Algoritmy ve výuce na základní a střední škole

Algorithms in teaching at primary and secodary school

Bc. Adam Zluky

Diplomová práce
2010

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

*** nascannované zadání str. 1 ***

*** nascannované zadání str. 2 ***

ABSTRAKT

Tato práce se věnuje problematice algoritmů na úrovni žáků a studentů základní a střední školy. Rysy, principy a vlastnosti jsou představovány názornou formou. V textu nalezneme množství původních příkladů a přirovnání.

Téma třídících algoritmů doprovází vizualizace průchodu touž číselnou posloupností při použití různých algoritmů. Průběžně jsou představovány slabiny i silné stránky jednotlivých přístupů.

Poslední část se věnuje problému obchodního cestujícího. Obecněji pak problému P=NP. Popisu problému obchodního cestujícího předchází jemný úvod do teorie grafů.

Klíčová slova: algoritmus, výuka algoritmů, třídící algoritmy, problém obchodního cestujícího, teorie grafů

ABSTRACT

This thesis is deal with algorithms at the level of pupils and students of primary and secondary schools. Features, principles and characteristics are represented by visual means. In text we find many original examples and analogies.

Sorting algorithms topic accompanies the passage of same numeric sequence using different algorithms. There are clear strengths and weaknesses of different approaches.

In the last section we can find the traveling salesman problem. More generally, the P=NP problem. Description of the traveling salesman problem precedes gentle introductions to graph theory.

Keywords: algorithm, teaching algorithms, sorting algorithms, traveling salesman problem, graph theory

„Takovou knihu bych si býval přál číst, když jsem se poprvé začal počítači zabývat. Na rozdíl od většiny ostatních knih o počítačích - které jsou buď o tom, jak je používat, nebo o technologii, ze které jsou tvořeny (ROM, RAM, diskové jednotky atd.) - tohle je kniha o myšlenkách.„

Willian Daniel Hillis, Vzor v kameni.

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu
  využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně

…….
podpis diplomanta

OBSAH

ÚVOD

Osnovy pro předmět informatika, výpočetní technika, práce s počítačem, informační a komunikační technologie či různé jiné názvy jednotlivých škol můžeme rozdělit do několika kategorií, které se vyskytují téměř všude. Bez výjimky je do hodin, ať již teoreticky či prakticky, zasazena část věnující se kancelářskému balíku od Microsoftu, eventuálně volně dostupné alternativě OpenOffice.org.

Práce s textovým procesorem, tabulkovým kalkulátorem či prostředím pro tvorbu prezentací je vděčné téma. Spatřuji několik faktorů, proč je právě tato látka tak častá. Jednak je to dané dostupností pro základní i střední školy. Dále jasným výstupem a postupem. V neposlední řadě také vysokou mírou uplatnění. Většina kapitol, jako styly a formátování, generování obsahu, grafů a podobně lze využít napříč předměty během celého školního roku.

Pakliže budu nadále hovořit o středních školách v celé jejich šíři, můžeme se setkat i dalšími typickými bloky. Základy ovládání programů pro práci s audiem, videm, či grafikou, práce s internetem, princip tvorby webových stránek, hardware, software, netware.

Většina žáků a studentů se ve škole nesetká s teoretickými základy informatiky, byť v jejich primitivní formě.

Existují dva základní přístupy k výuce informatiky. Zjednodušeně je můžeme nazvat odshora dolů a odzdola nahoru. Princip odshora dolu si dovolím představit v následujícím příkladě.

Se studenty budu probírat tvorbu webových stránek. Představím jim nějaký hotový projekt, jednoduchými modifikacemi kódu si budeme ukazovat změny ve struktuře, či formátování stránky. V určitém časovém horizontu se dopracujeme ke všem použitým deklaracím. Opačný postup je použit v případě, že prvních několik hodin si budeme představovat konkrétní element jeho způsob použití a možné úpravy. Postupným rozšiřováním portfolia dospějeme ke stadiu jednoduché stránky a a po konečném počtu kroků teoreticky dospějeme do stejného cíle jako při použití první metody.

Jistě se naleznou argumenty pro použití obou metod. Můj pohled na volbu postupu je poměrně jednoznačně daný vztahem žáka k tématu s pohledem na budoucí použití.

Pokud budeme předpokládat, že myšlenky na předmět opouští studenty s prvními kroky mimo učebnu zvolím postup odshora dolů, poněvadž je pro ně efektivnější. V případě, že jsem si jistý, nebo alespoň já, či student máme ten pocit, že se oboru chce věnovat i nadále, budeme látku probírat odzdola nahoru. Důsledky, které z toho vyplývají jsou myslím srozumitelné bez podrobnějšího popisu.

Moje diplomová práce je příručka pro výuku informatiky odzdola nahoru, tedy pro žáky a studenty, u kterých je předpoklad, že se informatice, potažmo výpočetní technice budou věnovat i nadále.

Pro četbu, případně studium této práce nejsou zapotřebí vysoké vstupní znalosti z oblasti počítačů.

I. TEORETICKÁ ČÁST

1 OBECNĚ O ALGORITMECH

1.1 Inspirace.

Inspirací pro tuto práci mi byla kniha Daniela Hillise Vzor v kameni [1]. Cituji: „*Takovou knihu bych si býval přál číst, když jsem se poprvé začal počítači zabývat. Na rozdíl od většiny ostatních knih o počítačích - které jsou buď o tom, jak je používat, nebo o technologii, ze které jsou tvořeny (ROM, RAM, diskové jednotky atd.) - tohle je kniha o myšlenkách.*„ Stejný vztah mám i já k této své práci. Přál bych si jí přečíst před tím, než jsem se začal věnovat studiu informatiky. Při pochopení toho, co je algoritmus, jaké má vlastnosti, které z nich jsou žádoucí a které ne, se nám bude snadněji psát první program. Mnohem lépe však ten druhý a třetí, pokud prvním rozumíme Hallo World[1].

1.2 Algoritmus

Nechci zde vysvětlovat původ a lexikální význam slova algoritmus. Ten lze v odborných publikacích či na internetu nalézt bez větších potíží. Pro naše potřeby bude stačit tvrzení, že algoritmus je návod.

Návod, či postup, který transformuje vstupní data na výstupní. Typickou ukázkou v literatuře bývá kuchařský recept. Vstupní data jsou ingredience, algoritmem rozumíme práci s jednotlivými přísadami a výstupem je hotový pokrm.

1.3 Zápis algoritmu

Algoritmus lze zapsat několika způsoby a nelze jednoznačně říci, která volba je univerzálně nejlepší. Jak je tomu v počítačové vědě často, nejvhodnější bývá kombinace několika postupů.

Pro porovnání jednotlivých přístupů si uvedeme jednoduchý příklad fiktivního automatu na jízdenky.

1 Zpravidla první program v daném jazyce. Nemá širší využití. Jediné co dělá je zobrazení nápisu Hello World.

1.3.1 Přirozený jazyk

Ze zápisu přirozeným jazykem jsou snadno viditelné jeho limity. Jednak je to redundance znaků v textu, dále omezení daná syntaktickou a sémantickou stránkou jazyka a ve složitějších případech vysoká míra nepřehlednosti.

Pokud zákazník vhodil více peněz než je cena jízdenky, jízdenka se vytiskne a vrátíme zákazníkovi rozdíl z ceny jízdenky a z vhozené částky.

1.3.2 Pseudokód

Pseudokód jakýsi mezičlánek mezi přirozenou řečí a programovacím jazykem. Je nezávislý na platformě, měl by být snadno přenositelný mezi jednotlivými jazyky, jak procedurálními, tak neprocedurálními. Používá některé obecně známé programátorské nástroje jako jsou podmínky, cykly typu while a for.

```
If vhozeno vetsi nez cena
print jizdenka
vratit vhozeno-cena
```

1.3.3 Programovací jazyk

Zápis algoritmu programovacím jazykem je pro běžného člověka nečitelný, na druhou stranu běžný člověk nepotřebuje znát princip, stačí mu výsledky. Často se volí právě tento postup, neboť pouze takto si můžeme uvěřit správnost zápisu překladem a spuštěním programu. Nadto pokud není zápis v nějakém exotickém programovacím jazyku je přenositelný pouhou změnou syntaxe. Ukázka kódu je v jazyce Java.

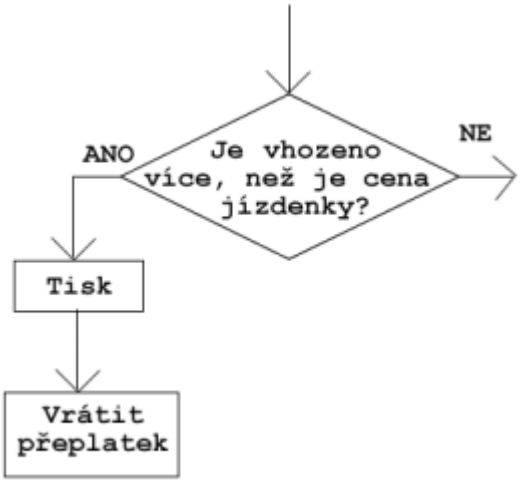
```
Public int vhozeno;
public int cenaJizdenky;
public int vratit;

Public void tisk()
{if cenaJizdenky <= vhozeno}
(
      systém.out.println(„Jizdenka“);
      vratit = vhozeno-cenaJizdenky;
)
```

1.3.4 Vývojový diagram

Vývojový diagram je silný nástroj. Často je čitelný i pro neprogramátory, protože používá obecně platné konvence. Průběh algoritmem je intuitivní a použitelný i v těch případech, kdy přirozený jazyk selhává.

Diagramy upravuje Česká norma ISO 5807 z roku 1996 [2]. Dle ní jsou předdefinované geometrické tvary pro zpracování, rozhodování, vstupní a výstupní data, komentáře a jiné.

Princip vývojových diagramů, jakožto názorného předvedení struktury využívají četné aplikace a vývojová prostředí. Existuje také modelovací jazyk UML, který je určen na vizualizaci struktury programů a softwarových systémů.



Obr. 1: Vývojový diagram

1.4 Vlastnosti algoritmu

Aby mohl být postup, či návod nazván algoritmem, musí splňovat některé podmínky. Autoři jich obvykle vypočítávají pět, lze vypozorovat odlišné pojmenování, případně překrývání významu, avšak v základních principech se shodují.

Na začátku jsme si řekli, že algoritmem můžeme rozumět i kuchařský recept. Po přečtení následující kapitoly zjistíme, že tomu tak ve skutečnosti není. Recept na guláš

obsahuje věty jako: „přidáme na kostky nakrájené maso“ nebo „Směs vmícháváme do guláše do té doby než se nám bude zdát, že je dost hustý.“

Pokyn na přidání masa nakrájeného na kostky může být splněn kostkami různé velikosti, na druhou stranu když bude maso na nakrájené na kvádříky či hranůlky, i nepravidelných tvarů na výstup, tj. hotový guláš to nebude mít zásadní vliv. Předpokládám, že nedostatky druhé citace není ani potřeba nijak podrobněji rozebírat.

1.4.1 Konečnost

Postup lze nazvat algoritmem, pokud po konečném počtu kroků dopěje ke konci. Jinými slovy vrátí odpovídající výstup. Počet elementárních kroků průběhu algoritmu může být libovolně velký a k jeho měření se a porovnávání slouží hodnota nazývaná složitost. Té se věnujeme v samostatné kapitole.

S pojmem konečnost úzce souvisí heuristika. V jistém smyslu lze heuristický postup chápat jako protiklad algoritmu, neboť při jeho použití víme, že výstup nebude přesný. Výstupem bude pouze odhad, ten se bude mimo jiné lišit počtem použitých kroků. Existují však, a to nejen v počítačovém světě problémy a situace, kdy je lépe znát nepřesný výsledek, neboť k přesnému bychom při použití algoritmu nejspíše nedospěli, nebo dokonce takový algoritmus ani nemusí existovat. Klasickou ukázkou je *problém obchodního cestujícího*.

S negativní stránkou heuristických analýz, to je pouze s odhadem správného výstupu, se mnozí z nás setkali, aniž by si to uvědomili. Spamový filtr použitý v emailových schránkách, nebo desktopových antivirových programech třídí poštu na základě heuristické analýzy [3]. Proto se může stát, a souvisí to především s programovým řešením a citlivostí nastavení, že ve složce spam skončí i zpráva, která tam nepatří. Stále je to však lepší stav, než když bychom takový nástroj neměli. Troufám si tvrdit, že algoritmus, který bezpečně pozná každý spam a zároveň bezchybně určí, co spam není, neexistuje. Nadto, když jej často na první pohled nerozpozná ani člověk. Dnes je řada emailů vygenerována roboty a rozhodně nejsou nevyžádané. S nadsázkou lze říci slovy klasika „*Nemusí pršet, jen když kape.*“

1.4.2 Hromadnost

Algoritmus musí řešit problém obecně. Chybný postup by byl na příkladě algoritmu pro hru pexeso tento: *Pokud v prvním tahu otočíte míč a v druhém také, získáváte jeden bod.* Správně samozřejmě musí být: *Pokud se dva obrázky shodují, máte jeden bod.*

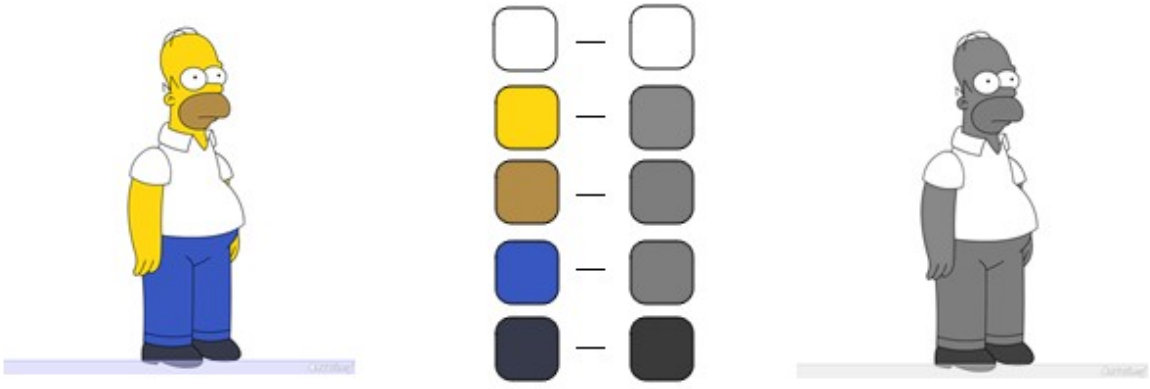
Chybou je také, když správně (korektně a s nízkou časovou složitostí) pracuje pouze při vhodné konstelaci vstupních dat. Takovýto algoritmus není dostatečně obecný, v našem případě nesplňuje podmínku hromadnosti.

Jiným příkladem však mohou být algoritmy speciální, či specializované. Kupříkladu *bublinkové třídění* je vhodné použít, pokud jsou vstupní data téměř seřazena, tj. přibližně 80% prvků je na správné pozici.

1.4.3 Určitost

Celý postup algoritmem je rozdělen na několik elementárních kroků. Každý krok musí být přesně určen. Následující příklad ukazuje princip filtrů používaných v programech pro vektorovou grafiku. Náš jednoduchý filtr převede barevný obrázek [4] na obrázek zobrazený ve stupních šedi, lidově černobílý. Algoritmus říká, vezmi si jeden pixel, na základě RGB složky najdi jeho ekvivalent ve stupních šedi, nahraď jej. Toto udělej pro každý pixel.

Při opakovaném použití jednoduchých kroků, dosáhneme dosáhneme komplexní změny celku. V naší ukázce nahrazení obrázku jeho ekvivalentem v odstínech šedi.



Obr. 2: Použití elementárních kroků na celek

1.4.4 Další

Donald Knuth [5] například ve své knize The Art of Computer Programming jako vlastnost uvádí i vstup a výstup. Samozřejmě, že bez vstupních a výstupních dat se těžko provádí testování nějaké skupiny úloh, nicméně já tyto dva pojmy nepovažuji za *vlastnost*, nýbrž za *součást*. Z mého pohledu spatřuji analogii v přirovnání: auto má ty *vlastnosti*, že umí jezdit dopředu, dozadu a zatáčet. Čtyři kola pak nejsou *vlastnosti* automobilu, avšak jeho *součástí*.

1.5 Složitost

Jako stěžejní ukazatel efektivity algoritmu se uvádí složitost. Tato se rozděluje podle dvojího pohledu. Složitost časová a paměťová jsou víceméně spojené nádoby, nemají spolu však přímo úměrný vztah.

Počítačová věda obecně pracuje s velkým rozsahem jednotek, často až na hranici představivosti běžného člověka. A to jak v mocninách kladných či záporných. Stačí se podívat do poštovní schránky. Leták na připojení rychlostí 10Mbps, chápejme tak, že jsme schopni přijmout deset miliónů jednotek informace, reprezentovaných nějakou fyzikální veličinou za jednu sekundu ve směru z internetu k nám. Druhá propagační tiskovina nám za akční cenu nabízí notebook s frekvencí procesoru 1,3 GHz. Tento počítač bude schopen udělat jednu elementární operaci za 1,3 miliardtinu vteřiny.

Mexapixely a *gigabajty* používají v běžném hovoru i lidé, kteří jejich význam pouze tuší. Leckdy jejich bezděčná záměna znalého člověka pobaví. Pravdou je, že si v rozhazování miliónů a miliard kolem sebe informatika a výpočetní technika v ničím nezadá s ekonomikou.

Při studiu algoritmů můžeme narazit na neschopnost počítačů daný úkol vyřešit. Ne ve smyslu, že počítač nepochopí, co má dělat, implementace v programovacím jazyku může být bezchybná, limituje nás však technologie.

1.5.1 Paměťová složitost

Vedle velikosti paměti, ať už pevného či jiného disku a frekvence procesoru je třetím základním ukazatelem *výpočetní síly* každého počítače velikost jeho operační paměti. Nejen v informatice platí, celek je tak silný, jak je silný jeho nejslabší článek. Operační

paměť, je objem dat, se kterým je schopen stroj pracovat. Logický argument proti tomuto omezení může znít, že počítače jsou čím dál rychlejší a to, co zpracovávaly před dvaceti lety je téměř nesouměřitelné s dnešními úlohami.

V souvislosti s paměťovou složitostí nelze nezmínit slavnou předpověď spoluzakladatele společnosti Intel Gordona Moora, známou jako Moorův zákon [6]. Moore v roce 1965 řekl, že počet tranzistorů na čipu se zdvojnásobí každé dva roky. Dlužno říci, že takto přesná předpověď, nadto v dynamicky se rozvíjející počítačové vědě, hraničí s výrokem Nostradama.

Roku 1995 prof. Niklaus Wirth, držitel prestižní Fellow Award [7] reagoval na Moorův zákon ironickým výrokem: „*Software gets slower faster than hardware gets faster.*“ Podstatou jeho myšlenky je, že ačkoliv hadware je stále rychlejší, spolu s ním roste i náročnost aplikací, a tím se celkový vývoj zpomaluje.

1.5.2 Časová složitost

Určující vlastností algoritmu je jeho časová složitost. Označuje se velký písmenem řecké abecedy Θ – Théta, nebo tak zvanou O notací.

Časová složitost je nezávislá na platformě, hardwaru či použitém programovacím jazyku. Je zřejmé, že implementace téhož algoritmu pro stejná vstupní data poběží řádově jinou dobu na dnešním vícejádrovém procesoru, než na Commodore 64. Složitost je však stále tatáž.

Na časovou složitost lze pohlížet z několika směrů. Pohled pesimistický, průměrný a optimistický. Zpravidla se v našich příkladech budeme zabývat pouze složitostí průměrnou. V případech kdy se implementují algoritmy do lékařského, leteckého či vojenského prostředí je nutné počítat s pesimistickou složitostí, neboť pouhá špatná konstelace vstupních dat při jinak správném algoritmu by mohla mít fatální následky.

Pro jednoduché případy si lze složitost představit jako graf funkce.

1.5.3 Lineární složitost

Dva chlapci Adam a Bohouš se chtějí dostat na druhou stranu města. Můžou jít buď pěšky nebo jet městskou hromadnou dopravou. Adam jde pěšky, délku mezi dvěma zastávkami ujde přibližně za pět minut. Bohouš se rozhodne jet autobusem, ten ale jede až

za dvanáct minut. V době, kdy Bohouš nastupuje do autobusu, již má za sebou Adam dvě zastávky pěší chůzí. Bohouš však po několika minutách dotahuje náskok a pak už bude všude jenom dříve. Na konečnou, v našem případě šestou zastávku se dostane za šest minut, Adamovi to bude trvat půl hodiny.

Obr. 3: Lineární složitost

V čem tento příběh souvisí s algoritmickou složitostí?

Představme si, že je zvolený způsob dopravy algoritmus. Pro nízká vstupní data, v našem případě pouze tři zastávky je jedno zda půjdeme pěšky nebo autobusem. V obou případech nám to zabere patnáct minut. Pro vysoký vstup, tedy pokud chceme jet na šestou, dvanáctou nebo třicátou zastávku, je výrazně jednodušší jet autobusem.

Z hlediska časové složitosti jsou ale oba algoritmy stejné. Křivka A má tvar funkce

$$y=x,$$

křivka B

$$y=2x-1.$$

Na tomto příkladu jsme si ukážeme jedno z pravidel pro určení časové složitosti algoritmu. Je jím zanedbání konstant. Obě funkce jsou přímky a mají tedy lineární složitost O(n).

Čím dál pojedeme, tím déle nám to bude trvat.

1.5.4 Kvadratická složitost

Při ukázce kvadratické složitosti již nám příklad s dopravou pokulhává, protože bychom museli mít takový dopravní prostředek, který čím jede déle, tím jede rychleji.

Nabízí se však podobnost s fenoménem, se kterým se setkalo mnoho lidí, kteří se na internetu pohybují. Mám na mysli takzvané sociální sítě.

Představme si fiktivní síť, v níž každý člen má na začátku dva kamarády. Za kamaráda je myšlen i kamarád kamaráda, kterého de facto ani nemusím znát. Pakliže do okruhu svých blízkých vpustím nového člena, získám členy dva. Se vzrůstající základnou přátel první generace mi bude kvadraticky vzrůstat počet přátel druhé generace, tím mám na mysli přátele přátel.

Algoritmus s kvadratickou složitostí bude při dvojnásobném počtu vstupních dat pracovat čtyřikrát déle.

1.5.5 Logaritmická složitost

Řada třídících algoritmů, které se představíme v další části textu pracuje s logaritmickou složitostí. Graf logaritmické funkce napovídá, že při několikanásobném zvýšení počtu vstupních dat se délka běhu programu - počet elementárních kroků - zvýší nejméně z uvedených složitostí.

Populární třídící algoritmus *quicksort* pracuje se složitostí n·log(n). Ta je ideální. Na začátku kapitoly jsme si řekli, že se budeme věnovat složitosti průměrné, existují totiž případy, kdy jindy velice rychlý algoritmus *quicksort* pracuje s kvadratickou složitostí. Tím případem je pole čísel setříděné podle přesně opačného klíče, tj. sestupně.

Pro představu rychlosti takového algoritmu nám poslouží metoda půlení intervalu, u které se obecně uvádí, že má logarimtickou složitost. Běžně tuto metodu používáme při vyhledávání v knihách. Ať konkrétní stránky, nebo konkrétního hesla v abecedně seřazeném seznamu. V tištěném slovníku cizích slov hledám například pojem *precedens*. Otevřu jej náhodně zhruba uprostřed, na písmenu M. Dále mě již zajímá pouze druhá polovina knihy. V jejím přibližném středu jsou slova na písmeno S, dále budu pracovat pouze s první polovinou aktuální části. Pravdou je, že tento algoritmus bude s logaritmickou, popř. lineárně logaritmickou složitostí použitelný pouze v tom případě, že známe správné pořadí písmen v abecedě.

1.5.6 Další typy složitostí

Konstantní složitost. Algoritmus s konstantní složitostí proběhne při libovolné délce a seřazení vstupních dat za jeden elementární krok. Příkladem může být výběr *prvního prvku* ze seznamu nebo výběr *i-tého prvku* z indexovaného pole.

Kubická, polynomická složitost. Pokud jsme si řekli, že při kvadratické složitosti se při dvojnásobném vstupu zečtyřnásobí časová složitost, při kubické bude vyšší osmkrát. Při polynomické n-krát.

Exponenciální a faktoriálové složitosti jsou již v praxi nepoužitelné. Při použití pouhých osmnácti prvků by měl algoritmus s exponenciální složitostí dvěstěšedesáttisíc kroků, s faktoriálovou by to bylo číslo šest s patnácti nulami kroků. Bohužel však existují problémy jež umíme řešit pouze algoritmy s exponencionální složitostí. Je jím opět výše zmiňovaný *problém obchodního cestujícího*, kterému věnujeme samostatnou kapitolu.

1.5.7 Složitost v reálném světě.

Závěrem této části si ukážeme tabulku, která velice výmluvně ukazuje, jak je analýza a určení složitosti algoritmu důležitá a dokonce i Moorova teze o stále se zrychlujících počítačích je v některých případech bezzubá.

Dlužno říci, že tato nebo velmi podobná tabulka se vyskytuje ve většině přednášek či vysokoškolských skript uvádějících studenty do problematiky algortimů. Namátkou Ing. Ladislav Beránek, CSc., MBA [8] z Jihočeské Univerzity nebo RNDr. Arnošt Večerka [9] z Univerzity Palackého.

Notebook, na kterém píši tuto práci má procesor Intel Atom s frekvencí 1,60 GHz. Znamená to tedy, že za jednu vteřinu je schopen zpracovat 1 600 000 instrukcí. Jinými slova jedna instrukce mu trvá 0.6 μs. Tabulka ukazuje jak dlouho by tento procesor zpracovával algoritmus s danou složitostí (řádek) pro daný počet prvků (sloupec).

Složitost\Počet instrukcí	***10***	***20***	***50***	***100***	***1000***
1	1	1	1	1	1
n	6μs	12μs	30μs	60μs	0.6ms
log (n)	0.7μs	1μs	1.4μs	1.8μs	2.8μs
n log(n)	4.7μs	13μs	44μs	0.1ms	1.6ms
n^2	36μs	0.14ms	0.9ms	3.6ms	0.36s
n^3	0.2ms	1.7ms	27ms	0.2s	3,6min
2^n	64μs	4ms	18min	36 558 let	
n!	0.7ms	8min	$8 \cdot 10^{18}$ let		

II. PRAKTICKÁ ČÁST

2 TŘÍDĚNÍ

Z obsáhlého rádiu algoritmických úloh a problémů jsem si vybral třídění. Ačkoliv anglický název pro tuto činnost je *sorting* a i v češtině se algoritmy, které si představíme v následující kapitole nazývají *třídící* věřím, že vhodnější slovo je *seřazovací*. Třídění je dle mého činnost, kdy prvek vkládáme do skupiny, s jejímiž prvky ho pojí stejné vlastnosti. Střepy z rozbitého okna patří do stejného zeleného kontejneru jako láhev od vína. Naopak polystyren patří do skupiny žluté. A takové zrcadlo nesmí přijít ani do jednoho ze známých kontejnerů, čili skupina ostatní. Tato činnost je správně nazývána tříděním. Pokud ale uspořádáváme prvky v některé struktuře, žáky podle prospěchu, města podle počtu obyvatel, soubory podle abecedy, provádíme činnost seřazovací.

Lingvistickou nesrovnalost úvodem zastíní přednosti algoritmických úloh spojených s tříděním. Není to ryze akademické téma, s jejími principy lze pracovat i s nízkou měrou abstrakce, řadu postupů v běžném světě užíváme, aniž bychom si to uvědomovali.

2.1 Než začneme třídit

Třídění je činnost, kde se svět výpočetní techniky dotýká s reálným v přesně žádané míře, jaká je u výuky informatiky nutná. I nad tak zdánlivě triviální činností, jakou je setřídění žolíkových karet v ruce, se dá vystavět celé jedno vědecké odvětví.

2.1.1 Příklad první, karty

Předpokládám, že většina hráčů žolíků a kanasty si karty v ruce srovnává podle barev. Tento předpoklad je čistě subjektivní a nahlížením do cizích karet jsem si ho neověřoval. V každé této podskupině si pak srovnávají karty podle velikosti. Tedy od dvojky do desítky a tak dále ve standardním pořadí. Eso pak umístím na začátek nebo na konec podle toho, zda mám k dispozici spíše dvojku a trojku, nebo krále a královnu.

K zadanému cíli, tak jak jsem jej popsal vede pochopitelně několik cest. Zmíním se o dvou, se kterými je možné se u stolu setkat. První je intuitivní. Vezmu z balíku první kartu, druhou umístím před ní nebo za ní. Třetí pak před obě, za obě, nebo mezi ně. Tímto způsobem mám v ruce stále setříděné karty podle klíče, který jsem si zvolil na začátku.

Jiný způsob je, vezmu karty tak, jak jsem je dostal a rozvinu je do vějíře. Až poté několikerým přesunem karet, buďto té, které mi někde překáží, nebo naopak jinde chybí docílím přehledného rozvržení.

Zkusme si představit výhody a nevýhody těchto dvou postupů, lépe řečeno možná vlastnosti. První postup má složitost *N*. Pokud jednu kartu vkládám právě jednu vteřinu a budeme hrát žolíky, v jakýchkoliv kombinacích nebudu mít karty seřazené dříve než za dvanáct vteřin.

Druhý způsob, který jsme si představili, a to ten, že si karty nejprve prohlédneme a analyzujeme může být buďto výrazně rychlejší nebo pomalejší. Optimistický pohled říká, že pokud by se mi na stůl dostaly karty setříděné nebo téměř setříděné, mohl bych je mít v ruce srovnané za vteřinu, za dvě nebo za tři. Opět za předpokladu, že za jednu vteřinu kartu z vějíře vyjmu a umístím na patřičné místo.

Co to má společné s počítači, potažmo algoritmy? Existují úlohy, u nichž je určitá pravděpodobnost, že na vstupu dostaneme již setříděnou, nebo téměř setříděnou řadu prvků. Potom jsme schopni výběrem vhodného algoritmu vypočítat úlohu ze zlomek času, než když bychom použili jiný. Vzpomeňme také na kapitolu o složitostech a pesimistických či průměrných odhadech.

Tak jak jsme si nyní na několika řádcích ukázali funguje algoritmus *insertsort*, metoda přímým vkládáním.

2.1.2 Příklad druhý, tělocvik

V hodinách tělesné výchovy je zvykem nechat žáky nastoupit vedle sebe seřazené podle velikosti. Jelikož já, jako učitel nevím, kolik který žák měří a často to neví ani oni sami, nechám je nastoupit do řady. Nejspíše se nestane, že by takto náhodně utvořená řada byla setříděná podle velikosti. Jsem však schopen porovnat, který ze dvou vedle stojících je vyšší. Vytvořím proto takovouto sadu pravidel.

```
1)Vezmu si první dva žáky vlevo.
2)Porovnám jejich výšku.
3)Pokud vpravo stojící je vyšší, prohodím jejich pozice. Vezmu
si další aktuální dvojici a vracím se do bodu 2).
4)Pokud jsem vyzkoušel poslední dvojici vracím se do bodu 1)
5)Pokud jsem při porovnání všech dvojic ani jednou neprohodil
jejich pořadí. Jsou seřazeni podle velikost.
```

Při prvním návrhu tohoto algoritmu jsem se domníval, že stačí pouze první tři body. Ale takto nastavená pravidla by plnila úkol typu *Najdi nejvyšší/nejnižší číslo*, v našem případě nejnižšího chlapce a toho umísti na konec řady. Proto je nutné průchod celou řadu opakovat. Jinými slovy lze říci, že při prvním průchodu nalezneme nejnižšího chlapce umístíme jej na odpovídající pozici, tj. na konec řady. Při druhém průchodu najdeme druhého nejnižšího a umístíme jej jako druhého od konce.

I tento algoritmus má svůj název. *Bubblesort*, neboli bublinkové třídění bývá v textech o třídících algoritmech často zmiňován mezi první. Právě pro jeho jednoduchý a pochopitelný princip.

2.2 Řadící algoritmy podrobněji

V odborné literatuře i na internetu lze dohledat kolem deseti známých řadících algoritmů. Platí mezi nimi nepřímá úměra ve tvaru, čím rychlejší, tím obtížnější na výklad, případně naprogramování a naopak. Některé z dále uvedených jsou více akademické a spjaté s matematikou a diskrétní matematikou. Jiné bychom mohli pojmenovat jako nepraktické, avšak snadněji vysvětlitelné, tedy didaktické. A poslední skupinu bych nazval praktickou, ve smyslu využívání v praxi. Do té se často řadí kombinace různých přístupů.

Jak lze odtušit z principu této publikace, mám zájem věnovat se ponejvíce algoritmům, které jsem si nazval jako didaktické. Na závěr kapitoly mohu přislíbit odlehčení ve formě několika vtipných algoritmů.

2.2.1 Další vlastnosti řadících algoritmů.

Kromě již zmíněné složitosti se u řadících algoritmů určují dvě vlastnosti, a to přirozenost a stabilita.

Doc. Virius [10] říká, že algoritmus je stabilní pokud prvky se stejnou hodnotou klíče budou v cílové posloupnosti uloženy ve stejném pořadí jako v posloupnosti zdrojové. V knize Roberta Sedgewicka [11] se zase lze dočíst, že třídící metodu lze označit za stabilní, jestliže zachovává relativní uspořádání duplicitních klíčů souboru. Jinými slovy, pokud algoritmus při zpracovávání řady prohodí dvě *stejná* čísla, stabilní není.

Přirozenost je obecně vnímána jako pozitivní vlastnost. U algoritmů tomu není jinak. Přirozený algoritmus rychleji zpracuje částečně seřazené řadu. I když si člověk intuitivně říká, že přirozený algoritmus bude rychlejší ostatních, není to vždy pravda.

Při uvedení algoritmu do praxe je nutné zaměřit se ještě na jednu důležitou vlastnost. A tou je paměťová náročnost. Většina algoritmů se snaží pracovat s množinou na místě, tedy není potřeba další paměti, než ve které jsou uložena data, a pokud ano tak pouze velmi málo. Vyjádřeno vzorcem, pokud je počet prvků n, vystačíme s pamětí

$$n + c,$$

kde c je velmi malá konstanta.

Jak už tomu bývá v různých oborech a informatice zejména, k jednomu cíli vede několik cest. A samozřejmě, že existují řadící algoritmy, které polovinu paměti spotřebují na data a druhou na jejich zpracování.

V následující části, v níž si představíme, jak algoritmy pracují uvnitř i navenek, si ukážeme, že se skládají z elementárních kroků. To samozřejmě odpovídá úvodu práce, v němž je tato vlastnost zmíněná jako podmínka algoritmu. Vnitřní činnosti při seřazování prvků je jejich porovnání a přesun. Přesun prvků je zpravidla více paměťově náročný než porovnání, proto jedna z cest pro vylepšení třídícího algoritmů bude sledování počtu přesunů a porovnání. A samozřejmě snižování přesunů na úkor porovnání.

Záměrně je vynechána formule o porovnávání klíčů. Klíč je součást prvku, který chceme třídit a podle kterého chceme třídit. Zpravidla numericky nebo podle abecedy. Klíč může být někdy pouze malá součást celého prvku.

Představme si prvek, záznam v databázi zaměstnanců. Obsahuje jméno zaměstnance, příjmení, datum narození, rodné číslo, oddělení na kterém pracuje, počet odpracovaných hodin za týden, průměrný měsíční plat, životopis, počet najezděných kilometrů ve služebním autě atd. Klíčem k seřazení zaměstnanců, například pro seznam docházky na školení, bude pouze jejich příjmení. Abstrahujme nyní od zaměstnanců se stejným příjmením.

Pokud tedy na následujících stránkách budeme mluvit o porovnávání prvků, vězte že je řeč o porovnávání jejich klíčů.

2.2.2 Vizualizace

V souladu s úvodním citátem si představujeme algoritmy jako myšlenky a nápady.

Zobrazení jednotlivých algoritmů jsem se rozhodl realizovat na osmi náhodně seřazených prvcích. Prvky jsou označeny číslicemi a zároveň pro vyšší přehlednost odstupněny barvou. Úsečkou je zobrazen přesun, který je na následujícím řádku zrealizován. V příloze práce pak lze najít zde uvedené obrázky rozpohybované formou flashové animace.

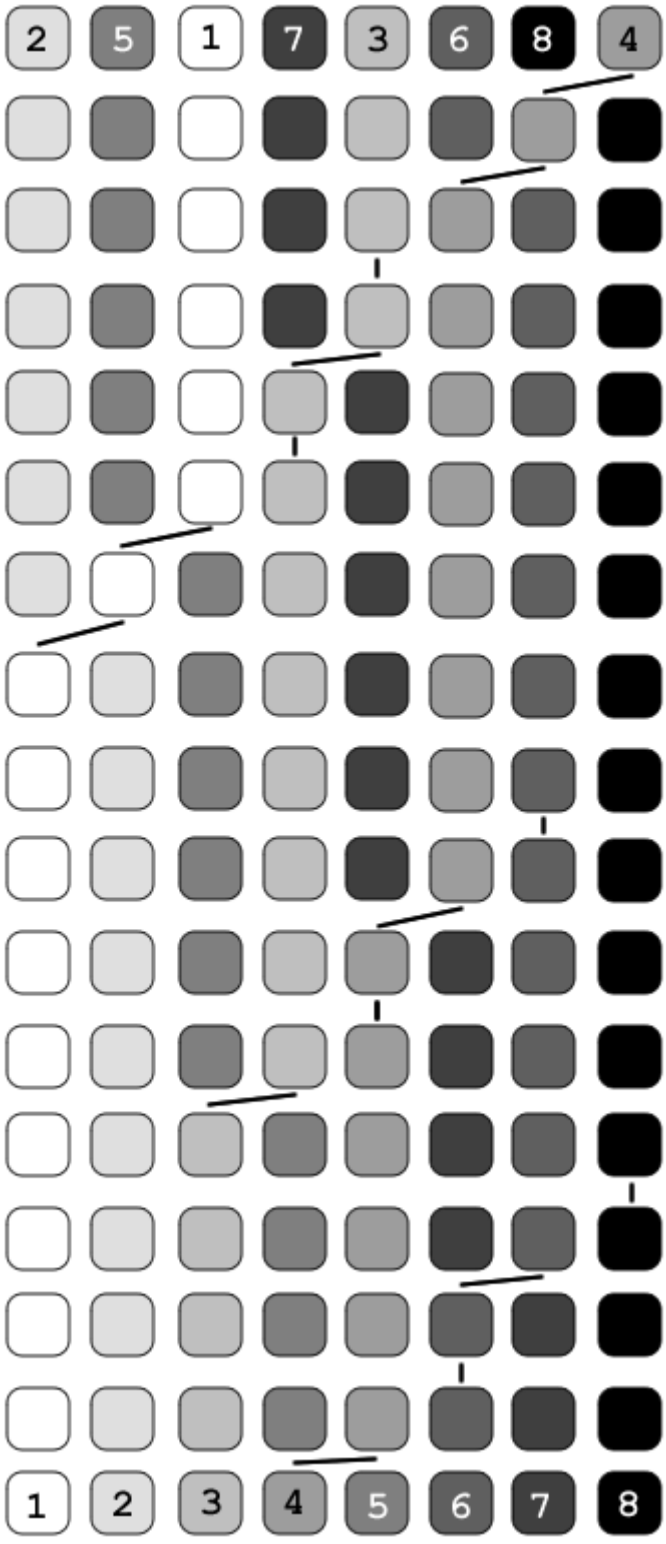
Na internetu je pak k dispozici celá řada interaktivních flashových či javovských aplikací a to i v češtině. Některé z jich jsou velice sofistikované, variabilní a jsou úzce provázány s programovým kódem.

2.3 Bublinkové řazení

Bublinkové řazení, neboli *bubblesort* jsme zmínili již na začátku této kapitoly. Mezi ostatními řadícími algoritmy má pozici otloukánka, který je pomalý neefektivní avšak z didaktického hlediska nelze začít jiným, než právě tímto algoritmem.

Bubblesortu, jako outsidera, se dotkl i současný prezident USA Barack Obama, když byl v roce 2008 na jednom pódiu s předsedou správní rady Goggle Inc. Ericem Schmidtem [12]. Ten se jej žertem zeptal jaká je nejúčinnější cesta k setřídění milionu dvaatřicetibitových celých čísel. Po několika okamžicích ticha, kdy se možná i Schmidt zalekl vlastní drzé otázky, Obama překvapil v podstatě nešpatnou odpovědí. Nezaměňovat pojem nešpatný za správný. Doslova odpověděl: *„I think, the bubble sort would be the wrong way to go.“*

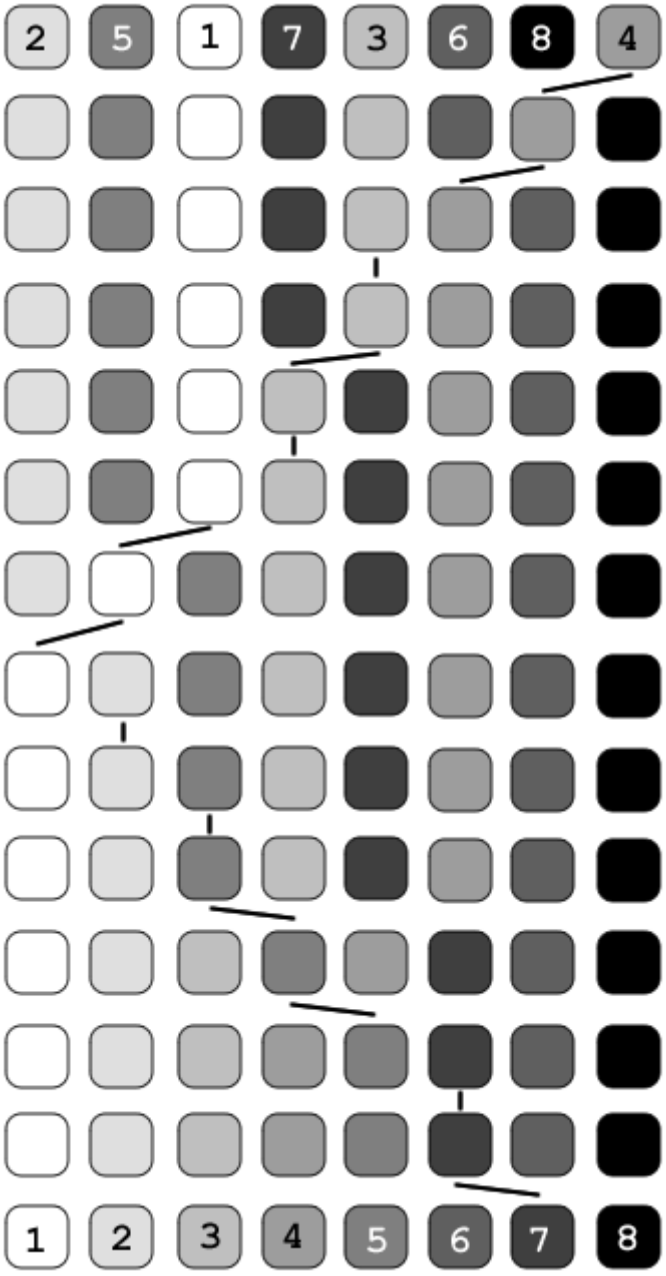
Na příkladu seřazení žáků podle velikosti jsme si bublinkové třídění ukázali i se slovním popisem algoritmu. Ve zkratce, nebo pro ty, jež přeskakují úvody, porovná se vždy dvojice sousedních prvků a buďto se prvky přehodí, nebo zůstanou. Takto se pokračuje s další dvojicí. Při jednotlivých průchodech polem probublávají na začátek aktuálně nejnižší prvky.



Obr. 4: Bubble sort

Při představě sta, tisíce nebo milionů prvků tuší člověk, který princip pochopil, značnou neefektivnost a pomalost. Neúčinnost toho algoritmu se snaží nepatrně snížit jeho

vylepšená verze zvaná třídění natřásáním, s*hakesort*. Oba názvy jsou vlastně dosti přiléhavé, nejlehčí bublinka vystoupá na povrch nejrychleji a naopak nejtěžší a největší kameny oddělíme při použití prvního kátru. *Shakesort* kombinuje oba tyto přístupy. Tedy při prvním průchodu polem najdeme a umístíme nejmenší prvek a zároveň cestou zpátky hledáme nejvyšší a ten vložíme na druhý (nebo první?) konec. Tedy na začátek. Ubyde tak počet průchodů celým polem, resp. Jejich délka.



Obr. 5: Shakesort

Algoritmus *bubblesort* je rychlý a použitelný v jediném případě, a to když je pole prvků téměř seřazené.

Průměrná složitost: $O(n^2)$

Stabilní: ANO

Přirozený: ANO

2.4 Řazení přímým vkládáním

Řazení přímým vkládáním, nazývané *insertsort*, jsme si již také dotkli. Je to ono řazení si karet v ruce. Nyní se na něj podíváme hlouběji.

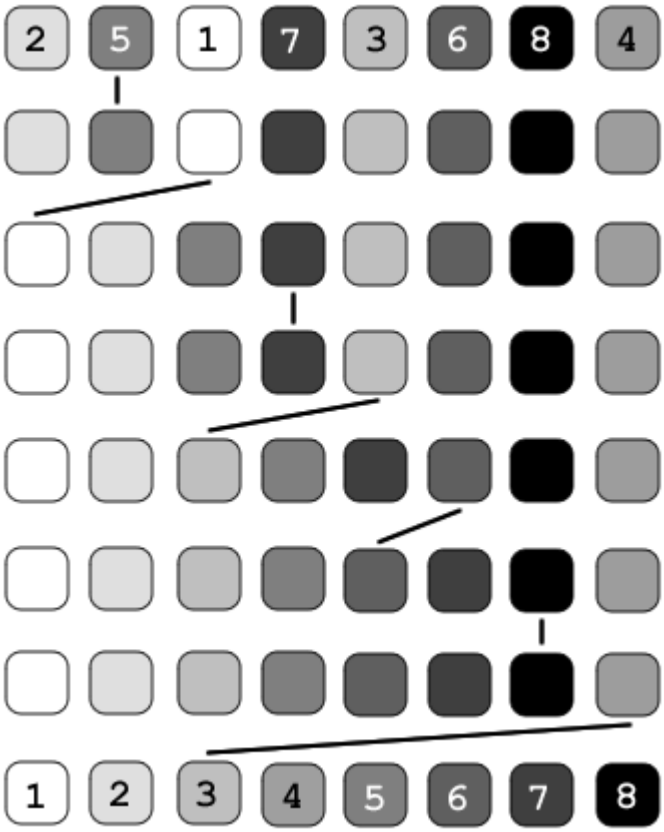
Pracuje na principu vkládání prvku ze zdrojové do cílové posloupnosti. Tedy z nesetříděné do setříděné. Na začátku tvoří setříděnou posloupnost jediný, a to první prvek. Algoritmus vezme do mezipaměti (hráč do ruky) následující prvek a vloží jej před první, nebo jej ponechá na místě. V závislosti na předem zadaných pravidlech. Takto pokračuje pro každý následující prvek (kartu). Tedy při každém průchodu polem vezme aktuální prvek zdrojové a vloží jej na odpovídající pozici v cílové posloupnosti.

Je řazení vkládáním stabilní? Představme si, že aktuální prvek je číslo 37. V definici algoritmu, jehož výstupem má být vzestupná posloupnost, se říká, pokud je aktuální prvek nižší než poslední, vlož jej na správné místo, pokud ne, ponech jej. Žádná zmínka o tom, že jsou stejné. Narazí-li na sebe ale při porovnání dva prvky s hodnotou 37, nepracuje se s nimi dál, neboť není splněna podmínka, že by jeden byl větší druhého. Algoritmus je tedy stabilní.

Podobně je to s jeho přirozeností. U takovéhoto algoritmu, jež se věřím dá představit i prostým slovním popisem, odtušíme, že částečně seřazenou posloupnost zpracuje rychleji, než nahodilou. Třídění přímým vkládáním tedy je i přirozené.

I takovýto jednoduchý princip má v sobě ještě potenciál vylepšení. A opět se dotýká reálného světa. Metoda půlení intervalu je základní a intuitivní přístup k vyhledávání v setříděné posloupnosti. I ten jsme si ukázali v předchozích kapitolách a víme že má logaritmickou složitost. Tedy zlepšení třídění přímým vkládáním je v druhé části algoritmu. Tedy v samotném vkládání. Vhodná pozice pro prvek nebude vyhledána postupně, tedy lineárně (se složitostí O(n)), ale pomocí metody půlení intervalu, jinak nazývané binárním vyhledáváním.

Tedy tak, jako bublinkové třídění lze vylepšit tříděním natřásáním, i třídění přímým vkládáním je možno teoreticky zrychlit na třídění binárním vkládáním. Ve skutečnosti neušetříme počet přesunů ani porovnání, a proto zrychlení není až tak markantní.



Obr. 6: Insertsort

Průměrná složitost: $O(n^2)$

Stabilní: ANO

Přirozený: ANO

2.5 Řazení přímým výběrem

Poslední z elementárních řadících algoritmů je třídění přímým výběrem, *selectsort*. Jde opačnou cestou než předchozí představitel. A sice, v každém kroku algoritmu, projde celou nesetříděnou posloupnost a najde nejmenší prvek, pokud tedy chceme seřazovat vzestupně. Tento umístí na odpovídající, tedy první pozici. V dalším kroku hledá druhý nejmenší prvek, nebo jinými slovy nejmenší prvek pro aktuální nesetříděnou posloupnost.

Jak algoritmus postupuje, zvětšuje se, podobně jako u *insertsortu* setříděná posloupnost na úkor nesetříděné na začátku řady. Na první pohled se zdá, že jsme se od předchozího postupu nijak výrazně dále nehnuli, ale není to docela pravda.

Elementární kroky uvnitř třídících algoritmů jsou porovnání prvků a přesuny prvků. Základní a správnou myšlenkou *selectsortu* je, že přesuny jsou časově náročnější než porovnání. A platí to opět, jak u karet, tak v počítačích. Třídění přímým výběrem jde cestou minimalizace přesunů, samozřejmě na úkor porovnání.

Uvažujme následující posloupnost deseti prvků, tedy čísel: 34, 29, 15, 23, 23, 25, 49, 17, 61, 7. Nemá v sobě žádnou pravidelnost, ani extrémní rozvržení, tomu se budeme věnovat dále. Animace algoritmů a datových struktur Jana Michelfeita [13] nám vyčíslí počty přesunů a porovnání za použití jednotlivých postupů.

X	*Přesuny*	*Porovnání*
Bubblesort	24	51
Insertsort	33	33
Selectsort	9	45

Ačkoliv to není reprezentativní vzorek ani pokus, přesto dává tušit, že cesta zminimalizování počtu přesunů prvků bude správná. A na tuto posloupnost bude dosahovat nejlepších výsledků.

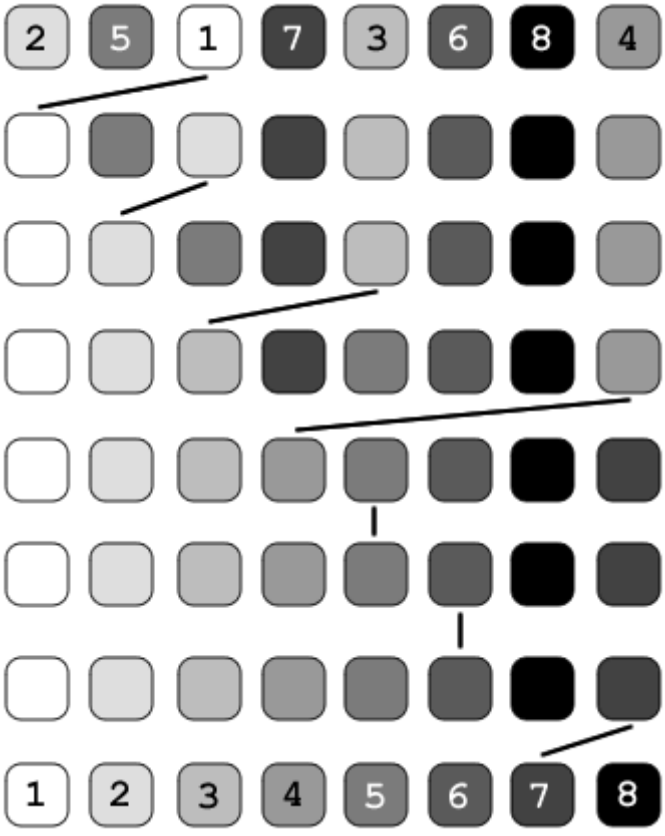
Jak už to bývá nejen v informatice, jedna výhoda je vykoupena druhou. Řazení přímým výběrem je nepřirozený a v závislosti na implementaci může být nestabilní algoritmus. Jeho nepřirozenost je projeví zvláště v téměř seřazené posloupnosti.

Příkladem budiž stejná množina čísel, avšak v téměř seřazeném pořadí. 7,15,23,17,23,25,19,49,29,61

X	*Přesuny*	*Porovnání*
Bubblesort	5	22
Insertsort	14	14
Selectsort	9	45

Opět zdůrazňuji, že tento příklad neslouží k nějaké generalizaci. Má pouze ukázat, že vlastnosti jednotlivých algoritmů, které jsme si ukázali si může každý jednoduše ověřit. Co tedy lze z tabulek vyčíst? I na takto malé množině je zdůrazněná výrazná přirozenost bublinkového třídění, které se zjednodušeně řečeno zrychlilo na třetinu. Třídění přímým vkládáním na této množině také dosáhlo lepších výsledků. A jak se zprvu zdálo ryze přirozená vlastnost, a to přirozenost, u třídění přímým výběrem chybí.

To, zda budou položky v tříděné posloupnosti seřazené, nebo částečně seřazené leckdy nejsme s to zjistit. Co nám ale může pomoci při výběru algoritmu, jsou vlastnosti prvků, resp. jejich klíčů. Pokud bude posloupnost složena ze prvků, které budou mít malé klíče, ale celkově budou obsáhlé. Vzpomeňme na úvod, kde jsme si řekli, že klíč může být pouze malou součástí prvku. V takovýchto případech může být volba *insertsortu* velice vhodná, neboť režie s přesouváním objemných prvků může být zásadní.



Obr. 7: Selectsort

Průměrná složitost: $O(n^2)$

Stabilní: v závislosti na implementaci

Přirozený: NE

2.6 Ještě než budeme rychle třídit

Algoritmy, které jsme si ukázali dosud patří do skupiny didaktických. Jejich princip je zřejmý a i začínající programátor by ho měl být schopen sepsat minimálně v pseudokódu. Pokud budeme třídit malé množiny, řádově do stovek prvků, nemusí být volba některého z těchto základních třídících algoritmů špatná. S narůstajícím počtem prvků, ale kvadraticky vzrůstá doba zpracování a na setřídění milionu dvaatřicetibitových čísel opravdu *bubblesort* není správnou cestou.

Ještě než si ukážeme algoritmy patřící do skupiny rychlých představme se několik pojmů, s nimiž budeme dále pracovat.

Rozděl a panuj (*divide et impera, divide and conquer*) je způsob, jakým lze zvládnout obtížné a obsáhlé problémy. Autorem citátu je italský spisovatel a politik Niccolò Machiavelli a čas ukázal, že je použitelný i na řadu jiných oblastí, než je politika a umění vládnout. Princip je jasný, rozdělit problém na menší, ty poté na ještě menší, dokud nenarazíme na triviální, které již řešit lze. Zpětným poskládáním dosáhneme vyřešení, či zvládnutí celku. My si ukážeme aplikaci tohoto přístupu záhy.

Backtracking. Může připomínat metodu zvládnutí problému hrubou silou. Leckdy se mu však vyhnout nelze. Nazývá se také prohledávání do hloubky, nebo prohledávání s návratem. Backtracking používáme, když hledáme cestu z bludiště. Obecné pravidlo může znít: *„Pokud dojdeš do slepé uličky, vrať se na poslední křižovatku a rozhodni se jinak.“*

Rekurze je označení, volání nebo popis něčeho částí sebe sama.

Medián. Pojem ze statistiky, který označuje prvek, jež splňuje tato pravidla. 50% ostatních je menší nebo rovno mediánu a 50% ostatních je větší nebo rovno mediánu.

$$50\% \text{ prvků} \geq \text{medián} \geq 50\% \text{ prvků}$$

Není to průměr? Není. Představme si firmu, která má pět zaměstnanců. Ředitel bere měsíčně sto tisíc korun, čtyři dělníci dostávají po deseti tisících. V průměru si každý zaměstnanec firmy vydělá dvacet osm tisíc měsíčně. Medián platů by však byl oněch deset tisíc korun. Existují tedy situace, kdy použití průměru není vhodné, nebo dokonce možné.

2.7 Quicksort.

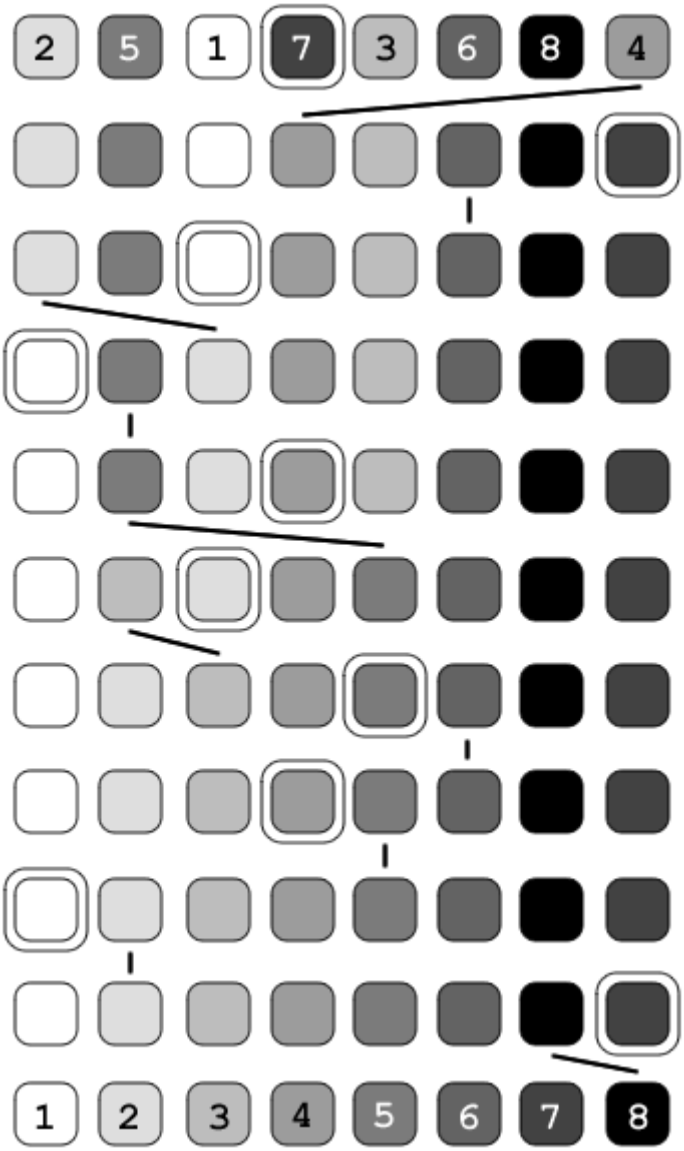
Tento algoritmus publikoval v roce 1962 sir Charles Antony Richard Hoare v britském měsíčníku *Computer Journal* [14]. Již název dává tušit, že bude třídit rychle. Skutečně, na náhodně, nebo pseudonáhodně seřazených posloupnostech dosahuje nejlepších výsledků, ze všech, co jsme si dosud představili. Udává se, že má složitost $O(n \cdot \log(n))$. Ve výjimečných případech, jako je například reverzně seřazená posloupnost dosahuje však složitosti $O(n^2)$. K tomu se ale dostaneme až na závěru kapitoly.

Princip a základní myšlenka *quicksortu* je v přemisťování prvků na velké vzdálenosti. Ty na předcházejících ukázkách nefungují efektivně.

Na nesetříděné posloupnosti zvolíme prvek a označíme jej jako *pivota*. Rozdělíme ji, tak aby prvky s klíčem menším než má pivot byly vlevo a s větším vpravo. Totéž pak můžeme udělat s jednotlivými částmi. A s částmi částí. Záměrně nepoužívám slovo polovina, neboť polovina to být nemusí. Čím se tříděné části stávají menší, tím jsme blíže stavu, kdy část bude tvořit jeden prvek. Panování pak probíhá cestou zpět, kdy se seřazují větší a větší části, až po celek.

Klíčovým bývá vhodné zvolení pivota. Zdá, že ideálním pivotem by byl výše zmíněný medián (pozor, ne průměr, protože prvek s průměrem klíčů se v posloupnosti nemusí vůbec vyskytovat). Na to abych však zjistili, který prvek je medián je zapotřebí projít celou posloupnost, nebo alespoň její velkou část a to se zbytečně negativně projeví na složitosti.

Další cestou pro zvolení pivota je, vzít první, prostřední nebo náhodně vybraný prvek. Jak se lze dočíst u Večerky [9] ve standardních případech opravdu algoritmus dosahuje složitosti $O(n \cdot \log(n))$ a pesimistická složitost se na běžných datech nevyskytuje.



Obr. 8: Quicksort

Průměrná složitost: O(n·log(n))

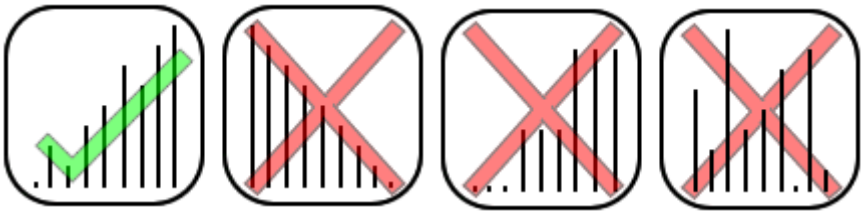
Stabilní: v závislosti na implementaci

Přirozený: NE

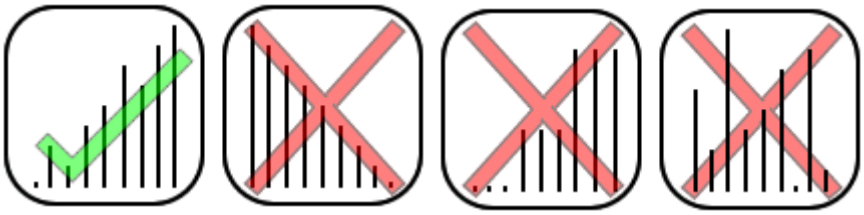
2.8 Extrémní rozvržení.

Dá se říci, že pro každý námi popsaný algoritmus lze nalézt vhodnou i nežádoucí posloupnost. Některé jsme si popsali v průběhu průvodních textů a jsou odvislé od jejich přirozenosti a stability.

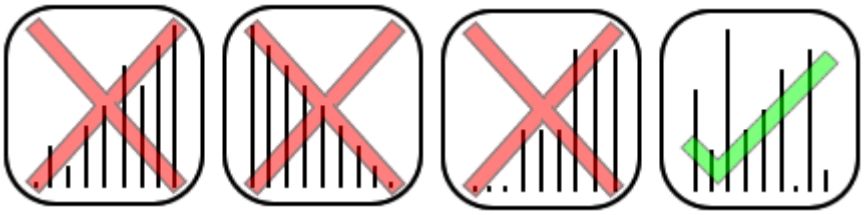
Extrémními rozvrženími rozumím posloupnost téměř seřazenou, obráceně seřazenou a tvořenou pouze několika unikátními klíči. Pro úplnost přidáme i piktorgram náhodné posloupnosti.



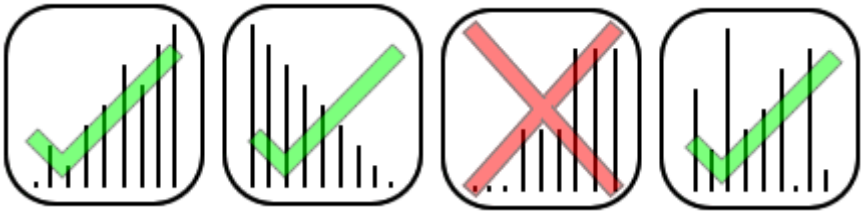
Obr. 9: Použití bubblesort



Obr. 10: Použití insertsort



Obr. 11: Použití selectsort



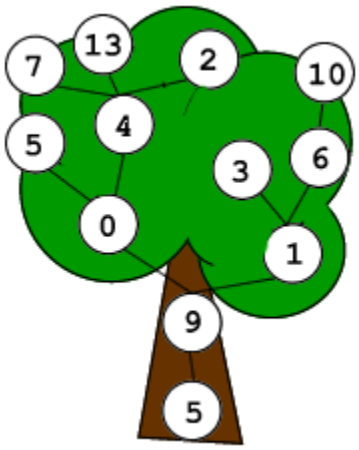
Obr. 12: Použití quicksort

2.9 Třídění haldou

Vnitřní pochody posledních dvou algoritmů si již nebudeme rozebírat tak podrobně jako dosud. Spíše si představíme na jakých myšlenkách jsou navrženy.

Heapsort, představil J. W. J. Williams v magazínu Communications of the ACM roku 1964 [15].

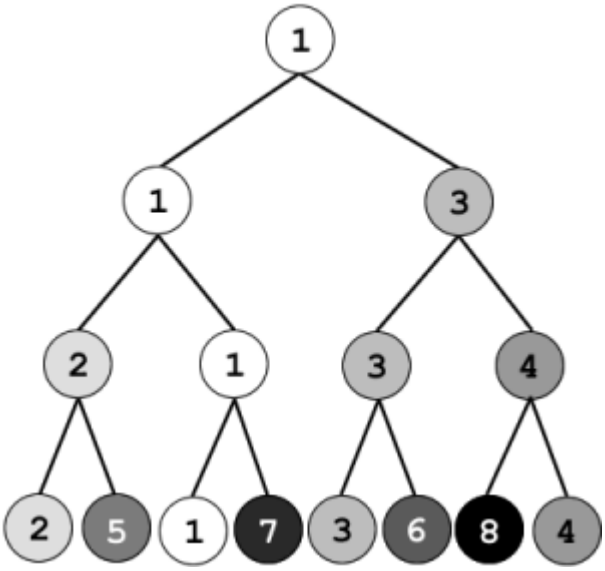
Halda, anglicky *heap*, je datová struktura, která vychází ze stromu. I strom je datová struktura, tvořená prvky (nazývanými také uzly) a ukazately na prvky. Jeden uzel je označený jako kořen, ten nemá žádné ukazatele na předky. Každý další prvek má předka a může mít potomka či potomky. Pokud je nemá je to list.



Obr. 13: Strom

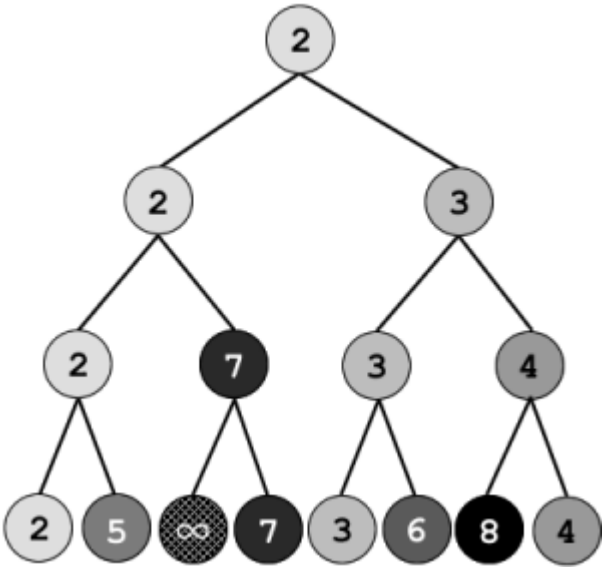
Binární strom je zvláštní případ stromu, kdy předek může mít nejvýše dva potomky.

Porovnávací strom prvků je zvláštní případ binárního stromu, kde jako předek je uvedený menší z obou potomků. V informatice a diskrétní matematice se stromy kreslí téměř bez výjimky kořenem vzhůru.



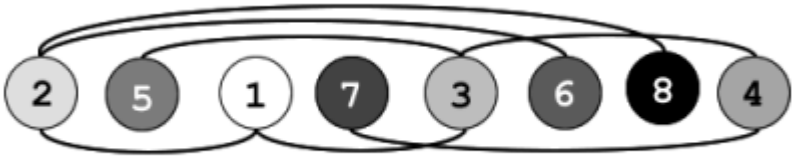
Obr. 14: Porovnávací strom

S pomocí toho stromu lze třídit, a to odebíráním kořene a jeho zařazováním do pole, které již bude setříděné. Nejmenší prvek na pozici listu nahradíme $+\infty$ a strom prvků opět překreslíme. Takto postupně odebíráme kořeny a naplňujeme strom nekonečny.



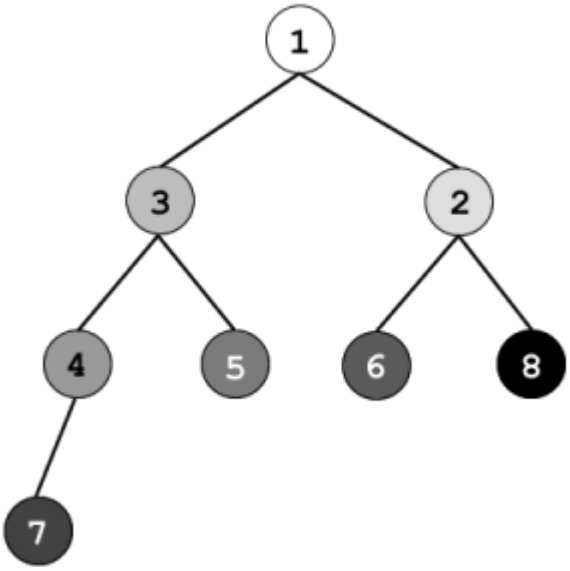
Obr. 15: Porovnávací strom - odebírání

Toto třídění má ale negativní důsledek. A sice, na vytvoření stromu a umísťování prvků potřebujeme další paměť, téměř dvojnásobnou. Tuto a ještě několik dalších nevýhod řeší datová struktura halda. Tu ji vystavět přímo na poli.



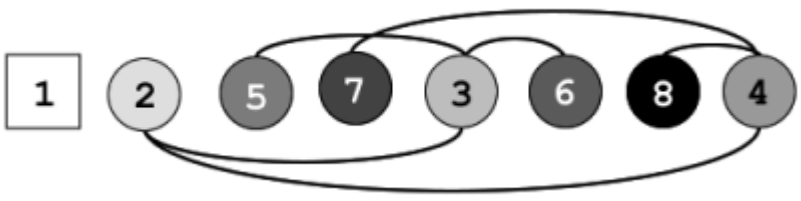
Obr. 16: Halda

Pro lepší představu si ji překreslíme.



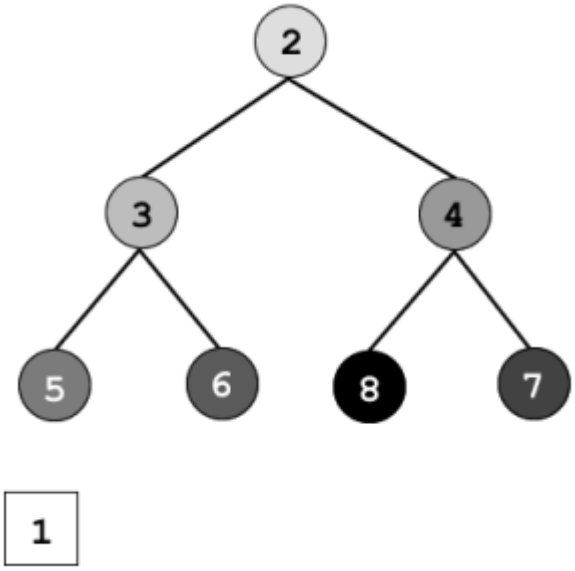
Obr. 17: Překreslená halda

Při třídění haldou tedy v prvním kroku sestrojíme nad daty haldu a v druhém odebíráme nejnižší prvky, vkládáme je na začátek nebo konec pole a haldu přestavujeme tak, aby byla vždy haldou.



Obr. 18: Halda - odebírání

A ještě jednou halda překreslená.



Obr. 19: Překreslená halda - odebírání

Heapsort je jeden z nejrychlejších známých třídících algoritmů. Má zaručenou složitost O(n·log(n)). Vzpomeňme na *quicksort*, jeho logaritmická složitost byla pouze průměrná a při nevhodně seřazených datech se mohla vyšplhat až na kvadratickou. Tento fakt diskvalifikuje *quicksort* v situacích, kdy je nutno se na dobu trvání běhu algoritmu spolehnout.

Průměrná složitost: O(n·log(n))

Stabilní: NE

Přirozený: NE

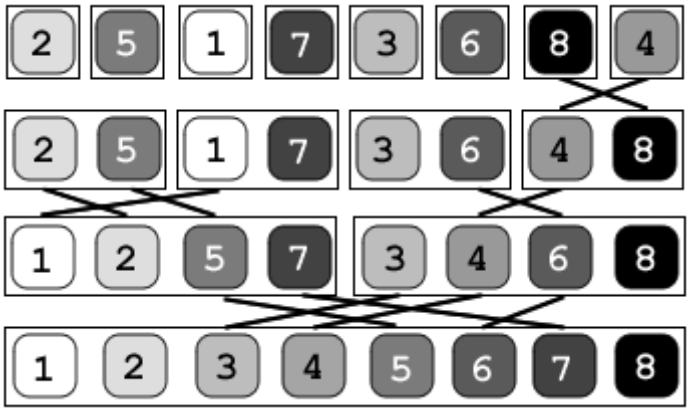
2.10 Třídění slučováním

Poslední způsob třídění, který si ukážeme je třídění přímým slučováním. Navrhl jej pionýr moderní počítačové vědy John von Neumann roku 1945 [16]. Ukázkovější použití přístupu *rozděl a panuj* v této práci nenajdete.

Myšlenka je velice jednoduchá. Není problém seřadit dvě již seřazené posloupnosti. Proto algoritmus v první části rozděluje prvky do přihrádek, nejmenší přihrádka má

velikost jednoho prvku, je tedy seřazená. Dále již pouze slučuje seřazené posloupnosti. Tedy dvojice, čtveřice atd. Posledním krokem je seřadit dvě již seřazené poloviny celkové posloupnosti.

A samozřejmě jako v jiných částech práce si řekneme, čím jsou vykoupeny výhody toho algoritmu. Nespornou výhodou je zaručená logaritmická složitost, přirozenost a stabilita. Handicapem pak je nutnost alokace paměti ve stejné velikosti jako je soubor dat.



Obr. 20: Mergesort

Průměrná složitost: O(n·log(n))

Stabilní: NE

Přirozený: NE

2.11 Ostatní třídící algoritmy

Nevyčerpali jsme výčet všech existijících algoritmů. Známe ještě celou řadu buďto jiných přístupů nebo modifikací námi popsaných. Některé algoritmy neporovnávají klíče prvků, ale snaží se vypočítat jeho pozici v seřazené posloupnosti (*counting sort*) nebo porovnávají pouze jejich části klíčů (*radix sort*).

Na závěr kapitoly si popíšeme slíbený stupidsort. Tento algoritmus má faktoriálovou složitost; vzpomeňme, na podkapitolu o složitostech a tabulku s přibližnou dobou výpočtu. Pricnip je takový, že náhodně (nebo pseudonáhodně) zamícháme všechny prvky a zkontrolujeme, zda nejsou seřazené. Takto pokračujeme dále, dokud nebudou všechny prvky seřazené. Existuje ještě několik modifikací, například prohodíme dva náhodně vybrané prvky a zkontrolujeme, zda není seřazeno.

3 PROBLÉM OBCHODNÍHO CESTUJÍCÍHO.

3.1 Algoritmus je výkonný nástroj, ...

V prvních dvou kapitolách jsme si algoritmy představili jako výkonné nástroje. Spousta jejich variant, modifikací a specializací se hodí na různé druhy úloh a různé soubory dat. Některé jsou univerzálnější, jiné méně použitelné.

Také na ukázkách třídění pouhých prvků s číselnými klíči jsme představili řadu přístupů. Klíče mohou být a často také jsou řetězce, tedy slova. V této sféře se tedy skrývá další manipulační prostor pro různé postupy a myšlenky, jak dojít k cíli.

V poslední části se naopak seznámíme s problémy reálného světa, na které *dosud* neexistují algoritmická řešení. Tudíž ani nejmodernější počítače je nejsou schopny vyřešit a dokud budou počítače současné, tedy Von Neumannovy architektury, nebudou moci a v budoucnosti.

3.2 ... někdy ale nestačí.

Úlohy které si představíme v poslední části textu jsou jednoduché zadáním, ale prozatím algoritmicky neřešitelné. Jejich zadání bývají názorná a jsou úzce spjata s reálným životem. Podobně jako s problémem obchodního cestujícího se v životě můžeme setkat například s problémem dvou loupežníků.

Ti mají před sebou lup sestávající se z různých bankovek, mincí a předmětů různé hodnoty. Jak se mají rozdělit spravedlivě? Jediná možnost, kterou bychom mohli uvažovat je vyzkoušet všechny kombinace. Počet řešení nám s počtem prvků roste exponenciálně. Opět tedy vzpomeňme na tabulku v kapitole 2.5.7. Pro dvacet uloupených předmětů existuje asi *jeden milion* kombinací. Pro dvacet pět asi *třicet tři milionů* kombinací. V reálném lese by se s touto situací loupežníci vypořádali asi tak, že by se rozdělili přibližně a zbytek by společně propili.

Počítačové řešení obdobných problémů je v zásadě stejné. Spokojíme se pouze s přibližným a dostatečným výsledkem. Takovýto postup se nazývá heuristický.

3.3 Matematikou k bohatství.

Podle magazínu Forbes [17] byl v roce 2010 šestým nejbohatším člověkem světa Lawrence Ellison. Americký podnikatel a spoluzakladatel jedné z největších společností zabývajících se databázemi – Oracle, vystoupil v roce 2000 na půdě prestižní americké univerzity Yale. V jeho ostrém satirickém projevu mimo jiné zaznělo: *„Podívejte se na člověka vlevo, břídil, podívejte se na člověka vpravo, břídil. Jste banda břídilů. Bill Gates, nejbohatší člověk světa – nedokončil vysokou školu, Michael Dell, třetí nejbohatší člověk světa – nedokončil vysokou školu, já, šestý nejbohatší člověk světa jsem nedokončil vysokou školu. Vy už jste ztracení, ale obracím se studenty z nižších ročníků, pokud chcete v životě něčeho dosáhnout, odejděte ze školy. Sbalte si vaše myšlenky a vypadněte. Vaše barety a taláry vás budou jen tahat dolů.“*

Tento příběh je smyšlený a patří do souboru takzvaných městských legend [18], o kterých si mnoho lidí, co je slyšelo nebo četlo, myslí, že jsou pravdivé. Ve skutečnosti poukazuje mimo jiné na to, že v očích mnoha lidí, je vědecká a akademická práce zcela jistě náročná, avšak nevedoucí k úspěchu a prestiži a bohatství. To na nás čeká především ve světě byznysu.

Nechci se pouštět do sociologických a statistických průzkumů, ale minimálně v jednom případě tato myšlenka neplatí. Clayův matematický institut [19] vyhlásil v roce 2000 sedm problémů tisíciletí, za jejichž vyřešení nabízí jeden milion dolarů. Většina zadaní je na tak vysoké úrovni matematiky, že není vhodné ani nutné je sem psát. Jedno však úzce souvisí s tématem této práce a věnuje se mu celá tato kapitola.

Pokud se vám podaří zodpovědět otázku zda

$$P = NP$$

nebo nikoliv vězte, že vás odměna jednoho milionu dolarů nemine a ať bude odpověď kladná či záporná bude mít dalekosáhlé důsledky.

Satirický komentář k tomuto tématu by jistě zmínil ruského matematika Pelermana, který jednu z otázek tisíciletí vyřešil. Není však výjimečná genialita vykoupená jinými charakterovými vlastnostmi? Dr. Pelerman, totiž ocenění i cenu již dvakrát odmítl.

3.4 P = NP?

Tato rovnice, či výraz jehož platnost není dosud potvrzena ani vyvrácena předpokládá, že jsou různě složité problémy. Ptá se zda lze ty složité (*NP*) převést na ty jednoduché (*P*)?

Abychom pochopili formální popis jednotlivých tříd složitosti vysvětlíme si nejprve význam některých slov.

Determinismus ve volném významu předpokládá, že následky jsou přesně určeny příčinami. V našem případě, že při stejné kombinaci vstupních dat dosáhneme vždy stejného výstupu. Lze si představit, že když na stejnou posloupnost čísel použijeme například *mergesort* i po tisící dostaneme stejnou řadu seřazených čísel. Třídění přímým slučováním je tedy *deterministický* algoritmus.

Nedeterministický algoritmus však na stejný soubor vstupních dat může odpovídat různým výstupem. V naprosté většině případů ale dostačujícím. Jestliže si loupežníci rozdělí kořist v poměru 100:98 nebo 102:99 v dané situaci není podstatné a v obou případech bude spokojenost vzájemná.

Polynomiální čas můžeme chápat jako kvadratickou či kubickou složitost. Polynomiální čas je pro nás limitou, přes kterou se při navrhování a používání algoritmů nechceme dostat. Algoritmus pracující v polynomiálním čase je brán jako rychlý.

Problémy zařazené do třídy P mohou být *deterministicky* vyřešeny v *polynomiálním* čase a lze v něm najít výsledek. Problémy obsažené v třídě NP mohou být v *polynomiálním* čase vyřešeny *nedeterministicky* a lze je v něm ověřit.

Tedy třídění, například některou z metod, které jsme si ukázali v druhé kapitole patří mezi problémy třídy složitosti P. A problém obchodního cestujícího, který si představíme v této je z třídy složitosti NP.

Pojem, se kterým se může v této souvislosti ještě setkat je třída NPC, neboli NP – kompletní. Takto bývají označovány nejtěžší problémy ze třídy NP.

Problém *P* vs. *NP* definovali nezávisle na sobě profesor kanadské univerzity v Torontu Stephen Cook a původem ruský vědec a nyní profesor na Bostonské univerzitě Leonid Levin již v roce 1971 [19].

3.5 Jemný úvod do teorie grafů.

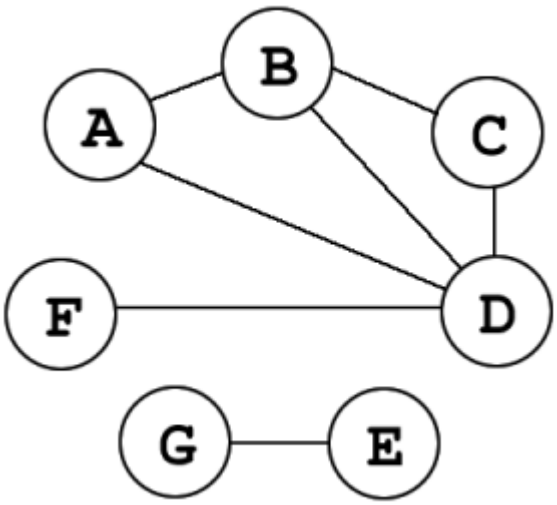
Ještě, než si ukážeme problém obchodního cestujícího, představíme si několik základních pojmů a principů z odvětví diskrétní matematiky nazývaného teorie grafů.

Graf je soustava *vrcholů* a *hran*. Jistým druhem grafu je i výše zmiňovaný strom. *Vrcholy* jsou nazývány prvky (i kořeny i listy) a *hranami* ukazatele či spojnice mezi jednotlivými vrcholy. Typů grafů existuje značné množství a lze jimi demonstrovat řada jevů.

Jako přiléhavá ukázka grafu a jeho různých variant nám může posloužit, ať se držíme pedagogické oblasti, tzv. *sociogram*. Sociogram je vizualizace vztahů ve třídě nebo v jiné skupině. Přehledně reprezentuje vazby mezi žáky, ty mohou být kladné i negativní. Mohou vyjadřovat různou intenzitu, nebo naopak pouze binární hodnotu, kamarádím, nekamarádím. Ve formě grafu lze odhadnout sociální vyloučení, potenciální hrozbu šikany.

3.5.1 Jednoduchý graf.

Uvažujme tedy malou skupinu žáků. Adéla, Bára, Cyril, David, Eva, Franta a Gábina. Pokud bychom měli v psaném textu popsat vazby mezi žáky, tedy kdo s kým kamarádí bylo by čtení dat nepřehledné a ve větším počtu žáků téměř nemožné.



Obr. 21: Základní graf

Použití grafu se však v tomto případě jeví jako užitečné. Kromě toho, že jsme zjistili, že Gábina kamarádí pouze s Evou a David je nejpřátelštějším členem skupiny jsme z hlediska teorie grafů zjistili, že se jedná o nespojitý graf se sedmi vrcholy a šesti hranami.

Aby se s grafy mohlo počítat často jsou reprezentovány pomocí tabulek, které se v matematice nazývají *matice*. Představme si tento graf jako matici. Ve sloupci i v řádku jsou jména týchž žáků. Tabulka je naplněna nulami. Existující vztah mezi dvěma lidmi představuje nula nahrazená jedničkou.

	A	B	C	D	E	F	G
A	0	1	0	1	0	0	0
B	1	0	0	1	0	0	0
C	0	0	0	1	0	0	0
D	1	1	1	0	0	1	0
E	0	0	0	0	0	0	1
F	0	0	0	1	0	0	0
G	0	0	0	0	1	0	0

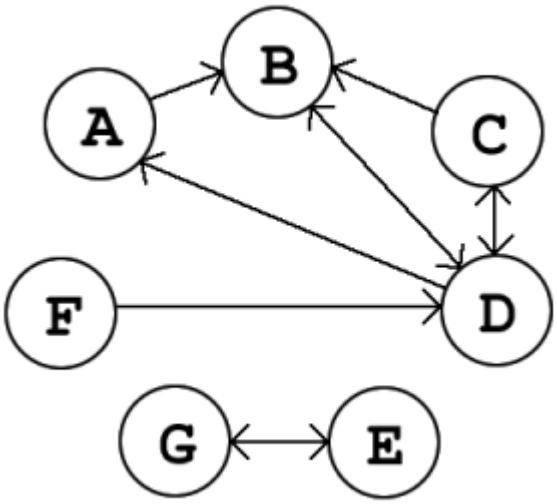
Obr. 22: Matice grafu

Matice je v tomto případě symetrická. Je to dáno zjednodušením prvního příkladu grafu. Zaměřením se na konkrétní řádek zjistíme například, že Gábina kamarádí pouze s Evou a nejvíce jedniček, tedy nejvíce kamarádů má David. V takovémto malém množství se možná jeví přehlednější grafické znázornění, nicméně při počtu stovek a tisíců prvků by už papír pod nánosem inkoustu nebyl viditelný.

V ukázkách dalších typů grafů si již jen naznačíme jak by vypadala jeho matice.

3.5.2 Orientovaný graf.

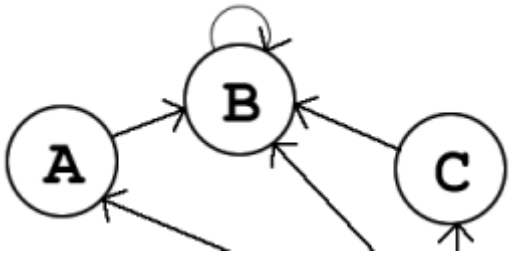
Předchozí graf, chcete-li sociogram je nereálný. Zobrazuje pouze vzájemné vztahy. Ve skutečném životě se setkáme i s příklady vztahů jednostranných. Může jimi být platonická láska o které nikdo další neví nebo třeba přetvářka a pokrytectví, kdy nás domnělý kamarád za zády pomlouvá a necítí vzájemnou vazbu nebo pouze nedorozumění.



Obr. 23: Orientovaný graf

Co jsme zjistili z hlediska sociogramu? Gábina s Evou jsou jsou mezi sebou uzavřené kamarádky. Franta, ač se tak necítí je *outsider* a nejspíše nějakým způsobem namyšlená či přetvařující se bude Bára, protože necítí k žádnému spolužáku kamarádství.

Pohledem teorie grafů jsme si určili další vlastnost a tedy orientovanost. Hrana může být orientovaná z jednoho prvku do druhého (Franta s Davidem), nebo z jednoho prvku do druhého a zpět (Gábina s Evou). Hrana může být také orientovaná z jednoho prvků do téhož. Na příkladu sociogramu to může vyznívat komicky, ale lze se domnívat že by do sebe zahleděná avšak všemi oblíbená mohla být Bára. Vrchol reprezentovaný narcistickou Bárou se nazývá smyčka, anglicky *loop*.



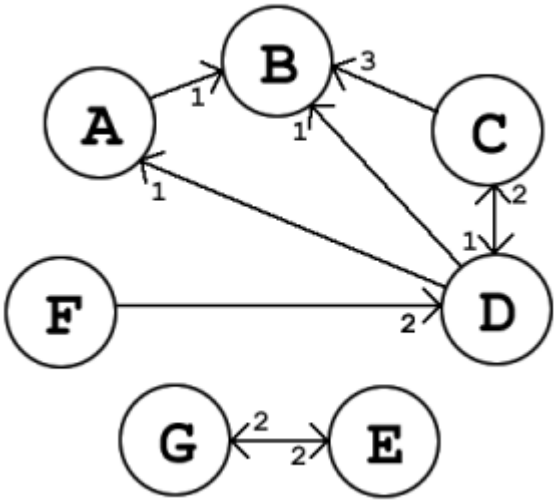
Obr. 24: Smyčka

Jak by se orientovaný graf projevil na matici? V našem případě, což ale nemusí být nutné vždy, by matice byla nesymetrická a Bářin nepěkný charakter by byl v matici označen jako jednička na spojnici písmen B a B.

3.5.3 Ohodnocený graf.

Dalším způsobem, jak graf přiblížit realitě je ohodnocení hran. Hrana jako spojnice mezi dvěma vrcholy uchovává ještě číselnou hodnotu, které může znamenat například míru obtížnosti dosažení, míru intenzity vztahu či vzdálenost mezi dvěma vrcholy.

Ohodnocený graf může i nemusí být orientovaný. V naší ukázce si vazby mezi spolužáky rozdělíme na tři úrovně. Kamarádství (označené jedničkou), přátelství (dvojkou) a láska (označená číslicí tři).



Obr. 25: Ohodnocený graf

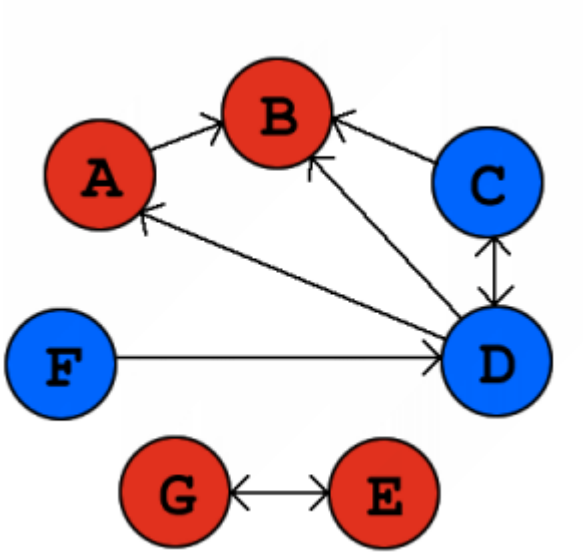
Sociálně jsme zjistili upřímné přátelství mezi Evou a Gábinou. Platonickou a nenaplněnou lásku Cyrila k Báře a nešťastného Frantu, který se cítí být Davidovým přítelem.

Z hlediska grafu je nejzajímavější částí vztah Cyrila a Davida. Totiž hrana Davida k Cyrilovi nemá stejná ohodnocení jako Cyrila k Davidovi. Tato situace může u grafu samozřejmě nastat. Ovšem pouze u grafu orientovaného. Graf, který nemá orientované hrany (šipky) je musí mít ohodnoceny pouze jednou hodnotou. Nejkratší možná cesta z

Českých Budějovic do Tábora musí být stejně dlouhá jako trasa opačným směrem. Předpoklad, že jsou cestou jednosměrné objížďky, které činí jednu z tras delší, již nesplňují podmínku neorientovanosti grafu. Tudíž je vše v pořádku.

3.5.4 Barevný graf.

Poslední modifikací grafu, kterou si ukážeme je jeho obarvení. Barva je vlastnost vrcholu. Barvou můžeme určit, že některé vrcholy pojí s jinými jistý rys, jenž může být pro danou úlohu určující. Z následujícího obarveného grafu můžeme vyčíst pohlaví studentů.

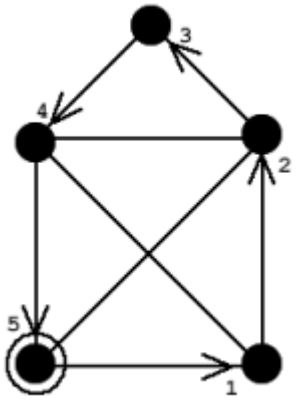


Obr. 26: Barevný graf

3.5.5 Hamiltonovský graf.

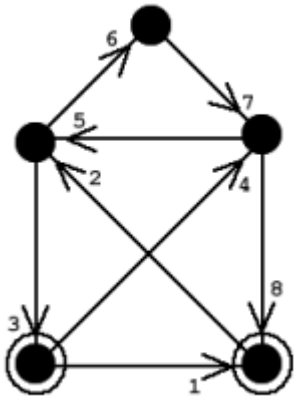
Opustíme mladý kolektiv a představíme si poslední ani ne tak typ, jako spíše vlastnost grafu. Hamiltonovským grafem je nazýván takový graf, jenž obsahuje hamiltonovskou kružnici. Jako budoucí pedagog bych se dozajista s takovýmto vysvětlením nespokojil. Proto podrobněji. Hamiltonovská kružnice je taková cesta grafem, která spojuje všechny vrcholy, každý navštíví právě jednou a počáteční vrchol je zároveň cílový.

Je dětský rébus domeček jedním tahem hamiltonovský graf?



Obr. 27: Hamiltonovská kružnice

Ano je, ale pouze pokud bude cesta mezi vrcholy vést po obvodu grafu.



Obr.28: Nehamiltonovská kružnice

Pokud by však cesta vedla tak jak je zvykem kreslit ho jedním tahem tah by nebyl úplný. Lišil by se počáteční a cílový vrchol. Přesto graf hamiltonovský je, protože v něm existuje alespoň jedna hamiltonovská kružnice.

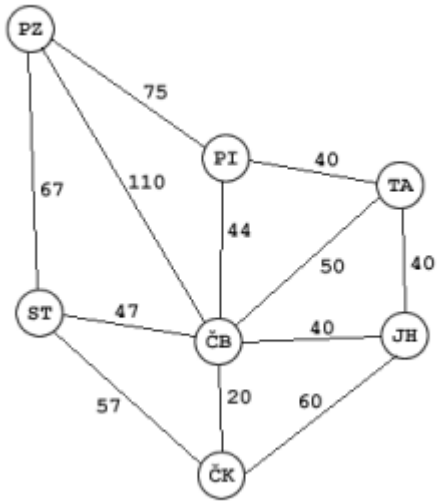
3.6 Problém obchodního cestujícího.

Po jemném úvodu do terminologie grafů pochopíme zadání problému obchodního cestujícího i po formální stránce. Úkolem je najít v ohodnoceném grafu nejkratší hamiltonovskou kružnici.

Problém obchodního cestujícího (*Travelling salesman problem*) je další ukázkou těsného propojení reálných každodenních problémů se světem vysoké matematiky a informatiky.

Činnost obchodního cestujícího sestává mimo jiné z organizace cesty a pořadí cílů, které musí navštívit. Jistě je jeho zájmem minimalizovat náklady a čas strávený na cestě.

Oblastí našeho obchodního cestujícího jsou Jižní Čechy a Plzeň. Má za úkol navštívit kromě již zmíněné Plzně ještě Písek, Tábor, Jindřichův Hradec, Český Krumlov, Strakonice a samozřejmě České Budějovice.



Obr. 29: Graf Jižních Čech

Řešením je vymezit si všechny trasy a spočítat jejich délky. Toto řešení hrubou silou má faktoriálovou složitost. V naší ukázce to není docela pravda, protože mi neexistuje přímá cesta například z Plzně do Tábora a další. Tím se výrazně sníží počet kombinací. Obecně se ale udává, že problém obchodního cestujícího faktoriálovou složitost má. Pro počet našich sedmi měst nalezneme *pět tisíc* různých posloupností jak je navštívit za sebou.

7! = 5 040

Když by náš obchodní cestující dostal za úkol starat se o zákazníky v *deseti* městech, počet kombinací vzroste na *tři a půl milionu*.

10! = 3 628 800

Cesty, kterými se ubírají řešitelé problému obchodního cestujícího a podobných jsou nalezením alespoň přijatelně krátké cesty. Jsou to tedy nedeterministické algoritmy a ty, jak jsme si řekli výše mohou udávat pro stejná vstupní data různé výsledky.

3.7 Skutečné řešení problému obchodního cestujícího.

Několik vědců a výzkumníků sdružených pod CRPC (The Center for Research on Paralel Computing at Rice Univertsity) [20] se problému obchodního cestujícího intenzivně věnuje od roku 1988. Ubírají se cestou optimalizace, paralelních výpočtů a výpočetních clusterů, což jsou vzájemně propojené kooperující počítače. První úspěch nastal v roce 1992. Byl zveřejněn časopisem Discover mezi padesáti nejdůležitějšími vědeckými objevy toho roku. Vypočítali cestu mezi třemi tisíci americkými městy a algoritmy zpracovávalo padesát počítačů. Další výsledky byli zvěřejněny roku 1993, tehdy tým řešil 4 461 měst.

Dosud největší úspěch byl překonání desetitisícové hranice. Problém obchodního cestujícího pro 13 509 měst vypočítal cluster tří Digital AlphaServerů 4100s a třiceti dvou PC Pentium II v roce 2003. Výpočet trval přibližně tři měsíce.

Jedním z řešitelů byl i původem český profesor Václav Chvátal. Ten se k výzkumu vyjádřil slovy: „Je to návykové. Nezáleží na tom jakého pokroku dosáhnete, vždy jste otravováni pocitem, že jenom malý krůček dopředu může znamenat kvantový skok. A dnes, když jsme zastavili, absťák. Nicméně objem toho, co jsme dokázali je ohromující.“

3.8 Využití NP problémů.

NP problémy zatím nejsme schopni *deterministicky* řešit v *polynomiálním* čase. Proto mají využití v kryptografii a vývojáři sofistikovaných šifrovacích metod doufají, že otázka *P = NP* nebude nikdy zodpovězena a pokud ano tak zamítavě. Za předpokladu, že by se zjistilo, že NP problémy lze *deterministicky* řešit v *polynomiálním* čase, otřáslo by to jejich

odvětvím, které mnohdy spoléhá na faktoriálovou složitost, která i při desítkách prvků dosahuje miliard a biliard kombinací.

Proč by měl programátor o NP, popřípadě NP kompletních problémech vědět? Mohl by se s nimi totiž ve své praxi setkat, avšak neuměl by je rozeznat. Vyplýtval bych pak mnoho energie na jejich řešení a nejpravděpodobněji by na něj nepřišel. Pakliže však identifikuje NP kompletní problém ví, že snazší bude postupovat cestou heuristiky.

Pokud některý ze čtenářů této práce má před sebou verzi elektronickou, vězte že ukázka NP kompletního problému je mu blíž než se zdá. Součástí většiny operačních systémů Windows je logická hra hledání min, *Minesweeper*. Profesor Birminghamské univerzity Richard Kaye [21] na semináři v roce 2003 odprezentoval, že hra hledání min je NP kompletní.

Větší část odborné veřejnosti je toho názoru, že P se NP nerovná. Například profesor Lance Fortnow [22] tvrdí, že pokud by se dokázal opak, celý internet by v historii vypadal jako poznámka pod čarou. Zajímavé, že vědci se neshodnou ani na tom, zda bude tato otázka někdy zodpovězena.

Velká naději si odborná veřejnost dělala v osmdesátých letech, kdy vědci doufali, že se se zvyšující se výpočetní silou počítačů blíží řešení otázky. Nadšení však postupně opadalo.

Zastánci vyřešení tohoto největšího problému současné výpočetní techniky a informatiky naopak argumentují například vyřešením Fermatovy věty. Tato jednoduše položená otázka odolávala řešitelům více než třista let. A jistě by se v její historii našla celá řada skeptiků.

ZÁVĚR

Název této práce je Algoritmy ve výuce na základní a střední škole. Pouhé přehození slov tedy, výuka algoritmů na základní a střední škole by celé práci udávala jiný směr. Druhý jmenovaný název si lze představit jako samostatný předmět, který se vyučuje na vysokých školách, popřípadě odborně založených středních. Slouží pak budoucím programátorům jako základ, na němž lze stavět a psát nové programy.

Algoritmy ve výuce na základní a střední škole si nekladou za cíl být samostatným předmětem, ale možnou součástí stávajících. Na gymnásiích se studenti podrobně seznamují se stavbou živočichů až na úroveň buněk, s literaturou dnes již nečitelných a nečtených autorů či řadou časů, jež se používají německý jazyk.

Pokud bychom současné století nazvali stoletím *informace*, jistě by si informatika zasloužila ve výuce větši pozornosti, než je vytváření tabulek a grafů.

Celý text jsem pojal tak, aby byl použitelný a čitelný i mezi neinformatiky a aby si z něj každý něco odnesl. Není nutné, aby to bylo hned zapálení pro psaní programů. Ale pouhé zamyšlení a uvědomění si, že i tak triviální věci, jako je seřazení souborů podle velikosti nebo podle abecedy, nalezení nejkratší nebo nejrychlejší trasy na mapě a řada dalších věcí a úkonů, které každý činí několikrát denně pouhým máčkáním prstů, mají hlubokou strukturu a smysl a na jejich rychlém a bezproblémovém chodu pracovalo a stále pracuje mnoho lidí z akademické a komerční sféry.

Pakliže by někdo ve své pedagogické praxi tento text použil, pocit zadostiučinění by mohl zažít, pokud by se se studentem sešel po letech a rozhovor by se ubírat tímto směrem.

„Pane učiteli, stále si pamatuji z vašich hodin, že algoritmus je konečný sled jednoduchých kroků a že jej lze použít na skupinu obdobných úloh.

A metoda Rozděl a panuj? Tu jsem v životě použil mockrát a programováním se neživím. Tehdy, pokud si dobře pamatuji se snad ani nevědělo, zda se P rovná NP.

Co ale platí stále je, že když je něco rychlejší nemusí to být vždy stabilní a přirozené.“

CONCLUSION

The title of this thesis is Algorithms in teaching in primary and secondary school. Merely throwing those two words, teaching of algorithms for primary and secondary school, work would show a whole different direction. The latter name can be thought of as a separate subject, which is taught at universities.Then serve as a basis for future developers on which to build, and write new programs.

Algorithms in teaching in primary and secondary school does not aim to be a separate subject, but part of the existing ones. At the high school all students will get acquainted with the construction of animals in detail to the level of cells, the literature now unread and unreadable authors and a number of times, which used the German language.

When we called this century, the century of *information*, the Computer science would have more attention than the creation of tables and graphs.

I took whole text so that it is usable and readable even by people not coming from the field of informatics. It is not necessary that it was just enthusiasm for writing programs.Take the mere thought and awareness, so that even trivial things such as sorting files by size or alphabetically, find the shortest or fastest route on the map and many other things and operations, which are several times done every day just simply by pressing the fingers,have a deep structure and meaning, and on their smooth and speedy operation worked and still works a lot of people from academic and commercial spheres.

I will feel satisfaction, when sometime in future can somebody listen this interview between teacher and ex-student.

„Hi Mr., I still remember, that algorithm is a finite sequence of simple steps and you can use it to group of similar tasks.

And a method divide and conquer? I have used many times in my life although I am not programmer.

If I remember, in that time people doesn´t know, that P equals NP. But what still hodls true is, that if something is faster, does not need to be allways stable and natural.“

SEZNAM POUŽITÉ LITERATURY

[1] HILLIS, Daniel W. *Vzor v kameni : Jednoduché myšlenky, které řídí počítače*. Vydání 1. Praha : Academica, nakladatelství Akademie věd. České republiky, 2003. 158 s. ISBN 80-200-1067-x.

[2] *Úřad pro technickou normalizaci, metrologii a státní zkušebnictví* [online]. 1.1.1996 [cit. 2010-05-24]. Česká norma. Dostupné z WWW: <http://csnonline.unmz.cz/Detailnormy.aspx?k=18290>.

[3] *Avast! Support Center* [online]. 21. 1. 2010 [cit. 2010-05-24]. Avast! 5.0 FAQ. Dostupné z WWW: <http://support.avast.com>.

[4] *Anvari.org, Personal blog* [online]. 5.12.2007 [cit. 2010-05-24]. The Simpsons characters picture gallery. Dostupné z WWW: <http://www.anvari.org/cols/The_Simpsons_Characters_Picture_Gallery/Homer_Simpson.html>.

[5] KNUTH, Donald E. *Umění programování : 1. díl, Základní algoritmy*. Vydání první. Brno : Computer Press a.s., 2008. 672 s. ISBN 978-80-251-2025-5.

[6] *Cramming more components into integrated circuits* [online]. 1965 [cit. 2010-05-24]. Moore's Law. Dostupné z WWW: <http://download.intel.com/museum/Moores_Law/Articles-Press_Releases/Gordon_Moore_1965_Article.pdf>.

[7] *Computer History Museum* [online]. 2010 [cit. 2010-05-24]. Fellow Awards. Dostupné z WWW: <http://www.computerhistory.org/fellowawards/hall_of_fellows.html>.

[8] BERÁNEK, Ladislav. Algoritmy a datové struktury II.. *EAMOS - výukový systém* [online]. 2010, [cit. 2010-05-24]. Dostupný z WWW: <http://www.eamos.cz>.

[9] VEČERKA, Arnošt. *Základní algoritmy*. Olomouc : Univerzita Palackého, 2004. 90 s.

[10] VIRIUS, Miroslav. *Základy algoritmizace*. Praha : ČVUT, 1995. 179 s. ISBN 80-01-01346-4.

[11] SEDGEWICK, Robert. *ALGORITHMS*: Addison-Wesley Publishing Company, Inc., 1983. 560 s. ISBN o-201-06672-6.

[12] *Google : Vedení společnosti Google* [online]. 2010 [cit. 2010-05-24]. Informace o korporaci. Dostupné z WWW: <http://www.google.com/corporate/execs.html#eric>.

[13] MICHELFEIT, Jan. *Mifeetova animace algoritmů a datových struktur* [online]. 2008 [cit. 2010-05-24]. Verze 1.1. Dostupné z WWW: <http://mifeet.alpaka.cz/algo/algorithms.swf>.

[14] HOARE, Charles Antony Richard. Quicksort. *The Computer Journal* [online]. 1962, 5, [cit. 2010-05-24]. Dostupný z WWW: <http://comjnl.oxfordjournals.org>.

[15] WILLIAMS, J. W. J. . Algorithm 232: Heapsort. *Communication of the ACM*. 1964, 7, s. 347-348.

[16] KASAHARA, Masahiro; MORISHITA, Shinichi. *Large-scale genome sequence procesing*. London : Imprerial College Press, 2006. 237 s. ISBN 1-86094-635-6.

[17] *Forbes.com* [online]. 12. 2. 2010 [cit. 2010-05-24]. The World's Billionaires. Dostupné z WWW: <http://www.forbes.com/lists/2010/10/billionaires-2010_The-Worlds-Billionaires_Rank.html>.

[18] *About.com:Urban Legends* [online]. 2010 [cit. 2010-05-24]. Larry Ellison's Speech to Yale Class of 2000. Dostupné z WWW: <http://urbanlegends.about.com/library/blellison.htm>.

[19] *Clay Mathematics Institute* [online]. 2010 [cit. 2010-05-24]. The Millennium Prize Problems. Dostupné z WWW: <http://www.claymath.org/millennium/>.

[20] *CRPC The Centrer for Research on Parallel Computation at Rice University* [online]. 2003 [cit. 2010-05-24]. CRPC Researchers Solve Traveling Salesman Problem for Record-Breaking 13,509 Cities. Dostupné z WWW: <http://www.crpc.rice.edu/newsArchive/tsp.html>.

[21] KAYE, Richard. *University of Birmingham : School of Mathematics* [online]. 2003 [cit. 2010-05-24]. Richard Kaye's Minesweeper Pages. Dostupné z WWW: <http://web.mat.bham.ac.uk/R.W.Kaye/minesw/>.

[22] FORTNOW, Lance. *The University od Chicago : Department of Computer Science* [online]. 2009 [cit. 2010-05-24]. The Status of the P versus NP Problem. Dostupné z WWW: <http://people.cs.uchicago.edu/~fortnow/papers/pnp-cacm.pdf>.

[23] NÝDL, Václav. *Diskrétní matematika I : Skriptum PF JČU*. České Budějovice : Jihočeská Universita v Českých Budějovicích, 2006. 71 s.

[24] NÝDL, Václav. *Diskrétní matematika II : Skriptum PF JČU*. České Budějovice : Jihočeská Universita v Českých Budějovicích, 2001. 86 s.

[25] MIČKA, Pavel. *Algoritmy.net* [online]. 2010 [cit. 2010-05-24]. Dostupné z WWW: <http://www.algoritmy.net>.

[26] MARTIN, David R. *Sorting Algorithm Animations* [online]. 2007 [cit. 2010-05-24]. Dostupné z WWW: <http://www.sorting-algorithms.com/>.

SEZNAM OBRÁZKŮ