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 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

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

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

Definition: consistency

A heuristic $h$ is consistent if

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

**Condition 1:** needed for UCS to work (triangle inequality).

**Condition 2:** $\text{FutureCost}(s_{\text{end}}) = 0$ so match it.
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:

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

**Efficiency:**

\[ h(s) = \text{FutureCost}_{rel}(s) \] must be easy to compute

Closed form, easier search, independent subproblems

**Tightness:**

heuristic \( h(s) \) should be close to \( \text{FutureCost}(s) \)

Don’t remove too many constraints
Which heuristic would you use to solve our problem more efficiently? 

*Hint: Relaxation!*
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?
Original Graph

State Graph

State $s = (i, d)$ (current city, #odd-#even)
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 \( O(N^2) \).
How to compute $h$?

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

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

<table>
<thead>
<tr>
<th>city</th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
<th>C5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h$</td>
<td>14</td>
<td>9</td>
<td>13</td>
<td>7</td>
<td>0</td>
</tr>
</tbody>
</table>
Original Graph

<table>
<thead>
<tr>
<th>city</th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
<th>C5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h$</td>
<td>14</td>
<td>9</td>
<td>13</td>
<td>7</td>
<td>0</td>
</tr>
</tbody>
</table>

Modified State Graph
(updated edge costs)

State $s = (i, d)$ (current city, #odd-#even)
Simulation of UCS (A*)

Explored:
(C1, 1) : 0
Frontier:
(C2, 0) : 0
(C3, 2) : 2

State $s = (i, d)$ (current city, #odd-#even)
Simulation of UCS (A*)

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

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

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

(C1, 1) → (C2, 0) : 0 → (C4, -1) : 0 → (C5, 0)
(C3, 1) → (C4, 0) : 0 → (C5, 1)
(C3, 2) → (C4, 1) : 0 → (C5, 2)
Simulation of UCS (A*)

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

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

State \( s = (i, d) \) (current city, #odd-#even)
Simulation of UCS (A*)

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

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

State $s = (i, d)$ (current city, #odd-#even)
Simulation of UCS (A*)

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

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

State $s = (i, d)$ (current city, #odd-#even)
Simulation of UCS (A*)

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

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

State $s = (i, d)$ (current city, #odd-#even)
Simulation of UCS (A*)

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

Frontier:
- (C3, 1) : 5

STOP!

State $s = (i, d)$ (current city, #odd-#even)
Simulation of UCS (A*)

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

Frontier:
(C3, 1) : 5

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

State $s = (i, d)$ (current city, #odd-#even)
Comparison of States visited

<table>
<thead>
<tr>
<th>Explored:</th>
<th>UCS</th>
<th>UCS(A*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(C1, 1): 0</td>
<td>(C1, 1): 0</td>
<td>(C1, 1): 0</td>
</tr>
<tr>
<td>(C3, 2): 3</td>
<td>(C2, 0): 0</td>
<td>(C2, 0): 0</td>
</tr>
<tr>
<td>(C2, 0): 5</td>
<td>(C4, -1): 0</td>
<td>(C4, -1): 0</td>
</tr>
<tr>
<td>(C3, 1): 6</td>
<td>(C5, 0): 0</td>
<td>(C5, 0): 0</td>
</tr>
<tr>
<td>(C4, -1): 7</td>
<td>(C3, 2): 2</td>
<td>(C3, 2): 2</td>
</tr>
<tr>
<td>(C4, 1): 9</td>
<td>(C4, 1): 2</td>
<td>(C4, 1): 2</td>
</tr>
<tr>
<td>(C4, 0): 12</td>
<td>(C4, 1): 2</td>
<td>(C5, 2): 2</td>
</tr>
<tr>
<td>(C5, 0): 14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(C5, 2): 16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frontier:</td>
<td>(C5, 1): 19</td>
<td>(C3, 1): 5</td>
</tr>
</tbody>
</table>

Frontier: (C5, 1): 19

Frontier: (C3, 1): 5
## Comparison of States visited

<table>
<thead>
<tr>
<th></th>
<th>UCS</th>
<th>UCS(A*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explored:</td>
<td>(C1, 1) : 0</td>
<td>(C1, 1) : 0</td>
</tr>
<tr>
<td></td>
<td>(C3, 2) : 3</td>
<td>(C2, 0) : 0</td>
</tr>
<tr>
<td></td>
<td>(C2, 0) : 5</td>
<td>(C4, -1) : 0</td>
</tr>
<tr>
<td></td>
<td>(C3, 1) : 6</td>
<td>(C5, 0) : 0</td>
</tr>
<tr>
<td></td>
<td>(C4, -1) : 7</td>
<td>(C3, 2) : 2</td>
</tr>
<tr>
<td></td>
<td>(C4, 1) : 9</td>
<td>(C4, 1) : 2</td>
</tr>
<tr>
<td></td>
<td>(C4, 0) : 12</td>
<td>(C5, 2) : 2</td>
</tr>
<tr>
<td></td>
<td>(C5, 0) : 14</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(C5, 2) : 16</td>
<td></td>
</tr>
</tbody>
</table>

**UCS explored 9 states**

**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!
MDPs: overview
Markov decision process

Definition: Markov decision process

States: the set of states
- $s_{\text{start}} \in \text{States}$: starting state

Actions($s$): possible actions from state $s$

$T(s' | s, a)$: probability of $s'$ if take action $a$ in state $s$

Reward($s, a, s'$): reward for the transition ($s, a, s'$)

IsEnd($s$): whether at end

$0 \leq \gamma \leq 1$: discount factor (default: 1)
What is a solution?

Search problem: path (sequence of actions)

MDP:

**Definition: policy**

A *policy* $\pi$ is a mapping from each state $s \in \text{States}$ to an action $a \in \text{Actions}(s)$.

**Example: volcano crossing**

<table>
<thead>
<tr>
<th>$s$</th>
<th>$\pi(s)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1,1)</td>
<td>S</td>
</tr>
<tr>
<td>(2,1)</td>
<td>E</td>
</tr>
<tr>
<td>(3,1)</td>
<td>N</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
MDPs: policy evaluation
## Discounting

**Definition: utility**

Path: $s_0, a_1 r_1 s_1, a_2 r_2 s_2, \ldots$ (action, reward, new state).

The utility with discount $\gamma$ is

$$u_1 = r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 r_4 + \cdots$$

Discount $\gamma = 1$ (save for the future):

[stay, stay, stay, stay]: $4 + 4 + 4 + 4 = 16$

Discount $\gamma = 0$ (live in the moment):

[stay, stay, stay, stay]: $4 + 0 \cdot (4 + \cdots) = 4$

Discount $\gamma = 0.5$ (balanced life):

[stay, stay, stay, stay]: $4 + \frac{1}{2} \cdot 4 + \frac{1}{4} \cdot 4 + \frac{1}{8} \cdot 4 = 7.5$
Policy evaluation

**Definition: value of a policy**

Let $V_\pi(s)$ be the expected utility received by following policy $\pi$ from state $s$.

**Definition: Q-value of a policy**

Let $Q_\pi(s, a)$ be the expected utility of taking action $a$ from state $s$, and then following policy $\pi$.
Policy evaluation

Plan: define recurrences relating value and Q-value

\[ V_\pi(s) = \begin{cases} 
0 & \text{if IsEnd}(s) \\
Q_\pi(s, \pi(s)) & \text{otherwise.} 
\end{cases} \]

\[ Q_\pi(s, a) = \sum_{s'} T(s'|s, a)[\text{Reward}(s, a, s') + \gamma V_\pi(s')] \]
Policy evaluation

Key idea: iterative algorithm

Start with arbitrary policy values and repeatedly apply recurrences to converge to true values.

Algorithm: policy evaluation

Initialize $V_\pi^{(0)}(s) \leftarrow 0$ for all states $s$.

For iteration $t = 1, \ldots, t_{PE}$:

For each state $s$:

$$V_\pi^{(t)}(s) \leftarrow \sum_{s'} T(s'|s, \pi(s))[\text{Reward}(s, \pi(s), s') + \gamma V_\pi^{(t-1)}(s')]$$
MDPs: value iteration
Optimal value and policy

Goal: try to get directly at maximum expected utility

Definition: optimal value

The optimal value $V_{\text{opt}}(s)$ is the maximum value attained by any policy.
Optimal values and Q-values

Optimal value if take action $a$ in state $s$:

$$Q_{opt}(s, a) = \sum_{s'} T(s, a, s')[\text{Reward}(s, a, s') + \gamma V_{opt}(s')]$$.

Optimal value from state $s$:

$$V_{opt}(s) = \begin{cases} 0 & \text{if IsEnd}(s) \\ \max_{a \in \text{Actions}(s)} Q_{opt}(s, a) & \text{otherwise}. \end{cases}$$
Optimal policies

Given $Q_{opt}$, read off the optimal policy:

$$\pi_{opt}(s) = \arg\max_{a \in \text{Actions}(s)} Q_{opt}(s, a)$$
Value iteration

Algorithm: value iteration [Bellman, 1957]

Initialize $V^{(0)}_{opt}(s) \leftarrow 0$ for all states $s$.
For iteration $t = 1, \ldots, t_{VI}$:
For each state $s$:

\[
V^{(t)}_{opt}(s) \leftarrow \max_{a \in \text{Actions}(s)} \sum_{s'} T(s, a, s')[\text{Reward}(s, a, s') + \gamma V^{(t-1)}_{opt}(s')]
\]

Time: $O(t_{VI} SAS')$
Convergence

**Theorem: convergence**

Suppose either

- discount $\gamma < 1$, or
- MDP graph is acyclic.

Then value iteration converges to the correct answer.

**Example: non-convergence**

discount $\gamma = 1$, zero rewards
Summary of algorithms

- **Policy evaluation**: $(\text{MDP, } \pi) \rightarrow V_\pi$

- **Value iteration**: $\text{MDP} \rightarrow (Q_{\text{opt}}, \pi_{\text{opt}})$
MDPs: reinforcement learning