

CS221 Section 3: Search

*DP, UCS, A**

10/13/2016

A map of California illustrating three driving routes from San Francisco to Yosemite National Park. The routes are color-coded: blue for the fastest and shortest, and grey for the longest and slowest. Each route is accompanied by a callout box showing the total distance and estimated driving time. The map also shows major cities, highways, and geographical features like Sierra National Forest.

| Route Color | Distance (miles) | Estimated Time |
|-------------|------------------|----------------|
| Blue | 187 | 3 h 22 min |
| Blue | 177 | 3 h 12 min |
| Grey | 202 | 3 h 36 min |

A screenshot from the Atari game 'The Great Escape'. The top of the screen shows a score of '053' and a level indicator '3 1'. The main play area is a black rectangle representing a prison yard. At the top of this area is a horizontal band of colors: orange, yellow, and green, representing a sunset or sunrise. A small black silhouette of a guard is visible on the horizon line. In the foreground, a small red silhouette of a prisoner is visible. The entire scene is framed by a thick black border, which is itself surrounded by a blue border.

The screenshot shows a calendar application with a weekly view for December 7-13. The calendar is organized into columns for each day and rows for each hour. Five team members are listed at the top: Daniel Owens, Emma Clark, Matt Edwards, Sam Smith, and Darren Owens. Events are represented by colored blocks with labels and times. For example, on Monday, Daniel Owens has a 'Account 1 issue' from 8:00 to 10:00, Emma Clark has a 'Account 2 issue' from 10:00 to 12:00, and Matt Edwards has a 'Account 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31' from 12:00 to 2:00. The calendar also shows a 'Today' button and a 'Calendar' button in the top left corner.

Scheduling Problems

Search in AI:

- A tool for goal-based sequential problem solving
 - Many real-world applications!

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
- **Is End** (s): found solution?

Search Algorithms

- Breadth-First Search (BFS)
- Depth-First Search (DFS)
- DFS with Iterative Deepening (DFS-ID)
- Backtracking
- Dynamic Programming (DP)
- Uniform Cost Search (UCS)
- UCS with A* heuristic



Search Algorithms

- Breadth-First Search (BFS)
- Depth-First Search (DFS)
- DFS with Iterative Deepening (DFS-ID)
- Backtracking
- Dynamic Programming (DP)
- Uniform Cost Search (UCS)
- UCS with A* heuristic



Section Problem

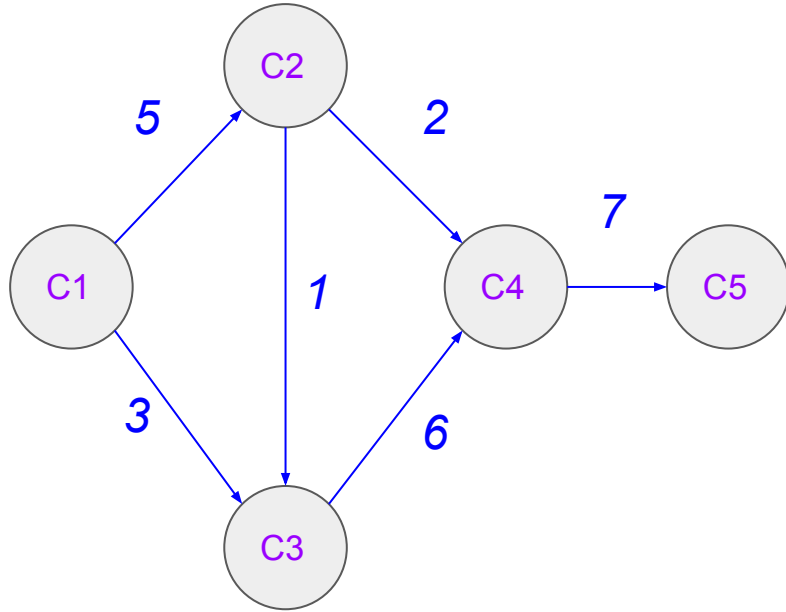
There exists **N cities**, labeled from $C1$ to CN .

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).

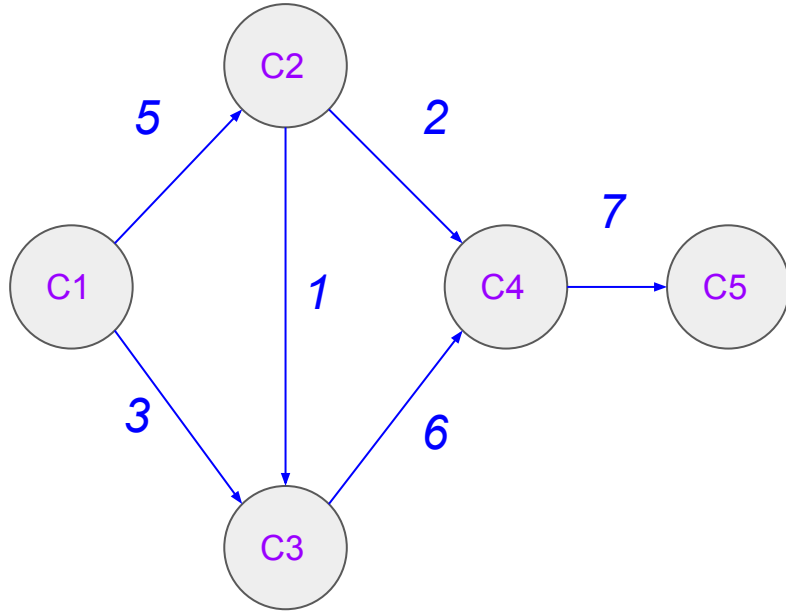
From city $C1$, we want to travel to city CN . What is the **shortest time** required to make this trip, given the **constraint** that we should visit **more odd-labeled cities than even labeled cities**?

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

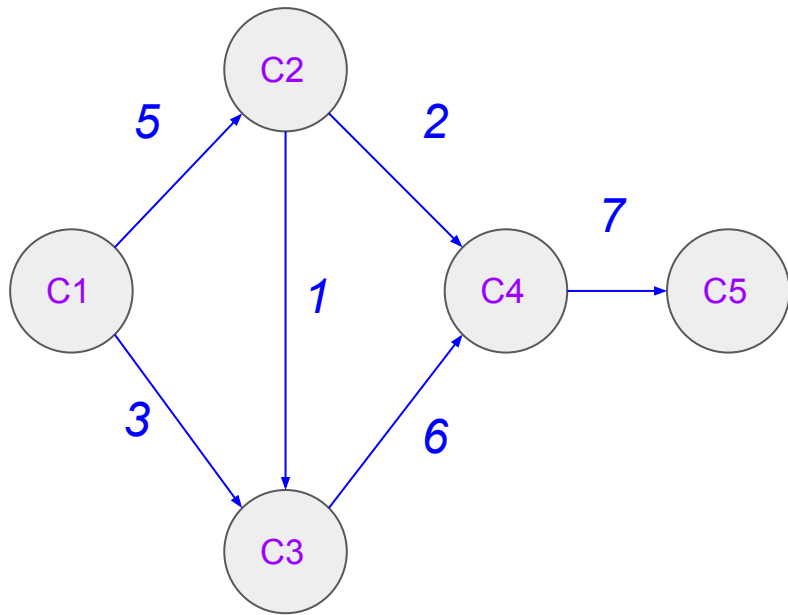
Can We Do Better?

Check if all the information is really required

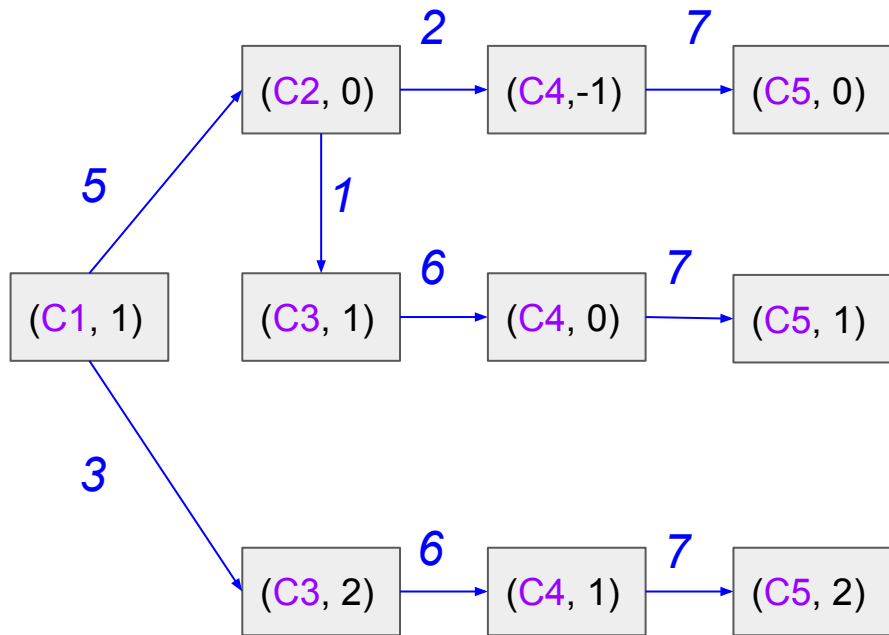
Instead of storing **#odd** and **#even**, we can store **#odd - #even** directly; this still allows us to check whether **#odd - #even > 0** at **(N, #odd, #even)**

(current_city, #odd - #even) → O(N²) 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)

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

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

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

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

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

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

*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

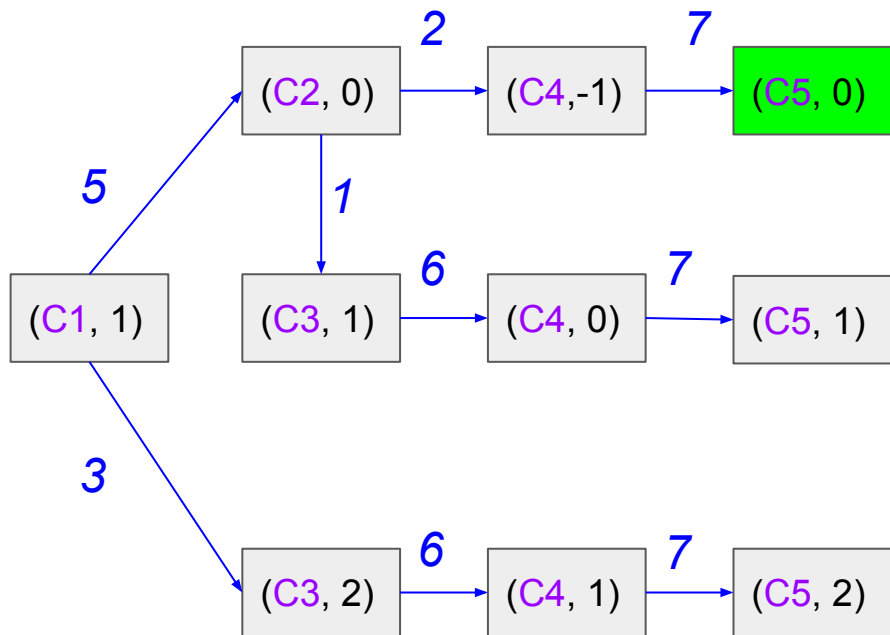
Since we have a **DAG** and all edges are positive, both work!

Solving the Problem: Dynamic Programming

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

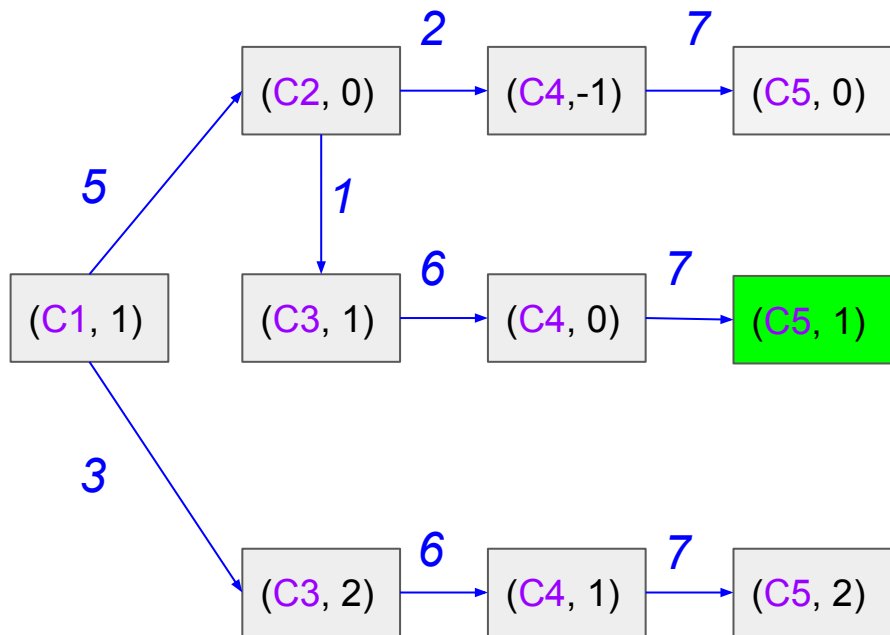
If s has no successors, we set it as *undefined*

Simulation of DP



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

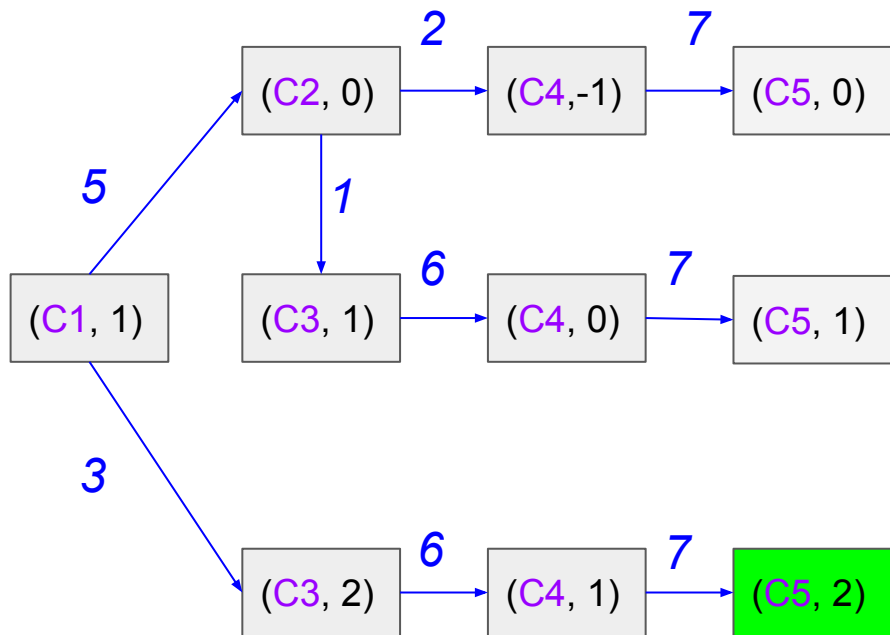
Simulation of DP



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

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

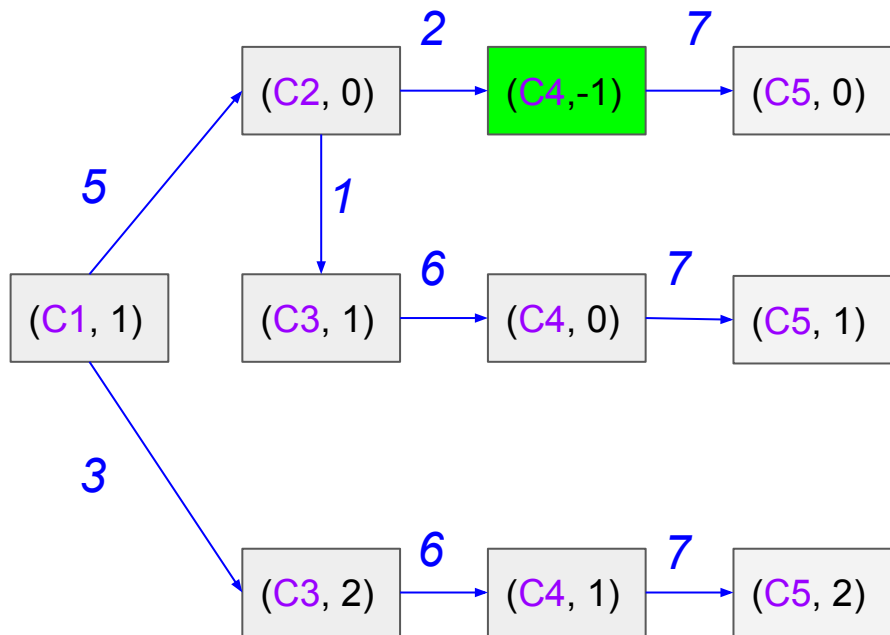
Simulation of DP



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

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

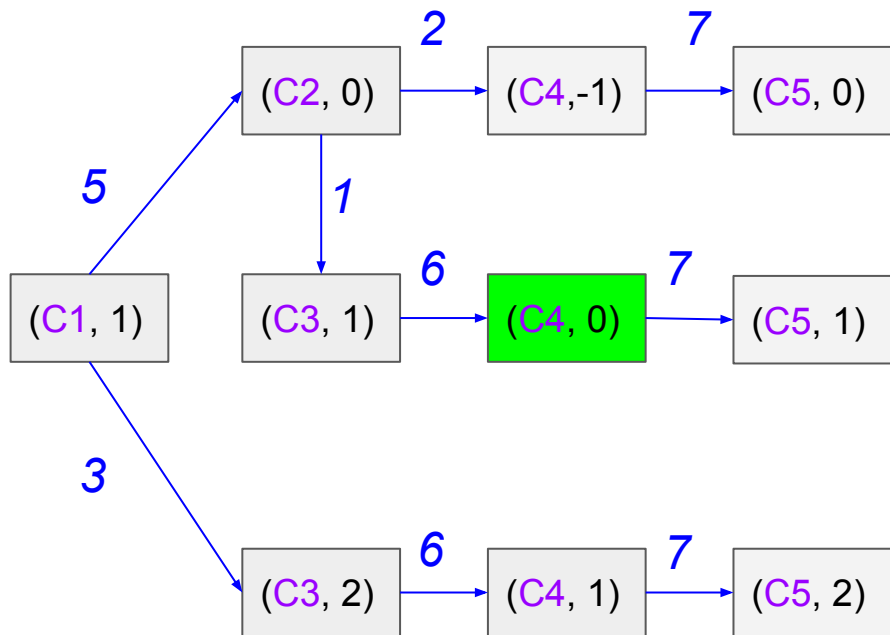
Simulation of DP



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

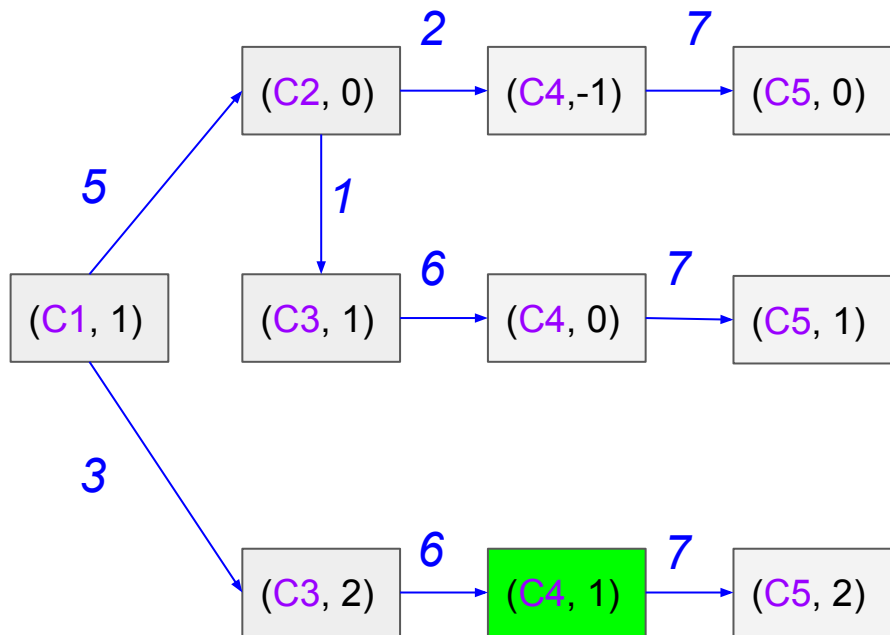
| | | #odd - #even | | | | |
|------|----|--------------|---|---|---|---|
| | | -1 | 0 | 1 | 2 | 3 |
| city | C1 | - | - | - | - | - |
| | C2 | - | - | - | - | - |
| | C3 | - | - | - | - | - |
| | C4 | ? | - | - | - | - |
| | C5 | - | ? | 0 | 0 | - |

Simulation of DP



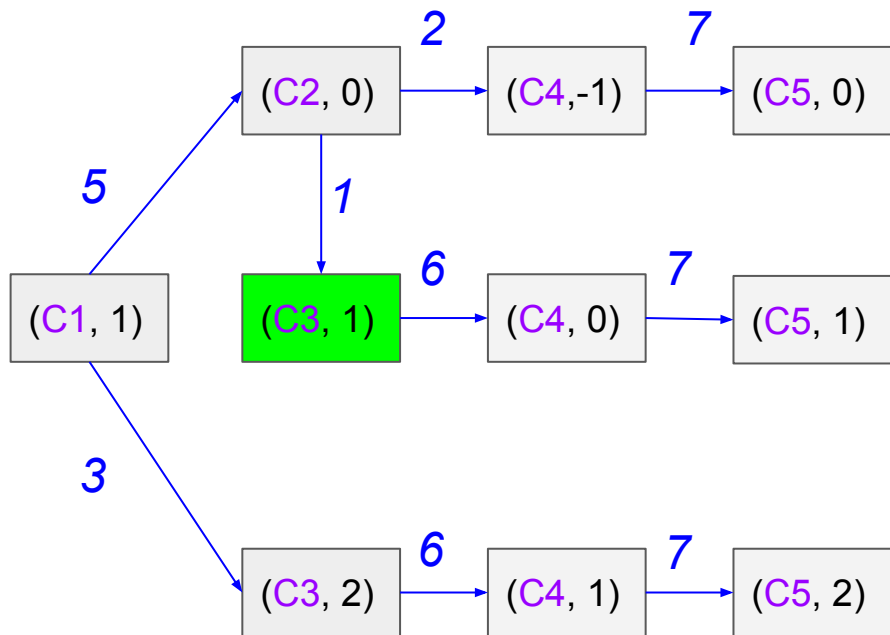
| | #odd - #even | | | | |
|------|--------------|---|---|---|---|
| | -1 | 0 | 1 | 2 | 3 |
| city | C1 | - | - | - | - |
| | C2 | - | - | - | - |
| | C3 | - | - | - | - |
| | C4 | ? | 7 | - | - |
| | C5 | - | ? | 0 | 0 |

Simulation of DP



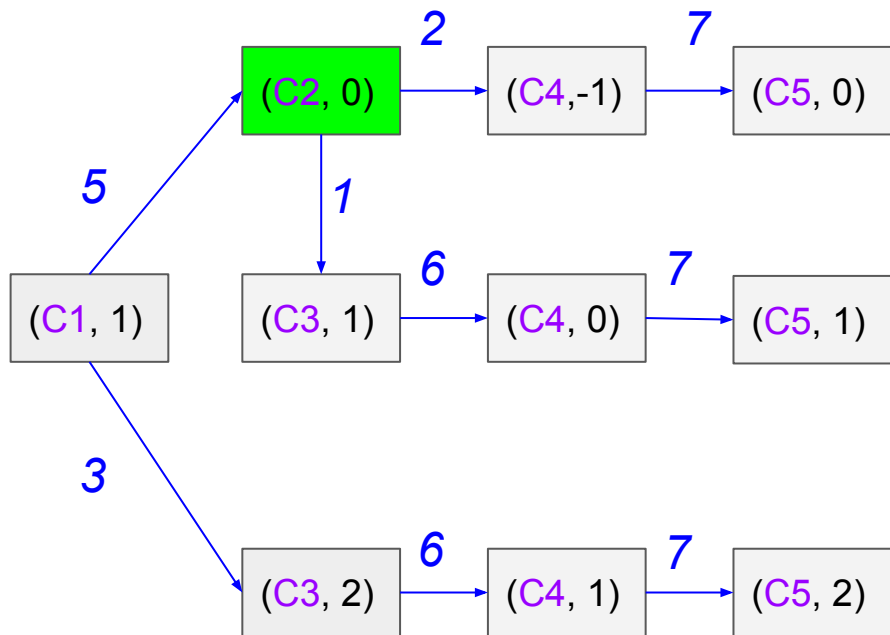
| | | #odd - #even | | | | |
|------|----|--------------|---|---|---|---|
| | | -1 | 0 | 1 | 2 | 3 |
| city | C1 | - | - | - | - | - |
| | C2 | - | - | - | - | - |
| | C3 | - | - | - | - | - |
| | C4 | ? | 7 | 7 | - | - |
| | C5 | - | ? | 0 | 0 | - |

Simulation of DP



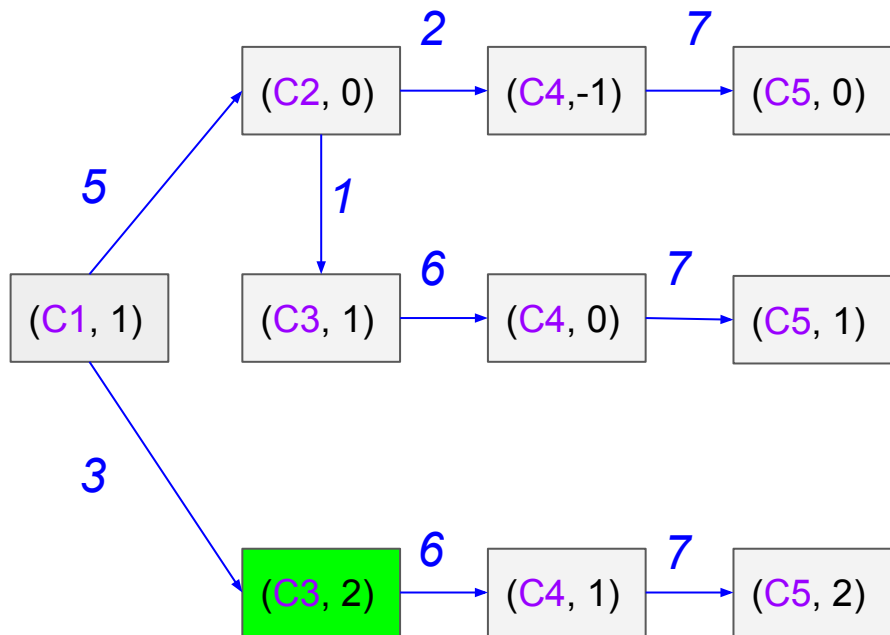
| | | #odd - #even | | | | |
|------|----|--------------|---|----|---|---|
| | | -1 | 0 | 1 | 2 | 3 |
| city | C1 | - | - | - | - | - |
| | C2 | - | - | - | - | - |
| | C3 | - | - | 13 | - | - |
| | C4 | ? | 7 | 7 | - | - |
| | C5 | - | ? | 0 | 0 | - |

Simulation of DP



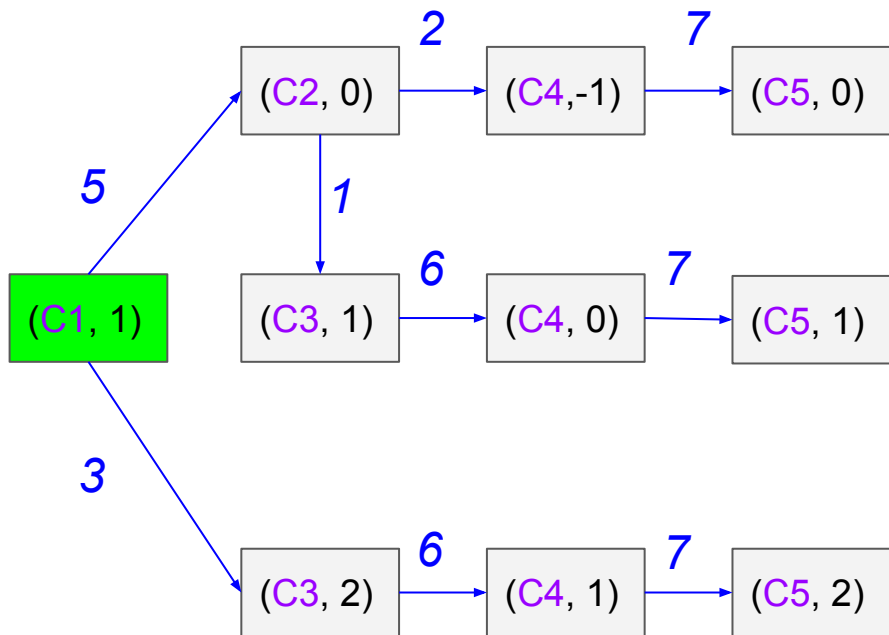
| | | #odd - #even | | | | |
|------|----|--------------|----|----|---|---|
| | | -1 | 0 | 1 | 2 | 3 |
| city | C1 | - | - | - | - | - |
| | C2 | - | 14 | - | - | - |
| | C3 | - | - | 13 | - | - |
| | C4 | ? | 7 | 7 | - | - |
| | C5 | - | ? | 0 | 0 | - |

Simulation of DP



| | #odd - #even | | | | |
|------|--------------|---|----|----|----|
| | -1 | 0 | 1 | 2 | 3 |
| city | 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 | | | | |
|------|----|--------------|----|----|----|---|
| | | -1 | 0 | 1 | 2 | 3 |
| city | 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_{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**

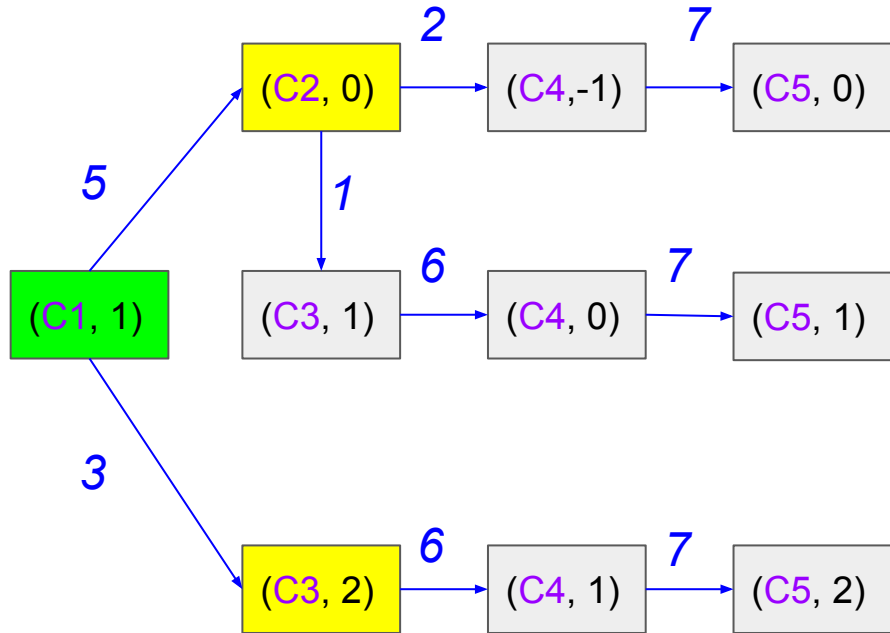
 For each action $a \in \text{Actions}(s)$:

 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



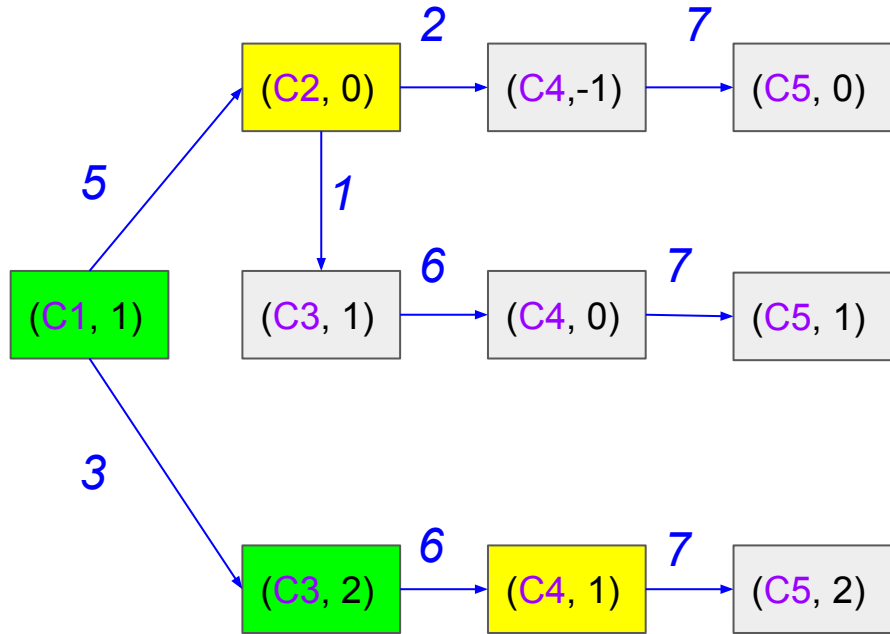
State $s = (i, d)$ (current city, #odd-#even)

Explored:
(C1, 1) : 0

Frontier:
(C3, 2) : 3
(C2, 0) : 5

→ Frontier is a
priority queue.

Simulation of UCS

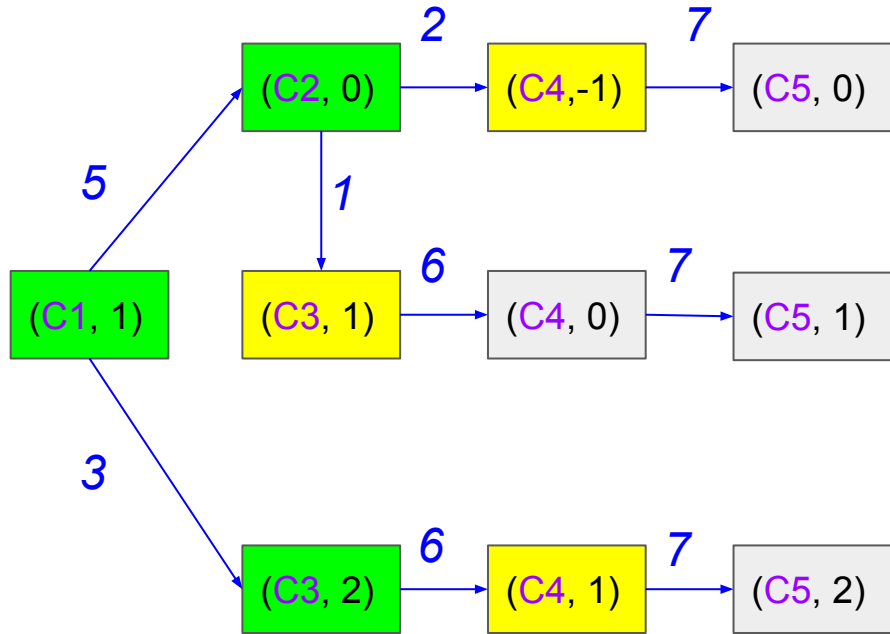


State $s = (i, d)$ (current city, #odd-#even)

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:

$(C1, 1) : 0$

$(C3, 2) : 3$

$(C2, 0) : 5$

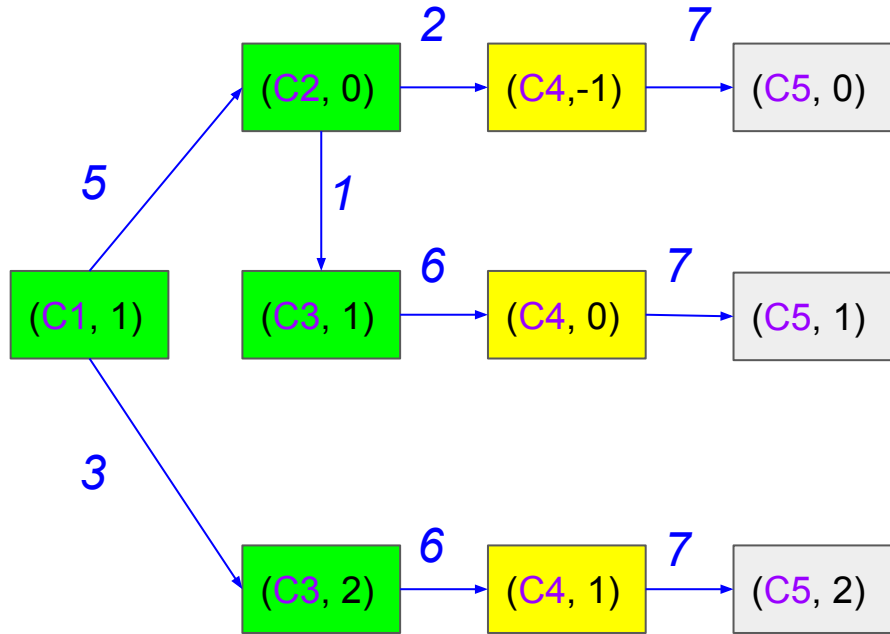
Frontier:

$(C3, 1) : 6$

$(C4, -1) : 7$

$(C4, 1) : 9$

Simulation of UCS



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

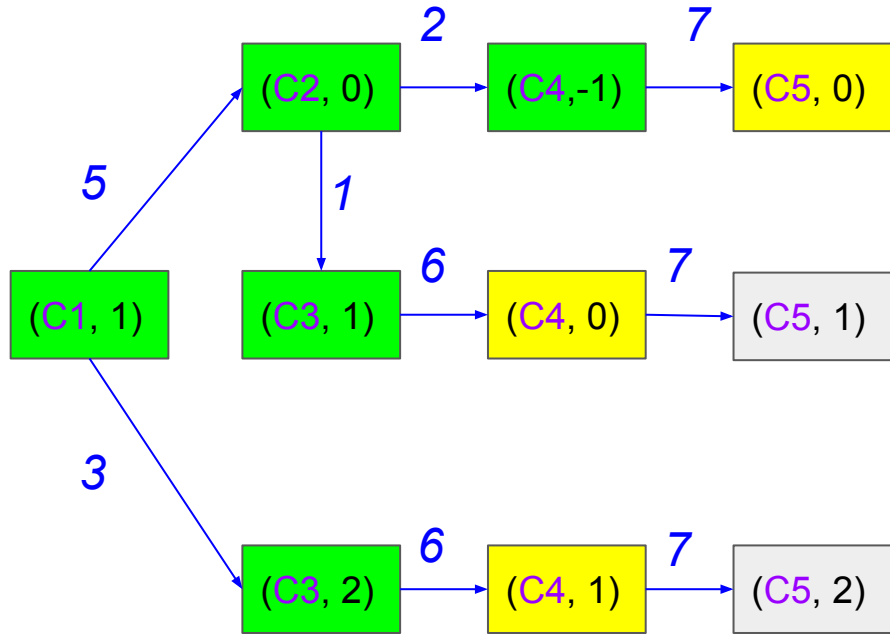
Explored:

(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)

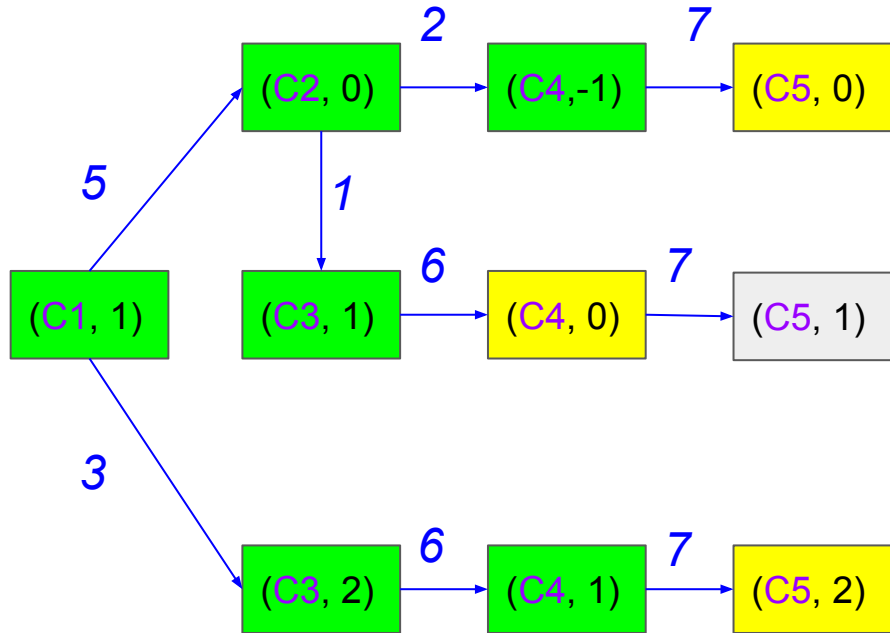
Explored:

(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)

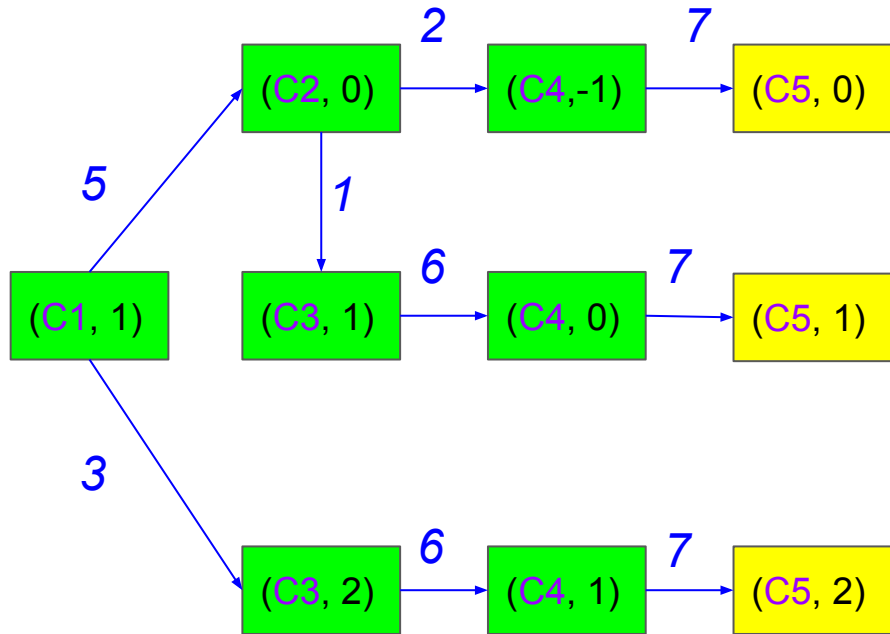
Explored:

(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)

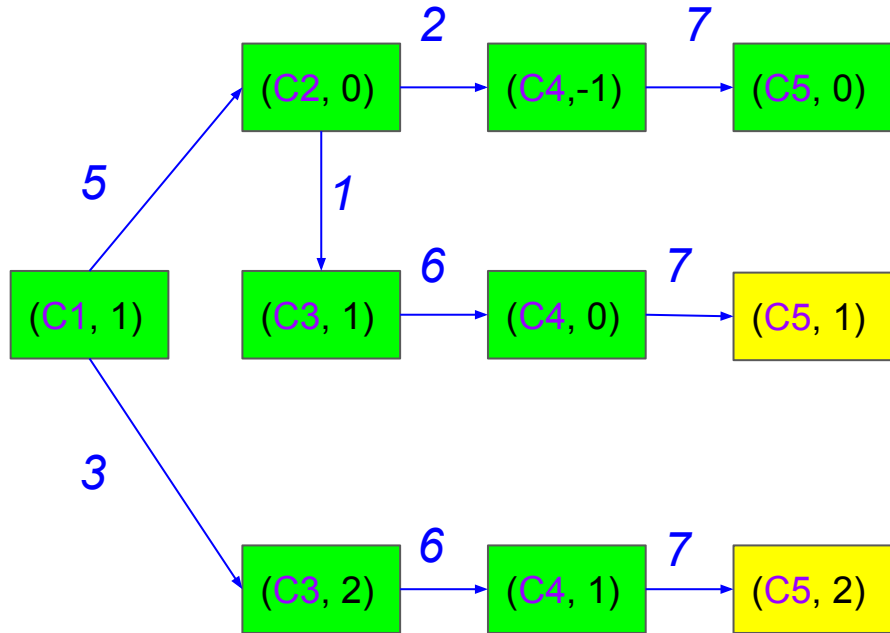
Explored:

(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)

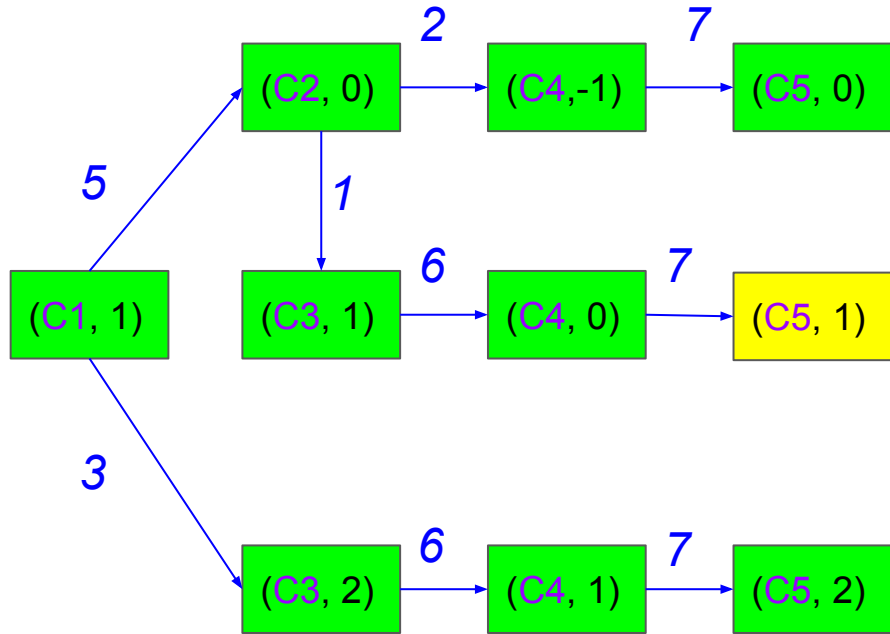
Explored:

(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!

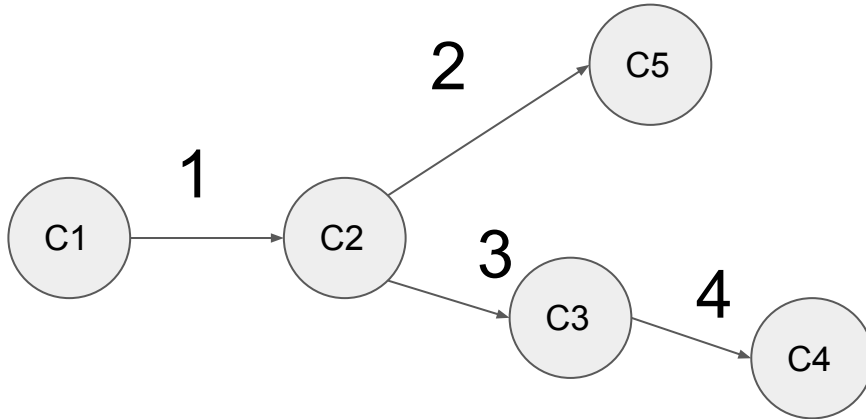
(Since we found
C5 with
#odd-#even > 0)

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 $O(n \log n)$



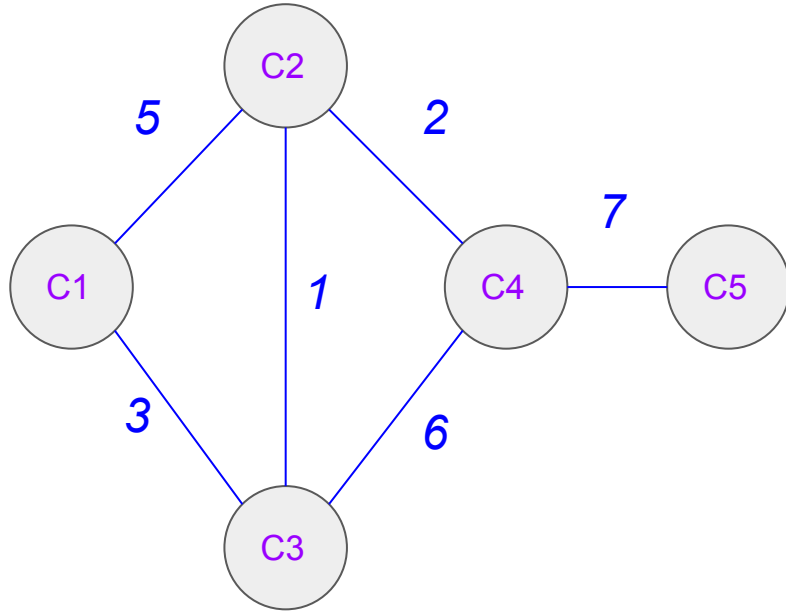
Example:

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.

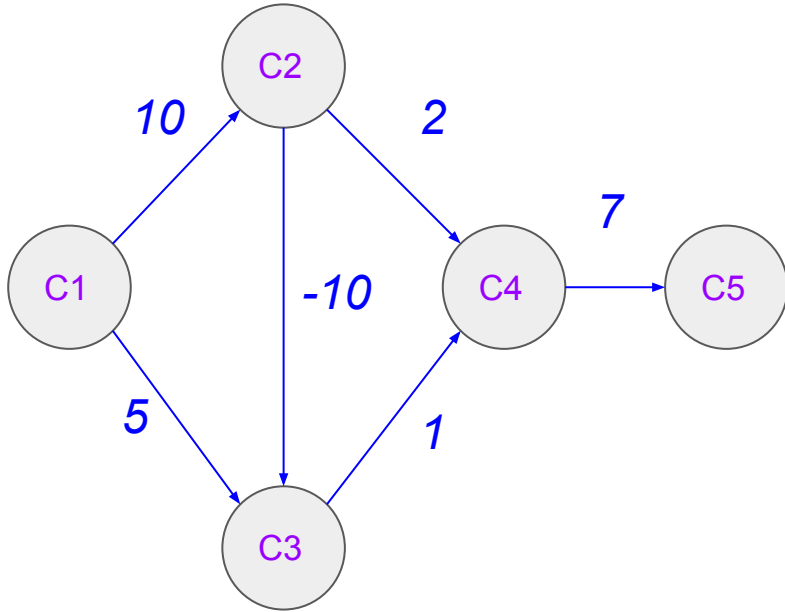
DP cannot handle cycles



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

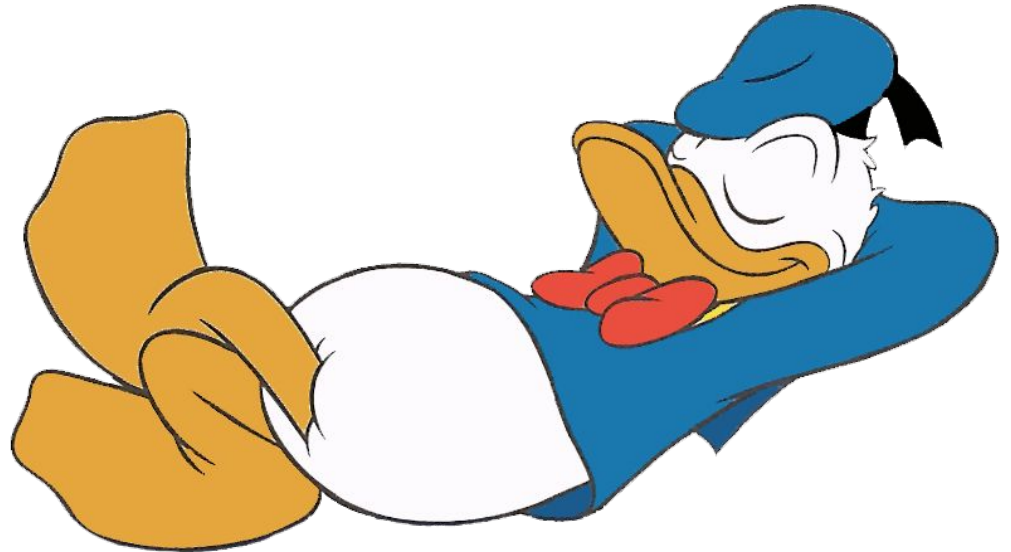
- $\text{Cost}'(s, a) = \text{Cost}(s, a) + h(\text{Succ}(s, a)) - h(s) \geq 0$
- $h(s_{\text{end}}) = 0$.

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.

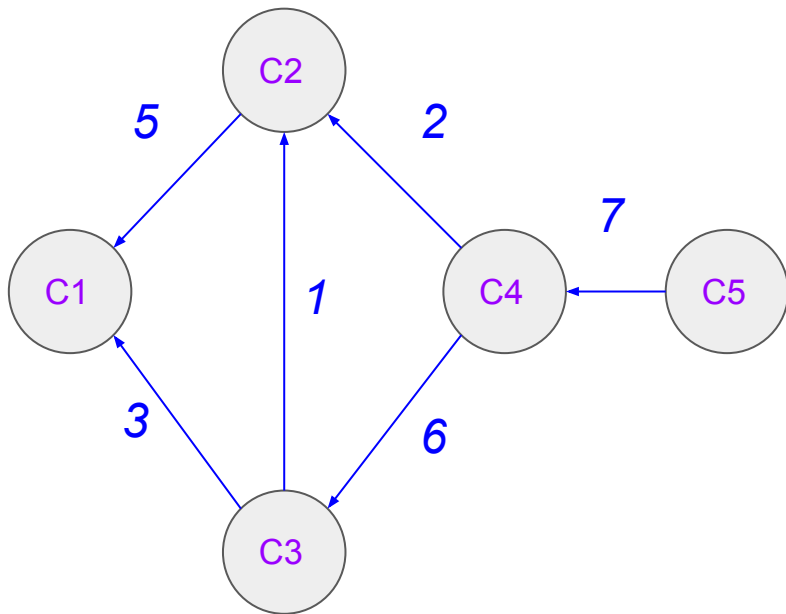
$h(s) = h((i, d)) = \text{length of shortest path from city } i \text{ to city } N$

Note that the modified shortest path problem has $O(N)$ states instead of $O(N^2)$.

Checking consistency

- $\text{Cost}(s, a) + h(\text{Succ}(s, a)) - h(s) \geq 0$ (Triangle Inequality)
 - Suppose $s = (C_i, d)$ and $\text{Succ}(s, a) = (C_j, d')$
 - Note that $h((C_i, d)) - h((C_j, d')) \leq c(C_i, C_j) = \text{Cost}(s, a)$
- $h((C_N, d)) = 0$

How to compute h ?

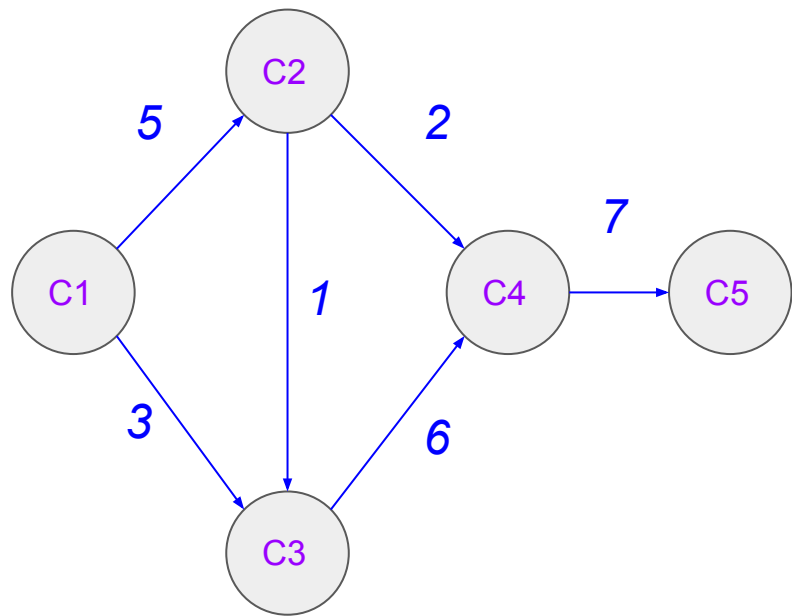


Reverse all edges, then perform UCS starting at C5 until C1 is found.

→ $O(n \log n)$ 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 |

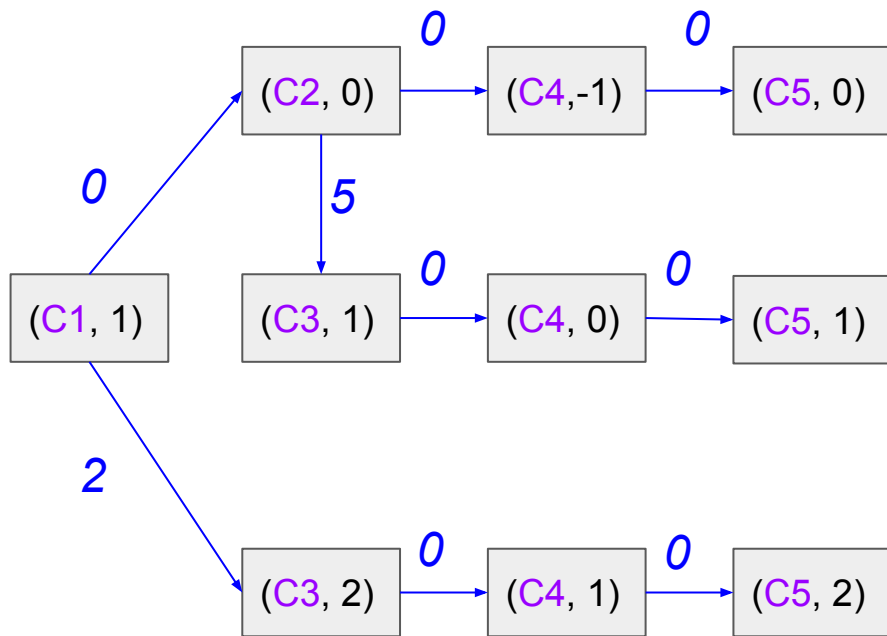
Original Graph



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

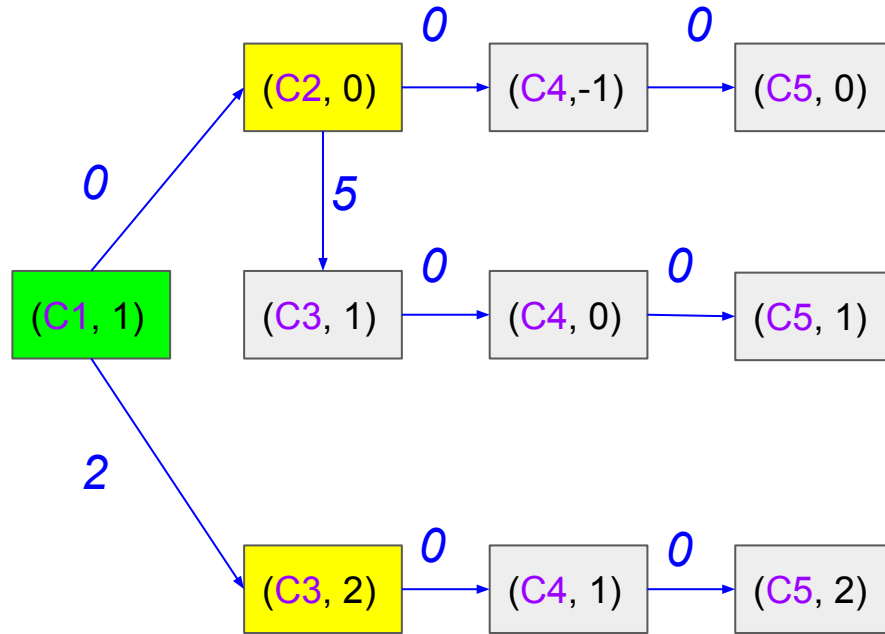
Modified State Graph

(updated edge costs)



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

Simulation of UCS (A*)

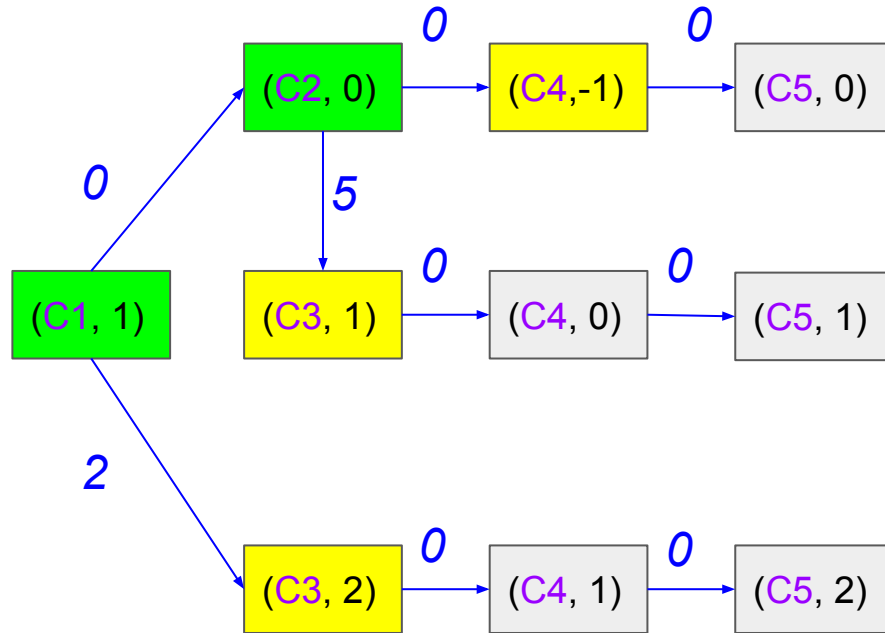


State $s = (i, d)$ (current city, #odd-#even)

Explored:
 $(C1, 1) : 0$

Frontier:
 $(C2, 0) : 0$
 $(C3, 2) : 2$

Simulation of UCS (A*)



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

Explored:

$(C1, 1) : 0$

$(C2, 0) : 0$

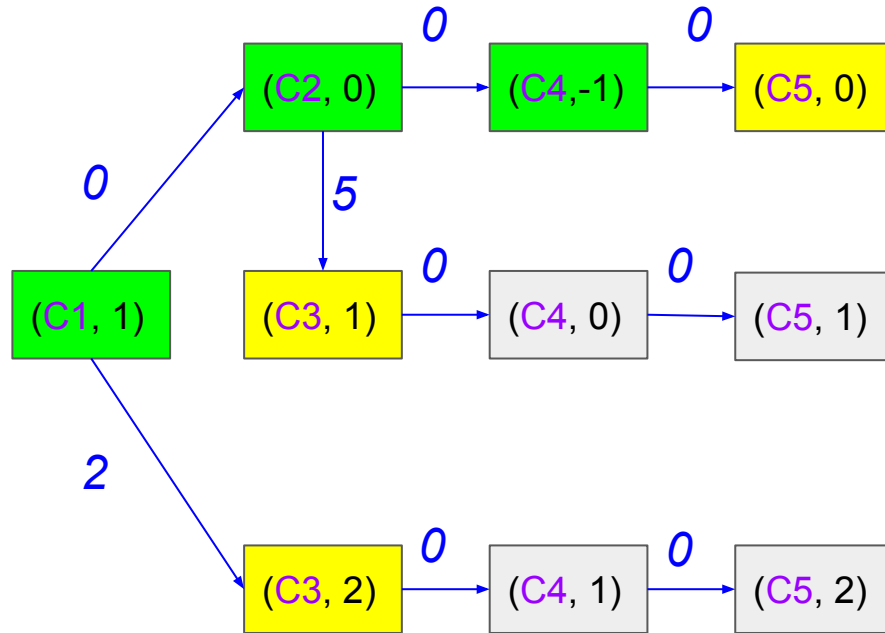
Frontier:

$(C4, -1) : 0$

$(C3, 2) : 2$

$(C3, 1) : 5$

Simulation of UCS (A*)

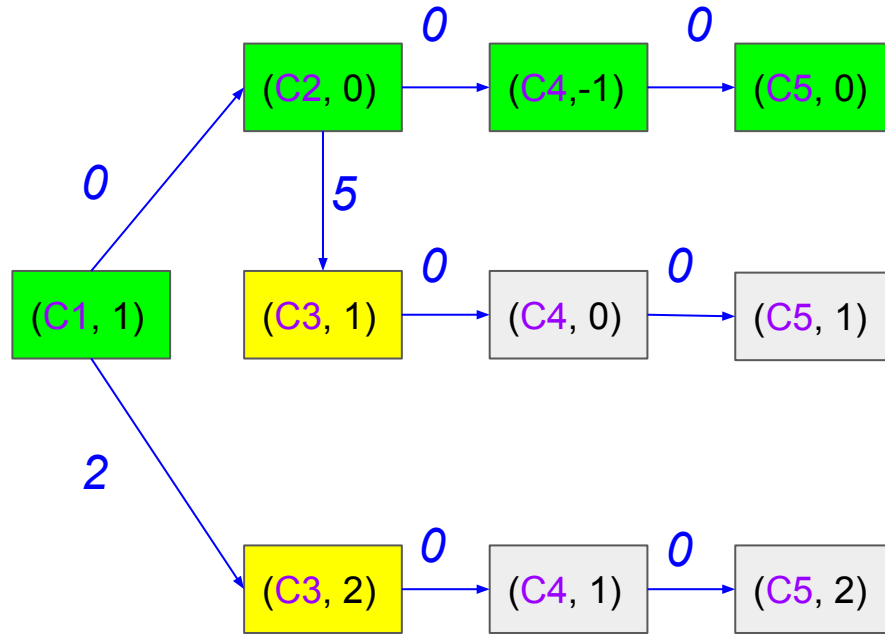


State $s = (i, d)$ (current city, #odd-#even)

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

Frontier:
 $(C5, 0) : 0$
 $(C3, 2) : 2$
 $(C3, 1) : 5$

Simulation of UCS (A*)



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

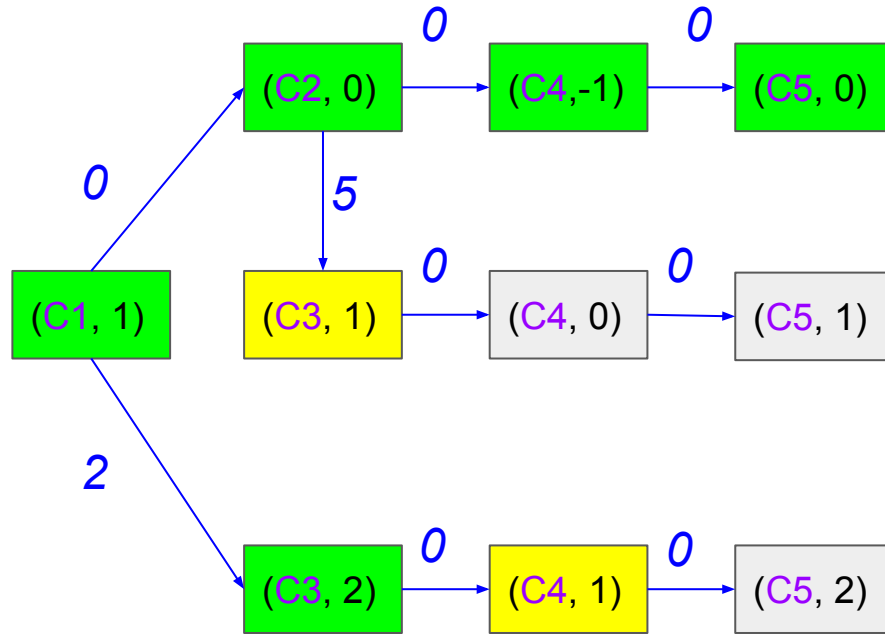
Explored:

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

Frontier:

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

Simulation of UCS (A*)



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

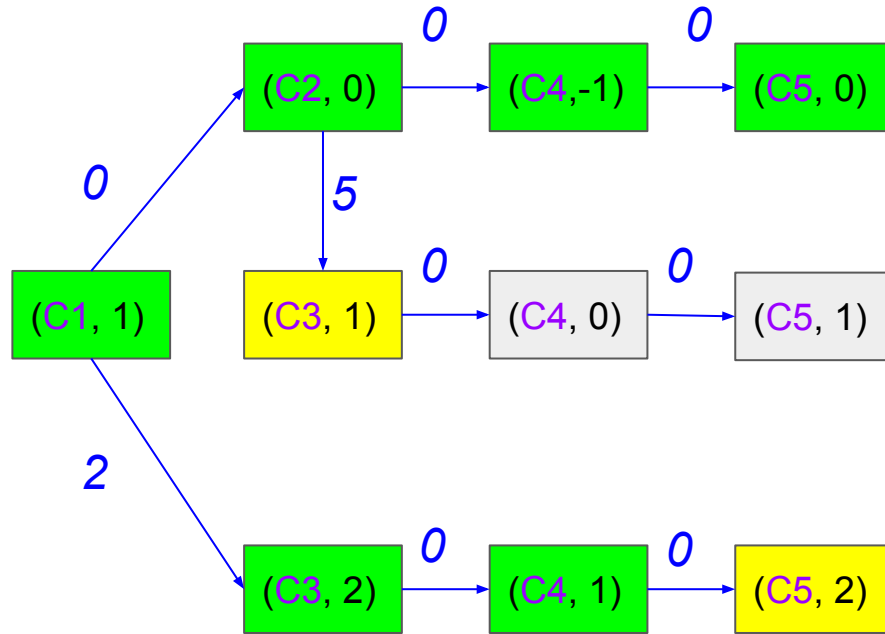
Explored:

(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)

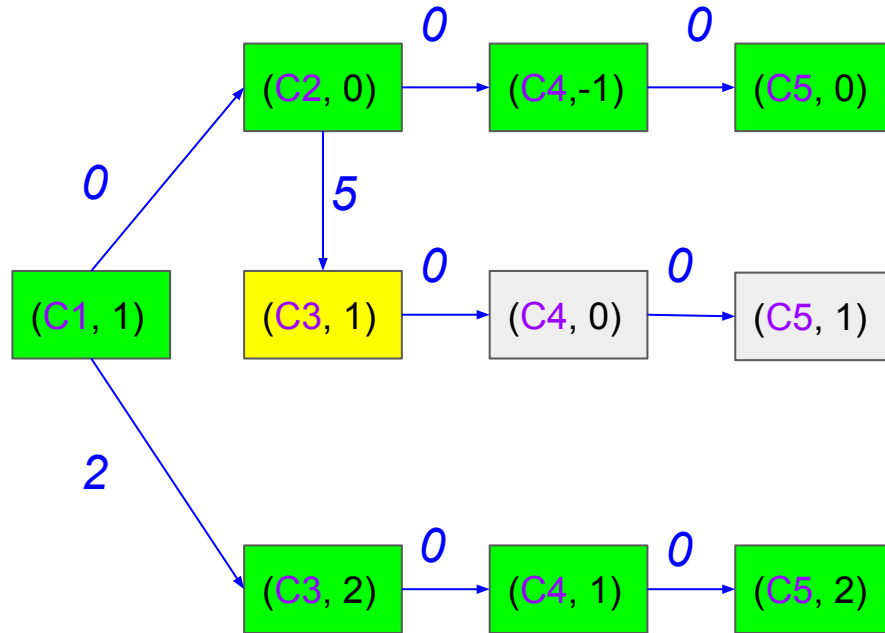
Explored:

(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*)



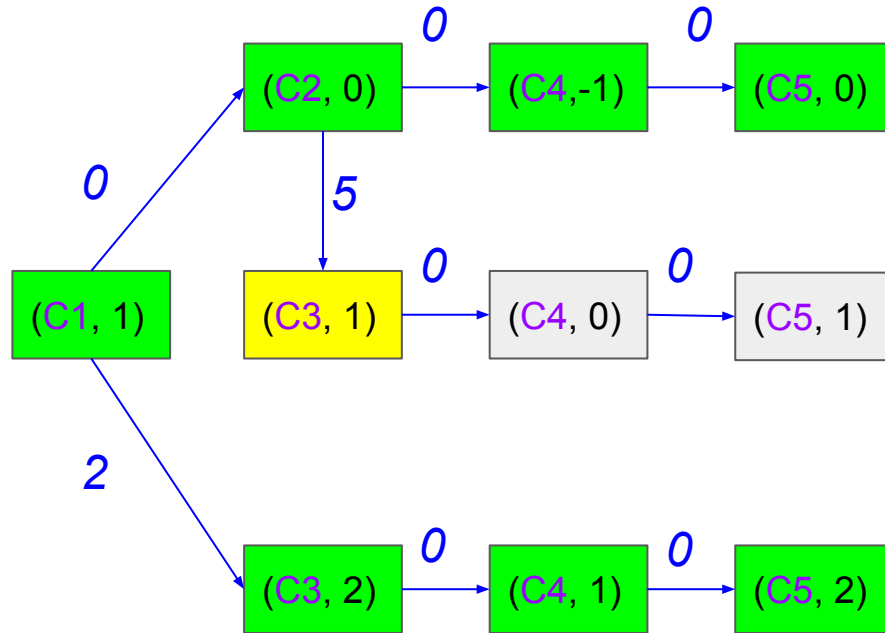
State $s = (i, d)$ (current city, #odd-#even)

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: | Frontier: |
|--------------|--------------|
| (C1, 1) : 0 | (C5, 1) : 19 |
| (C3, 2) : 3 | |
| (C2, 0) : 5 | |
| (C3, 1) : 6 | |
| (C4, -1) : 7 | |
| (C4, 1) : 9 | |
| (C4, 0) : 12 | |
| (C5, 0) : 14 | |
| (C5, 2) : 16 | |

UCS(A*)

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

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

That's it! Thanks for coming to section!

