Projekt 1
Prehľadávanie stavového priestoru

Laura Fulajtárová

Fakulta informatiky a informačných technológií STU

xfulajtarova@stuba.sk

ID: 120782

UI – Utorok 14:00

5.10.2023

Contents

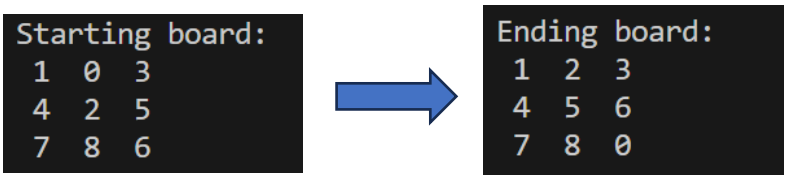
Riešený problém 3
Opis riešenia 3
A* algoritmus 3
Heuristika 1 - Wrong tiles 4
Heuristika 2 - Manhattan distance 4
Rotácia puzzle vzhľadom na polohu prázdneho políčka 5
Generovanie možných ťahov a pridávanie uzlov do haldy 5
Reprezentáciu údajov problému 6
Spôsob testovania a výsledky experimentov 6
Ľahká obtiažnosť 6
Stredná obtiažnosť 7
Ťažká obtiažnosť 8
Neriešiteľné príklady 9
Zhodnotenie riešenia a dosiahnutých výsledkov 10
Podľa času 10
Podľa krokov 10
Podľa vytvorených uzlov 11
Používateľská príručka na spustenie programu 11
Použitá literatúra 12

Riešený problém 2 úloha e)

Zadanie spočíva v riešení 8-hlavolamu na hracej doske o veľkosti MxM, pričom som testovala problémy s doskami o rozmeroch 3x3 a 4x4. Hlavolam pozostáva z osmi číselných políčok a jedného prázdneho miesta. Hráč môže presúvať políčka nahor, nadol, vľavo alebo vpravo, avšak iba v smere prázdneho miesta. Začiatočný a cieľový stav hlavolamu sú vopred definované, a úlohou je nájsť optimálnu postupnosť krokov, ktorá vedie z počiatočného stavu k cieľovému.

Na riešenie tohto problému je použitý A* algoritmus spolu s heuristickými funkciami. Tieto heuristiky poskytujú doplnkové informácie o vzdialenosti medzi aktuálnym stavom a cieľovým stavom. Konkrétne heuristiky zahŕňajú počet políčok, ktoré sa nenachádzajú na svojich správnych pozíciách, a súčet vzdialeností jednotlivých políčok od ich cieľových pozícií. Tieto heuristiky pomáhajú rozhodnúť, akým smerom sa vydať, aby sa čo najefektívnejšie dosiahol cieľový stav.

Konkrétny príklad:



V rámci tohto problému existuje konkrétna postupnosť krokov, ktorá vedie k riešeniu, napríklad kroky DOLE, DOPRAVA a znovu DOLE. Navrhovaná metóda algoritmu musí byť schopná identifikovať a zabezpečiť túto konkrétnu postupnosť krokov prostredníctvom výpočtu heuristík a porovnávania krokov.

Opis riešenia

A* algoritmus

A* je algoritmus na hľadanie optimálnej cesty v grafe. Využíva otvorený zoznam na uchovávanie nepreskúmaných uzlov a uzavretý zoznam na preskúmané uzly. Priorita uzlov sa určuje kombináciou ceny dosiahnutej cesty (g) a heuristického odhadu (h), f = g + h.

Moja implementácia:

Najskôr som si inicializovala a prípadne vynulovala všetky potrebné premenné. Na uchovávanie uzlov a ich funkcií som využívala otvorený a uzavretý zoznam (open a closed list).

Na začiatku som pridala počiatočnú konfiguráciu hracej plochy do prioritnej fronty (priority queue) pomocou štruktúry dát s názvom "heapq". Táto štruktúra "heapq" umožňuje efektívne spravovať prioritné rady (min-heap), kde prvok s najnižšou prioritou je na vrchole. Funkcia heappush mi umožnila pridávať uzly do tohto min-heapu podľa priority.

Potom som pomocou cyklu while postupne pridávala možné riešenia do haldy (heap) a monitorovala som počet vytvorených uzlov pre prípad, že by som chcela program ukončiť v prípade vysokého času.

Keď som pridávala deti do haldy, použila som funkciu heappush, ktorá umožňuje vložiť prvok do haldy tak, aby sa zachovala jej vlastnosť min-heapu.

Ak som objavila riešenie vo funkcii insert(node, board_end, open_list, n, choice), zaznamenala som počet vytvorených uzlov, vypísala som cestu k riešeniu pomocou funkcie print_solution_path(solution_node, choice) a ukončila som hlavnú funkciu.

```
def astar(board_start, board_end, n, choice):
    global closed_set, solution, num_of_moves_h1, num_of_moves_h2, num_of_nodes_h1, num_of_nodes_h2
    closed_set = set()
    solution = None

    start_node = Node(board_start)
    open_list = []
    node_count = 0
```

```
    heapq.heappush(open_list, (start_node.f, start_node))

    while open_list:
        _, current_node = heapq.heappop(open_list)
        node_count += 1

        if node_count >= 50000:
            print("Reached 50000 nodes, exiting...")
            if choice == 1:
                num_of_moves_h1 = 0
            elif choice == 2:
                num_of_moves_h2 = 0
            break

        insert(current_node, board_end, open_list, n, choice)

        if solution:
            if choice == 1:
                num_of_nodes_h1 = node_count
            elif choice == 2:
                num_of_nodes_h2 = node_count
            print_solution_path(solution, choice)
            break
```

Heuristika 1- Wrong tiles

Jednou z heuristík, ktorú som použila, je "počet zle umiestnených políčok." Táto heuristika spočíta, koľko políčok nie je na správnom mieste vzhľadom na cieľový stav.

```
Starting board:
 0  1  2
 3  4  5
 6  7  8
```

```
Ending board:
 1  0  2
 4  8  5
 3  7  6
```

h1=5

```
def heuristic_1_wrong_tiles(board, board_end):
    count = 0
    for i in range(len(board)):
        if board[i] != board_end[i] and board[i] != 0:
            count += 1
    return count
```

Heuristika 2- Manhattan distance

Druhou použitou heuristikou je "Manhattanova vzdialenosť." Táto metrika meria najkratšiu vzdialenosť medzi dvoma bodmi na mriežke podľa počtu krokov, ktoré sú potrebné vo vodorovnom a zvislom smere, aby sa dostali z jedného bodu do druhého.

```
# Function to calculate the Manhattan distance for a single tile
def manhattan_distance(tile, current_position, goal_position, n):
    if tile == 0:
        return 0
    tile -= 1
    current_row, current_col = current_position // n, current_position % n
    goal_row, goal_col = goal_position // n, goal_position % n
    return abs(current_row - goal_row) + abs(current_col - goal_col)

# Function to calculate the sum of Manhattan distances for all tiles
def heuristic_2_tiles_distance(board, goal, n):
    distance = 0
    for i in range(len(board)):
        tile_value = board[i]
        goal_position = goal.index(tile_value)
        distance += manhattan_distance(tile_value, i, goal_position, n)
    return distance
```

```
Starting board:
 5  ➡  ➡
 1  2  ⬇
 7  6  8
```

```
Ending board:
 8  0  1
 4  7  5
 3  2  6
```

distance for single tile 5 = 3

distance for every tile: $3_5 + 3_4 + 3_1 + 1_2 + 3_3 + 2_7 + 1_6 + 4_8 = 20$

Rotácia puzzle vzhľadom na polohu prázdneho políčka

Pre rotáciu hracej dosky som vytvorila štyri funkcie: "hore", "dole", "vpravo" a "vľavo". Každá z týchto funkcií má za úlohu vymeniť prázdne políčko, ktoré je označené 0, so svojím susedom. Tieto funkcie sú totožné okrem časti kódu, ktorá je vyznačená na obrázku. V tejto časti kontrolujeme hraničné podmienky, kde sa prázdne políčko nemôže posunúť, a tiež určujeme, s ktorým susedom sa má vymeniť, vzhľadom na smer, v ktorom chceme otáčať puzzle. Každá z týchto funkcií vytvára kópiu hracej plochy, aby sa zachoval pôvodný stav, a potom vykoná presun prázdneho políčka. Po vykonaní pohybu funkcia vráti nový stav hracej plochy.

```
def up(board, n):
    board_copy = copy.deepcopy(board)
    for i in range(len(board_copy)):
        if board_copy[i] == 0:
            if i - n >= 0:
                board_copy[i], board_copy[i - n] = board_copy[i - n], board_copy[i]
                return board_copy
    return board_copy


down    if i + n < len(board_copy):
            board_copy[i], board_copy[i + n] = board_copy[i + n], board_copy[i]

left    if i % n != 0:
            board_copy[i], board_copy[i - 1] = board_copy[i - 1], board_copy[i]

right   if (i + 1) % n != 0:
            board_copy[i], board_copy[i + 1] = board_copy[i + 1], board_copy[i]
```

Generovanie možných ťahov a pridávanie uzlov do haldy

Na vloženie detí do prioritnej fronty som vytvorila funkciu s názvom "insert". Najskôr kontrolujeme, či aktuálna hracia doska nie je naším konečným cieľom. Ak áno, uložíme ju ako riešenie a ukončíme vyhľadávanie. Potom kontrolujeme, či je aktuálna hracia doska už v uzavretých stavoch. Ak áno, preskočíme ju a nepokračujeme v jej ďalšom spracovaní. V opačnom prípade ju pridáme do množiny uzavretých stavov.

Ďalej funkcia generuje všetky možné rotácie hracej dosky, teda otáčanie doľava, doprava, nahor a nadol. Tieto vytvorené dosky skúmame tak, že ich porovnávame s aktuálnou doskou, pretože v prípade, že by posun bol mimo hracej dosky, stav by zostal nezmenený, a preto môžeme pokračovať len s platnými pohybmi.

Pre každé takto vytvorené dieťa vytvoríme uzol s príslušným rozložením dosky, operátorom, aktualizovanou hĺbkou, vypočítanou funkciou pre danú heuristiku a odkazom na jeho predchodcu, tzv. jeho rodiča.

Ďalej kontrolujeme, či takto vytvorené dieťa už nie je v open liste. Ak nie je, pridáme ho do priority queue vzhľadom na jeho f hodnotu pomocou funkcie "heappush".

```
def insert(node, board_end, open_list, n, choice):
    if is_same(node.board, board_end):
        global solution
        solution = copy.deepcopy(node)
        return None

    if tuple(node.board) in closed_set:
        return node

    closed_set.add(tuple(node.board))

    l_ch = left(node.board, n)
    r_ch = right(node.board, n)
    u_ch = up(node.board, n)
    d_ch = down(node.board, n)

    if not is_same(l_ch, node.board):
        child = Node(
            l_ch,
            "left",
            node.depth + 1,
            parent=node,
            f=node.depth + 1 + calculate_heuristic(choice, l_ch, board_end, n),
        )
        if child.board not in open_list:
            heapq.heappush(open_list, (child.f, child))
```

```
    ....... toto opakujeme pre r_ch, u_ch, d_ch len s iným operandom

    return node
```

Reprezentáciu údajov problému

Pre reprezentáciu stavu a nájdenú cestu som vytvorila triedu Node. V tejto triede som uchovávala rôzne typy údajov, ako je rozloženie hracej dosky, operátor, ktorým sme sa dostali k tomuto stavu (ľavý, pravý, hore, dole), hĺbku uzla (na výpočet funkcie "f"), samotnú funkciu "f" a odkaz na rodiča (na neskoršie vypísanie cesty).

```
class Node:
    def __init__( self, board=[], operand=None, depth=0, parent=None, f=None,):
        self.board = board
        self.operand = operand
        self.depth = depth
        self.f = f
        self.parent = parent
```

Spôsob testovania a výsledky experimentov

Určila som štyri rôzne scenáre na testovanie, pričom som brala do úvahy obtiažnosť a riešiteľnosť problému.

Ľahká obtiažnosť

Na začiatok som oba prístupy testovala na veľmi jednoduchých príkladoch s maximálne 5 krokmi, pri ktorých obe heuristiky rýchlo našli riešenie. V tomto prípade sa heuristika 2, založená na Manhattanských vzdialenostiach, v časovej náročnosti ukázala ako výhodnejšia, avšak rozdiel nebol tak výrazný ako pri iných skúmaných scenároch. Obidve heuristiky viedli k rovnakému počtu krokov a taktiež zväčša aj k rovnakému počtu vytvorených uzlov.

3x3 BOARD

```
Starting board:
 1  0  3
 4  2  5
 7  8  6
```

```
Ending board:
 1  2  3
 4  5  6
 7  8  0
```

```
Solving the puzzle with 1. heuristic-wrong tiles...

Solution Path:
Move: 1         down        [1, 2, 3, 4, 0, 5, 7, 8, 6]
Move: 2         right       [1, 2, 3, 4, 5, 0, 7, 8, 6]
Move: 3         down        [1, 2, 3, 4, 5, 6, 7, 8, 0]
```

```
Solving the puzzle with 2. heuristic-manhattan distance...

Solution Path:
Move: 1         down        [1, 2, 3, 4, 0, 5, 7, 8, 6]
Move: 2         right       [1, 2, 3, 4, 5, 0, 7, 8, 6]
Move: 3         down        [1, 2, 3, 4, 5, 6, 7, 8, 0]
```

```
Comparing the two heuristics:

Time taken for heuristic-wrong tiles: 3.078 ms
Time taken for heuristic-manhattan distance: 1.743 ms

Number of steps for heuristic-wrong tiles: 3
Number of steps for heuristic-manhattan distance: 3

Total number of nodes generated with heuristic-wrong tiles: 4
Total number of nodes generated heuristic-manhattan distance: 4

Heuristic-manhattan distance is 1.766 times faster than heuristic-wrong tiles
Heuristic-wrong tiles and heuristic-manhattan distance take the same number of steps
Heuristic-wrong tiles and heuristic-manhattan distance generate the same number of nodes
```

4x4 BOARD

```
Starting board:
 1  2  3  4
 5  6  7  8
 9 10  0 11
13 14 15 12
```

```
Ending board:
 1  2  3  4
 5  6  7  8
 9 10 11 12
13 14 15  0
```

```
Solving the puzzle with 1. heuristic-wrong tiles...

Solution Path:
Move: 1         right       [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 0, 13, 14, 15, 12]
Move: 2         down        [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0]
```

```
Solving the puzzle with 2. heuristic-manhattan distance...

Solution Path:
Move: 1         right       [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 0, 13, 14, 15, 12]
Move: 2         down        [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0]
```

```
Comparing the two heuristics:

Time taken for heuristic-wrong tiles: 2.109 ms
Time taken for heuristic-manhattan distance: 1.544 ms

Number of steps for heuristic-wrong tiles: 2
Number of steps for heuristic-manhattan distance: 2

Total number of nodes generated with heuristic-wrong tiles: 3
Total number of nodes generated heuristic-manhattan distance: 3

Heuristic-manhattan distance is 1.366 times faster than heuristic-wrong tiles
Heuristic-wrong tiles and heuristic-manhattan distance take the same number of steps
Heuristic-wrong tiles and heuristic-manhattan distance generate the same number of nodes
```

Stredná obtiažnosť

V prípade stredne náročných príkladov s približne dvadsiatimi krokmi excelovala heuristika 2 v rýchlosti a efektívnosti vzhľadom na počet vytvorených uzlov, pričom cesta riešenia zostávala totožná.

3x3 BOARD

```
Starting board:
 7  6  4
 5  3  2
 8  0  1
```

```
Ending board:
 4  0  5
 8  2  1
 3  7  6
```

```
Solving the puzzle with 1. heuristic-wrong tiles...

Solution Path:
Move: 1        up          [7, 6, 4, 5, 0, 2, 8, 3, 1]
Move: 2        left        [7, 6, 4, 0, 5, 2, 8, 3, 1]
Move: 3        up          [0, 6, 4, 7, 5, 2, 8, 3, 1]
Move: 4        right       [6, 0, 4, 7, 5, 2, 8, 3, 1]
Move: 5        right       [6, 4, 0, 7, 5, 2, 8, 3, 1]
Move: 6        down        [6, 4, 2, 7, 5, 0, 8, 3, 1]
Move: 7        left        [6, 4, 2, 7, 0, 5, 8, 3, 1]
Move: 8        left        [6, 4, 2, 0, 7, 5, 8, 3, 1]
Move: 9        up          [0, 4, 2, 6, 7, 5, 8, 3, 1]
Move: 10       right       [4, 0, 2, 6, 7, 5, 8, 3, 1]
Move: 11       down        [4, 7, 2, 6, 0, 5, 8, 3, 1]
Move: 12       left        [4, 7, 2, 0, 6, 5, 8, 3, 1]
Move: 13       down        [4, 7, 2, 8, 6, 5, 0, 3, 1]
Move: 14       right       [4, 7, 2, 8, 6, 5, 3, 0, 1]
Move: 15       up          [4, 7, 2, 8, 0, 5, 3, 6, 1]
Move: 16       up          [4, 0, 2, 8, 7, 5, 3, 6, 1]
Move: 17       right       [4, 2, 0, 8, 7, 5, 3, 6, 1]
Move: 18       down        [4, 2, 5, 8, 7, 0, 3, 6, 1]
Move: 19       down        [4, 2, 5, 8, 7, 1, 3, 6, 0]
Move: 20       left        [4, 2, 5, 8, 7, 1, 3, 0, 6]
Move: 21       up          [4, 2, 5, 8, 0, 1, 3, 7, 6]
Move: 22       up          [4, 0, 5, 8, 2, 1, 3, 7, 6]
```

```
Solving the puzzle with 2. heuristic-manhattan distance...

Solution Path:
Move: 1        up          [7, 6, 4, 5, 0, 2, 8, 3, 1]
Move: 2        left        [7, 6, 4, 0, 5, 2, 8, 3, 1]
Move: 3        up          [0, 6, 4, 7, 5, 2, 8, 3, 1]
Move: 4        right       [6, 0, 4, 7, 5, 2, 8, 3, 1]
Move: 5        right       [6, 4, 0, 7, 5, 2, 8, 3, 1]
Move: 6        down        [6, 4, 2, 7, 5, 0, 8, 3, 1]
Move: 7        left        [6, 4, 2, 7, 0, 5, 8, 3, 1]
Move: 8        left        [6, 4, 2, 0, 7, 5, 8, 3, 1]
Move: 9        up          [0, 4, 2, 6, 7, 5, 8, 3, 1]
Move: 10       right       [4, 0, 2, 6, 7, 5, 8, 3, 1]
Move: 11       down        [4, 7, 2, 6, 0, 5, 8, 3, 1]
Move: 12       left        [4, 7, 2, 0, 6, 5, 8, 3, 1]
Move: 13       down        [4, 7, 2, 8, 6, 5, 0, 3, 1]
Move: 14       right       [4, 7, 2, 8, 6, 5, 3, 0, 1]
Move: 15       up          [4, 7, 2, 8, 0, 5, 3, 6, 1]
Move: 16       up          [4, 0, 2, 8, 7, 5, 3, 6, 1]
Move: 17       right       [4, 2, 0, 8, 7, 5, 3, 6, 1]
Move: 18       down        [4, 2, 5, 8, 7, 0, 3, 6, 1]
Move: 19       down        [4, 2, 5, 8, 7, 1, 3, 6, 0]
Move: 20       left        [4, 2, 5, 8, 7, 1, 3, 0, 6]
Move: 21       up          [4, 2, 5, 8, 0, 1, 3, 7, 6]
Move: 22       up          [4, 0, 5, 8, 2, 1, 3, 7, 6]
```

```
Comparing the two heuristics:

Time taken for heuristic-wrong tiles: 1929.317 ms
Time taken for heuristic-manhattan distance: 43.839 ms

Number of steps for heuristic-wrong tiles: 22
Number of steps for heuristic-manhattan distance: 22

Total number of nodes generated with heuristic-wrong tiles: 12568
Total number of nodes generated heuristic-manhattan distance: 685

Heuristic-manhattan distance is 44.009 times faster than heuristic-wrong tiles
Heuristic-wrong tiles and heuristic-manhattan distance take the same number of steps
Heuristic-manhattan distance generates 18.347 times less nodes than heuristic-wrong tiles
```

4x4 BOARD

```
Starting board:
 6  2  4  0
 1  5  3  7
 9 14 12  8
10 13 11 15
```

```
Ending board:
 1  2  3  4
 5  6  7  8
 9 10 11 12
13 14 15  0
```

```
Solving the puzzle with 1. heuristic-wrong tiles...

Solution Path:
Move: 1         left        [6, 2, 0, 4, 1, 5, 3, 7, 9, 14, 12, 8, 10, 13, 11, 15]
Move: 2         left        [6, 0, 2, 4, 1, 5, 3, 7, 9, 14, 12, 8, 10, 13, 11, 15]
Move: 3         left        [0, 6, 2, 4, 1, 5, 3, 7, 9, 14, 12, 8, 10, 13, 11, 15]
Move: 4         down        [1, 6, 2, 4, 0, 5, 3, 7, 9, 14, 12, 8, 10, 13, 11, 15]
Move: 5         down        [1, 6, 2, 4, 9, 5, 3, 7, 0, 14, 12, 8, 10, 13, 11, 15]
Move: 6         down        [1, 6, 2, 4, 9, 5, 3, 7, 10, 14, 12, 8, 0, 13, 11, 15]
Move: 7         right       [1, 6, 2, 4, 9, 5, 3, 7, 10, 14, 12, 8, 13, 0, 11, 15]
Move: 8         up          [1, 6, 2, 4, 9, 5, 3, 7, 10, 0, 12, 8, 13, 14, 11, 15]
Move: 9         left        [1, 6, 2, 4, 9, 5, 3, 7, 0, 10, 12, 8, 13, 14, 11, 15]
Move: 10        up          [1, 6, 2, 4, 0, 5, 3, 7, 9, 10, 12, 8, 13, 14, 11, 15]
Move: 11        right       [1, 6, 2, 4, 5, 0, 3, 7, 9, 10, 12, 8, 13, 14, 11, 15]
Move: 12        up          [1, 0, 2, 4, 5, 6, 3, 7, 9, 10, 12, 8, 13, 14, 11, 15]
Move: 13        right       [1, 2, 0, 4, 5, 6, 3, 7, 9, 10, 12, 8, 13, 14, 11, 15]
Move: 14        down        [1, 2, 3, 4, 5, 6, 0, 7, 9, 10, 12, 8, 13, 14, 11, 15]
Move: 15        right       [1, 2, 3, 4, 5, 6, 7, 0, 9, 10, 12, 8, 13, 14, 11, 15]
Move: 16        down        [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 0, 13, 14, 11, 15]
Move: 17        left        [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 0, 12, 13, 14, 11, 15]
Move: 18        down        [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 0, 15]
Move: 19        right       [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0]
```

```
Solving the puzzle with 2. heuristic-manhattan distance...

Solution Path:
Move: 1         left        [6, 2, 0, 4, 1, 5, 3, 7, 9, 14, 12, 8, 10, 13, 11, 15]
Move: 2         left        [6, 0, 2, 4, 1, 5, 3, 7, 9, 14, 12, 8, 10, 13, 11, 15]
Move: 3         left        [0, 6, 2, 4, 1, 5, 3, 7, 9, 14, 12, 8, 10, 13, 11, 15]
Move: 4         down        [1, 6, 2, 4, 0, 5, 3, 7, 9, 14, 12, 8, 10, 13, 11, 15]
Move: 5         down        [1, 6, 2, 4, 9, 5, 3, 7, 0, 14, 12, 8, 10, 13, 11, 15]
Move: 6         down        [1, 6, 2, 4, 9, 5, 3, 7, 10, 14, 12, 8, 0, 13, 11, 15]
Move: 7         right       [1, 6, 2, 4, 9, 5, 3, 7, 10, 14, 12, 8, 13, 0, 11, 15]
Move: 8         up          [1, 6, 2, 4, 9, 5, 3, 7, 10, 0, 12, 8, 13, 14, 11, 15]
Move: 9         left        [1, 6, 2, 4, 9, 5, 3, 7, 0, 10, 12, 8, 13, 14, 11, 15]
Move: 10        up          [1, 6, 2, 4, 0, 5, 3, 7, 9, 10, 12, 8, 13, 14, 11, 15]
Move: 11        right       [1, 6, 2, 4, 5, 0, 3, 7, 9, 10, 12, 8, 13, 14, 11, 15]
Move: 12        up          [1, 0, 2, 4, 5, 6, 3, 7, 9, 10, 12, 8, 13, 14, 11, 15]
Move: 13        right       [1, 2, 0, 4, 5, 6, 3, 7, 9, 10, 12, 8, 13, 14, 11, 15]
Move: 14        down        [1, 2, 3, 4, 5, 6, 0, 7, 9, 10, 12, 8, 13, 14, 11, 15]
Move: 15        right       [1, 2, 3, 4, 5, 6, 7, 0, 9, 10, 12, 8, 13, 14, 11, 15]
Move: 16        down        [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 0, 13, 14, 11, 15]
Move: 17        left        [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 0, 12, 13, 14, 11, 15]
Move: 18        down        [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 0, 15]
Move: 19        right       [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0]
```

```
Comparing the two heuristics:

Time taken for heuristic-wrong tiles: 98.688 ms
Time taken for heuristic-manhattan distance: 20.657 ms

Number of steps for heuristic-wrong tiles: 19
Number of steps for heuristic-manhattan distance: 19

Total number of nodes generated with heuristic-wrong tiles: 1286
Total number of nodes generated heuristic-manhattan distance: 189

Heuristic-manhattan distance is 4.778 times faster than heuristic-wrong tiles
Heuristic-wrong tiles and heuristic-manhattan distance take the same number of steps
Heuristic-manhattan distance generates 6.804 times less nodes than heuristic-wrong tiles
```

Ťažká obtiažnosť

Náročné príklady s približne tridsiatimi krokmi predstavovali výzvu, pri ktorej prvá heuristika, ktorá počet vytvorených uzlov prekročila, musela byť prerušená. Naopak, heuristika 2 zvládla tento problém v akceptovateľnom čase s primeraným počtom vytvorených uzlov.

3x3 BOARD

```
Starting board:
 5  6  7
 4  0  8
 3  2  1
```

```
Ending board:
 1  2  3
 8  0  4
 7  6  5
```

```
Solving the puzzle with 1. heuristic-wrong tiles...

Reached 50000 nodes, exiting...
```

```
Solving the puzzle with 2. heuristic-manhattan distance...

Solution Path:
Move: 1         left        [5, 6, 7, 0, 4, 8, 3, 2, 1]
Move: 2         down        [5, 6, 7, 3, 4, 8, 0, 2, 1]
Move: 3         right       [5, 6, 7, 3, 4, 8, 2, 0, 1]
Move: 4         right       [5, 6, 7, 3, 4, 8, 2, 1, 0]
Move: 5         up          [5, 6, 7, 3, 4, 0, 2, 1, 8]
Move: 6         up          [5, 6, 0, 3, 4, 7, 2, 1, 8]
Move: 7         left        [5, 0, 6, 3, 4, 7, 2, 1, 8]
Move: 8         left        [0, 5, 6, 3, 4, 7, 2, 1, 8]
Move: 9         down        [3, 5, 6, 0, 4, 7, 2, 1, 8]
Move: 10        down        [3, 5, 6, 2, 4, 7, 0, 1, 8]
Move: 11        right       [3, 5, 6, 2, 4, 7, 1, 0, 8]
Move: 12        right       [3, 5, 6, 2, 4, 7, 1, 8, 0]
Move: 13        up          [3, 5, 6, 2, 4, 0, 1, 8, 7]
Move: 14        up          [3, 5, 0, 2, 4, 6, 1, 8, 7]
Move: 15        left        [3, 0, 5, 2, 4, 6, 1, 8, 7]
Move: 16        left        [0, 3, 5, 2, 4, 6, 1, 8, 7]
Move: 17        down        [2, 3, 5, 0, 4, 6, 1, 8, 7]
Move: 18        down        [2, 3, 5, 1, 4, 6, 0, 8, 7]
Move: 19        right       [2, 3, 5, 1, 4, 6, 8, 0, 7]
Move: 20        right       [2, 3, 5, 1, 4, 6, 8, 7, 0]
Move: 21        up          [2, 3, 5, 1, 4, 0, 8, 7, 6]
Move: 22        up          [2, 3, 0, 1, 4, 5, 8, 7, 6]
Move: 23        left        [2, 0, 3, 1, 4, 5, 8, 7, 6]
Move: 24        left        [0, 2, 3, 1, 4, 5, 8, 7, 6]
Move: 25        down        [1, 2, 3, 0, 4, 5, 8, 7, 6]
Move: 26        down        [1, 2, 3, 8, 4, 5, 0, 7, 6]
Move: 27        right       [1, 2, 3, 8, 4, 5, 7, 0, 6]
Move: 28        right       [1, 2, 3, 8, 4, 5, 7, 6, 0]
Move: 29        up          [1, 2, 3, 8, 4, 0, 7, 6, 5]
Move: 30        left        [1, 2, 3, 8, 0, 4, 7, 6, 5]
```

```
Comparing the two heuristics:

Heuristic-wrong tiles was not able to find a solution

Time taken for heuristic-manhattan distance: 78.17 ms
Number of steps for heuristic-manhattan distance: 30
Total number of nodes generated heuristic-manhattan distance: 986
```

4x4 BOARD

```
Starting board:
 6  5  4  3
 2  1  8  7
 9 10 15 11
13 14 12  0
```

```
Ending board:
 1  2  3  4
 5  6  7  8
 9 10 11 12
13 14 15  0
```

```
Solving the puzzle with 1. heuristic-wrong tiles...

Reached 50000 nodes, exiting...
```

```
Solving the puzzle with 2. heuristic-manhattan distance...

Solution Path:
Move: 1         left        [6, 5, 4, 3, 2, 1, 8, 7, 9, 10, 15, 11, 13, 14, 0, 12]
Move: 2         up          [6, 5, 4, 3, 2, 1, 8, 7, 9, 10, 0, 11, 13, 14, 15, 12]
Move: 3         right       [6, 5, 4, 3, 2, 1, 8, 7, 9, 10, 11, 0, 13, 14, 15, 12]
Move: 4         up          [6, 5, 4, 3, 2, 1, 8, 0, 9, 10, 11, 7, 13, 14, 15, 12]
Move: 5         left        [6, 5, 4, 3, 2, 1, 0, 8, 9, 10, 11, 7, 13, 14, 15, 12]
Move: 6         left        [6, 5, 4, 3, 2, 0, 1, 8, 9, 10, 11, 7, 13, 14, 15, 12]
Move: 7         left        [6, 5, 4, 3, 0, 2, 1, 8, 9, 10, 11, 7, 13, 14, 15, 12]
Move: 8         up          [0, 5, 4, 3, 6, 2, 1, 8, 9, 10, 11, 7, 13, 14, 15, 12]
Move: 9         right       [5, 0, 4, 3, 6, 2, 1, 8, 9, 10, 11, 7, 13, 14, 15, 12]
Move: 10        down        [5, 2, 4, 3, 6, 0, 1, 8, 9, 10, 11, 7, 13, 14, 15, 12]
Move: 11        right       [5, 2, 4, 3, 6, 1, 0, 8, 9, 10, 11, 7, 13, 14, 15, 12]
Move: 12        up          [5, 2, 0, 3, 6, 1, 4, 8, 9, 10, 11, 7, 13, 14, 15, 12]
Move: 13        left        [5, 0, 2, 3, 6, 1, 4, 8, 9, 10, 11, 7, 13, 14, 15, 12]
Move: 14        down        [5, 1, 2, 3, 6, 0, 4, 8, 9, 10, 11, 7, 13, 14, 15, 12]
Move: 15        down        [5, 1, 2, 3, 6, 10, 4, 8, 9, 0, 11, 7, 13, 14, 15, 12]
Move: 16        right       [5, 1, 2, 3, 6, 10, 4, 8, 9, 11, 0, 7, 13, 14, 15, 12]
Move: 17        right       [5, 1, 2, 3, 6, 10, 4, 8, 9, 11, 7, 0, 13, 14, 15, 12]
Move: 18        up          [5, 1, 2, 3, 6, 10, 4, 0, 9, 11, 7, 8, 13, 14, 15, 12]
Move: 19        left        [5, 1, 2, 3, 6, 10, 0, 4, 9, 11, 7, 8, 13, 14, 15, 12]
Move: 20        down        [5, 1, 2, 3, 6, 10, 7, 4, 9, 11, 0, 8, 13, 14, 15, 12]
Move: 21        left        [5, 1, 2, 3, 6, 10, 7, 4, 9, 0, 11, 8, 13, 14, 15, 12]
Move: 22        up          [5, 1, 2, 3, 6, 0, 7, 4, 9, 10, 11, 8, 13, 14, 15, 12]
Move: 23        left        [5, 1, 2, 3, 0, 6, 7, 4, 9, 10, 11, 8, 13, 14, 15, 12]
Move: 24        up          [0, 1, 2, 3, 5, 6, 7, 4, 9, 10, 11, 8, 13, 14, 15, 12]
Move: 25        right       [1, 0, 2, 3, 5, 6, 7, 4, 9, 10, 11, 8, 13, 14, 15, 12]
Move: 26        right       [1, 2, 0, 3, 5, 6, 7, 4, 9, 10, 11, 8, 13, 14, 15, 12]
Move: 27        right       [1, 2, 3, 0, 5, 6, 7, 4, 9, 10, 11, 8, 13, 14, 15, 12]
Move: 28        down        [1, 2, 3, 4, 5, 6, 7, 0, 9, 10, 11, 8, 13, 14, 15, 12]
Move: 29        down        [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 0, 13, 14, 15, 12]
Move: 30        down        [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0]
```

```
Comparing the two heuristics:

Heuristic-wrong tiles was not able to find a solution

Time taken for heuristic-manhattan distance: 25024.133 ms
Number of steps for heuristic-manhattan distance: 30
Total number of nodes generated heuristic-manhattan distance: 36194
```

Neriešiteľné príklady

Posledným skúmaným scenárom boli príklady, ktoré obidve heuristiky nedokázali vyriešiť, označované ako neriešiteľné. V tomto prípade sme porovnali rýchlosť, s akou oba prístupy zistili neriešiteľnosť problému, pričom heuristika 2 opäť ukázala svoju spoľahlivosť.

3x3 BOARD

```
Starting board:
 3  7  0
 6  4  8
 1  2  5
```

```
Ending board:
 3  6  2
 7  5  4
 0  1  8
```

```
Solving the puzzle with 1. heuristic-wrong tiles...

Reached 50000 nodes, exiting...
```

```
Solving the puzzle with 2. heuristic-manhattan distance...

Reached 50000 nodes, exiting...
```

```
No solution found in either case

Time taken for heuristic-wrong tiles: 25359.715 ms
Time taken for heuristic-manhattan distance: 19473.613 ms
```

4x4 BOARD

```
Starting board:
 1  2  3  4
 5  6  7  8
 9 10 11 12
13 14 15  0
```

```
Ending board:
 1  2  3  4
 5  6  7  8
 9 11 10 12
13 14 15  0
```

```
Solving the puzzle with 1. heuristic-wrong tiles...

Reached 50000 nodes, exiting...
```

```
Solving the puzzle with 2. heuristic-manhattan distance...

Reached 50000 nodes, exiting...
```

```
No solution found in either case

Time taken for heuristic-wrong tiles: 51575.955 ms
Time taken for heuristic-manhattan distance: 40536.453 ms
```

Zhodnotenie riešenia a dosiahnutých výsledkov

V kontexte mojho riešenia tejto úlohy môžem konštatovať, že A* algoritmus sa ukázal ako výnimočne účinný a rýchly v porovnaní s prístupom nahodilého výberu krokov. Jeho schopnosť využívať heuristiky pre rýchle rozhodovanie o výbere uzlov v procese hľadania riešenia je pozoruhodná. Avšak rozhodnutie o vhodnej heuristike zásadným spôsobom ovplyvnilo dosiahnuté výsledky.

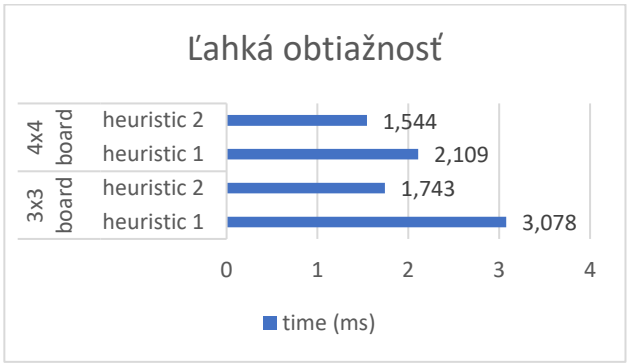
V tomto kontexte vynikla heuristika 2, založená na Manhattanských vzdialenostiach, ako veľmi rýchla a efektívna voľba pri hľadaní riešenia hracej dosky. Táto heuristika často generovala menší počet uzlov v stromovom vyhľadávacom priestore a zásadne prispela k rýchlejšiemu približovaniu sa k riešeniu. Zaujímavé je, že postupnosti krokov na dosiahnutie riešenia ostali totožné, ale heuristika 2 sa ukázala výrazne efektívnejšou.

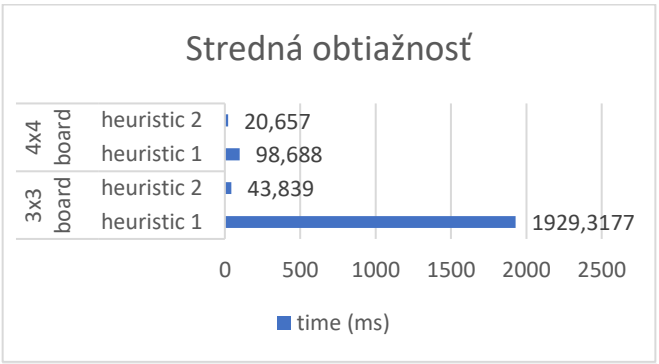
Obmedzenie na maximálny počet vytvorených uzlov (50000) viedlo k tomu, že heuristika 1 (počet nesprávne umiestnených políčok) v niektorých prípadoch neobjavila správnu cestu k riešeniu. Naopak, heuristika 2 bola schopná nájsť riešenie, hoci to trvalo trochu dlhšie, čím potvrdila svoju spoľahlivosť pri hľadaní optimálneho riešenia.

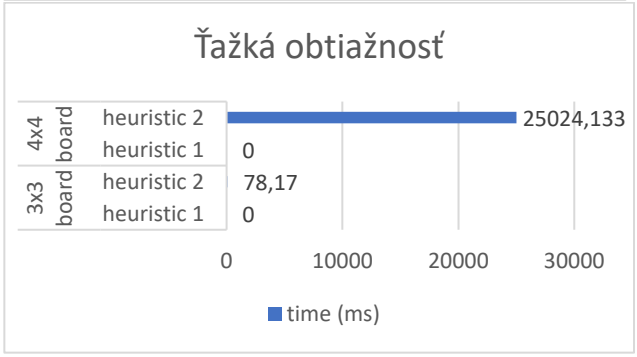
Na základe týchto pozorovaní by som bez váhania odporučila použitie heuristiky 2 (Manhattanská vzdialenosť) v tejto úlohe, keďže sa ukázala ako rýchlejšia, efektívnejšia a spoľahlivejšia vo voľbe heuristiky pre A* algoritmus pri riešení hracej dosky.

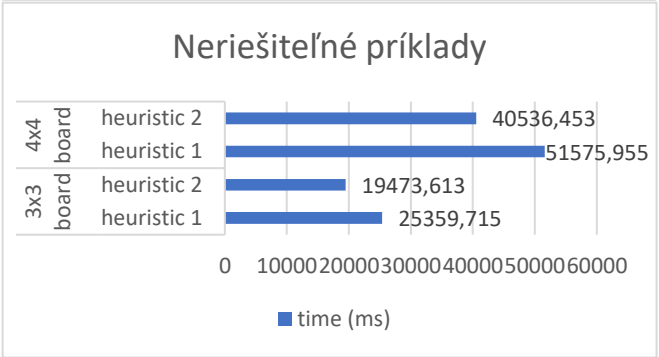
Nasledujúce grafy boli zostavené podľa získaných informácií v časti Spôsob testovania a výsledky experimentov.

Podľa času

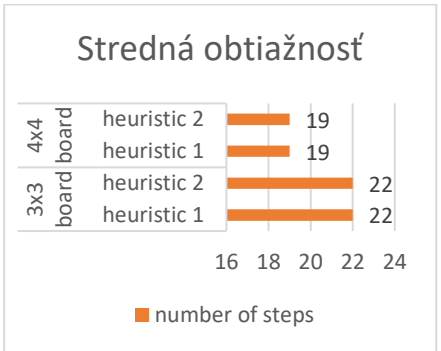


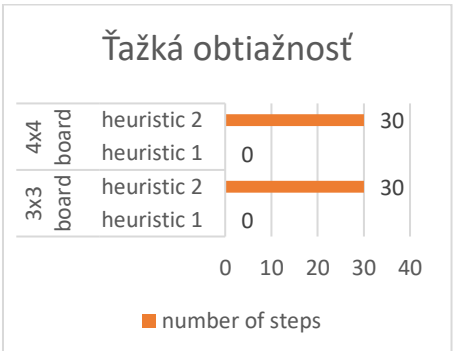




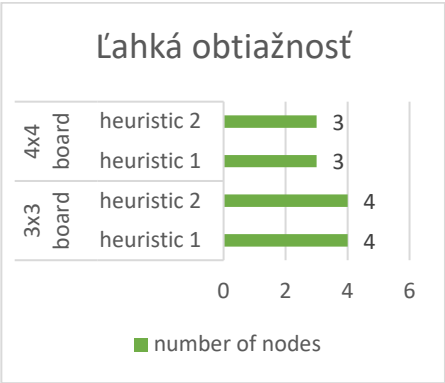


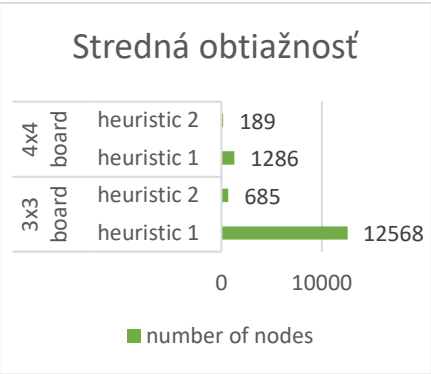
Podľa krokov

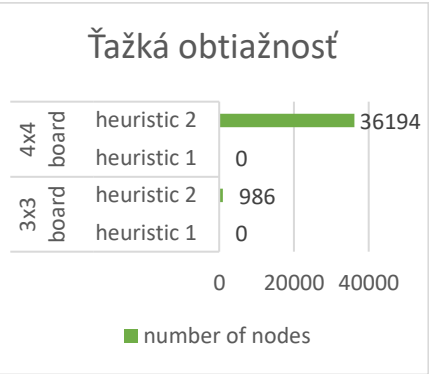




Podľa vytvorených uzlov







Používateľská príručka na spustenie programu

Moja práca sa skladá z dvoch súborov: main.py a game.py. V súbore game.py som naprogramovala logiku projektu a obsahuje rôzne metódy a triedy pre vytvorenie uzlov, algoritmus A*, vloženie detí, výpis cesty a ďalšie. Všetky podstatné informácie som taktiež uviedla v dokumentácii v časti s názvom Opis riešenia.

Súbor main.py slúži na spustenie celého programu. Navrhla som jednoduché rozhranie pre interakciu s programom. Po spustení programu vás program privíta a ponúkne vám tri základné možnosti: ukončiť hru, pokračovať s vami navrhnutým rozložením počiatočnej a cieľovej hracej dosky alebo automaticky vygeneruje náhodné rozloženia. Taktiež od vás program očakáva, aby ste zadali počet stĺpcov a riadkov hracej dosky, pričom odporúčam veľkosť 3x3, ale program bol testovaný aj na hracích doskách o veľkosti 4x4. Následne program zobrazí hracie dosky, s ktorými bude pracovať, a začne ich riešiť podľa heuristiky 1 alebo heuristiky 2. Po nájdení riešenia program zobrazí cestu a počet krokov. Taktiež poskytne porovnanie týchto heuristík vzhľadom na čas, počet krokov k správnemu riešeniu a počet uzlov, ktoré boli potrebné na vytvorenie riešenia. Po týchto štatistikách si môžete hru zahrať znova alebo ju ukončiť stlačením klávesy "q".

Použitá literatúra

1. https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/a-star-algorithm
2. https://www.geeksforgeeks.org/heap-queue-or-heapq-in-python/