Středoškolská technika 2019

Setkání a prezentace prací středoškolských studentů na ČVUT

Řešení algoritmických problémů

Šejvl Petr

SPŠ a VOŠ Písek, Karla Čapka 402, 397 11 Písek

Anotace

Práce se zabývá algoritmickým řešením úlohy z teorie her s konečným počtem řešení. Věnuje se návrhu samotných algoritmů, dokazování jejich správnosti, analýze jejich asymptotické složitosti a testování výsledků. Hlavním cílem je vytvořit alespoň dva algoritmy řešící daný problém v polynomiálním čase.

Klíčová slova

Teorie her, algoritmus, analýza, asymptotická složitost

Annotation

The thesis deals with algorithmic solving of a task from the game theory with a finite count of solutions. The thesis proposes these algorithms, proves their correctness, analyzes their asymptotic complexity and tests the results. The primary goal is to create at least two such algorithms solving the given problem in polynomial time.

Keywords

Game theory, algorithm, analysis, asymptotic complexity

Obsah

1. Úvod

První část práce se zabývá řešením úlohy z teorie her, konkrétní zadání: https://fiks.fit.cvut.cz/files/tasks/season4/round2/teleportace.pdf. Úloha nabízí různé postupy a způsoby řešení. V rámci práce jsou navrženy dva takové algoritmy, které mají polynomiální asymptotickou složitost. V rámci práce je jejich asymptotická složitost a funkčnost dopodrobna analyzována a dokázána.

Pro srovnání je navrženo a naimplementováno ještě jedno řešení, které má asymptotickou časovou složitost nepolynomiální.

V druhé části jsou pak rozebírány způsoby řešení problémů z teorie her s velmi vysokými počty řešení a řešení heuristická.

1.1 Teorie her

Pod pojmem "Teorie her" rozumíme oblast problémů spadající pod aplikovanou matematiku. Zabývá se situacemi, ve kterých hráči (hráči mohou být např. i obchodní společnosti soupeřící mezi sebou o přízeň klientů) dělají rozhodnutí. Jejich rozhodnutí je pak dovedou buď k nějaké odměně, nebo k nějakému trestu. Hráči se při hře vzájemně ovlivňují.

Základní dvě rozdělení jsou na cooperative (spolupracující) a non-cooperative (nespolupracující).

- Cooperative:
  - hráči spolupracují na dosažení společného cíle,
  - benefity, vyplývající z dosaženého cíle se pak mezi sebou snaží rozdělit úměrně k zásluhám na jeho dosažení,
  - optimálním rozdělením výsledků této společné snahy je tzv. Shapleyova hodnota.
- Non-cooperative:
  - v takových hrách hráči nespolupracují, naopak mezi sebou vzájemně soutěží,
  - všichni hráči hrají tak, aby získali ze hry co nejvíce. Výsledkem hraní takové přímo dominantní strategie je Nash equilibrium, to vyjadřuje stav hry po tom, co se oba hráči snaží získat co nejvíce pro sebe,

- známým příkladem je tzv. Vězňovo dilema, ukážeme si příklad jednokolové hry dvou hráčů.
  - Dva zločinci jsou chyceni a zatčeni. Policie má dostatek důkazů, aby je oba uvěznila na dva roky. Pokud by ale jeden z nich spolupracoval, získá tak svobodu a jeho komplic dostane 10 let. Pokud ovšem budou spolupracovat oba, každý z nich bude zatčen na šest let.
  - Pojmenujme vězně Adam a Bob a podívejme se na obrázek 1.

	Bob mlčí.	**Bob mluví.**
Adam mlčí.	Oba odsoudí na 2 roky.	Adam dostane 10 let, Bob bude volný.
Adam mluví.	Adam bude volný, Bob dostane 10 let.	Oba odsoudí na 6 let.

Obrázek 1 - Vězňovo dilema

Protože se nemohou domluvit, a oba chtějí mít co nejmenší možný trest, oba budou spolupracovat s policí. To je sice dovede k většímu trestu, než kdyby nespolupracoval ani jeden, ale to nemohou vědět.

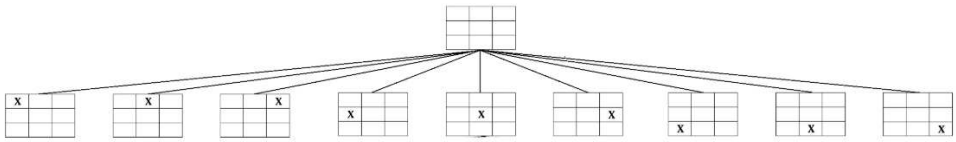
- V nekooperativních hrách se tedy každý hráč snaží získat co nejlepší situaci, i když to může vést k výše zmíněnému paradoxu.

Využití teorie her nacházíme v širokém množství sociálních věd, počítačových věd a dalších. V rámci teorie her se analyzuje široké spektrum konfliktních rozhodovacích situací. Takové situace mohou nastat všude, kde dochází k nějakému střetu zájmů. Tyto situace se pak snažíme analyzovat a nalézt co nejlepší strategie a konkrétní řešení pro jejich účastníky.

Tyto situace a úlohy se dají simulovat a řešit pomocí algoritmů, díky různým rozhodnutím pak vznikají herní stromy.

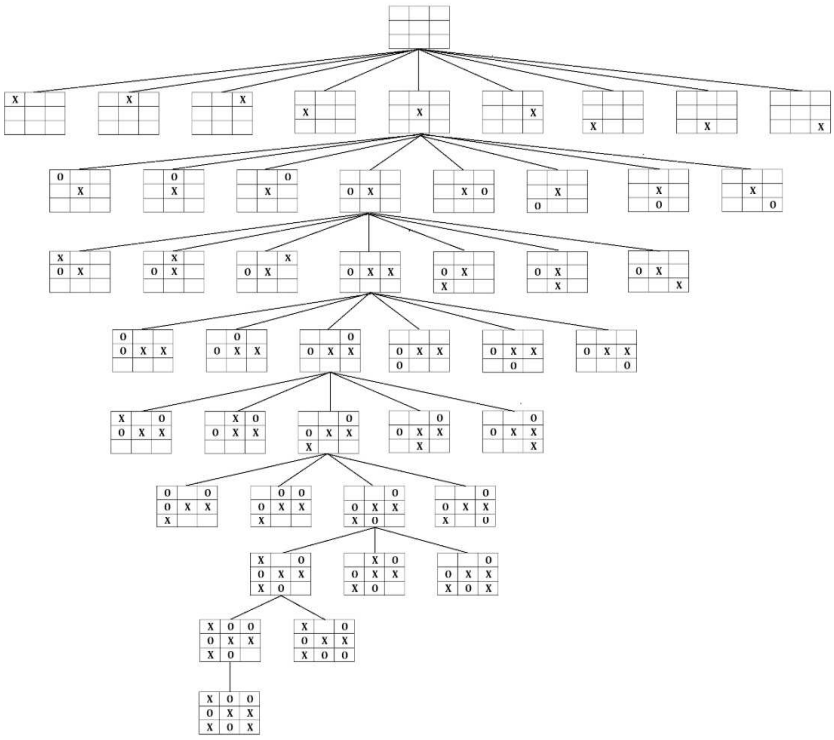
1.1.1 Herní strom

Herní strom je orientovaný graf, kde uzly reprezentují pozice a hrany jsou tahy, které na tyto pozice vedly. Podívejme se například na piškvorky na hřišti 3 x 3, na začátku máme prázdné hřiště a můžeme zahrát celkem 9 tahů.



Obrázek 2 - první úroveň herního stromu

Jak je vidět, každý tah vede k nové pozici. Pro všechny nové pozice je počáteční stav hry, tedy prázdné hřiště, jejich rodič a nové pozice jsou jeho děti. Takové hry, které se již dále v rámci stromu nevětví (i takové, které se větvit můžou, jen nemají větvení ještě propočtené) se pak nazývají listy. Pomocí takového stromu lze vytvořit a projít všechny možné herní situace. V tomto případě by složitost herního stromu byla tolik, kolik je faktoriál herních polí, v tom případě tedy $(3 * 3)! = 362\,880$. Ukážeme si větvení pro jednu z možných her na obrázku 3.



Obrázek 3 - jedno z možných větvení herního stromu

1.2 Algoritmus

Algoritmus je schématický postup pro řešení určitého druhu problémů. Je prováděn pomocí konečného počtu přesně definovaných instrukcí.

1.2.1 Vlastnosti algoritmů

- Konečnost
  - Každý algoritmus musí mít konečný počet stavů.
  - Musí být konečný v závislosti na potřebné paměti.
- Obecnost
  - Algoritmus neřeší konkrétní problémy
    - Např. neřeší kolik je konkrétní výsledek výpočtu výrazu $3 * 7$, ale řeší vypočtení obecně součinu dvou čísel.
- Výstup
  - Algoritmus má vždy nějaký výstup.
- Úplnost
  - Algoritmus musí popsat všechny stavy, které mohou nastat.

1.3 Asymptotická složitost

Jedná se o přibližné složitosti algoritmů, obecně začínají platit až od určitě velikosti vstupů. K jejímu zápisu se užívá níže popsaná Landauova notace.

Díky složitostem se dají algoritmy snadno srovnávat s ohledem na jejich efektivitu v závislosti na velikosti vstupních údajů, které dostávají ke zpracování. Podle počtu kroků potřebných k vypočtení výsledku v závislosti na velikosti vstupu se pak podle jejich složitosti algoritmy dělí do časových složitostních tříd. Pro vypočtení této asymptotické složitosti záleží pouze na škálování s ohledem na velikost vstupu, všechny multiplikativní konstanty je pak tedy možné zanedbat

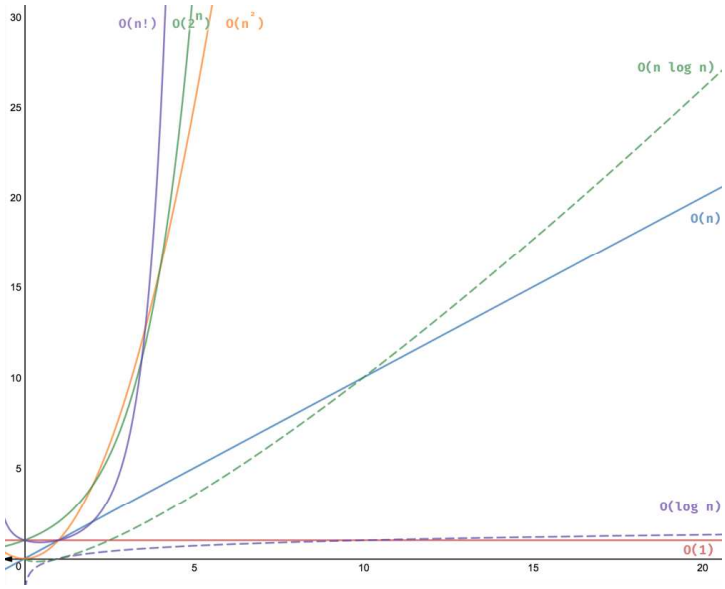
- Např. algoritmus, který pro každý jeden prvek na vstupu udělá 100 operací by měl asymptotickou časovou složitost O $(100 * N)$. Algoritmus, který pro každý jeden prvek na vstupu udělá 1 operaci, má asymptotickou časovou složitost O (N). Všechny multiplikativní konstanty nemají z hlediska těchto asymptotických složitostí význam, proto spadají oba algoritmy do stejné třídy asymptotických složitostí a budou mít tedy asymptotickou časovou složitost stejnou, konkrétně O (N).

1.3.1 Landauova notace

Jinak nazývána “Big O notation” je způsob zápisu asymptotických složitostí. Jde o matematickou notaci vyjadřující chování funkce, když se její argument blíží určité hodnotě nebo nekonečnu.

1.3.2 Základní třídy asymptotické složitosti

- Konstantní - O (1)
    - Algoritmu s takovou složitostí nezáleží na velikosti vstupu, např. bude potřebovat stejný počet kroků pro vstup o velikost 100 i o velikosti 1000
    - Např. časová složitost hledání prvku v poli dle jeho indexu
- Logaritmická - O ($\log N$)
    - Algoritmus potřebuje k dokončení asi tolik kroků, kolik je logaritmus velikosti jeho vstupu.
    - Např. časová složitost binárního vyhledávání
- Lineární - O (N)
    - Je třeba asi tolik kroků, jak je vstup velký.
    - Jednoduchým příkladem je algoritmus, který v jednom cyklu prochází vstupy
- Linearitmetická - O ($N * \log N$)
    - Počet kroků odpovídá počtu prvků krát logaritmus tohoto počtu
    - Např. časová složitost Merge sort a další řadící algoritmy
- Kvadratická - O (N^2)
    - Jedna z již horších složitostí, počet kroků odpovídá velikosti vstupu na druhou
    - Např. paměťová složitost při reprezentaci grafu maticí
- Exponenciální - O (2^N)
    - Velmi velká složitost
    - Např. časová složitost problém obchodního cestujícího řešeného pomocí dynamického programováním
- Faktoriálová – O ($N!$)
    - Extrémní složitost
    - např. časová složitost problému obchodního cestujícího hrubou silou



Obrázek 4 - srovnání časových složitostí

1.4 P x NP

Tento problém úzce souvisí s asymptotickými složitostmi a jelikož se v rámci práce navrhují řešení s polynomiální časovou složitostí, tak i s touto prací.

1.4.1 P problémy

Jedná se o problémy, jejichž složitost řešení je při nejhorším polynom. Exponenty v polynomech jsou vždy kladná celá čísla nebo nula, nikdy jimi nebude proměnná.

Příkladem problému s polynomiální složitostí je algoritmus se složitostí O $(N^5 + N^3)$

1.4.2 NP problémy

Problémy, jejichž složitost řešení je větší než polynomiální. Taková složitost je například exponenciální O (2^N), faktoriálová O $(N!)$ apod. Řešení NP problémů je ovšem možné v polynomiálním čase zkontrolovat

1.4.3 P x NP srovnání

Zatímco u polynomiální složitosti roste doba potřebná k dokončení algoritmu relativně přijatelně, u nepolynomiálních složitostí tomu tak není. Problémy v této třídě složitostí je prakticky nemožné vyřešit pro větší vstupy. Proto se snažíme mít algoritmy s maximálně polynomiální složitostí. Jestli je ale vždy možné upravit algoritmy s nepolynomiální složitostí na algoritmy se složitostí polynomiální, nebo to možné není, nebylo zatím nikým prokázáno.

2. Zadání úlohy

Úloha představuje hru, ve které se dva hráči pohybují po poli o velikosti N x M. Začínají v levém horním rohu a mohou se pohybovat pouze dolů, doprava, nebo dolů a doprava zároveň. K dispozici mají omezený počet konkrétních pohybů, které mohou používat opakovaně. Hráči se pohybují společně, stojí tedy pokaždé oba na stejném místě. Při tomto společném přemisťování se střídají v tom, kdo je na řadě s tahem, tedy vybráním jednoho z pohybů, o který se na herní ploše přemístí. Ten hráč, při jehož tahu už nezbyde jiná možnost, než vystoupit z výše zmíněného herního pole pak prohrává. Naším cílem je ze vstupů vždy určit, který hráč vyhraje (ten který začíná, nebo ten který hraje jako druhý) a který prohraje, hrají-li oba optimálně.

Úloha se řeší pro vstupy v přesně definovaném rozmezí.

2.1 Vstupy

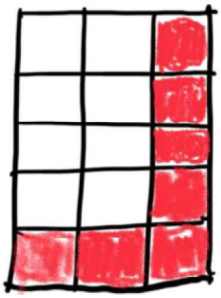
- Velikost pole N x M
  - 1 <= N, M, <= 400
  - N vyjadřuje šířku pole (velikost na ose X)
  - M vyjadřuje výšku pole (velikost na ose Y)
- Počet pohybů P
  - K dispozici bude tolik různých pohybů, kolik je tento vstup P. Tyto pohyby je možné neomezeně opakovat.
  - 1 <= P <= 20
- Celkem P konkrétních pohybů ve formátu A B
  - 0 =< A, B <= 500
  - Alespoň jedna souřadnice pohybu A, nebo B musí být nenulová
  - Číslo A vyjadřuje, o kolik se posuneme na ose X
  - Číslo B vyjadřuje, o kolik se posuneme na ose Y

3. Analýza úlohy

Protože taková hra má konečné množství řešení, můžeme tedy ze známých vstupů určit a vypočítat, kdo bude jejím vítězem. Víme, že oba hráči volí ideální strategii a bude-li možné se zachovat tak, že jejich tahy povedou k jisté výhře, udělají to. Druhý hráč pak prohraje pouze kvůli tomu, že mu pohyby na těchto rozměrech hřiště nedaly jinou možnost, nikoliv kvůli špatně zvolené strategii.

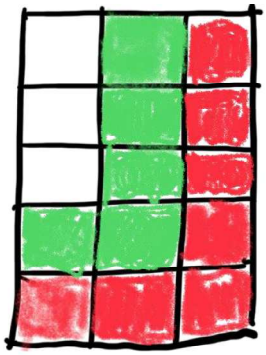
Hru hrají právě dva hráči, pro přehlednost je pojmenujeme, jako v originálním zadání, Martien (vždy začíná) a Pepie.

Vlastnosti políček se musí určovat od konce, protože tam už hráčům zbývá pouze možnost, jejíž výsledek je ze zadání definován (vykročení ven). Díky tomu můžeme definovat další políčka a zpětně se dostat až na políčko počáteční. Podíváme-li se na mřížku, kterou nám hřiště N x M vytvoří a na pohyby, které jsme získali na vstupu a na fakt, že hráč, který vykročí z herního pole ven, prohrál, můžeme ihned určit všechna políčka, na kterých už nepůjde nic, než prohrát. Jedná se o políčka, ze kterých může hráč už pouze vykročit ven a prohrát. Na obrázku 5 je příklad s pohybem [1, 1].



Obrázek 5 - proherní políčka s pohybem [1, 1]

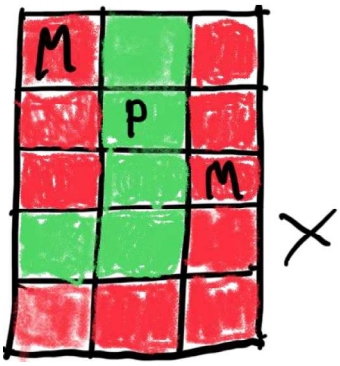
Na těchto políčkách již tedy může daný hráč pouze vykročit ven a prohrát, říkejme jim tedy "proherní". Protože oba hráči hrají optimálně a ten, který neprohraje, pochopitelně vyhrává, je tedy cílem oba hráčů nechat soupeře prohrát. Soupeře donutí prohrát tak, že se s ním přemístí na výše popsané proherní políčko, tam se změní kdo je na tahu a soupeř prohraje. Ukažme si nyní na stejném příkladu všechna políčka, odkud může (např. Martien) vždy jít na proherní pole, kde bude na tahu Pepie, a tím pádem vyhraje Martien:



Obrázek 6 - výherní políčka s pohybem [1, 1]

Když se podíváme na zeleně vybarvená políčka, je jasné, že hráč na zeleném může zahrát nějaký tah a dostat tak hráče na políčka červená, která jsou proherní. Políčka, na kterých je jistá výhra si tedy nazvěme "výherní".

Testování, je-li políčko výherní, bude poměrně jednoduché, protože to je každé políčko, ze kterého se dá dostat na alespoň jedno proherní. Proherní je pak políčko tehdy, když se z něj nedá dostat na jiné než výherní (zahrát na výherní nikdo nechce, protože tam se změní kdo je na tahu a hráč, který zahrál na výherní, pak prohraje). Když se podíváme na náš rozpracovaný příklad, z pozice v levém horní rohu se dá tahem [1, 1] dostat pouze ne výherní pozici. Tuto situaci tedy vyhraje Pepie.



Obrázek 7 - kompletní hra s pohybem [1, 1] na hřišti 5 x 3

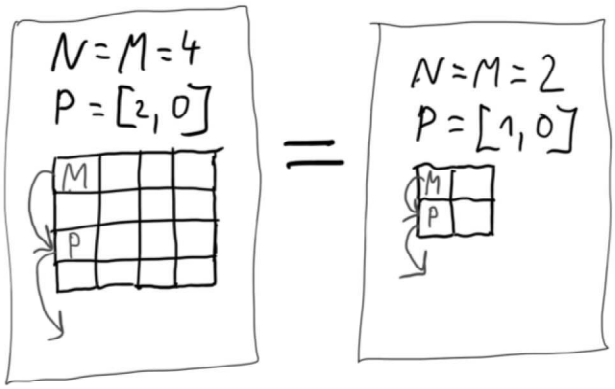
Pomocí výše popsaných pravidel pro odhalování výherních a proherních pozic pak budeme tvořit algoritmy na vytvoření úlohy.

3.1 Optimalizace

3.1.1 Zjednodušení hřiště a pohybů

Pokud by byla všechna čísla (rozměry hřiště a jednotlivé souřadnice všech pohybů) dělitelná jedním číslem větším než 1, můžeme celou hru poněkud zjednodušit. Protože hra na hřišti 4 x 4 s pohybem [2, 0] je vlastně stejná jako hra na poli 2 x 2 s pohybem

[1, 0], můžeme nalezeným společným dělitelem všechna čísla vydělit a hrát tak na menším hřišti.



Obrázek 8 - optimalizace zmenšením hřiště a pohybů

3.1.2 Odstranění nepoužitelných pohybů

Na začátku hry se ještě vyplatí projít všechny pohyby a zjistit, jestli je vůbec lze někdy zahrát. Např. když je hřiště 4 x 4 a jeden pohyb by byl [5, 0], je jasné že tento pohyb již ze začátku hry nelze hrát. Takové pohyby můžeme rovnou vyfiltrovat a odstranit.

Pokud by takovém filtrování nezbyl žádný pohyb, je jasné že jsme měli jen takové pohyby, které vedly hned z prvního tahu ven a vyhrál tedy Pepie.

4. Řešení úlohy hrubou silou

Prvně rozebereme naivní neefektivní řešení hrubou silou. Takové řešení sice bude na větších vstupech nepoužitelné, ale pomůže nám lépe pochopit úlohu a z tohoto řešení hrubou silou pak můžeme navrhnout další optimalizace a vylepšení.

Řešením bude algoritmus rekurzivně procházející herní pole do hloubky, půjde z levého horního rohu a bude hrát stále první pohyb, dokud nevykročí z hřiště ven. Pak se začne vracet zpět a popřípadě zkoumat další možnosti. Hru řešíme tak, že zkoumáme, jestli vyhraje Mariten (vrátí se True), nebo prohraje (vrátí False).

Řešením tedy bude rekurzivní funkce s následujícími parametry:

- X - souřadnice testované pozice na ose X
- Y - souřadnice testované pozice na ose Y
- Playground - rozměry hřiště
- Spells - seznam pohybů, název tohoto parametru vychází z původního zadání
- Player - boolean, jejíž hodnota vyjadřuje kdo je právě na tahu

Na začátku načteme vstupy a zavoláme výše popsanou rekurzivní funkci. Tato funkce nejprve zkontroluje, jestli je stále na herním poli. Pokud není, vrátí negaci parametru Player (tím vyjádří, kdo byl v předchozím tahu na řadě a vykročil tedy ven). Pokud je stále na herní ploše, pokusí se hrát dál. Z každé pozice, na kterou takto dojde, odehraje všechny ze vstupu získané pohyby a bude sledovat, jaké hodnoty se vrací.

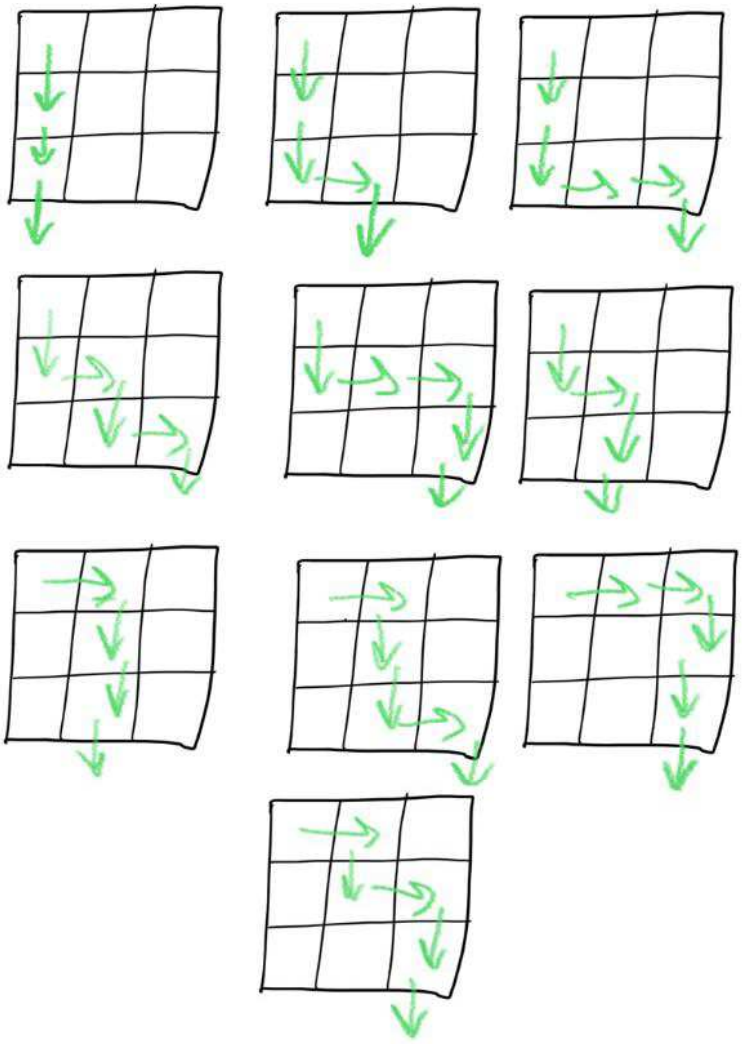
Když dojde k vykročení z mapy, nebo by byl tah z jedné pozice na pozici proherní, pokusíme se odehrát z té pozice ještě další tahy, zda nás nedovedou k něčemu lepšímu. Pokud v rámci procházení tahu bude alespoň jeden na tah na pozici proherní, další zkoumání nemá význam, protože z dané pozice je výhra očividná.

Tento algoritmus je hodně jednoduchý, problém je ovšem v jeho složitosti.

4.1 Asymptotická časová složitost

Časová složitost tahového řešení hrubou silou je nepolynomiální. Možností na projití mřížky o velikost N x M, budou-li pouze dva pohyby, a to o 1 dolů a o 1 doprava je možných celkem $\frac{(N+M)!}{N! * M!}$. Konkrétně je složitost celkového algoritmu minimálně O ($P^{\max(N,M)}$). K takové složitosti by se došlo, pokud by se jeden z rozměrů hřiště rovnal jedné, na druhém by pak bylo nutné pro každou z dosažitelných pozic zahrát

všechny pohyby, a to včetně pozic dosažitelných vícekrát. V případě, kdy je druhý rozměr větší než jedna, by pak složitost byla ještě větší. Zde je pro představu náčrt, jak by vypadala polovina všech možných řešení, pokud bychom měli pole 3 x 3 a pohyby [1, 0] a [0, 1]. Načrtl jsem pouze situace, kde hra končí pod osou Y, pod osou X by jich pak bylo stejné množství a došli bychom k nim stejným způsobem, jelikož by stačilo hřiště pootočit o 90°.



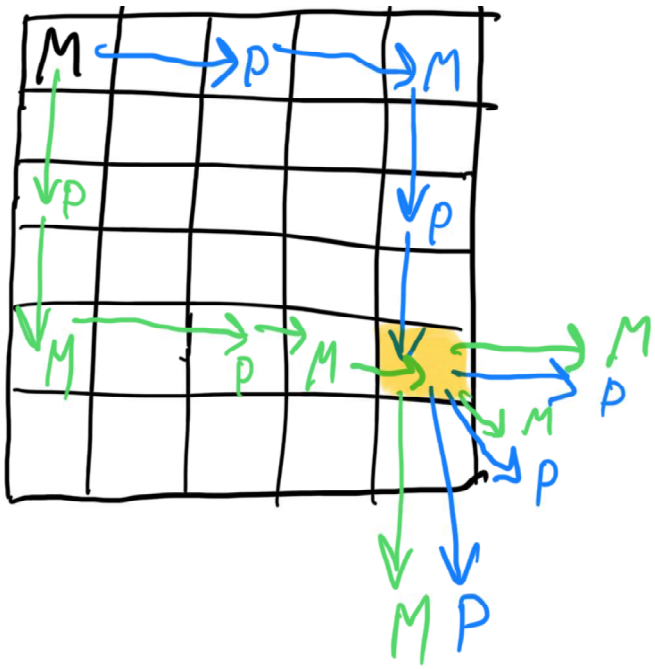
Obrázek 9 - herní možnosti při řešení hrubou silou

4.2 Asymptotická paměťová složitost

Protože takto navržený algoritmus si nic do paměti neukládá (pouze načítá vstupy na začátku), jeho paměťová složitost je tedy konstantní O (1).

5. Řešení prohledáváním hřiště do hloubky dynamickým programováním

Podíváme-li se na rozbor úlohy, je jasné, že řešení hrubou silou není optimální. V předchozím řešení se prochází všechny možnosti a výsledky se vrací s ohledem na to, kdo byl na tahu. Pokud např. Martien z jednoho pole prohraje, je jasné, že stojíc na tomto poli by prohrál i Pepie. Procházet všechny možnosti a střídat tahy hráčů tedy nemá význam, důležité je sledovat vlastnosti polí (proherní/výherní) bez ohledu na to, kdo je na tahu. Ukázka, jak by z jednoho pole vyhrál jednou Martien a podruhé Pepie. Ukázka je na hřišti 6 x 5 se třemi pohyby [2, 0], [0, 2] a [0, 1] je na obrázku 10.



Obrázek 10 - proč při klasifikaci polí nezáleží na hráčích

K původní funkci tedy přidáme ještě další parametr, tím bude resultMap. Jedná se o dvourozměrné pole N x M, do kterého budeme zapisovat stavy polí po tom, co je zjistíme. Naopak boolean, pomocí které jsme sledovali kdo je právě na tahu, můžeme odstranit.

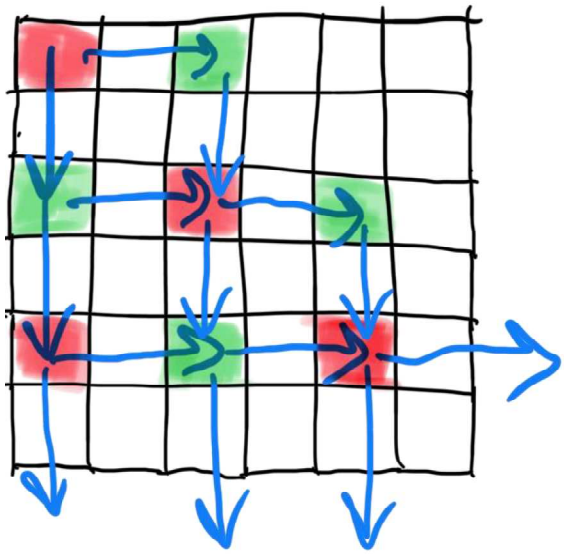
Ve výše zmíněné výsledkové mapě bude výherní pole značeno jako True a proherní jako False.

Nejprve se opět podíváme, jestli je testovaná pozice stále na herním poli.

- Pokud není, vrátíme True (vystoupení z pole ven znamená pro toho, kdo zde stojí a je na tahu výhru, tím pádem lze v tomto případě brát jako výherní).

- Pokud je a již bylo navštíveno, pouze vrátíme tu hodnotu, kterou máme uloženou v naší výsledkové mapě.
- Pokud je a navštíveno ještě nebylo, budeme jej tedy testovat.
    - Nejprve předpokládejme, že je proherní.
    - Budeme pro všechny dostupné pohyby volat tuto rekurzivní funkci a pokud se budeme moct dostat na alespoň jednu proherní pozici, bude testovaná pozice výherní. Pokud nenajdeme žádnou proherní pozici, je proherní i tato pozice jak jsme na začátku předpokládali.
    - Výsledek uložíme do výsledkové mapy a vrátíme ho, takže s ním budou moci pracovat předchozí volání rekurzivní funkce a zjišťovat podle toho správné výsledky. Tímto způsobem se tedy budou vracet správné výsledky, vždy zjišťované od pozic, kde se dá již pouze vystoupit a budou se propagovat hřištěm zpět.

Fungování algoritmu si ukážeme na hřišti 6 x 6 s pohyby [2, 0] a [0, 2], to by sice šlo dle předchozího popisu zjednodušit, takhle to bude přehlednější.



Obrázek 11 - řešení hry dynamickým programováním

Tuto hru by tedy vyhrál Pepie, protože Martien ho z prvního pole může poslat pouze na výherní pole.

5.1 Asymptotická časová složitost

Vylepšením zavedením této tabulky pro průběžné ukládání hodnot polí se časová složitost velmi výrazně sníží. Přes každé pole přejdeme maximálně jednou a polí je

N x M. Na každém poli pak můžeme zahrát až P pohybů. Nejhorší časová složitost bude tedy přesně taková, tedy O ($N * M * P$). Taková složitost je polynomiální a pro hřiště, jehož velikost je maximálně 400 x 400 s maximálně 20 pohyby bude velmi efektivní.

5.2 Asymptotická paměťová složitost

Snížení asymptotické časové složitost s sebou ovšem nese zvýšení složitosti paměťové. Aby bylo možné přejít přes každé políčko maximálně jednou, musíme je ukládat do paměti. Pro herní pole M x N musí mít taky M x N záznamů, paměťová asymptotická složitost toho algoritmu bude tedy stejná jako ta časová, tedy O ($N * M$). Při velikostech vstupů definovaných v zadání, je tato složitost poměrně nízká.

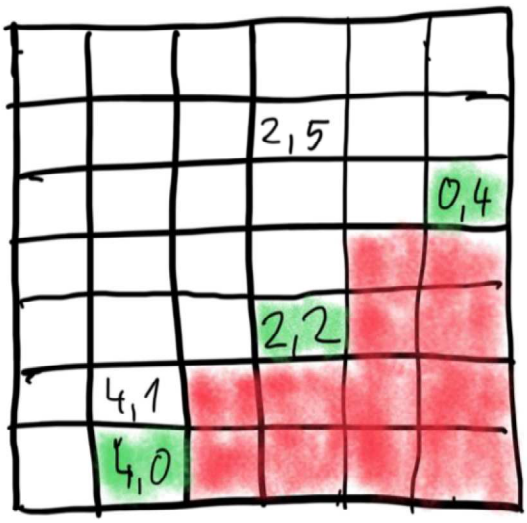
6. Řešení prohledáváním do šířky

Další možný způsob řešení je navržen úplně opačným způsobem než předchozí dva postupy. Zatímco řešení hrubou silou i jeho dynamické vylepšení prochází hřiště do hloubky, a pak zpětně propaguje získané výsledky, tento způsob na řešení půjde opačně. Půjdeme odspoda (tj. od proherních pozic na hranici hřiště) a půjdeme do šířky. Prohledáváním hřiště do šířky je pak myšleno, že budeme vždy předně hrát všechny pohyby ze všech pozic a prozkoumávat tak hřiště postupně, ne jako v předchozích případech, kde jsme nejprve prozkoumávali jednu cestu až do jejího konce a podle toho, co nám řekla, jsme upravovali cestu, kterou šla.

Opět budeme pro lepší efektivitu využívat tabulku, do které budeme průběžně ukládat hodnoty polí.

Jak víme, když se z nějakého pole dá pouze vykročit ven, tak je proherní. Když se podíváme na naše dostupné pohyby a rozměry hřiště, můžeme ihned odhalit všechny takové pozice. Najdeme je pomocí "hraničních pohybů", tedy takových pohybů, které půjdou zahrát tam, kde už žádné jiné. Podíváme-li se například na pohyby [1, 1] a [4, 4], je poměrně jasné, že mohou nastat takové situace, kde půjde první z pohybů zahrát a druhý však nikoliv.

Hraniční pohyby, které potřebujeme najít, se značí tím, že mají posun alespoň po jedné ose menší než všechny pohyby ostatní. Hraničních pohybů může ale být více než dva, například vezměme situaci s pohyby [4, 0], [0, 4] a [1, 1]. Je jasně vidět, že i když první dva pohyby mají posuny každé po jedné ose nejmenší, pohyb [1, 1] i tak ale půjde zahrát tam, kde tyto dva ne. Funkce hledající hraniční pohyby bude rekurzivní a bude pohyby hledat do té doby, dokud nebudou pohyby rozděleny do dvou skupin. Buď budou mezi hraničními, nebo budou oba jejich posuny větší než oba posuny jednoho z pohybů, který je mezi hraničními, a tudíž nebudou pro tuto zónu nijak zajímavé. Ukázka, červeně jsou vyznačena pole, kde již nejde nic zahrát. Zeleně je pak vyznačeno, kde by šly jednotlivé hraniční pohyby zahrát, a vedly by do pravého dolního rohu. Pohyby [4, 0], [0, 4], [2, 2], [4, 1], [2, 5].



Obrázek 12 - prohernní zóna

Podíváme-li se na červeně vybarvená políčka, je jasné že vytváří určitou zónu, ze které již nelze hrát jinak než ven. Jsou tedy prohern. Protože víme, že výherní pole jsou taková, která vedou na prohern, můžeme je z této zóny ihned najít. Vezmeme odtud všechny naše pohyby a zahrajeme je zpět (tímto jdeme tedy do šířky) a všechna tato pole můžeme označit jako výherní.

Situace nyní je poněkud komplikovanější, protože všechna prohern pole vedou pouze na výherní pole. Opět vytáhneme všechny tahy do šířky zpět, tentokrát z nově objevených výherních polí. Tyto tahy si zaznamenáme a budeme testovat, jestli jsou prohern nebo výherní. Testování proběhne tak, že budeme nejprve opět předpokládat, že jsou prohern. Pokud se půjde dostat na jedno jiné prohern, tohoto předpokladu se vzdáme a označíme pole jako výherní. Pole pak odstraníme ze seznamu, ve kterém bylo zaznamenané. Pokud se ovšem půjde dostat na takové pole, které ještě není klasifikováno a na žádné prohern pole, nemůžeme aktuálně testované pole rozhodně hodnotit. Ponecháme ho tedy v proměnné, do které jsme si ho zaznamenali a jeho hodnotu ve výsledkové mapě nebudeme nijak měnit, vrátíme se k němu v další iteraci. Pokud nebude možné se dostat do ani jedné ze dvou popsaných situací, pole je opravdu prohern, a tak ho tedy i označíme. Tímto způsobem pak zjistíme, vlastnosti všech potřebných políček, včetně toho počátečního a vrátíme správný výsledek.

6.1 Asymptotická časová složitost

Opět se složitosti budou odvíjet od využité tabulky na průběžné zapisování výsledků, která je velikosti N x M. Tentokrát má ještě výraznější vliv, kolik pohybů máme

k dispozici, protože při hře do šířky je hrajeme z klasifikovaných pozic často všechny. Časová asymptotická složitost bude opět O ($N * M * P$). Algoritmus je tedy polynomionální. Jeho průměrná časová složitost bude ale výrazně horší, než při dynamickém procházení do hloubky. Při šíření do šířky totiž klasifikujeme mnoho polí i zbytečně, protože v rámci reálné hry se na ně nikdy nepůjde dostat. Příkladem je hra 10 x 10 s pohybem [0, 2]. V takové hře (bez optimalizací) by dynamické prohledávání do hloubky nikdy nešlo na políčka na lichých řádcích (počítaje počáteční políčko v nultém sloupci a na nultém řádku), zatímco toto šíření do šířky je klasifikuje. Jejich klasifikace je jasně zbytečná, protože se na ni z počáteční pozice nikdy nedá dostat.

6.2 Asymptotická paměťová složitost

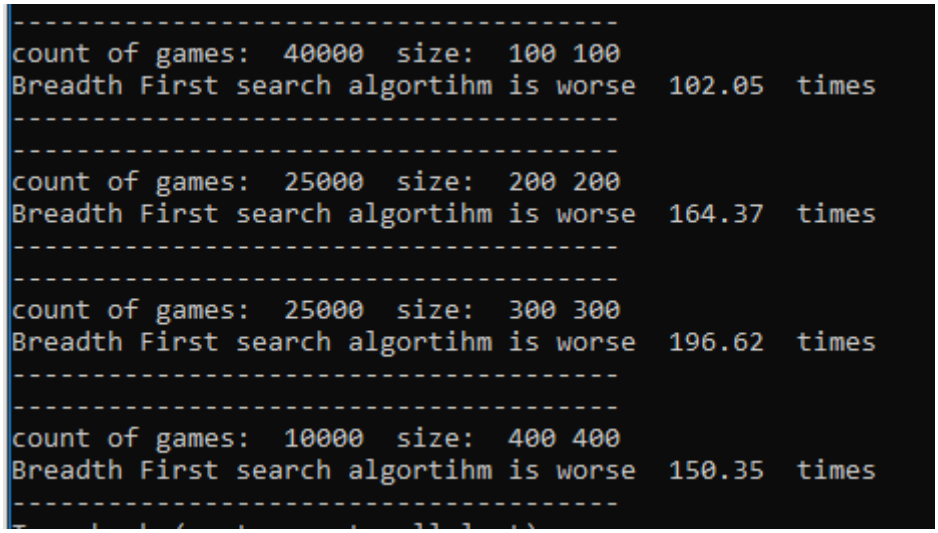
Co se týká paměti, situace je stejná jako u dynamického prohledávání do hloubky. V paměti držíme výsledkovou mapu hřiště M x N, taková je tedy i tato paměťová složitost O ($N * M$).

7. Srovnání algoritmů

7.1 Z hlediska časové složitosti

Nejlepší je řešit algoritmus dynamickým procházením do hloubky. Algoritmus hledající do šířky nabídl také časově přijatelné řešení, je také polynomiální, jen díky procházení a klasifikování i zbytečných polí a často testování všech pohybů ze všech pozic je horší. Řešení hrubou silou, které ani není polynomiální, je pak výrazně horší. Procházení všech pozic vytvoří extrémní množství vícekrát hraných stejných her a s těmi předchozími se nemůže rovnat.

Pro porovnání toho, jak dlouho algoritmus opravdu trvá, jsem nechal oba polynomiální algoritmy řešit zadání s různými složitostmi vstupů a srovnával, kolikrát byl algoritmus prohledávající do šířky horší:



Obrázek 13 - srovnání polynomiálních algoritmů

Celkově byl druhý algoritmus tedy horší asi 146 krát, v průměru z průměrů asi 153 krát. Podíváme-li se, jak rostou vstupy, tak rozdíly ve složitosti příliš nerostou a ve dvou ze čtyř hodnot pak skoro odpovídají celkovému průměru. Je z toho tedy vidět, že algoritmy jsou opravdu ve stejné třídě složitostí, přestože jeden je efektivnější.

7.2 Z hlediska paměťové složitosti

V tomto ohledu jsou oba polynomiální algoritmy v podstatě stejné, oba musí držet tabulku výsledků a od té se odvíjí i jejich paměťová složitost. Řešení hrubou silou je na tom lépe, tato složitost je konstantní. Algoritmus ale nelze použít, jak bylo již zmíněno,

na těžších vstupech, ani tato konstantní složitost mu tedy nezíská příliš mnoho preferencí.

7.3 Z hlediska stability

Zde jsou všechna řešení stejná, jejich stabilita je 100%. Všechna řešení byla testována proti referenčnímu řešení, které vyšlo k úloze a vždy dosáhla stejných výsledků. Konkrétně pro polynomiální řešení bylo při hlavním testování provedeno celkem 2 000 000 testů opět na různě složitých hrách a všechny skončily úspěšně. Pro řešení hrubou silou bylo třeba velikost vstupů značně omezit, konkrétně na hřiště o velikost maximálně 15 x 15, protože i tak malé hřiště nabízí až 155 117 520 různých her pro nejhorší případ. I zde byly všechny testy úspěšné, proběhlo jich 1 000 000.

Všechny tři algoritmy tedy byly řádně otestovány a testováním úspěšně prošly.

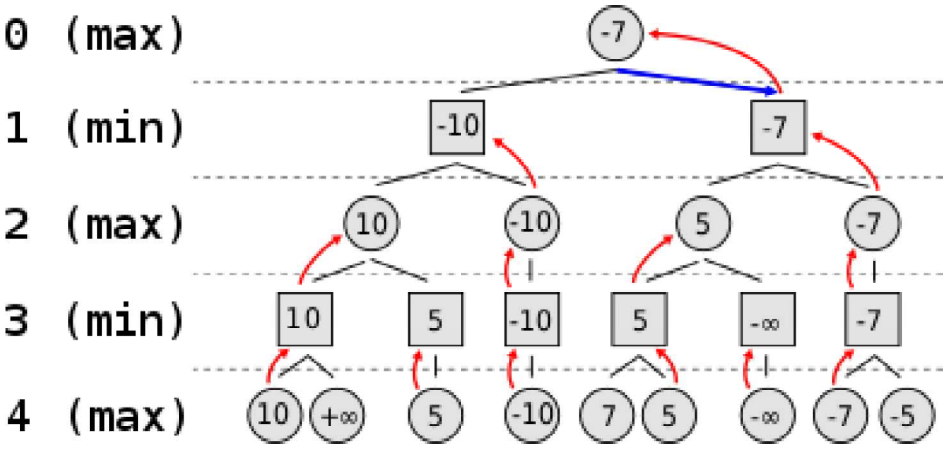
8. Rozbor řešení problémů z teorie her

Tato část práce se věnuje rozboru řešení složitých problémů z teorie her. U takových problémů již neznáme žádné polynomiální řešení problémů a musí se tedy, chceme-li problémy vyřešit, přistoupit k jiným metodám.

8.1 Řešení problémů s velmi velkým počtem možných řešení

Obrovské množství možných her se objeví například v šachách, kde se mluví až o 10^{120} možných hrách. [1] Pro představu, ve vesmíru se odhaduje celkem 10^{80} atomů. [2] Je tedy jasné, že šachy hrubou silou řešit nepůjde.

Jeden z možných postupů pro řešení takových her a například výše zmíněných šachů je prohledávání herního stromu do pouze omezené hloubky a využití minimax algoritmu. Tento způsob řešení vytvoří strom větvící se do určité hloubky a na koncových uzlech pak zhodnotí pozice (způsobem neprocházejícím všechny možnosti pak přihlíží, co nastane dál, jestli například hned v dalším tahu neprohraje apod.). Zpětně si pak rodiče těchto listů berou buď nejmenší, nebo největší z hodnot jejich dětí. To znamená, že pokud je na tahu hráč, pro nějž hledáme nejlepší pozice, zahraje tu nejlépe hodnocenou možnost. Pokud je ovšem na tahu jeho soupeř, předpokládáme, že chce vyhrát a zahraje nejlepší možnost pro sebe, tedy tu nejhorší pro prvního hráče. V tu chvíli tedy musíme pro tento uzel stromu vybrat možnost hodnocenou nejhůře. Tímto způsobem je pak algoritmus schopný rozhodnout, do které větve herního stromu je pro něj nejvýhodnější zahrát.



Obrázek 14 - minimax

Tento způsob funguje například v šachách, máme ale i složitější hry, kde ani takto optimalizovaným minimax algoritmem nedosáhneme dobrých výsledků. Jednou z těchto her je GO. Zde je složitost herního stromu až $2 * 10^{171}$. [3] Z každé pozice je pak možné vytvořit ohromné množství pozic jiných. Zde je třeba přijít s jiným řešením a na řadu přichází heuristika.

8.2 Heuristické řešení problémů z teorie her

Heuristika uplatňuje takové způsoby řešení, které sice nemají 100% garanci úspěšnosti, za to jsou ale výpočetně výrazně jednodušší. Využívají se, pokud je hledání přesného optima buď nemožné, nebo až příliš složité. Takové herní strategie fungují tak, že pracují a odvozují svá rozhodnutí z předchozích situací, kterými si prošly.

Za programem AlphaGo, který dokázal porazit nejlepší světové hráče ve hře GO, stojí takovéto heuristické algoritmy. [4] Konkrétně se podíváme na Monte Carlo Tree Search, který dokáže s použitím heuristických metod prohledat herní strom ve stejném čase do větší hloubky, než by bylo možné prohledáním všech možností.

8.2.1 Monte Carlo Tree Search

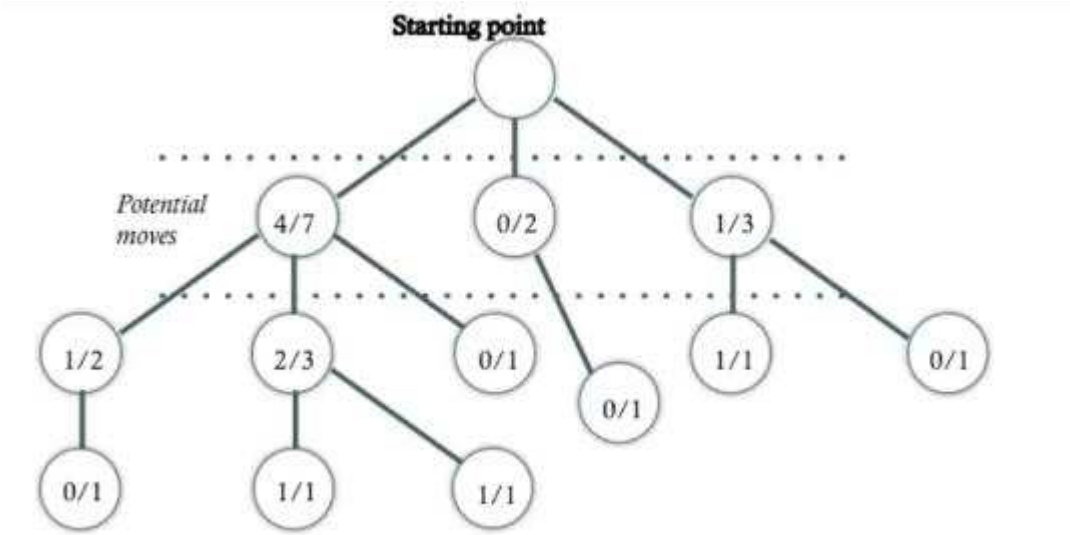
Tento algoritmus prohledává herní strom ve čtyřech opakujících se krocích.

Uvažme, že prohledáváme herní strom z bodu R. V tuto chvíli začneme jinak druhým krokem, a to Expanzí. Zahrajeme všechny tahy z bodu R. Nyní přijde na řadu další krok, který se jmenuje Simulace. Algoritmus nyní zahraje náhodně jednu hru ze všech expanzí objevených dětí. Poté co získá výsledek, přistoupí k dalšímu kroku, kterým je Aktualizace. Výsledek, ke kterému náhodným hraním došel, si připíše k testovanému dítěti bodu R, a to tak, že k počtu her simulovaných z bodu připočte jednu a k výsledkům simulovaných her připočte výsledek této hry (např. 5/11 by znamenalo, že z toho bodu bylo odehráno 11 her a v 5 z nich hráč rozhodující se z pozice R vyhrál).

Nyní vstoupí algoritmus do další iterace a začne tím, že si vybere jeden z listů stromu s přihlédnutím na to, který je nejslibnější a na to, které body navštěvoval jak často (tento krok se nazývá Selekce). Po vybrání bodu provede Expanzi. Poté opět simulaci a poté aktualizaci, tentokrát ale kromě bodu, ze kterého provedl expanzi, provede aktualizace i jeho rodiče, tedy bodu vybraného v selekci.

Tímto způsobem tedy algoritmus heuristicky určí, která herní cesta je nejslibnější, aniž by musel hrubou silou projít všechny možnosti.

Podívejme se na příklad stromu, který Monte Carlo Search Tree tvoří.



Obrázek 15 - Monte Carlo Search Tree

Jak je vidět, vždy odehraje z listů právě jednu hru a její výsledky pak zpětně propaguje jejich rodičům a rodičům rodičů atd.

9. Závěr

Podařilo se kompletně analyzovat úlohu a rozebrat všechny možnosti, které mohou nastat v rámci jejího řešení.

Vytvořil jsem algoritmy pro její řešení, dva polynomiální a jeden nepolynomiální. V rámci srovnání složitostí těchto algoritmů vyplynulo, proč jsou pro řešení na větší vstupy použitelné pouze ty dva polynomiální a naivní řešení hrubou silou nemá příliš velký význam.

Při odhadování složitostí jsem pak algoritmy analyzoval a popsal jejich škálování v závislosti na velikostech vstupů, a to z hlediska paměťového i časového. Ohledně stability jsem pak provedl řadu testů.

Pro testování polynomiálních algoritmů byly generovány vstupy náhodně a v různých rozsazích. Generoval jsem hry, jejichž velikost hřiště byla až 100 x 100, takové hry pro jejich jednoduchost tvořily až 40 % ze všech testů.

Dalších 25 % testů tvořily hry, jejichž velikost hřiště byla až 200 x 200, stále tedy relativně jednoduché hry.

S další sadou testů, konkrétně se hřišti až 300 x 300 jsem se již přiblížil reálnému zadání a nechal jsem opět proběhnout velké množství testů, tato sada tvořila 20 %.

Testů na reálných vstupech, tedy na hřištích až 400 x 400 bylo 15 % ze všech provedených testů, kvůli již vyšší časové náročnosti.

Celkem mluvíme o 2 000 000 testů při hlavním testování, které prokázaly funkčnost a stabilitu mých polynomiálnách algoritmů. Tyto algoritmy na menším počtu vstupů (100 000) vzájemně porovnal. Nepolynomiální řešení, jsem testoval na menších vstupech, konkrétně s hřištěm až 15 x 15. Zde jsem provedl celkem 1 000 000 testů (což je skoro tolik, kolik hřiště nabízí možných různých her) a i zde všechny prošly. Testování výsledků probíhalo mezi algoritmy vzájemně a s referenčním řešením úlohy.

V druhé části práce jsem pak rozebral řešení problémů, kde je možných řešení tolik, že ani pomocí optimalizací nejdou v polynomiálním čase projít všechny. V rámci toho jsem i vysvětlil, fungování a použití heuristiky, včetně rozboru konkrétního algoritmu.

10. Zdroje

https://en.wikipedia.org/wiki/Polynomial

https://en.wikipedia.org/wiki/Game_theory

https://en.wikipedia.org/wiki/Time_complexity

https://en.wikipedia.org/wiki/Asymptotic_computational_complexity

https://en.wikipedia.org/wiki/P_versus_NP_problem

https://cs.wikipedia.org/wiki/V%C4%9Bz%C5%88ovo_dilema

https://en.wikipedia.org/wiki/Big_O_notation

https://cs.wikipedia.org/wiki/Asymptotick%C3%A1_slo%C5%BEitost

http://i.imgur.com/L6uAKBD.png

https://adrianmejia.com/images/big-o-running-time-complexity.png

https://upload.wikimedia.org/wikipedia/commons/thumb/6/6f/Minimax.svg/400px-Minimax.svg.png

https://www.youtube.com/watch?v=Fbs4lnGLS8M

https://fiks.fit.cvut.cz/ulohy/rocnik-4/kolo-2

11. Seznam obrázků

12. Seznam citací

[1] Shannon number. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2019 [cit. 2019-03-18]. Dostupné z: https://en.wikipedia.org/wiki/Shannon_number

[2] Observable universe. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2019 [cit. 2019-03-18]. Dostupné z: https://en.wikipedia.org/wiki/Observable_universe

[3] GO (game). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2019 [cit. 2019-03-18]. Dostupné z: https://en.wikipedia.org/wiki/Go_(game)

[4] AlphaGo. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2019 [cit. 2019-03-18]. Dostupné z: https://en.wikipedia.org/wiki/AlphaGo

13. Seznam příloh

dynamic.py – algoritmus řešící úlohu pomocí dynamického programování

breadthFirst.py – algoritmus řešící úlohu prohledáváním do šířky

bruteforce.py – algoritmus řešící úlohu hrubou silou

checkResult.py – script pro ověření správnosti výsledků

generateGraph.py – script pro generování testovacích vstupů

reducer.py – knihovna pro optimalizaci a zjednodušení vstupů úlohy

Teleportace.cpp – referenční řešení úlohy

readme.pdf – návod k testování algoritmů