CS221 Section 3: Search

DP, UCS and A*

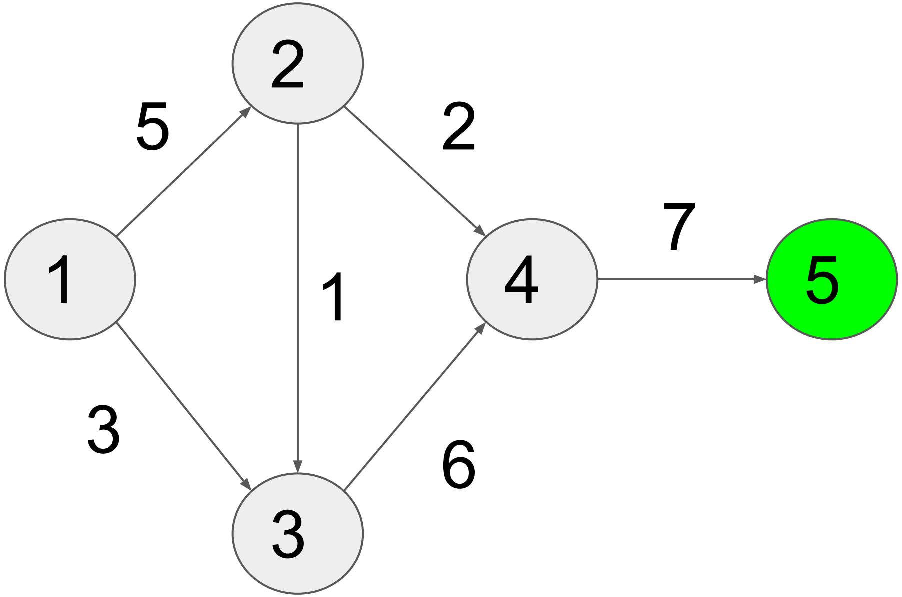
Problem

There exists N cities, conveniently labelled from 1 to N.

There are roads connecting some pairs of cities. The road connecting city i and city j takes c(i,j) time to traverse. However, one can only travel from a city with smaller label to a city with larger label (i.e. each road is one-directional).

From city 1, we want to travel to city N. What is the shortest time required to make this trip, given the additional constraint that we should visit more odd-labeled cities than even labeled cities?

Example



Best path is [1, 3, 4, 5] with cost 16.

[1, 2, 4, 5] has cost 14 but visits equal number of odd and even cities.

State Representation

Key idea: state

A **state** is a summary of all the past actions sufficient to choose future actions **optimally**.

State Representation

We need to know where we are currently at: **current_city**

We need to know how many odd and even cities we have visited thus far: **#odd, #even**

State Representation: (**current_city, #odd, #even**)

Total number of states: $O(N^3)$

Can We Do Better?

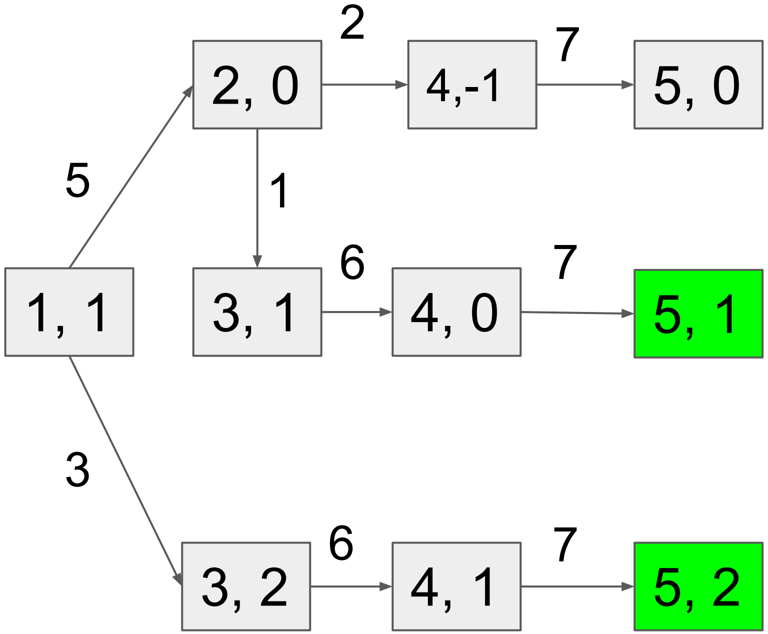
Check if all the information is really required

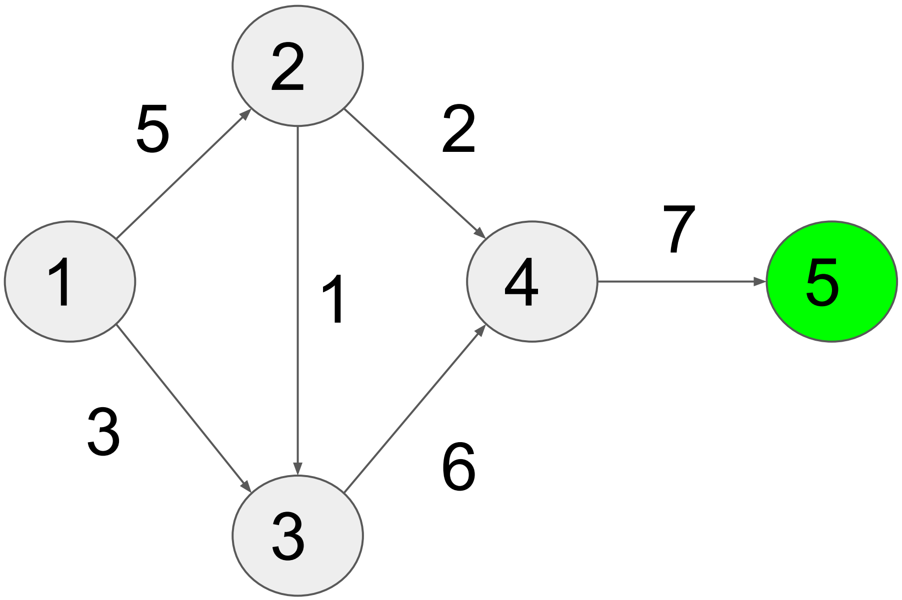
We store **#odd** and **#even** so that we can check whether **#odd** - **#even** > 0 at (N, **#odd**, **#even**)

Why not store **#odd - #even** directly instead?

(**current_city, #odd - #even**) -- $O(N^2)$ states

State Graph





Precise Formulation of Problem

Let E be the set of roads between cities

State s = (i, d) -- (current_city, #odd - #even)

Actions(s): {move(j) for $(i,j) \in E$}

Cost(s, move(j)): c(i,j)

Succ(s, a): (j, d + 1) if j is odd, (j, d - 1) otherwise

Start: (1, 1)

isGoal(s): i = N and d > 0

Solving the Problem

Since we are computing shortest path, which is some form of optimization, we consider DP and UCS.

Recall

- DP can handle negative edges but works only on DAGs
- UCS works on general graphs, but cannot handle negative edges

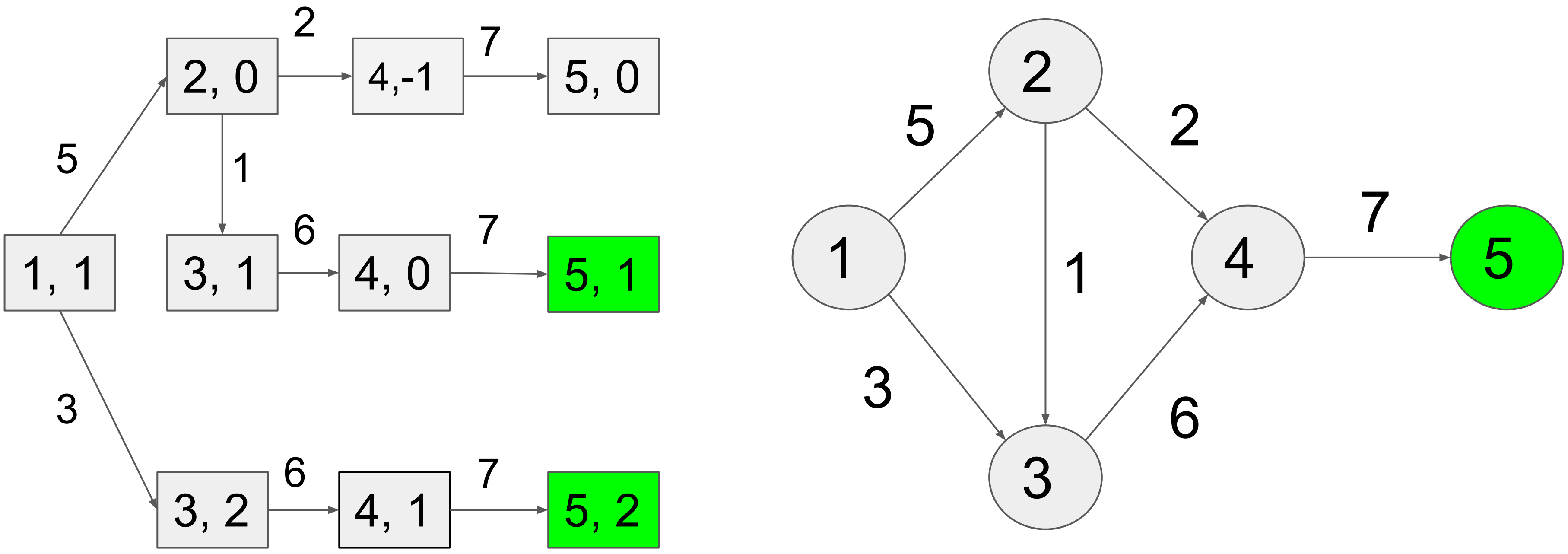
Since we have a DAG and all edges are positive, both work!

Solving the Problem: Dynamic Programming

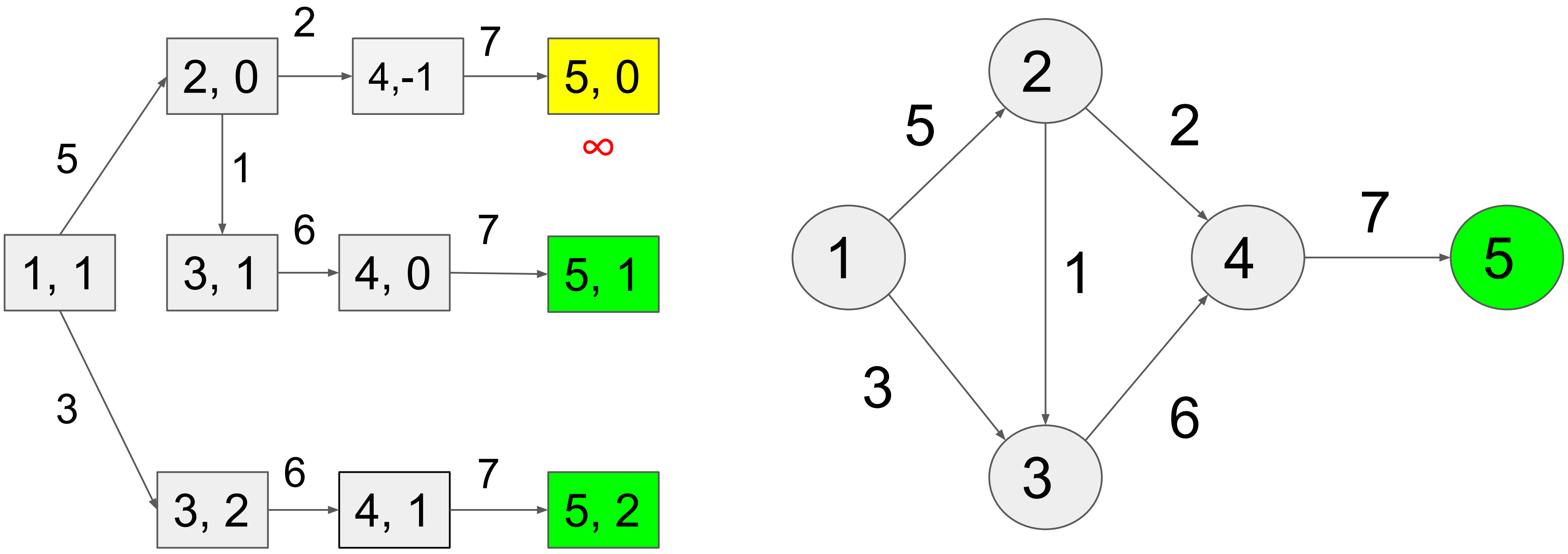
$$\text{FutureCost}(s) = \begin{cases} 0 & \text{if IsGoal}(s) \\ \min_{a \in \text{Actions}(s)} [\text{Cost}(s, a) + \text{FutureCost}(\text{Succ}(s, a))] & \text{otherwise} \end{cases}$$

If s has no successors, we set it’s FutureCost as infinite.

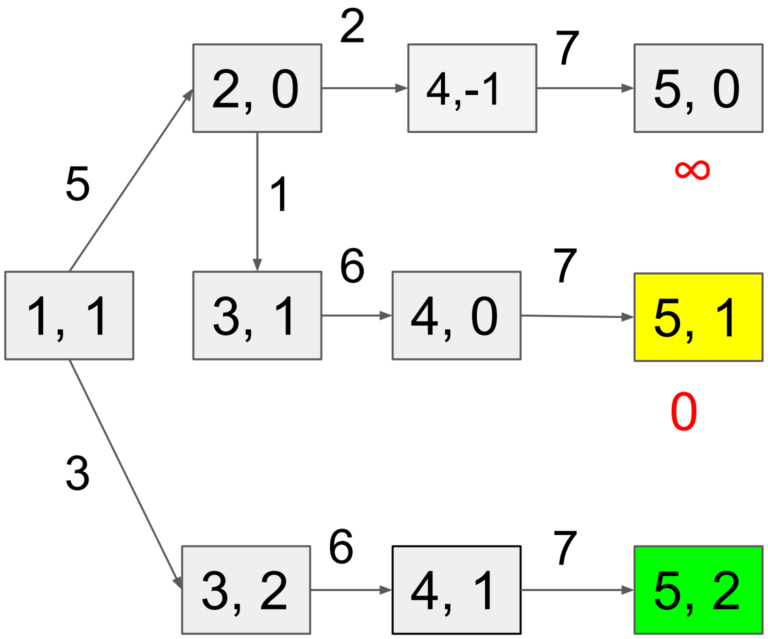
Simulation of Dynamic Programming

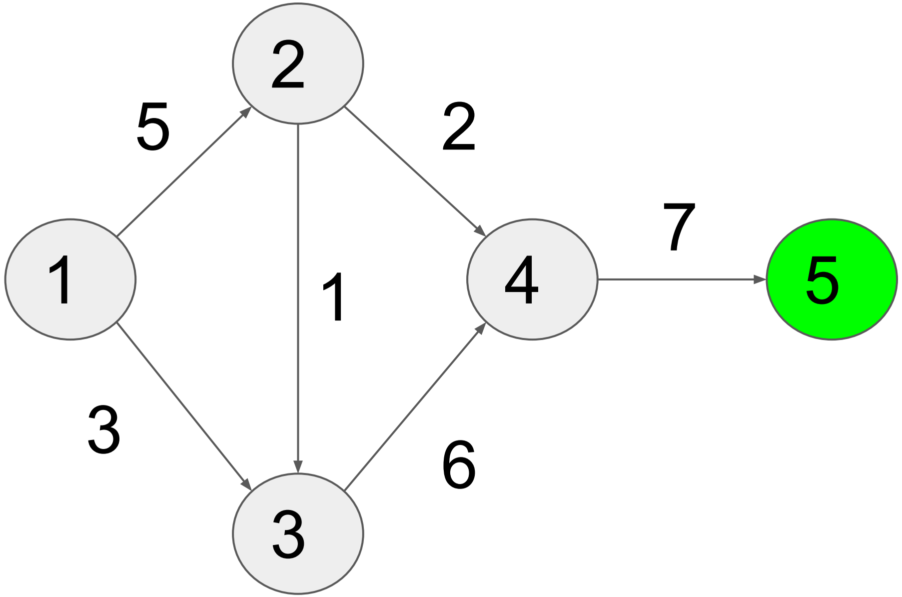


Simulation of Dynamic Programming

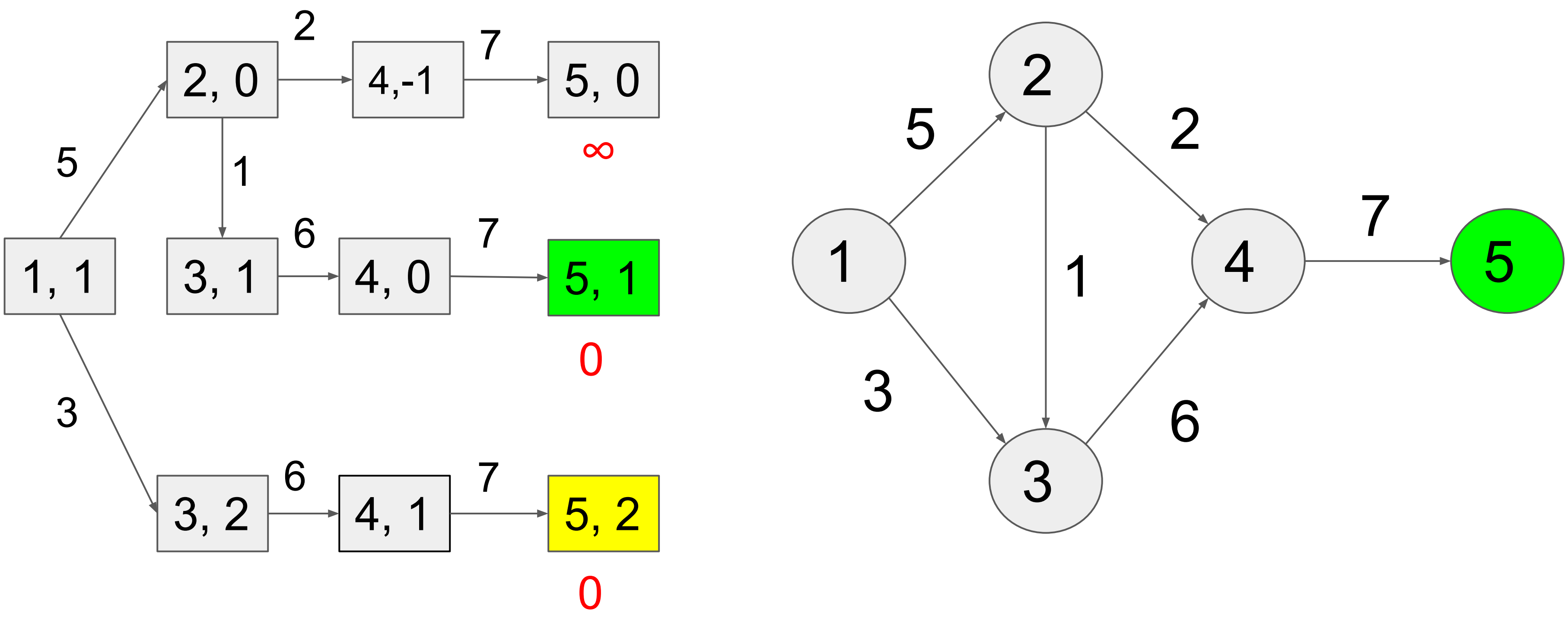


Simulation of Dynamic Programming

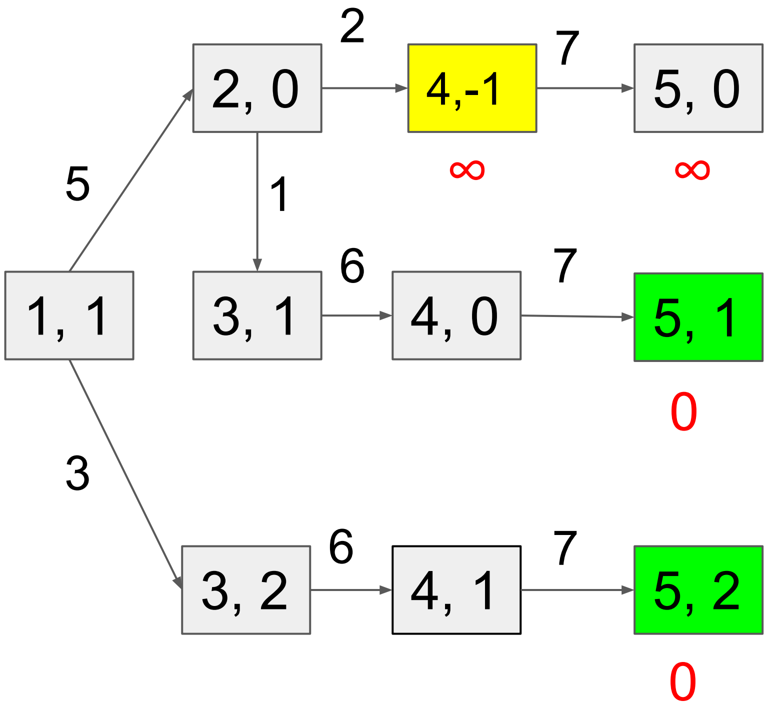


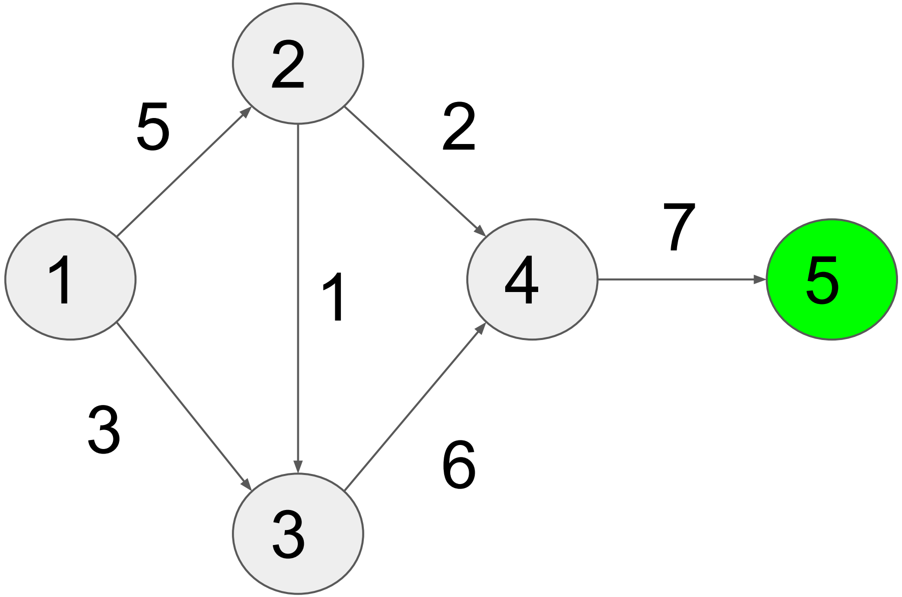


Simulation of Dynamic Programming

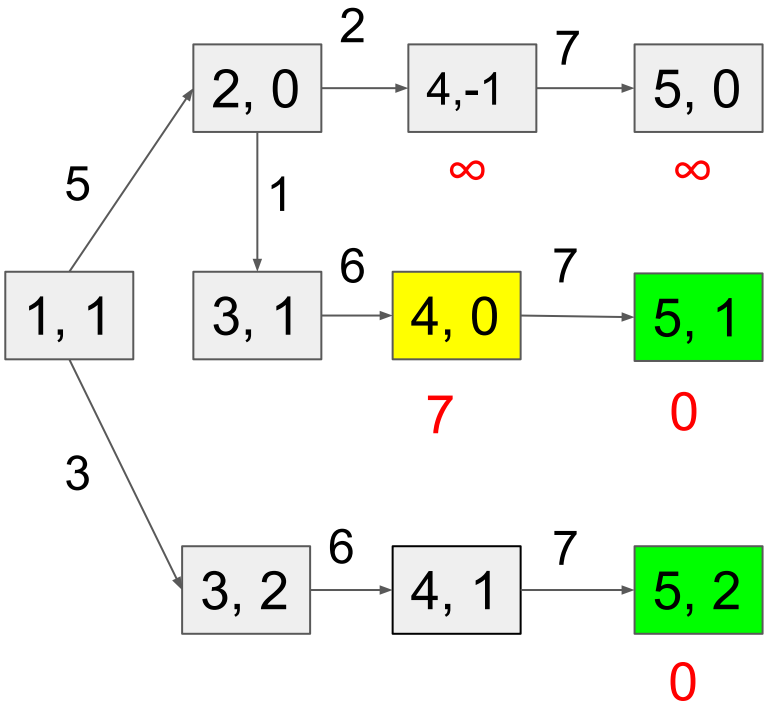


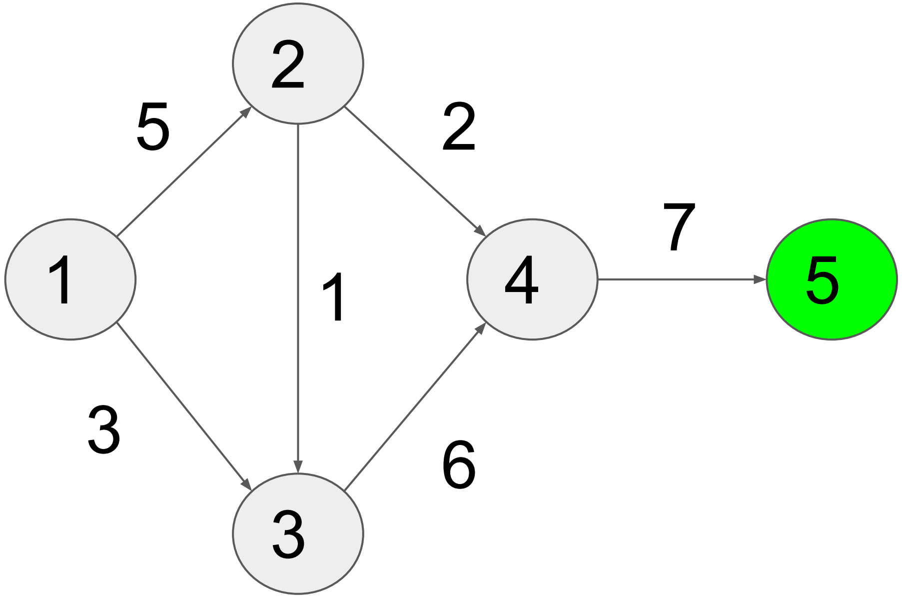
Simulation of Dynamic Programming



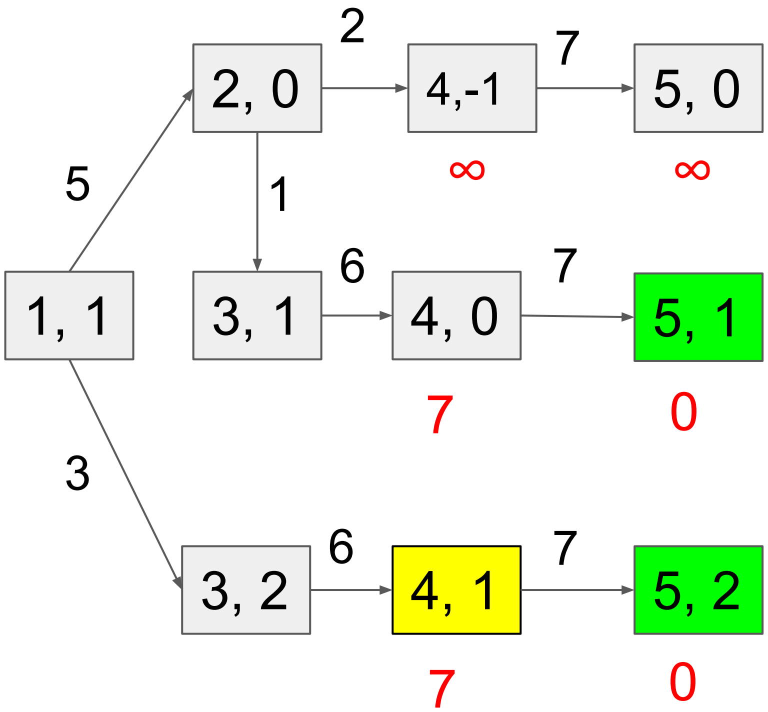


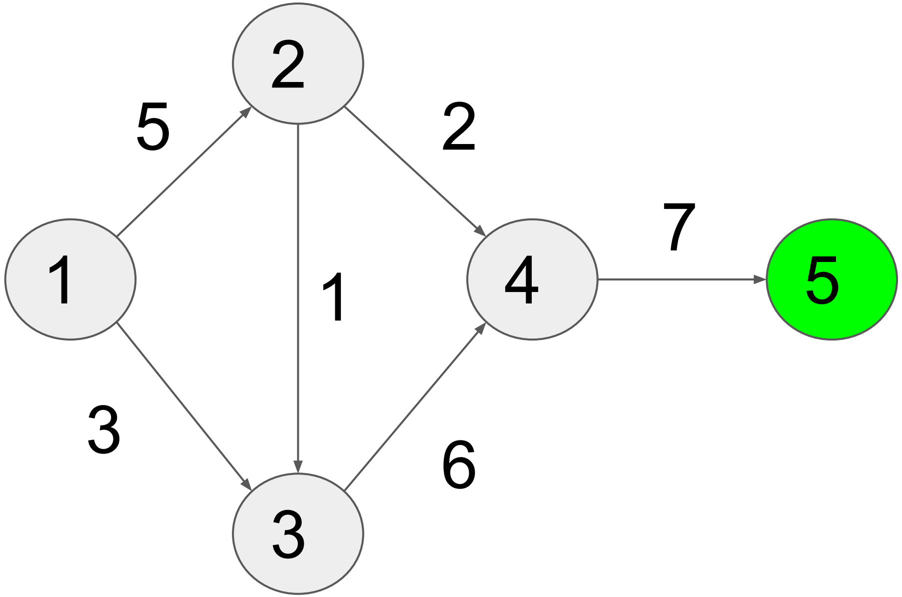
Simulation of Dynamic Programming



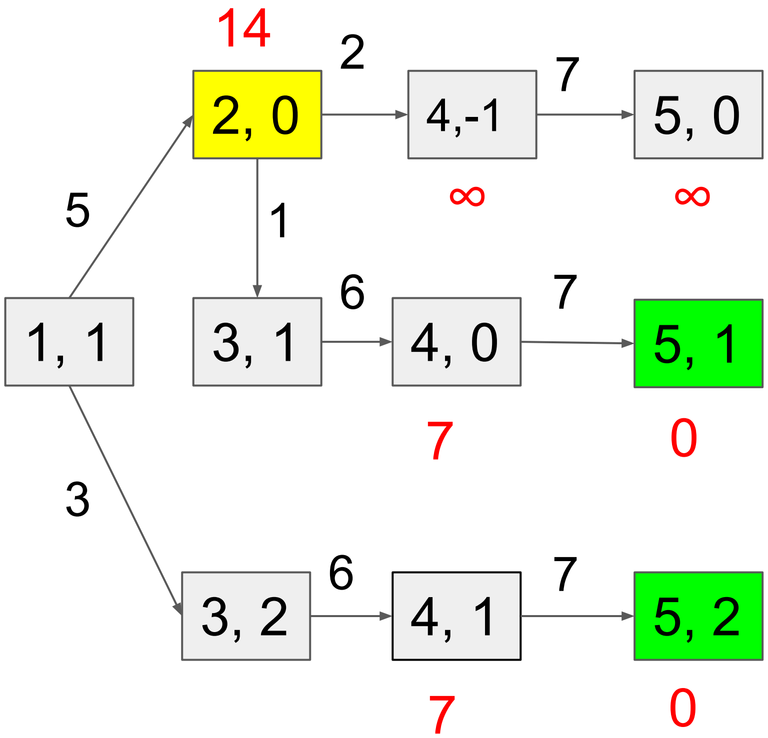


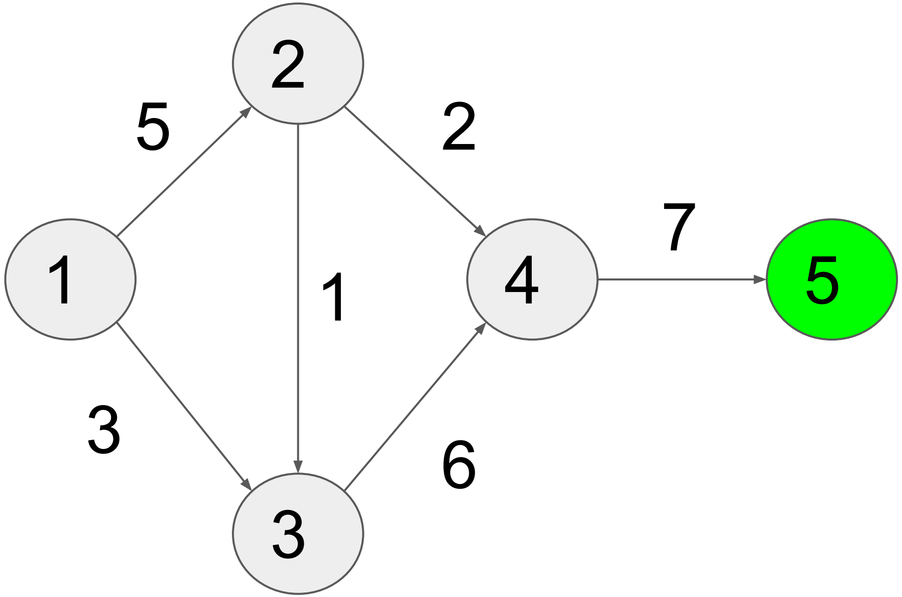
Simulation of Dynamic Programming



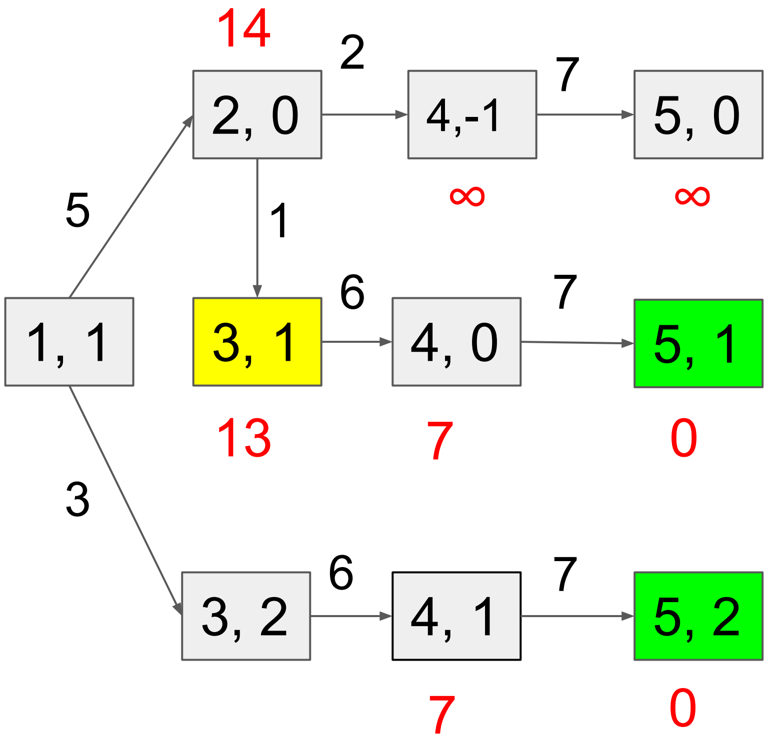


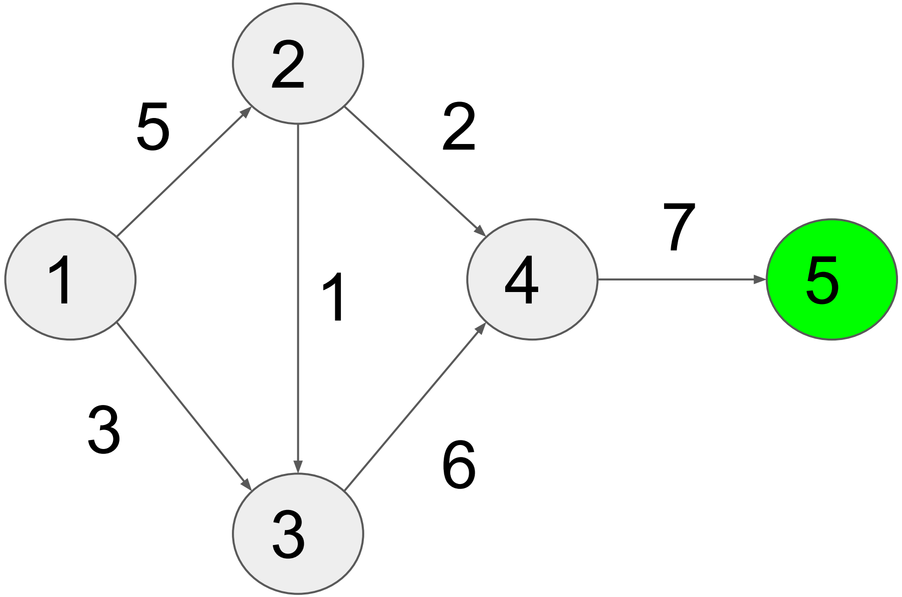
Simulation of Dynamic Programming



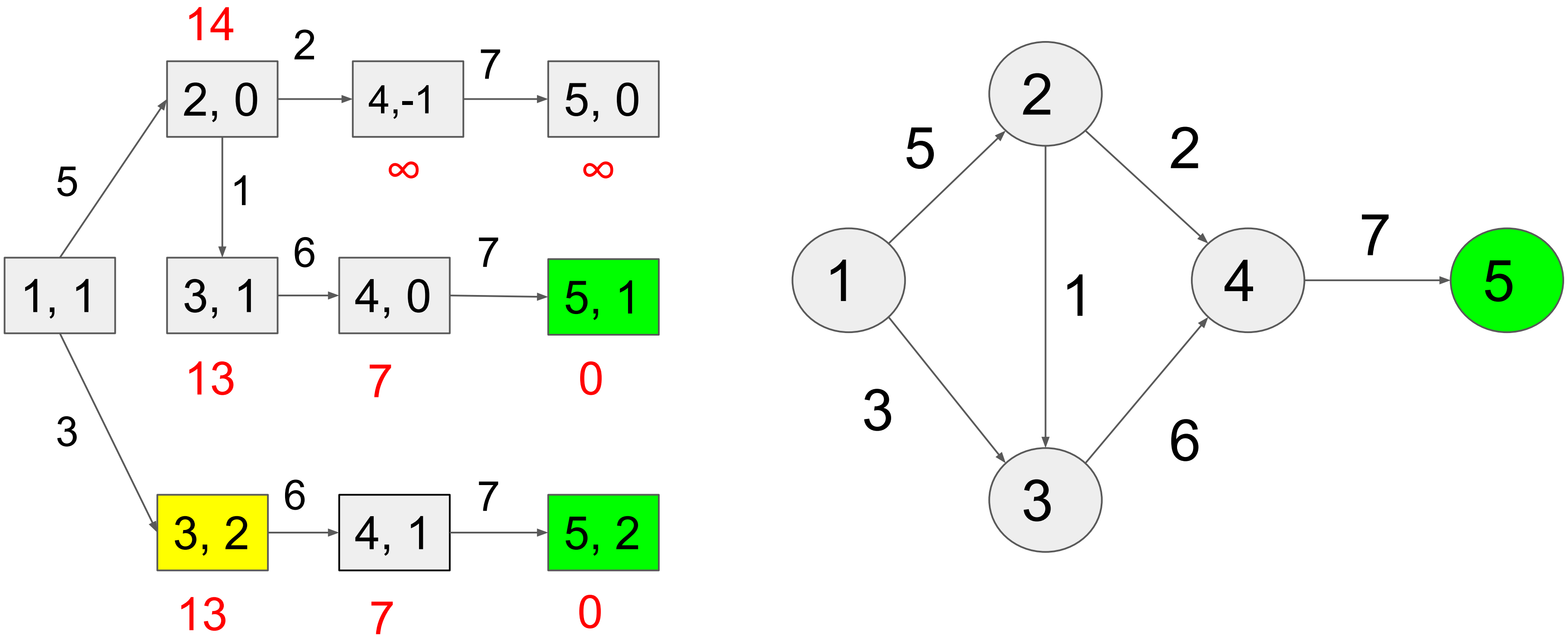


Simulation of Dynamic Programming

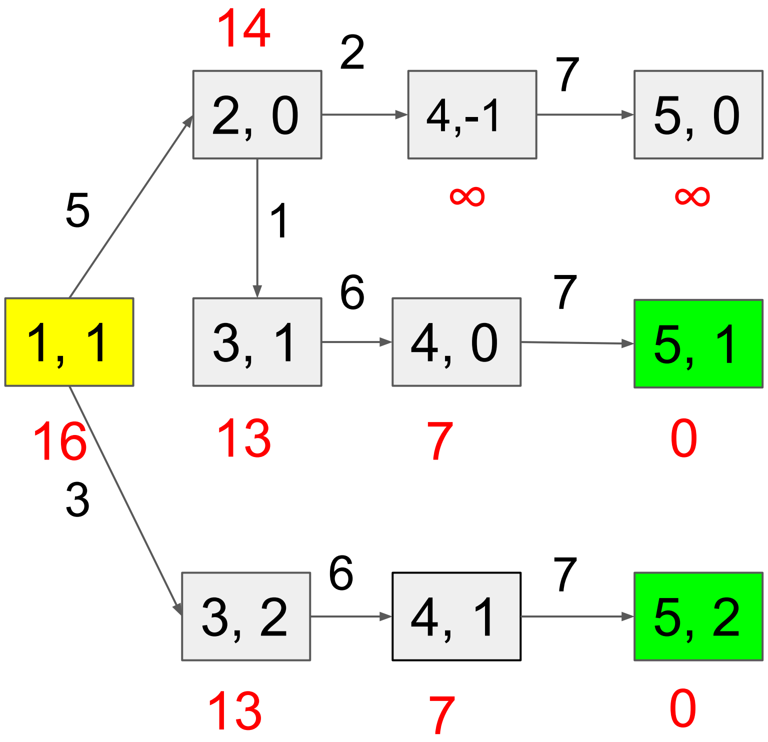


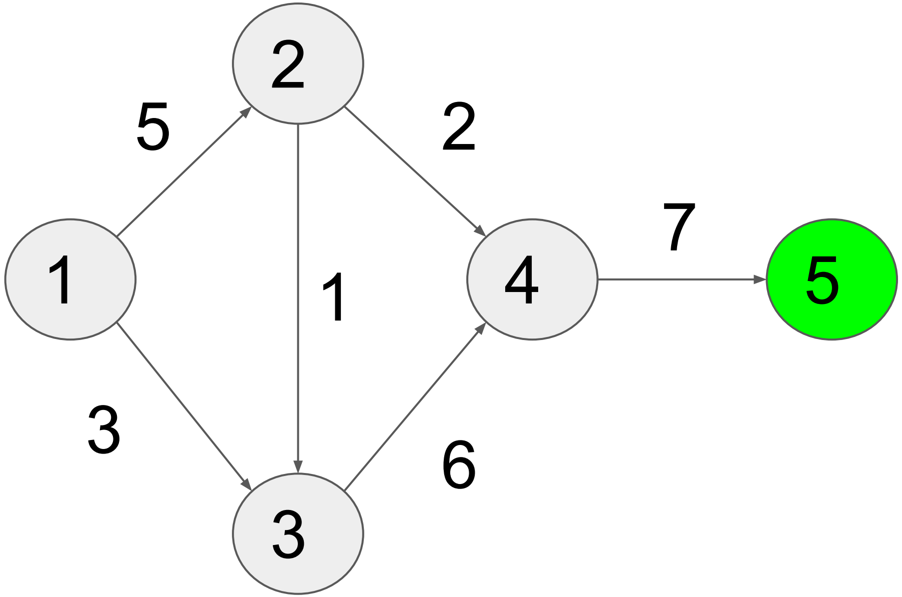


Simulation of Dynamic Programming



Simulation of Dynamic Programming





Simulation of Dynamic Programming

#odd - #even

city

	-1	0	1	2	3
1	-	-	16	-	-
2	-	14	-	-	-
3	-	-	13	13	-
4	∞	7	7	-	-
5	-	∞	0	0	-

Solving the Problem: Uniform Cost Search

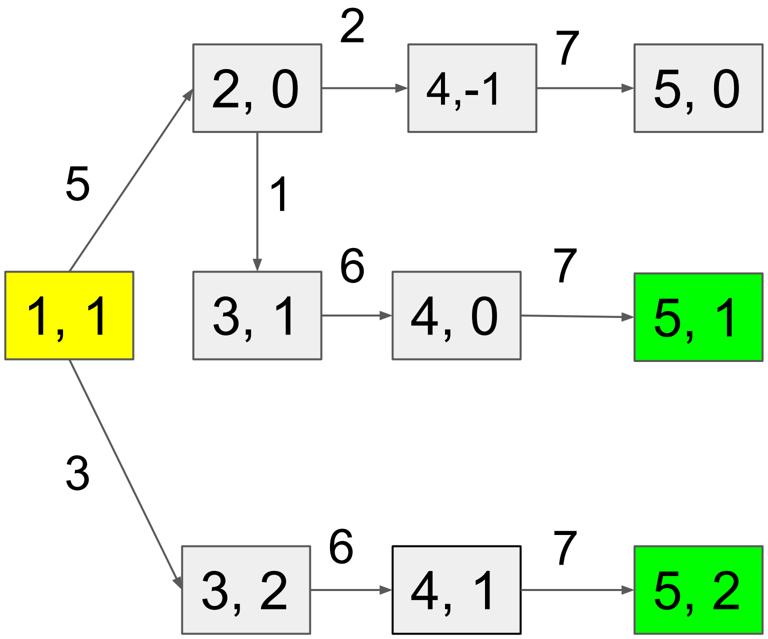
Algorithm: uniform cost search [Dijkstra, 1956]

Add s_{start} to **frontier** (priority queue)

Repeat until frontier is empty:

- Remove s with smallest priority p from frontier
- If IsGoal(s): return solution
- Add s to **explored**
- For each action $a \in$ Actions(s):
  - Get successor $s' \leftarrow$ Succ(s, a)
  - If s' already in explored: continue
  - Update **frontier** with s' and priority $p +$ Cost(s, a)

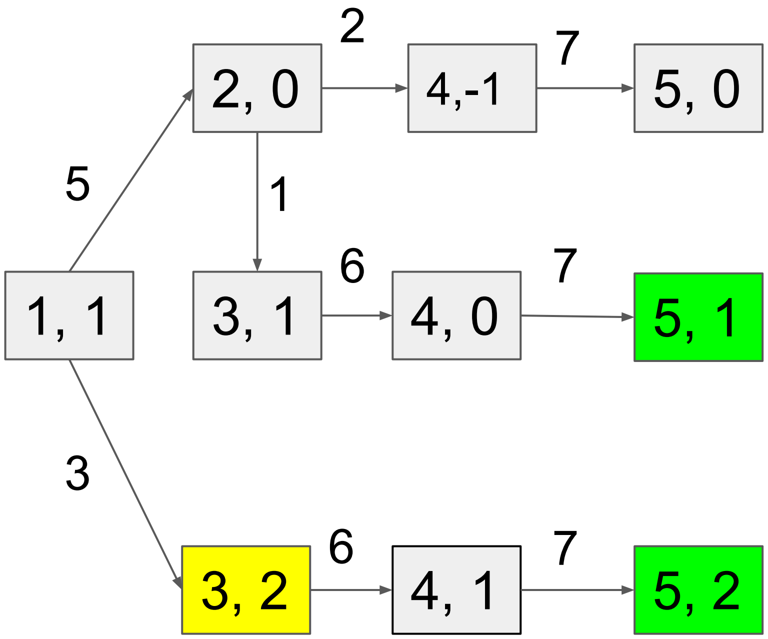
Simulation of UCS



Explored:
(1, 1) : 0

Frontier:
(2, 0) : 5
(3, 2) : 3

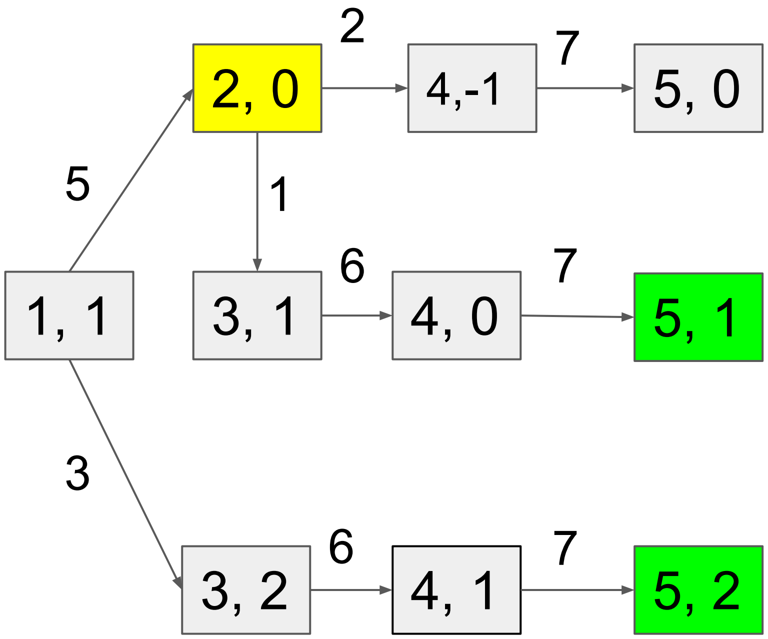
Simulation of UCS



Explored:
(1, 1) : 0
(3, 2) : 3

Frontier:
(2, 0) : 5
(4, 1) : 9

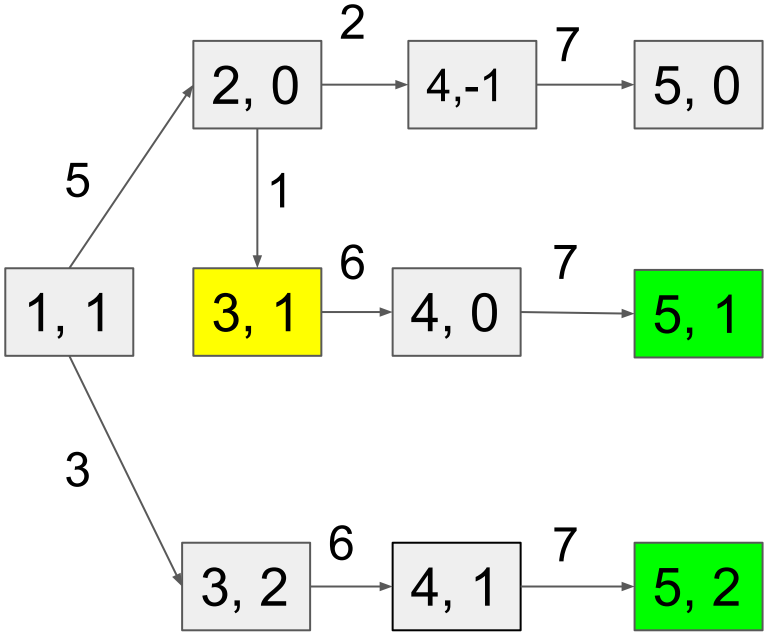
Simulation of UCS



Explored:
(1, 1) : 0
(3, 2) : 3
(2, 0) : 5

Frontier:
(4, 1) : 9
(4, -1) : 7
(3, 1) : 6

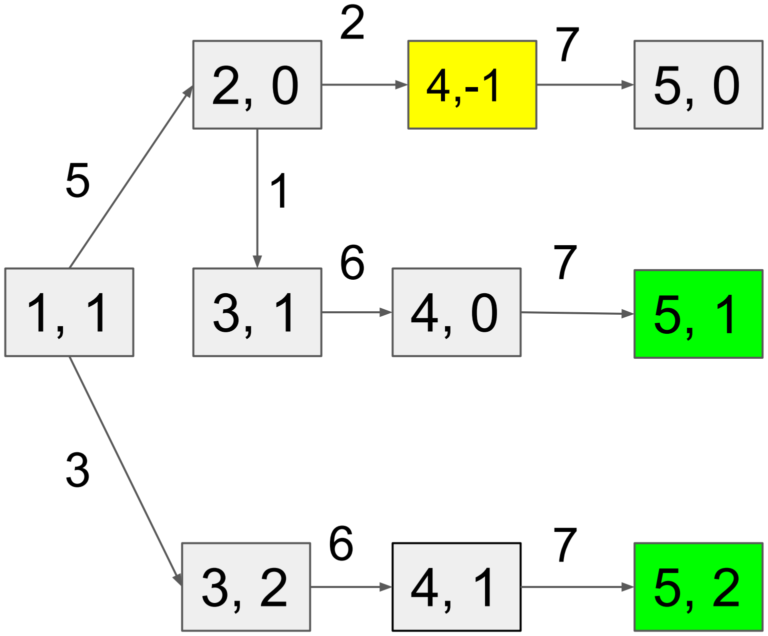
Simulation of UCS



Explored:
(1, 1) : 0
(3, 2) : 3
(2, 0) : 5
(3, 1): 6

Frontier:
(4, 1) : 9
(4, -1) : 7
(4, 0): 12

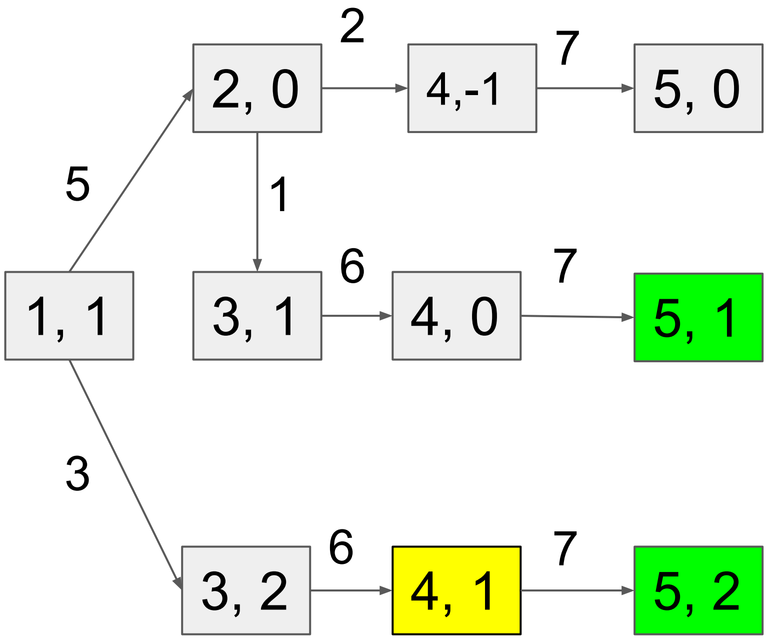
Simulation of UCS



Explored:
(1, 1) : 0
(3, 2) : 3
(2, 0) : 5
(3, 1) : 6
(4, -1) : 7

Frontier:
(4, 1) : 9
(4, 0) : 12
(5, 0) : 14

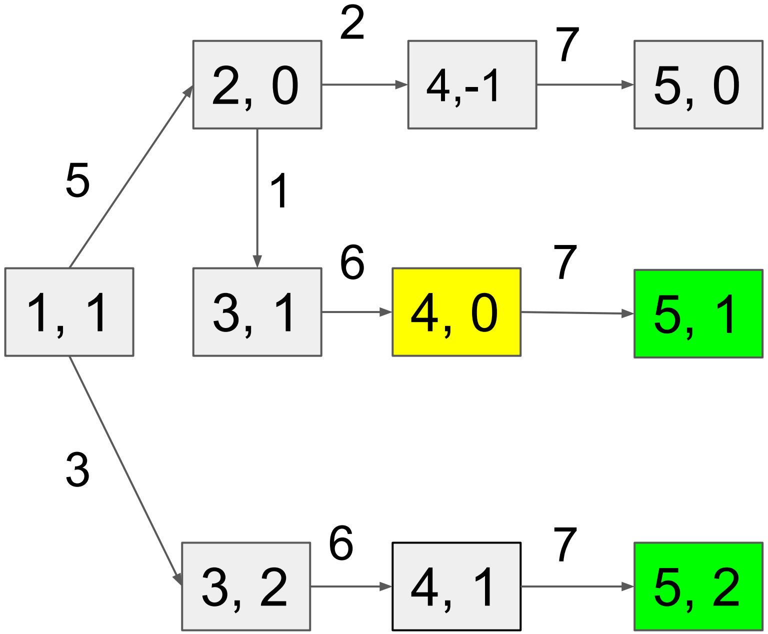
Simulation of UCS



Explored:
(1, 1) : 0
(3, 2) : 3
(2, 0) : 5
(3, 1) : 6
(4, -1) : 7
(4, 1) : 9

Frontier:
(4, 0) : 12
(5, 0) : 14
(5, 2) : 16

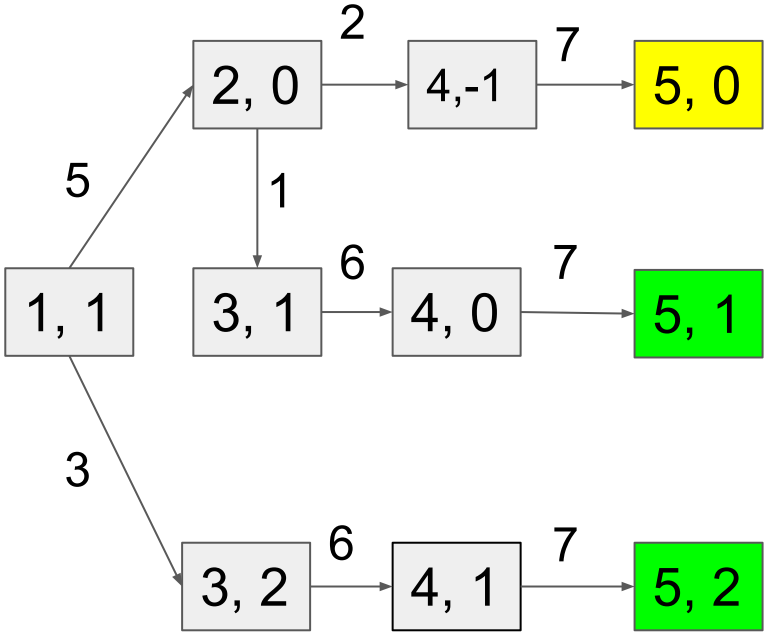
Simulation of UCS



Explored:
(1, 1) : 0
(3, 2) : 3
(2, 0) : 5
(3, 1) : 6
(4, -1) : 7
(4, 1) : 9
(4, 0) : 12

Frontier:
(5, 0) : 14
(5, 2) : 16
(5, 1) : 19

Simulation of UCS



Explored:

(1, 1) : 0

(3, 2) : 3

(2, 0) : 5

(3, 1) : 6

(4, -1) : 7

(4, 1) : 9

(4, 0) : 12

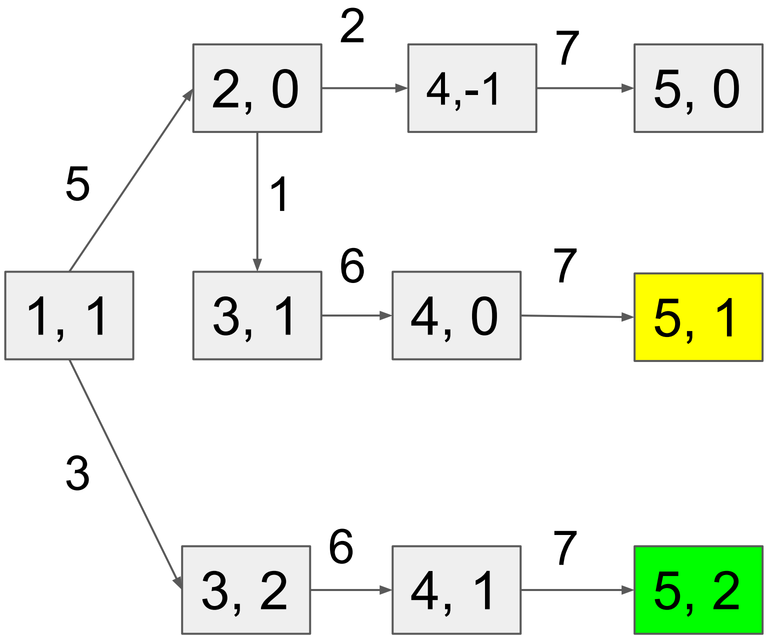
(5, 0) : 14

Frontier:

(5, 2) : 16

(5, 1) : 19

Simulation of UCS



Explored:
(1, 1) : 0
(3, 2) : 3
(2, 0) : 5
(3, 1) : 6
(4, -1) : 7
(4, 1) : 9
(4, 0) : 12
(5, 0) : 14
(5, 2) : 16

Frontier:
(5, 1) : 19

STOP!

Comparison between DP and UCS

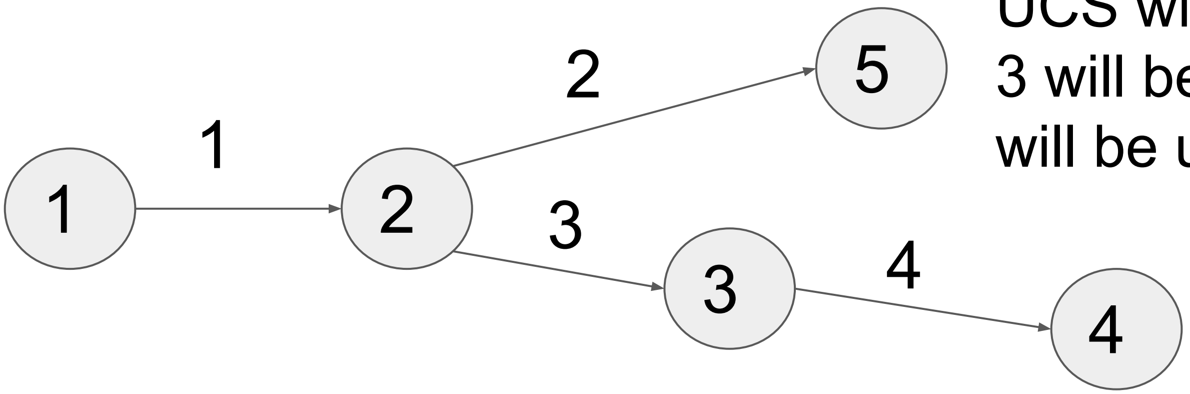
N total states, n of which are closer than goal state

Runtime of DP is O(N)

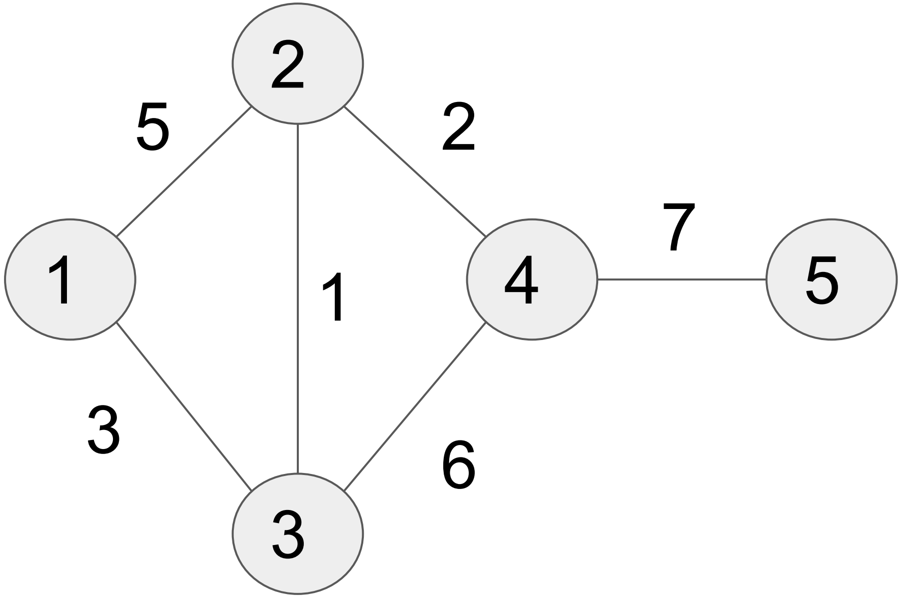
Runtime of UCS is O(n log n)

DP explores O(N) states.

UCS will explore {1, 2, 5} only. 3 will be in the frontier and 4 will be unexplored.



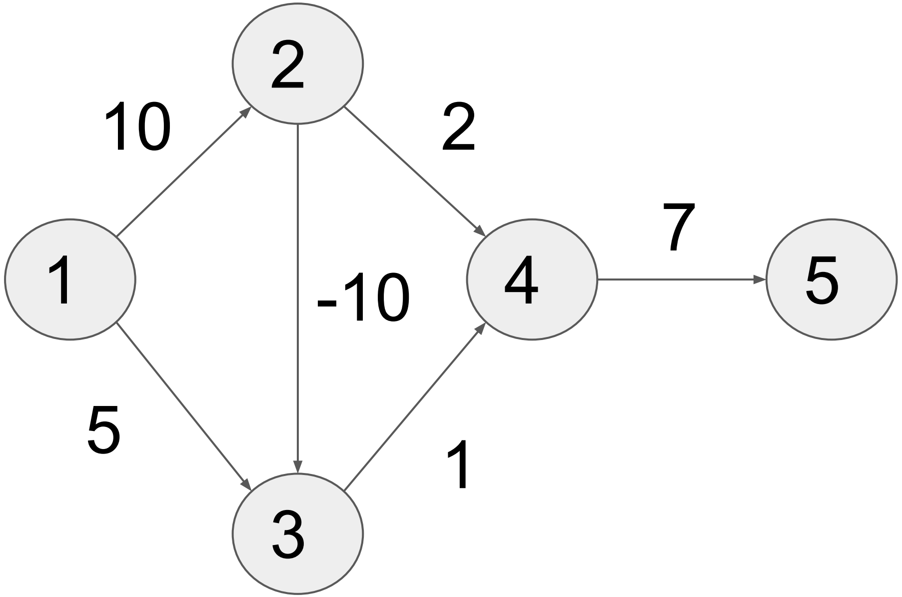
DP cannot handle cycles



Shortest path is [1, 3, 2, 5] with cost 13.

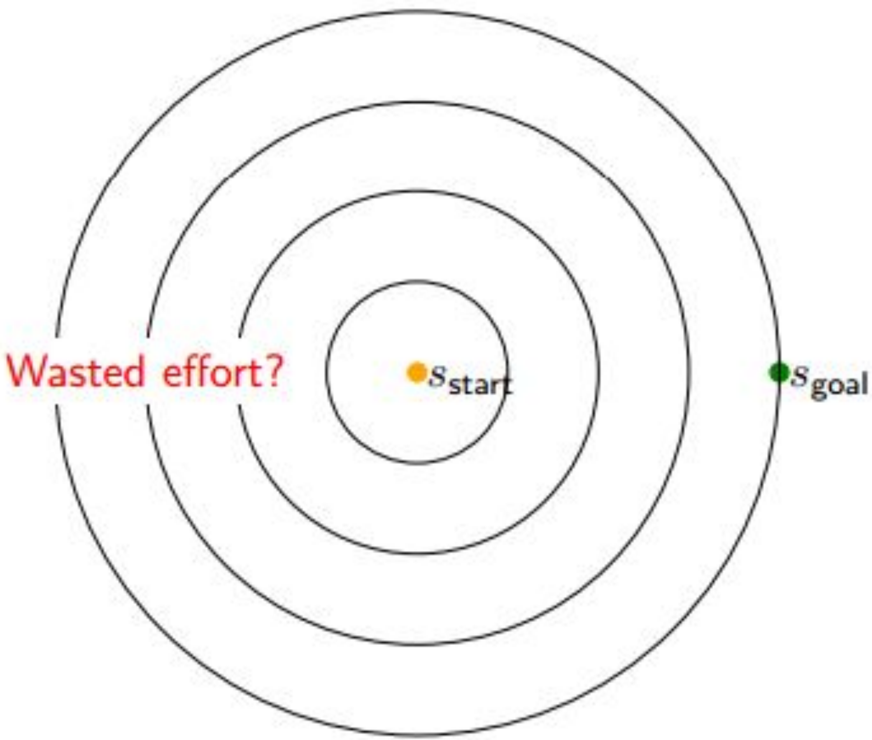
Hard to define subproblems in undirected graphs

UCS cannot handle negative edge weights



Best path is [1,2,3,4,5] with cost of 8, but UCS will output [1,3,4,5] with cost of 13 because 3 is set as ‘explored’ before 2.

Improve UCS: A* Search



Recap of A* Search

- Modify the cost of edges and run UCS on the new graph
  - Cost'(s, a) = Cost(s, a) + h(Succ(s, a)) - h(s)
- h(s) is a heuristic that is our estimate of FutureCost(s)
- If h(s) is consistent then the modified edge weights will return min cost path
- One can find a good consistent h by performing relaxation
- If c is min cost on original graph, c' is min cost on modified graph, then c' = c + h(s_goal) - h(s_start)

Relaxation

A good way to come up with a reasonable heuristic is to solve an easier (less constrained) version of the problem

For example, we can remove the constraint that we visit more odd cities than even cities.

h(s) = h((i, d)) = length of shortest path from city i to city N

Note on Relaxation

The main point of relaxation is to attain a problem that **can be solved more efficiently**.

In our case, the modified shortest path problem has O(N) states instead of O(N^2) can thus can be solved more efficiently

Checking consistency

- $Cost(s, a) + h(Succ(s, a)) - h(s) \geq 0$ (Triangle Inequality)
  - Suppose $s = (i, d)$ and $Succ(s, a) = (j, d')$
  - Note that $h((i, d)) - h((j, d')) \leq c(i, j) = Cost(s, a)$
- $h((N, d)) = 0$

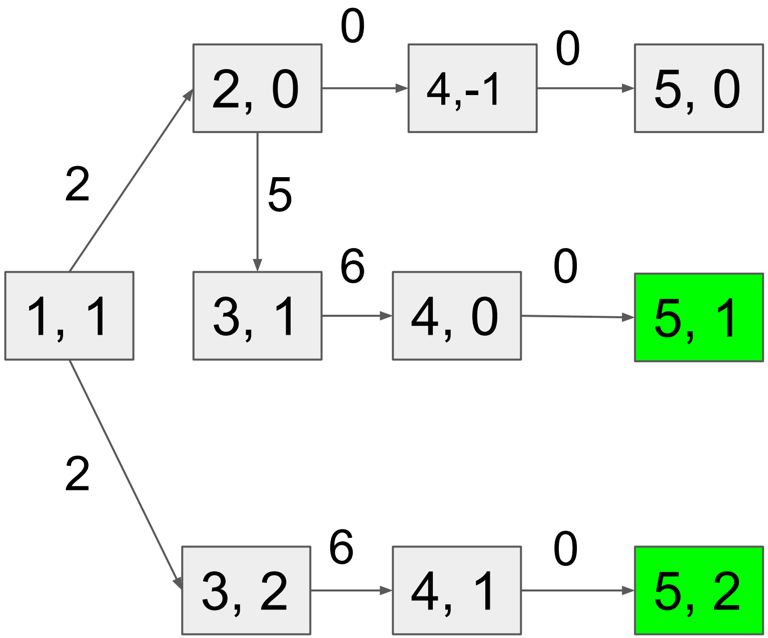
How to compute h?

We can reverse the direction of all edges, and then perform UCS starting from city N, and our goal state is city 1.

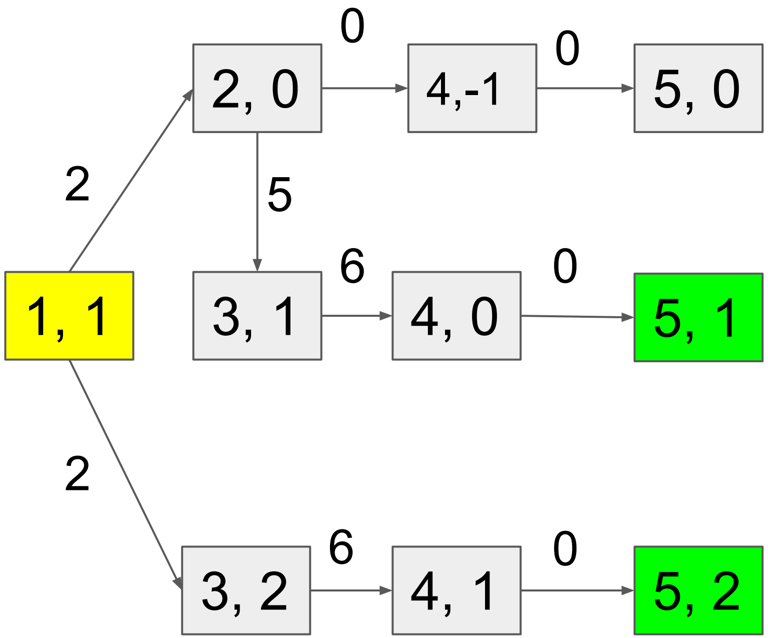
This takes O(n log n) time, where n is the number of states whose distance to city N is no farther than the distance of city 1 to city N

city	1	2	3	4	5
h	14	9	13	7	0

Modified State Graph



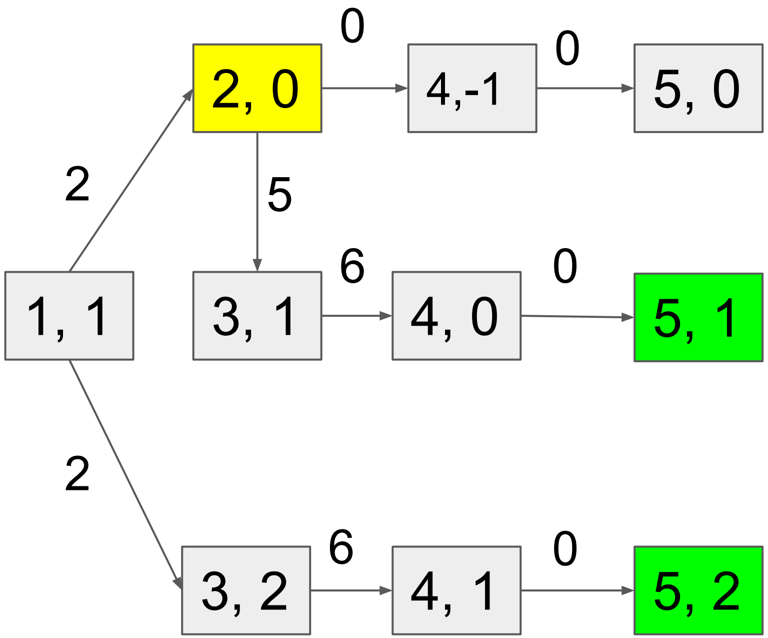
Simulation of UCS (A*)



Explored:
(1, 1) : 0

Frontier:
(2, 0) : 0
(3, 2) : 2

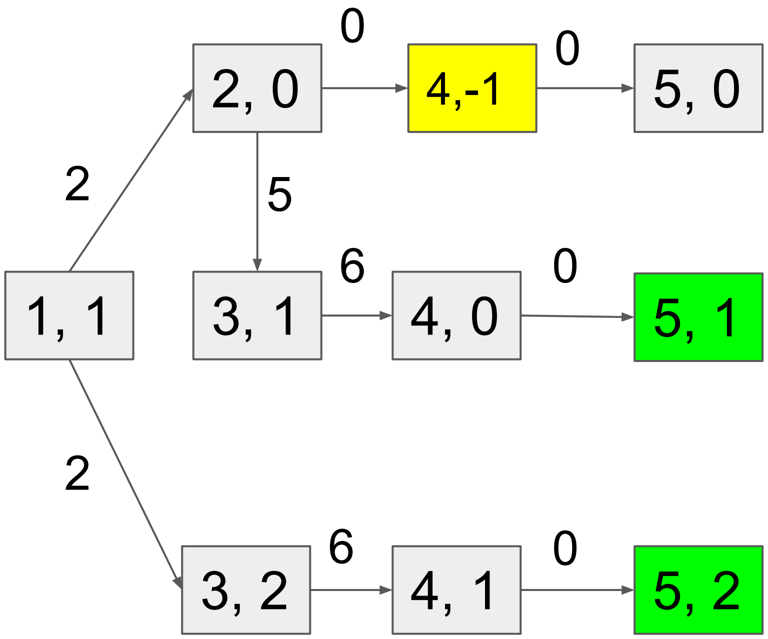
Simulation of UCS (A*)



Explored:
(1, 1) : 0
(2, 0) : 0

Frontier:
(3, 2) : 2
(3, 1) : 3
(4, -1) : 0

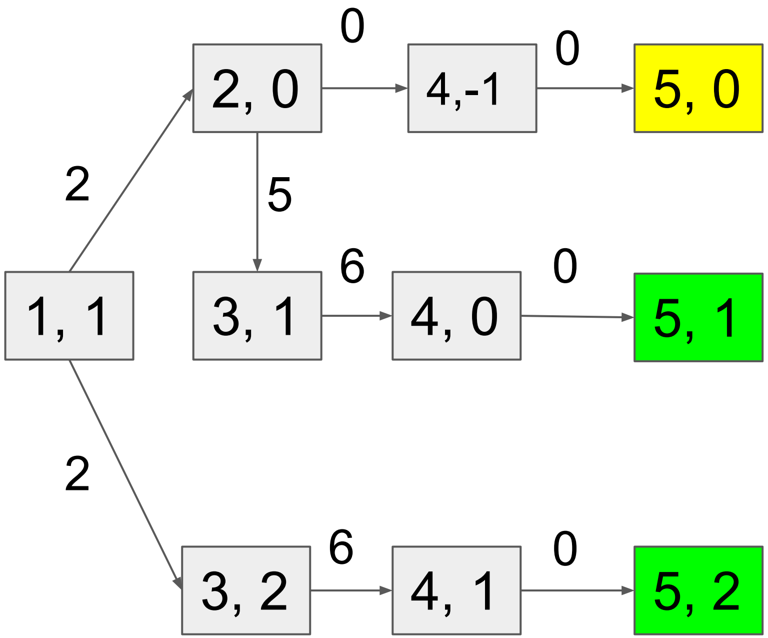
Simulation of UCS (A*)



Explored:
(1, 1) : 0
(2, 0) : 0
(4, -1) : 0

Frontier:
(3, 2) : 2
(3, 1) : 3
(5, 0) : 0

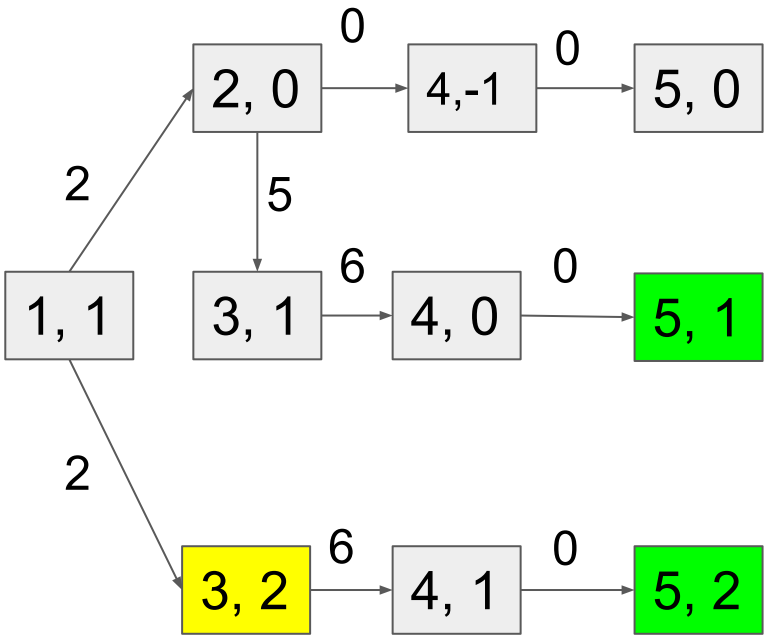
Simulation of UCS (A*)



Explored:
(1, 1) : 0
(2, 0) : 0
(4, -1) : 0
(5, 0) : 0

Frontier:
(3, 2) : 2
(3, 1) : 3

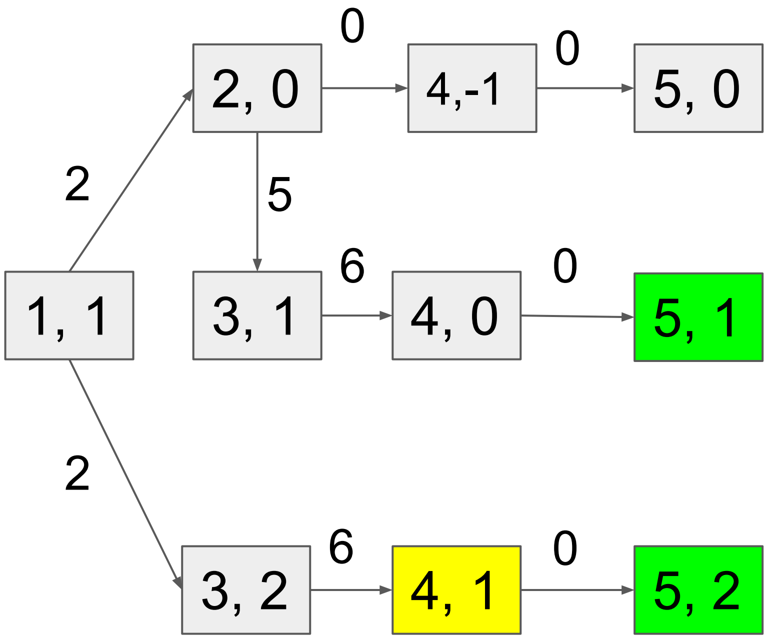
Simulation of UCS (A*)



Explored:
(1, 1) : 0
(2, 0) : 0
(4, -1) : 0
(5, 0) : 0
(3, 2) : 2

Frontier:
(3, 1) : 3
(4, 1) : 2

Simulation of UCS (A*)



Explored:

(1, 1) : 0

(2, 0) : 0

(4, -1) : 0

(5, 0) : 0

(3, 2) : 2

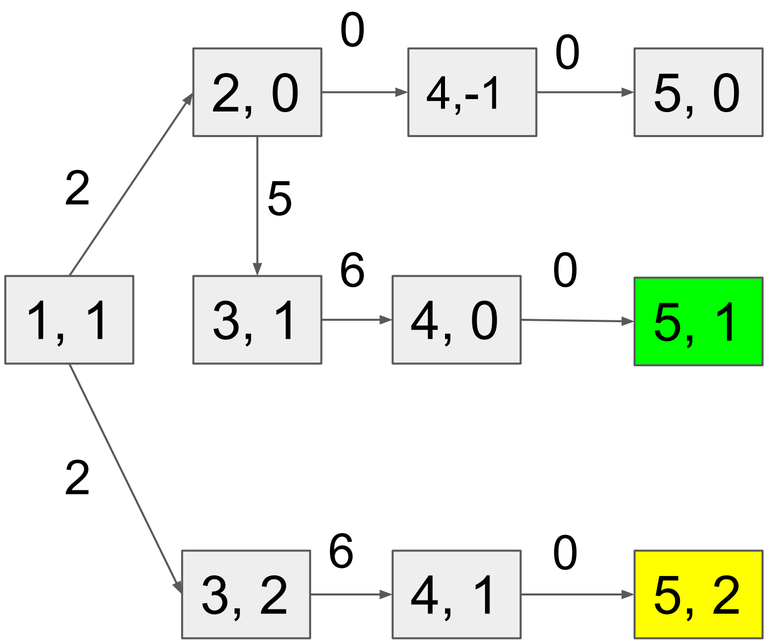
(4, 1): 2

Frontier:

(3, 1) : 3

(5, 2) : 2

Simulation of UCS (A*)



Explored:
(1, 1) : 0
(2, 0) : 0
(4, -1) : 0
(5, 0) : 0
(3, 2) : 2
(4, 1) : 2
(5, 2) : 2

Frontier:
(3, 1) : 3

STOP!

Actual Cost is 2 + h(1) = 2 + 14 = 16

Comparison of States visited

UCS

Explored:
(1, 1) : 0
(3, 2) : 3
(2, 0) : 5
(3, 1) : 6
(4, -1) : 7
(4, 1) : 9
(4, 0) : 12
(5, 0) : 14
(5, 2) : 16

Frontier:
(5, 1) : 19

UCS(A*)

Explored:
(1, 1) : 0
(2, 0) : 0
(4, -1) : 0
(5, 0) : 0
(3, 2) : 2
(4, 1) : 2
(5, 2) : 2

Frontier:
(3, 1) : 3

Summary

- States Representation/Modelling
  - make state representation as compact as possible, remove unneccesary information
- DP
  - underlying graph cannot have cycles
  - visit all reachable states, but no log overhead
- UCS
  - actions cannot have negative cost
  - visit only a subset of states, log overhead
- A*
  - ensure that relaxed problem can be solved more efficiently