CS221 Section 3: Search

*DP, UCS, A**

What are the “ingredients” for a well-defined search problem?

Definition: search problem

- s_{start}: starting state
- $\text{Actions}(s)$: possible actions
- $\text{Cost}(s, a)$: action cost
- $\text{Succ}(s, a)$: successor
- $\text{Is }\mathbf{End}(s)$: found solution?

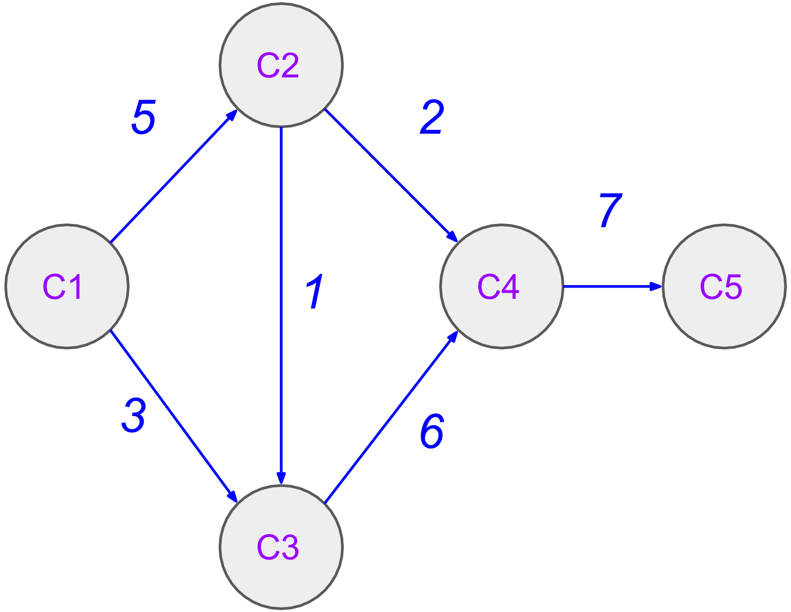
Section Problem

There exists ***N* cities**, labeled from *1* to *N*.

There are one-way 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** (each road is one-directional).

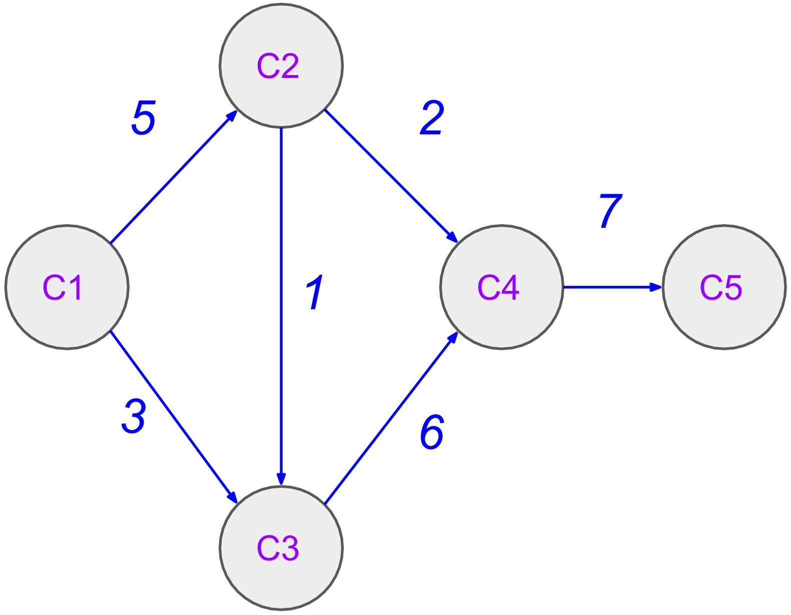
From city *1*, we want to travel to city *N*. What is the **shortest time** required to make this trip, given the **constraint** that we should visit **more odd-labeled cities than even labeled cities**?

Example



1. What is the **shortest path** (without constraint)?
2. What is the **shortest path under the given constraint** (visit more odd than even cities)?

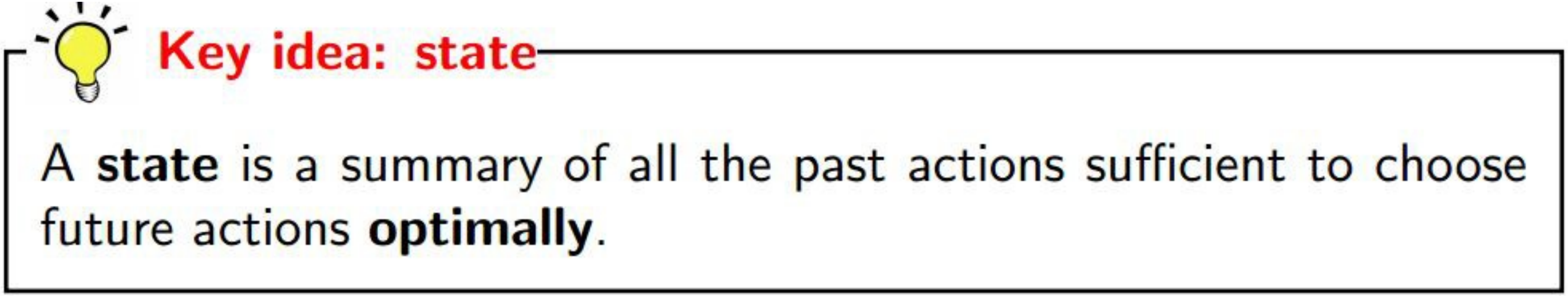
Example



[C1, C2, C4, C5] has cost 14 but visits equal number of odd and even cities.

Best path is [C1, C3, C4, C5] with cost 16.

State Representation



Key idea: state

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

How would you represent a state for this problem?

🤔

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: **$O(N^3)$**

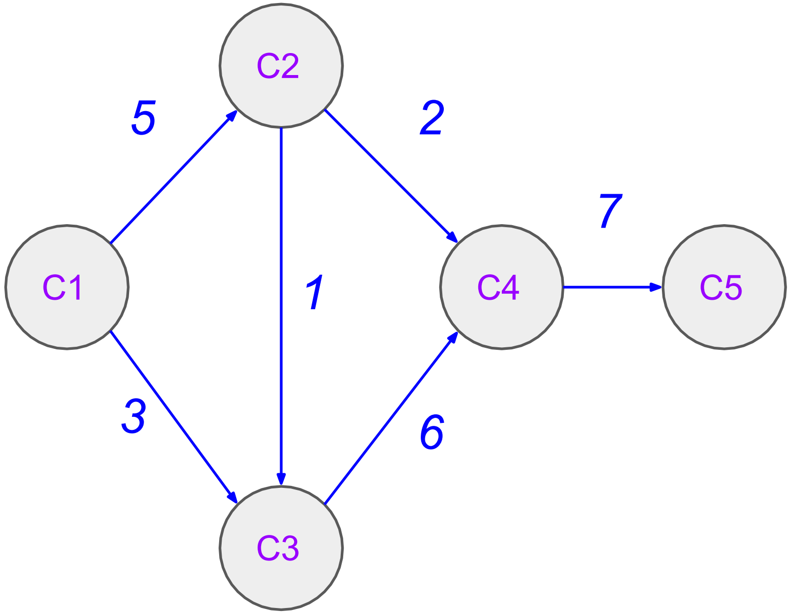
Can We Do Better?

Check if all the information is really required

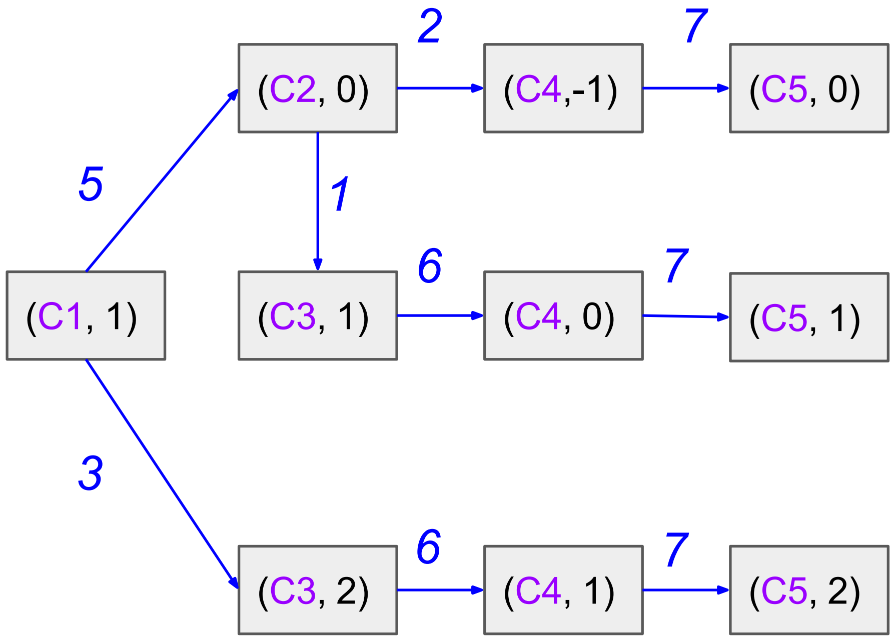
Instead of storing **#odd** and **#even**, we can store **#odd - #even** directly; this still allows us to check whether **#odd** - **#even** > 0 at (N, **#odd**, **#even**)

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

Original Graph



State Graph



State $s = (i, d)$ (current city, #odd-#even)

Precise Formulation of Problem

State $s := (i, d)$ (current city, #odd-#even)

$E := \{(i, j) \mid \exists \text{ road from i to j}\}$

$\text{Actions}(s) := \{move(j) \mid (i, j) \in E\}$

$\text{Cost}(s, move(j)) := c(i, j)$

$$\text{Succ}(s, a) := \begin{cases} (j, d+1) & j \text{ odd} \\ (j, d-1) & j \text{ even} \end{cases}$$

$\text{Start} := (1, 1)$

$\text{isEnd}(s) := i = N \text{ and } d > 0$

Which algorithms can you use to solve this problem?
Any pros and cons?

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

➢ *Which one works for our problem?*

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

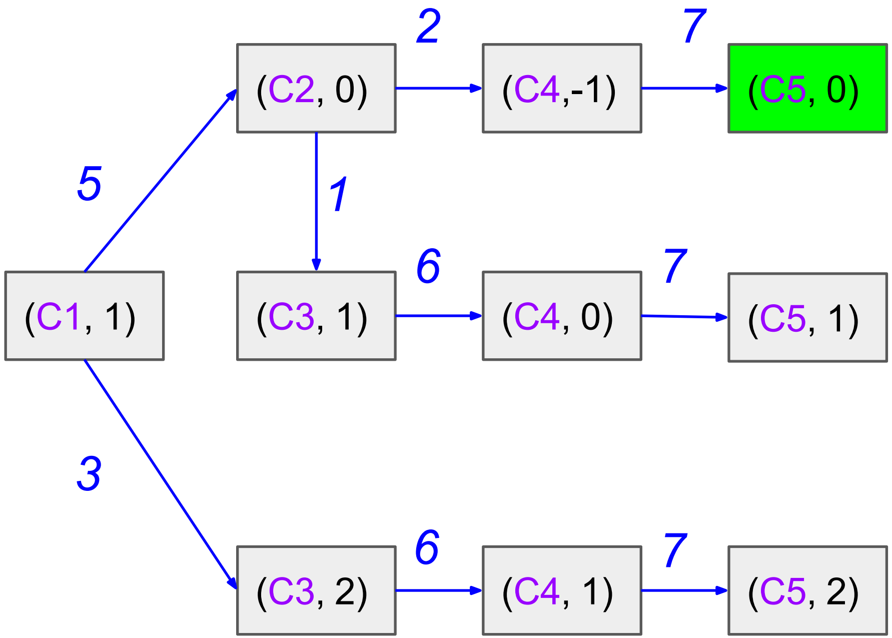
Since we have a **DAG** and all edges are positive, both work!

Solving the Problem: Dynamic Programming

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

If *s* has no successors, we set it as *undefined*

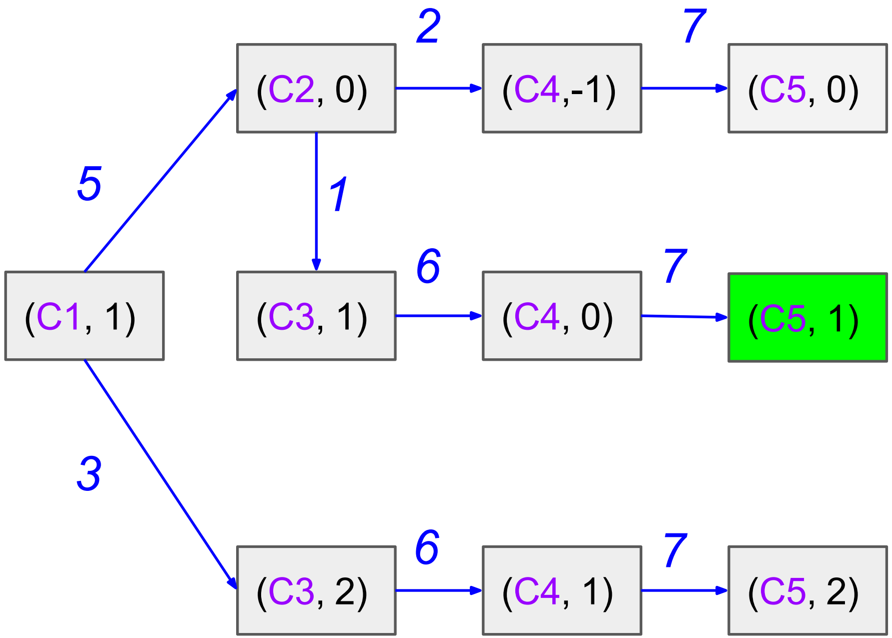
Simulation of DP



State $s = (i, d)$ (current city, #odd-#even)

		#odd - #even				
		-1	0	1	2	3
city	C1	-	-	-	-	-
	C2	-	-	-	-	-
	C3	-	-	-	-	-
	C4	-	-	-	-	-
	C5	-	?	-	-	-

Simulation of DP



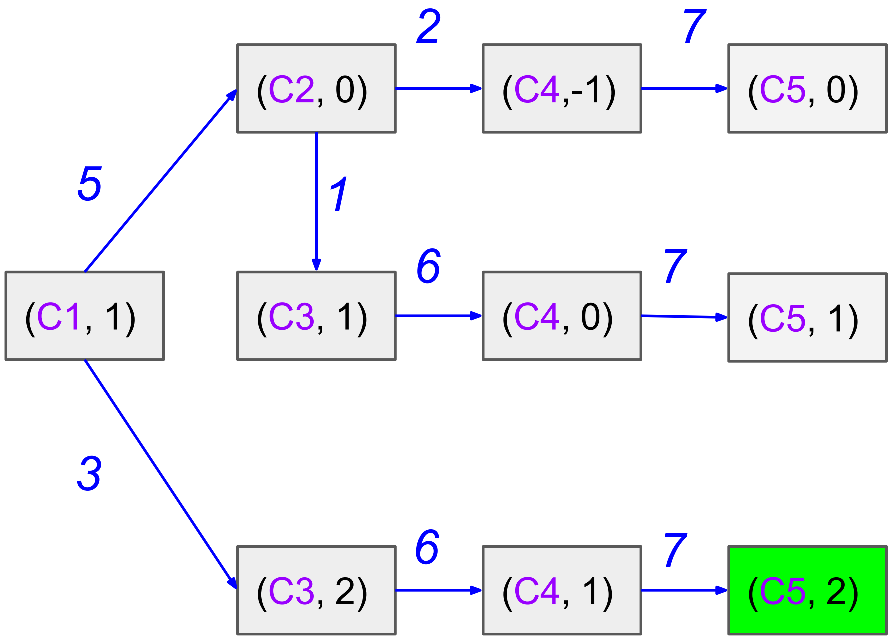
State $s = (i, d)$ (current city, #odd-#even)

#odd - #even

city

	-1	0	1	2	3
C1	-	-	-	-	-
C2	-	-	-	-	-
C3	-	-	-	-	-
C4	-	-	-	-	-
C5	-	?	0	-	-

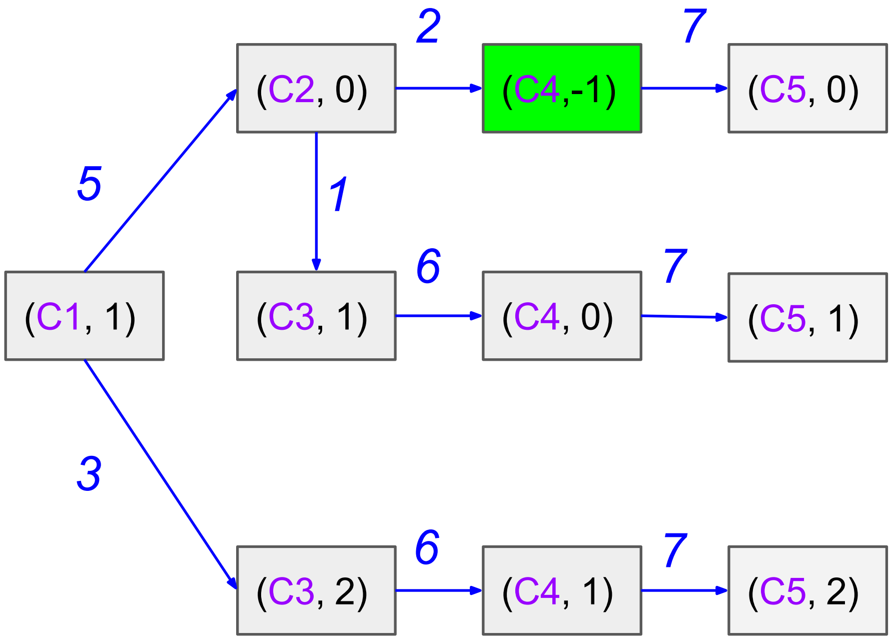
Simulation of DP



State $s = (i, d)$ (current city, #odd-#even)

		#odd - #even				
		-1	0	1	2	3
city	C1	-	-	-	-	-
	C2	-	-	-	-	-
	C3	-	-	-	-	-
	C4	-	-	-	-	-
	C5	-	?	0	0	-

Simulation of DP

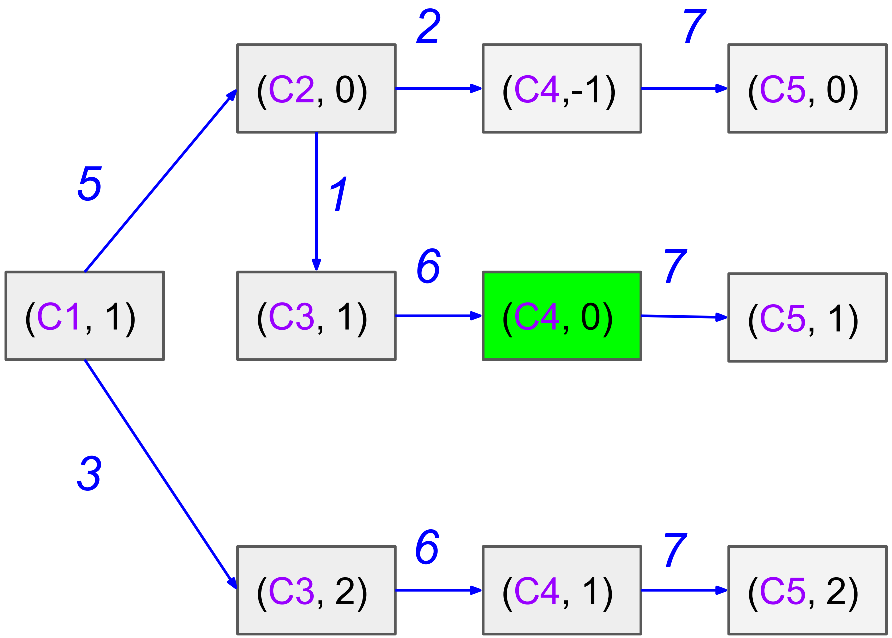


State $s = (i, d)$ (current city, #odd-#even)

#odd - #even

city	-1	0	1	2	3
C1	-	-	-	-	-
C2	-	-	-	-	-
C3	-	-	-	-	-
C4	?	-	-	-	-
C5	-	?	0	0	-

Simulation of DP



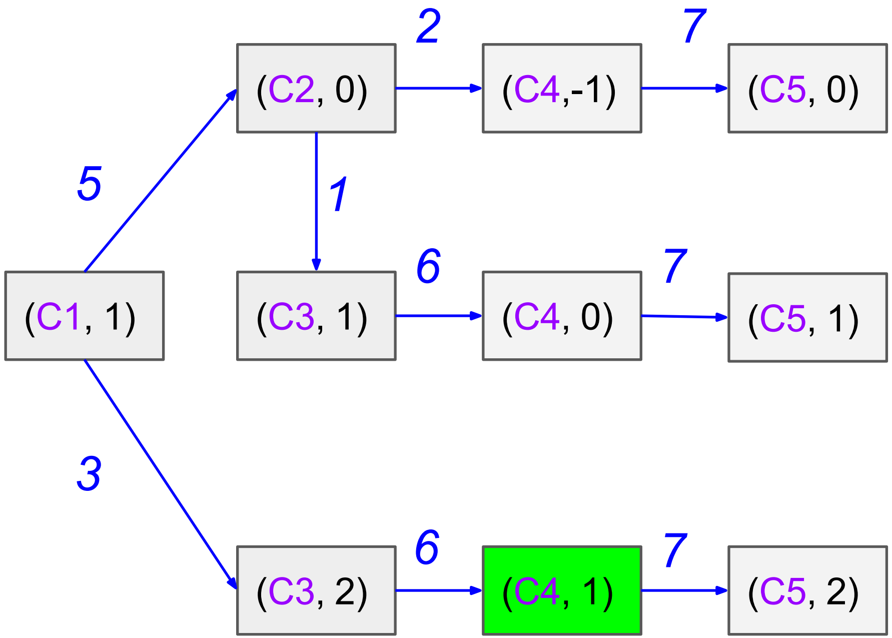
State $s = (i, d)$ (current city, #odd-#even)

#odd - #even

city

	-1	0	1	2	3
C1	-	-	-	-	-
C2	-	-	-	-	-
C3	-	-	-	-	-
C4	?	7	-	-	-
C5	-	?	0	0	-

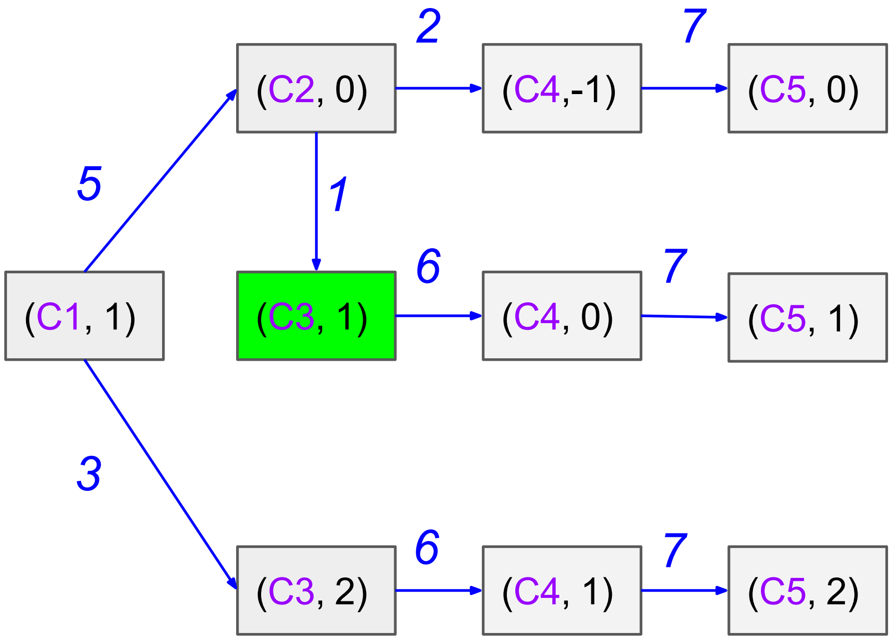
Simulation of DP



State $s = (i, d)$ (current city, #odd-#even)

		#odd - #even				
		-1	0	1	2	3
city	C1	-	-	-	-	-
	C2	-	-	-	-	-
	C3	-	-	-	-	-
	C4	?	7	7	-	-
	C5	-	?	0	0	-

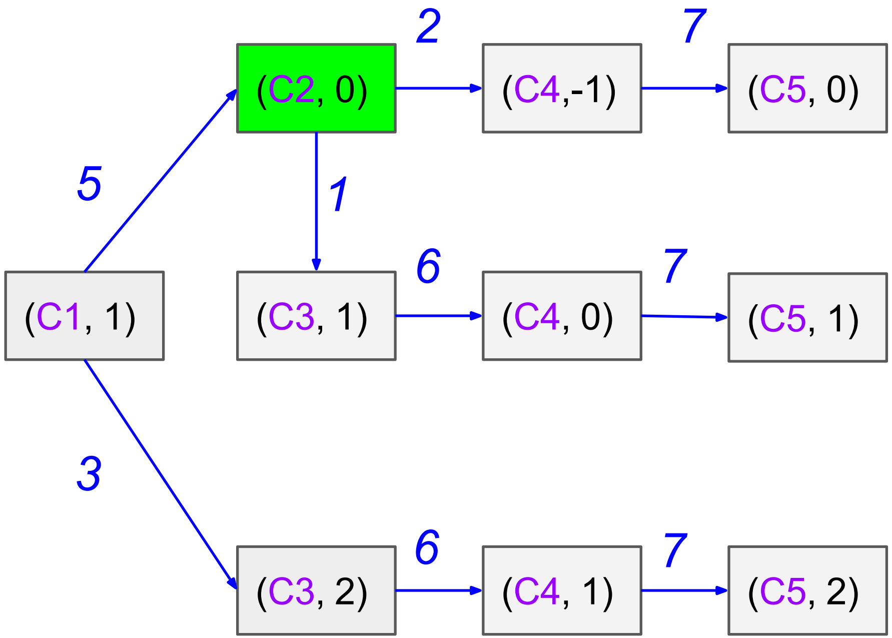
Simulation of DP



State $s = (i, d)$ (current city, #odd-#even)

	#odd - #even				
city	-1	0	1	2	3
C1	-	-	-	-	-
C2	-	-	-	-	-
C3	-	-	13	-	-
C4	?	7	7	-	-
C5	-	?	0	0	-

Simulation of DP



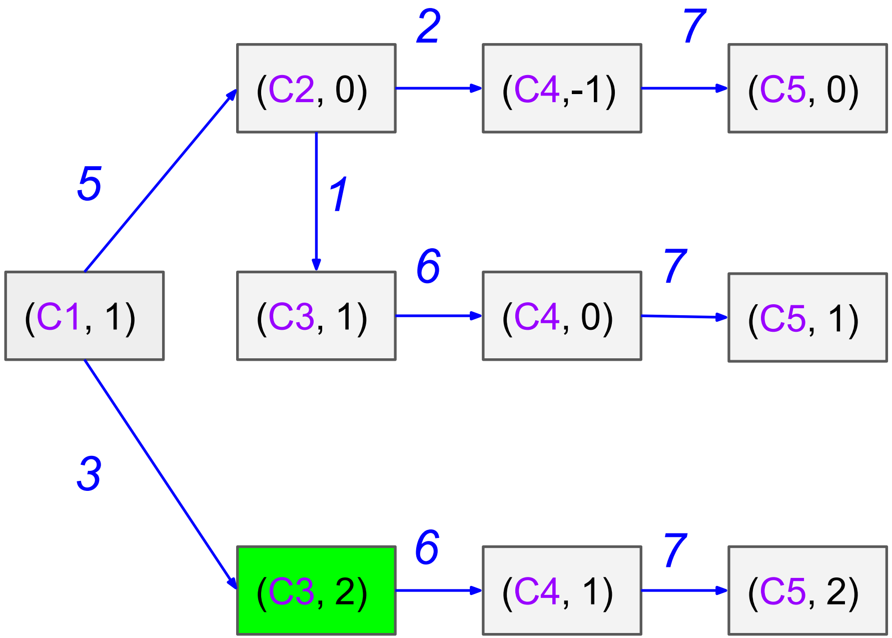
State $s = (i, d)$ (current city, #odd-#even)

#odd - #even

city

	-1	0	1	2	3
C1	-	-	-	-	-
C2	-	14	-	-	-
C3	-	-	13	-	-
C4	?	7	7	-	-
C5	-	?	0	0	-

Simulation of DP



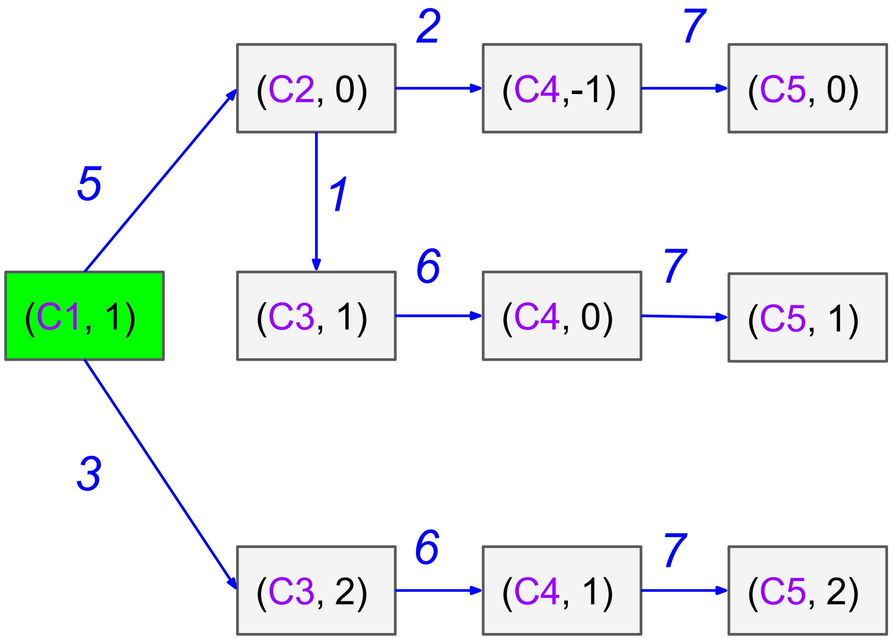
State $s = (i, d)$ (current city, #odd-#even)

#odd - #even

city

	-1	0	1	2	3
C1	-	-	-	-	-
C2	-	14	-	-	-
C3	-	-	13	13	-
C4	?	7	7	-	-
C5	-	?	0	0	-

Simulation of DP

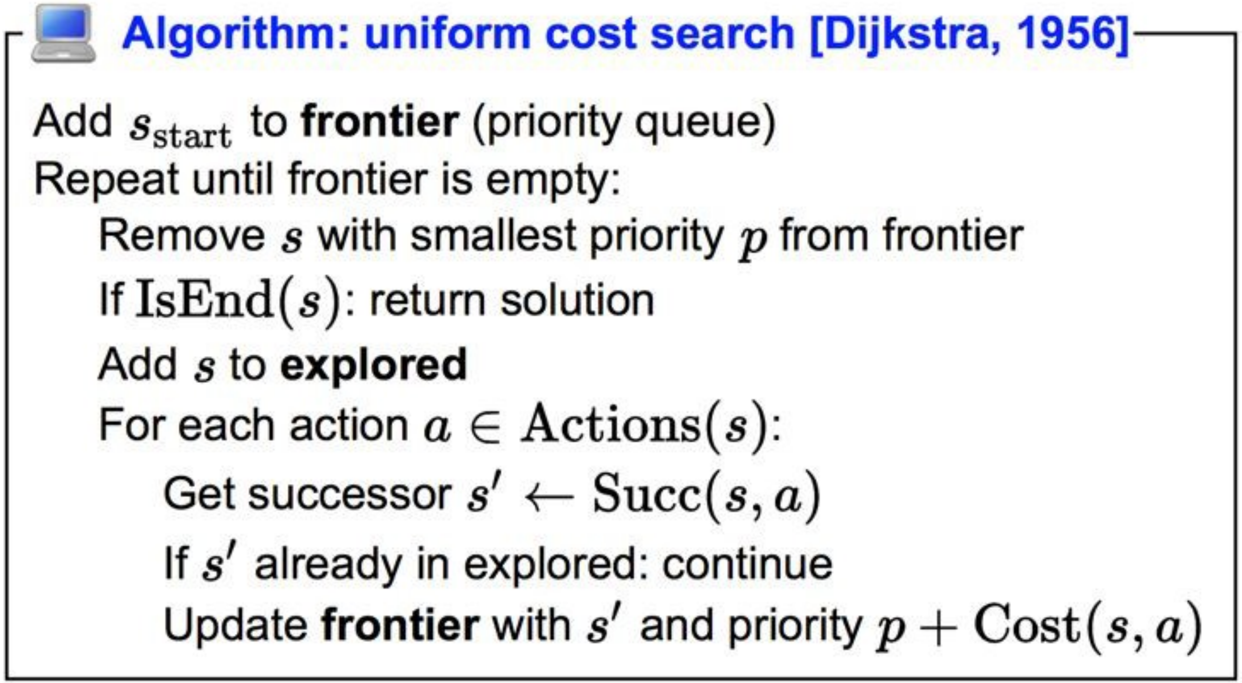


State $s = (i, d)$ (current city, #odd-#even)

#odd - #even

city	-1	0	1	2	3
C1	-	-	16	-	-
C2	-	14	-	-	-
C3	-	-	13	13	-
C4	?	7	7	-	-
C5	-	?	0	0	-

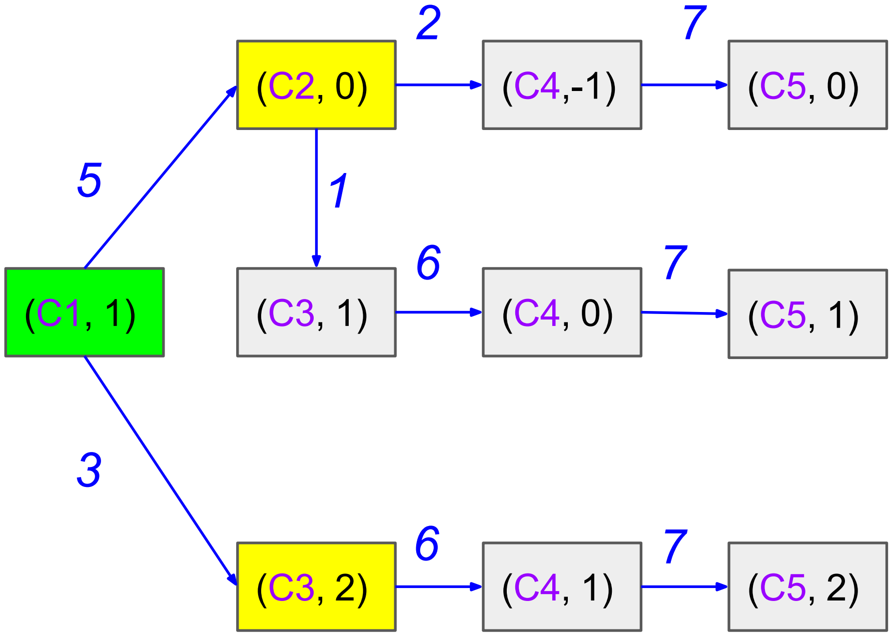
Solving the Problem: Uniform Cost Search



Algorithm: uniform cost search [Dijkstra, 1956]

```
Add $s_{\text{start}}$ to **frontier** (priority queue)
Repeat until frontier is empty:
    Remove $s$ with smallest priority $p$ from frontier
    If $\text{IsEnd}(s)$: return solution
    Add $s$ to **explored**
    For each action $a \in \text{Actions}(s)$:
        Get successor $s' \leftarrow \text{Succ}(s, a)$
        If $s'$ already in explored: continue
        Update **frontier** with $s'$ and priority $p + \text{Cost}(s, a)$
```

Simulation of UCS



State $s = (i, d)$ (current city, #odd-#even)

Explored:

(C1, 1) : 0

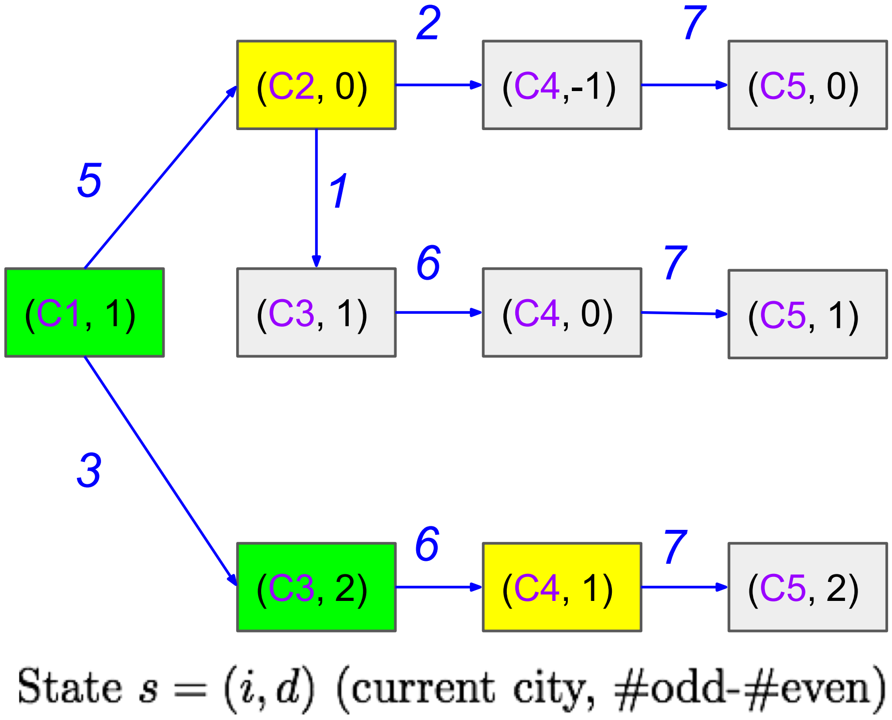
Frontier:

(C3, 2) : 3

(C2, 0) : 5

→ Frontier is a priority queue.

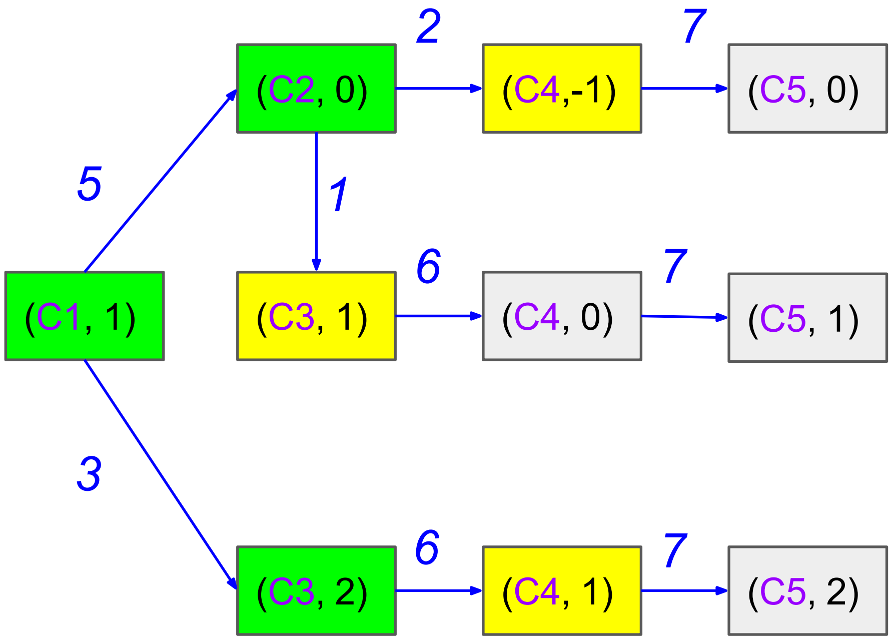
Simulation of UCS



Explored:
(C1, 1) : 0
(C3, 2) : 3

Frontier:
(C2, 0) : 5
(C4, 1) : 9

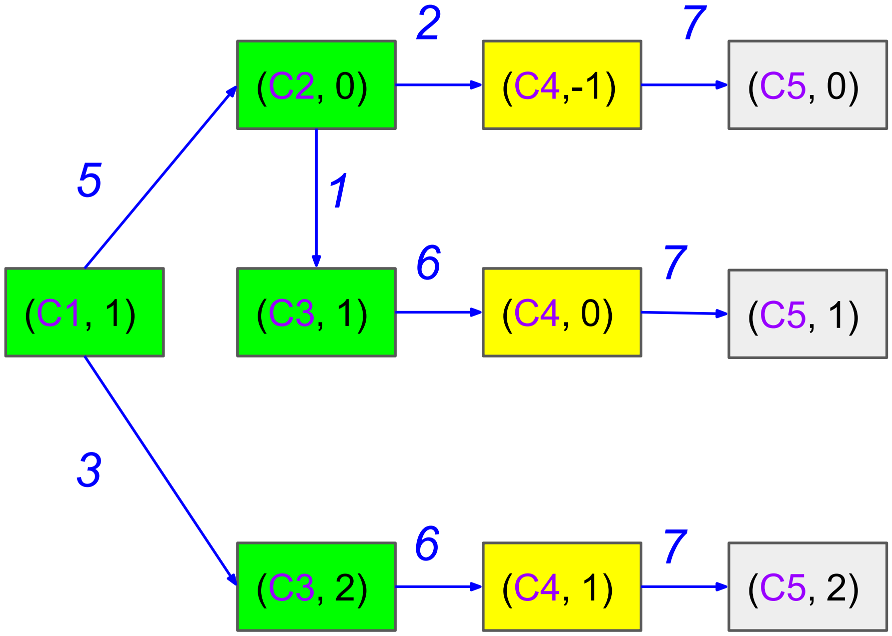
Simulation of UCS



State $s = (i, d)$ (current city, #odd-#even)

Explored:	Frontier:
(C1, 1) : 0	(C3, 1) : 6
(C3, 2) : 3	(C4, -1) : 7
(C2, 0) : 5	(C4, 1) : 9

Simulation of UCS

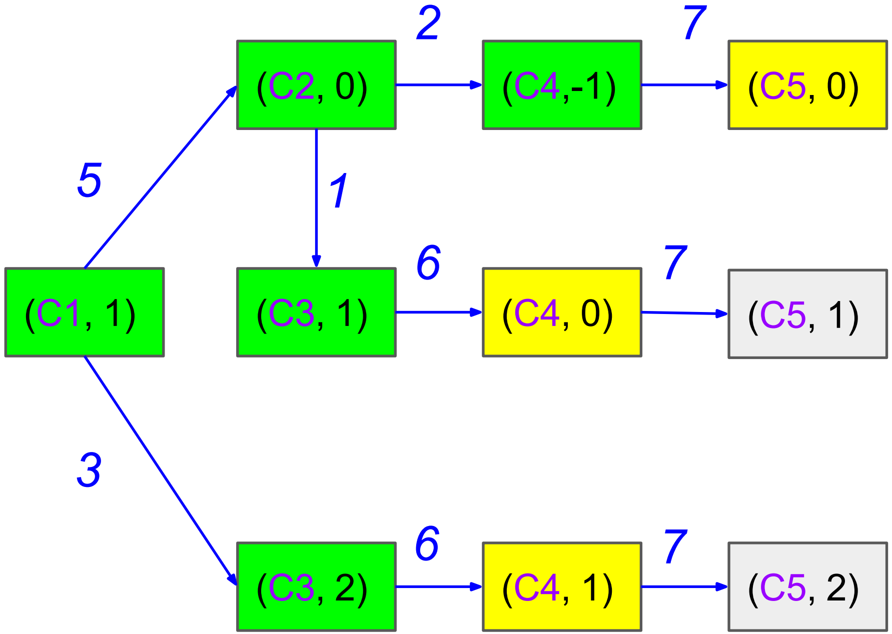


State $s = (i, d)$ (current city, #odd-#even)

Explored:
(C1, 1) : 0
(C3, 2) : 3
(C2, 0) : 5
(C3, 1): 6

Frontier:
(C4, -1) : 7
(C4, 1) : 9
(C4, 0): 12

Simulation of UCS



State $s = (i, d)$ (current city, #odd-#even)

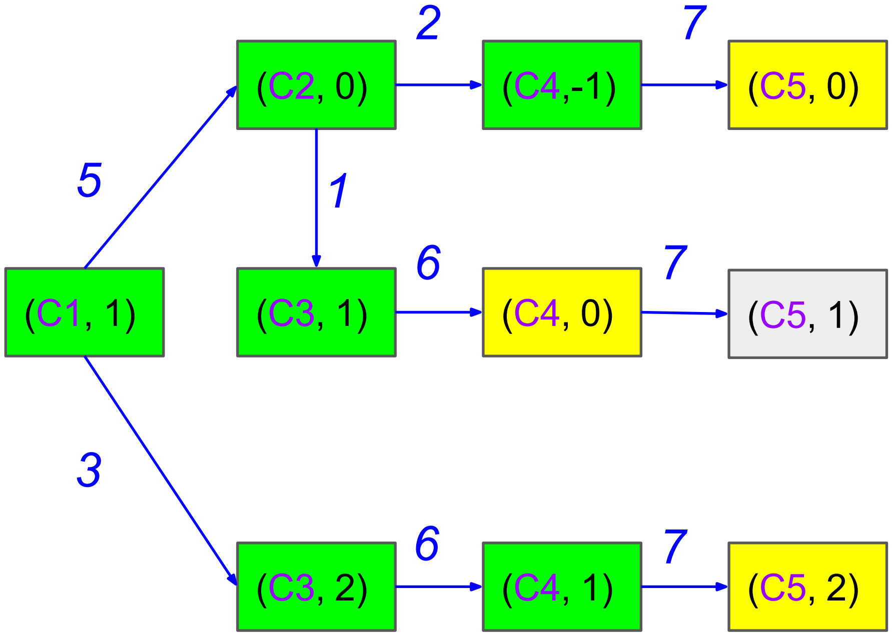
Explored:

(C1, 1) : 0
(C3, 2) : 3
(C2, 0) : 5
(C3, 1) : 6
(C4, -1) : 7

Frontier:

(C4, 1) : 9
(C4, 0) : 12
(C5, 0) : 14

Simulation of UCS



State $s = (i, d)$ (current city, #odd-#even)

Explored:

(C1, 1) : 0

(C3, 2) : 3

(C2, 0) : 5

(C3, 1) : 6

(C4, -1) : 7

(C4, 1) : 9

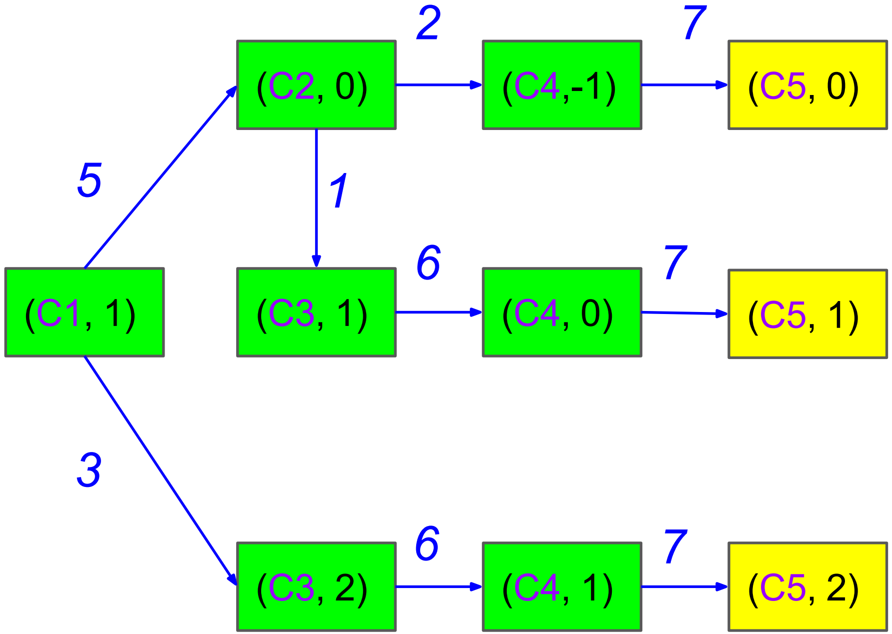
Frontier:

(C4, 0) : 12

(C5, 0) : 14

(C5, 2) : 16

Simulation of UCS



State $s = (i, d)$ (current city, #odd-#even)

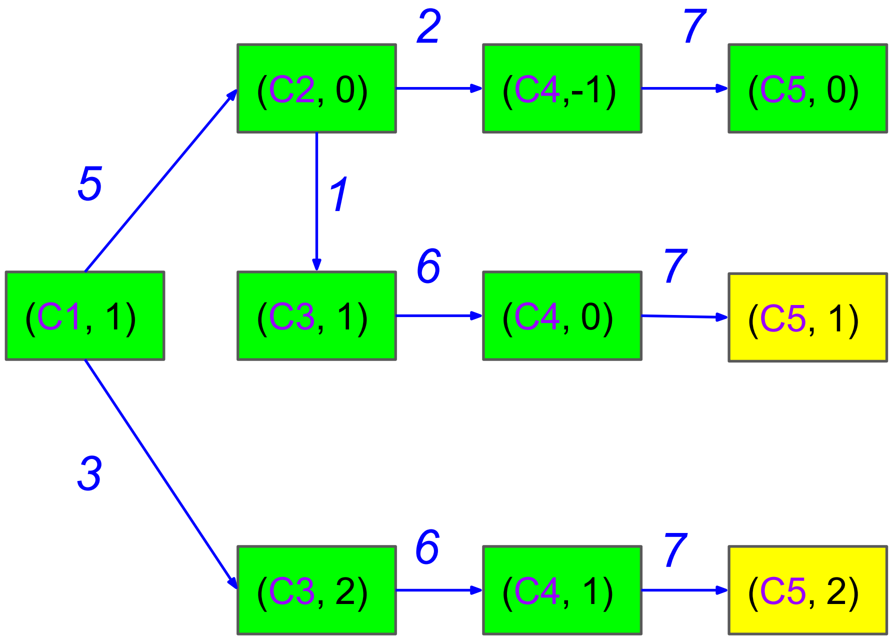
Explored:

(C1, 1) : 0
(C3, 2) : 3
(C2, 0) : 5
(C3, 1) : 6
(C4, -1) : 7
(C4, 1) : 9
(C4, 0) : 12

Frontier:

(C5, 0) : 14
(C5, 2) : 16
(C5, 1) : 19

Simulation of UCS



State $s = (i, d)$ (current city, #odd-#even)

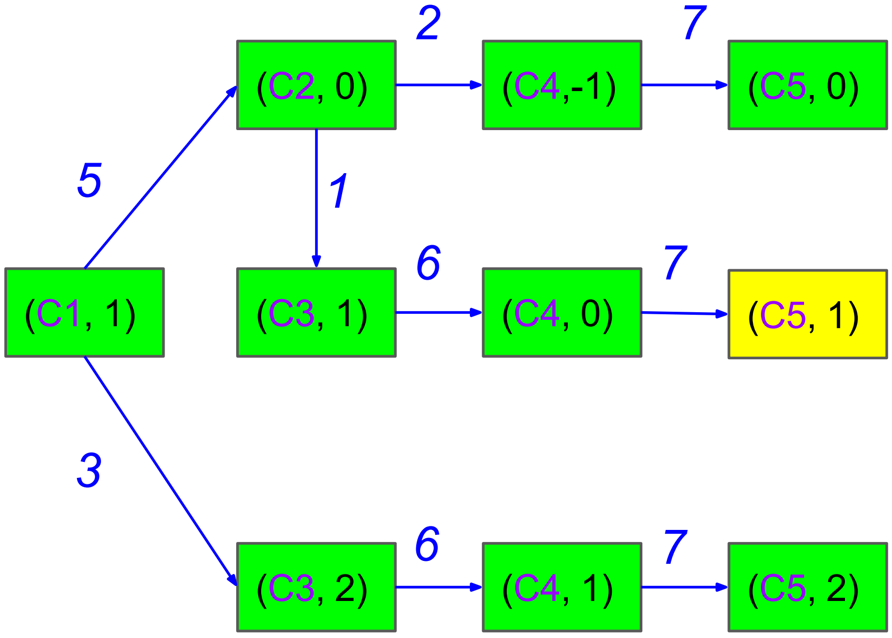
Explored:

(C1, 1) : 0
(C3, 2) : 3
(C2, 0) : 5
(C3, 1) : 6
(C4, -1) : 7
(C4, 1) : 9
(C4, 0) : 12
(C5, 0) : 14

Frontier:

(C5, 2) : 16
(C5, 1) : 19

Simulation of UCS



State $s = (i, d)$ (current city, #odd-#even)

Explored:

(C1, 1) : 0
(C3, 2) : 3
(C2, 0) : 5
(C3, 1) : 6
(C4, -1) : 7
(C4, 1) : 9
(C4, 0) : 12
(C5, 0) : 14
(C5, 2) : 16

Frontier:

(C5, 1) : 19

STOP!

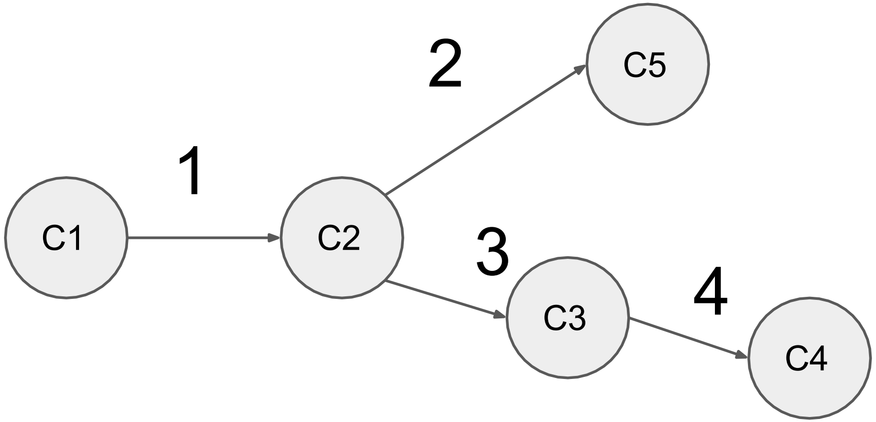
(Since we found C5 with #odd-#even > 0)

Comparison between DP and UCS

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

Runtime of DP is O(N)

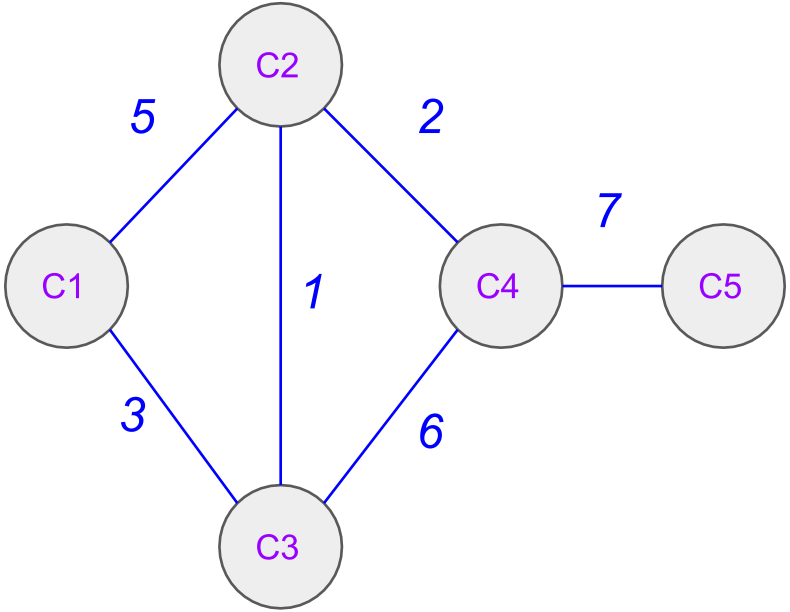
Runtime of UCS is O(n log n)



Example:
Start state C1, end state C5

-DP explores O(N) states.
-UCS will explore {C1, C2, C5} only. C3 will be in the frontier and C4 will be unexplored.

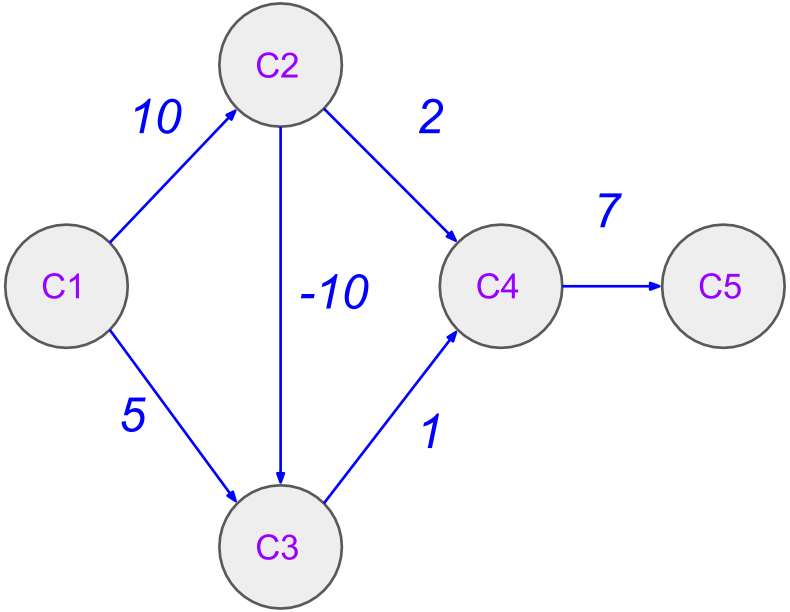
DP cannot handle cycles



Shortest path is [C1, C3, C2, C5] with cost 13.

Hard to define subproblems in undirected or cyclic graphs.

UCS cannot handle negative edge weights



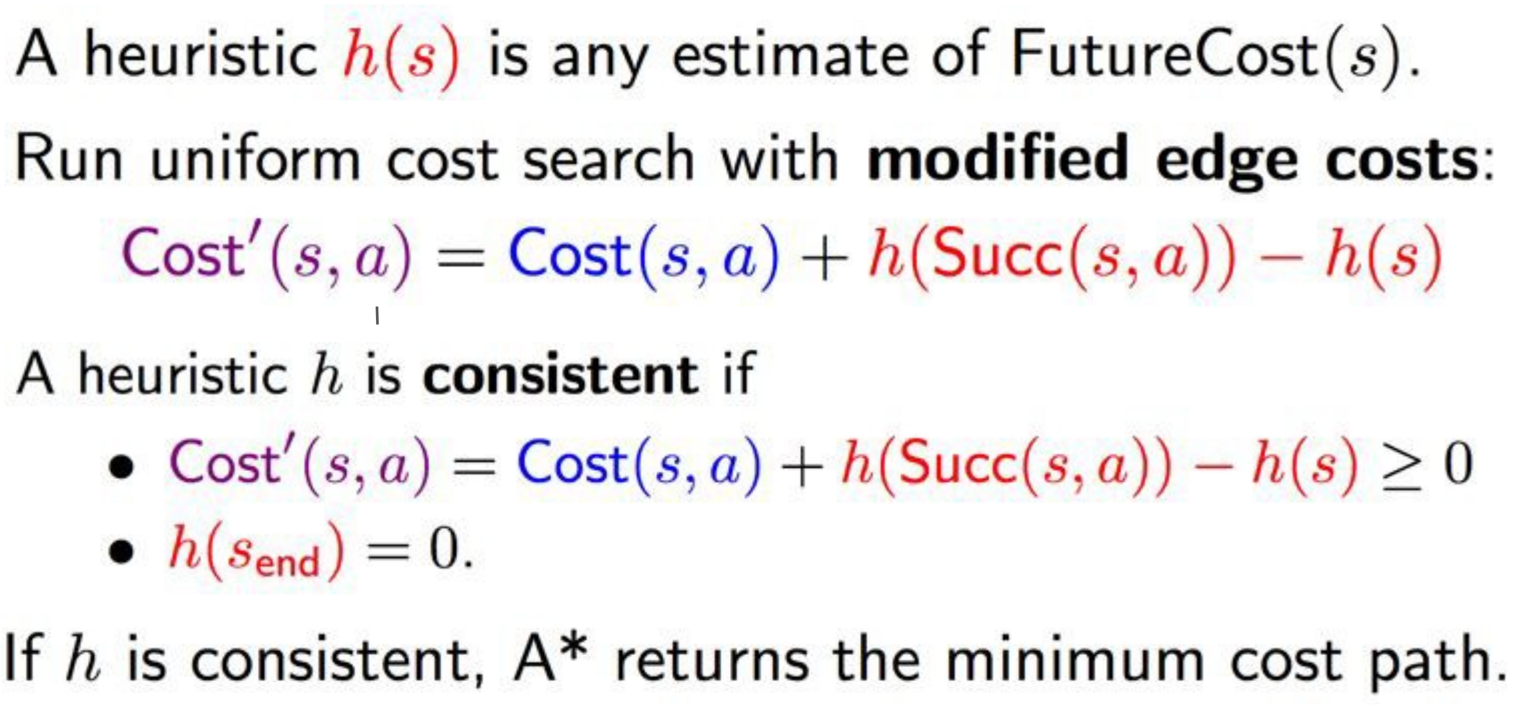
Best path is [C1,C2,C3,C4,C5] with cost of 8, but UCS will output [C1,C3,C4,C5] with cost of 13 because C3 is marked as ‘explored’ before C2.

Back to our section problem,
can we do the search faster than UCS?

Use A!*

https://qiao.github.io/PathFinding.js/visual/

Recap of A* Search from Lecture



A heuristic $h(s)$ is any estimate of $\text{FutureCost}(s)$.

Run uniform cost search with **modified edge costs**:

$$\text{Cost}'(s,a) = \text{Cost}(s,a) + h(\text{Succ}(s,a)) - h(s)$$

A heuristic h is **consistent** if

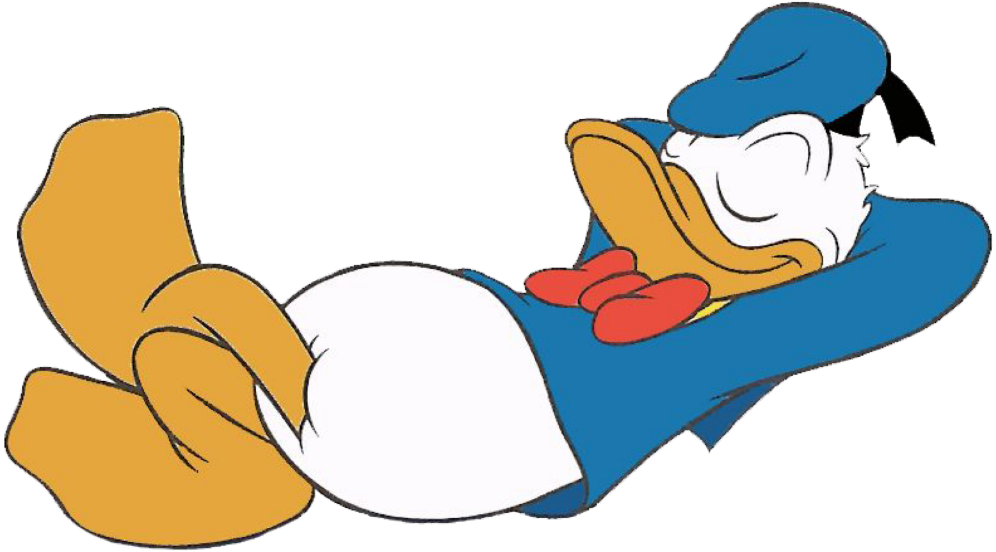
- $\text{Cost}'(s,a) = \text{Cost}(s,a) + h(\text{Succ}(s,a)) - h(s) \geq 0$
- $h(s_{\text{end}}) = 0$.

If h is consistent, A* returns the minimum cost path.

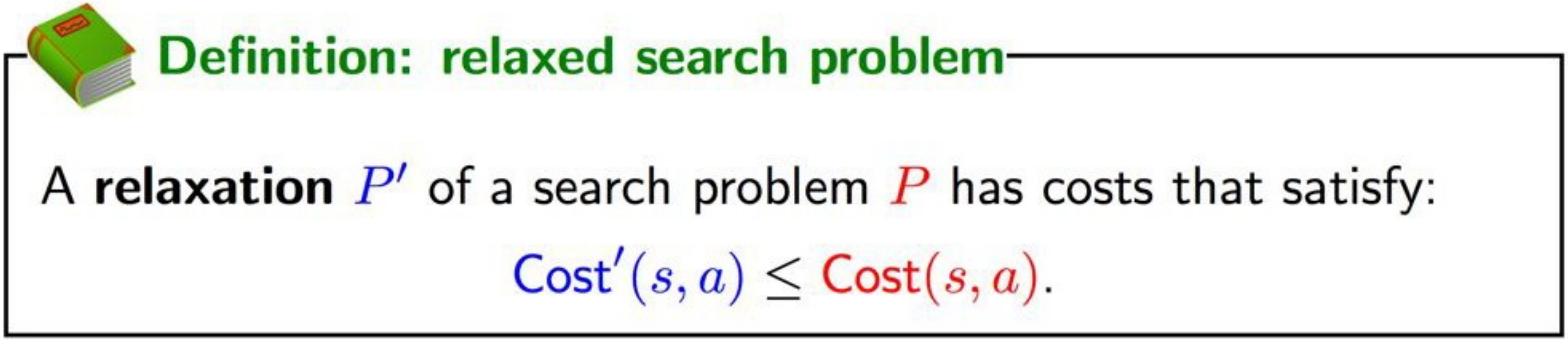
Finding a Heuristic by **Relaxation**

→ try to solve an easier (less constrained) version of the problem

→ attain a problem that **can be solved more efficiently**



Relaxation, more formally:



Definition: relaxed search problem

A **relaxation** P' of a search problem P has costs that satisfy:

$$\text{Cost}'(s, a) \leq \text{Cost}(s, a).$$

Which heuristic would you use to solve our problem more efficiently?

Hint: Relaxation!

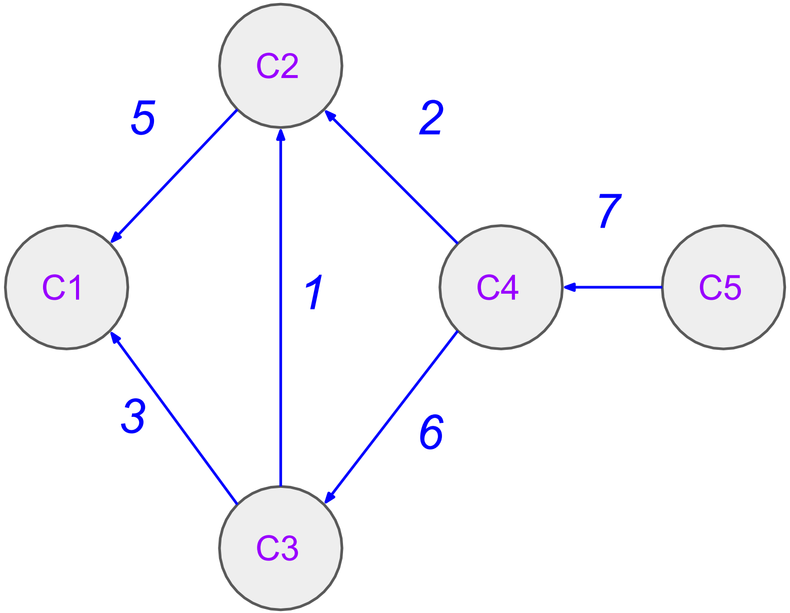
Heuristic for our problem

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 that the modified shortest path problem has ***O(N)*** states instead of ***O*(*N*^{2})**.

How to compute *h*?

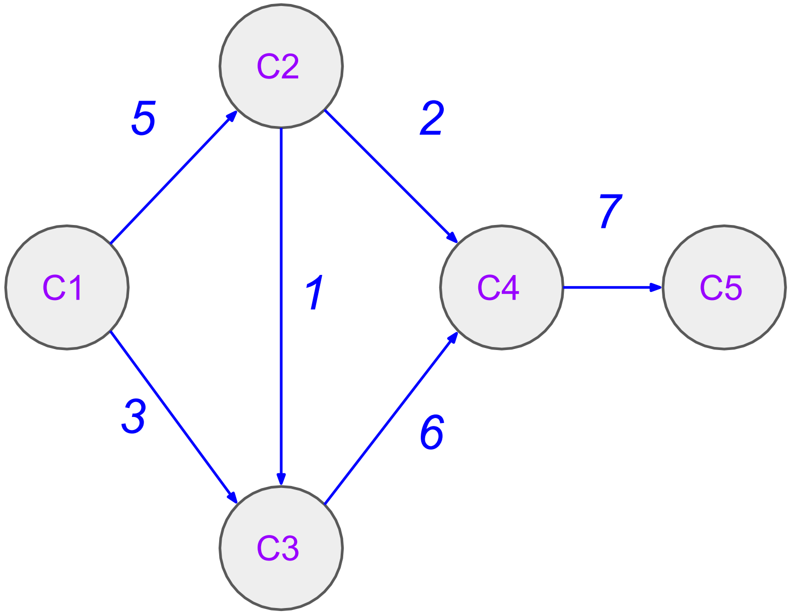


Reverse all edges, then perform UCS starting at C5 until C1 is found.

→ O(n log n) time (where n is # states whose distance to city CN is no farther than the distance of city C1 to city CN)

city	C1	C2	C3	C4	C5
h	14	9	13	7	0

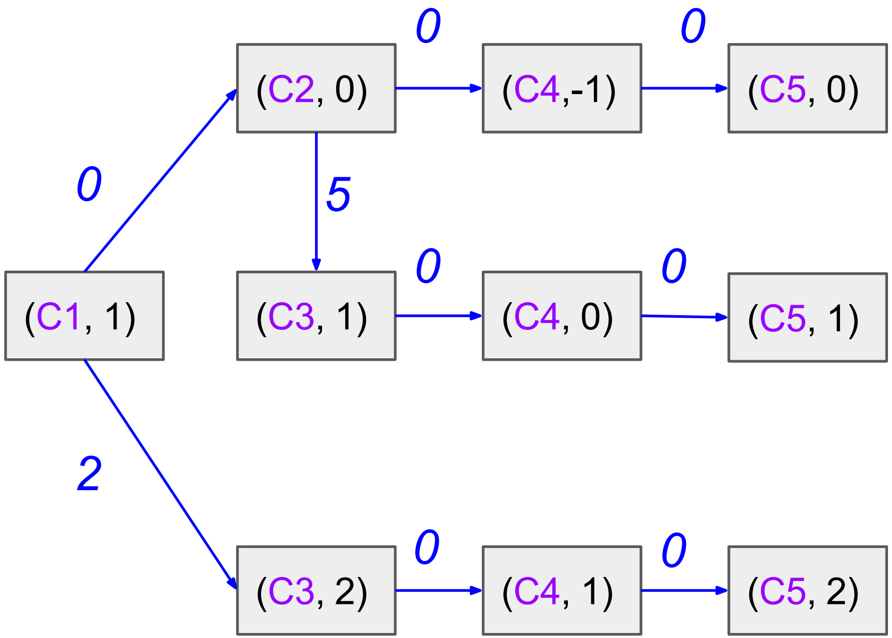
Original Graph



city	C1	C2	C3	C4	C5
h	14	9	13	7	0

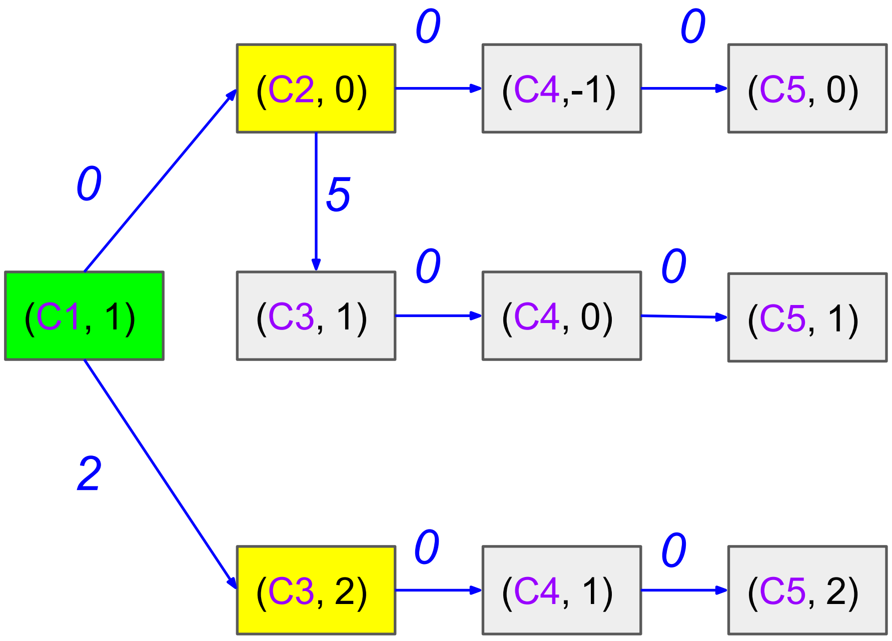
Modified State Graph

(updated edge costs)



State $s = (i, d)$ (current city, #odd-#even)

Simulation of UCS (A*)



State $s = (i, d)$ (current city, #odd-#even)

Explored:

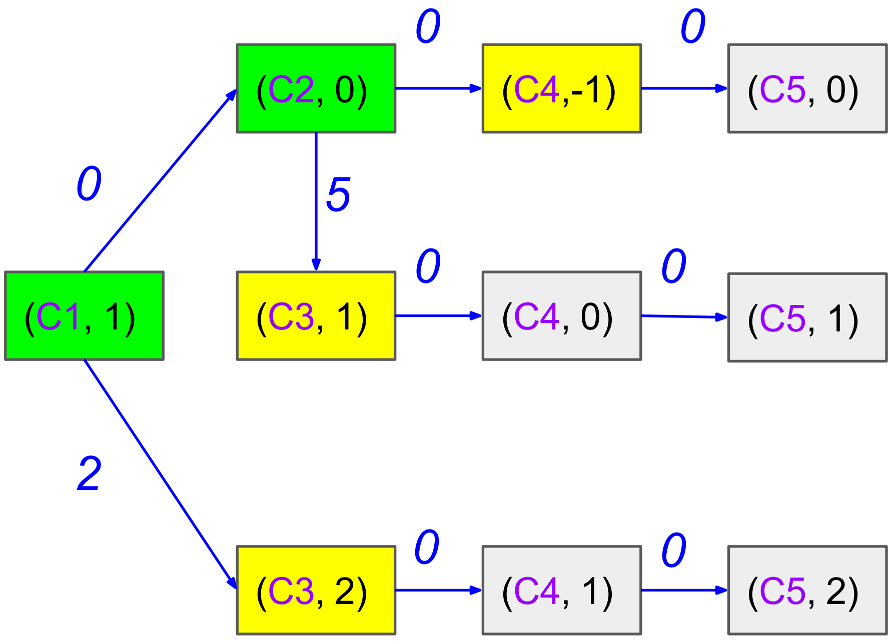
(C1, 1) : 0

Frontier:

(C2, 0) : 0

(C3, 2) : 2

Simulation of UCS (A*)

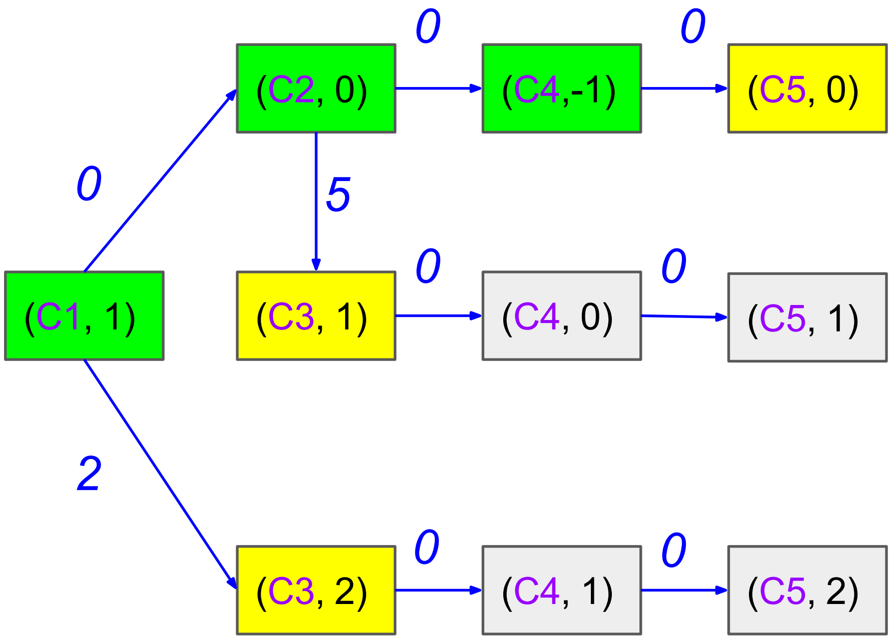


State $s = (i, d)$ (current city, #odd-#even)

Explored:
(C1, 1) : 0
(C2, 0) : 0

Frontier:
(C4, -1) : 0
(C3, 2) : 2
(C3, 1) : 5

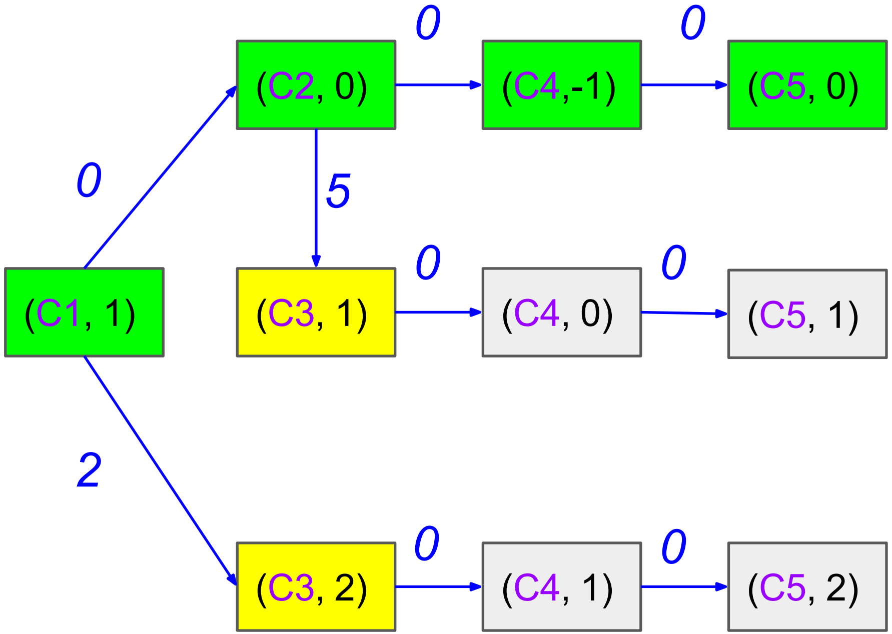
Simulation of UCS (A*)



State $s = (i, d)$ (current city, #odd-#even)

Explored:	Frontier:
(C1, 1) : 0	(C5, 0) : 0
(C2, 0) : 0	(C3, 2) : 2
(C4, -1) : 0	(C3, 1) : 5

Simulation of UCS (A*)

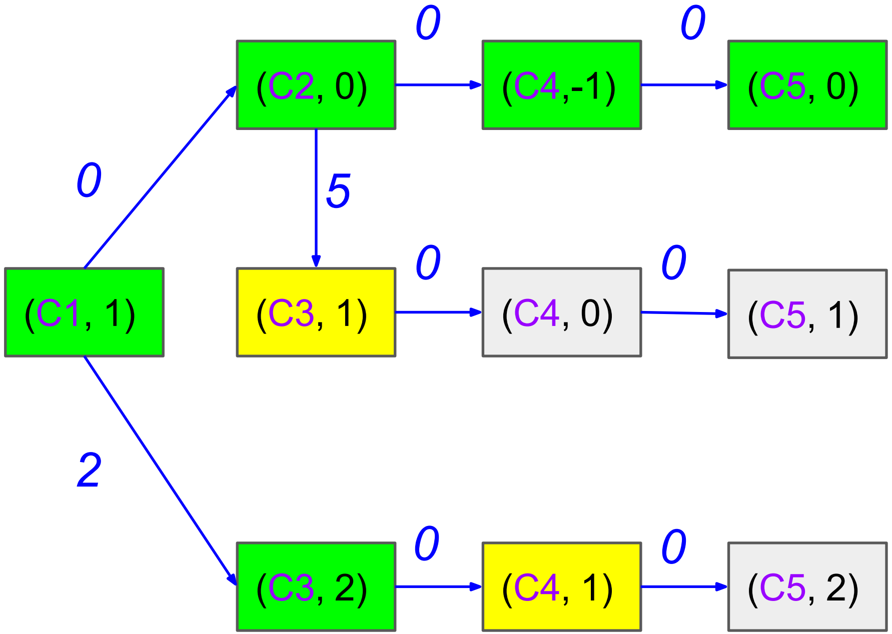


State $s = (i, d)$ (current city, #odd-#even)

Explored:
(C1, 1) : 0
(C2, 0) : 0
(C4, -1) : 0
(C5, 0) : 0

Frontier:
(C3, 2) : 2
(C3, 1) : 5

Simulation of UCS (A*)



State $s = (i, d)$ (current city, #odd-#even)

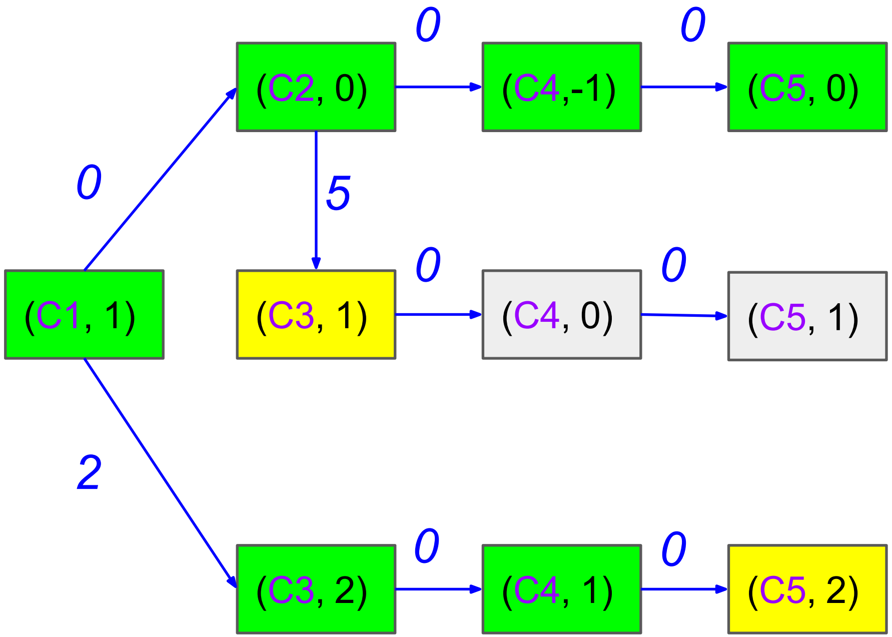
Explored:

(C1, 1) : 0
(C2, 0) : 0
(C4, -1) : 0
(C5, 0) : 0
(C3, 2) : 2

Frontier:

(C4, 1) : 2
(C3, 1) : 5

Simulation of UCS (A*)



State $s = (i, d)$ (current city, #odd-#even)

Explored:

(C1, 1) : 0

(C2, 0) : 0

(C4, -1) : 0

(C5, 0) : 0

(C3, 2) : 2

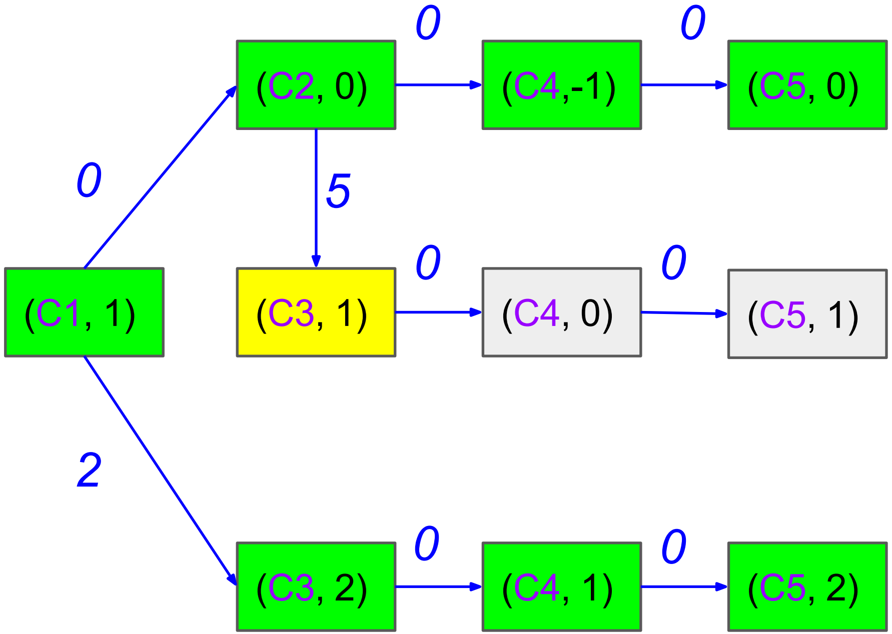
(C4, 1): 2

Frontier:

(C5, 2) : 2

(C3, 1) : 5

Simulation of UCS (A*)



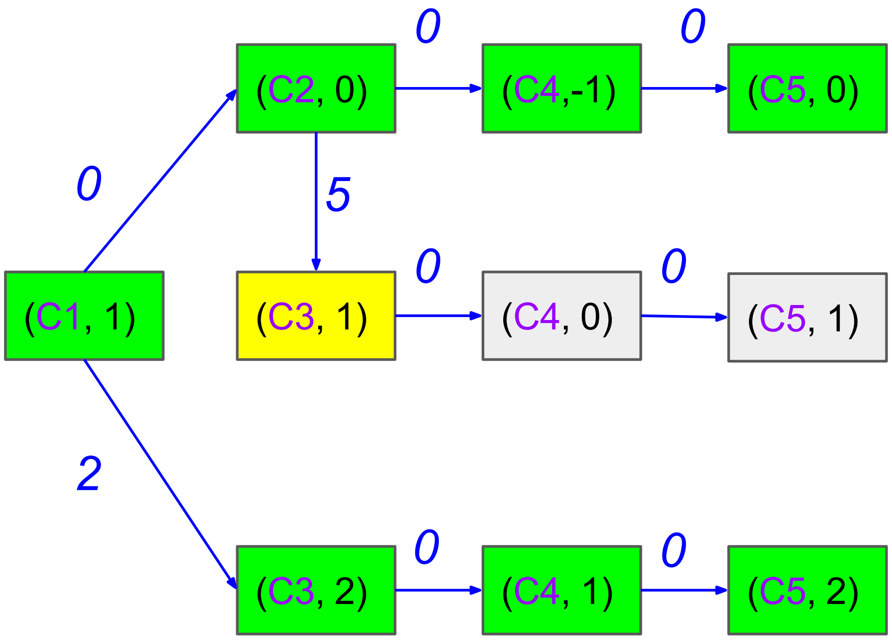
State $s = (i, d)$ (current city, #odd-#even)

Explored:
(C1, 1) : 0
(C2, 0) : 0
(C4, -1) : 0
(C5, 0) : 0
(C3, 2) : 2
(C4, 1) : 2
(C5, 2) : 2

Frontier:
(C3, 1) : 5

STOP!

Simulation of UCS (A*)



State $s = (i, d)$ (current city, #odd-#even)

Explored:
(C1, 1) : 0
(C2, 0) : 0
(C4, -1) : 0
(C5, 0) : 0
(C3, 2) : 2
(C4, 1) : 2
(C5, 2) : 2

Frontier:
(C3, 1) : 5

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

Comparison of States visited

UCS

Explored:

(C1, 1) : 0
(C3, 2) : 3
(C2, 0) : 5
(C3, 1) : 6
(C4, -1) : 7
(C4, 1) : 9
(C4, 0) : 12
(C5, 0) : 14
(C5, 2) : 16

Frontier:

(C5, 1) : 19

UCS(A*)

Explored:

(C1, 1) : 0
(C2, 0) : 0
(C4, -1) : 0
(C5, 0) : 0
(C3, 2) : 2
(C4, 1) : 2
(C5, 2) : 2

Frontier:

(C3, 1) : 5

Comparison of States visited

UCS

Explored:
(C1, 1) : 0
(C3, 2) : 3
(C2, 0) : 5
(C3, 1) : 6
(C4, -1) : 7
(C4, 1) : 9
(C4, 0) : 12
(C5, 0) : 14
(C5, 2) : 16

Frontier:
(C5, 1) : 19

UCS explored 9 states

UCS(A*)

Explored:
(C1, 1) : 0
(C2, 0) : 0
(C4, -1) : 0
(C5, 0) : 0
(C3, 2) : 2
(C4, 1) : 2
(C5, 2) : 2

Frontier:
(C3, 1) : 5

UCS(A*) explored 7 states

Summary

- ***States Representation/Modelling***
  - make state representation compact, remove unnecessary 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****
  - Introduce heuristic to guide search
  - ensure that relaxed problem can be solved more efficiently

Now let’s practice modeling our search problems!