KECERDASAN BUATAN

Informed Search 8-PUZZLES

Team Member
Timothy Hosia (5025211098)
Yohanes Teguh (5025211179)
Shazia Naveeda (5025211203)

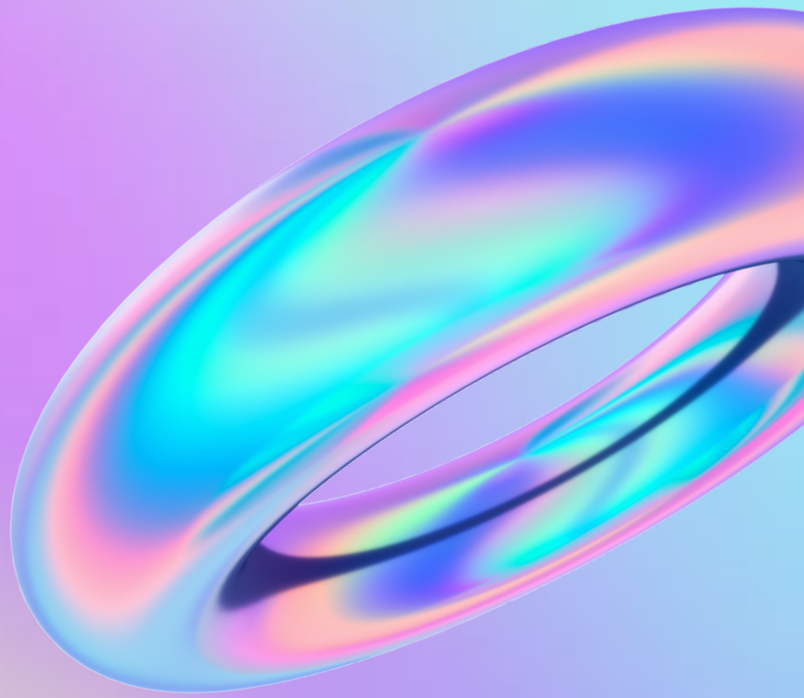
Presentation Highlights

Key points we will be discussing:

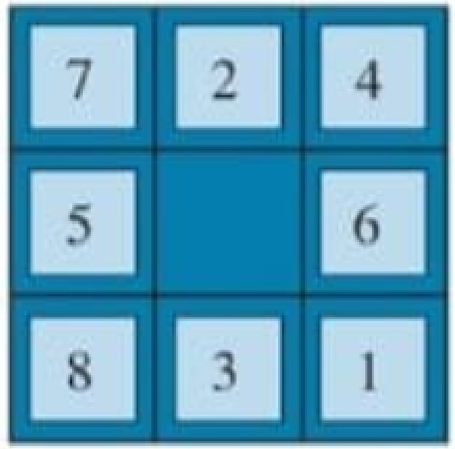
- Heuristic in AI
- H1 Manhattan Distance
- H2 Misplaced Tiles
- Implementation

INFORMED SEARCH

Informed search = Heuristic search
it is a technique designed to solve a problem quickly



START STATE



GOAL STATE

Terdapat puzzle berukuran 3x3 dengan 8 bilah dimana 1 bilah dikosongkan sehingga puzzle tersebut dapat digerakan ke atas, bawah, kanan dan kiri. Posisi awal yang tidak beraturan menjadi misi awal untuk kita membuat goal state dimana setiap bilah akan berubah menjadi urutan angka.

MANHATTAN DISTANCE

metode perhitungan jarak pada ruang jarak dengan menerapkan konsep selisih mutlak

$$Mdist = |x2 - x1| + |y2 - y1|$$

Implementasi heuristics manhatan distance

```python
import heapq
import math

# fungsi heuristik jarak manhattan
def manhattan_distance(state):
    goal = [1, 2, 3, 4, 5, 6, 7, 8, 0]
    distance = 0
    for i in range(9):
        if state[i] != 0:
            x = abs(i % 3 - goal.index(state[i]) % 3)
            y = abs(i // 3 - goal.index(state[i]) // 3)
            distance += x + y
    return distance

# fungsi untuk mengembalikan langkah-langkah yang ditempuh untuk mencapai solusi
def get_path(node):
    path = []
    while node:
        path.append(node[3])
        node = node[2]
    return path[::-1]

```

Implementasi heuristics manhatan distance

```
24  # fungsi untuk mencari solusi
25  def solve(initial_state):
26      heap = [(manhattan_distance(initial_state), 0, None, initial_state)]
27      visited = set()
28      while heap:
29          (h, g, parent, state) = heapq.heappop(heap)
30          if state == [1, 2, 3, 4, 5, 6, 7, 8, 0]:
31              return get_path((h, g, parent, state))
32          visited.add(tuple(state))
33          i = state.index(0)
34          if i not in [0, 1, 2]:  # move up
35              new_state = state[:]
36              new_state[i], new_state[i - 3] = new_state[i - 3], new_state[i]
37              if tuple(new_state) not in visited:
38                  heapq.heappush(heap, (g + 1 + manhattan_distance(new_state), g + 1, (h, g, parent, state), new_state))
39          if i not in [6, 7, 8]:  # move down
40              new_state = state[:]
41              new_state[i], new_state[i + 3] = new_state[i + 3], new_state[i]
42              if tuple(new_state) not in visited:
43                  heapq.heappush(heap, (g + 1 + manhattan_distance(new_state), g + 1, (h, g, parent, state), new_state))
```

Implementasi heuristics manhatan distance

```
        if i not in [0, 3, 6]:  # move left
            new_state = state[:]
            new_state[i], new_state[i - 1] = new_state[i - 1], new_state[i]
            if tuple(new_state) not in visited:
                heapq.heappush(heap, (g + 1 + manhattan_distance(new_state), g + 1, (h, g, parent, state), new_state))
        if i not in [2, 5, 8]:  # move right
            new_state = state[:]
            new_state[i], new_state[i + 1] = new_state[i + 1], new_state[i]
            if tuple(new_state) not in visited:
                heapq.heappush(heap, (g + 1 + manhattan_distance(new_state), g + 1, (h, g, parent, state), new_state))
    return None  # tidak ditemukan solusi
```

```
# contoh penggunaan
initial_state = [7, 2, 4, 5, 0, 6, 8, 3, 1]
path = solve(initial_state)
if path:
    print("Langkah-langkah yang ditempuh untuk mencapai solusi:")
    for i, state in enumerate(path):
        print(f"Langkah {i}:")
        print(state[0:3])
        print(state[3:6])
        print(state[6:9])
        print()
else:
    print("Tidak ditemukan solusi.")
```

```
Langkah 24:
[3, 1, 2]
[6, 4, 5]
[0, 7, 8]

Langkah 25:
[3, 1, 2]
[0, 4, 5]
[6, 7, 8]

Langkah 26:
[0, 1, 2]
[3, 4, 5]
[6, 7, 8]
```

Misplaced Tiles

Fungsi heuristik yang menghitung jumlah angla yang berada pada node yang salah

Implementasi Heuristic Misplaced Tiles

```python
import heapq

# fungsi heuristik banyaknya puzzle yang salah penempatan
def misplaced_tiles(state):
    goal = [0, 1, 2, 3, 4, 5, 6, 7, 8]
    misplaced = sum([1 for i in range(9) if state[i] != goal[i]])
    return misplaced

# fungsi untuk mengembalikan langkah-langkah yang ditempuh untuk mencapai solusi
def get_path(node):
    path = []
    while node:
        path.append(node[3])
        node = node[2]
    return path[::-1]
```

```python
# fungsi untuk mencari solusi
def solve(initial_state):
    heap = [(misplaced_tiles(initial_state), 0, None, initial_state)]
    visited = set()
    while heap:
        (h, g, parent, state) = heapq.heappop(heap)
        if state == [1, 2, 3, 4, 5, 6, 7, 8, 0]:
            return get_path((h, g, parent, state))
        visited.add(tuple(state))
        i = state.index(0)
        if i not in [0, 1, 2]:  # move up
            new_state = state[:]
            new_state[i], new_state[i - 3] = new_state[i - 3], new_state[i]
            if tuple(new_state) not in visited:
                heapq.heappush(heap, (g + 1 + misplaced_tiles(new_state), g + 1, (h, g, parent, state), new_state))
        if i not in [6, 7, 8]:  # move down
            new_state = state[:]
            new_state[i], new_state[i + 3] = new_state[i + 3], new_state[i]
            if tuple(new_state) not in visited:
                heapq.heappush(heap, (g + 1 + misplaced_tiles(new_state), g + 1, (h, g, parent, state), new_state))
        if i not in [0, 3, 6]:  # move left
            new_state = state[:]
            new_state[i], new_state[i - 1] = new_state[i - 1], new_state[i]
            if tuple(new_state) not in visited:
                heapq.heappush(heap, (g + 1 + misplaced_tiles(new_state), g + 1, (h, g, parent, state), new_state))
        if i not in [2, 5, 8]:  # move right
            new_state = state[:]
            new_state[i], new_state[i + 1] = new_state[i + 1], new_state[i]
            if tuple(new_state) not in visited:
                heapq.heappush(heap, (g + 1 + misplaced_tiles(new_state), g + 1, (h, g, parent, state), new_state))
    return None  # tidak ditemukan solusi
```

```
# contoh penggunaan
initial_state = [7, 2, 4, 5, 0, 6, 8, 3, 1]
path = solve(initial_state)
if path:
    print("Langkah-langkah yang ditempuh untuk mencapai solusi:")
    for i, state in enumerate(path):
        print(f"Langkah {i}:")
        print(state[0:3])
        print(state[3:6])
        print(state[6:9])
        print()
else:
    print("Tidak ditemukan solusi.")
```

```
Langkah 16:
[0, 2, 3]
[1, 4, 5]
[7, 8, 6]

Langkah 17:
[1, 2, 3]
[7, 8, 6]
[7, 8, 6]

Langkah 20:
[1, 2, 3]
[4, 5, 6]
[7, 8, 0]
```