SHORTEST PATH ON A GRID

BFS | Dijkstra | A*

Algorithm and Lab -010

</>

TEAM MEMBERS

1. **Tillabaev Yosinbek** (25013516) | BFS algorithm, grid structure, UI Development |
2. **Obidov Islom** (21012944) | Dijkstra algorithm, custom heap |
3. **Rifat Md Redowan Al Rafi** (24013601) | A* algorithm, testing |
4. **Tuli Kazi Suraiya Tahmin** (24013581) | Timing analysis, slides |

PROBLEM STATEMENT

What we built:

- A program that finds the shortest path on a grid
- User draws walls and picks start/goal
- Program shows the path visually

Input: Grid with empty cells, walls, weighted cells
Output: Shortest path shown as highlighted cells

OUR APPROACH

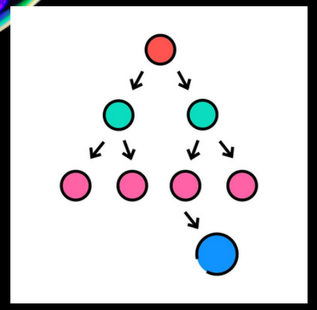
Grid as a graph:

- Each cell is a node
- 4 neighbors: up, down, left, right
- Walls block movement
- Weighted cells cost more to cross

Cell values:

- 0 = empty (cost 1)
- -1 = wall (blocked)
- >0 = weighted (cost = value)

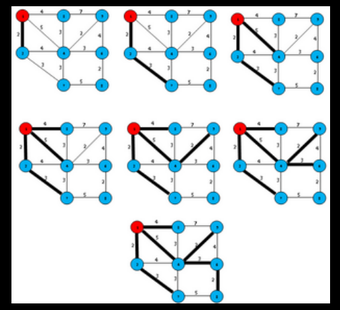
ALGORITHMS WE IMPLEMENTED



BFS (Breadth-First Search)

Breadth-First Search (BFS) is a graph-search algorithm that explores nodes level by level, starting from a source node. It visits all nodes at the current distance before moving to the next distance. BFS guarantees the shortest path in an unweighted graph or grid.

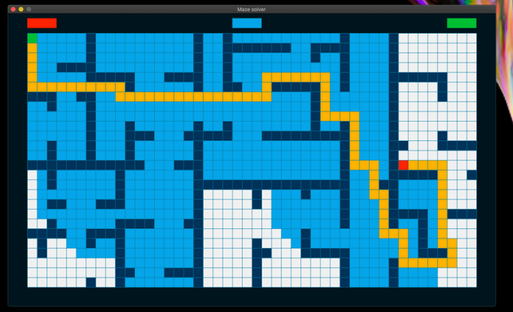
- For unweighted grids
- Explores level by level
- Guarantees shortest path



Dijkstra

Dijkstra's Algorithm is a shortest-path algorithm used to find the minimum-cost path from a starting node to all other nodes in a weighted graph with non-negative edge weights. It works by repeatedly choosing the unexplored node with the lowest known distance, updating its neighbors, and building the shortest path step by step.

- For weighted grids
- Always picks cheapest next step
- Uses priority queue (custom heap)



A* (Stretch Goal)

A* (A-star) is a pathfinding algorithm that finds the shortest path in a weighted graph by combining two things:

1. Actual cost so far from the start (called g)
2. Estimated cost to the goal using a heuristic (called h)

- Like Dijkstra but smarter
- Uses Manhattan distance to guide search
- Explores fewer nodes

DATA STRUCTURES

NO BUILT-IN HEAPQ, DICT, OR SET USED IN ALGORITHMS

We used:

- visited[] — boolean array, tracks seen cells
- parent[] — integer array, tracks path
- dist[] — integer array, tracks costs
- Custom MinHeap — for Dijkstra and A*

Index formula: id = row × width + col

COMPLEXITY ANALYSIS

Algorithm	Time	Space
BFS	O(V + E)	O(V)
Dijkstra	O((V + E) log V)	O(V)
A*	O((V + E) log V)	O(V)

V = number of cells

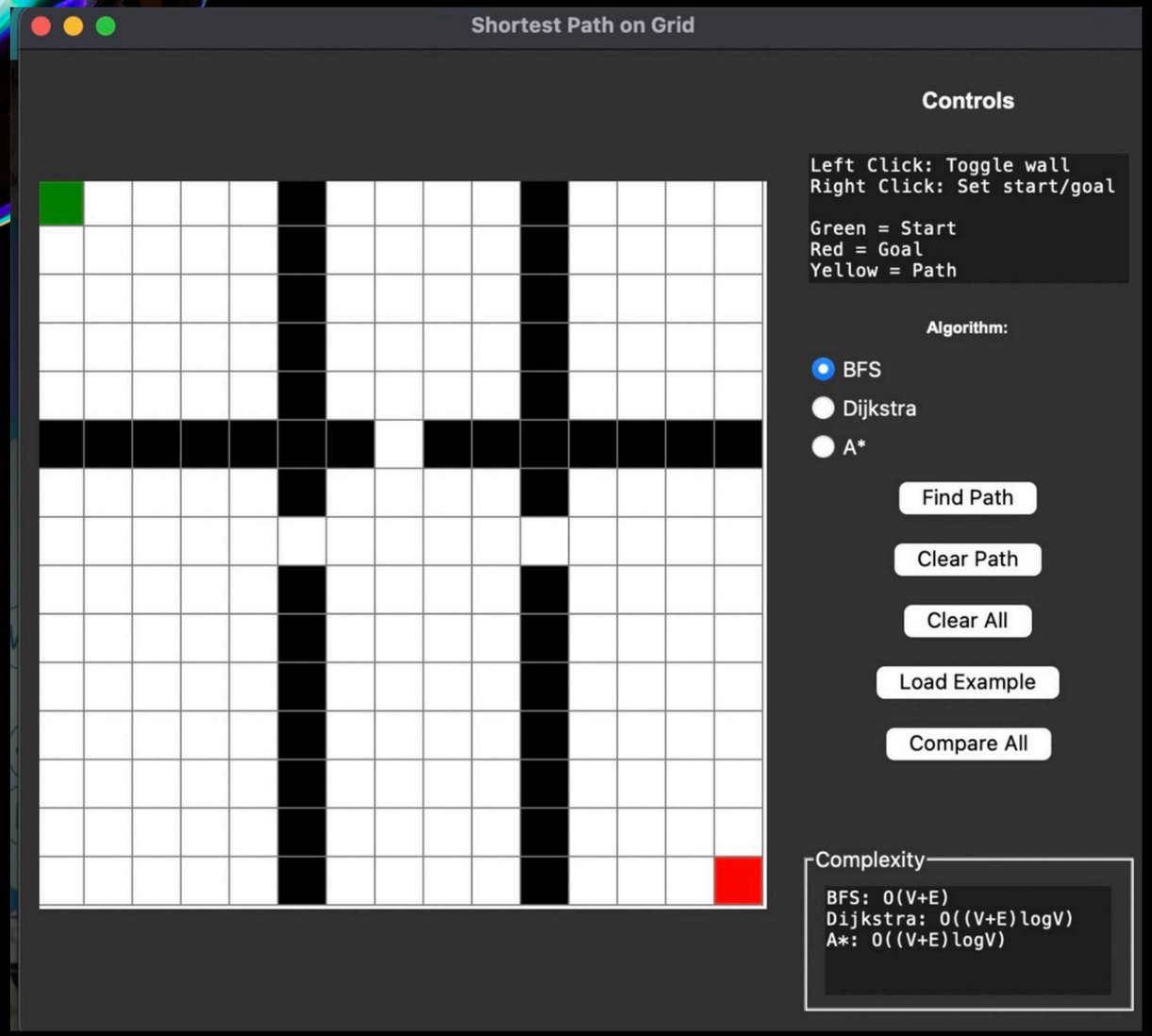
E = number of edges (4 per cell max)

A* explores fewer nodes due to heuristic guidance

EMPIRICAL TIMING

Grid	BFS explored	Dijkstra explored	A* explored
10×10	73	70	48
20×20	254	258	101
30×30	551	549	149

Observation: A* explores 3–4× fewer nodes than BFS/Dijkstra



USER INTERFACE

Built with Tkinter (Python GUI library)

Features:

- Left click — toggle wall on/off
- Right click — set start (green), then goal (red)
- Path shown in yellow
- Buttons: Find Path, Clear, Load Example, Compare All

Compare All: runs BFS, Dijkstra, A* and shows explored nodes

TESTING

Unit test cover:

- Open grid — finds correct path length
- Wall detour — path goes around walls
- Unreachable goal — returns "no path"
- Weighted detour — cheaper path preferred
- A* correctness — same result as Dijkstra

EDGE CASE:

- Goal completely separated by walls → algorithm explores cells, then reports “unreachable”

Run Tests:

python tests.py

RESULTS

WHAT WE ACHIEVED:

- ✓ All three algorithms work correctly
- ✓ Paths match expected shortest distance
- ✓ A* is most efficient (fewer nodes explored)
- ✓ UI is simple and easy to use
- ✓ All requirements met (from-scratch DS, tests, complexity)

CHALLENGES WE FACED

1. No heapq allowed
   → Built custom min-heap with bubble up/down

2. No dict/set for tracking
   → Used arrays indexed by cell ID

3. Handling blocked goals
   → Return None and show message in UI

4. Keeping code simple
   → Focused on readability over optimization

THANK YOU

Ready for question!