GYMNÁZIUM JÍROVCOVA

MATURITNÍ PRÁCE

Vizualizace významných algoritmů

Alexandr Bihun
vedoucí práce: Dr. rer. nat. Michal Kočer

V Českých Budějovicích 14. února 2024 **2023/2024**

Prohlášení

Prohlašuji, že jsem tuto práci vypracoval samostatně s vyznačením všech použitých pramenů.

V Českých Budějovicích dne .... podpis ....

Alexandr Bihun

Abstrakt

Tato maturitní práce se zaměřuje na vystětlení chodu známých algoritmů v oblasti pathfindingu (vyhledávání cest), rovněž jako na jejich analýzu a příblížení jejich využití v opravdovém světě. Dále bude naznačeno, jak jsem implementoval za pomoci knihovny Pygame v jazyce Python uživatelsky přívětivou aplikaci pro vizualici těchto algoritmů, která umožňuje uživatelům hlubší porozumnění a poskytuje skutečný vhled na fuknci těchto algoritmů. blablabla

Klíčová slova

algoritmy, analýza algoritmů, vyhledávání cest, grafy, vizualizace, python, pygame

Poděkování

Tady bude poděkování.

Obsah

Úvod

Přesto, že si to většina lidí nejspíše neuvědumuje, využívají algoritmy na denním pořádku. blabla... V této práci se zaměřím na algoritmy pro hledání cest blabla Cílem je vytvo

Část I

Představení a analýza vybraných algoritmů

1 Algoritmus

Samotné slovo algoritmus vzniklo zkomolením jména významného perského matematika, kterým byl Abu Jafar Muhammada ibn Mūsā al-Chwārizmi (asi 780-850 n. l.). Ten ve svých dílech položil základy algebry a způsobů řešení lineárních a kvadratických rovnic. Přeložením jeho spisu o indickém početním systému, ve kterém ukazuje, jak provádět základní početní operace, nabylo jeho jméno nového významu. Do latiny byl totiž přeložen jako *Algoritmi de Numero Indorum* (česky "Algoritmi o číslech od Indů"), kde slovo Algortimi je latinizovaná forma jeho jména. Tato forma jeho jména se pak začala používat jako označení různých matematických postupů. [8] [6] [5]

1.1 Definice algoritmu

Obecně se dá říci, že algoritmus je nějaká přesně daná posloupnost kroků, kterou lze dosáhnout kýženého výsledku. Tím pádem definici algoritmu splňují například recepty z kuchařek, návody na kosntrukci nábytku, pracovní postupy a podobně. [6]

Nejčastěji se ale s algoritmy setkáváme v kontextu matematické informatiky, kde popisují početní proceduru, kterou lze řešit konkrétní úlohy. Tyto algoritmy pak musí být schopné přijmout jakýkoli vstup popisující zadaný problém a vyřešit ho, tj. vyprodukovat korektní výstup. Zároveň musí být tak pečlivě a přesně zapsány, aby jim porozuměl počítač. K tomuto účelu slouží *programovací jazyky*, které se skládají ze slov s jasně danými významy. Spustitelný algortimus přepsaný ve vhodném programovacím jazyce nazýváme *program*. [3]

1.1.1 Vlastnosti algoritmu

Podle [7] a [8] od algoritmu požadujeme (většinou)[1] tyto vlastnosti:

1. Elementárnost - algoritmus sestává z konečného počtu jednoduchých kroků.

[1]Existují algoritmy, které např. generují pouze přibližné řešení.

2. Konečnost - algoritmus doběhne v konečném množství kroků.
3. Korektnost - algoritmus produkuje pro každý správný vstup korektní výsledek.
4. Obecnost - algoritmus řeší všechny instance daného problému. [2]
5. Determinovanost - každý krok vykonávání algoritmu je jednoznačně určený.

1.2 Ukázky jednoduchých algoritmů

Nejstarší dochované algoritmy se datují již do Sumerské říše, odkud pochází hliněná tabulka s prvním dochovaným algoritmem na dělení, její odhadované stáří činí 4500 let. V antickém Řecku vznikaly první algoritmy pro aritmetiku, jako například Euklidův algoritmus, či Eratosthenovo síto. [1]

1.2.1 Eratosthenovo síto

Tento algoritmus pro hledání prvočísel popsal poprvé řek Nikómachos z Gerasy, připisuje ho Eratosthenovy z Kyrény. Jeho algoritmus vygeneruje všechna prvočísla menší než nějaké číslo n podle jednoduché procedury. [1] Toto číslo n, podle kterého se odvíjí průběh algoritmu, označujeme jako vstup algoritmu.
Samotné kroky algoritmu pak jsou:

1. Vytvoř posloupnost čísel od 2 do n.
2. Vyber nejmenší dosud nevybrané číslo poslopnosti, nechť je prvočíslo.
3. Odstraň všechny násobky právě vybráného prvočísla.
4. Vrať se na krok 2, pokud si naposledy nevybral číslo větší než $\sqrt{n}$.
5. Na konci zůstanou v posloupnosti pouze prvočísla.

Tento algoritmus jsme právě popsali v prostém jazyce. Jak je vidět, je pochopitelný pro člověka a bezchybným provedením všech kroků dojde ke korektnímu výsledku. Mohli bychom ho stejně tak vyjádřit v *pseudokódu*, což je speciální druh jazyka, který připomíná běžné programovací jazyky. Pseudokód se však výhýbá implementačním detailům a konkrétním

[2] Instance problému je jeden konkrétní vstup pro tento problém.

standartům opravdových jazyků, zároveň je však tak přesný, aby šel s trochou snahy jednoduše převést do vhodného programovacího jazyka a hlavně vyjádřil myšlenku. [4]

1.2.2 Euklidův algoritmus

Euklidův algoritmus je dodnes používaný algoritmus pro nalezení největšího společného dělitele dvou přirozených čísel [4]. Jeho vyjádření v pseudokódu vypadá následovně:

Algoritmus 1: Euklides [4]

```
Vstup: x, y ∈ ℕ
a ← x, b ← y
Dokud můžeš dělej
    Pokud a < b pak
        prohoď a s b
    Pokud b = 0 pak
        vyskoč z cyklu
    a ← a mod b                ◁ mod značí zbytek po vydělení a hodnotou b
Výstup: Největší společný dělitel a = gcd(x, y)
```

V hlavičce je algoritmus pojmenovaný a očíslovaný v rámci celého dokumentu. Výraz $a \leftarrow x$ vyjadřuje vytvoření nové proměnné a (pokud do té doby neexistovala) a uložení hodnoty proměnné x do a. Svislé čáry značí bloky kódu, v bloku kódu se nejčastěji vyskytuje vnitřní logika cyklu, podmínky nebo funkce. Dále cokoliv za značkou $\triangleleft$ je komentář, čili text pouze pro vysvětlení samotného kódu.

Existují i jiné způsoby zápisu algoritmů jako např. grafický zápis flowchartem neboli vývojovým diagramem, či pomocí struktogramu [7]. V této práci budeme nadále používat pro popis složitějších algoritmů pouze pseudokód, pro jeho jednoduchost a zároveň přesnost.

2 Analýza algoritmů

Pro jeden problém obvykle existuje více algoritmů, které ho řeší. Abychom mohli porovnávat různé algoritmy mezi sebou, potřebujeme zavést nějaké metriky či veličiny, které nám budou popisovat jejich vlastnoti.

Pro nás nejdůležitějšími vlastnostmi algoritmu jsou jeho doba běhu a množství paměti potřebné pro jeho běh. Důvodem je, že samotná konečnost algoritmu není zárukou toho, že se po jeho spuštění dočkáme výsledku. Může se totiž stát, že instrukcí bude tak moc, že bychom se jejich zpracování a tudíž výsledku nemuseli vůbec dočkat.

Obdobně na dnešních počítačích nemáme neomezené množství výpočetní paměti, přestože trendem v této oblasti je neustálý růst, stejně jako u rychlosti výpočetních jednotek[1]. Proto musíme algoritmy optimalizovat i z tohoto hlediska. [8]

2.1 Časová a prostorová složitost

Časovou složitost algoritmu definujeme jako funkci f přiřazující každé velikosti vstupu počet elementárních instrukcí nutných pro vykonání algoritmu se vstupem této velikosti. Elementárními instrukcemi pak rozumíme aritmetické operace, porovnání apod. jednoduše to, co zvládne běžný procesor jednou nebo pár instrukcemi. Dále prohlásíme, že každá jedna instrukce trvá vždy konstantně času. Vstupů jedné velikosti bude obvykle více, proto vždy vybereme ten, který vyžaduje nejvíc instrukcí. Tím pádem bude funkce dávat počty instrukcí v nejhorším případě.

Prakticky to znamená, že si můžeme napsat algoritmus v pseudokódu a spočítat kolikrát se vykoná každá instrukce pro různě velké vstupy. Obvykle bude tato funkce rostoucí a nás nejvíce zajímá, jak rychle roste vzhledem k růstu velikosti vstupu. To znamená, že nás zajímá limitní chování funkce složitosti. Proto se zavádí takzvaná *asymptotická notace*.

Prostorová složitost je zavedena obdobně, s rozdílem, že místo počtu instrukcí určuje,

[1]Fenomén, že se přibližně každé dva roky zdojnásobí výkon nových počítačů, se někdy nazývá *Moorův zákon*.

kolik výpočetní paměti algoritmus potřebuje pro svůj běh v závisloti na velikosti vstupu. [4]

2.2 Asymptotická notace

Asymptotická notace je způsob, jak vyjádřit řád růstu funkce. Jejím úkolem je zjednodušit funkci složitosti algortimu s ohledem na to, že s dostatečně velkými vstupy bude rychlost růstu funkce určovat jen nejvýznamnější, tj. nejrychleji roustoucí člen. Toho docílí eliminací všech méně významných členů včetně konstant. Rozlišují se tři notace: $\mathcal{O}$-notace, Ω-notace, Θ-notace. [2]

2.2.1 $\mathcal{O}$-notace

$\mathcal{O}$-notace udává asymptotické omezení shora. Určuje, že funkce roste maximálně stejně rychle jako určitá míra.

Formálně definujeme, že funkce $f(n)$ náleží do třídy složitosti $\mathcal{O}(g(n))$, pokud existuje konstanta $c > 0$ a n_0 takové, že pro každé $n > n_0$ platí $f(n) \leq c \cdot g(n)$.

Situaci, kdy $f(n)$ náleží do $\mathcal{O}(g(n))$ značíme $f(n) = \mathcal{O}(g(n))$.

Pokud by např. funkce $2n^2 + 100n + 3000$ charakterizovala časovou složitost nějakého algortimu, zapíšeme skutečnost, že její řád růstu je n^2 následovně: $2n^2+100n+3000 = \mathcal{O}(n^2)$. Tvrdíme, že časová složitost takového algortimu je $\mathcal{O}(n^2)$. Je vidět, že $\mathcal{O}$ "seškrtne" všechny méně významné členy, rovněž jako konstanty[2] násobící všechny členy. Takto zavedená notace zjednodušuje porovnávání různých algoritmů mezi sebou.

$\mathcal{O}$-notace udává dobu běhu programu v nejhorším případě, tj. na asymptoticky nejsložitějším vstupu. Je možné, že existují i vstupy, pro které má algortimus lepší asymptotickou časovou složitost než $\mathcal{O}(g(n))$, které vyšlo pro nejhorší případ. Přesto, jelikož $\mathcal{O}$ omezuje ze shora, nebude tvrzení, že algoritmus má v každém případě složitost $\mathcal{O}(g(n))$ chybné. Uvažme, že funkce $h = n^2$ je nejen $\mathcal{O}(n^2)$, ale i $\mathcal{O}(n^3)$, obecně je $\mathcal{O}(n^c)$, pro $c \geq 2$.

Ω-notace a Θ-notace jsou zavedeny obdobně. Ω-notace udává asymptotické omezení zdola, tj. určuje funkce asypmtoticky roustoucí alespoň stejně rychle jako nějaká daná míra.

Θ-notace udává nejtěsnější mez, a to oboustrannou, tj. říká, že funkce roste stejně rychle jako daná míra. Pokud platí $f(n) = \mathcal{O}(g(n))$ a $f(n) = \Omega(g(n))$, pak platí $f(n) = \Theta(g(n))$.

[2] V praxi se může velká konstanta promítnout do doby běhu programu, proto se někdy zohledňuje, obvzlášť vybíráme-li mezi dvěma algortimy se stejnými asymptotickými složitostmi.

Ω značení pak používáme pro charakterizaci složitosti nejlepšího případu, $\Theta(g)$ se používá pro průměrný případ. Formální definice jsou uvedeny v [2].

3 Vsuvka z teorie grafů

Tohle zatím přeskočím, možná tohle ani nebude muset být samotná kapitola, ale jen podkapitola v další kapitole.

4 Algoritmy pro hledání cest

V této kapitole budou podrobně popsány a analyzovány některé algoritmy z oblasti pathfinding(heldání cest). Ty budou pak hlavím předmětem mého vizualizačního programu.

4.1 Prohledávání do hloubky

4.2 Prohledávání do šířky

4.3 Dijkstrův algoritmus

4.4 Uspořádáné vyhledávání

4.5 Algoritmus A*

Část II

Implementace vizualizačního programu

5 Plánování

5.1 Představení použitého software

6 Implementace

7 Ukázky využití

8 Výpisy použitých programů

Závěr

Tady bude závěr.

Bibliografie

1. CHABERT, Jean-Luc et al. *A History of Algorithms: From the Pebble to the Microchip*. Springer Berlin, Heidelberg, 1999. ISBN 9783540633693.
2. CORMEN, Thomas H.; STEIN, Clifford; RIVEST, Ronald L.; LEISERSON, Charles E. *Introduction to Algorithms*. Fourth edition. Cambridge, Massachusetts: The MIT press, 2022. ISBN 9780262046305.
3. DVORSKÝ, Jiří. *Algoritmy I*. 2007. Dostupné také z: `https://fei.znoj.cz/soubory/ALGI/skripta.pdf`. Verze ze dne 28. února 2007.
4. MAREŠ, Martin; VALLA, Tomáš. *Průvodce labyrintem algoritmů*. Druhé vydání. Praha: CZ.NIC, 2022. ISBN 978-80-88168-63-8.
5. MEHRI, Bahman. From Al-Khwarizmi to Algorithm. *Sharif University of Technology, Iran*. 2017.
6. NECKÁŘ, Jan. *Algoritmy*. © 2016. Dostupné také z: `https://www.algoritmy.net/`. (cit. 20. 1. 2024).
7. VIRIUS, Miroslav. *Základy algoritmizace*. Praha: ČVUT, 2008. ISBN 978-80-01-04003-4. Dostupné také z: `http://www.jaderny-prvak.8u.cz/wp-content/uploads/2013/02/Základy-algoritmizace-skripta-21.pdf`.
8. ČERNÝ, Jakub. *Základní grafové algoritmy*. MFF UK, 2010. Dostupné také z: `https://docplayer.cz/4802558-Zakladni-grafove-algoritmy.html`.

Seznam obrázků

Seznam tabulek

Přílohy

A Příloha s kódem