Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra informatiky a výpočetní techniky

SEMESTRÁLNÍ PRÁCE z UZI

Lloydova 15

20.12.2023

Miroslav Martínek

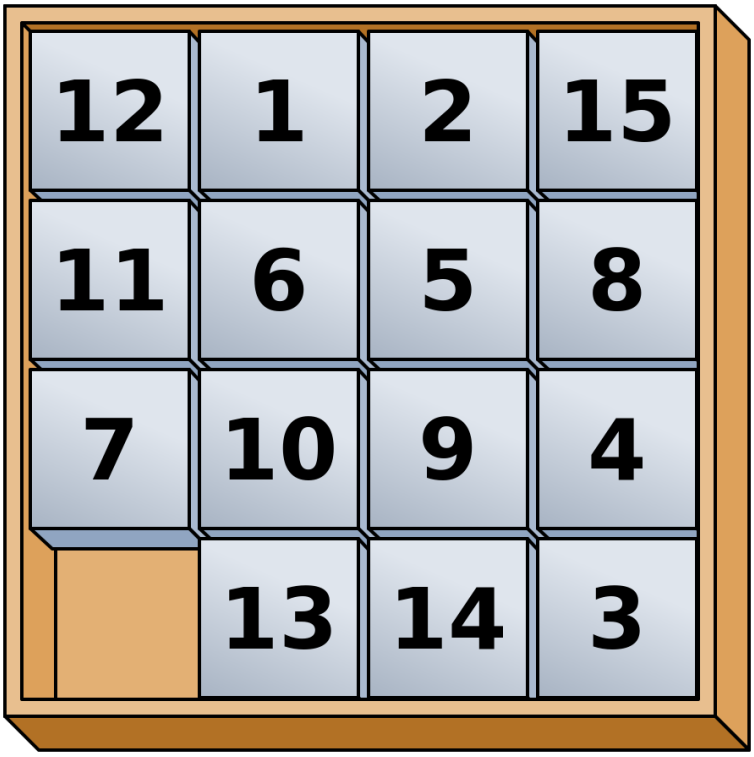
Zadání

Originální znění

Ve čtverci N*N je N*N-1 kostiček a jedno volné pole. Vytvořte program, který složí kostičky tak, aby byly uspořádané. Uspořádání vhodně zvolte.

Rozbor

Při řešení problému tak známe počáteční stav zadaný uživatelem, koncový stav je pro dané N vždy stejný. Cíl programu je nají postup legálních tahů, které převedou počáteční stav do koncového. Legální tahy jsou přesuny přímo sousedících dílků do volného místa.



Obrázek 1 - Ukázka hry. Zdroj: https://en.wikipedia.org/wiki/15_Puzzle#/media/File:15-puzzle_magical.svg

Vstup a výstup proveden pomocí dialogu v příkazové řádce.

Analýza úlohy

Možnosti

Před samotným hledáním řešení je vhodné ověřit, zda je řešení vůbec možné. Pro tuto úlohu je to možné zjistit pomocí vztahů mezi lichostmi a sudostmi počtů inverzí v hracím poli, velikosti hracího pole a řádce na které se nachází volné místo.

Nejčastější řešení takového problému spočívá v prohledávání stavového prostoru.

Prohledávání stavového prostoru je klíčovým prvkem v umělé inteligenci a algoritmice, zejména při řešení problémů, které mohou být reprezentovány ve formě stavového stromu nebo grafu. Způsobů a strategii okolo prohledávání stavového prostoru je však mnoho.

Hledat můžeme od uživatelem daného počátečního stavu do koncového nebo naopak.

Při hledání je možné použít různé způsoby výběru expanze listu ve stromu. Zde je několik hlavních způsobů popsáno:

- Prohledávání do šířky (Breadth-First Search - BFS): Tato metoda postupuje do šířky a nejprve prozkoumá všechny uzly ve stejné hloubce, než se posune na další úroveň. Využívá frontu k ukládání uzlů, které čekají na prozkoumání.
- Prohledávání do hloubky (Depth-First Search - DFS): Na rozdíl od BFS se DFS pohybuje do hloubky, tj. prozkoumává co nejhlubší část stromu předtím, než se vrátí k prozkoumání dalších větví. Využívá zásobník k ukládání uzlů, které čekají na prozkoumání.
- Omezené prohledávání do hloubky (Limited Depth-First Search - LDFS): Omezené DFS stanovuje maximální hloubku prohledávání, což pomáhá zabránit nekonečnému prozkoumávání v neomezeném stromu.
- Iterativní prohlubování do hloubky (Iterative Deepening Depth-First Search - IDDFS): Tato metoda kombinuje DFS a BFS. Postupně zvyšuje hloubku prohledávání od 0 až po dosažení cílového stavu.
- Hladový algoritmus (Greedy algorithm): Jedná se heuristický algoritmus, který v každém kroku vybírá nejlépe ohodnocený list. Hledáme tak lokálně nejlepší řešení, které nemusí být optimální.
- Algoritmus A* (A* algorithm): Jedná se heuristický algoritmus, který v každém kroku vybírá list podle kombinace heuristického ohodnocení a ceny dosažení uzlu. Snažíme se tak najít optimální řešení.

Při použití heuristiky je nutné vybrat heuristickou funkci, takových funkcí existuje několik. Zde je několik příkladů:

- Manhattan vzdálenost: Ohodnocení stavu je součet horizontálních a vertikálních vzdáleností každého dílku od jeho správné pozice.
- Hammingova vzdálenost: Ohodnocení stavu je počet dílků, které nejsou na správné pozici.

Vybraná alternativa

Pro řešení problému jsem zvolil hladový algoritmus s Manhattan vzdáleností pro heuristiku.

Heuristické řešení se jeví pro problém jako vhodné, protože dokážeme dobře odhadnout, jak blízko se daný stav nachází od koncového stavu podle rozmístění dílků. Manhattan vzdálenost je pro problém lepší než Hammingova vzdálenost, protože nám dokáže dát detailnější ohodnocení. Přesunout dílek na druhou stranu herní plochy je určitě náročnější než prohodit dílky, které se nachází blízko sebe.

Z heuristických algoritmů jsem preferoval hladový algoritmus oproti A* algoritmu. Hlavní z předností A* algoritmu je nalezení optimálního řešení, což není v zadání specifikováno a jelikož se jedná o hádanku pro jednoho hráče, optimální řešení zde nemusí být cílem. Hladový algoritmus má jednodušší implementaci a měl by vést k rychlejšímu řešení problému.

Popis algoritmu řešení

Program se spouští scriptem main.py, který má za úkol spustit funkci inferenčního modulu. Inferenční modul slouží k řízení programu a vyřešení úlohy využitím ostatních modulů. Inferenční modul tak obsahuje hlavní smyčku programu, ve které je získán uživatelský vstup, který je následně zpracován a výsledek zpracování je předán uživateli. Nakonec může uživatel program ukončit nebo znovu zadávat vstup pro řešení dalšího hlavolamu.

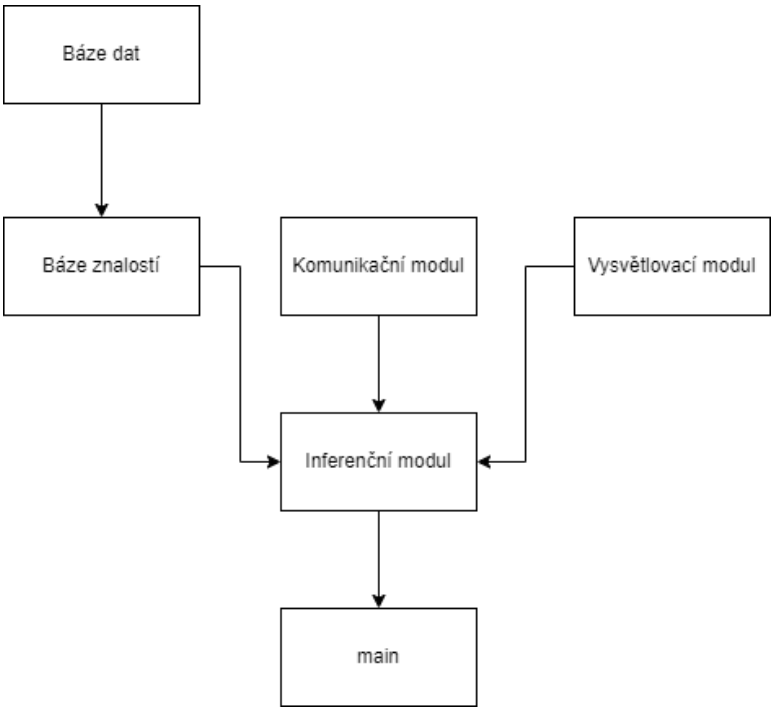
Během procesu jsou pomocí komunikačního modelu vypisovány zprávy pro uživatele.

Uživatelský vstup a dynamické vysvětlování je realizováno pomocí vysvětlovacího modulu. Vysvětlovací modul vstup ošetřuje a získané informace předává ve formě, kterou je zbytek programu schopen jednoduše využít. Získaný uživatelský vstup se skládá z velikosti hracího pole a způsobu, jakým se má počáteční stav vytvořit, jestli manuálně nebo automaticky, a zda uživatel chce pokračovat. V případě, že uživatel vybere manuální zadání, je uživatel opakovaně tázán na herní kameny na místě určeném řádkou a sloupcem. Modul také vysvětluje řešení problému pomocí výpisu nalezeného sledu kroků, které převedou počáteční stav hry do konečného stavu.

Při řešení hry je nutné pracovat s daty, která jsou definována v bázi dat. V tomto případě jsou data reprezentována pomocí třídy GameState. Jedná se o objektové řešení, kde objekty nejen sdružují proměnné dohromady, ale i některé operace (metody objektu).

Pro nalezení řešení hry vzhledem k zadaným parametrům slouží báze znalostí. Báze znalostí obsahuje funkce pro manipulaci s daty a funkci lloyds_n, která realizuje hledání řešení hry. Tato funkce provádí verzi greedy algoritmu, který opakovaně rozvíjí nejlepší list podle heuristického ohodnocení ve snaze najít řešení. Z důvodu optimalizace rychlosti běhu před pracným hledáním řešení funkce ještě provede kontrolu, zda počáteční stav (kořen stromu) je řešitelný. Herní stavy (instance GameState) jsou udržovány v open a closed listech. Closed list je implementován ve formě slovníku pro rychlejší hledání a obsahuje expandované uzly. Open list obsahuje seřazené dosud neexpandované uzly (listy stromu). Greedy algoritmus má následující kroky, které se opakovaně provádí:

1. Kontrola, zda nejlepší stav v open listu není řešení. Pokud se jedná o řešení, vrátíme ho a hledání tak končí.
2. Nejlepší stav je expandován a přesunut do closed listu.
3. Vytvořené potomky profiltrujeme tak, že odstraníme ty, které jsou ekvivalentní nějakému stavu v closed listu.
4. Potomky přidáme do open listu tak, aby zůstal seřazený.

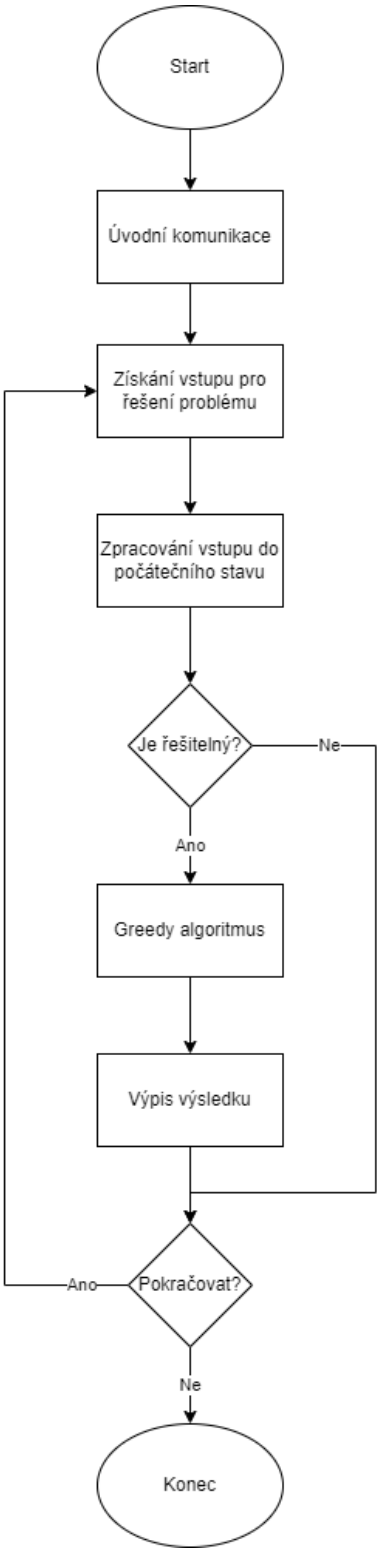


Obrázek 2 - diagram závislosti modulů

Popis programu

Program je vytvořen v jazyku Python. Python je obecný interpretovaný programovací jazyk s jednoduchou, ale silnou základní syntaxí. Jazyk také disponuje rozsáhlou standardní knihovnou. Ta je schopna řešit mnoho základních programovacích problému, což je v části řešení využito. Python podporuje OOP, které jsem využil pro reprezentaci stavového prostoru.

Pro reprezentaci dat jsem implementoval třídu GameState v bázi dat. Třída obsahuje 2D list reprezentující herní pole, x a y souřadnice prázdného políčka, své ohodnocení a referenci na svého předka. Z objektů tak lze vytvořit strom herních stavů. Tento objektový přístup zjednodušuje manipulaci s daty a jejich organizaci.



Obrázek 3 - Obecný vývojový diagram programu

Použité symboly jsou popsány komentáři v kódu, a jejich jména byla zvolena ve snaze vyjádřit jejich význam.

Obsah modulů

Báze dat:

- GameState třída
  - __init__
  - __lt__
  - modification
  - root
  - rate
  - make_children

Báze znalostí:

- same_state
- solvability
- prepare_root
- lloyds_n

Inferenční modul:

- start

Komunikační modul:

- intro
- solution_message
- unsolvable
- start_state_massage

Vysvětlovací modul:

- parameter_input
- field_input
- continue_input
- print_field
- print_step_number

Main:

- main

Popis obsluhy programu

Jelikož je program vytvořen v jazyku Python, stačí mít instalovaný jeho interpret (verze 3). Soubory programu rozbalte a otevřete ve složce se soubory příkazovou řádku a spusťte interpretem script main.py.

Obsluha programu je realizována programem vyžádanými vstupy. Program při žádání o vstup vypíše zprávu, která vysvětluje, o jakou informaci si program žádá.

Ukázka obsluhy:

```
Program resi problem Lloydova 15 pro pocatecni stav o velikosti N*N.
Program dokaze nahodny pocatecni stav vygenerovat nebo ho lze zadat manualne.
Zadejte velikost(cele kladne cislo) hraciho pole N.
```

```
3
Pro nahodny pocatecni stav zadejte 'n'. Pro manualne zadany pocatecni stav zadejte 'm'.
n
Hledam reseni pro pocatecni stav:
-----------------
|3    2    6    |
|8    5    X    |
|7    1    4    |
-----------------
Kroky reseni:
Krok cislo 0
-----------------
|3    2    6    |
|8    5    X    |
|7    1    4    |
-----------------
.
.
.
Krok cislo 71
-----------------
|1    2    3    |
|4    5    6    |
|7    8    X    |
-----------------
Pokud chcete pokracovat zadejte 'ano'.
ano
Zadejte velikost(cele kladne cislo) hraciho pole N.
4
Pro nahodny pocatecni stav zadejte 'n'. Pro manualne zadany pocatecni stav zadejte 'm'.
m
Zadejte hodnotu('X' pro volne misto) pro kamen v 1. sloupci 1. radce:X
Zadejte hodnotu('X' pro volne misto) pro kamen v 2. sloupci 1. radce:1
.
.
.
Zadejte hodnotu('X' pro volne misto) pro kamen v 4. sloupci 4. radce:15
Hledam reseni pro pocatecni stav:
----------------------
|X    1    2    3    |
|4    5    6    7    |
|8    9    10   11   |
|12   13   14   15   |
----------------------
Reseni pro pocatecni stav neexistuje.
Pokud chcete pokracovat zadejte 'ano'.

KONEC
```

Rozbor výsledků

Program je schopen efektivně řešit problém pro přiměřený rozsah uživatelských vstupů. Počáteční stavy velikosti 3 a méně jsou řešeny téměř instantně a stavy velikosti 4 můžou trvat déle (na PC s mobilním Core i5 7 generace od 5 sekund do ~10 minut). Program používá kombinaci prověřených postupů řešení s jednoduchými úpravami pro zvýšení efektivity.

Obsluha programu je základní a prováděna textově. Tento způsob je vhodný pro použití zkušeným expertem, který je schopný program rychle a efektivně obsluhovat, a pro kterého by komplexní grafické řešení představovalo zbytečné nabobtnání programu.

Program by měl být bez větších problémů rozšířitelný. Kód je okomentovaný a rozdělený do modulů. V kódu jsou používány logické názvy proměnných, tříd a funkcí/metod. Komentáře se snaží vysvětlit jejich použití a složitější části kódu bez toho, aby kód zahltily a udělali ho těžce čitelným.

Závěr

Dle mého úsudku práce splňuje zadání. Program je schopen řešit problém Lloydova 15 a jeho struktura je v souladu se strukturou znalostního systému. Obsluha je na požadované úrovni znalce.

Program by však určitě šlo vylepšit. Při hledání řešení by nejspíše šlo najít způsoby dalšího zrychlení procesu nebo případně dát uživateli na výběr z několika algoritmů. Obsluhu by také šlo rozšířit o variantu pro méně znalé uživatele.