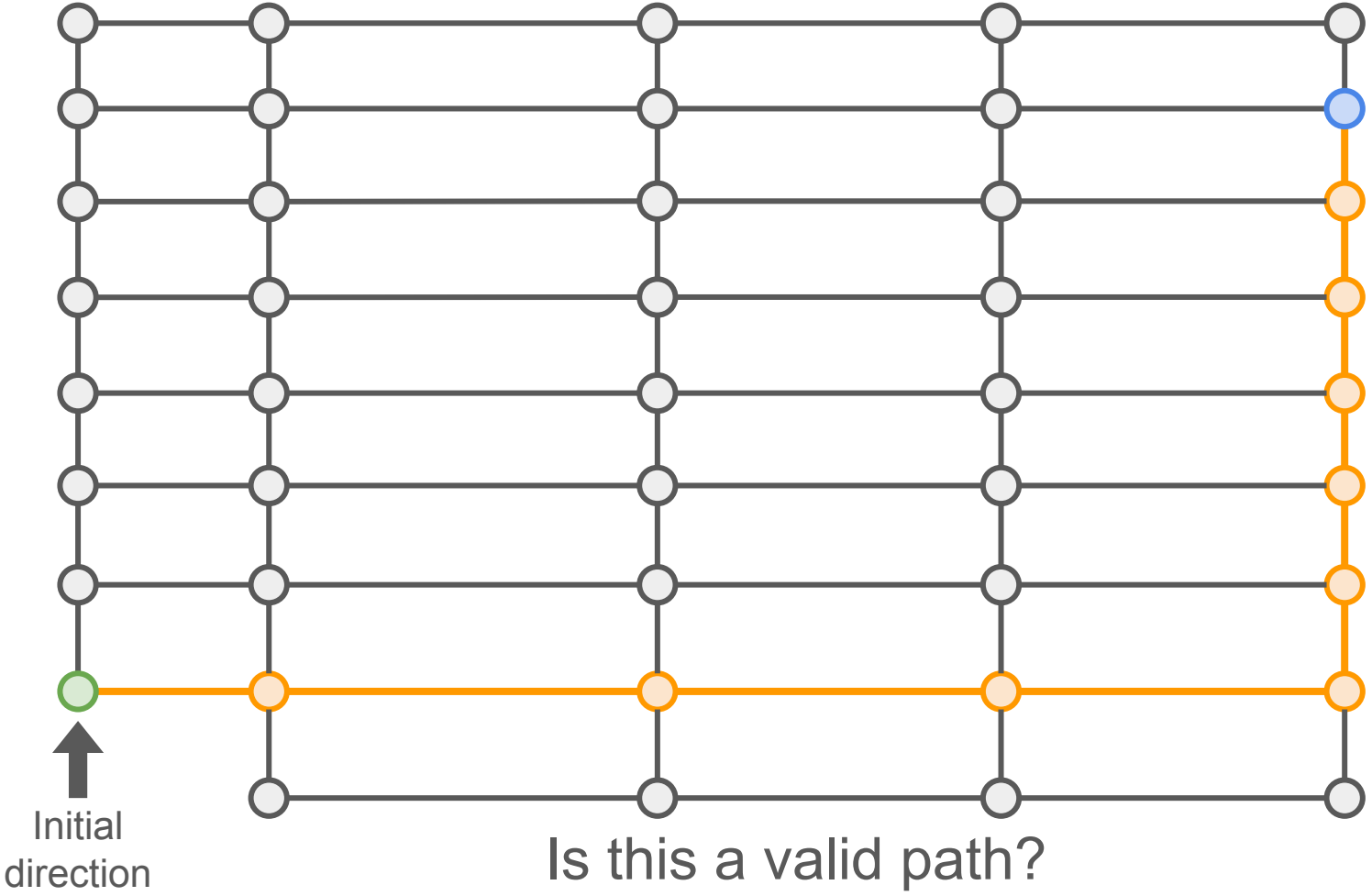
Dest:

Google

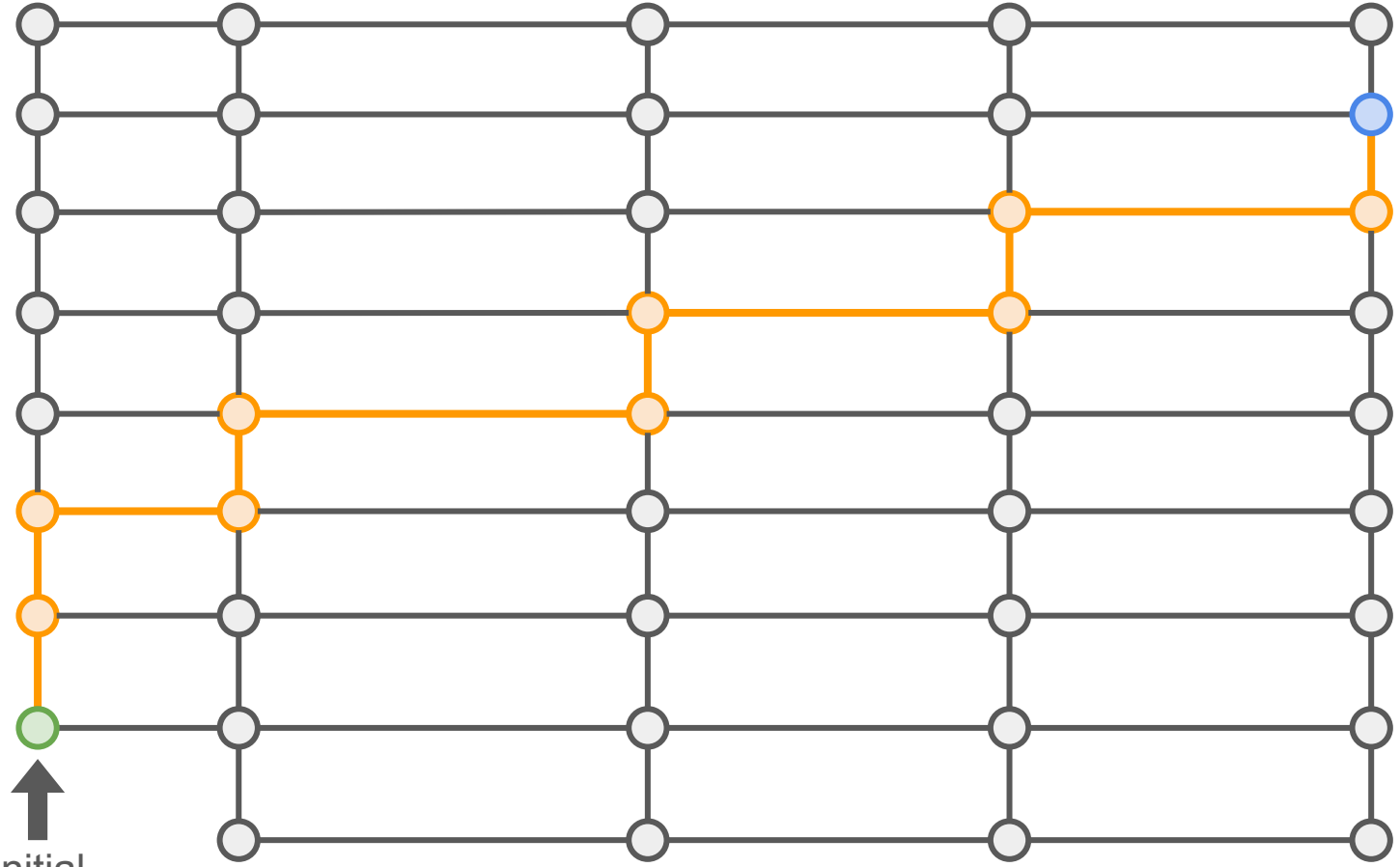
111 8th Ave, New York, NY 10011

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

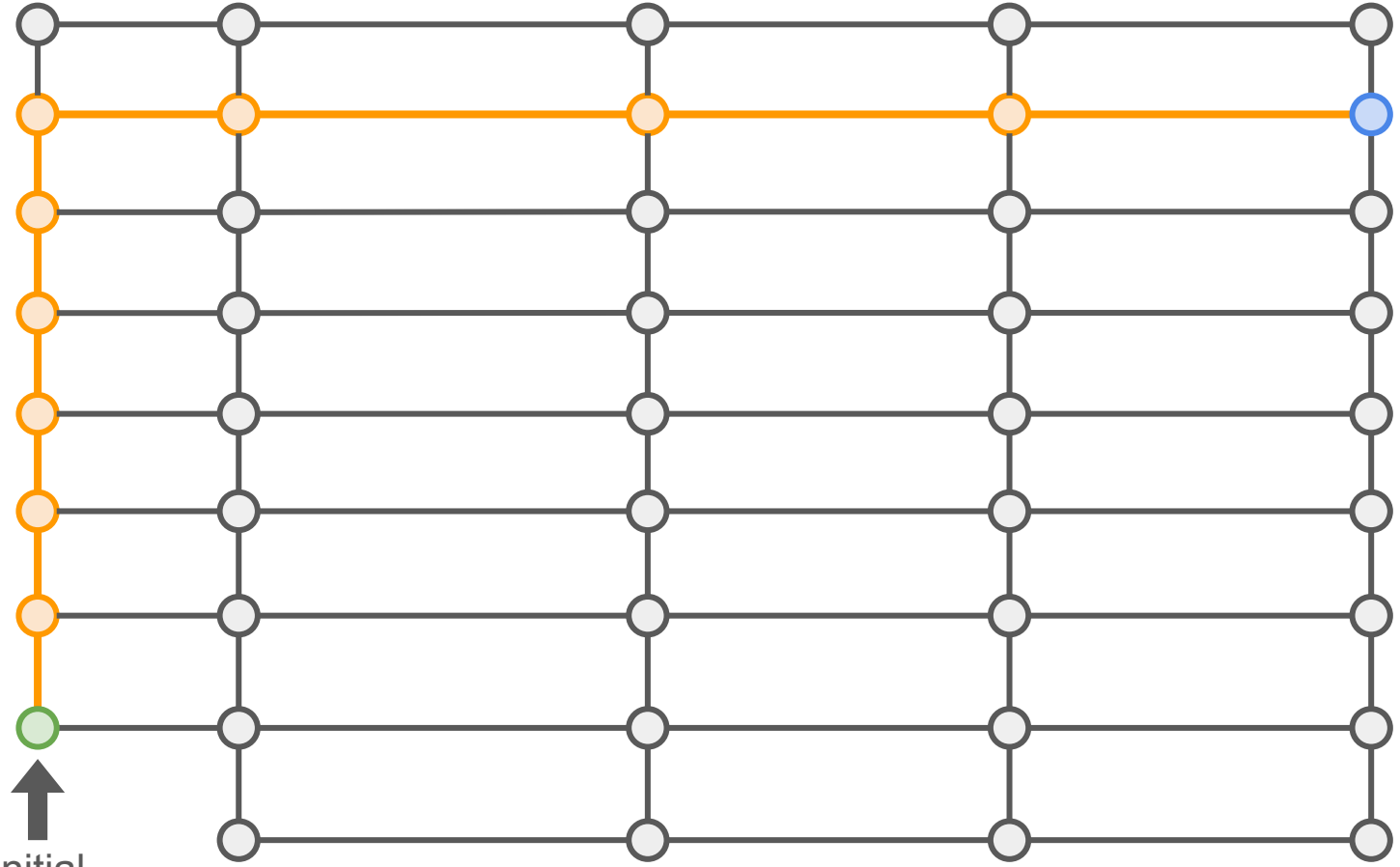

Dest
Source
Let’s assume two-way streets for now



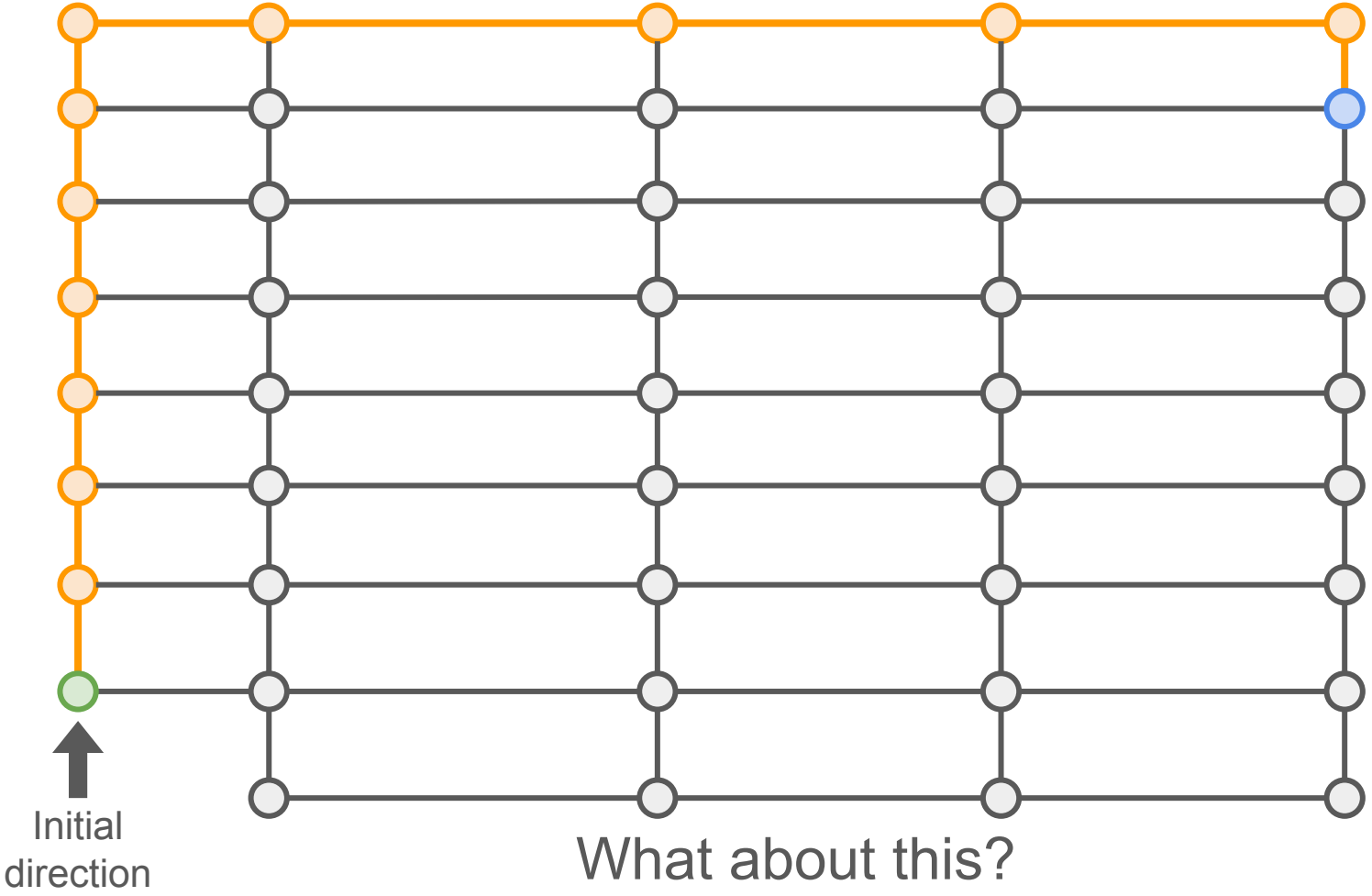
Is this a valid path?



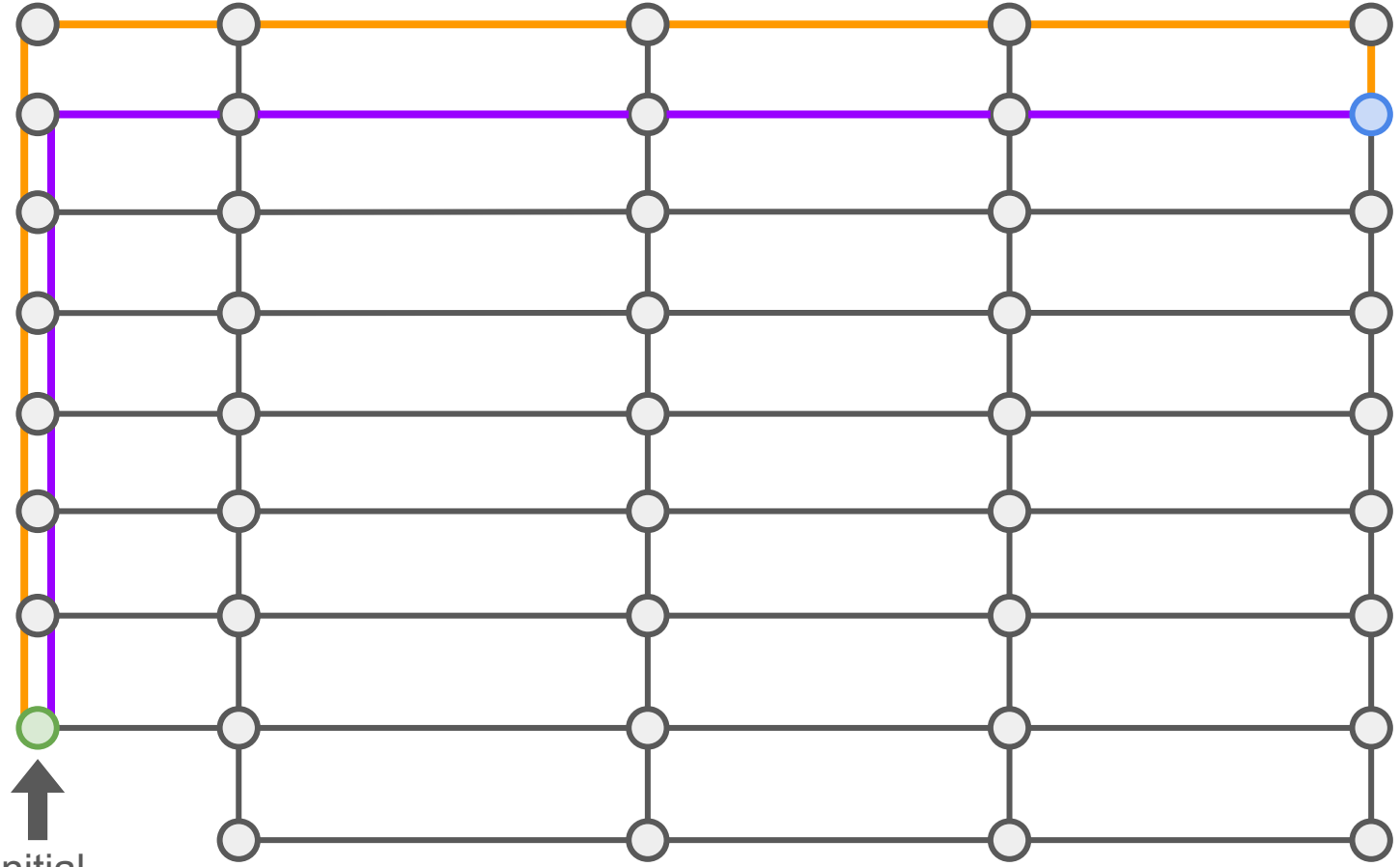
What about this?



What about this?



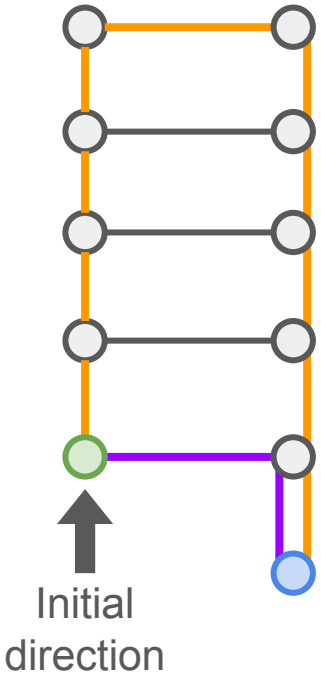
What about this?



So which is a better path?

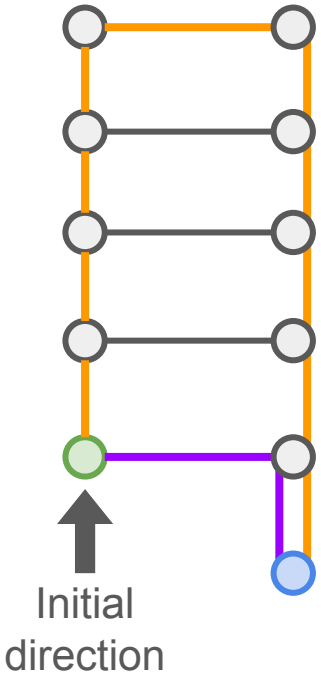
Test yourself!

Which is a better path in this scenario?



Test yourself!

Which is a better path in this scenario?

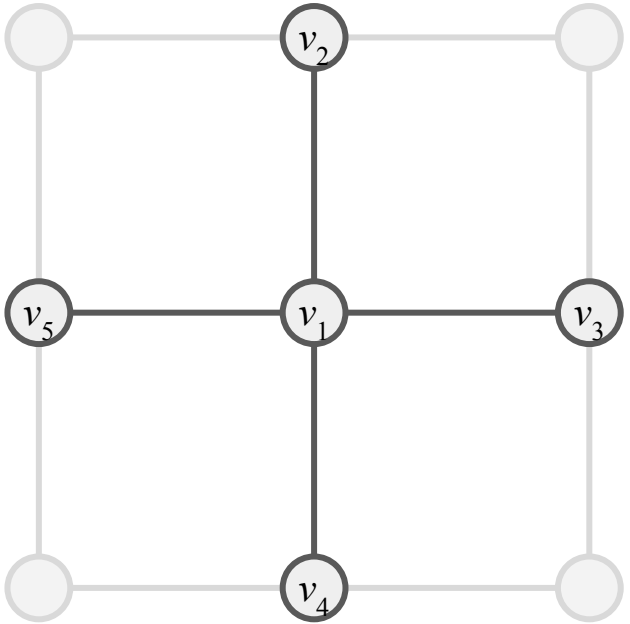


Answer:

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

Input graph

Why does this not work?

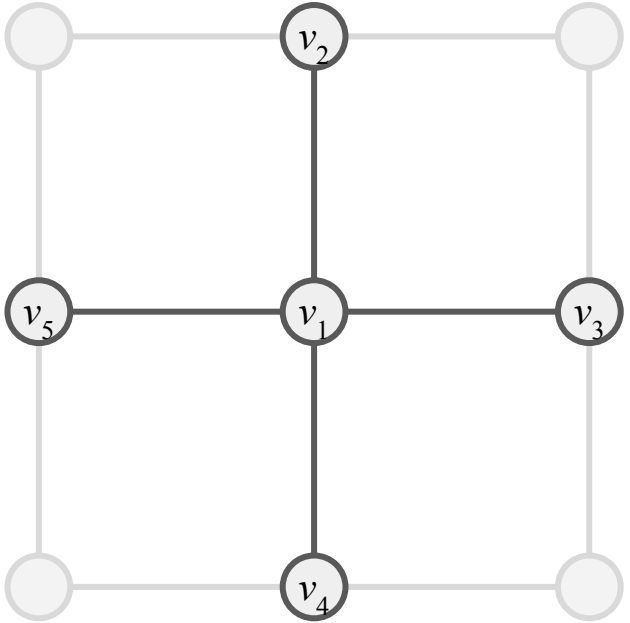


Input graph

Why does this not work?

- Where you can go should be “independent” of where you were
- Where you are now (100%) determines where you can go!

Randomized equivalent: Markov property.



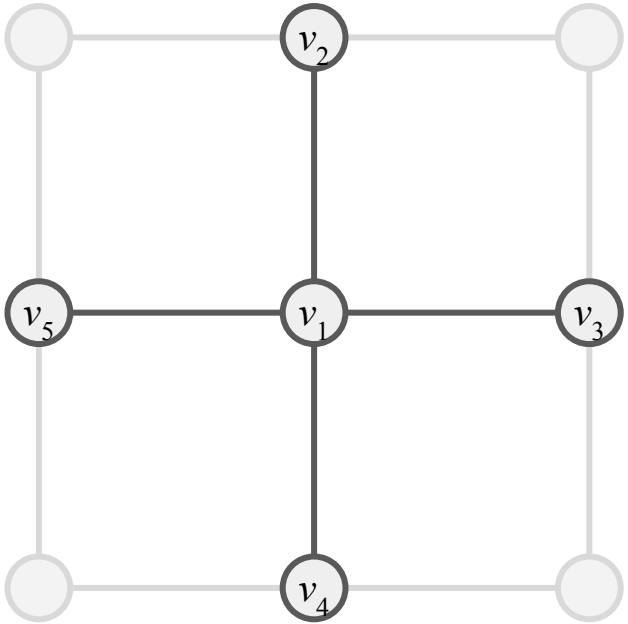
Input graph

Why does this not work?

- Where you can go should be “independent” of where you were
- Where you are now (100%) determines where you can go!

Randomized equivalent: Markov property.

Core concept:
Memoryless



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
  - Edge list

Not relevant here

Concept of graph

- Captures “local topology”
  - Nodes (state)
  - Edges (state transitions)
    - Weighted for reasons:
      - Cost of state transition
      - Probability of state transition

- Do NOT try to conjure your own graph algorithm for your non-graph
  - Transform the “graph” instead

Guiding question

What are the problem states in this question?

Guiding question

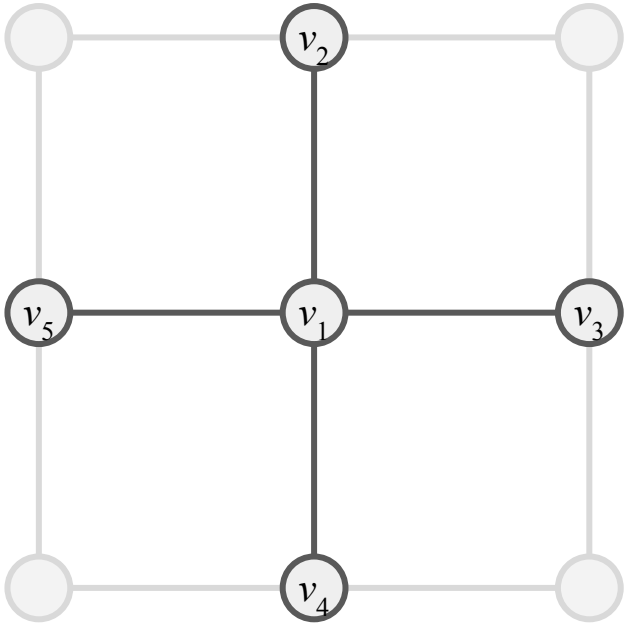
What are the problem states in this question?

Answer:

1. Current location
2. Current direction

Guiding question

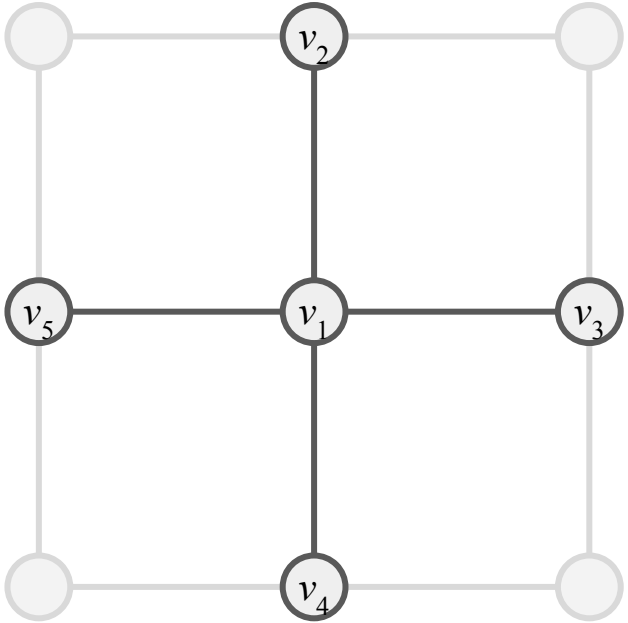
What is the problem with the “raw” input graph?



Guiding question

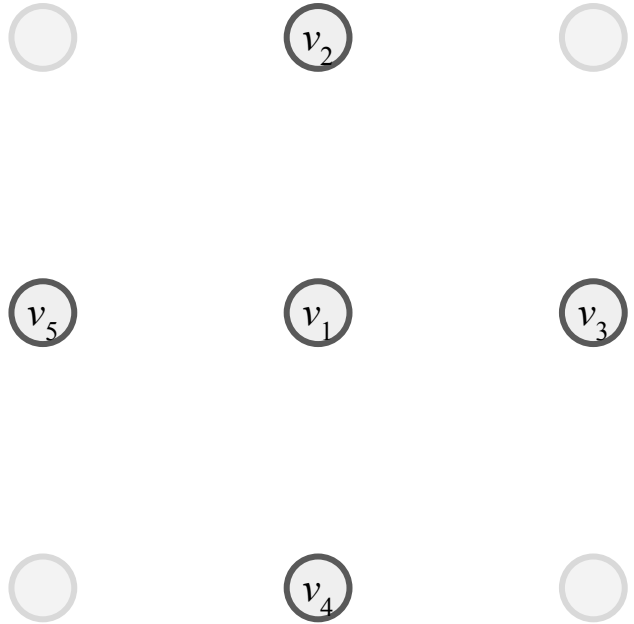
What is the problem with the “raw” input graph?

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?



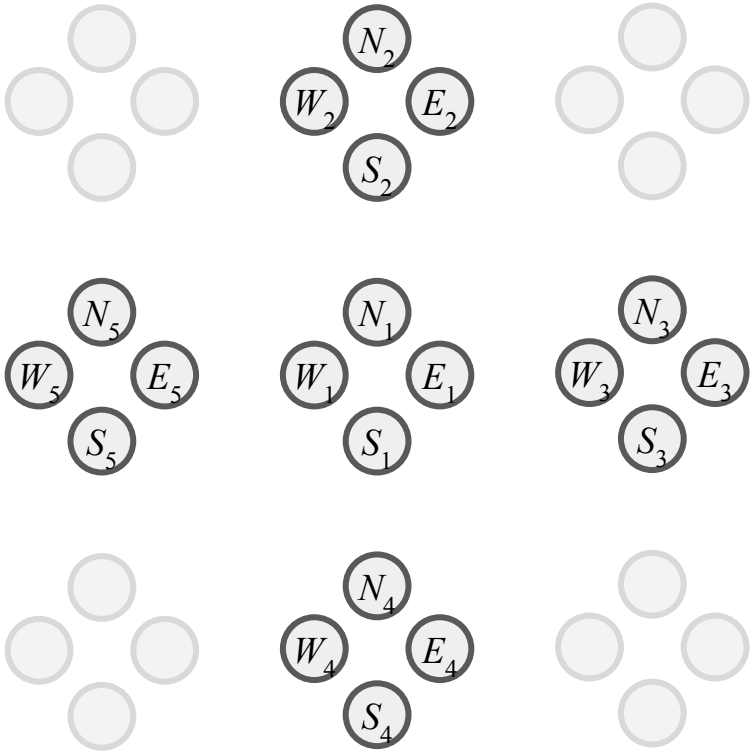
What a real junction looks like

From city skylines

Guiding question

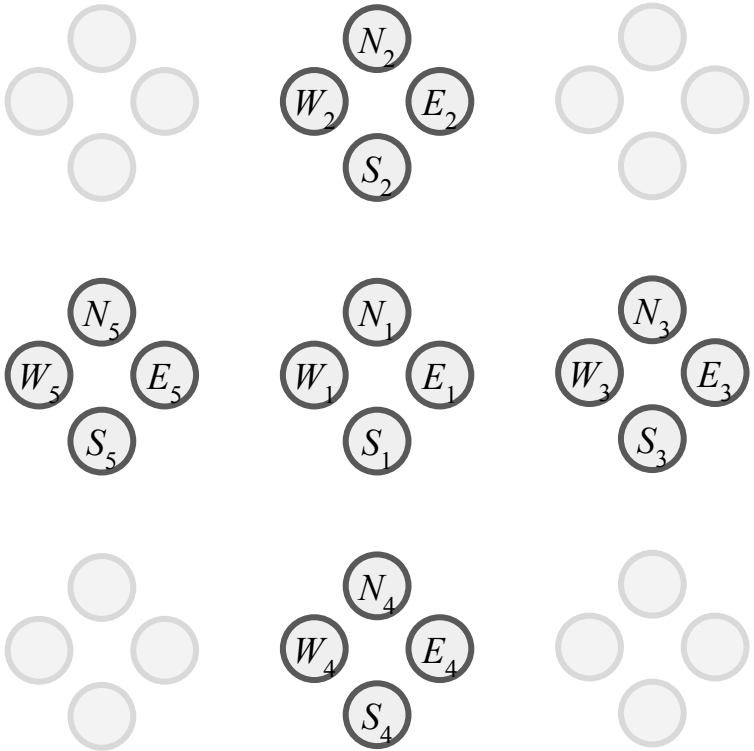
How can we transform the vertices to represent every state fully?

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?

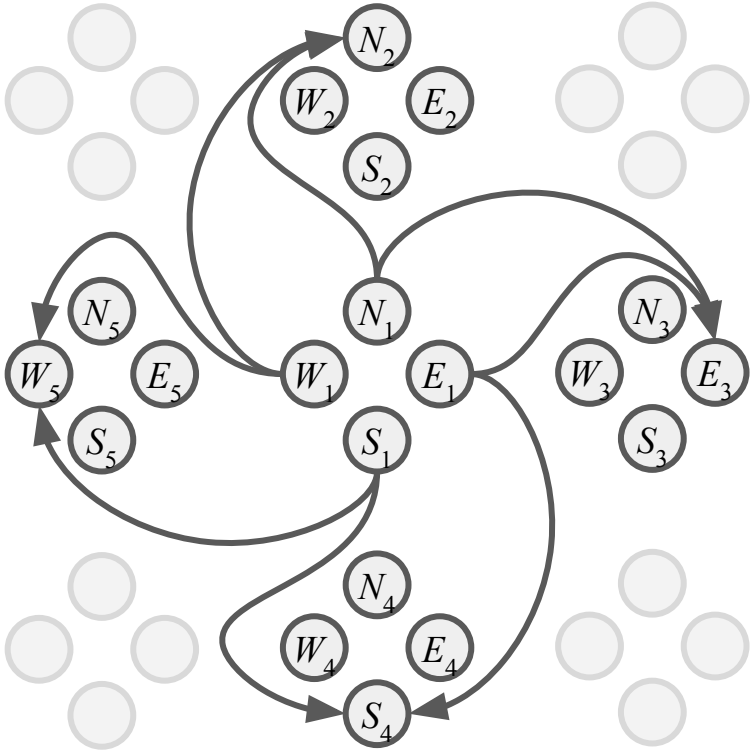


Guiding question

How can we capture all possible state transitions using edges?

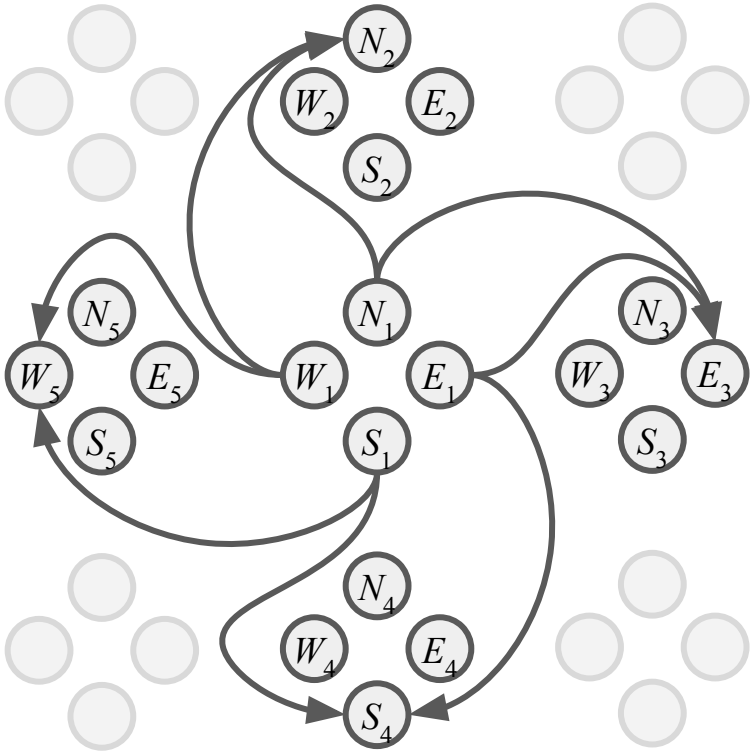
Answer:

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?

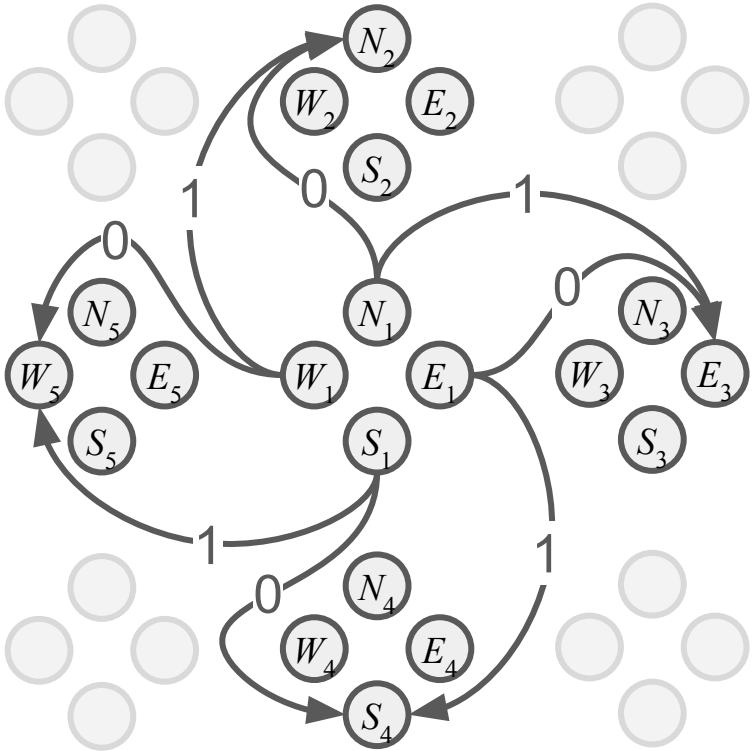


Guiding question

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.

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:
  - Each month has 30 days
  - 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


RUSSIA
Moscow
Tallinn
ESTONIA
LATVIA
Riga
LITHUANIA
Vilnius
BELARUS
Minsk
Mahilyow
Vitsyebsk
Smolensk
Kaliningrad
RUSSIA
Gotland
Baltic Sea
Öland
Stockholm
Oslo
Bergen
Stavanger
Göteborg
Malmö
Copenhagen
DENMARK
North Sea
Skagerrak
Kattegat
ALAND ISLANDS
Aberdeen
Glasgow
Edinburgh
UNITED
KINGDOM
Leeds
Manchester
Birmingham
Cardiff
London
Amsterdam
NETH.
Rotterdam
Bruss
Lille
BEL.
Channel
Essen
Cologne
Bonn
GERMANY
Frankfurt
Luxembourg
LUX.
Strasbourg
Stuttgart
Munich
Hamburg
Bremen
Berlin
Leipzig
Poznań
Warsaw
POLAND
Łódź
Wrocław
Gdańsk
Bornholm
Brest
Hrodna
Homyel'
Chernihiv
Kyiv
Zhytomyr
UKRAINE
L'viv
Vinny tsya
Chernivtsi
Mykolayiv
Odesa
Iaşi
MOLDOVA
Cluj-Napoca
Budapest
HUNGARY
Prague
CZECH REPUBLIC
Brno
SLOVAKIA
Bratislava
Vienna
AUSTRIA
Kraków
Paris
Nantes
FRANCE
Lyon
Bordeaux
Toulouse
Andorra
MONACO
Geneva
Zürich
LIECH.
Vaduz
Milan
Turin
Venice
Genoa
Florence
SAN MARINO
Ligurian Sea
Ljubljana
SLOVENIA
Zagreb
CROATIA
BOSNIA AND HERZEGOVINA
Sarajevo
SERBIA
Pristina
MONT.
ROMANIA
Bucharest
Constanța
Black Sea
Varna
BULGARIA
Sofia
Istanbul
Danube

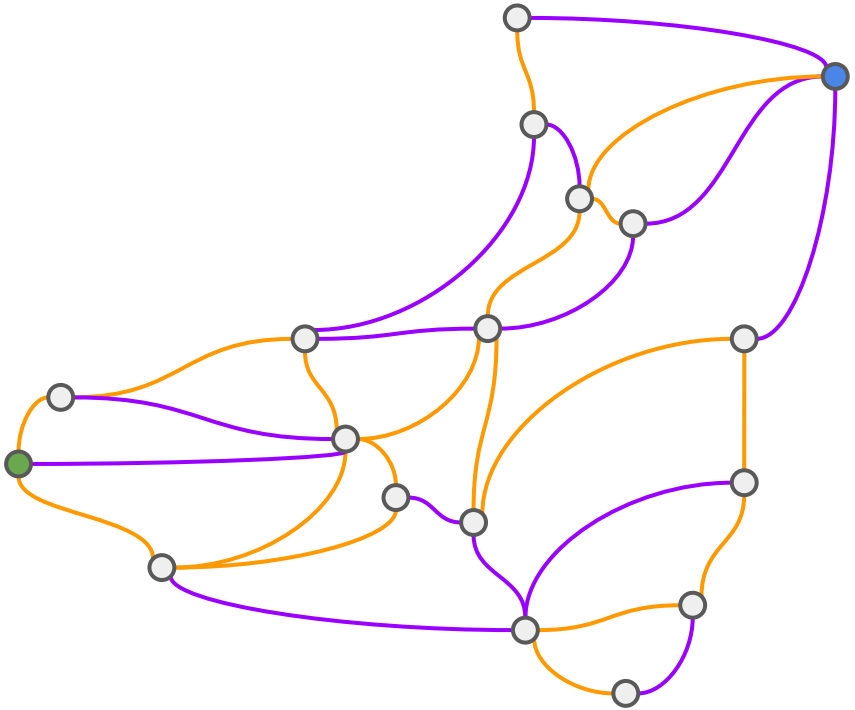
Input graph


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

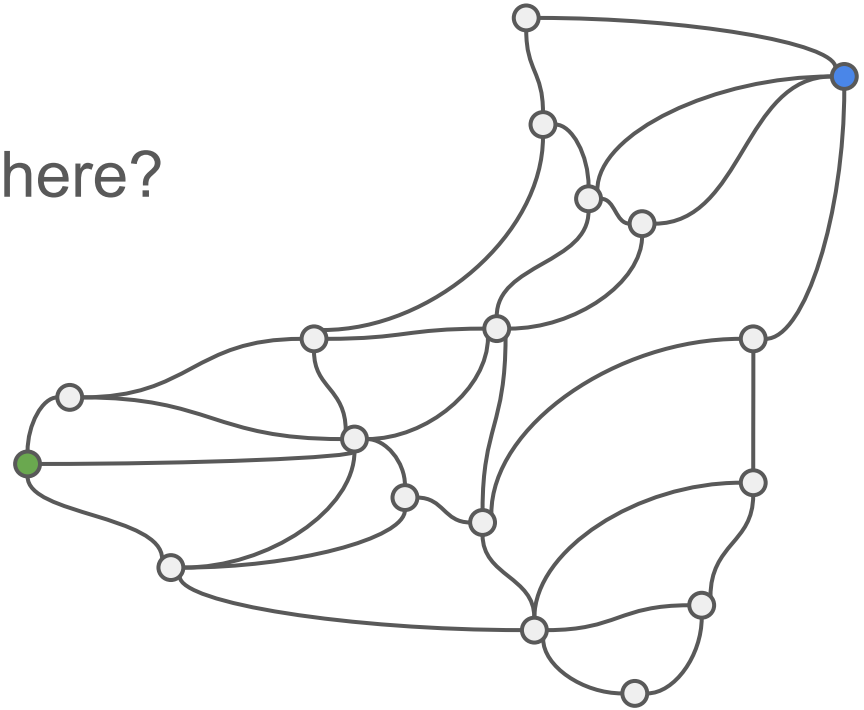
How is the notion of *fastest* captured in the graph here?

Answer: The path with the least number of edges.

Solution

Just run BFS!

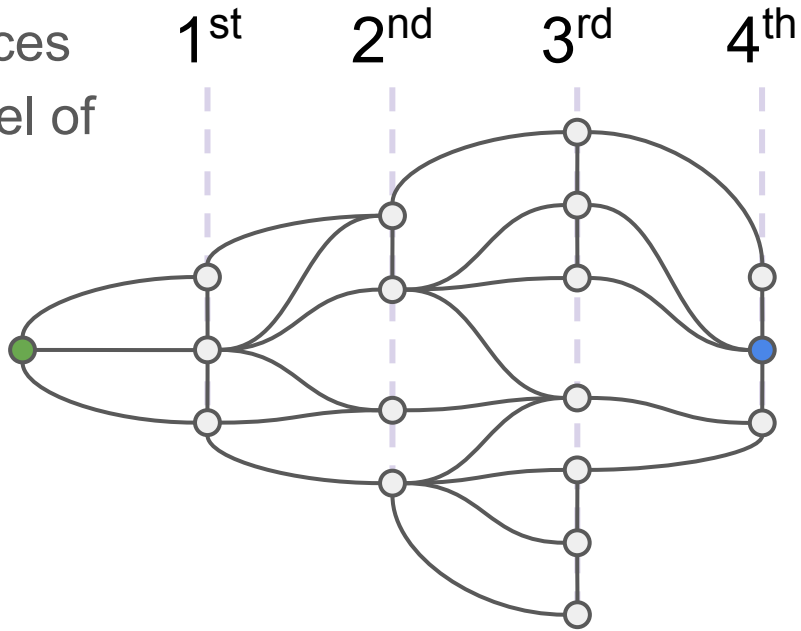
Why does it work here?



Why BFS works

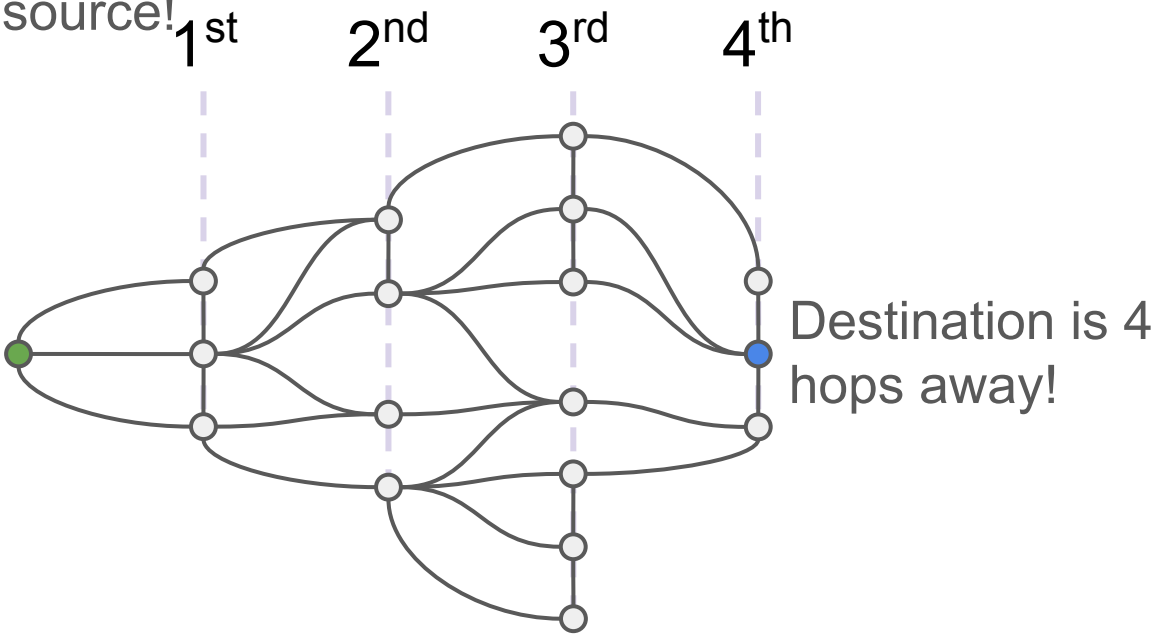
Recall that BFS conducts *level-order* traversal.

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?

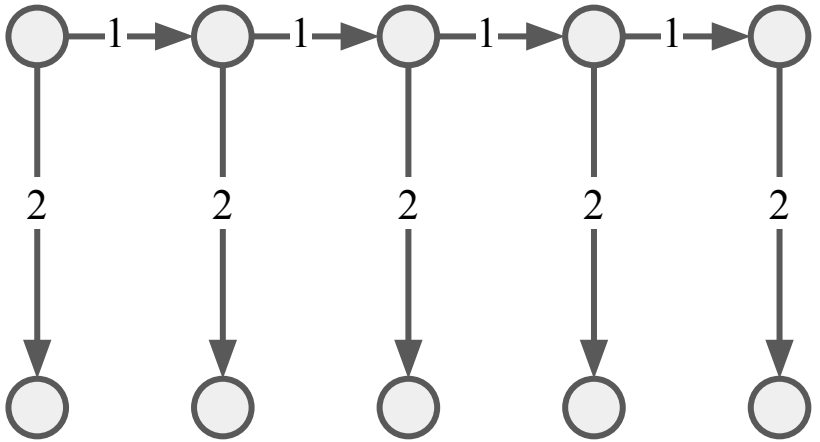
Test yourself!

How do you recover the path?

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

Test yourself!

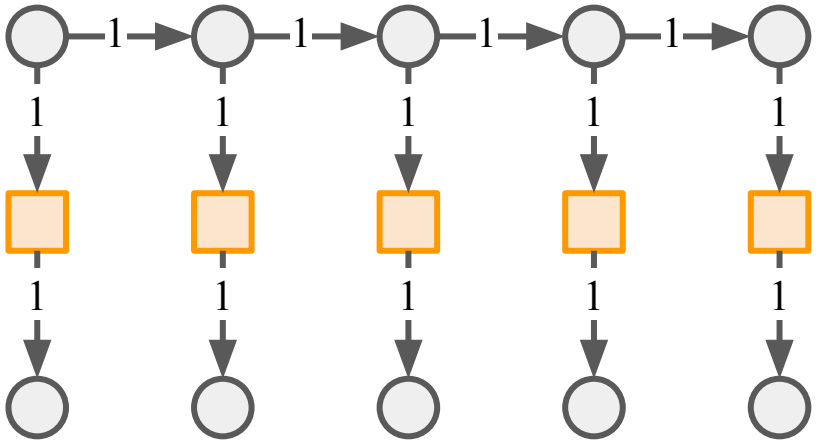
What if some roads take 1 day to traverse while others takes 2? Consider the directed graph below with only edge weights 1 and 2, how can we transform the graph such that we can just run BFS on it to obtain the SSSP?



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.

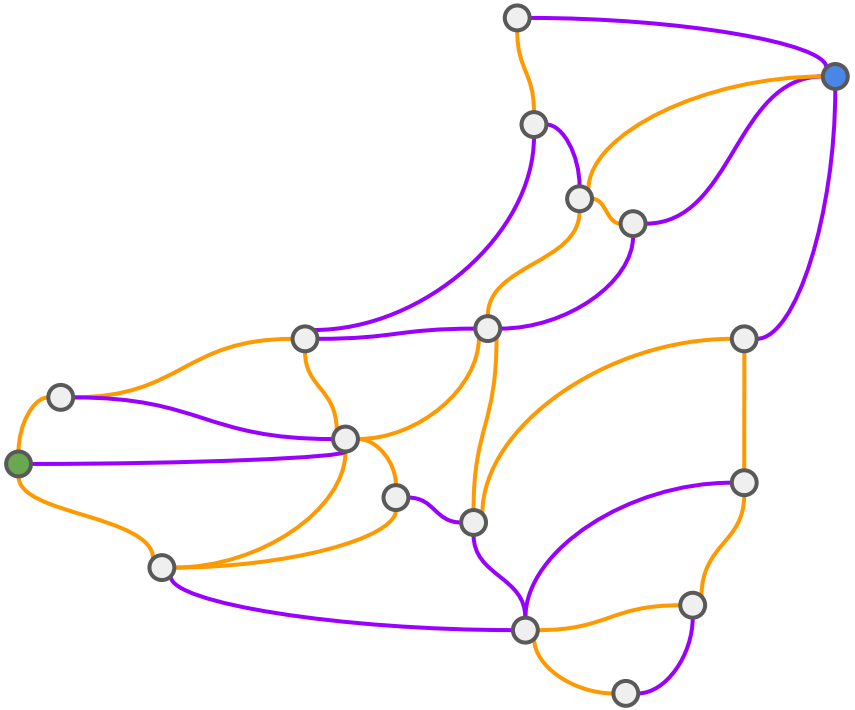
Suppose now Manon will have to take into account the even-odd scheme because her planned trip will *not* overlap with any days of December. Suppose also that she will have to leave from Paris on an *even* day and she will *not* be spending overnight in any city.

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.

Input graph

Even

Odd



Guiding question

What are the problem that the graph should capture?

Guiding question

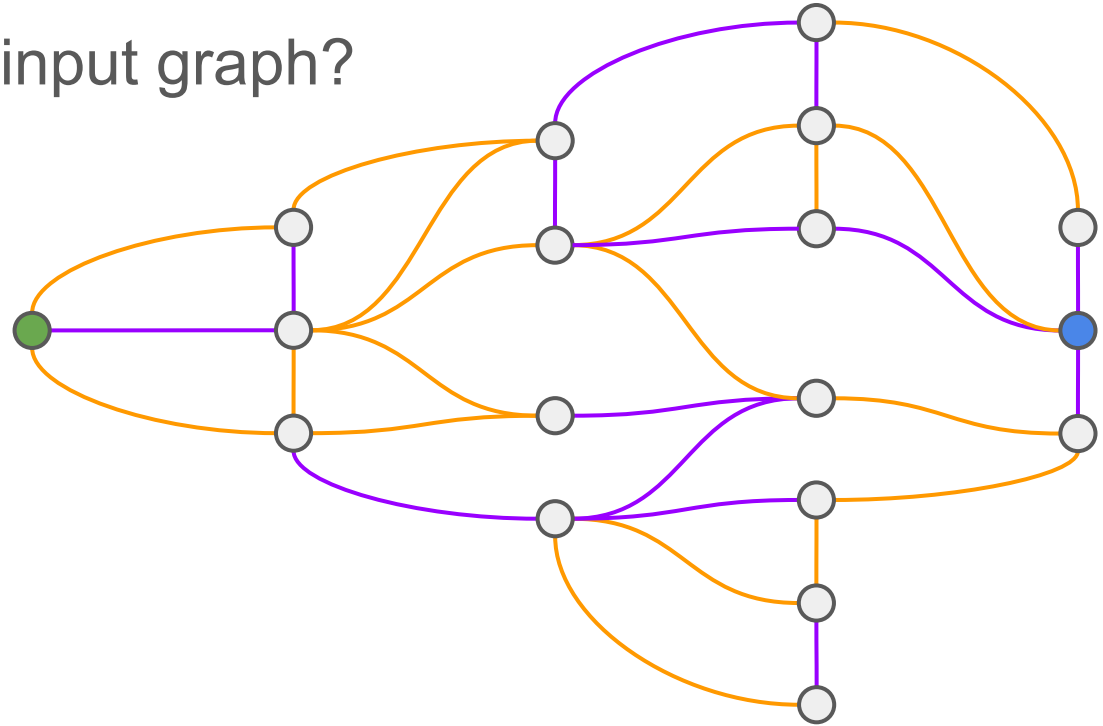
What are the problem that the graph should capture?

Answer:

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

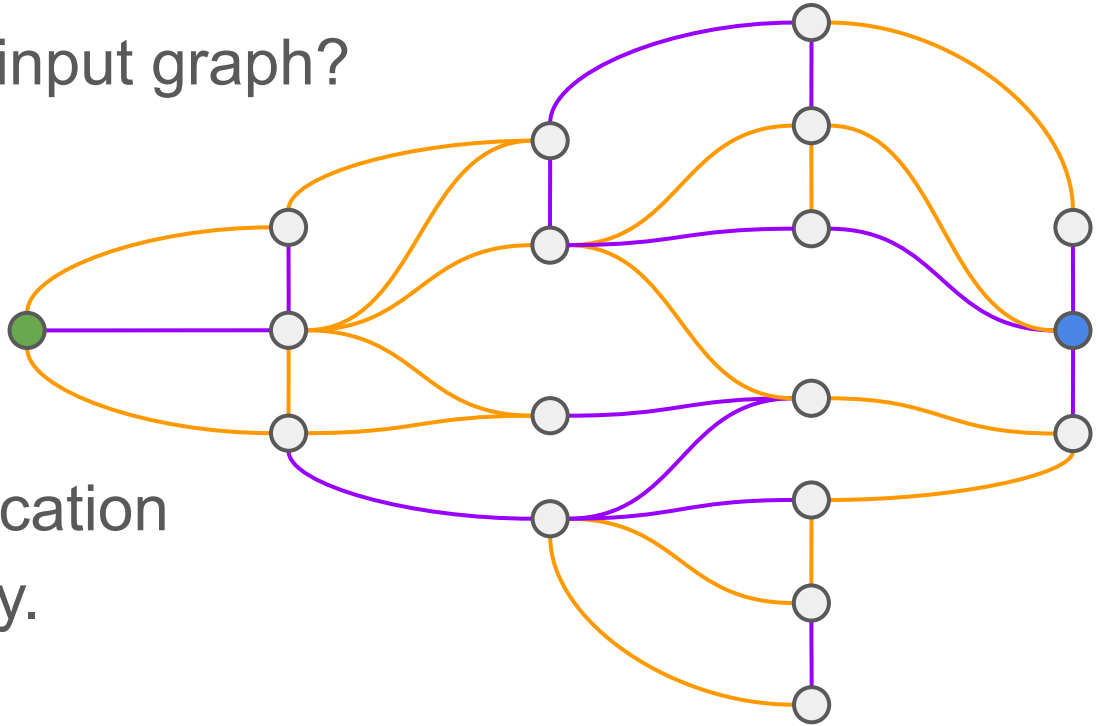
Guiding question

Should we use this raw input graph?



Guiding question

Should we use this raw input graph?

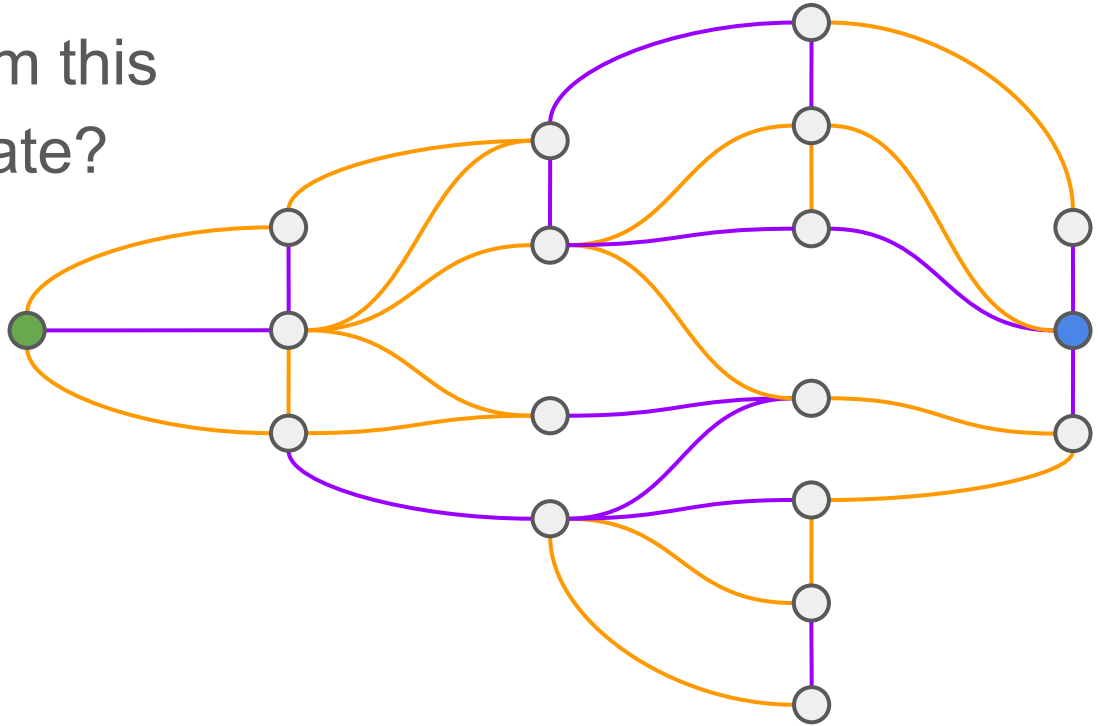


Answer:

No it captures current location but not current day parity.

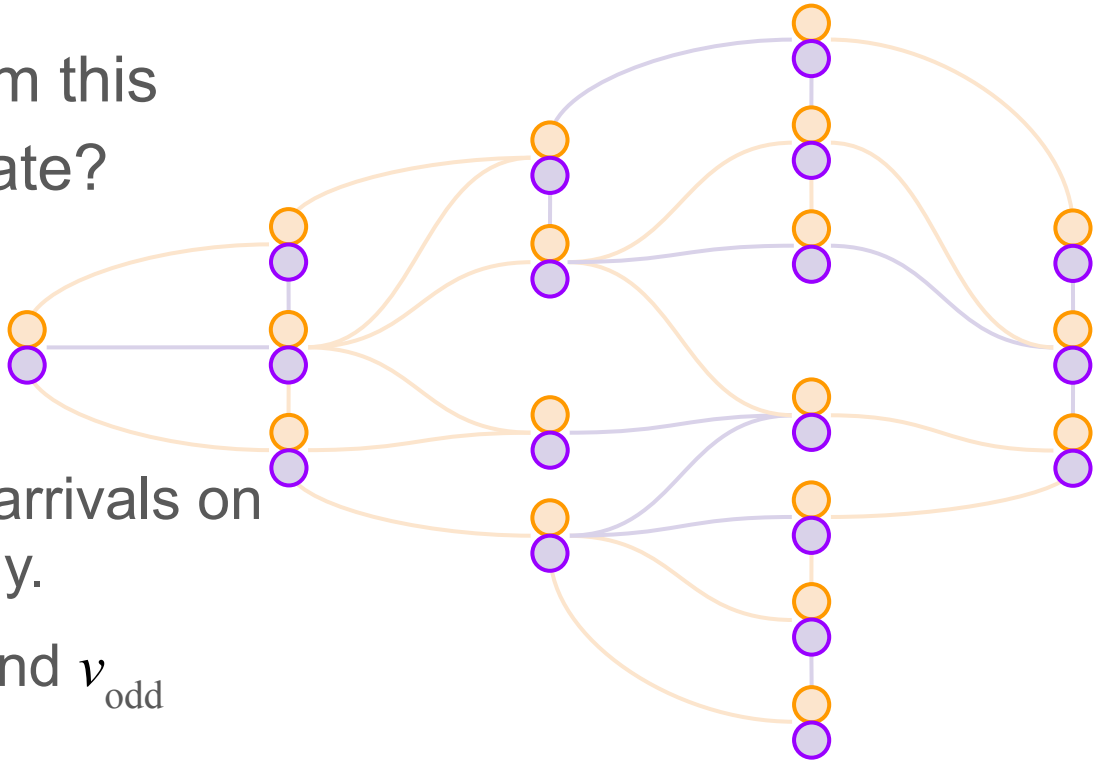
Guiding question

How should we transform this graph to fully capture state?



Guiding question

How should we transform this graph to fully capture state?



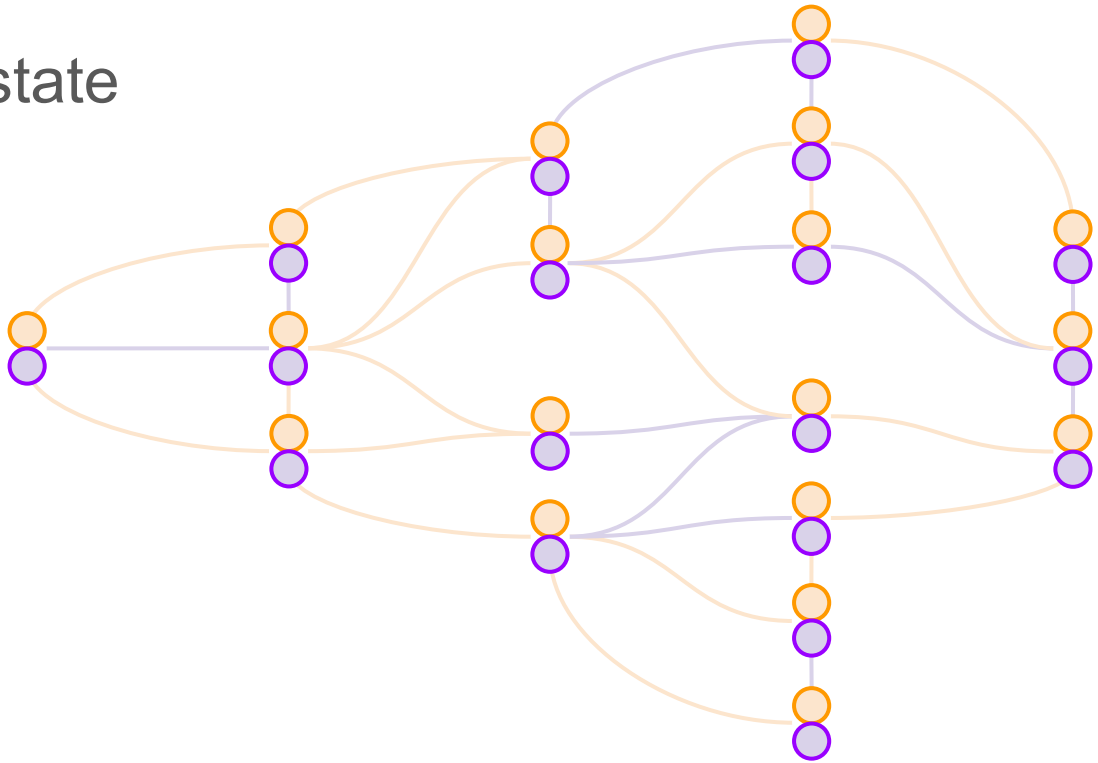
Answer:

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

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

Guiding question

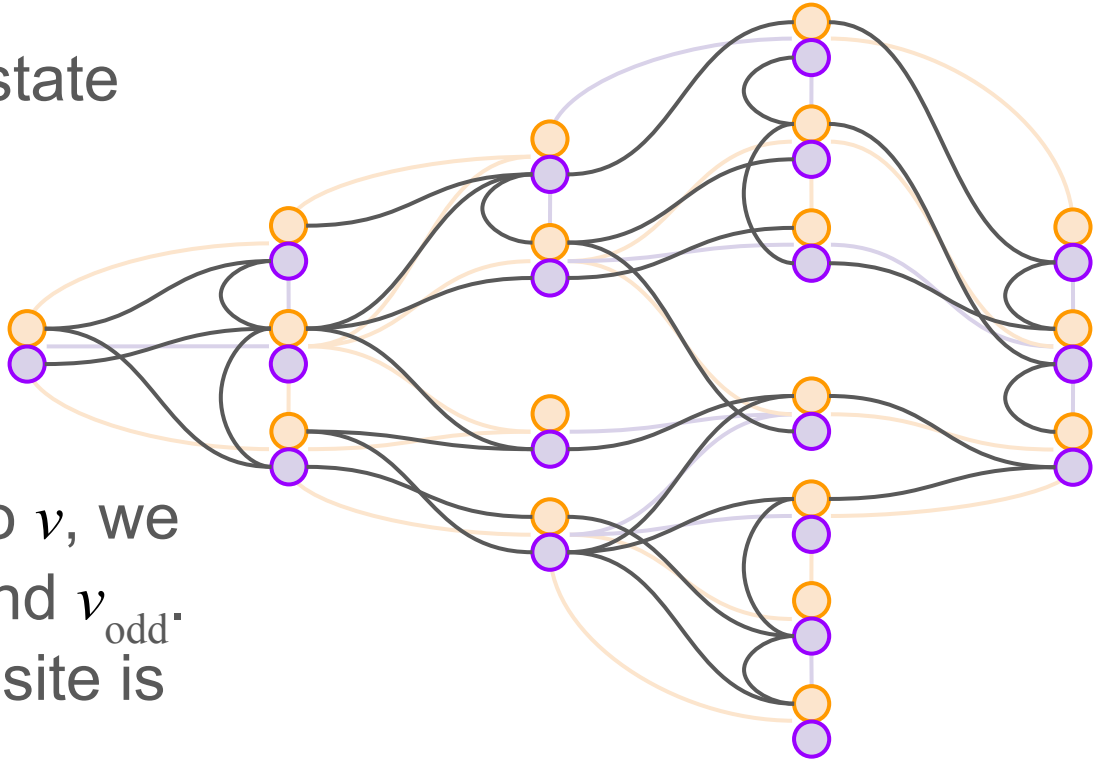
How should we handle state transitions now?



Note: This graph is DIRECTED.

Guiding question

How should we handle state transitions now?



Answer:

For *even* roads from u to v, we connect between u_{even} and v_{odd}. For odd roads, the opposite is done.

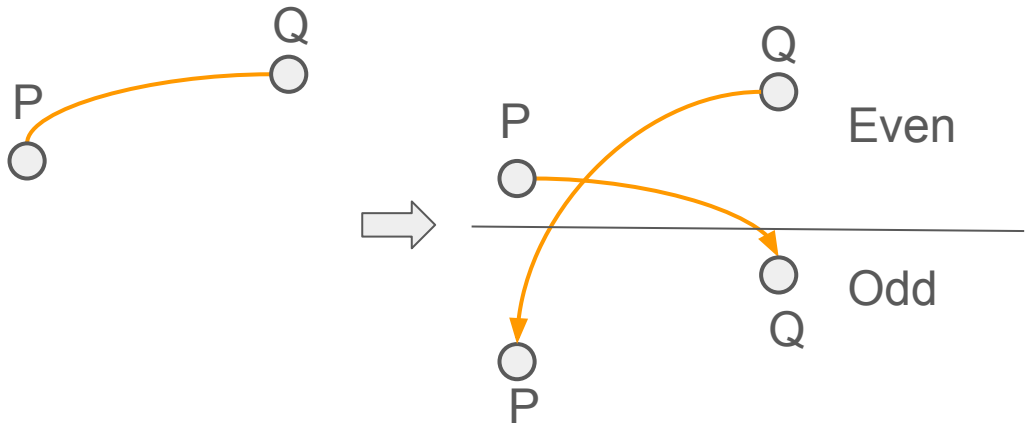
Directed graph

Suppose exist even edge P <-> Q

Then

- P_even -> Q_odd
- Q_even -> P_odd

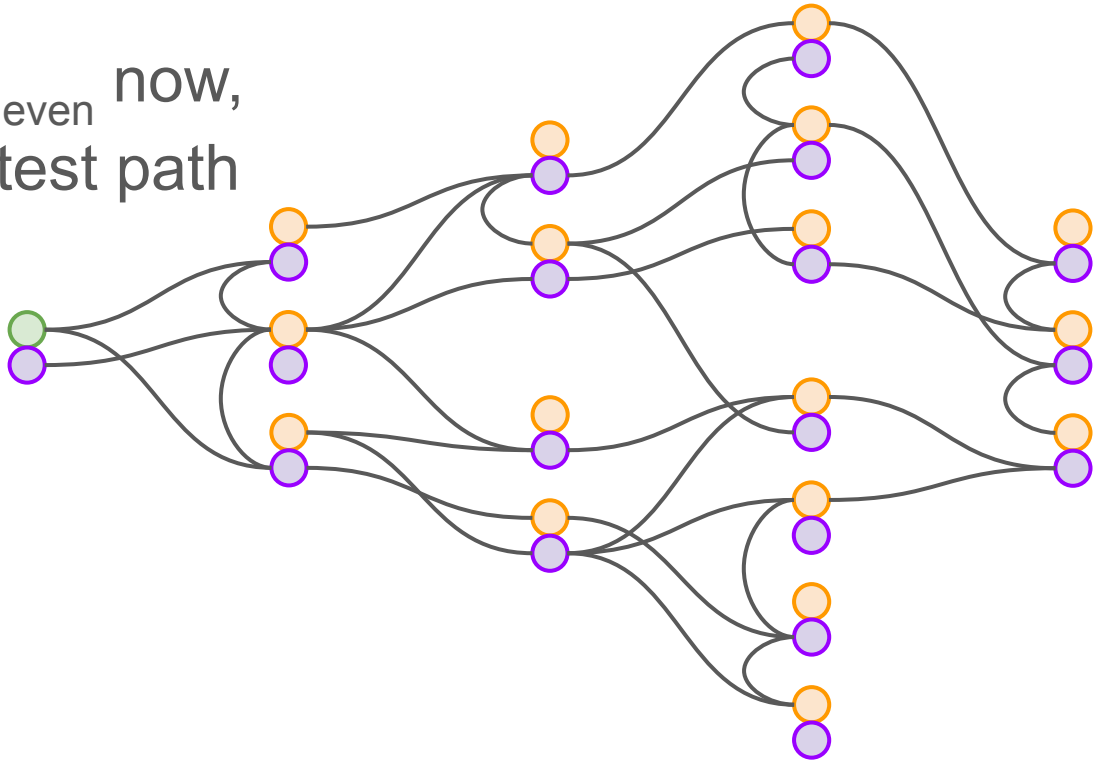
Q_odd -> P_even doesn't exist!



(I.e. when travelling on an even day, you MUST end up on an odd day)

Guiding question

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

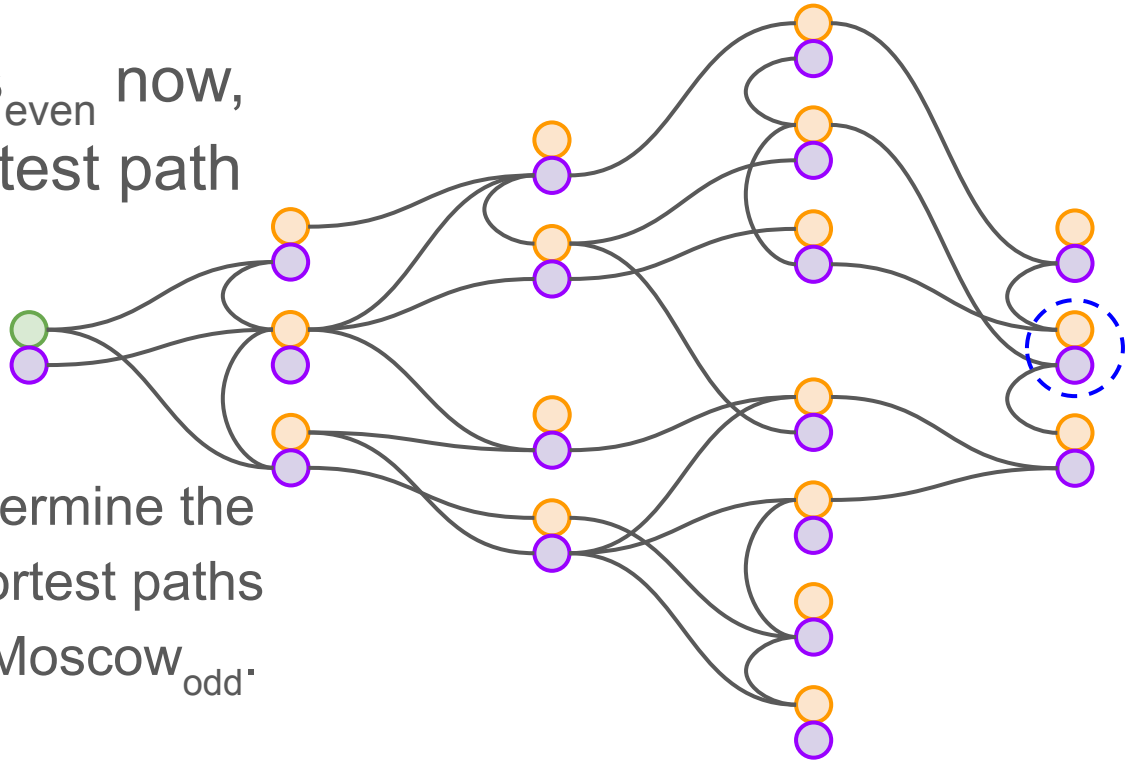


Guiding question

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

Answer:

One final step is need to determine the shorter one between the shortest paths to vertices $\text{Moscow}_{\text{even}}$ and $\text{Moscow}_{\text{odd}}$.



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?

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?

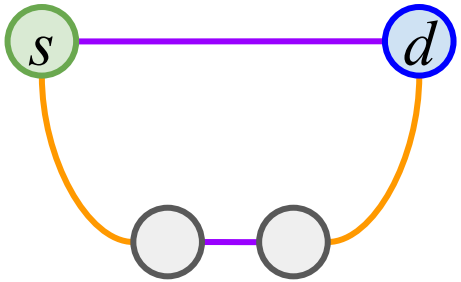
Guiding question

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?

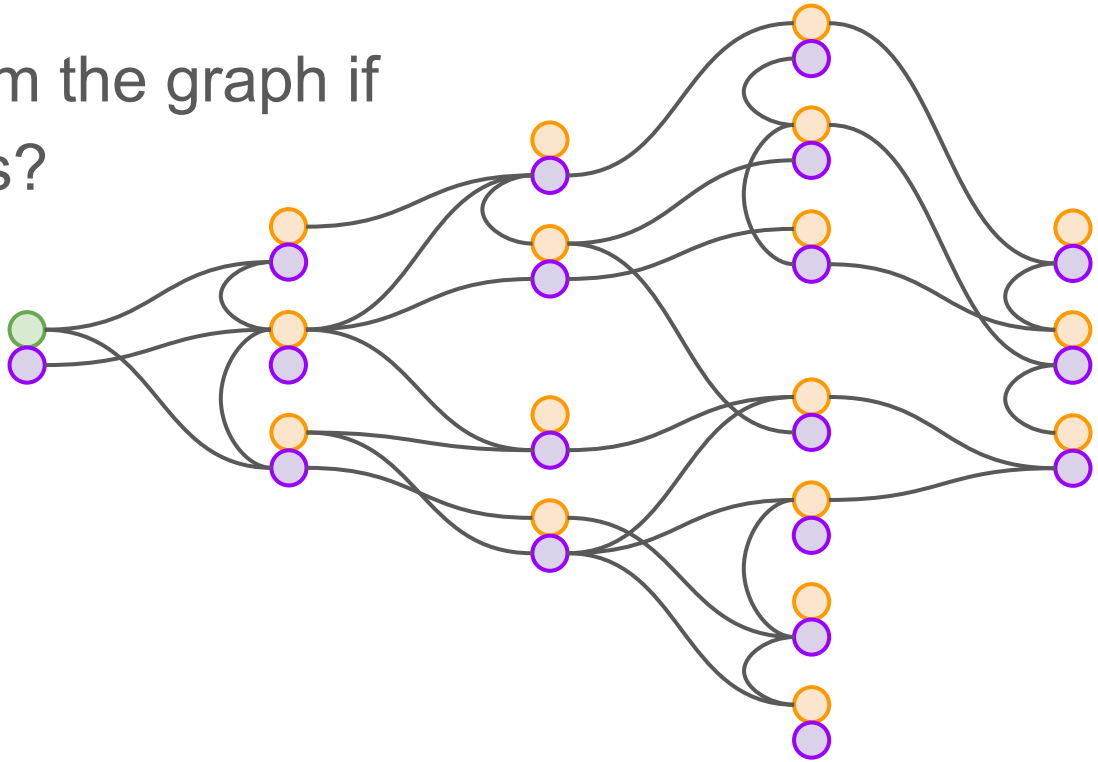
Answer: Yes, consider the scenario on the right where we leave s on an even day and destination is d.



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

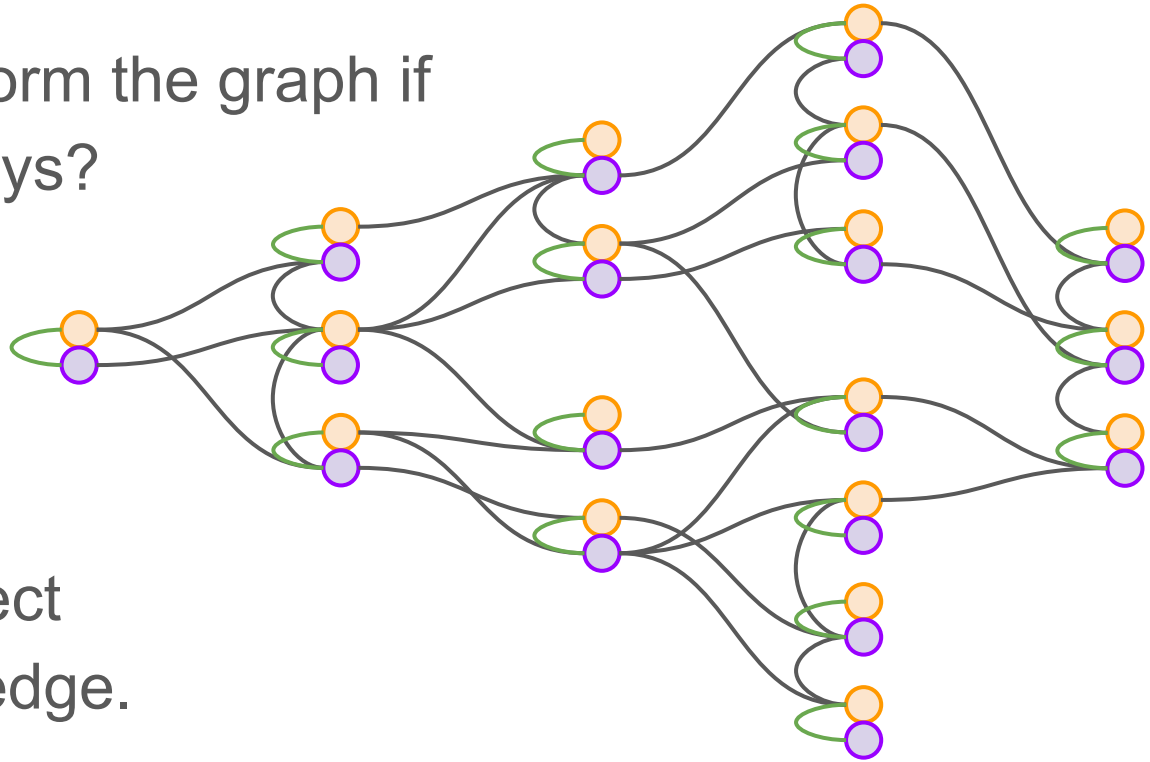
Guiding question

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 an edge.

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

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?

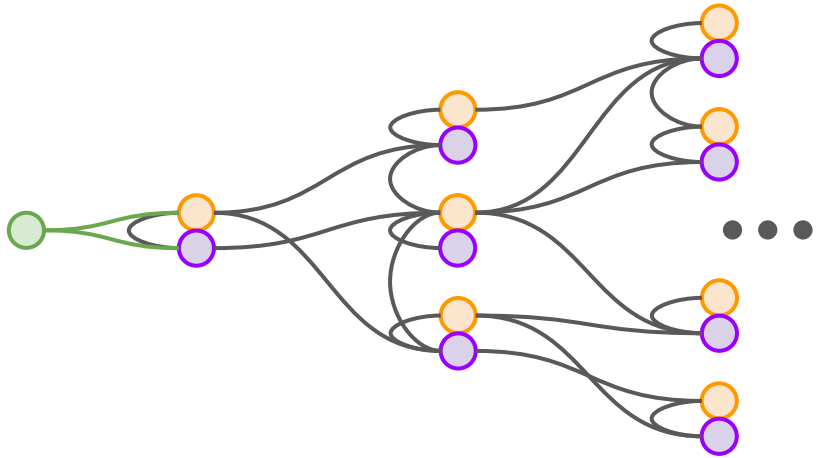
Guiding question

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?

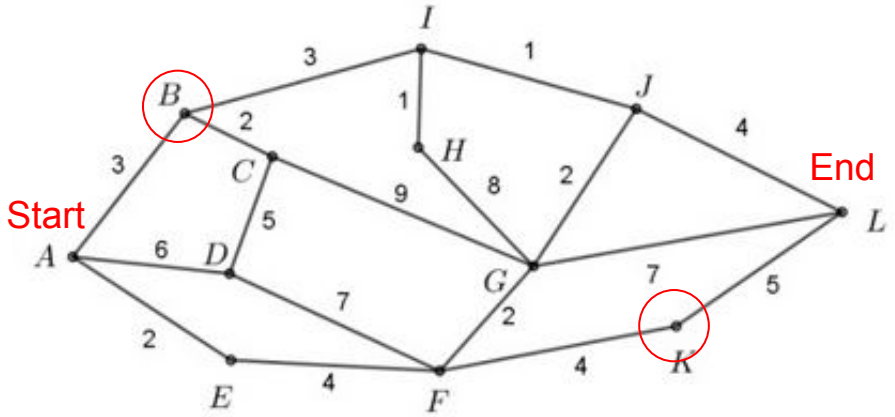
Answer: The solution is *not* unique!

Problem 3: Pizza trip (Bonus)

Problem 3a

You have just ended work and are going to a friend's house for a party.

You want to stop to buy pizza on the way at any pizza place P_i.

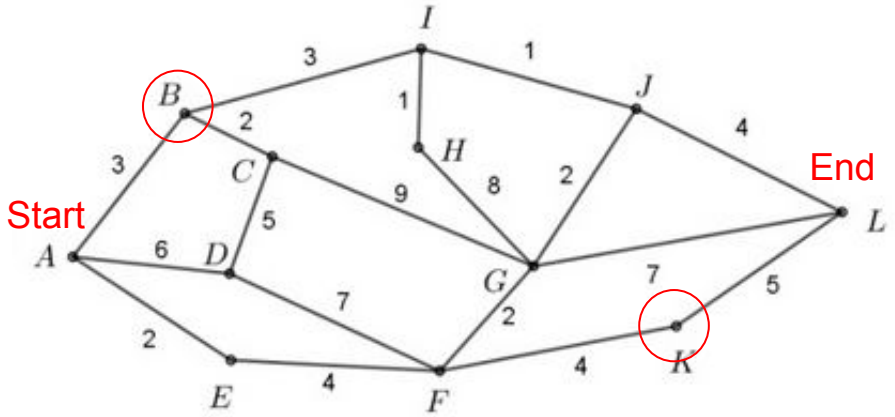


How to transform graph to use SSSP?

Problem 3b

Each of the Pizza places P_i takes t_i to produce your order.

Modify the graph accordingly.



Problem 3c

What if you have to pick up both Pizza and Sushi?

Modify the graph accordingly.