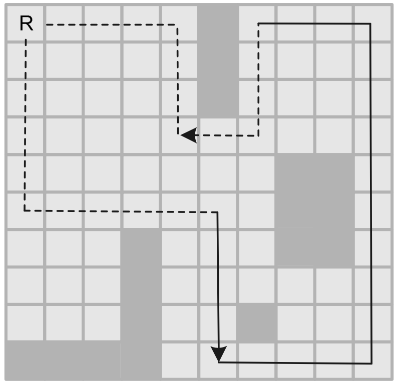
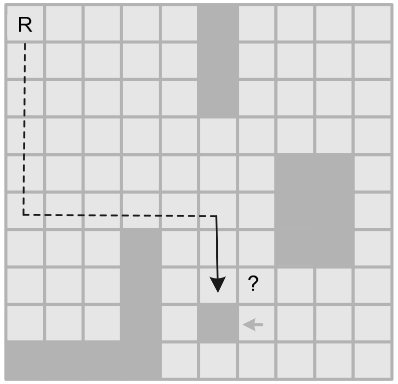
Preliminaries and Problem Formulation

Simplification 1—Roomba stores the complete map; Computation is done off-line

Calculated solution

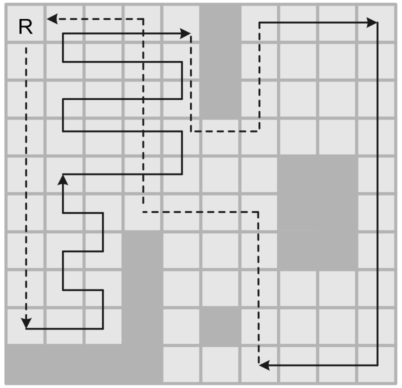


In real life

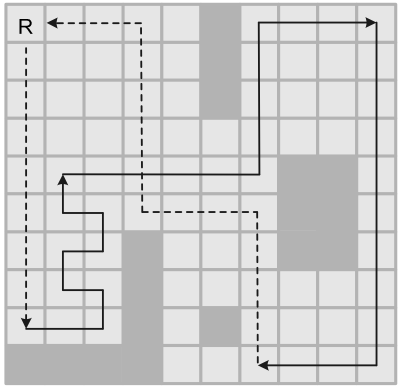


Simplification 2—Roomba does not switch between cleaning and traveling modes within one trip

In real life



Our problem formulation



Algorithm Design

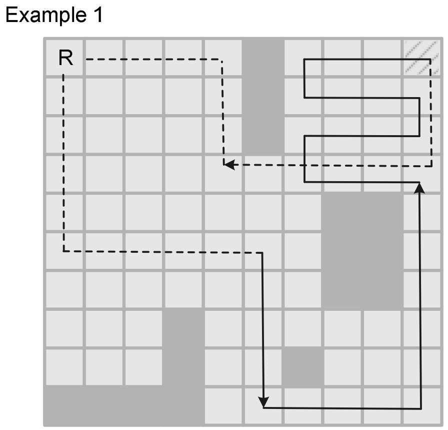
Key idea:

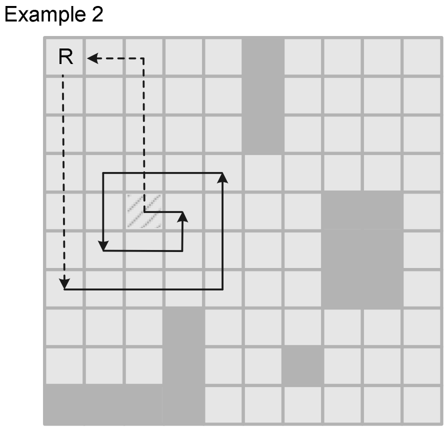
Design a brute-force algorithm that checks all feasible paths when battery constraint allows.

Actions:

Step 1: **Loop through all possible starting points.**
Step 2: **For each, try moving all possible directions.**

Base Case 1: **Dead end**





Base Case 2: **Battery exhausted. Can’t afford cleaning and moving back from the current vertex.**

Pseudo-code

```
Graph input;
BooleanMatrix matrix;
int maxGoodness;
GraphSolution output;
Stack<Vertex> sequence;

JADE-MESH-OUTERLOOP(Graph graphInput)
    input ← graphInput;
    matrix ← initialize as the size of graphInput and populate with false;
    maxGoodness ← the minimum integer;
    output ← null;
    sequence ← initialize as a new object;
    for int i ← 0 to i ← graphInput.width; i++ {
        for int j ← 0 to j ← graphInput.height; j++ {
            JADE-MESH-RECURSION (i, j, 0,
              graphInput.capacity – capacityToBase(i,j));
        }
    }
    return output;

JADE-MESH-RECURSION(int x, int y, int goodness, int capacity)
    if isBlocked(x, y) = true or isBatteryExhausted(x, y, capacity) = true {
        If output = null or goodness > maxGoodness {
            maxGoodness ← goodness;
            output ← new GraphSolution(sequence, goodness);
        }
        return;
    }
    sequence.push(new Vertex(x, y));
    matrix.mark(x, y);
    int newGoodness ← goodness + input.priority(x, y);
    int newCapacity ← capacity – input.comsumption(x, y) – 1;
    JADE-MESH-RECURSION(x – 1, y, newGoodness, newCapacity);
    JADE-MESH-RECURSION(x + 1, y, newGoodness, newCapacity);
    JADE-MESH-RECURSION(x, y – 1, newGoodness, newCapacity);
    JADE-MESH-RECURSION(x, y + 1, newGoodness, newCapacity);
    sequence.pop();
    matrix.unmark(x, y);
```

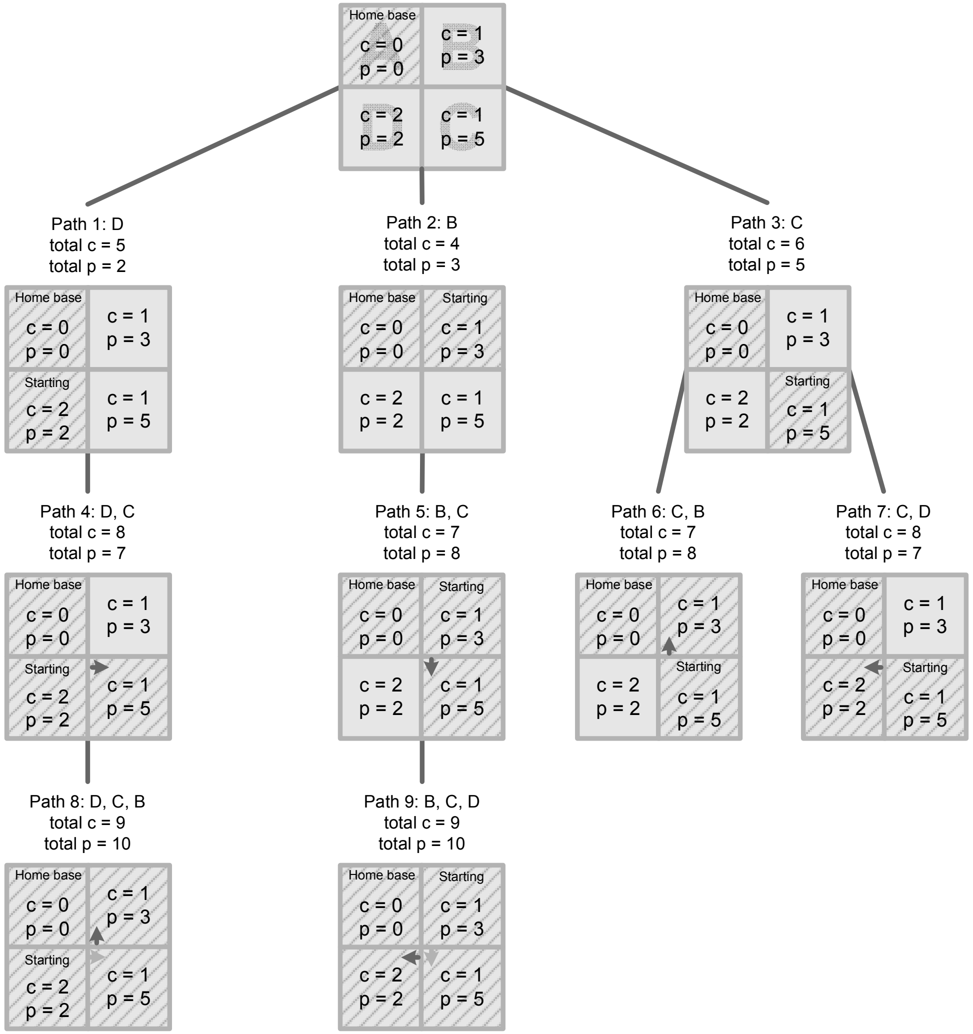
Helper methods that are needed:

```
CAPACITYTOBASE(int x', int y')
    return distance(input.base.x , input.base.y, x', y');

ISBATTERYEXHAUSTED(int x, int y, int capacity)
    if capacityToBase(x, y) + input.consumption(x, y) + 1 >
      capacity {
        return true;
    }
    return false;

ISBLOCKED(int x, int y)
    if x < 0 or x <= matrix.width
      or y < 0 or y >= matrix.height {
        return true;
    }
    if x = input.base.x and y = input.base.y {
        return true;
    }
    return matrix.isMarked(x, y);
```

Simplified Example



Home base A c = 0 p = 0 | B c = 1 p = 3
D c = 2 p = 2 | C c = 1 p = 5

Path 1: D
total c = 5
total p = 2

Home base c = 0 p = 0 | c = 1 p = 3
Starting c = 2 p = 2 | c = 1 p = 5

Path 2: B
total c = 4
total p = 3

Home base c = 0 p = 0 | Starting c = 1 p = 3
c = 2 p = 2 | c = 1 p = 5

Path 3: C
total c = 6
total p = 5

Home base c = 0 p = 0 | c = 1 p = 3
c = 2 p = 2 | Starting c = 1 p = 5

Path 4: D, C
total c = 8
total p = 7

Home base c = 0 p = 0 | c = 1 p = 3
Starting c = 2 p = 2 | c = 1 p = 5

Path 5: B, C
total c = 7
total p = 8

Home base c = 0 p = 0 | Starting c = 1 p = 3
c = 2 p = 2 | c = 1 p = 5

Path 6: C, B
total c = 7
total p = 8

Home base c = 0 p = 0 | c = 1 p = 3
c = 2 p = 2 | Starting c = 1 p = 5

Path 7: C, D
total c = 8
total p = 7

Home base c = 0 p = 0 | c = 1 p = 3
c = 2 p = 2 | Starting c = 1 p = 5

Path 8: D, C, B
total c = 9
total p = 10

Home base c = 0 p = 0 | c = 1 p = 3
Starting c = 2 p = 2 | c = 1 p = 5

Path 9: B, C, D
total c = 9
total p = 10

Home base c = 0 p = 0 | Starting c = 1 p = 3
c = 2 p = 2 | c = 1 p = 5

All candidate solution is battery allows:

Path	c Consumption	p Summation
D, C, B	9	10
B, C, D	9	10
C, B	7	8
B, C	7	8
D, C	8	7
C, D	8	7
C	6	5
B	4	3
D	5	2

Final solutions for specific battery input:

Battery Capacity	Final Solution
C >= 9	D, C, B or B, C, D
C = 7, 8	B, C or C, B
C = 6	C
C = 4, 5	B
C < 4	null

Complexity Analysis

Time Complexity

Jade-Mesh-OuterLoop Method: $O(V)$

Jade-Mesh-Recursion Method: $O(4^v)$

Overall Time Complexity: $O(V) \times O(4^v) = O(V \times 4^v)$

Space Complexity $O(V)$

Improvement Attempt

The worst case scenario:

Battery capacity is sufficient and does not act like a constraint in the problem.

The worst case of a particular map:

Can be addressed as a variant of the <u>Longest Path Problem</u>—a know NP-Complete .

Conclusion:

It is unlikely for us to find a polynomial algorithm for this problem set.