Isabel Chen

# Balancing

# Balancing Criteria

# 1. Balancing Criteria:

- The difference between the left side weight, and the right side weight is no more than 10%
- If there is only 1 container, it’s balanced
- **Balanced** = max(weight(right_side),weight(left_side)) / min(weight(right_side),weight(right_side)) <1.1

Assume x represents the weight of the heavier side, and y represents the weight of the lighter side, and T represents the total weight x+y

x/y < 1.1,

x<1.1y,

T-y < 1.1y we can get

y> T/2.1 and x<((1.1)/(2.1))T

**W_min = (total weight)/2.1**

**W_max = (1.1 / 2.1) * (total weight)**

When we explore the states, we don’t have to recalculate both side, only need to consider one side.

# Moving Criteria

| | | | | | |
|---|---|---|---|---|---|
| 5 | unused | unused | unused | unused | unused |
| 10 | unused | unused | unused | unused | 4 |
| 7 | 1 | unused | unused | 9 | 2 |
| NAN | NAN | unused | unused | NAN | NAN |

# 2. What can be moved in a grid:

| | | | | | |
|---|---|---|---|---|---|
| 5 | unused | unused | unused | unused | unused |
| 10 | unused | unused | unused | unused | 4 |
| 7 | unused | unused | unused | 9 | 2 |
| NAN | NAN | unused | 1 | NAN | NAN |

What can we move in a grid:
the topmost container in its column

# 3. A container can be moved to:

| | | | | | |
|---|---|---|---|---|---|
| 5 | unused | unused | unused | unused | unused |
| 10 | unused | unused | unused | unused | 4 |
| 7 | unused | unused | unused | 9 | 2 |
| NAN | NAN | unused | 1 | NAN | NAN |

A container **can be moved to**:

(1) An unused spot that is :

  (a) Directly above a NAN slot, or

  (b) On the first row, or

  (c) Directly above other container

**Can't be moved to** its own column

The **Cost** of each move is the **Manhattan distance** between, for example, moving the container with weight 5 (at [1,4]) to unused space [4,2] would be |(4-1)| + |(2-4)| = 5

Note: haven't consider:

(a) the 9th and 10th rows that we can stack temporarily

(b) buffer

Isabel Chen

| | | | | | |
|---|---|---|---|---|---|
| 5 | unused | unused | unused | unused | unused |
| 10 | unused | unused | unused | unused | 4 |
| 7 | unused | unused | unused | 9 | 2 |
| NAN | NAN | unused | 1 | NAN | NAN |

| | | | | | |
|---|---|---|---|---|---|
| 5 | unused | unused | unused | unused | unused |
| 10 | unused | unused | unused | unused | 4 |
| 7 | unused | unused | unused | 9 | 2 |
| NAN | NAN | unused | 1 | NAN | NAN |

| | | | | | |
|---|---|---|---|---|---|
| 5 | unused | unused | unused | unused | unused |
| 10 | unused | unused | unused | unused | 4 |
| 7 | unused | unused | unused | 9 | 2 |
| NAN | NAN | unused | 1 | NAN | NAN |

# 4.1 Cost:

| | | | | | |
|---|---|---|---|---|---|
| 5 | unused | unused | unused | unused | unused |
| 10 | unused | unused | unused | unused | 4 |
| 7 | unused | unused | unused | 9 | 2 |
| NAN | NAN | unused | 1 | NAN | NAN |

The **Cost** of each move is the **Manhattan distance** between, for example, moving the container with weight 5 (at [1,4]) to unused space [4,2] would be |(4-1)| + |(2-4)| = 5

Isabel Chen

# 4.2 Cost:

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 4 | 5 | unused | unused | unused | unused | unused |
| 3 | 10 | unused | unused | unused | unused | unused |
| 2 | 7 | 1 | unused | unused | 9 | 2 |
| 1 | NAN | NAN | unused | unused | NAN | NAN |

We have to consider the **initial crane move cost**:

- At the first round, the cost will be the time the crane move to the slot position

Starting position is (5,1)
For example,
starting from slot (4,1) the cost is 1 //(4,1) - (5,1) - 1
Starting from slot (2,2) the cost is 4 //(2,2) - (5,1) = 4

# 4.2 Cost:

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 4 | unused | unused | unused | unused | unused | unused |
| 3 | 10 | unused | unused | unused | unused | unused |
| 2 | 7 | 1 | unused | unused | 9 | 2 |
| 1 | NAN | NAN | unused | 5 | NAN | NAN |

(?) We have to consider the **last crane move**:

For example,
starting from slot (1,4) the cost is 7  //|(1,4) - (5,1)| = 7
Starting from slot (2,2) the cost is 4  //(2,2) - (5,1) = 4

# Thought

# 5.1 Thought: Methods

- **Side_min = (total weight)/2.1**
  **Side_max = (1.1 / 2.1) * (total weight)**
  When we explore the states, we don't have to recalculate both side, only need to consider one side.
-

| **Method1(current search)** | **Method2 (heuristic)** |
| --- | --- |
| • Find the movable containers in the gird<br>• For each movable containers, we find the valid destination to move to<br>• Move them, push the new gridstate and cost into the frontier | • We have a goal side weight range.<br>• Pick the containers that can fulfill the weight range to the otherside.<br>• Find the best route for moving these container to the other side |

# 5 Thought: tracing back

- In GridState:
- Attribute ParentGrid
- Attribute CranePosition
- Tracing back from destination gridstate to the root gridstate to
  - show path : CranePosition
  - show cost: Each step’s cost = current grid cost - parent’s grid cost

# Heuristic

- Method 1 : Weight difference (left_weight - right_weight)
  - Pro: fast to compute, easy to understand
  - Con: less specific, doesn’t guide the search toward the feasible goal states.
- **Method 2: Valid Combinations weight (used for the software)**
  - **Goal_combination: can_balance() returns the list of combinations that matches goal weights range**
    - **Each item has a “ container combination”**

- Method 3: Misplaced Containers
  - Use the number of misplaced containers
-

- When receiving a balancing quest, check if it can be balanced by solving the subset sum (using bitmask dp)
  - Return a list of combinations
    - ShipCase3 Balanceable! Combinations: [
      ([600, 9041, 10], [10001, 500, 100]),
      ([600, 100, 9041], [10001, 500, 10]),
      ([600, 100, 9041, 10], [10001, 500]),
      ([10001], [500, 600, 100, 9041, 10]),
      ([10001, 10], [500, 600, 100, 9041]),
      ([10001, 100], [500, 600, 9041, 10]),
      ([10001, 100, 10], [500, 600, 9041]),
      ([500, 600, 9041], [10001, 100, 10]),
      ([500, 600, 9041, 10], [10001, 100]),
      ([500, 600, 100, 9041], [10001, 10]),
      ([500, 600, 100, 9041, 10], [10001]),
      ([10001, 500], [600, 100, 9041, 10]),
      ([10001, 500, 10], [600, 100, 9041]),
      ([10001, 600], [500, 100, 9041, 10])]
    - ShipCase4 [
      ([2000, 2007, 2011, 2020, 3044], [10000, 1100]), //left side, right side combination
      ([10000, 1100], [2000, 2007, 2011, 2020, 3044]) //left side, right side combination
      ]

- Method 2: Valid Combinations weight

Isabel Chen

State 1

| | | | | | |
|---|---|---|---|---|---|
| | | | | | |
| | 300 | | | 20 | 200 |
| | 100 | | | 70 | 30 |

State 1 (300,100), (20, 70, 200, 30)
Possible Goal Combinations are

- **a.** (300,20,30), (200,100,70)
- **b.** (300,70),(200,20,30,100)

Calculate the Heuristic from State 1 to state **a** (300,20,30), (200,100,70)

Heuristic = move 20 to left,
move 30 to left and
move 100 to right
2+3+2 = 7

Calculate the Heuristic from State 1 to state **b** (300,70), (200,100,20,30)

Heuristic = move 70 to left,
move 100 to right
2+2 = 4

Ignore obstacles and stacking requirements.
Calculate the Manhattan distance directly between the current position of the target container and the nearest available slot on the other side.

| | | | | | |
|---|---|---|---|---|---|
| | | | | | |
| | 300 | | | 20 | 200 |
| | 100 | | | 70 | 30 |

| | | | | | |
|---|---|---|---|---|---|
| | | | | | |
| | 300 | | | 20 | 200 |
| | 100 | | | 70 | 30 |

| | | | | | |
|---|---|---|---|---|---|
| | | | | | |
| | 300 | 20 | | | 200 |
| | | 30 | 100 | 70 | |

| | | | | | |
|---|---|---|---|---|---|
| | | | | | |
| | 300 | | | 20 | 200 |
| | | 70 | 100 | | 30 |

Isabel Chen

- Method 3: Misplaced Container (not selected)

State 1

| | | | | | |
|---|---|---|---|---|---|
| | | | | | |
| | 300 | | | 20 | 200 |
| | 100 | | | 70 | 30 |

State 1 (300,100), (20, 70, 200, 30)
Possible Goal Combinations are

a. (300,20,30), (200,100,70)
b. (300,70),(200,20,30,100)

Calculate the Heuristic from State 1 to state **a** (300,20,30), (200,100,70)
Misplaced Container = 3

Calculate the Heuristic from State 1 to state **b** (300,70), (200,100,20,30)
Misplaced Container = 2

| | | | | | |
|---|---|---|---|---|---|
| | | | | | |
| | 300 | | | 20 | 200 |
| | 100 | | | 70 | 30 |

| | | | | | |
|---|---|---|---|---|---|
| | | | | | |
| | 300 | | | 20 | 200 |
| | 100 | | | 70 | 30 |

| | | | | | |
|---|---|---|---|---|---|
| | | | | | |
| | 300 | 20 | | | 200 |
| | | 30 | 100 | 70 | |

| | | | | | |
|---|---|---|---|---|---|
| | | | | | |
| | 300 | | | 20 | 200 |
| | | 70 | 100 | | 30 |

# SIFT

take everything out and put them back in weight order

BUFFER

# 5. Goal:

- balanced ship with the minimum cost

| **Frontier.py** - Using a heapq to store nodes and a set to track explored states | |
|---|---|
| **class Frontier** | def init |
| | def insert(state):<br>- Add a state to the frontier with priority based on its cost<br>- Track the explored set to avoid re-visiting |
| | def pop(state):<br>- Remove and return the state with the lowest cost from the heap |
| | def contains(state): check if a specific state is already in the frontier |
| | def is_empty: |
| | def max_queue_size: track the max size of the frontier for testing purposes |

| Balance.py | |
|---|---|
| **class BalanceProblem** | def Init:<br>- set up initial grid, desired balanced weight range<br>- Create frontier |
| | def solve:<br>- A star procedure<br>- Loop the |

| **Grid_state_balance.py (1)** | |
|---|---|
| **class Grid** | def Init:<br>- Initialize the grid with slots, based on manifest<br>- Left weight, Right Weight, Total weight |
| | def get_balance: calculate and return the left and right side weights |
| **class Slot** | Represent individual slot in the grid |
| | Attributes:<br>- position<br>- state<br>- container |
| **class Container** | Represent a container |
| | Attributes:<br>- weight<br>- name<br>- position |

| Grid_state_balance.py (2) | |
|---|---|
| **def manhattan_distance(pos1, pos2)** | - Compute the Manhattan distance between two positions |
| **def move(container, target_position)** | - Move a container to a specified target position if it’s a valid move<br>- Update attributes: grid slot, container position, left and right weights |
| **def expand** | - Generate all possible moves for the current grid state<br>- For each container that can be moved, create a new grid state with the container moved to each valid target position<br>- Return a list of new grid states (children) |
| **print_path** | Traverse back to the root to print the sequence of move |

# Test and Debug

## sample_manifest_children_test_1

| | | | | | |
|---|---|---|---|---|---|
| 5 | unused | unused | unused | unused | unused |
| 10 | unused | unused | unused | unused | 4 |
| 7 | 1 | unused | unused | unused | 2 |
| NAN | NAN | unused | unused | NAN | NAN |

## Result for sample_manifest_children_test_1

| | | | | | |
|---|---|---|---|---|---|
| 5 | unused | unused | unused | unused | unused |
| 10 | unused | unused | unused | unused | 4 |
| 7 | unused | unused | unused | unused | 2 |
| NAN | NAN | unused | 1 | NAN | NAN |

Total states expanded:149

# Result for sample_manifest_children_test_1

| | | | | | |
|---|---|---|---|---|---|
| 5 | unused | unused | unused | unused | unused |
| 10 | unused | unused | unused | unused | 4 |
| 7 | unused | unused | unused | unused | 2 |
| NAN | NAN | unused | 1 | NAN | NAN |

What can we move in a grid:

- If in the slot, there is a container:
  - No other container is above it (topmost container in its column)

Isabel Chen

# Valid slots position to move to

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused |
| 6 | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused |
| 5 | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused |
| 4 | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused |
| 3 | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused |
| 2 | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused |
| 1 | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused |
| 0 | NAN | Cat 99 | Dog100 | unused | unused | unused | unused | unused | unused | unused | unused | NAN |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |

Isabel Chen

# Test1

Test1

Balance Moves: [Move container from position (1, 3) to position (1, 7)]

Test2

Balance Moves: [Move container from position (1, 4) to position (1, 7), Move container from position (1, 9) to position (1, 6)]

Test3

Balance Moves: [Move container from position (2, 1) to position (1, 7), Move container from position (1, 3) to position (2, 7), Move container from position (1, 4) to position (1, 8)]

Test4

Infinite

Test5

infinite

SilverQueen

Balance Moves: [Move container from position (1, 4) to position (1, 7), Move container from position (2, 2) to position (2, 7), Move container from position (1, 3) to position (1, 8)]

# Test2

Isabel Chen

Crane Starting point

| row/col | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused |
| 6 | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused |
| 5 | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused |
| 4 | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused |
| 3 | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused |
| 2 | Cat 40 | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused |
| 1 | NAN | Dog 50 | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused |
| 0 | NAN | NAN | NAN | unused | unused | Owl 35 | Ram 120 | unused | unused | unused | unused | NAN |

(0, 0), 0, NAN, 0

--------------------------------

(0, 1), 0, NAN, 0

--------------------------------

(0, 2), 0, NAN, 0

--------------------------------

(0, 3), 2, Ram, 120

--------------------------------

(0, 4), 1, UNUSED, 0

--------------------------------

(0, 5), 1, UNUSED, 0

--------------------------------

(0, 6), 1, UNUSED, 0

--------------------------------

(0, 7), 1, UNUSED, 0

--------------------------------

(0, 8), 2, Owl, 35

--------------------------------

(0, 9), 0, NAN, 0

--------------------------------

(0, 10), 0, NAN, 0

--------------------------------

(0, 11), 0, NAN, 0

--------------------------------

(1, 0), 0, NAN, 0

--------------------------------

(1, 1), 2, Dog, 50

--------------------------------

(1, 2), 1, UNUSED, 0

--------------------------------

(1, 3), 1, UNUSED, 0

--------------------------------

(1, 4), 1, UNUSED, 0

--------------------------------

(1, 5), 1, UNUSED, 0

--------------------------------

# Test4 Debug

Isabel Chen

# Test4 Debug

| | Crane Starting point | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | unused | unused | unused | unused | Pig 3044 | unused | unused | unused | unused | unused | unused | unused |
| 6 | unused | unused | unused | unused | Doe 1100 | unused | unused | unused | unused | unused | unused | unused |
| 5 | unused | unused | unused | unused | Owl 2020 | unused | unused | unused | unused | unused | unused | unused |
| 4 | unused | unused | unused | unused | Ewe 10000 | unused | unused | unused | unused | unused | unused | unused |
| 3 | unused | unused | unused | unused | Cow 2011 | unused | unused | unused | unused | unused | unused | unused |
| 2 | unused | unused | unused | unused | Dog 2007 | unused | unused | unused | unused | unused | unused | unused |
| 1 | NAN | unused | unused | unused | Cat 2000 | unused | unused | unused | unused | unused | unused | NAN |
| 0 | NAN | NAN | NAN | NAN | NAN | NAN | NAN | NAN | NAN | NAN | NAN | NAN |

Isabel Chen

# Test4 Debug

| | Crane Starting point | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | unused | unused | unused | unused | Pig 3044 | unused | unused | unused | unused | unused | unused | unused |
| 6 | unused | unused | unused | unused | Doe 1100 | unused | unused | unused | unused | unused | unused | unused |
| 5 | unused | unused | unused | unused | Owl 2020 | unused | unused | unused | unused | unused | unused | unused |
| 4 | unused | unused | unused | unused | Ewe 10000 | unused | unused | unused | unused | unused | unused | unused |
| 3 | unused | unused | unused | unused | Cow 2011 | unused | unused | unused | unused | unused | unused | unused |
| 2 | unused | unused | unused | unused | Dog 2007 | unused | unused | unused | unused | unused | unused | unused |
| 1 | NAN | unused | unused | unused | Cat 2000 | unused | unused | unused | unused | unused | unused | NAN |
| 0 | NAN | NAN | NAN | NAN | NAN | NAN | NAN | NAN | NAN | NAN | NAN | NAN |

Isabel Chen

```
for child_state, move in state.getPossibleStatesMoves():
    if child_state not in self.closed_set:
```

```
closed_set = {(0, 0), 0, NAN, 0
  function variables
  4391026352 = (0, 0), 0, NAN, 0
    special variables
    function variables
    columns = 12
    goal_weight = (10563, 11619)
    grid = [[(0, 0), 0, NAN, 0
      special variables
      function variables
      0 = [(0, 0), 0, NAN, 0
      1 = [(1, 0), 0, NAN, 0
      2 = [(2, 0), 1, UNUSED, 0
      3 = [(3, 0), 1, UNUSED, 0
      4 = [(4, 0), 1, UNUSED, 0
      5 = [(5, 0), 1, UNUSED, 0
      6 = [(6, 0), 1, UNUSED, 0
      7 = [(7, 0), 1, UNUSED, 0
      len() = 8
    id = UUID('2a54651b-25a6-4ca7-bdf4-1e7987918f73')
    left_weight = 22182
    right_weight = 0
    rows = 8
```

```
child_state = (0, 0), 0, NAN, 0
  special variables
  function variables
  columns = 12
  goal_weight = (10563, 11619)
  grid = [[(0, 0), 0, NAN, 0
    special variables
    function variables
    0 = [(0, 0), 0, NAN, 0
    1 = [(1, 0), 0, NAN, 0
    2 = [(2, 0), 1, UNUSED, 0
    3 = [(3, 0), 1, UNUSED, 0
    4 = [(4, 0), 1, UNUSED, 0
    5 = [(5, 0), 1, UNUSED, 0
    6 = [(6, 0), 1, UNUSED, 0
    7 = [(7, 0), 1, UNUSED, 0
    len() = 8
  id = UUID('2a54651b-25a6-4ca7-bdf4-1e7987918f73')
  left_weight = 22182
  right_weight = 0
```

```
> move = Move container from position (7, 4) to position (1, 3)
> neighbor_states_moves = [((0, 0), 0, NAN, 0
v new_grid = (0, 0), 0, NAN, 0
  > special variables
  > function variables
    columns = 12
  > goal_weight = (10563, 11619)
  > grid = [[(0, 0), 0, NAN, 0
  > id = UUID('2a54651b-25a6-4ca7-bdf4-1e7987918f73')
    left_weight = 22182
    right_weight = 0
    rows = 8
    total_weight = 22182
```

# Test5

Isabel Chen

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Crane Starting point | | | | | | | | | | | |
| 7 | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused |
| 6 | unused | unused | unused | unused | Doe 1100 | unused | unused | unused | unused | unused | unused | unused |
| 5 | unused | unused | unused | unused | Owl 2020 | unused | unused | unused | unused | unused | unused | unused |
| 4 | unused | unused | unused | unused | Ewe 10000 | unused | unused | unused | unused | unused | unused | unused |
| 3 | unused | unused | unused | unused | Cow 2011 | unused | unused | unused | unused | unused | unused | unused |
| 2 | unused | unused | unused | unused | Dog 2007 | unused | unused | unused | unused | unused | unused | unused |
| 1 | NAN | unused | unused | Pig 3044 | Cat 2000 | unused | unused | unused | unused | unused | unused | NAN |
| 0 | NAN | NAN | NAN | NAN | NAN | NAN | NAN | NAN | NAN | NAN | NAN | NAN |
| row/col | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |

# Test5

Isabel Chen

Crane Starting point

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused |
| 6 | unused | unused | unused | unused | | unused | unused | unused | unused | unused | unused | unused |
| 5 | unused | unused | unused | unused | | unused | unused | unused | unused | unused | unused | unused |
| 4 | unused | unused | | unused | | unused | unused | unused | unused | unused | unused | unused |
| 3 | unused | unused | unused | unused | | unused | Cat 2011 | unused | unused | unused | unused | unused |
| 2 | unused | unused | unused | unused | Ewe 10000 | unused | Owl 2020 | unused | unused | unused | unused | unused |
| 1 | NAN | unused | | | Doe 1100 | unused | Pig 3044 | Dog 2007 | Cat 2000 | unused | unused | NAN |
| 0 | NAN | NAN | NAN | NAN | NAN | NAN | NAN | NAN | NAN | NAN | NAN | NAN |
| row/col | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |

```
> function variables
> 0 = [(0, 0), 0, NAN, 0
> 1 = [(1, 0), 0, NAN, 0
> 2 = [(2, 0), 1, UNUSED, 0
> 3 = [(3, 0), 1, UNUSED, 0
> 4 = [(4, 0), 1, UNUSED, 0
> 5 = [(5, 0), 1, UNUSED, 0
> 6 = [(6, 0), 1, UNUSED, 0
> 7 = [(7, 0), 1, UNUSED, 0
  len() = 8
> id = UUID('ef9c4643-7524-4165-ac8d-59e3e8248d73')
  left_weight = 22182
  right_weight = 0
  rows = 8
  total_weight = 22182
```

```
∨ 08 = ((0, 0), 0, NAN, 0
  > special variables
  > function variables
  ∨ 0 = (0, 0), 0, NAN, 0
    > special variables
    > function variables
      columns = 12
    > goal_weight = (10563, 11619)
    > grid = [[(0, 0), 0, NAN, 0
    > id = UUID('d13d0e20-7dc3-4637-96a7-935218896cbe')
      left_weight = 22182
      right_weight = 0
      rows = 8
      total_weight = 22182
  > 1 = Move container from position (7, 4) to position (1, 9)
```

# Right_weight not updated

```
> function variables
    columns = 12
  > goal_weight = (10563, 11619)
  > grid = [[(0, 0), 0, NAN, 0
  > id = UUID('1416946e-8157-41fd-8f2c-39ad334b2fa1')
    left_weight = 22182
    right_weight = 0
    rows = 8
    total_weight = 22182
  f_cost = 7
  g_cost = 7
  h_cost = 0
> initial_crane_position = (8, 0)
> move = Move container from position (1, 3) to position (7, 4)
```

```
def balance_heuristic(self, state):
    left_w, right_w, total_w = state.calculate_weights()
    return abs(left_w - right_w)
```

Weight is not updated immediately after move but recalculate again during search, i think it’s not the most efficient way

```
    for child_state, move in state.getPossibleStatesMove
        if child_state not in self.closed_set:
            new_g_cost = g_cost + move.get_cost()
            h_cost = self.balance_heuristic(child_state)
            new_f_cost = new_g_cost + h_cost
            new_path = path + [move]

            heapq.heappush(self.open_set, (new_f_cost, n
```

# Todo

- 1. We need to consider the cost between moves
    - The cost is stored at the movement now
- 2. Heuristic

# Cost Debug

Isabel Chen

| | Crane Starting point | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | unused | unused | unused | unused | Pig 3044 | unused | unused | unused | unused | unused | unused | unused |
| 6 | unused | unused | unused | unused | Doe 1100 | unused | unused | unused | unused | unused | unused | unused |
| 5 | unused | unused | unused | unused | Owl 2020 | unused | unused | unused | unused | unused | unused | unused |
| 4 | unused | unused | unused | unused | Ewe 10000 | unused | unused | unused | unused | unused | unused | unused |
| 3 | unused | unused | unused | unused | Cow 2011 | unused | unused | unused | unused | unused | unused | unused |
| 2 | unused | unused | unused | unused | Dog 2007 | unused | unused | unused | unused | unused | unused | unused |
| 1 | NAN | unused | unused | unused | Cat 2000 | unused | unused | unused | unused | unused | unused | NAN |
| 0 | NAN | NAN | NAN | NAN | NAN | NAN | NAN | NAN | NAN | NAN | NAN | NAN |

Isabel Chen

# Cost Debug

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Crane Starting point | | | | | | | | | | | |
| 7 | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused |
| 6 | unused | unused | unused | | | unused | unused | unused | unused | unused | unused | unused |
| 5 | unused | unused | unused | unused | Owl 2020 | unused | unused | unused | unused | unused | unused | unused |
| 4 | unused | unused | unused | unused | Ewe 10000 | unused | unused | unused | unused | unused | unused | unused |
| 3 | unused | unused | unused | unused | Cow 2011 | unused | unused | unused | unused | unused | unused | unused |
| 2 | unused | unused | unused | unused | Dog 2007 | unused | unused | unused | unused | unused | unused | unused |
| 1 | NAN | unused | unused | Pig 3044 | Cat 2000 | Doe 1100 | unused | unused | unused | unused | unused | NAN |
| 0 | NAN | NAN | NAN | NAN | NAN | NAN | NAN | NAN | NAN | NAN | NAN | NAN |
| row/col | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |

```
open_set = [(7, 7, [...], (0, 0), 0, NAN, 0
  > special variables
  > function variables
  > 00 = (7, 7, [Move container from position (7, 4) to position (1, 3)], (0, 0), 0, NAN, 0
  > 01 = (7, 7, [Move container from position (7, 4) to position (1, 5)], (0, 0), 0, NAN, 0
  > 02 = (8, 8, [Move container from position (7, 4) to position (1, 6)], (0, 0), 0, NAN, 0
  > 03 = (9, 9, [Move container from position (7, 4) to position (1, 1)], (0, 0), 0, NAN, 0
  > 04 = (8, 8, [Move container from position (7, 4) to position (1, 2)], (0, 0), 0, NAN, 0
  > 05 = (9, 9, [Move container from position (7, 4) to position (2, 0)], (0, 0), 0, NAN, 0
  > 06 = (9, 9, [Move container from position (7, 4) to position (1, 7)], (0, 0), 0, NAN, 0
  > 07 = (10, 10, [Move container from position (7, 4) to position (1, 8)], (0, 0), 0, NAN, 0
  > 08 = (11, 11, [Move container from position (7, 4) to position (1, 9)], (0, 0), 0, NAN, 0
  > 09 = (12, 12, [Move container from position (7, 4) to position (1, 10)], (0, 0), 0, NAN, 0
  > 10 = (12, 12, [Move container from position (7, 4) to position (2, 11)], (0, 0), 0, NAN, 0
    len() = 11
```

```python
        while self.open_set:
            f_cost, g_cost, path, state = heapq.heappop(self.open_set)

            if state.isBalanced():
```

```python
def __lt__(self, other):
    return self.get_cost() < other.get_cost()
```

The problem is, now it only consider from point A to B, not the accumulated cost

Isabel Chen

| | Crane Starting point | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | unused | unused | unused | unused | Pig 3044 | unused | unused | unused | unused | unused | unused | unused |
| 6 | unused | unused | unused | unused | Doe 1100 | unused | unused | unused | unused | unused | unused | unused |
| 5 | unused | unused | unused | unused | Owl 2020 | unused | unused | unused | unused | unused | unused | unused |
| 4 | unused | unused | unused | unused | Ewe 10000 | unused | unused | unused | unused | unused | unused | unused |
| 3 | unused | unused | unused | unused | Cow 2011 | unused | unused | unused | unused | unused | unused | unused |
| 2 | unused | unused | unused | unused | Dog 2007 | unused | unused | unused | unused | unused | unused | unused |
| 1 | NAN | unused | unused | Pig 3044 | Cat 2000 | unused | unused | unused | unused | unused | unused | NAN |
| 0 | NAN | NAN | NAN | NAN | NAN | NAN | NAN | NAN | NAN | NAN | NAN | NAN |

```
> move = Move container from position (6, 4) to position (2, 11)
v possible_moves = [Move container from position (1, 3) to position (2
  > special variables
  > function variables
  > 00 = Move container from position (1, 3) to position (2, 0)
  > 01 = Move container from position (1, 3) to position (1, 1)
  > 02 = Move container from position (1, 3) to position (1, 2)
  > 03 = Move container from position (1, 3) to position (7, 4)
  > 04 = Move container from position (1, 3) to position (1, 5)
  > 05 = Move container from position (1, 3) to position (1, 6)
  > 06 = Move container from position (1, 3) to position (1, 7)
  > 07 = Move container from position (1, 3) to position (1, 8)
  > 08 = Move container from position (1, 3) to position (1, 9)
  > 09 = Move container from position (1, 3) to position (1, 10)
  > 10 = Move container from position (1, 3) to position (2, 11)
  > 11 = Move container from position (6, 4) to position (2, 0)
  > 12 = Move container from position (6, 4) to position (1, 1)
  > 13 = Move container from position (6, 4) to position (1, 2)
  > 14 = Move container from position (6, 4) to position (2, 3)
  > 15 = Move container from position (6, 4) to position (1, 5)
  > 16 = Move container from position (6, 4) to position (1, 6)
  > 17 = Move container from position (6, 4) to position (1, 7)
  > 18 = Move container from position (6, 4) to position (1, 8)
  > 19 = Move container from position (6, 4) to position (1, 9)
  > 20 = Move container from position (6, 4) to position (1, 10)
  > 21 = Move container from position (6, 4) to position (2, 11)
    len() = 22
> self = (0, 0), 0, NAN, 0
```

Isabel Chen

```
› 05 = Slot(grid_id=80172891-8fba-4981-90e0-970e0d1b7c8b, position=(0, 5), state=0, container=NAN, 0)
› 06 = Slot(grid_id=80172891-8fba-4981-90e0-970e0d1b7c8b, position=(0, 6), state=0, container=NAN, 0)
› 07 = Slot(grid_id=80172891-8fba-4981-90e0-970e0d1b7c8b, position=(0, 7), state=0, container=NAN, 0)
› 08 = Slot(grid_id=80172891-8fba-4981-90e0-970e0d1b7c8b, position=(0, 8), state=0, container=NAN, 0)
› 09 = Slot(grid_id=80172891-8fba-4981-90e0-970e0d1b7c8b, position=(0, 9), state=0, container=NAN, 0)
› 10 = Slot(grid_id=80172891-8fba-4981-90e0-970e0d1b7c8b, position=(0, 10), state=0, container=NAN, 0)
› 11 = Slot(grid_id=80172891-8fba-4981-90e0-970e0d1b7c8b, position=(0, 11), state=0, container=NAN, 0)
  len() = 12
1 = [Slot(grid_id=80172891-8fba-4981-90e0-970e0d1b7c8b, position=(1, 0), state=0, container=NAN, 0), Slot(grid_id=80172891-8
2 = [Slot(grid_id=80172891-8fba-4981-90e0-970e0d1b7c8b, position=(2, 0), state=1, container=UNUSED, 0), Slot(grid_id=8017289
3 = [Slot(grid_id=80172891-8fba-4981-90e0-970e0d1b7c8b, position=(3, 0), state=1, container=UNUSED, 0), Slot(grid_id=8017289
4 = [Slot(grid_id=80172891-8fba-4981-90e0-970e0d1b7c8b, position=(4, 0), state=1, container=UNUSED, 0), Slot(grid_id=8017289
5 = [Slot(grid_id=80172891-8fba-4981-90e0-970e0d1b7c8b, position=(5, 0), state=1, container=UNUSED, 0), Slot(grid_id=8017289
6 = [Slot(grid_id=80172891-8fba-4981-90e0-970e0d1b7c8b, position=(6, 0), state=1, container=UNUSED, 0), Slot(grid_id=8017289
7 = [Slot(grid_id=80172891-8fba-4981-90e0-970e0d1b7c8b, position=(7, 0), state=1, container=UNUSED, 0), Slot(grid_id=8017289
› special variables
› function variables
› 00 = Slot(grid_id=80172891-8fba-4981-90e0-970e0d1b7c8b, position=(7, 0), state=1, container=UNUSED, 0)
› 01 = Slot(grid_id=80172891-8fba-4981-90e0-970e0d1b7c8b, position=(7, 1), state=1, container=UNUSED, 0)
› 02 = Slot(grid_id=80172891-8fba-4981-90e0-970e0d1b7c8b, position=(7, 2), state=1, container=UNUSED, 0)
› 03 = Slot(grid_id=80172891-8fba-4981-90e0-970e0d1b7c8b, position=(7, 3), state=1, container=UNUSED, 0)
› 04 = Slot(grid_id=80172891-8fba-4981-90e0-970e0d1b7c8b, position=(7, 4), state=1, container=None)
› 05 Slot (grid_id=80172891-8fba-4981-90e0-970e0d1b7c8b, position=(7, 5), state=1, container=UNUSED, 0)
› 06 = Slot(grid_id=80172891-8fba-4981-90e0-970e0d1b7c8b, position=(7, 6), state=1, container=UNUSED, 0)
› 07 = Slot(grid_id=80172891-8fba-4981-90e0-970e0d1b7c8b, position=(7, 7), state=1, container=UNUSED, 0)
› 08 = Slot(grid_id=80172891-8fba-4981-90e0-970e0d1b7c8b, position=(7, 8), state=1, container=UNUSED, 0)
› 09 = Slot(grid_id=80172891-8fba-4981-90e0-970e0d1b7c8b, position=(7, 9), state=1, container=UNUSED, 0)
› 10 = Slot(grid_id=80172891-8fba-4981-90e0-970e0d1b7c8b, position=(7, 10), state=1, container=UNUSED, 0)
› 11 = Slot(grid_id=80172891-8fba-4981-90e0-970e0d1b7c8b, position=(7, 11), state=1, container=UNUSED, 0)
```

Isabel Chen

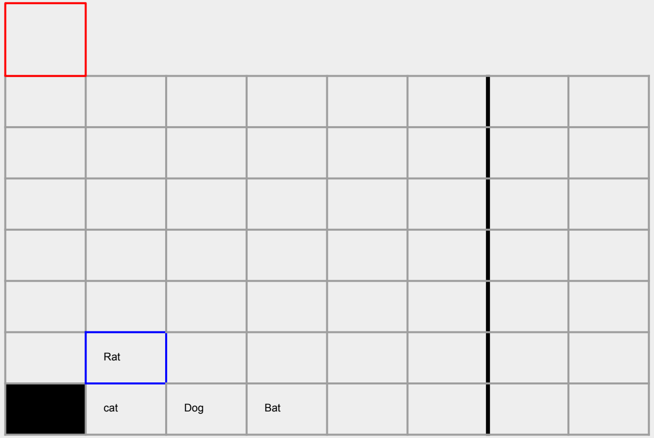
# Cost Debug

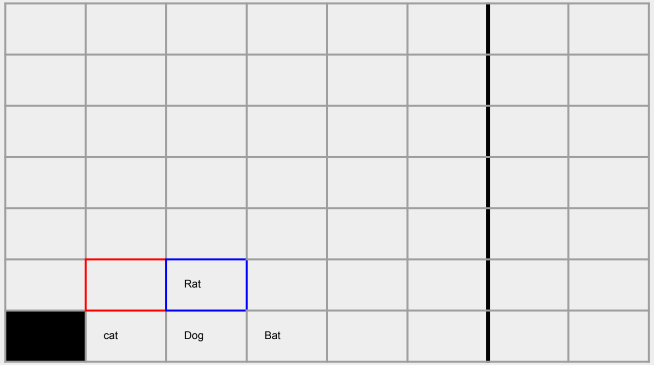
Crane Starting point

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused |
| 6 | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused |
| 5 | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused |
| 4 | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused |
| 3 | unused | unused | unused | unused | Cow 2011 | Owl 2020 | unused | unused | unused | unused | unused | unused |
| 2 | unused | unused | unused | unused | Dog 2007 | Doe 1100 | Ewe 10000 | unused | unused | unused | unused | unused |
| 1 | NAN | unused | unused | unused | Cat 2000 | Pig 3044 | Doe 1100 | unused | unused | unused | unused | NAN |
| 0 | NAN | NAN | NAN | NAN | NAN | NAN | NAN | NAN | NAN | NAN | NAN | NAN |
| row/col | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |

Isabel Chen

# Time cost for operator

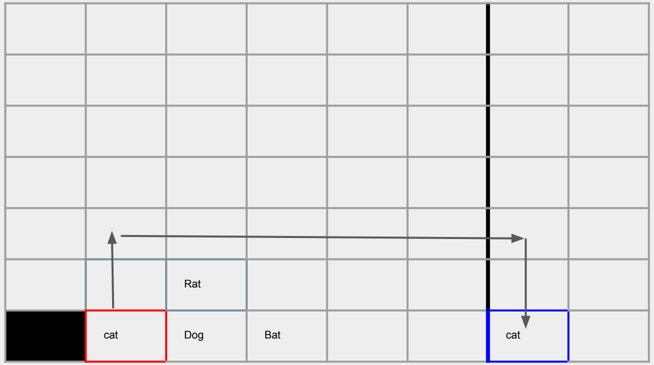



```
==========================================
Balancing job selected.
Balanceable! Combinations: [([60], [20, 20, 20])]
Balanced path found
Balance Moves:
Move container from position (1, 1) to position (1, 2)
Move container from position (0, 1) to position (0, 6)
==========================================
```

Actual Crane Path and Cost:

1. Move container from (1,1) to (1,2)
   a. (7,0) to (1,1) = 7
   b. (1,1) to (1,2) = 1
2. Move container from (0,1) to (0,6)
   There is an obstacle in the way, so the cost is move up, right, then down. The cost is 9. NOT 5.



Isabel Chen

Crane Starting point

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused |
| 6 | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused |
| 5 | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused |
| 4 | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused |
| 3 | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused |
| 2 | Cat | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused |
| 1 | NAN | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | NAN |
| 0 | NAN | NAN | NAN | unused | unused | unused | unused | unused | Owl | NAN | NAN | NAN |
| row/col | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |

```diff
    e)
-
    new_f_cost = new_g_cost + h_cost
                                    new_path
    = path + [move]
```

```diff
    e)
+
    new_f_cost += new_g_cost + h_cost
                                    new_path
    = path + [move]
```

# Transfer

# Loading

- move container from truck to the nearest unused slot

# Unloading

# Unloading scenario 1

| | | | | | |
|---|---|---|---|---|---|
| 5 | unused | unused | unused | unused | unused |
| 10 | unused | unused | unused | unused | 4 |
| 7 | 1 | unused | unused | 9 | 2 |
| NAN | NAN | unused | unused | NAN | NAN |

1. If the container is the topmost in its column, just unload it

Isabel Chen

# Unloading scenario 2-1

| | | | | | |
|---|---|---|---|---|---|
| 5 | unused | unused | unused | unused | unused |
| 10 | unused | unused | unused | unused | 4 |
| 7 | 1 | unused | unused | 9 | 2 |
| NAN | NAN | unused | unused | NAN | NAN |

1. If the container is the topmost in its column, just unload it
2. If the container below some container
    a. Move the above container(s) away
        i. To the nearest valid slot
    b. Take the container out

Isabel Chen

# Unloading scenario 2-2

| | | | | | |
|---|---|---|---|---|---|
| 5 | unused | unused | unused | unused | unused |
| 10 | 5 | unused | unused | unused | 4 |
| 7 unload | 1 | unused | unused | 9 | 2 |
| NAN | NAN | unused | unused | NAN | NAN |

For example

- Unload 7, cost = 16
  - Move 5 away, now the crane is at (3,2), cost 1+2 = 3
  - Move 10 away, cost = 1 + 2 = 3
  - Unload 7, cost = 3+ 3 + 4 (to car)

| | | | | | |
|---|---|---|---|---|---|
| unused | 10 | unused | unused | unused | unused |
| 10 | 5 | unused | unused | unused | 4 |
| 7 unload | 1 | unused | unused | 9 | 2 |
| NAN | NAN | unused | unused | NAN | NAN |

| | | | | | |
|---|---|---|---|---|---|
| unused | 10 | unused | unused | unused | unused |
| unused | 5 | unused | unused | unused | 4 |
| unused | 1 | unused | unused | 9 | 2 |
| NAN | NAN | unused | unused | NAN | NAN |

# Loading and Unloading

Isabel Chen

Load item1

Load Item2

| | | | | | |
|---|---|---|---|---|---|
| 5 | unused | unused | unused | unused | unused |
| 10 | unused | unused | unused | unused | 4 |
| 7 unload | 1 | unused | unused | 9 | 2 unload |
| NAN | NAN | unused | unused | NAN | NAN |

1. How do we know to Load or Unload first?

Isabel Chen

Load item1

Load Item2

| | | | | | |
|---|---|---|---|---|---|
| 5 | unused | unused | unused | unused | unused |
| 10 | unused | unused | unused | unused | 4 |
| 7 unload | 1 | unused | unused | unused | unused |
| NAN | NAN | unused | unused | NAN | NAN |

1. How do we know to Load or Unload first?
   a. Based on the cost.
2. For example
   a. For each state, we expand all the possible next moves,
   - Unload 7 cost = 16
   - Unload 2 cost = 22
   - Load item1
   - Load item2

Isabel Chen

Load Item2

| | | | | | |
|---|---|---|---|---|---|
| 5 | unused | unused | unused | unused | unused |
| 10 | item1 | unused | unused | unused | 4 |
| 7 unload | 1 | unused | unused | unused | 2 unload |
| NAN | NAN | unused | unused | NAN | NAN |

1. How do we know to Load or Unload first?
    a. Based on the cost.
2. For example
    a. For each state, we expand all the possible next moves,
    - Unload 7 cost = 16
    - Unload 2 cost = 22
    - Load item1 cost = 7
    - Load item2

Isabel Chen

Load Item1

| | | | | | |
|---|---|---|---|---|---|
| 5 | unused | unused | unused | unused | unused |
| 10 | item2 | unused | unused | unused | 4 |
| 7 unload | 1 | unused | unused | unused | 2 unload |
| NAN | NAN | unused | unused | NAN | NAN |

1. How do we know to Load or Unload first?
   a. Based on the cost.
2. For example
   a. For each state, we expand all the possible next moves,
   - Unload 7 cost = 16
   - Unload 2 cost = 22
   - Load item1 cost = 7
   - Load item2 cost = 7

Load item1 | Load Item2

| | | | | | |
|---|---|---|---|---|---|
| 5 | unused | unused | unused | unused | unused |
| 10 | unused | unused | unused | unused | 4 |
| 7 unload | 1 | unused | unused | 9 | 2 unload |
| NAN | NAN | unused | unused | NAN | NAN |

1. How do we know to Load or Unload first?
   a. Based on the cost.
2. For example
   a. For each state, we expand all the possible next moves,
      - Unload 7 cost = 16
      - Unload 2 cost = 22
      - Load item1 cost = 7
      - Load item2 cost = 7

Isabel Chen

Load item1 | Load Item2

| | | | | | |
|---|---|---|---|---|---|
| 5 | unused | unused | unused | unused | unused |
| 10 | unused | unused | unused | unused | 4 |
| 7 unload | 1 | unused | unused | 9 | 2 unload |
| NAN | NAN | unused | unused | NAN | NAN |

1. How do we know to Load or Unload first?
   a. Based on the cost.
2. For example
   a. For each state, we expand all the possible next moves,
   - *Unload 7 cost = 16 ←not correct*
   - *Unload 2 cost = 22 ←not correct*
   - Load item1 cost = 7
   - Load item2 cost = 7

Now the question is, during unloading, there are multiple steps, for each step, we should consider it as a new state and consider our sequence again.

Isabel Chen

Load item1 Load Item2

| | | | | | |
|---|---|---|---|---|---|
| 5 | unused | unused | unused | unused | unused |
| 10 | unused | unused | unused | unused | 4 |
| 7 unload | 1 | unused | unused | 9 | 2 unload |
| NAN | NAN | unused | unused | NAN | NAN |

| | | | | | |
|---|---|---|---|---|---|
| 5 | unused | unused | unused | unused | unused |
| 10 | 5 | unused | unused | unused | 4 |
| 7 unload | 1 | unused | unused | 9 | 2 unload |
| NAN | NAN | unused | unused | NAN | NAN |

Now the question is, during unloading, there are multiple steps, for each step, we should consider it as a new state and consider our sequence again.

1. For example, to Unload 7
   a. Move 5 away, cost: 1+ 2 = 3, at this point, it's a new grid. We expand the state
      - Unload 7 cost
      - Unload 2 cost
      - Load item1 cost
      - Load item2 cost

# Grid representation

Unload list = [] (include names of the unloading container)

Load list = [] (include names and weights of the loading containers)

Container= [] (keep track of containers on the ship for manifest)

# Process

If a problem is transfer,

Initial the the problem with the grid state with:(a) manifest, (b) unload list, (c) load list, (d) initial crane position

(a) Cannot be empty
(b) Or (c) could be empty, but not both. One of them must be not empty list

(d) initial/end crance position (8,0)

# Begin solving the problem

1. Let’s define which containers can be moved and where to be moved:
    a. Search for unload list. If we can unload that item(no blocking), that item can be moved. And the destination is the truck
    b. Search for unload list, if the item has containers above it, we can remove that container to other slots available (move to other column) in the gird.
    c. Search for load list, move a container from the truck into the grid. The start position is truck and the end position is inside the grid.

**Crane Position Representation**:

- Use the crane position as a simple attribute ("truck" or (row, col) for the grid).

**Cost Calculation**:

- The crane_position is only used to calculate the cost of moving the crane:
  - From the truck to the grid
  - From the grid to the truck
  - Between slots within the grid
  - 

A State Representation*:

- The grid state (Grid) remains the same as before, with crane_position simply stored for cost evaluation.
- The crane position doesn't add additional complexity to the A* search itself.. it’s just used for cost computations.

# Loading

If the crane starts at (8,0) :

Takes 2 minutes to move to the truck;

If the crane starts at other positions in the gris:

Move to (8,0) then + 2 minutes to the truck;

If the crane starts at the truck:the cost of crane to start is 0

Isabel Chen

# Load Bat

Crane Starting point

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused |
| 6 | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused |
| 5 | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused |
| 4 | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused |
| 3 | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused |
| 2 | Cat | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused | unused |
| 1 | NAN | Dog | unused | unused | unused | unused | unused | unused | unused | unused | unused | NAN |
| 0 | NAN | NAN | NAN | Ram | unused | unused | unused | unused | Owl | NAN | NAN | NAN |
| row/col | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |

Isabel Chen

# Debug Load unload

```
1  unload,Dog
2  load,Bat,431
```

```
state = Position: (0, 0), State: 0, Container Name: NAN, Container Weight: 0
  > special variables
  > function variables
    columns = 12
  > crane_position = (-1, -1)
    goal_weight = 0
  > id = UUID('a1a399c5-2da8-46e1-b74c-d7e729e5b869')
  > left_containers = {Cat, 99, Dog, 100}
    left_weight = 199
  > load_list = [('Bat', 431)]
  > right_containers = {}
    right_weight = 0
    rows = 8
  > slot = [[Slot(grid_id=a1a399c5-2da8-46e1-b74c-d7e729e5b869, position=(0, 0), state=0…
    total_weight = 199
  > unload_list = [Dog, 100]
  _ = 4529941648
> Globals
```