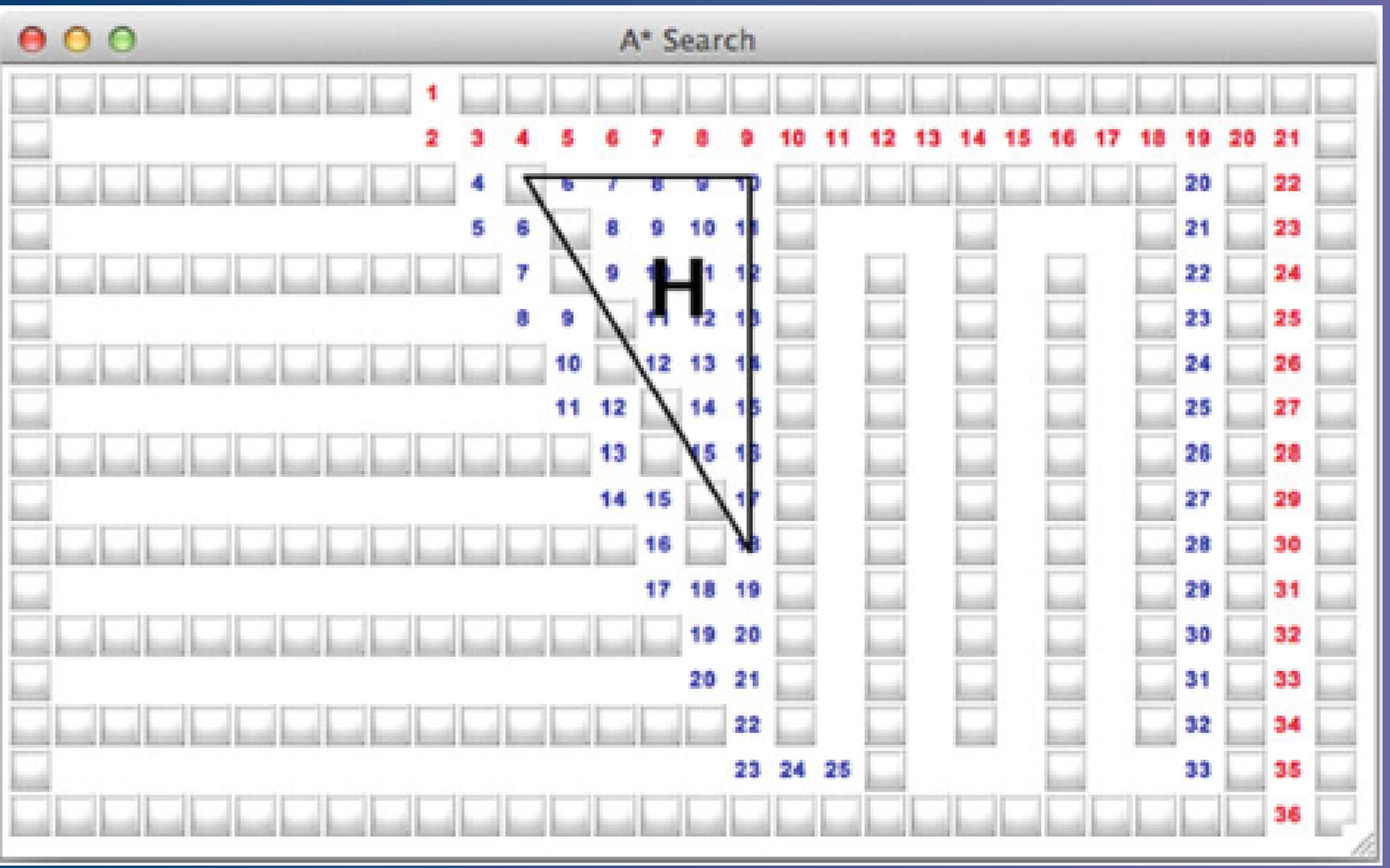
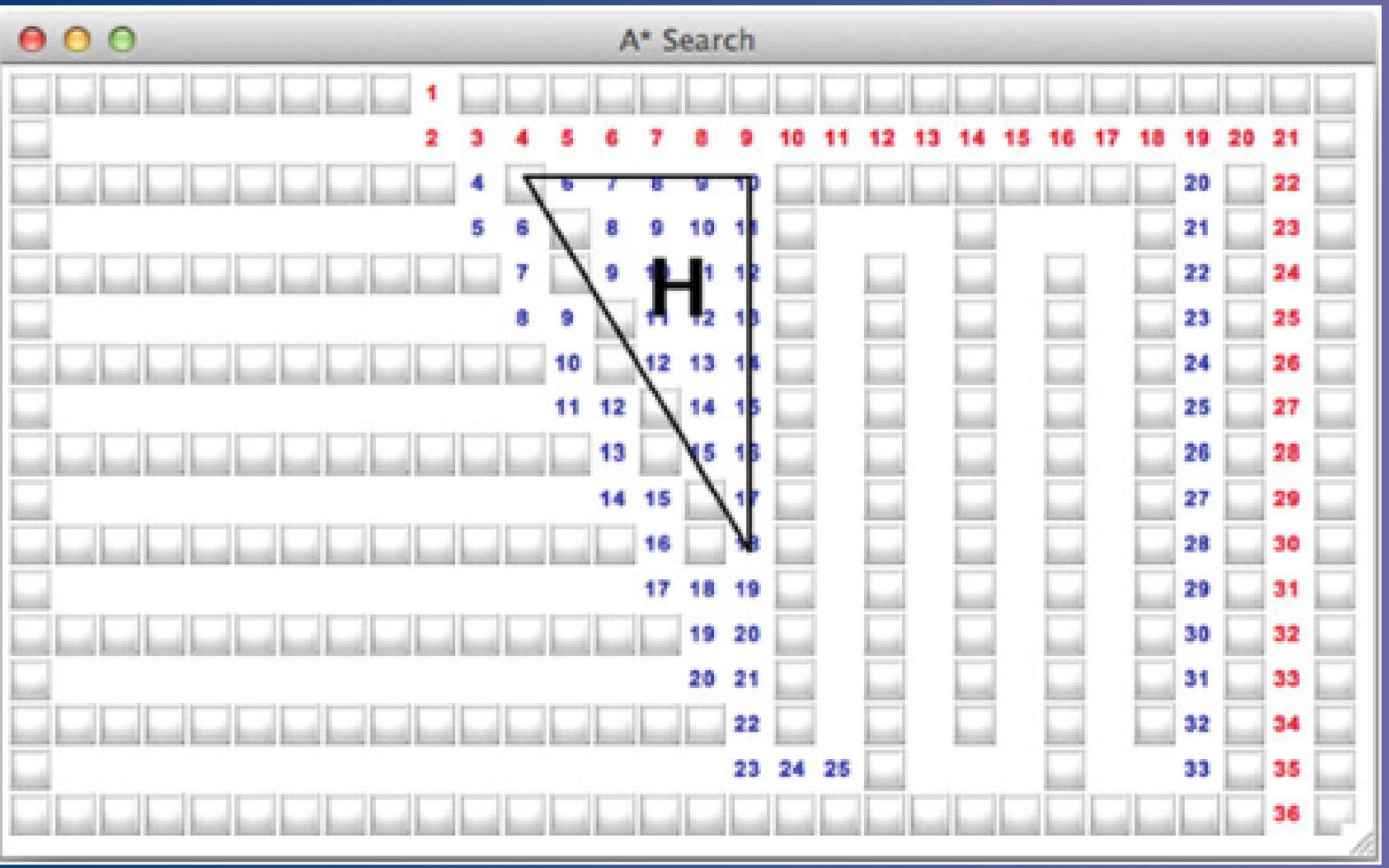
Best First Search of a Maze

GİRİŞ



BEST FİRST SEARCH OF A MAZE

Genel olarak, tepe tırmanışı, arama alanındaki konumların arama sırasına bağlı olarak en iyi öncelikli aramadan daha kötü sonuçlar verebilir. Ancak, ne tepe tırmanışı ne de en iyi öncelikli arama, genişlik öncelikli arama gibi optimal bir çözüm bulamadı. Her ikisi de labirentin ortasındaki uzun yola giderken sıkışıp kaldı.



A* ÖRNEĞİ

Şekilde, aynı yol ilk önce aşağıya ve sağa giderek labirentin alt kısmına, adım 25'e ulaşmaya çalışılır. Ardından, bu yol, kırmızı adımdaki adım 4'teki Manhattan mesafesiyle yolun uzunluğunun toplamından daha iyi olduğu için terkedilir. Tekrar arama aşağıya ve sağa doğru devam eder ve sonunda Şekil 12.1'deki H bölgesini doldurur. Bu noktada, arama, tekrar adım 19'dan üstten devam eder ve tekrar adım 33'e kadar iner, bu noktada kırmızı adım 20, sol taraftaki adım 34'ü almak yerine daha iyi görünür. Arama, mavi yolunu adım 33'te terk eder ve ardından kırmızı adım 20'den hedefe devam eder.

```python
import matplotlib.pyplot as plt
import numpy as np
import random
from queue import Queue

def create_maze(dim):
    # Create a grid filled with walls
    maze = np.ones((dim*2+1, dim*2+1))

    # Define the starting point
    x, y = (0, 0)
    maze[2*x+1, 2*y+1] = 0

    # Initialize the stack with the starting point
    stack = [(x, y)]
    while len(stack) > 0:
        x, y = stack[-1]

        # Define possible directions
        directions = [(0, 1), (1, 0), (0, -1), (-1, 0)]
        random.shuffle(directions)

        for dx, dy in directions:
            nx, ny = x + dx, y + dy
            if nx >= 0 and ny >= 0 and nx < dim and ny < dim and maze[2*nx+1, 2*
                maze[2*nx+1, 2*ny+1] = 0
                maze[2*x+1+dx, 2*y+1+dy] = 0
                stack.append((nx, ny))
                break
        else:
            stack.pop()

    # Create an entrance and an exit
    maze[1, 0] = 0
    maze[-2, -1] = 0

    return maze
```

ÖRNEK KOD