VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY

UNIVERSITY OF SCIENCE

FACULTY OF INFORMATION TECHNOLOGY

PROJECT REPORT

Lab 1: Search

Subject: Fundamentals of Artificial Intelligence

Trịnh Thế Sơn - 20127617

UNDER THE GUIDANCE OF

Lecture: Pham Trong Nghia
TA: Nguyen Thai Vu

I) PROJECT INTRODUCTION

a) Problem description

The robot has been sent to a maze of size M x N, and the robot has to find the path from the Source (starting position) to the Goal (ending position). The robot allows to move in 4 directions: up, down, left, right. In the maze, there are some obstacles.
The student as asked to implement 5 search algorithms:

- Breadth-first search
- Uniform-cost search
- Iterative deepening search that uses depth-first tree search as core component and avoids loops by checking a new node against the current path.
- Greedy-best first search using the Manhattan distance as heuristic.
- Graph-search A* using the Manhattan distance as heuristic.

b) Input/output format

The format of the input file:

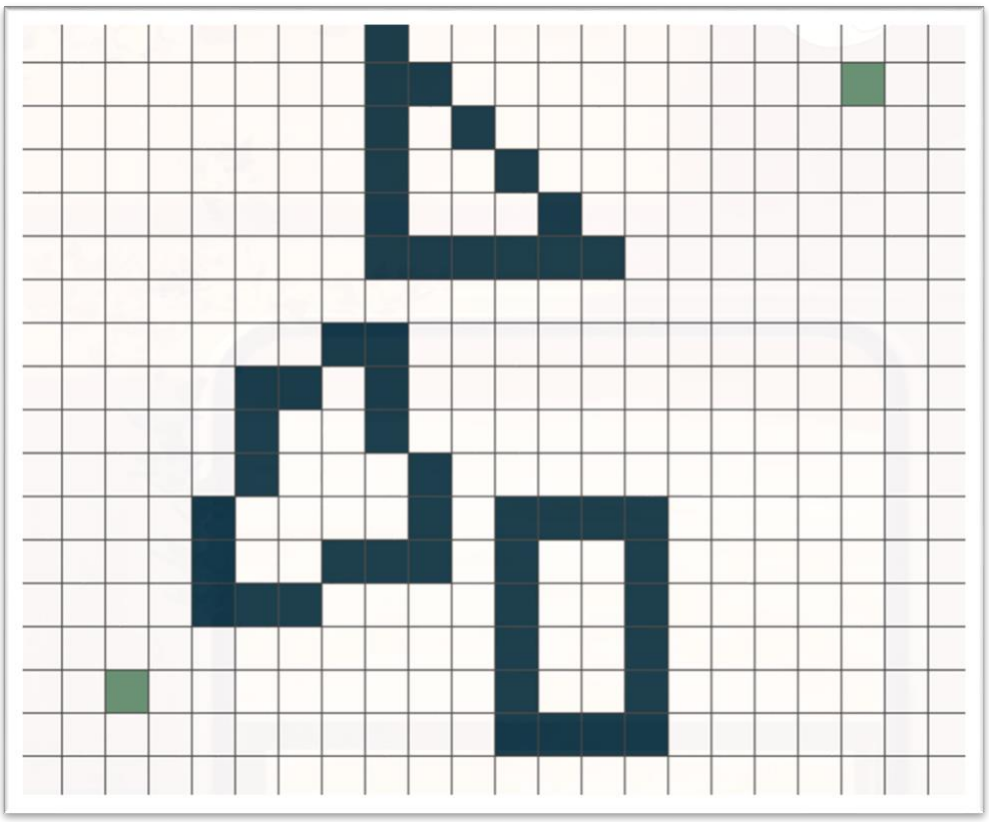
- First line: the size of the maze width, height.
- Second line: the position of the Source and Goal. For example: 2 2 19 16 meaning source point is (2, 2) and goal point is (19, 16).
- Third line: the number of the obstacles in the maze.
- The next following line, defining the obstacle by the rule:
  - The obstacle is a Convex polygon.
  - A polygon is a set of points that are next to each other clockwise. The last point will be implicitly concatenated to the first point to form a valid convex polygon.

The output:

- Graphical representation of polygons and path.
- Cost.

The example of input.txt (Everything is relative, depend on your implementation)

```
22 18
2 2 19 16
3
4 4 5 9 8 10 9 5
8 12 8 17 13 12
11 1 11 6 14 6 14 1
```



II) ANALYSIS

A. Breadth-first search

1. Idea

Breadth-first search (BFS) is a method for exploring a tree or graph. In a BFS, you first explore all the nodes one step away, then all the nodes two steps away, etc. Extra memory, usually a queue, is needed to keep track of the child nodes that were encountered but not yet explored.

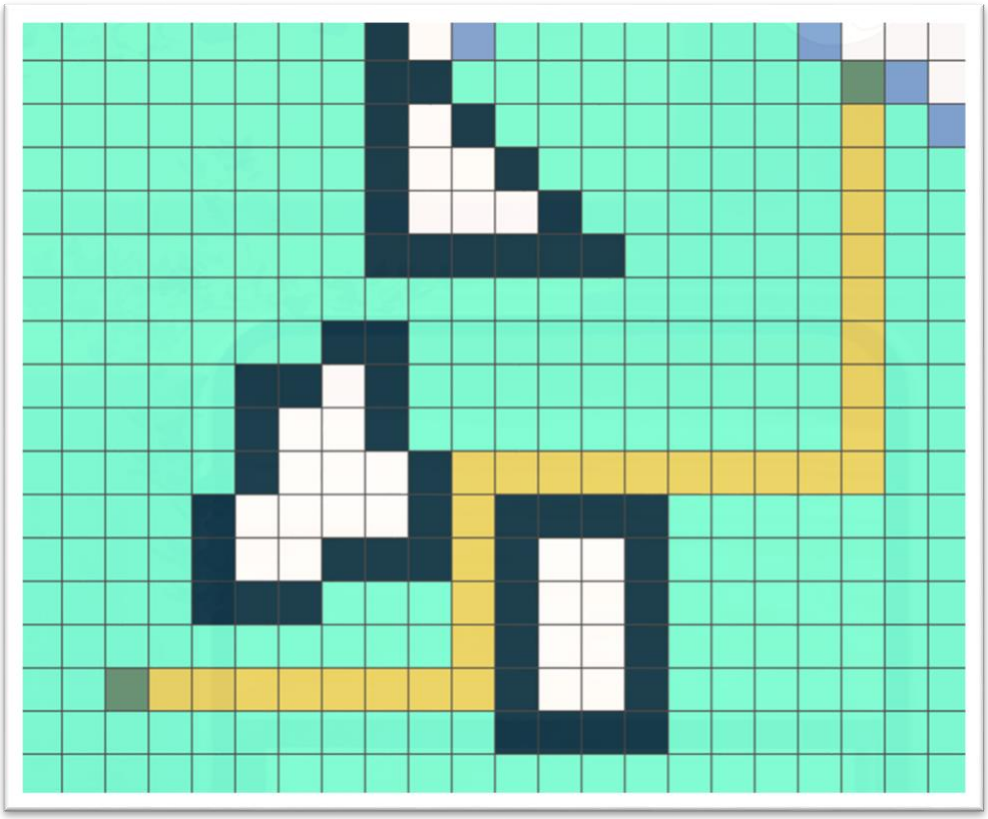
Breadth-first search is like throwing a stone in the center of a pond. The nodes you explore "ripple out" from the starting point.

Pseudocode

```
BFS(S, G):
    let Q be a queue
    Q.enqueue(S)
    S.visited = True
    while Q is not empty:
        let u be the first element of Q
        Q.dequeue()
        if u is Goal:
            return path
        else
            for each neighbor v of u:
                if v is not visited and v is not a wall:
```

```
            v.queue = True
            Q.enqueue(v)
```

2. Example



3. Conclusion

✓ **Pros**

- Always finds optimal solutions.
- Finds the closest goal in less time.
- Easy to implement.

× **Cons**

- Used lots of memory.
- When the search space is large the search performance will be poor compared to other heuristic searches.

B. Uniform-cost search

1. Idea

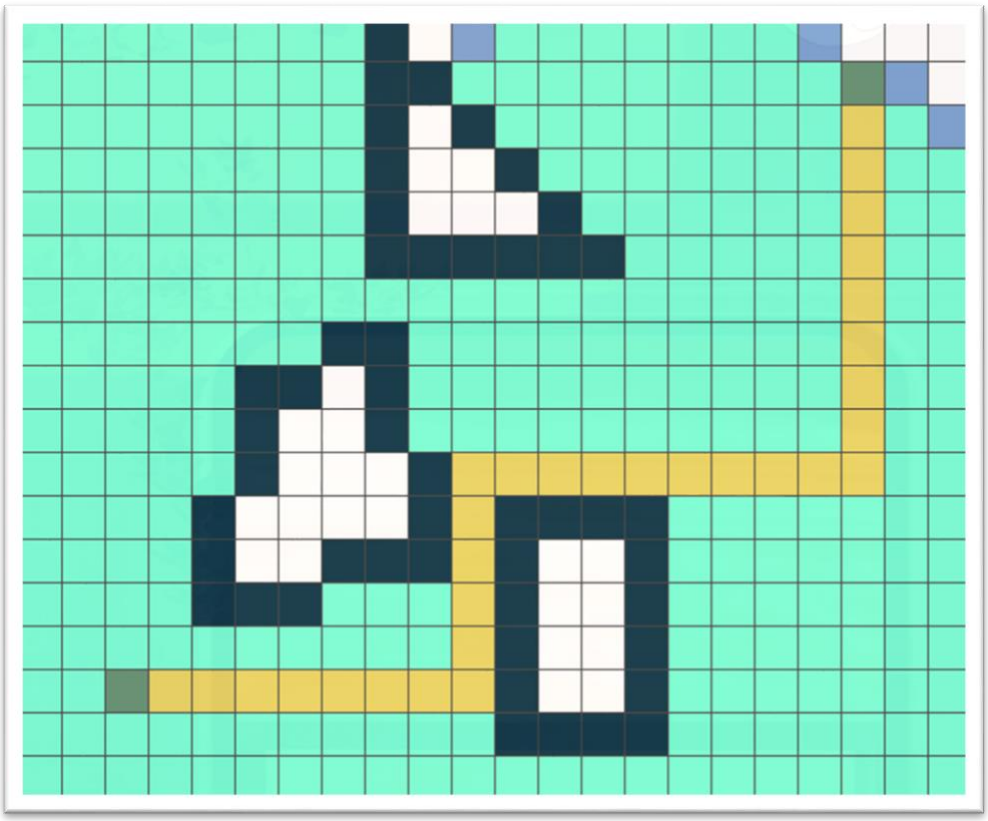
Uniform-cost search (UCS) is an uninformed search algorithm that uses the lowest cumulative cost to find a path from the source to the destination. Nodes are expanded, starting from the root, according to the minimum cumulative cost. The uniform-cost search is then implemented using a Priority Queue. Uniform cost search is equivalent to BFS algorithm if the path cost of all edges is the same.

Pseudocode

```
UCS(S, G):
    let Q be a priority queue
    Q.enqueue(S, 0)
    S.visited = True
```

```
while Q is not empty:
    let u be the first element of Q
    Q.dequeue()
    if u is Goal:
        return path
    for each neighbor v of u:
        if v is not visited and v is not a wall:
            v.queue = True
            Q.enqueue(v, g(u) + 1)
```

2. Example



3. Conclusion

✓ **Pros**

- Always finds optimal solutions.
- Easy to implement.

× **Cons**

- Used lots of memory.
- Only concerned about path cost, which this algorithm may be stuck in an infinite loop.

C. Iterative deepening search

1. Idea

Iterative deepening depth-first search (IDS or IDDFS) is a state space/graph search strategy in which a depth-limited version of depth-first search is run repeatedly with increasing depth limits until the goal is found. IDDFS is optimal like breadth-first search, but uses much less memory; at each iteration, it visits the nodes in the search

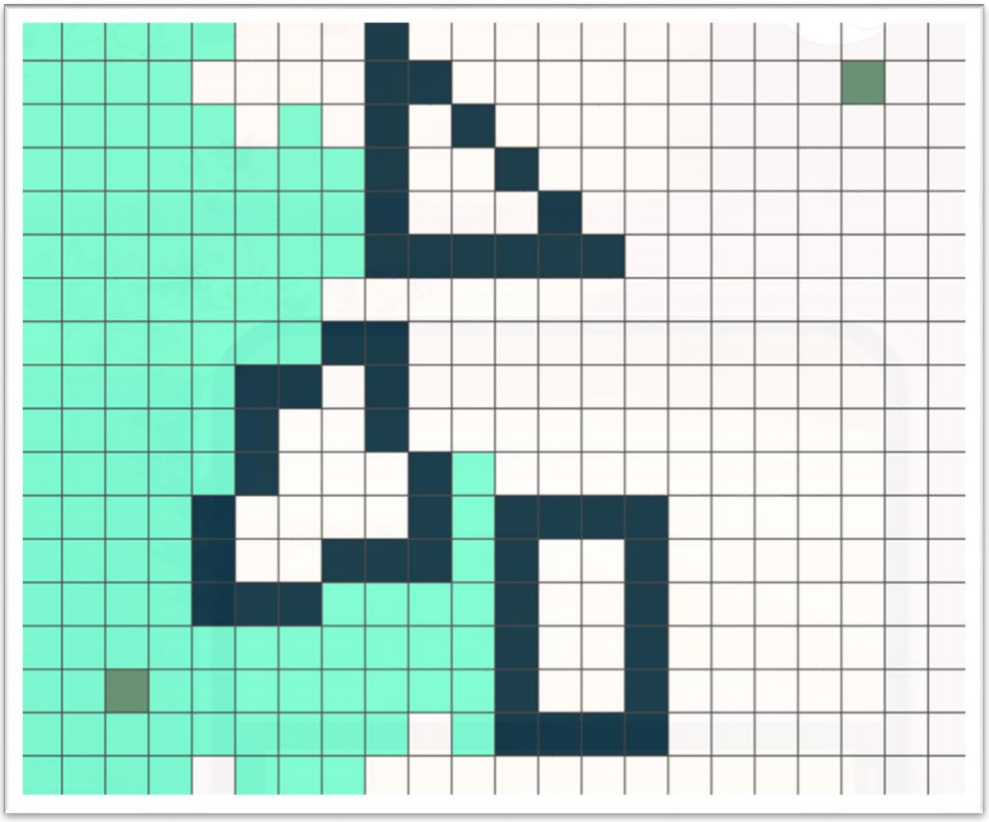
tree in the same order as depth-first search, but the cumulative order in which nodes are first visited is effectively breadth-first.

Pseudocode

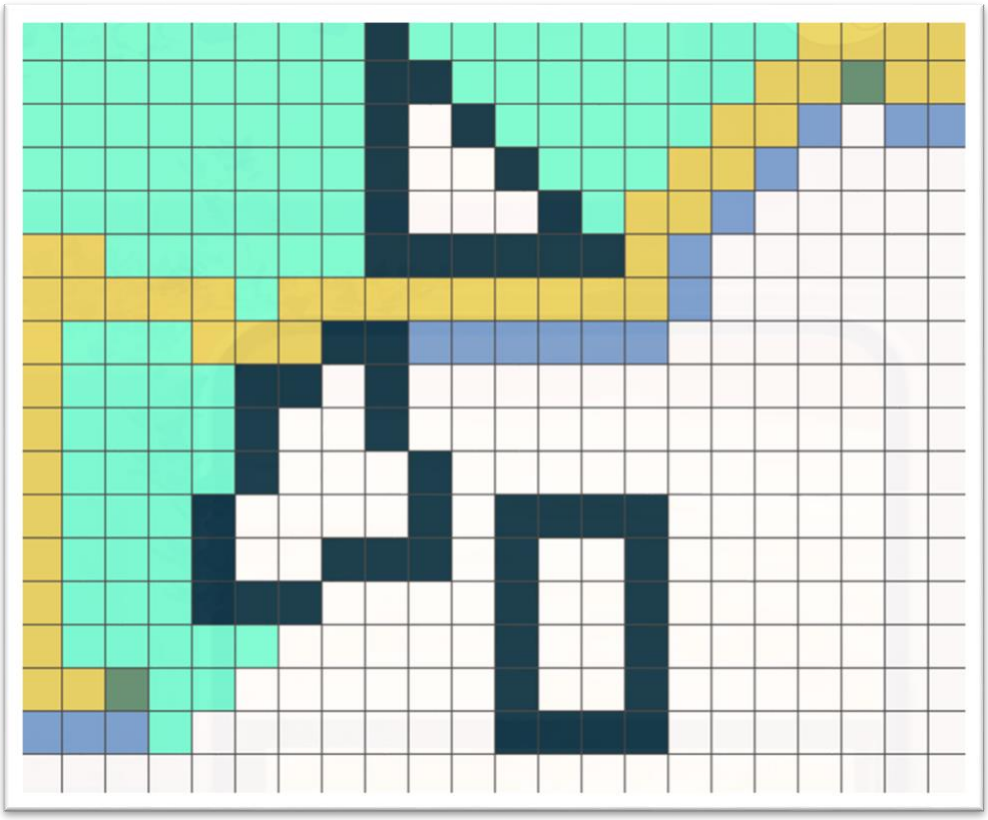
```
IDDFS(S, G, limit):
    let Q be a queue
    Q.enqueue(S)
    S.visited = True
    while Q is not empty:
        let u be the first element of Q
        Q.dequeue()
        if u is Goal:
            return path
        else
            if u.level > limit:
                continue
            for each neighbor v of u:
                if v is not visited and v is not a wall:
                    v.queue = True
                    Q.enqueue(v)
                    v.level = u.level + 1
```

2. **Example**

- Max-depth = 20



- Max-depth = 45



3. Conclusion

✓ **Pros**

- Uses much less memory than Breadth-First Search.
- Guaranteed to find an optimal path
- Solution can be found out by without much more search.

× **Cons**

- Not Guaranteed that it will give you solution.
- Cut-off depth is smaller so time complexity is more.
- Determination of depth until the search has proceeds

D. Greedy-best first search

1. Idea

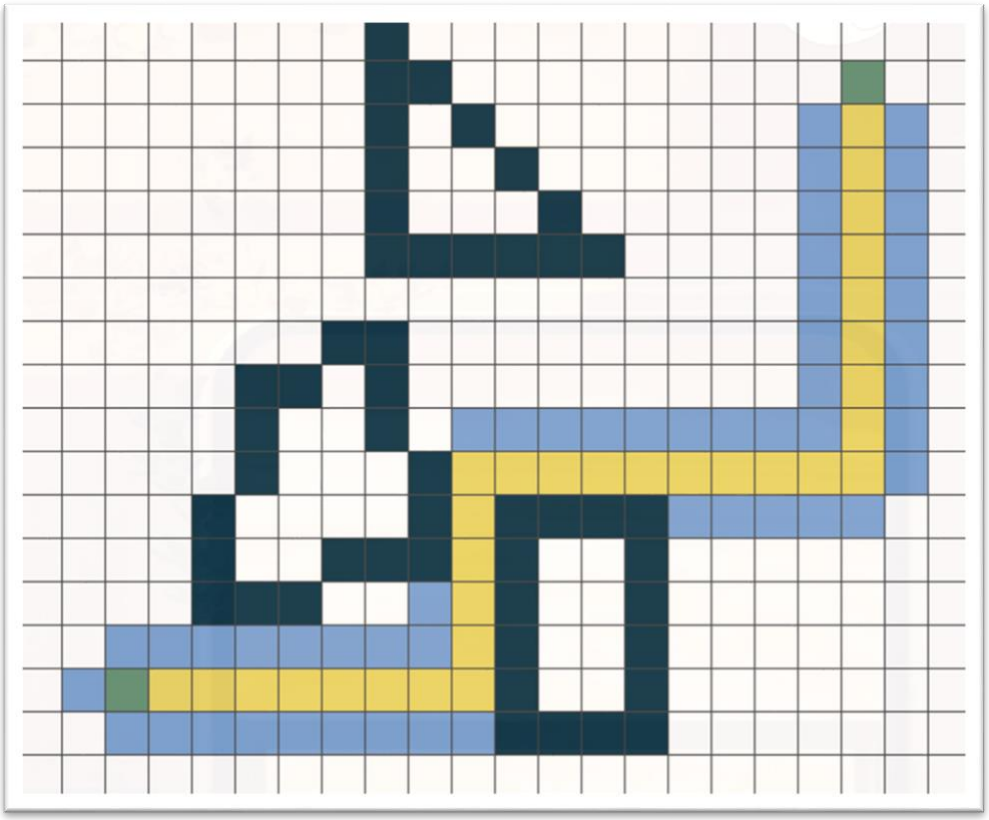
Greedy best-first search always selects the path which appears best at that moment. It is the combination of depth-first search and breadth-first search algorithms. It uses the heuristic function and search. Best-first search allows us to take the advantages of both algorithms. With the help of best-first search, at each step, we can choose the most promising node. In the best first search algorithm, we expand the node which is closest to the goal node and the closest cost is estimated by heuristic function.

Pseudocode

```
Greedy(S, G):
    let Q be a priority queue
    Q.enqueue(S, 0)
    S.visited = True
    f(S) = h(S, G)
```

```
while Q is not empty:
    let u be the first element of Q
    Q.dequeue()
    if u is Goal:
        return path
    else
        for each neighbor v of u:
            if v is not visited and v is not a wall:
                v.queue = True
                Q.enqueue(h(v, G), v)
```

2. Example



3. Conclusion

✓ **Pros**

- Can switch between BFS and DFS by gaining the advantages of both.
- More efficient than BFS and DFS.

× **Cons**

- It is not optimal.
- Can behave as an unguided depth-first search in the worst case scenario.
- It can get stuck in a loop as DFS.

E. A Star

1. Idea

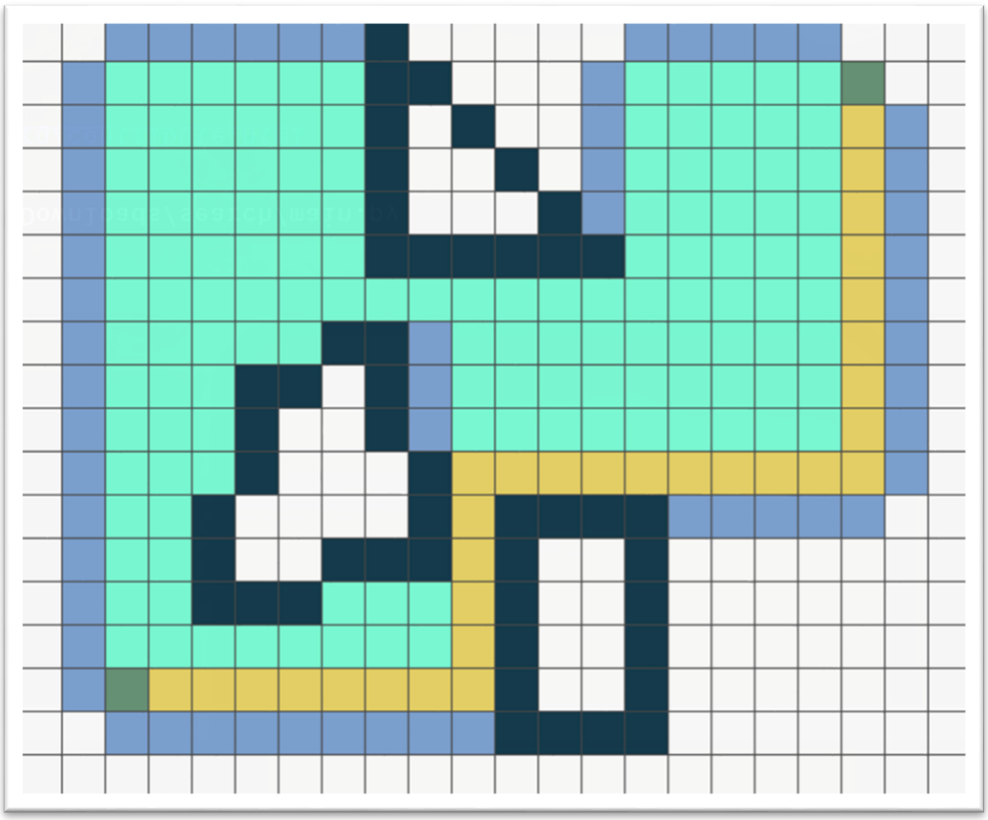
A Star search (A*) builds on the principles of Dijkstra's shortest path algorithm to provide a faster solution when faced with the problem of finding the shortest path

between two nodes. It achieves this by introducing a heuristic element to help decide the next node to consider as it moves along the path. Dijkstra's algorithm finds the shortest path between the start node and all other nodes. As well as being faster, the A* algorithm differs from Dijkstra's in that it seeks only the shortest path between the start node and the target node.

Pseudocode

```
AStar(S, G):
    let Q be a priority queue
    Q.enqueue(0, S)
    S.visited = True
    f(S) = h(S, G)
    while Q is not empty:
        let u be the first element of Q
        Q.dequeue()
        if u is Goal:
            return path
        else
            for each neighbor v of u:
                if v is not visited and v is not a wall:
                    v.queue = True
                    Q.enqueue(v, h(v, G) + g(v, u))
```

2. Example



3. Conclusion

✓ **Pros**

- It is one of the best heuristic search techniques.
- There is no other optimal algorithm guaranteed to expand fewer nodes than A*.
- It is used to solve complex search problems.

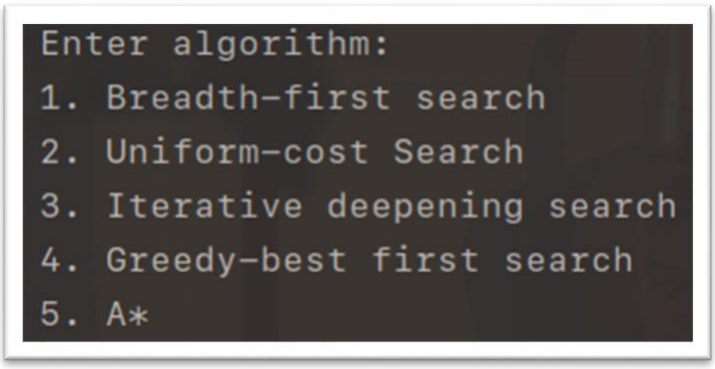
× **Cons**

- Complete if the branching factor is finite and every action has fixed cost.
- Performance is dependant on accuracy of heuristic algorithm used to compute the function h(n).

III) TUTORIAL

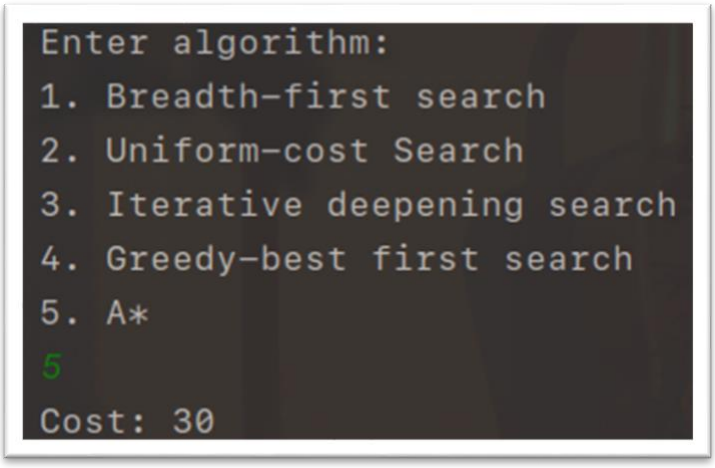
1. **Run program**

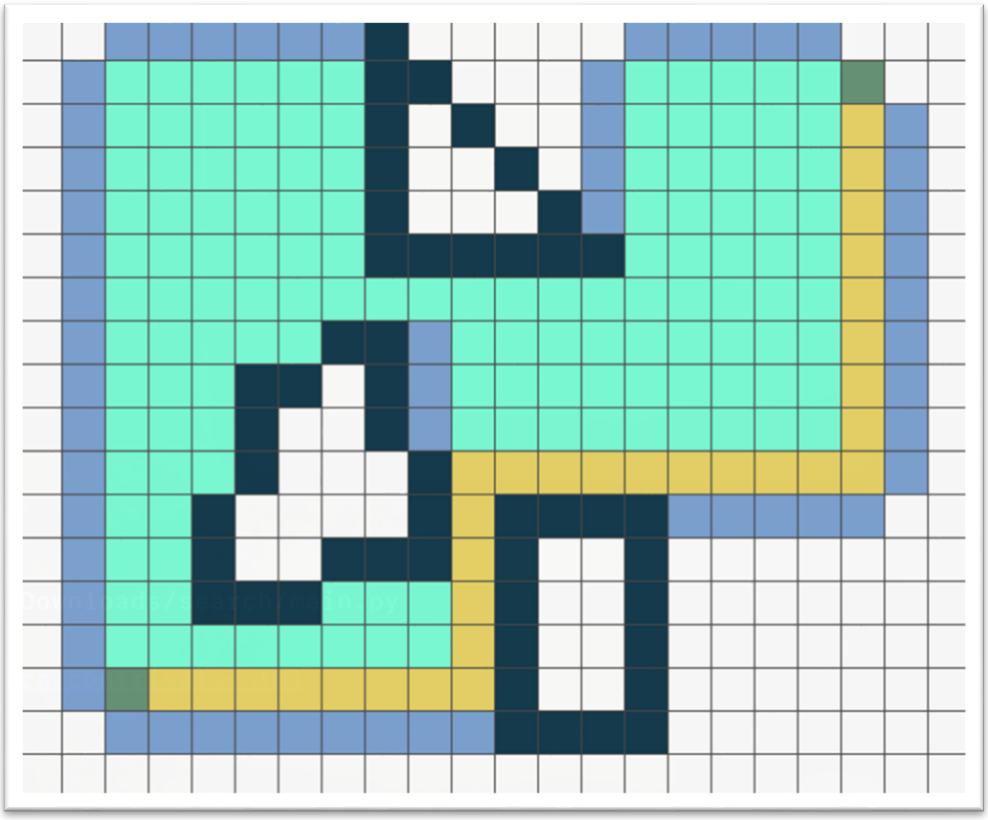
2. **Choose algorithms**



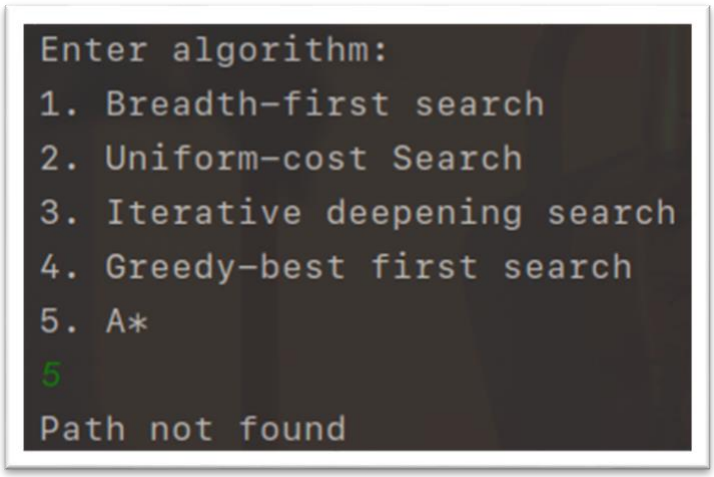
3. **Start visualize**

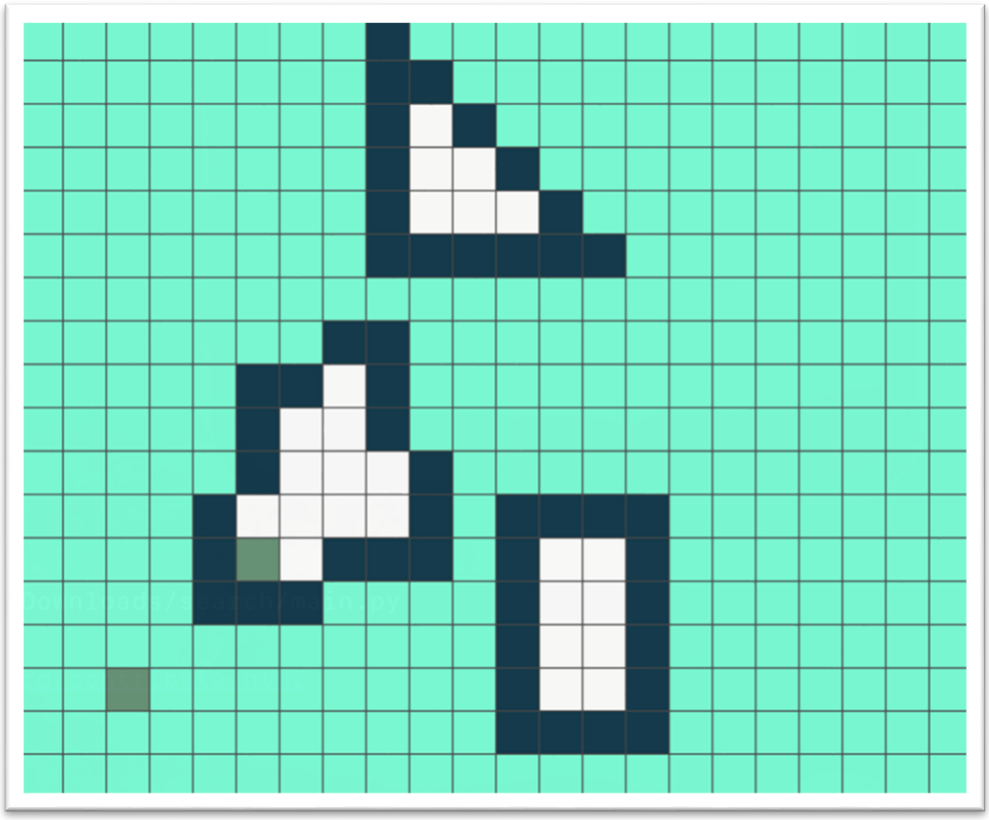
- Found Goal, return cost:





- Not found Goal, return “Path not found”





IV) BRIEF DESCRIPTION

- Programming language: python
- Python package: pygame
- IDE: PyCharm
- OS: Arch Linux

In ***main.py***, the input text file is read first and stored in to variabel *columns*, *rows*, *source_x*, *source_y*, *goal_x*, *goal_y*, *poly* and *polyList*.
To make a polygon implicity concatenated to form a valid convex, I'm using ***Bresenhen's*** algorithm.
I'm also define colour of block for visualization:

- Default: white
- Unvisited neighbours: light blue
- Visited: light green
- Source and Goal: dark green
- Wall: dark blue
- Path: yellow

Block: equivalent to a Node.
Grid [ ]: stored M×N Block.
Path [ ]: stored path from S to G.
Each function, variable are noted carefully as comment in my source code. This just is a brief, you need to read the source code for better understanding.

V) SUMMARY

Graph theory is an exceptionally rich area for programmers and designers. Graphs can be used to solve some very complex problems, such as least cost routing, mapping, program analysis, and so on. Graph search algorithms help you (or your friendly computer sidekick) traverse a graph dataset in the most efficient means possible. But “most efficient” depends on the results you’re looking for – BFS isn’t the most efficient if your results are better suited to UCS, IDDFS, Greedy search or A*. While graphs and our understanding of them are rooted in hundreds of years of mathematical study (i.e., graph theory), we continue to find new ways to apply them to our personal, social and business lives.

I'm sure there are some logical bugs and I’m unable to fix it in less than 2 weeks, but I’m trying my best to finish this Search project.

Overall: 90/100

VI) REFERENCES

Visualize. Youtube. https://www.youtube.com/watch?v=JtiK0DOeI4A&t=4517s

Visualize. Youtube. https://www.youtube.com/watch?v=QNpUN8gBeLY&t=1380s

BFS. Wiki. https://en.wikipedia.org/wiki/Breadth-first_search

UCS. Wiki. https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm#Practical_optimizations_and_infinite_graphs

IDDFS. Wiki. https://en.wikipedia.org/wiki/Iterative_deepening_depth-first_search

Greedy. Wiki. https://en.wikipedia.org/wiki/Best-first_search

AStar. Wiki. https://en.wikipedia.org/wiki/A*_search_algorithm