# Informed Search

Dr. Neil T. Dantam

CSCI-498/598 RPM, Colorado School of Mines

Spring 2018

# Outline

Planning and Search Problems

Dijkstra's Algorithm

Optimality and Heuristics

Greedy Search

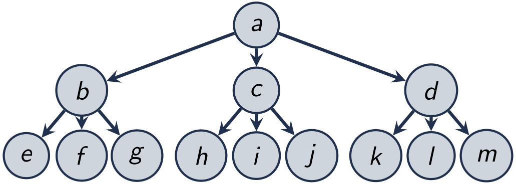
$A^*$ Search

# The Planning / Search Problem

Given:
1. State space: $\mathcal{Q}$
2. Transition function $\delta : \mathcal{Q} \mapsto \mathcal{P}(\mathcal{Q})$
3. Start state: $q_0 \in \mathcal{Q}$
4. Goal set: $A \subseteq \mathcal{Q}$

Find: Path $p = (p_0, \ldots, p_n)$ from start to goal such that:

- $p_0 = q_0$ is the start state
- $p_n \in A$ is a goal state
- Subsequent states are valid transitions: $p_{k+1} \in \delta(p_k)$



# Forward Search

**Procedure** search($f_{\text{ins}}, f_{\text{rem}}, q_0, \delta, A$)

```
T[q0] ← nil; // Search Tree
W ← f_ins(q0, nil); // Frontier
while W do
    let q = f_rem(W) in
        if q ∈ A then
            return tree-path(T, q);
        else
            foreach q' ∈ δ(q) do
                if ¬contains(T,q') then
                    T[q'] ← q;
                    W ← f_ins(q', W);
return nil;
```

**Procedure** depth-first-search($q_0, \delta, A$)

```
return search(push, pop, q0, δ, A);
```

**Procedure** breadth-first-search($q_0, \delta, A$)

```
return search(enqueue, dequeue, q0, δ, A);
```

# Planning Properties

Correctness: Do we get a right answer?

Completeness: Do we always get an answer?

Optimality: Do we get the best answer?

# Optimality

## Definition (Optimality)

A planning algorithm is optimal if it produces the lowest cost (/ highest reward) plan.

### State Cost

- $\delta : \mathcal{Q} \mapsto \mathcal{P}(\mathcal{Q})$
- $q_{i+1} \in \delta(q_i)$
- $C : \mathcal{Q} \mapsto \mathbb{R}$
- $p = \underset{p_0,\ldots,p_n}{\operatorname{argmin}} \left(\sum_{i=0}^{n} C(p_i)\right)$

### Transition Cost

- $\delta : \mathcal{Q} \mapsto \mathcal{P}(\mathcal{Q})$
- $q_{i+1} \in \delta(q_i)$
- $C : \mathcal{Q} \times \mathcal{Q} \mapsto \mathbb{R}$
- $p = \underset{p_0,\ldots,p_n}{\operatorname{argmin}} \left(\sum_{i=0}^{n-1} C(p_i, p_{i+1})\right)$

### Action Cost

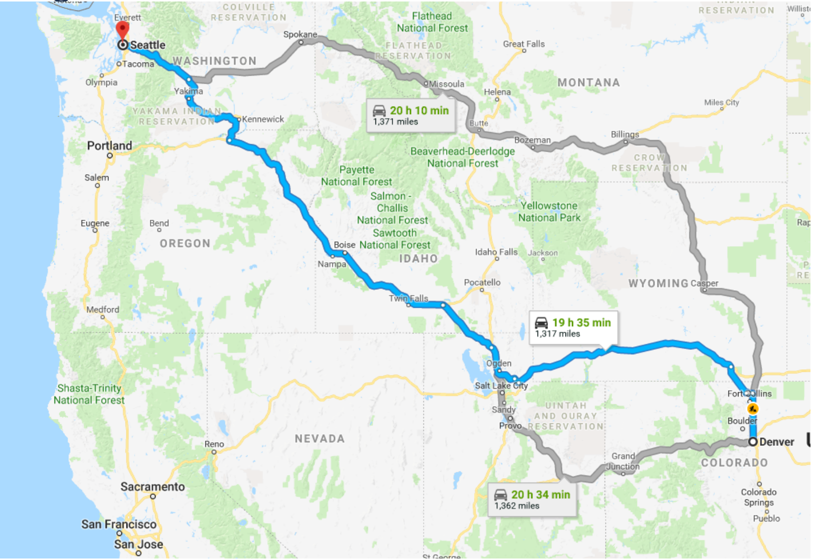
- $\delta : \overbrace{\mathcal{Q}}^{\text{state}} \times \overbrace{\mathcal{U}}^{\text{action}} \mapsto \mathcal{Q}$
- $q_{i+1} = \delta(q_i, u_i)$
- $C : \mathcal{U} \mapsto \mathbb{R}$
- $u = \underset{u_0,\ldots,u_n}{\operatorname{argmin}} \left(\sum_{i=0}^{n} C(u_i)\right)$

# Outline

# Dijkstra's Algorithm Overview

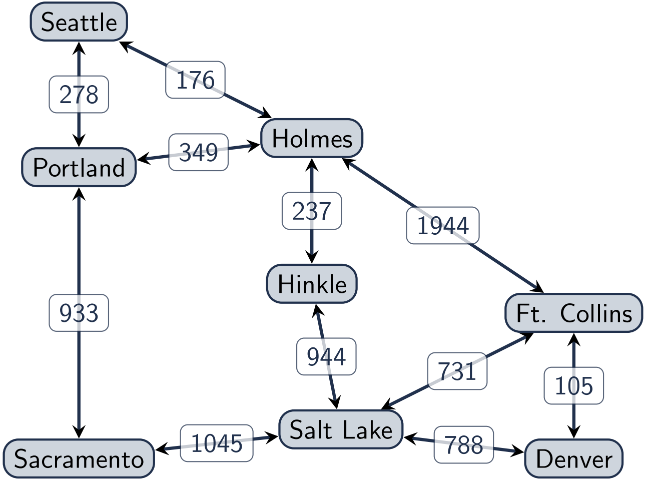
1. Store node work list in a priority queue
2. Order priority queue by cost to reach node
3. Each iteration, visit the least-cost unexplored node

# Example: Navigation



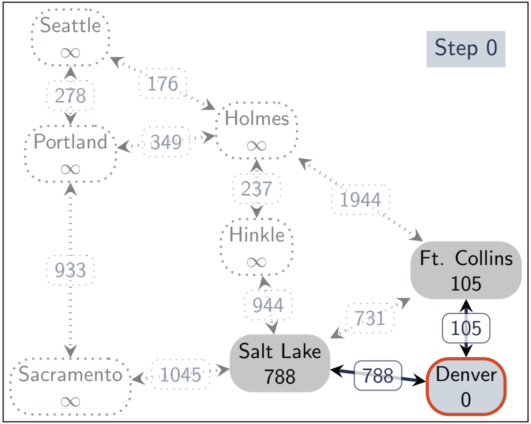
# Example: Dijkstra's Algorithm

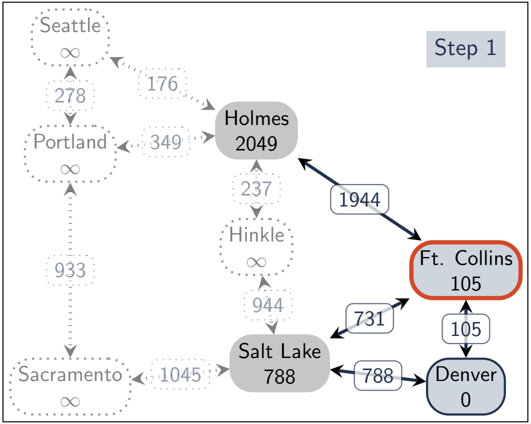
## Graph



# Example: Dijkstra's Algorithm

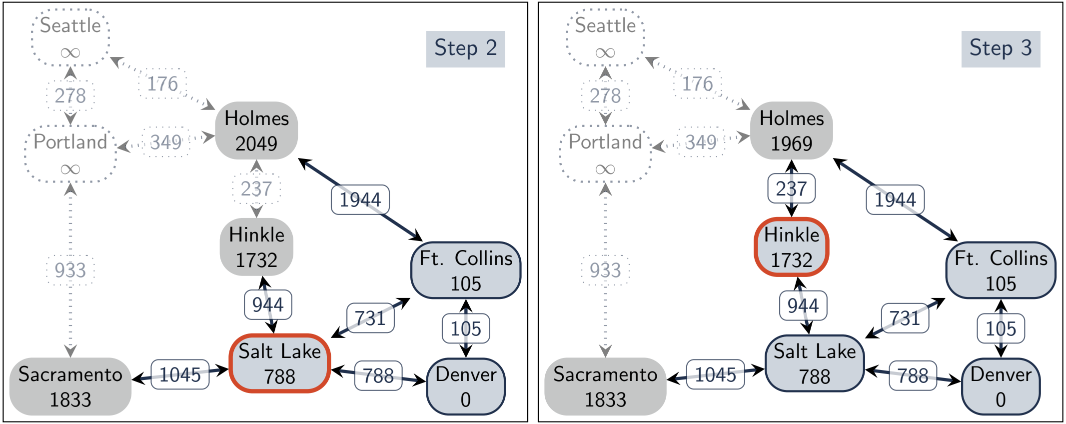
## continued – 1



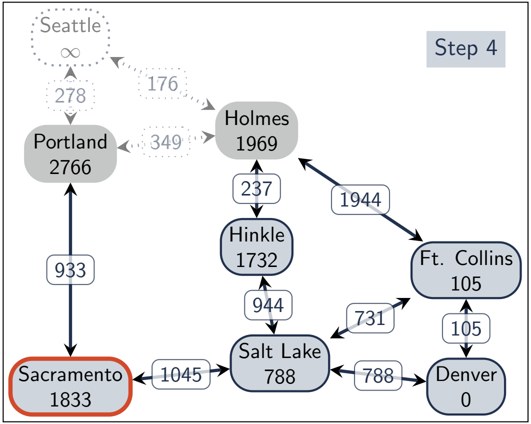
# Example: Dijkstra's Algorithm

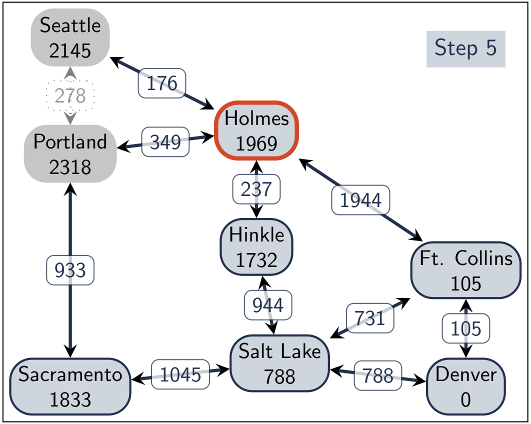
continued – 2



# Example: Dijkstra's Algorithm

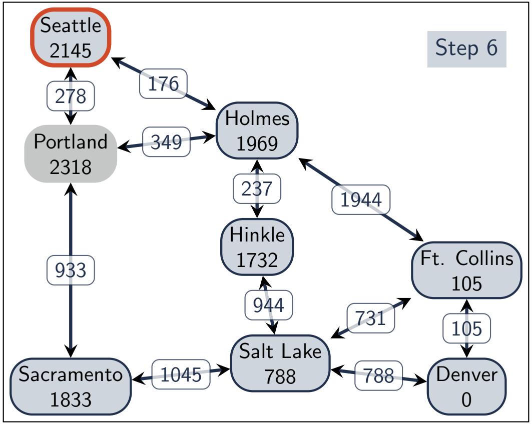
continued – 2



# Example: Dijkstra's Algorithm

continued – 3



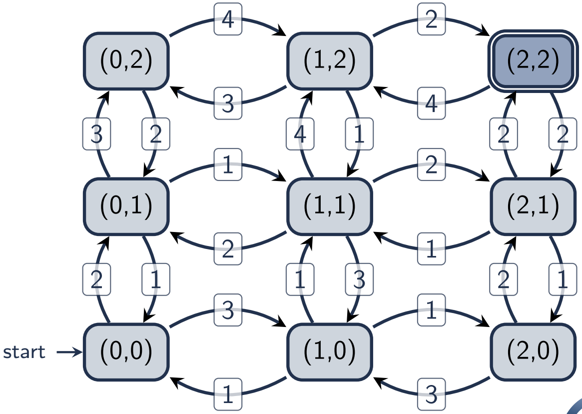
# Dijkstra's Algorithm – Transition Cost

**Procedure** djikstra-transition($C, q_0, \delta, A$)

```
T[q0] ← nil; // Search Tree: node ↦ parent
D[q0] ← 0; // Search Tree: node ↦ path cost
W ← insert (q0, 0); // Priority Queue
while W do
    let q = remove-min (W) in
        if q ∈ A then return tree-path (T, q) ;
        else
            foreach q′ ∈ δ(q) do
                let d′ = D[q] + C(q, q′) in
                    if ¬contains(T,q′) ∨ (d′ < D[q′]) then
                        T[q′] ← q;
                        D[q′] ← d′;
                        W ← insert (q′, d′);
return nil;
```

# Exercise: Dijkstra's Algorithm

| | | | |
|---|---|---|---|
| 2 | 3 | 4 | 2 |
| 1 | 2 | 1 | 2 |
| 0 | 1 | 3 | 1 |
| | 0 | 1 | 2 |



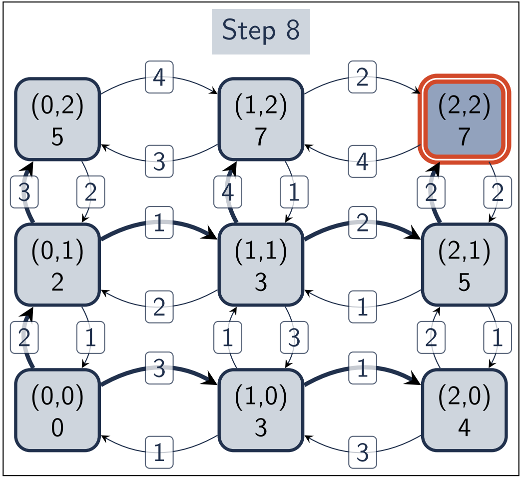
# Exercise: Dijkstra's Algorithm

# Exercise: Dijkstra's Algorithm

# Exercise: Dijkstra's Algorithm

# Exercise: Dijkstra's Algorithm

# Exercise: Dijkstra's Algorithm



# Dijkstra's Algorithm – Transition vs. State Cost

**Procedure** djikstra-transition$(C, q_0, \delta, A)$

$T[q_0] \leftarrow$ **nil**; `// Search Tree: node` $\mapsto$ `parent`
$D[q_0] \leftarrow 0$; `// Search Tree: node` $\mapsto$ `path cost`
$W \leftarrow$ `insert` $(q_0, 0)$; `// Priority Queue`
**while** $W$ **do**
  **let** $q =$ `remove-min` $(W)$ **in**
    **if** $q \in A$ **then** **return** `tree-path` $(T, q)$ ;
    **else**
      **foreach** $q' \in \delta(q)$ **do**
        **let** $d' = D[q] + \overbrace{C(q, q')}^{\text{transition cost}}$ **in**
          **if** $\neg$`contains`$(T, q')$ $\vee$ $(d' < D[q'])$ **then**
            $T[q'] \leftarrow q$;
            $D[q'] \leftarrow d'$;
            $W \leftarrow$ `insert` $(q', d')$;
**return** **nil**;

**Procedure** djikstra-state$(C, q_0, \delta, A)$

$T[q_0] \leftarrow$ **nil**; `// Search Tree: node` $\mapsto$ `parent`
$D[q_0] \leftarrow 0$; `// Search Tree: node` $\mapsto$ `path cost`
$W \leftarrow$ `insert` $(q_0, C(q_0))$; `// Priority Queue`
**while** $W$ **do**
  **let** $q =$ `remove-min` $(W)$ **in**
    **if** $q \in A$ **then** **return** `tree-path` $(T, q)$ ;
    **else**
      **foreach** $q' \in \delta(q)$ **do**
        **let** $d' = D[q] + \overbrace{C(q')}^{\text{state cost}}$ **in**
          **if** $\neg$`contains`$(T, q')$ $\vee$ $(d' < D[q'])$ **then**
            $T[q'] \leftarrow q$;
            $D[q'] \leftarrow d'$;
            $W \leftarrow$ `insert` $(q', d')$;
**return** **nil**;

# Exercise: Dijkstra's Algorithm – Action Cost

Transition Function:

$$\delta : \overbrace{\mathcal{Q}}^{\text{predecessor}} \times \overbrace{\mathcal{U}}^{\text{action}} \mapsto \overbrace{\mathcal{Q}}^{\text{sucessor}}$$

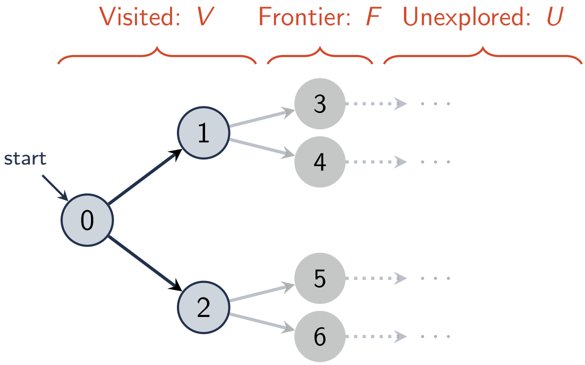
Cost Function:

$$C : \mathcal{U} \mapsto \mathbb{R}$$

# Dijkstra's Algorithm is Optimal

**Proof Outline**

Proof by induction: For each iteration of the loop, we visit the next unexplored node with least cost from $q_0$. $\square$




- $\tilde{C}(q_i, q_j) = \min\left(\sum_{i=0}^{n-1} C(p_i, p_{i+1})\right)$, where:
  - $p_0 = q_i$
  - $p_n = q_j$
  - $p_{i+1} \in \delta p_i$
- $\forall q_v \in V,\ \forall q_f \in F,\ \tilde{C}(q_0, q_v) \leq \tilde{C}(q_0, q_f)$
- $\forall q_v \in V,\ \forall q_u \in U,\ \tilde{C}(q_0, q_v) \leq \tilde{C}(q_0, q_u)$

# Edsger Wybe Dijkstra

"The question of whether a computer can think
is no more interesting
than the question of whether a submarine can swim."

–Edsger W. Dijkstra

# Outline

# Bellman's Principle of Optimality

**Definition: Principle of Optimality**

"An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision."

Richard Bellman

Goal State: $\forall p_a \in A, \quad \overbrace{f_{\mathrm{opt}}(p_a) = 0}^{\text{zero cost at goal}}$

Non-Goal State: $f_{\mathrm{opt}}(p_i) = \overbrace{C(p_i, \quad \underbrace{\delta_{\mathrm{opt}}(p_i)}_{\text{optimal decision}} \quad )}^{\text{cost of optimal decision}} \quad + \quad \overbrace{f_{\mathrm{opt}}(\delta_{\mathrm{opt}}(p_i))}^{\text{cost of remaining decisions}}$

*Optimal Cost: As hard to find as optimal decision/policy*

# Heuristic

**Definition: Heuristic**

An approximation used to quickly find a solution.

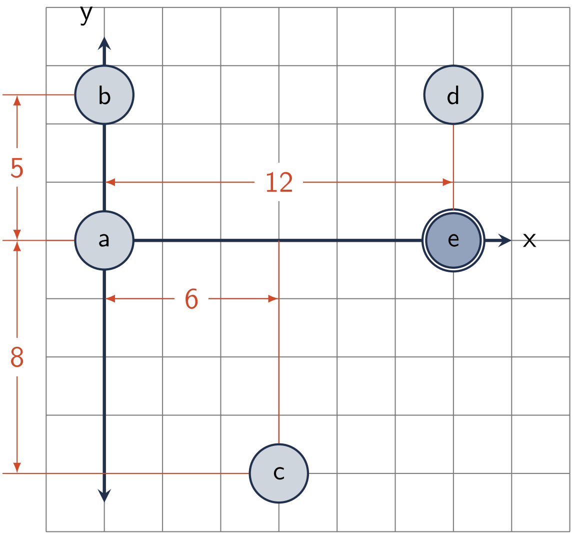
In search, an approximate ranking of branches at each step:

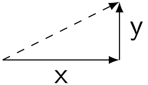
$$h : \mathcal{Q} \mapsto \mathbb{R}$$

From the Greek $\varepsilon \upsilon \rho \iota \sigma \kappa \omega$ “find” or “discover”.

*Heuristic approximates cost-to-go $f_{\mathrm{opt}}(p_i)$*

# Example: Euclidean distance heuristic



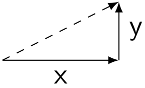




$$d_{\text{euclidean}} = \sqrt{x^2 + y^2}$$

- $h(a) = \|e - a\|_2 = \sqrt{12^2 + 0^2} = 12$
- $h(b) = \|e - b\|_2 = \sqrt{5^2 + 12^2} = 13$
- $h(c) = \|e - c\|_2 = \sqrt{6^2 + 8^2} = 10$
- $h(d) = \|e - d\|_2 = \sqrt{5^2 + 0^2} = 5$
- $h(e) = \|e - e\|_2 = \sqrt{0^2 + 0^2} = 0$

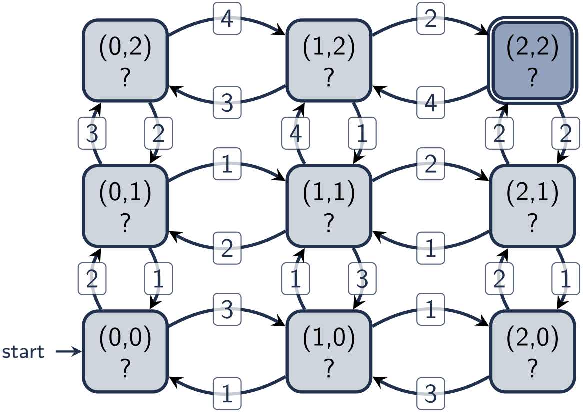
# Exercise: Manhattan Distance Heuristic

| | | |
|---|---|---|
| 3 | 4 | 2 |
| 2 | 1 | 2 |
| 1 | 3 | 1 |




$$d_{\text{euclidean}} = \sqrt{x^2 + y^2}$$

$$d_{\text{manhattan}} = |x| + |y|$$



# Historical Note

Solomon Lefschetz

John McCarthy

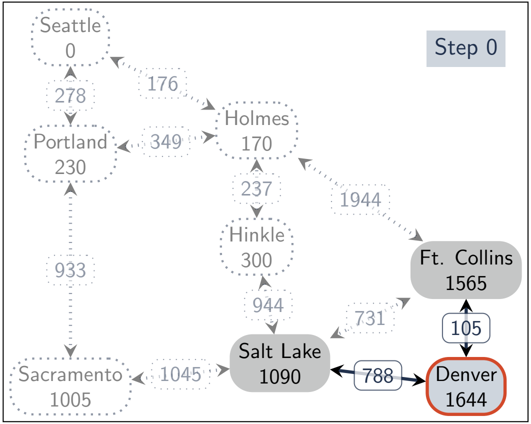
Richard Bellman

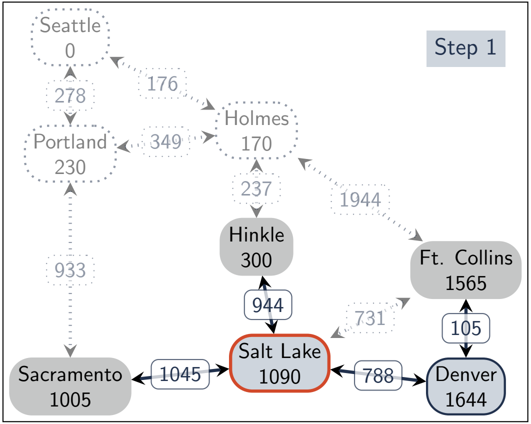
# Outline

# Greedy Search Overview

1. Store node work list in a priority queue
2. Order priority queue by heuristic cost-to-go
3. Each iteration, visit the least-heuristic-cost-to-go unexplored node

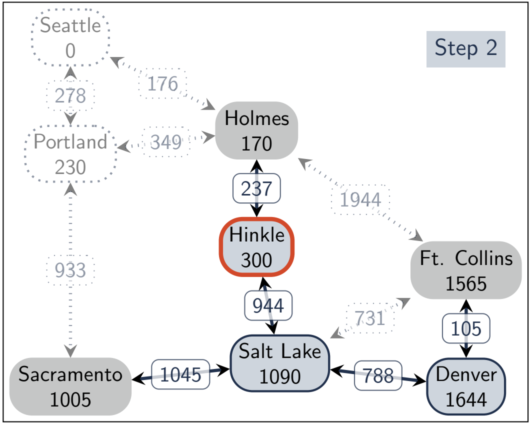
# Example: Greedy Search

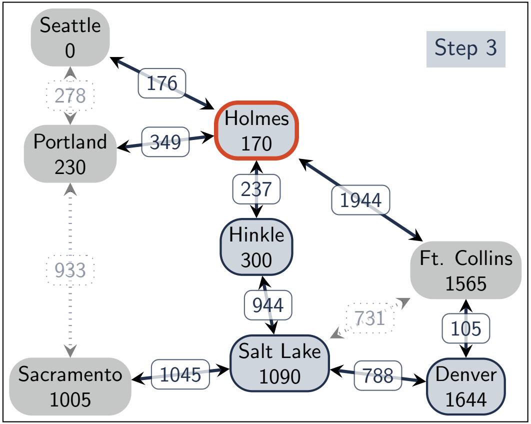


# Example: Greedy Search

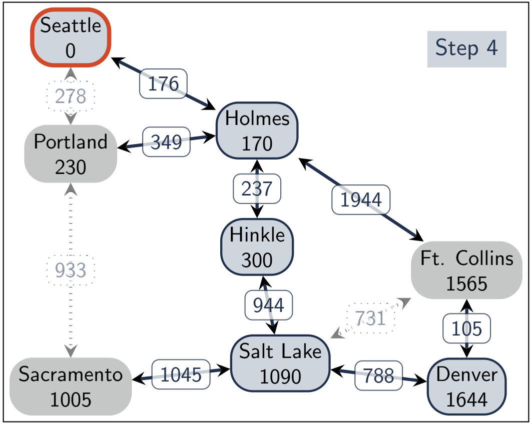
## continued – 1



# Example: Greedy Search

continued – 2



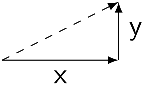
# Greedy Search Algorithm

**Procedure** greedy-search$(h, q_0, \delta, A)$

```
T[q0] ← nil; // Search Tree: node ↦ parent
W ← insert(q0, h(q0)); // Priority Queue
while W do
    let q = remove-min(W) in
        if q ∈ A then return tree-path(T, q);
        else
            foreach q′ ∈ δ(q) do
                if ¬contains(T,q′) then
                    T[q′] ← q;
                    W ← insert(q′, h(q′));
return nil;
```

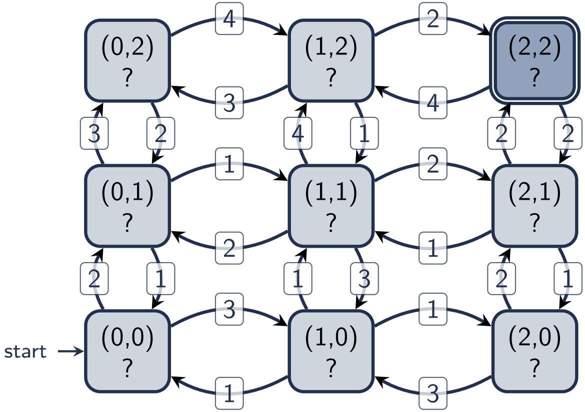
# Exercise: Greedy Search

| | | |
|---|---|---|
| 3 | 4 | 2 |
| 2 | 1 | 2 |
| 1 | 3 | 1 |




$$d_{\mathrm{euclidean}} = \sqrt{x^2 + y^2}$$

$$d_{\mathrm{manhattan}} = |x| + |y|$$



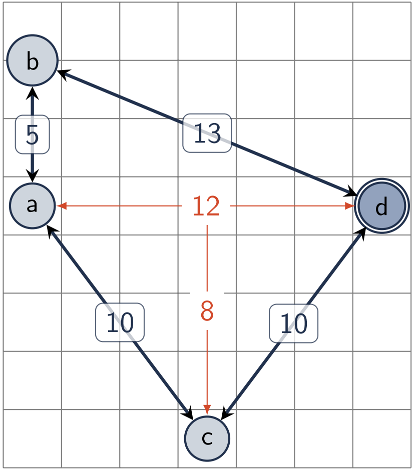
# Exercise: Greedy Search

# Exercise: Greedy Search

# Exercise: Greedy Search

# Greedy Search is Not Optimal

## Proof by Counterexample




Heuristic:
- $h(a) = 12$
- $h(b) = 13$
- $h(c) = 10$
- $h(d) = 0$

Optimal Path:
- $p = (a, b, d)$
- cost $= 18$

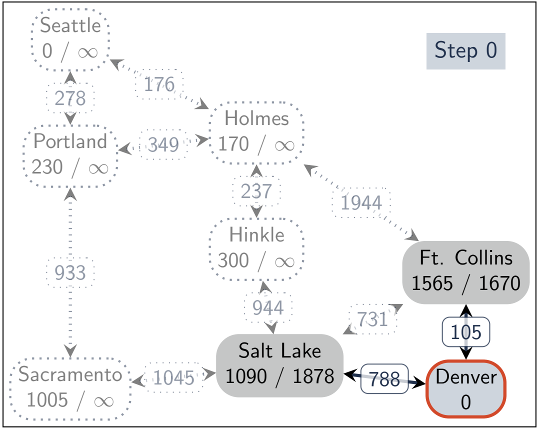
Greedy Path:
- $(a, c, d)$
- cost $= 20$

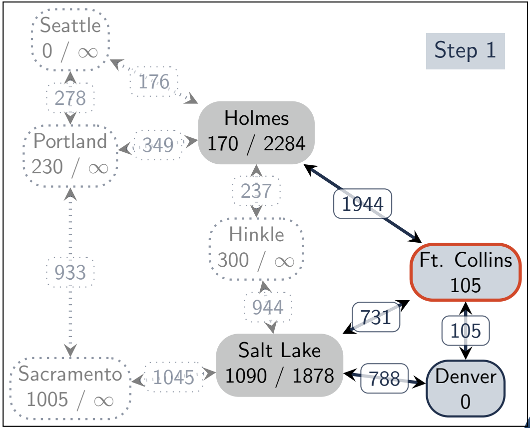
# Outline

# $A^*$ Search Overview

1. Store node work list in a priority queue
2. Order priority queue by sum of cost to reach node and heuristic cost-to-go
3. Each iteration, visit the least expected-total-cost unexplored node

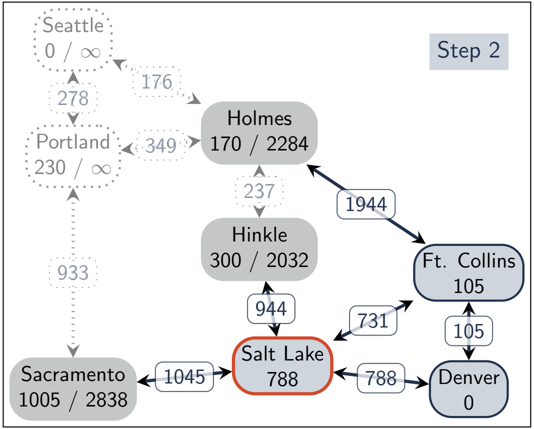
# Example: $A^*$ Search

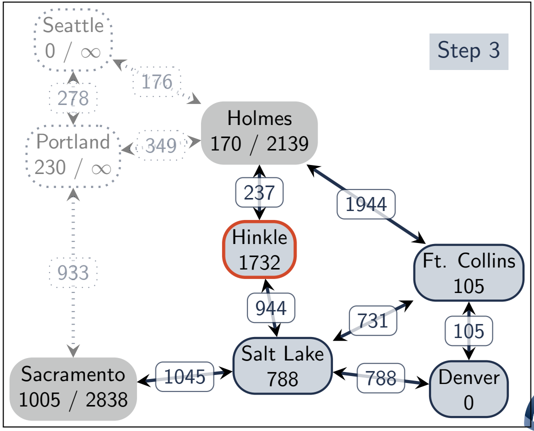


# Example: $A^*$ Search

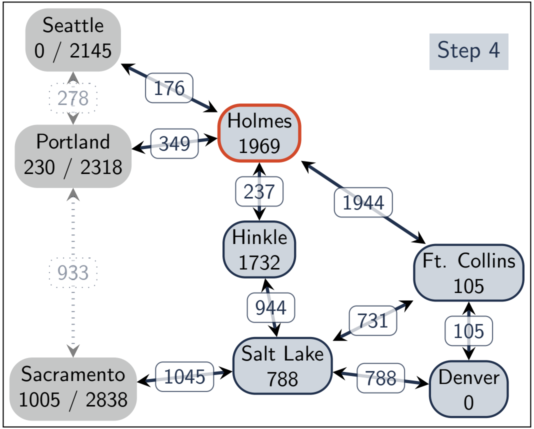
continued – 1

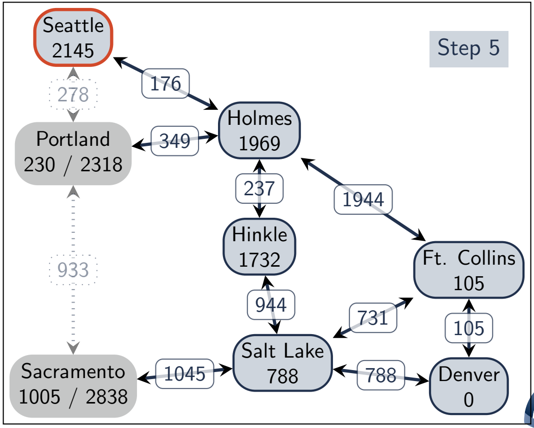


# Example: $A^*$ Search

continued – 2



# $A^*$ Search Algorithm

**Procedure** $A^*(C, h, q_0, \delta, A)$

$T[q_0] \leftarrow$ **nil**; `// Search Tree: node` $\mapsto$ `parent`

$D[q_0] \leftarrow 0$; `// Search Tree: node` $\mapsto$ `path cost`

$W \leftarrow$ `insert` $(q_0, h(q_0))$; `// Priority Queue`

**while** $W$ **do**

    **let** $q =$ `remove-min` $(W)$ **in**

        **if** $q \in A$ **then** **return** `tree-path` $(T, q)$ ;

        **else**

            **foreach** $q' \in \delta(q)$ **do**

                **let** $d' = D[q] + C(q, q')$ **in**

                    **if** $\neg$`contains` $(T, q')$ $\vee$ $(d' < D[q'])$ **then**

                        $T[q'] \leftarrow q$;

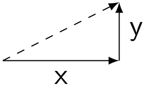
                        $D[q'] \leftarrow d'$;

                        $W \leftarrow \texttt{insert}\left(q', \underbrace{\overbrace{d'}^{q_0 \to q'} + \overbrace{h(q')}^{q' \to A}}_{\text{total cost estimate}}\right)$;

**return nil**;

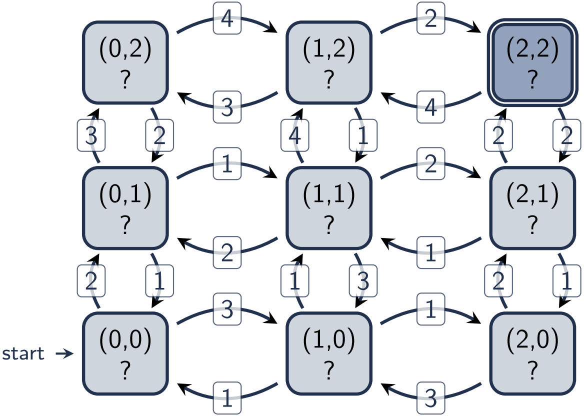
# Exercise: $A^*$ Search

| | | |
|---|---|---|
| 3 | 4 | 2 |
| 2 | 1 | 2 |
| 1 | 3 | 1 |




$$d_{\mathrm{euclidean}} = \sqrt{x^2 + y^2}$$

$$d_{\mathrm{manhattan}} = |x| + |y|$$



# Exercise: $A^*$ Search

# Exercise: $A^*$ Search

# Exercise: $A^*$ Search

# Exercise: $A^*$ Search

# Admissible Heuristic

**Definition: Admissible Heuristic**

An admissible heuristic never over-estimates the optimal cost to the goal:

$$\overbrace{h_{\mathrm{admiss}}(q)}^{\text{admissible heuristic}} \leq \overbrace{f_{\mathrm{opt}}(q)}^{\text{optimal cost}}$$

# Optimality of $A^*$

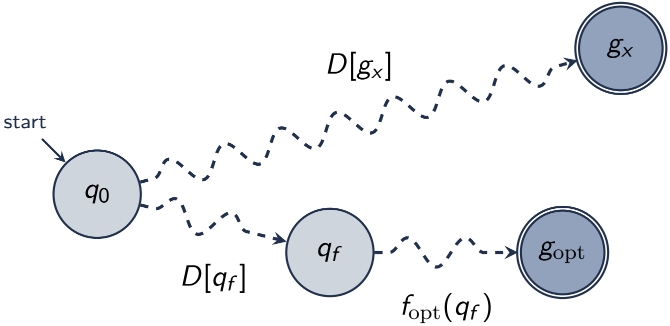
**Theorem**

When using an admissible heuristic, $A^*$ is optimal.

**Proof by Contradiction**

1. Assume we visit goal node $g_x$ (remove from $W$), where $g_x$ is suboptimal: $D[g_x] > f_{\mathrm{opt}}(q_0)$ and $h(g_x) = 0$
2. If $g_x$ is suboptimal, then the following must hold:
   2.1 There exists another path to a goal $g_{\mathrm{opt}}$ of cost $f_{\mathrm{opt}}(q_0)$, where $f_{\mathrm{opt}}(q_0) < D[g_x]$
   2.2 There is some frontier node $q_f$ in $W$ on the path to $g_{\mathrm{opt}}$ where $(D[q_f] + h(q_f)) \leq f_{\mathrm{opt}}(q_0)$
3. But, since $q_f$ has lower value that $g_x$, we would visit $q_f$ before visiting $g_x$
4. Contradiction

# $A^*$ Optimality Illustration




- $D[g_x] > f_{\mathrm{opt}}(q_0)$
- $D[q_f] + f_{\mathrm{opt}}(q_f) = f_{\mathrm{opt}}(q_0)$
- $D[q_f] + h(q_f) \leq f_{\mathrm{opt}}(q_0)$

*Must visit $q_f$ along optimal path before suboptimal $g_x$*

# Informed Search

**Procedure** informed-search($f_{\text{visit}}, q_0, A$)

```
T[q0] ← nil; // Search Tree
W ← insert(q0, 0); // Priority Queue
while W do
    let q = remove-min(W) in
        if q ∈ A then
            return tree-path(T, q)
        else
            f_visit(T, W, q);
return nil;
```

**Procedure** djikstra($C, q_0, \delta, A$)

```
D[q0] ← 0;
function visit(T,W,q) is
    foreach q' ∈ δ(q) do
        let d' = D[q] + C(q, q') in
            if ¬contains(T,q') ∨ (d' < D[q'])
              then
                T[q'] ← q;
                D[q'] ← d';
                W ← insert(q', d');
return informed-search(visit, q0, δ, A);
```

# Informed Search

**Procedure** greedy($h, q_0, \delta, A$)

```
function visit(T,W,q) is
    foreach q' ∈ δ(q) do
        if ¬contains(T,q') then
            T[q'] ← q;
            W ← insert(q', h(q'));
return informed-search(visit, q0, δ, A);
```

**Procedure** $A^*(C, h, q_0, \delta, A)$

```
D[q0] ← 0;
function visit(T,W,q) is
    foreach q' ∈ δ(q) do
        let d' = D[q] + C(q, q') in
            if ¬contains(T,q') ∨ (d' < D[q'])
              then
                T[q'] ← q;
                D[q'] ← d';
                W ← insert(q', d' + h(q'));
return informed-search(visit, q0, δ, A);
```

# Summary

Planning and Search Problems

Dijkstra's Algorithm

Optimality and Heuristics

Greedy Search

$A^*$ Search