UNIVERSITY OF SCIENCE

ARTIFICIAL INTELLIGENCE

CS420

Report Project 1 - Search Ares Adventure

Author:

Thien-Bao NGO
22125009

Thanh-Hai DAO
22125022

Nhan-Kiet KHUONG
22125043

Hoang-Ky NGUYEN
22125045

November 10, 2024

Contents

Chapter 1

Work Assignment

Requirements	Members	Completion Rate
Implement BFS	Đào Thanh Hải	100%
Implement DFS	Nguyễn Hoàng Kỳ	100%
Implement UCS	Khương Nhân Kiệt	100%
Implement A*	Ngô Thiên Bảo	100%
Test Case	Khương Nhân Kiệt, Đào Thanh Hải	100%
UI	Ngô Thiên Bảo, Nguyễn Hoàng Kỳ	100%
Report	Đào Thanh Hải, Ngô Thiên Bảo	100%
Video	Nguyễn Hoàng Kỳ, Khương Nhân Kiệt	100%

Chapter 2

Self-evaluation

No.	**Details**	**Score**	**Self Evaluation**
1	Implement BFS correctly.	10%	10%
2	Implement DFS correctly.	10%	10%
3	Implement UCS correctly.	10%	10%
4	Implement A* correctly.	10%	10%
5	Generate at least 10 test cases for each level with different attributes.	10%	10%
6	Result (output file and GUI).	15%	15%
7	Videos to demonstrate all algorithms for some test case.	10%	10%
8	Report your algorithm, experiment with some reflection or comments.	25%	25%
Total		**100%**	**100%**

Chapter 3

Search Problem Formulation

1. State Representation

Our state is a Python tuple consisting of:

- The position of the player (Ares).
- The positions of all stones on the grid.
- A string representing of the path (e.g. udlRul...).
- Total cost of all the previous states leading to the current state.
- A list of costs for all the previous states leading to this state (to display the weight for each step in the UI).
- (A* only) f, g and h costs.

2. Initial State

The starting position of Ares, the initial positions of stones, and the initial states of switches (if applicable).

3. Goal State

A state where all stones are on all switches (the number of stones is equal to the number of switches).

4. Actions

The player can move in four directions: up, down, left, or right.

- If a stone is adjacent to the player and there's an open cell in the same direction, the player can push the stone.
- Each action must:
  - Update the player's position.
  - Update stone positions if a push occurs.

5. Transition Model

Each action leads to a new state based on:

- The player's movement and the position of stones.
- Validity checks, ensuring moves that cause illegal configurations (e.g., pushing a stone into a wall, pushing a stone into another stone) are not allowed.

Costs can be weighted based on:

- The number of steps.
- The weights of stones.

6. Path Cost

For Uniform Cost Search (UCS), the path cost is based on:

- Total moves made by the player.
- Number of stones pushed.
- Additional weights associated with each stone.

For A* Search, the path cost is based on:

- $g(n)$ is calculated similar to the path cost of UCS.
- $f(n)$ is calculated by summing $g(n)$ and $h(n)$, with $h(n)$ being the heuristic.

Chapter 4

Search Implementation

4.1 DFS Algorithm

Initialization
This function starts by initializing a stack and a dictionary (`visited`) to track the smallest cost to the state.
Main Loop

- The algorithm pop the node we've push before, and then check goal state

4.2 BFS Algorithm

Initialization
This function starts by initializing a queue and a dictionary (`visited`) to track the smallest cost to the state.
Main Loop

- The algorithm op the node we've push before , and then check goal state

4.3 UCS Algorithm

Initialization
The function starts by initializing a priority queue to store nodes based on their cumulative cost. It initializes the queue with the player's starting position and the stones' initial positions, with an initial cost of 0. Additionally, a dictionary visited stores the minimum cost associated with each visited state (position and stones).
Main Loop

- The algorithm pops the node with the lowest cost from the priority queue, which includes the current position, stones' positions, cumulative cost, path taken, and weight history.

- It checks if all stones are positioned on their respective switches. If the goal is met, it stops the search, records performance metrics, and returns.

4.4 A* Search Algorithm

4.4.1 Heuristic

The heuristic function in this code calculates an estimated cost for reaching the solution state. It combines two factors to guide A* toward a solution effectively: the weighted Manhattan distance from each stone to its nearest switch and the minimum Manhattan distance from Ares to any stone.

The first part of the heuristic (weighted Manhattan Distance from each stone to its closest switch) estimates the difficulty of moving each stone to a switch based on its distance and weight. The heavier a stone, the more ”costly” it is to move.

For each stone in the list:

- Find the closest switch from that stone using the Manhattan distance: $distance = |x_{stone} - x_{switch}| + |y_{stone} - y_{switch}|$.
- Multiply this distance by the (stone's weight plus one). Adding one ensures there's a baseline movement cost, even for lightweight stones.
- Add this value to the total heuristic.

The weighted distance emphasizes moving heavier stones, which tend to require more effort to reach a switch, making this a key element in identifying ”cost-effective” moves first. The heavier the stone, the more influence it has on the heuristic score.

The second part of the heuristic (Manhattan distance from Ares to the closest stone) encourages A* to focus on the stones that are closest to Ares. Moving towards a stone that's closer reduces the time Ares spends traveling and reaching stones.

For each stone:

- Compute the Manhattan distance from Ares to that stone.
- Track the minimum distance among all stones.
- This minimum distance is added to the total heuristic, representing the ”entry cost” for Ares to start interacting with the stones.

This component makes the heuristic more goal-oriented by prioritizing configurations where Ares can immediately begin moving stones toward switches. A* is encouraged to explore paths where Ares can engage with stones sooner, rather than paths where Ares would need to travel long distances to begin the puzzle-solving.

Admissibility

Since each weighted Manhattan distance is an estimate of the minimum moves required to reach a switch and all stones must be placed on switches to reach a goal state, this heuristic is admissible.

Some Observations

This heuristic is particularly suited for puzzles with varying weights, as it incorporates the extra cost of moving heavier stones. Without weighting, A* might favor lighter stones and spend resources on less optimal moves.

4.4.2 Implementation Details

Initialization
The function starts by initializing a priority queue (frontier) and a dictionary (reached) to track visited states. The state is saved as tuple of values.
Main Loop

- The algorithm pops the node with the lowest cost from frontier, checking if all stones are on switches. If successful, the solution is returned with information about execution time, memory usage, path, and total weight moved.
- If not, it generates valid neighbor nodes.
- For each neighbor it computes the tentative cost and updates frontier with the new state if it improves upon a previously recorded path to the same state.

Neighbor Generation
This phase generates possible moves for Ares, accounting for pushing stones. It incorporates a check for ”corner deadlocks” to avoid configurations where a stone is pushed into an immovable position, unless that location contains a switch.

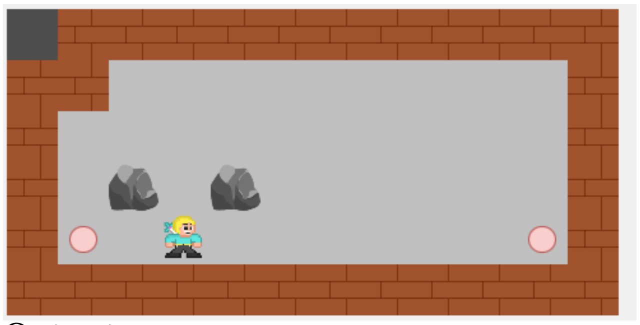
Chapter 5

Test case

With V representing vertices and E representing edges, C^* be the cost of the optimal solution,ϵ be the lower bound of the cost of each action ($\epsilon > 0$). We have the time and space complexities of four algorithms as follows:

Time complexity
BFS : $O(V + E)$
DFS : $O(V + E)$
A* : $O(Elog(V))$
UCS : $O\left(b^{1+\lceil \frac{C^*}{\epsilon} \rceil}\right)$
Space complexity
BFS : $O(V + E)$
DFS : $O(V)$
A* : $O(V)$
UCS : $O\left(b^{1+\lceil \frac{C^*}{\epsilon} \rceil}\right)$

5.1 Test 1



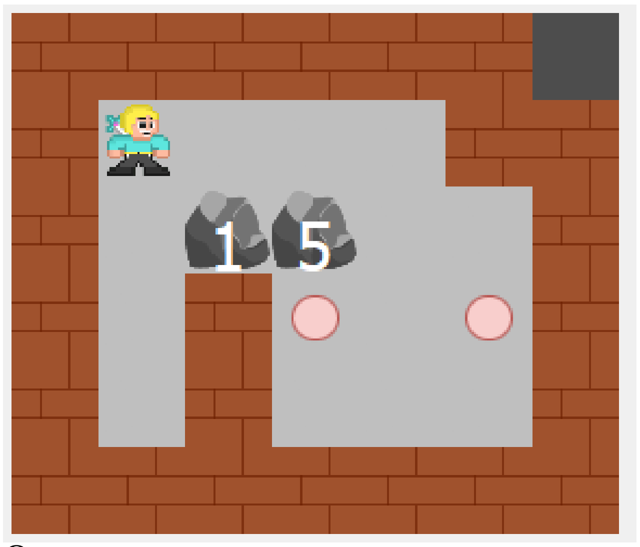
```
1 99
 ###########
##         #
#          #
# $ $      #
#. @      .#
############
```

Output
BFS
Steps: 16, Weight: 711, Node: 15391, Time (ms): 379.85, Memory (MB): 4.11
uLulDrrRRRRRRurD
DFS
Steps: 85, Weight: 792, Node: 1564, Time (ms): 30.72, Memory (MB): 0.10
rrrrrrruulllllllldRdrrrrrrrruullllllDRRRRRRllllllldRRRRRlllllluurrrrrrrrrDldLLLLLLL
UCS
Steps: 24, Weight: 429, Node: 37313, Time (ms): 2650.61, Memory (MB): 6.63
lluRurrDrdLLLuRRRRRRRurD
A*
Steps: 24, Weight: 431, Node: 223, Time (ms): 20.36, Memory (MB): 0.13

uurDrdLLLUluRRRRRRRRurDD

- **Map Description**: This is a 12 × 6 map with 2 stones and 2 switches - a medium-sized map with a small number of stones. The stones are closer to the switch on the left. The main challenge of this map is for the agent to prioritize moving the heavier stone to the left switch, despite the lighter stone being closer.
- **Experimental result**:
  - BFS reached the solution with fewest steps, yet it was not aware of the stone weights, thus, opted to move the stones to their closest switches. This is due to BFS favors the solution with the fewest step to solve. BFS requires more memory because it explores all neighbors of a node before moving to the next level. This results in storing all frontier nodes in memory.
  - Similarly, DFS also led to a suboptimal solution as it is not aware of the stone weights. One observation from this test is that due to the right direction neighbor being added last to the stack, it is the first one to be popped out, hence the agent rapidly moves right from the start. The same can be said for other directions. DFS requires very little memory due to it only keeping track of a single path from the root to a leaf node at any given time.
  - UCS had the optimal solution, yet explored the most nodes and was the slowest algorithm. UCS requires more memory mostly because it visits the cheapest state first, wasting time moving the lighter stone (due to it costing less).
  - A* behaved similar to UCS, however, in addition to cost, it also accounts for the heuristic, which favors the heavier stone moved to the left switch due to its significant weight (99) compared to the other stone (1). If the weight gap was smaller, (for example, 1 and 2), the heuristic would actually favors the lighter stone to the left switch, due to the difference in distance now taking over.

5.2 Test 2



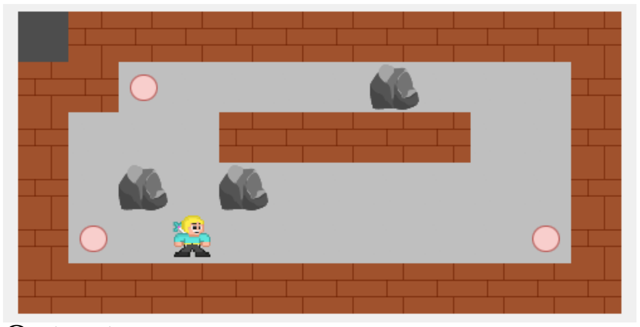
```
1 5
######
#@   ##
# $$  #
# #. .#
# #   #
#######
```

Output
BFS
Steps: 23, Weight: 42, Node: 1083, Time (ms): 24.57, Memory (MB): 0.22
rrDulldRRurDrddllURuulD
DFS

Steps: 29, Weight: 58, Node: 4746, Time (ms): 65.95, Memory (MB): 0.14
rrDulldRRurDrddllURuLulldRurD
UCS
Steps: 23, Weight: 42, Node: 1008, Time (ms): 14.92, Memory (MB): 0.07
rrDulldRRurDrddllURuulD
A*
Steps: 23, Weight: 42, Node: 167, Time (ms): 4.51, Memory (MB): 0.06
rrDulldRRurDrddllURuulD

- **Map Description**: This is a 7x6 map with an obstacle in the middle, two switches near the center, and two stones in the left corner. The main challenge of this map is for the agent to prioritize moving the heavier stone to the lower switch, then push the lighter stone in a way that avoids getting it stuck in the corner.
- **Experimental result**:
  - BFS reach the solution with the 23 steps and 42 weights like UCS and A*. But it take the most memory and time to run because the large node number of node in the frontier in this map.
  - DFS led to a suboptimal solution as it is not aware of the stone weight. The observation from this test is that DFS worst in case we need to move stone through breadth instead of to the depth.
  - UCS had the optimal solution. Although explore to all node in the map, it is the second fastest and require less memory algorithm because map has small sized.
  - A* give the optimal solution. Because it also accounts for the heuristic, it give the result with the fewest step, node generated time and memory.

5.3 Test 3



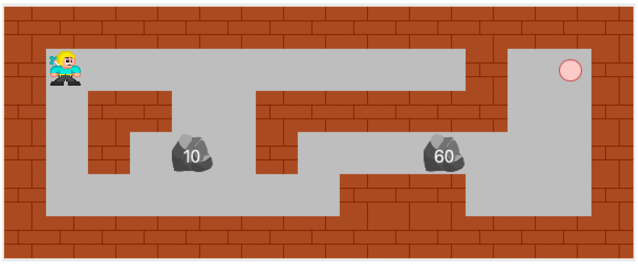
```
1 99 30
 ###########
##.    $   #
#   #####  #
# $ $      #
#. @      .#
############
```

Output
BFS
Steps: 25, Weight: 438, Node: 33106, Time (ms): 776.97, Memory (MB): 8.60
uLulDrrRRRRRurDuullLLLLL
DFS
Steps: 155, Weight: 628, Node: 1883, Time (ms): 34.14, Memory (MB): 0.15
rrrrrrruululLrrrddlllllLrrrrrruullllLrrrrddllllllldllluurrDrrrrrrruullllLrrrrrddlllllllLrrrr
rrrruulllllLrrrrrrddllllllllldRRRRRRRllllllluulDrrrrrrrrruullllllL
UCS
Steps: 25, Weight: 438, Node: 92066, Time (ms): 2762.62, Memory (MB): 21.74
uLulDrrRRRRRurDuullLLLLL
A*

Steps: 36, Weight: 509, Node: 3813, Time (ms): 127.82, Memory (MB): 1.82
uLulDrrdrruLrrrrruulLLLLLdDldRRRRRRR

- **Map Description**: This map is built based on the first map, but we add a long wall that turns the upper section into a long corridor, with an extra stone and a switch at the end of the corridor.
- **Experimental result**
  - As expected, BFS found the shortest path, which also happened to be the optimal path. It still used significant memory and was quite a bit slower, similar to test 1.
  - UCS also found the optimal path, though still faced the same problems as before.
  - DFS improves significantly due to the long corridor narrowing the number of next valid steps. The stone in the corridor also guaranteed that the path down that corridor is correct. This won't be the case if there are more corridors, especially ones with dead-ends.
  - A* was the most interesting case. Due to its heuristic favoring the stone weight and the distance from the agent to his nearest stone, the stone with 30 units of weight actually got pushed toward the direction of the switch on the top left corner before heading to the correct switch, leading to a rare sub-optimal solution. This would not be the case if that stone was lighter.

5.4 Test 4



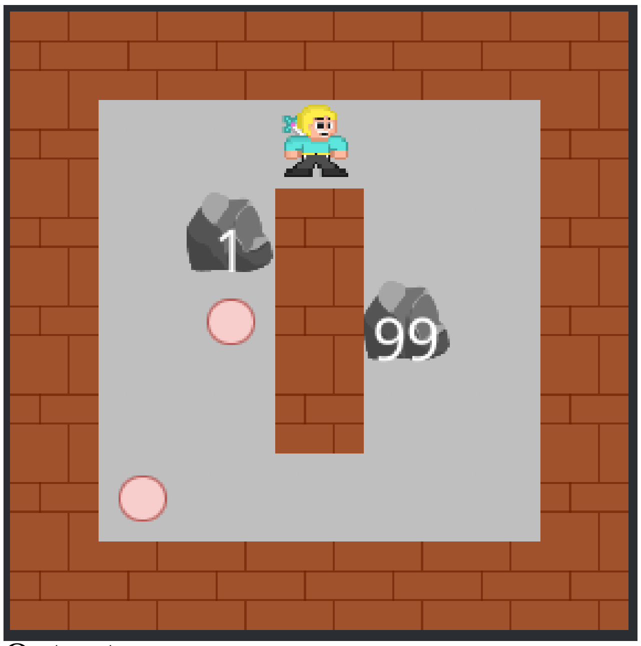
```
10 60
###############
#+         # .#
# ##  ######  #
# # $ #   $   #
#       ###   #
###############
```

Output
BFS
Steps: 31, Weight: 381, Node: 4404, Time (ms): 145.65, Memory (MB): 0.97
dddrrrUUruLLLrrdddrrurrRRRdrUU
DFS
Steps: 89, Weight: 459, Node: 12819, Time (ms): 335.96, Memory (MB): 0.25
rrrrdddrrurrRRRlllllldlllUrdrrurrrrrdrUllllldllllluuurrrDrddrrurrrrrUldllllldlllUUruLLL
UCS
Steps: 31, Weight: 381, Node: 8114, Time (ms): 232.16, Memory (MB): 1.17
dddrrrUUruLLLrrrdddrrurrRRRdrUU
A*
Steps: 31, Weight: 381, Node: 59, Time (ms): 10.26, Memory (MB): 0.03
dddrrrUUruLLLrrrdddrrurrRRRdrUU

- **Map Description**: This map builds on the findings of the last test, with more corridors and a dead end. This map also introduces a deadlock situation near the switch on the right.
- **Experimental result**:

  - With more corridors, BFS improved in performance due to tight space limiting the number of new neighbors generated.
  - DFS had significant challenges due to its tendency to go down a corridor, which may or may not be a dead end. In this example, knowing DFS has a tendency to travel right, a corridor with a dead end on the right was positioned right in front of the agent. Thus, DFS generated substantially more nodes.
  - UCS performed better for this map compared to the last map, due to moving lighter stones not leading to any pitfalls.
  - A* demonstrated great capability to navigate tight space with clear paths

5.5 Test 5



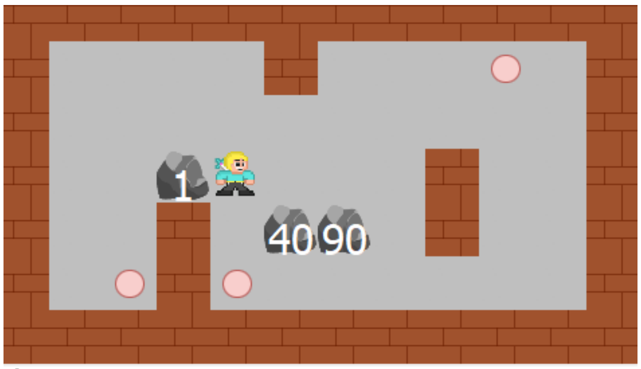
```
1 99
#######
#  @  #
# $#  #
# .#$ #
#  #  #
#.    #
#######
```

Output
BFS
Steps: 13, Weight: 509, Node: 384, Time (ms): 29.59, Memory (MB): 0.04
lDurrdDDrdLLL
DFS
Steps: 147, Weight: 1045, Node: 28317, Time (ms): 2766.57, Memory (MB): 0.42
rrdddlUrddllllluuuurDlddrdrrruuuulDrddldllluuurDlddrrrruuulDrddlllUUlddrrrruulDrdLruuluulll
ddrddRlluuuurDDluurrrrddddLruuluullldddrUluuurrrrddlddLL
UCS
Steps: 13, Weight: 509, Node: 2541, Time (ms): 238.96, Memory (MB): 0.19
lDurrdDDrdLLL
A*
Steps: 16, Weight: 512, Node: 229, Time (ms): 38.58, Memory (MB): 0.09
rdDDrdLLLuuluurD

- **Describe map**: This is a medium-level grid, it has been achieved with grids that are not so big but have lots of traps and the path is narrow. This calls for narrow approaches and many choke points: well, we can turn the stone but careful not to obstruct where they need access.
- **Experimental result**:

- BFS quickly found the solution with the minimum number of steps, and is also the most optimal solution with smallest weight.
- DFS found the solution with the maximum number of steps, and is also the least optimal solution. This map design very narrow path, so it makes DFS easier to find the solution than other wider path maps.
- UCS found the solution with the minimum number of steps, and is also the most optimal solution with smallest weight, but it takes more time than BFS because of the insert and pop operation in the priority queue takes more time than the queue in BFS.
- A* in this case not found the optimal solution, because the heuristic function is not good enough to estimate the cost to the goal.

5.6 Test 6



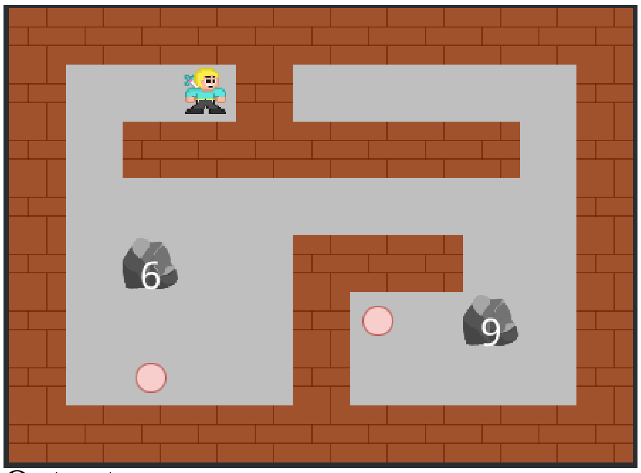
```
1 40 90
############
#    #   . #
#          #
#  $@    # #
#  # $$ #  #
# .#.      #
############
```

Output
BFS
Steps: 22, Weight: 645, Node: 224282, Time (ms): 16407.44, Memory (MB): 70.11 LulDDurrrDRdLrrUUUluRR
A*
Steps: 44, Weight: 762, Node: 41659, Time (ms): 1574.76, Memory (MB): 23.80 rDurrdLuullllldRRRRddLUruLLLulDDurrrrdrUUluRR
UCS
Steps: 33, Weight: 551, Node: 1451043, Time (ms): 54928.11, Memory (MB): 348.88 ulldRRRRDrdLLUrrUUluRRllddLLLulDD

- **Map description**: This is a 12x7 map with three stones of different weights and three switches. The stones are close to two of the switches, making it harder to achieve the optimal solution, as we must decide which stone to push to the nearby switches rather than the farther one.
- **Experimental result**:
  - BFS found the suboptimal solution with fewer weight than A*. This is because switch near the stone and BFS are good when searching in breadth.
  - On the other hand, we did not use DFS due to its poor performance. After spending three hours, it still yielded no result. On this map, moving the stones to the switches requires a lot of zig-zagging, which is a major weakness for DFS algorithms.

  - UCS found the optimal solution with the few steps and the lowest weight. However, UCS required more time and more nodes to explore compared to A* and BFS since it using a priority queue, which is slower than a queue in insertion and deletion operations.
  - A* quickly found a suboptimal solution due to the heuristic function. This is the first test case where A* has more steps and weight than both BFS and UCS. The use of the Manhattan and Ares heuristics caused the algorithm to push stone 40 (the stone nearest to Ares) to the nearest switch, stone 90 to the left switch behind an obstacle, and stone 1 to the far-right corner switch, which is not the optimal solution. Ideally, it should push stone 90 to the nearest switch, stone 40 to the second, and stone 1 to the far switch.

5.7 Test 7



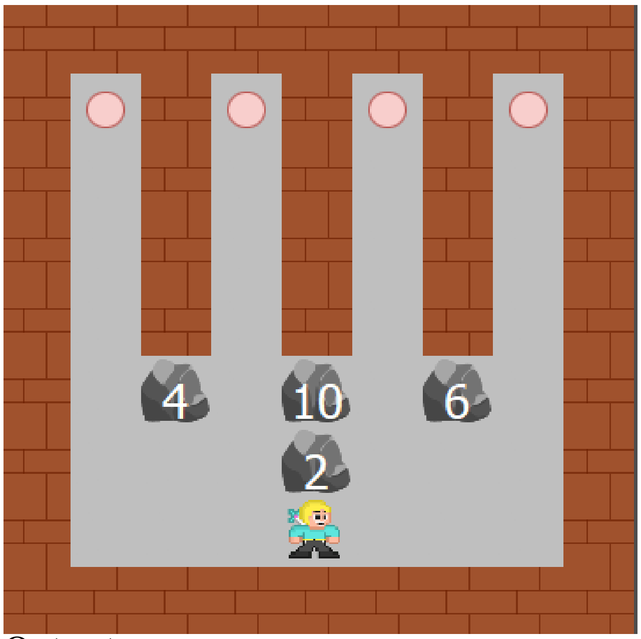
```
6 9
###########
#  @#     #
# ####### #
#         #
# $  ###  #
#    #. $ #
# .  #    #
###########
```

Output
BFS
Steps: 20, Weight: 50, Node: 1209, Time (ms): 105.36, Memory (MB): 0.30
llddrDDuurrrrrrdrdLL
DFS
Steps: 67, Weight: 109, Node: 83307, Time (ms): 8131.28, Memory (MB): 1.04
llddrrrrrrrddLruullllllllldRlddrrruuurrrrrddlLrruullllllDrdLruulldD
UCS
Steps: 20, Weight: 50, Node: 3006, Time (ms): 252.52, Memory (MB): 0.44
llddrDDuurrrrrrrddLL
A*
Steps: 20, Weight: 50, Node: 49, Time (ms): 10.80, Memory (MB): 0.02
llddrDDuurrrrrrdrdLL

- **Describe map**: This is a medium-difficulty grid with a compact size but many narrow paths and potential traps. The layout demands precise movements to avoid blocking critical paths. Stones must be carefully maneuvered to reach the switches without access to the blocked areas.
- **Experimental result**:
  - BFS found the optimal solution with the fewest steps and the lowest weight but required more time and more nodes to explore compared to A*.

- DFS found the solution with the maximum number of steps and the highest weight, making it the least optimal. However, due to the narrow paths in this map and a blocked area near the goal, DFS was able to find the solution easy.
- UCS found the optimal solution with the fewest steps and the lowest weight, similar to BFS. However, UCS required more time and more nodes to explore compared to A* and BFS since it using a priority queue, which is slower than a queue in insertion and deletion operations.
- A* quickly found the optimal solution with the fewest steps and the lowest weight, and also fewest nodes to explore. A* is the best choice for this map due to its efficiency in finding the optimal solution.

5.8 Test 8



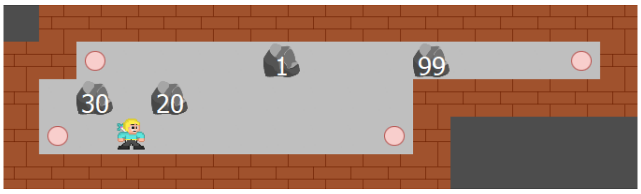
```
4 10 6 2
#########
#.#.#.#.#
# # # # #
# # # # #
# # # # #
# $ $ $ #
#   $   #
#   @   #
#########
```

Output
DFS
Steps: 286, Weight: 606, Node: 14009, Time (ms): 271.22, Memory (MB): 0.95
rrruuLrddllllllluuRlddrrrruLrUrrddllllluRRRRlllURRllllddrrrrrrUllllluUdrrdrrULrd
dllllllluurrrRlluUddrrdrruLrddllllllluurruuUdddrRllllddrrrrrruuLrUdlLrrddlllll
luurrRRUdllllddrrrrrruuLruUddlLrrddllllllluurrRRllllddrrrrrruuuuUdddLrddlllll
luurrrRuUddllllddrrrrrruuLLLruuUdddlLLrrrrddllllllluUUUU
A*
Steps: 52, Weight: 164, Node: 93, Time (ms): 4.00, Memory (MB): 0.08
luRluLdlUUUUdddrrdrdrUULrRlUUUdddldlUUUUdddrrrdrUUUU
UCS
Steps: 42, Weight: 154, Node: 3818753, Time (ms): 149903.97, Memory (MB): 719.47
luRdrUURdrUUUUdddllUUUdddLdlUUUUdddLdlUUUU
BFS
Steps: 42, Weight: 154, Node: 2812023, Time (ms): 269201.17, Memory (MB): 590.20
luRdrUUUUUdddRdrUUUUdddllLdlUUUUdddLdlUUUU

- **Describe** map: The player starts in an open space with 4 stones. The map is designed so that the player has to push the stones to the switches at the end of each narrow corridor. This design choice heavily influenced the performance of the algorithms.

- **Experimental result**:
  - Upon a first look, UCS and BFS returned the most optimal path, followed by A* then DFS. However, a closer inspection of the time consumed for each algorithm reveals that A* was the fastest, followed by DFS, both of which were a wide margin from UCS and BFS respectively.
  - It should be noted that:
    - A* was the fastest. Since this is a relatively small map, the heuristic guidance of the algorithm helped it achieve a fast solution by not expanding unnecessary nodes and focusing on the solution. However, the path returned was not optimal (164 as opposed to the 154 of UCS and BFS), this is likely because of the heuristic chosen.
    - DFS was the second fastest despite having the longest path. This map contains long corridors with not a lot of options for movement, the layout makes it so that the algorithm does not spend a lot of time expanding and exploring paths leading to non-solution. However, as the algorithm itself does not take into account the optimality of the path, the returned solution bodes both the highest cost and the most amount of steps.
    - BFS and UCS both returned a similar solution. Yet, UCS had a faster time while BFS explored fewer nodes:
      - The difference in nodes is clear because the UCS algorithm finds the optimal path based on the distance from the starting position. This inherently means that UCS has to explore until it finds the most optimal solution, whilst BFS does not have this trait as it does not account for the costs of the actions.
      - The difference in time consumed is harder to explain. This is most likely because UCS, as it prioritizes based on cost, may have a more efficient memory access pattern than the level-order expansion of BFS. Moreover, it could also be that the priority queue data structure of UCS was more optimized, which allowed for a better access time, this, however, is only conjecture.

5.9 Test 9



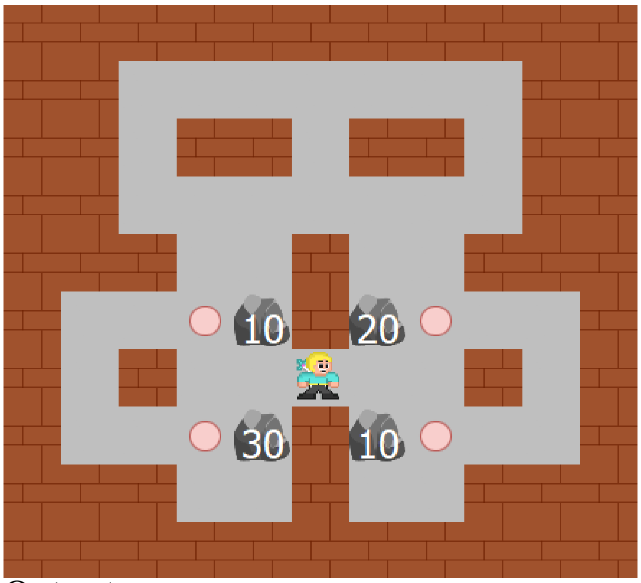
```
 1 99 30 20
 ################
##.    $   $   .#
# $ $      ######
#. @      .#
############
```

Output
A*
Steps: 56, Weight: 701, Node: 522493, Time (ms): 23486.48, Memory (MB): 129.18 rruLurrRRdrruRRRRllllLLLLdllullDRdLururDldRRRRRRuullllLLL
UCS
Steps: 33, Weight: 634, Node: 981003, Time (ms): 35866.26, Memory (MB): 203.29 uulDrRRRRRRurDuRRRRllllllLLLLLddL
DFS

Steps: 250, Weight: 1017, Node: 14099507, Time (ms): 336879.10, Memory (MB): 23.21
rrrrrrrruuRRRRllllllLrrrddlllllllluRdrrrrrrrrruulllLrrrrddlllllllluurrRRRRlllll
lddrrrrrrrrruuLrddllllllllluurrDrrrrruLrrddlllllLrrrrrruullLrrrddllllllLrrrrrr
ruullllLrrrrddlllllllLrrrrrrrrruullllLrrrrrddllllllluRRRRRRRllllllluRRRRllllld
drrrrrrruurDluLLLLLL
BFS
Steps: 33, Weight: 634, Node: 514651, Time (ms): 15985.86, Memory (MB): 132.79 uulDr-
RRRRRRurDuRRRRllllllLLLLLddL

- **Describe map**: The player starts in a rectangular room with four circular switches and four stones with varying weights. Upon a closer look, we can see that the map is wider than it is high, and there is a ”tunnel” of sorts on the right side of the map, this also impacts the performance of the algorithms.
- **Experimental result**:
  - From the output, we can see that BFS returned the most optimal path in the least amount of time, followed by UCS taking more time and returning the same result, and A* with a less optimal solution but less time than UCS. DFS performed the worst in terms of optimality and time.
  - From the map and the output we can interpret the algorithms:
    - BFS performed the best here because of the simplicity of the algorithm. We can see that while UCS and BFS returned the same solution, BFS was faster since it did not perform more sophisticated cost calculations. The layout of the map also contributed to BFS’s performance, because the solution itself is a non-complex one, requiring only 33 steps.
    - UCS performed worse than BFS because of the complicated cost calculations. UCS also uses a priority queue and a late-goal test (the goal is only checked when the node is expanded)
    - A* did not return an optimal result because of the chosen heuristic for this algorithm.
    - DFS performed the worst here because it kept visiting paths that led to a dead-end, then backtracked, and then visited another dead-end. This is because DFS lacks the heuristic or cost calculation to determine the efficiency of a path before it traverses that path. This is different from BFS because the layout of the map does not favor traversing down one path first since the switches are at the corners of the map.

5.10 Test 10



```
10 20 30 10
###########
##       ##
## ## ## ##
##       ##
###  #  ###
#  .$#$.  #
# #  @  # #
#  .$#$.  #
###  #  ###
###########
```

Output
A*
Steps: 110, Weight: 520, Node: 395738, Time (ms): 18131.50, Memory (MB): 314.20
llddrUluuururrdrddllLUlDuuurrrdDrdLLLdLUUllddRluurrrdrruuullDDldRRluuulDrddrRu
rrrddlldlUrUUluullddd rRurDuluurD

- **Describe map**: The player starts of surrounded by the stones and switches, each stone is located near a switch and an open space. A first look at the map shows that there are many narrow corridors, dead-ends and traps that can lead to non-solutions.

- **Experimental result**:

  - BFS, DFS, and UCS when tested on a 16GB RAM machine did not return a solution. A* was the only algorithm to return a solution with the limited resources (in theory, were there enough resources or better optimization, the other 3 algorithms could also find a solution).
  - A* found the optimal solution with 110 steps and 18131.50 ms. This is significantly better than the other 3 algorithms because:
    - A* is guided by a heuristic function, this helps the algorithm focus on moving the boxes closer to their goals. With the other 3 algorithms, the lack of a heuristic makes them explore more, which can lead to excessive expansion of nodes and repeated exploration of dead-ends.
    - The fact that A* is implemented with cost-awareness and a heuristic makes the algorithm less dependent on backtracking. Since the map has many dead-ends (non-goal states where the algorithms have to go back from) the reduced backtracking improves the performance of A* by a noticeable extent.

Bibliography

[1] Riverbank Computing Ltd, *PyQt5 Documentation*, Available online: `https://www.riverbankcomputing.com/static/Docs/PyQt5/`, Last accessed: 2024-11-09.

[2] Python Software Foundation. *Python Documentation*. Available at: `https://docs.python.org/3/`. Accessed: 2024-11-07.