Raport 2 - STRIPS - Szymon Frączek, Piotr Gąsiorek

STRIPS (Stanford Research Institute Problem Solver) to formalizm używany do reprezentowania problemów planowania w sztucznej inteligencji. STRIPS składa się z zestawu operatorów, które opisują, jak akcje zmieniają stan świata, oraz stanu początkowego i celowego, które definiują problem planowania. Każdy operator STRIPS ma listę preconditions (warunków wstępnych), które muszą być spełnione, aby operator mógł być zastosowany, oraz listę effects (efektów), które opisują, jak operator zmienia stan świata.

Blocksworld to klasyczny problem planowania w sztucznej inteligencji, który polega na przemieszczaniu bloków na stole, aby osiągnąć określony stan końcowy. Bloki mogą być układane jeden na drugim, ale tylko jeden blok może być przesuwany na raz. Problem Blocksworld jest często reprezentowany za pomocą formalizmu STRIPS, gdzie operatory opisują akcje przesuwania bloków, a stan początkowy i końcowy opisują układ bloków na początku i na końcu.

Funkcja pomocnicza do mierzenia czasu wykonywania innych funkcji.

In [ ]:
```python
import time

def timeit(func, func_name: str, *args, **kwargs):
    start = time.time()
    result = func(*args, **kwargs)
    end = time.time()
    print("=" * 160)
    print(f"Time taken by {func_name}: {end - start} seconds")
    return end - start, result
```

Stworzyliśmy dwie dziedziny problemów: *blocks1dom* oraz *blocks2dom*, reprezentujące róne konfiguracje świata bloków.

Zdefiniowane zostały 3 problemy planowania: *blocks1, blocks2, blocks3* - na kadym z nich jest wykonywana metoda search() z algorytmem A* do przeszukiwania grafu.

In [ ]:
```python
from stripsProblem import *
from searchMPP import *
from stripsForwardPlanner import *
from searchGeneric import Searcher

blocks1dom = create_blocks_world({'a', 'b', 'c', 'd'})
blocks2dom = create_blocks_world({'a', 'b', 'c', 'd', 'e'})
```

```
blocks1 = Planning_problem(blocks1dom,
    {
        on('a'):'table', clear('a'):False,
        on('b'):'c',  clear('b'):True,
        on('c'):'table', clear('c'):False,
        on('d'):'a', clear('d'): True,
    },
    {
        on('d'):'c', on('c'):'b', on('b'):'a'
    }
)

blocks2 = Planning_problem(blocks1dom,
    {
        clear('a'):True, on('a'):'b',
        clear('b'):False, on('b'):'c',
        clear('c'):False, on('c'):'d',
        clear('d'):False, on('d'):'table'
    },
    {
        on('d'):'c',on('c'):'b',on('b'):'a'
    }
)

blocks3 = Planning_problem(blocks2dom,
    {
        clear('a'):True, on('a'):'b',
        clear('b'):False, on('b'):'c',
        clear('c'):False, on('c'):'d',
        clear('d'):False, on('d'):'e',
        clear('e'):False, on('e'):'table'
    },
    {
        on('d'):'c',on('c'):'b',on('b'):'a', on('e'):'d'
    }
)

time1, result1 = timeit(lambda: AStarSearcher(Forward_STRIPS(blocks1)).se
time2, result2 = timeit(lambda: AStarSearcher(Forward_STRIPS(blocks2)).se
time3, result3 = timeit(lambda: AStarSearcher(Forward_STRIPS(blocks3)).se

print('=' * 160)
print(result1)
print('=' * 160)
print(result2)
print('=' * 160)
print(result3)
```

Time taken by AStarSearcher: 0.05001497268676758 seconds {'a_is_on': 'table', 'clear_a': False, 'b_is_on': 'c', 'clear_b': True, 'c_is_on': 'table', 'clear_c': False, 'd_is_on': 'a', 'clear_d': True} --move_d_from_a_to_table--> {'a_is_on': 'table', 'clear_a': True, 'b_is_on': 'c', 'clear_b': True, 'c_is_on': 'table', 'clear_c': False,

'd_is_on': 'table', 'clear_d': True} --move_b_from_c_to_a--> {'a_is_on': 'table', 'clear_a': False, 'b_is_on': 'a', 'clear_b': True, 'c_is_on': 'table', 'clear_c': True, 'd_is_on': 'table', 'clear_d': True} --move_c_from_table_to_b--> {'a_is_on': 'table', 'clear_a': False, 'b_is_on': 'a', 'clear_b': False, 'c_is_on': 'b', 'clear_c': True, 'd_is_on': 'table', 'clear_d': True, 'clear_table': True} --move_d_from_table_to_c--> {'a_is_on': 'table', 'clear_a': False, 'b_is_on': 'a', 'clear_b': False, 'c_is_on': 'b', 'clear_c': False, 'd_is_on': 'c', 'clear_d': True, 'clear_table': True}

{'clear_a': True, 'a_is_on': 'b', 'clear_b': False, 'b_is_on': 'c', 'clear_c': False, 'c_is_on': 'd', 'clear_d': False, 'd_is_on': 'table'} --move_a_from_b_to_table--> {'clear_a': True, 'a_is_on': 'table', 'clear_b': True, 'b_is_on': 'c', 'clear_c': False, 'c_is_on': 'd', 'clear_d': False, 'd_is_on': 'table'} --move_b_from_c_to_a--> {'clear_a': False, 'a_is_on': 'table', 'clear_b': True, 'b_is_on': 'a', 'clear_c': True, 'c_is_on': 'd', 'clear_d': False, 'd_is_on': 'table'} --move_c_from_d_to_b--> {'clear_a': False, 'a_is_on': 'table', 'clear_b': False, 'b_is_on': 'a', 'clear_c': True, 'c_is_on': 'b', 'clear_d': True, 'd_is_on': 'table'} --move_d_from_table_to_c--> {'clear_a': False, 'a_is_on': 'table', 'clear_b': False, 'b_is_on': 'a', 'clear_c': False, 'c_is_on': 'b', 'clear_d': True, 'd_is_on': 'c', 'clear_table': True}

{'clear_a': True, 'a_is_on': 'b', 'clear_b': False, 'b_is_on': 'c', 'clear_c': False, 'c_is_on': 'd', 'clear_d': False, 'd_is_on': 'e', 'clear_e': False, 'e_is_on': 'table'} --move_a_from_b_to_table--> {'clear_a': True, 'a_is_on': 'table', 'clear_b': True, 'b_is_on': 'c', 'clear_c': False, 'c_is_on': 'd', 'clear_d': False, 'd_is_on': 'e', 'clear_e': False, 'e_is_on': 'table'} --move_b_from_c_to_a--> {'clear_a': False, 'a_is_on': 'table', 'clear_b': True, 'b_is_on': 'a', 'clear_c': True, 'c_is_on': 'd', 'clear_d': False, 'd_is_on': 'e', 'clear_e': False, 'e_is_on': 'table'} --move_c_from_d_to_b--> {'clear_a': False, 'a_is_on': 'table', 'clear_b': False, 'b_is_on': 'a', 'clear_c': True, 'c_is_on': 'b', 'clear_d': True, 'd_is_on': 'e', 'clear_e': False, 'e_is_on': 'table'} --move_d_from_e_to_c--> {'clear_a': False, 'a_is_on': 'table', 'clear_b': False, 'b_is_on': 'a', 'clear_c': False, 'c_is_on': 'b', 'clear_d': True, 'd_is_on': 'c', 'clear_e': True, 'e_is_on': 'table'} --move_e_from_table_to_d--> {'clear_a': False, 'a_is_on': 'table', 'clear_b': False, 'b_is_on': 'a', 'clear_c': False, 'c_is_on': 'b', 'clear_d': False, 'd_is_on': 'c', 'clear_e': True, 'e_is_on': 'd', 'clear_table': True}

Stworzyliśmy klasę BlocksWorldNaiveHeuristic, która reprezentuje heurystykę dla problemu Blocksworld.

Bierze ona pod uwagę dwa aspekty:

1. Kolumny - dla każdego bloku, heurystyka oblicza, ile bloków jest nałożonych na nim. Ta informacja jest używana do szacowania, ile ruchów może być potrzebnych, aby przesunąć blok na właściwe miejsce.
2. Fundamenty - dla każdego bloku, heurystyka oblicza sekwencję bloków od

danego bloku do stołu. Ta informacja jest używana do szacowania, ile ruchów może być potrzebnych, aby przesunąć blok na właściwe miejsce, biorąc pod uwagę inne bloki, które mogą być na drodze.

Podczas obliczania wartości heurystyki dla danego stanu, heurystyka porównuje aktualne "kolumny" i "fundamenty" z oczekiwanymi "kolumnami" i "fundamentami". Jeśli "kolumna" lub "fundament" danego bloku nie pasuje do oczekiwanej "kolumny" lub "fundamentu", wartość heurystyki jest zwiększana.

Ta heurystyka jest "naiwna", ponieważ zakłada, że każda niezgodność w "kolumnach" lub "fundamentach" wymaga co najmniej jednego ruchu do naprawienia. Niektóre niezgodności mogą wymagać więcej ruchów do naprawienia, a niektóre ruchy mogą naprawić więcej niż jedną niezgodność na raz. Mimo to, ta heurystyka może być użyteczna do przyspieszenia procesu wyszukiwania, szczególnie w dużych problemach Blocksworld, gdzie przeszukiwanie wszystkich możliwych stanów byłoby zbyt czasochłonne.

In [ ]:
```python
from copy import deepcopy
from typing import Dict, List
from stripsProblem import Planning_problem

class BlocksWorldNaiveHeuristic():

    def __init__(self, problem: Planning_problem) -> None:
        self.expected_columns = self._calculate_columns(problem.goal)
        self.expected_fundaments = self._calculate_fundaments(problem.goa


    def _calculate_columns(self, goal) -> Dict[str, int]:
        unique_blocks = set()
        for k, v in goal.items():
            if v != 'table':
                unique_blocks.add(v)
            if k[0] != 'table':
                unique_blocks.add(k[0])

        goal = {k[0]: v for k, v in goal.items()}

        result = dict()
        for block in unique_blocks:
            curr = block
            deepness = 0

            while curr in goal and goal[curr] != 'table':
                curr = goal[curr]
                deepness += 1

            result[block] = deepness
```

```
        return result


    def _calculate_fundaments(self, goal) -> Dict[str, List[str]]:
        unique_blocks = set()
        for k, v in goal.items():
            if v != 'table':
                unique_blocks.add(v)
            if k[0] != 'table':
                unique_blocks.add(k[0])

        goal = {k[0]: v for k, v in goal.items()}

        result = dict()
        for block in unique_blocks:
            curr = block
            temp = list()

            while curr in goal and goal[curr] != 'table':
                curr = goal[curr]
                temp.append(curr)

            result[block] = temp

        return result

    def __call__(self, state, *args) -> int:

        temp = {k : v for k, v in state.items() if not k.startswith('clea
        state_columns = self._calculate_columns(temp)
        state_fundaments = self._calculate_fundaments(temp)

        result = 0
        checked = list()

        for element, expectedColumn in self.expected_columns.items():
            if element not in state_columns:
                print(state)
                print(state_columns)


            if expectedColumn != state_columns[element]:
                result += 1
                checked.append(element)

        for element, expectedFundaments in self.expected_fundaments.items
            if element in checked:
                continue

            for i in range(len(expectedFundaments)):
                if expectedFundaments[i] != state_fundaments[element][i]:
                    result += 2
```

```
        return result


heur1 = BlocksWorldNaiveHeuristic(blocks1)
heur2 = BlocksWorldNaiveHeuristic(blocks2)
heur3 = BlocksWorldNaiveHeuristic(blocks3)

time1_h, result1_h = timeit(lambda: AStarSearcher(Forward_STRIPS(blocks1,
time2_h, result2_h = timeit(lambda: AStarSearcher(Forward_STRIPS(blocks2,
time3_h, result3_h = timeit(lambda: AStarSearcher(Forward_STRIPS(blocks3,

print('=' * 160)
print(result1_h)
print('=' * 160)
print(result2_h)
print('=' * 160)
print(result3_h)
```

{'a_is_on': 'table', 'clear_a': False, 'b_is_on': 'c', 'clear_b': True, 'c_is_on': 'table', 'clear_c': False, 'd_is_on': 'a', 'clear_d': True} --move_b_from_c_to_table--> {'a_is_on': 'table', 'clear_a': False, 'b_is_on': 'table', 'clear_b': True, 'c_is_on': 'table', 'clear_c': True, 'd_is_on': 'a', 'clear_d': True} --move_d_from_a_to_table--> {'a_is_on': 'table', 'clear_a': True, 'b_is_on': 'table', 'clear_b': True, 'c_is_on': 'table', 'clear_c': True, 'd_is_on': 'table', 'clear_d': True} --move_b_from_table_to_a--> {'a_is_on': 'table', 'clear_a': False, 'b_is_on': 'a', 'clear_b': True, 'c_is_on': 'table', 'clear_c': True, 'd_is_on': 'table', 'clear_d': True, 'clear_table': True} --move_c_from_table_to_b--> {'a_is_on': 'table', 'clear_a': False, 'b_is_on': 'a', 'clear_b': False, 'c_is_on': 'b', 'clear_c': True, 'd_is_on': 'table', 'clear_d': True, 'clear_table': True} --move_d_from_table_to_c--> {'a_is_on': 'table', 'clear_a': False, 'b_is_on': 'a', 'clear_b': False, 'c_is_on': 'b', 'clear_c': False, 'd_is_on': 'c', 'clear_d': True, 'clear_table': True}

{'clear_a': True, 'a_is_on': 'b', 'clear_b': False, 'b_is_on': 'c', 'clear_c': False, 'c_is_on': 'd', 'clear_d': False, 'd_is_on': 'table'} --move_a_from_b_to_table--> {'clear_a': True, 'a_is_on': 'table', 'clear_b': True, 'b_is_on': 'c', 'clear_c': False, 'c_is_on': 'd', 'clear_d': False, 'd_is_on': 'table'} --move_b_from_c_to_a--> {'clear_a': False, 'a_is_on': 'table', 'clear_b': True, 'b_is_on': 'a', 'clear_c': True, 'c_is_on': 'd', 'clear_d': False, 'd_is_on': 'table'} --move_c_from_d_to_b--> {'clear_a': False, 'a_is_on': 'table', 'clear_b': False, 'b_is_on': 'a', 'clear_c': True, 'c_is_on': 'b', 'clear_d': True, 'd_is_on': 'table'} --move_d_from_table_to_c--> {'clear_a': False, 'a_is_on': 'table', 'clear_b': False, 'b_is_on': 'a', 'clear_c': False, 'c_is_on': 'b', 'clear_d': True, 'd_is_on': 'c', 'clear_table': True}

{'clear_a': True, 'a_is_on': 'b', 'clear_b': False, 'b_is_on': 'c', 'clear_c': False, 'c_is_on': 'd', 'clear_d': False, 'd_is_on': 'e', 'clear_e': False, 'e_is_on': 'table'} --move_a_from_b_to_table--> {'clear_a': True, 'a_is_on': 'table', 'clear_b': True,

'b_is_on': 'c', 'clear_c': False, 'c_is_on': 'd', 'clear_d': False, 'd_is_on': 'e', 'clear_e': False, 'e_is_on': 'table'} --move_b_from_c_to_a--> {'clear_a': False, 'a_is_on': 'table', 'clear_b': True, 'b_is_on': 'a', 'clear_c': True, 'c_is_on': 'd', 'clear_d': False, 'd_is_on': 'e', 'clear_e': False, 'e_is_on': 'table'} --move_c_from_d_to_b--> {'clear_a': False, 'a_is_on': 'table', 'clear_b': False, 'b_is_on': 'a', 'clear_c': True, 'c_is_on': 'b', 'clear_d': True, 'd_is_on': 'e', 'clear_e': False, 'e_is_on': 'table'} --move_d_from_e_to_c--> {'clear_a': False, 'a_is_on': 'table', 'clear_b': False, 'b_is_on': 'a', 'clear_c': False, 'c_is_on': 'b', 'clear_d': True, 'd_is_on': 'c', 'clear_e': True, 'e_is_on': 'table'} --move_e_from_table_to_d--> {'clear_a': False, 'a_is_on': 'table', 'clear_b': False, 'b_is_on': 'a', 'clear_c': False, 'c_is_on': 'b', 'clear_d': False, 'd_is_on': 'c', 'clear_e': True, 'e_is_on': 'd', 'clear_table': True}

Przedstawienie wyników dla problemów oraz porównanie z heurysytką i bez.

In [ ]:
```
import pandas as pd
from tabulate import tabulate

df = pd.DataFrame({
    "Problem": ["Blocks 1", "Blocks 2", "Blocks 3"],
    "Time without heuristic": [time1, time2, time3],
    "Time with heuristic": [time1_h, time2_h, time3_h],
    "Times faster": [time1/time1_h, time2/time2_h, time3/time3_h],
})
table = tabulate(df, headers='keys', tablefmt='pretty')
print(table)
```

```
+---+----------+-----------------------+-----------------------+--------
-----------+
|   | Problem  | Time without heuristic |  Time with heuristic   |    Time
s faster    |
+---+----------+-----------------------+-----------------------+--------
-----------+
| 0 | Blocks 1 |  0.03520774841308594   | 0.0012111663818359375  | 29.0692
91338582676 |
| 1 | Blocks 2 | 0.0018448829650878906  | 0.00021505355834960938 |  8.5787
1396895787  |
| 2 | Blocks 3 |  0.05001497268676758   | 0.0003829002380371094  | 130.621
4196762142  |
+---+----------+-----------------------+-----------------------+--------
-----------+
```