3. Data a datové struktury

Složitost algoritmů a operací nad datovými strukturami

Složitost algoritmů popisuje, jak rychle (časová složitost) nebo jak náročně na paměť (prostorová složitost) roste výpočetní náročnost algoritmu v závislosti na velikosti vstupních dat. Obvykle se složitost značí pomocí „velkého O“ (například $O(n)$, $O(n^2)$, $O(\log n)$), kde n je velikost vstupu.

- **Konstantní složitost O(1):** Výpočet zabere vždy stejný čas bez ohledu na velikost vstupu (např. přístup k prvku pole podle indexu).
- **Lineární složitost O(n):** Čas narůstá přímo úměrně s velikostí vstupu (např. sekvenční hledání v poli).
- **Kvadratická složitost $O(n^2)$:** Čas roste s druhou mocninou velikosti vstupu, typické u jednoduchých třídicích algoritmů jako bubble sort.
- **Logaritmická složitost O(log n):** Typická např. pro binární vyhledávání.
- **Amortizovaná složitost** znamená průměrnou časovou složitost operace v posloupnosti více operací (například dynamická pole).

Třída NP zahrnuje problémy, jejichž správné řešení lze v polynomiálním čase ověřit, ale není známý algoritmus, který by je v polynomiálním čase řešil ve všech případech (např. problém obchodního cestujícího, batohový problém apod.).

Amortizovaná časová složitost – průměrný čas k vykonání určité operace v sekvenci operací v nejhorším případě, není to pravděpodobnost (zaručenost), pro jeho stanovení je potřeba analyzovat sekvence operací, které mohou nastat a jak často nastávají, velmi často je odvozena od algoritmů pro práci s datovými strukturami (vyhledávání ve stromové struktuře, haldě, zásobníku atd.)

Příklad:

- Implementace dynamického pole, která zdvojnásobuje velikost pole pokaždé, když dojde k jeho naplnění.

Prohledávání grafů: do šířky a do hloubky

Prohledávání do šířky (BFS – Breadth-First Search)

- Zvolíme startovní vrchol
- Postupně procházíme vrcholy grafu tak, že nejprve projdeme všechny sousedy startovního vrcholu, poté sousedy sousedů atd. až projdeme celý graf.
- Předchůdce jednotlivých vrcholů si postupně zaznamenáváme. Tím vytváříme strom nejkratších cest k jednotlivým vrcholům ze startovního vrcholu, tzv. kořene.
- Při algoritmickém řešení se využívá tzv. FIFO fronta, kam ukládáme následníky vrcholu, který je právě zpracováván.

Použití:

Najde nejkratší cestu v neohodnoceném grafu, generování stromu nejkratších cest.

Složitost:

O(|V| + |E|), kde V je počet vrcholů, E počet hran.

Prohledávání do hloubky (DFS – Depth-First Search)

- Zvolíme startovní vrchol
- Postupně procházíme vrcholy grafu tak, že najdeme prvního souseda vrcholu a hned hledáme jeho prvního souseda atd. až narazíme na vrchol, který nemá dosud nenavštíveného souseda. V tom případě se vracíme zpět k předchozímu vrcholu a pokračujeme dále až projdeme celý graf.
- Procházením stromu opět vzniká strom.
- S výhodou se využívá rekurze.
- Algoritmus se využívá např. pro nalezení silných komponent souvislosti.

Použití:

Zjištění souvislých komponent, hledání cest, topologické řazení.

Složitost:

O(|V| + |E|).

Stromy, grafy a algoritmy na kostru grafu

Strom a kostra grafu

Strom

je souvislý graf bez cyklů. Každý konec stromu se nazývá list.

Kostra grafu

je podgraf obsahující všechny vrcholy původního grafu, který je stromem (tedy nemá cykly).

Minimální kostra

(Minimum Spanning Tree, MST) je kostra s minimálním možným součtem vah hran. Hledá se pro efektivní propojení všech bodů s minimálními náklady (např. sítě, silnice).

Strom – souvislý graf, který neobsahuje kružnice, koncové vrcholy se nazývají listy

Kostra grafu – nemusí být jednoznačná u grafů, které nejsou stromy (strom má pouze jednu), úplný graf s *n* vrcholy má 2 na *n-2* různých koster

Minimální kostra – optimalizační problém, jehož cílem je najít minimální (příp. maximální) kostru hranově ohodnoceného grafu, jednoznačná, pokud neexistují dvě stejně hodnocené hrany, celková cena musí být minimální:

- Chceme dopravně propojit mezi sebou několik míst, tj. vybudovat silnice tak, aby bylo možné se z libovolného místa dostat do jiného libovolného místa a přitom náklady byly minimální.
- Chceme v obci vybudovat internetové připojení ke všem domům za co nejmenší náklady, které se odvíjí od délky pokládaného kabelu.
- Máme síť silnic. Chceme zjistit, které cesty jsou z hlediska nákladů na jejich opravy největší.
- **Hladový (Kruskalův) algoritmus**
    - Uspořádáme všechny hrany do neklesající posloupnosti dle jejich váhy.
    - Postupně vytváříme množinu hran, do které přidáváme hrany z uspořádané posloupnosti hran tak, aby nevznikla kružnice.
    - Pokud nelze žádnou hranu přidat, našli jsme minimální kostru.
    - Při hledání maximální kostry uspořádáme hrany v opačném pořadí.

Jarníkův (Primův) algoritmus

- Vycházíme z degenerovaného stromu (strom o jednom vrcholu)
- Postupně připojujeme vrcholy s hranou o nejmenší váze, k již existujícímu stromu až vyčerpáme všechny vrcholy.
- Hrany oproti Kruskalovu algoritmu neseřazujeme, ale kostru vytvářím z libovolného vrcholu.
- Výhoda algoritmu je v tom, že nemusíme kontrolovat vznik kružnic, což je výpočetně náročné.
- Při hledání maximální kostry uspořádáme hrany v opačném pořadí.

Borůvkův algoritmus

- Na začátku je každý vrchol samostatnou komponentou souvislosti.
- Postupně spojujeme komponenty souvislosti do stále větších podgrafů.
- Nakonec dostaneme jedinou komponentu souvislosti, která je minimální kostrou.
- V každém kroku vybereme pro každou komponentu souvislosti hranu s co nejnižší cenou, která směřuje do jiné komponenty souvislosti a tu přidáme do kostry.
- Při hledání maximální kostry uvažujeme největší cenu.

Optimální (minimální/maximální) cesta – hledání cesty mezi dvěma vrcholy v rámci různých kritérií

- Neorientovaný sled (též jen Sled)
    - = posloupnost vrcholů a hran v, h, v, h, v, …, h, v, jestliže každá hrana h z této posloupnosti spojuje vrcholy v a v.
- Cesta
    - = neorientovaný sled, ve kterém se žádný vrchol (a tedy ani hrana) nevyskytuje dvakrát
- **Příklady na hledání optimální cesty:**
    - Najít **nejkratší** cestu mezi dvěma městy na mapě, přičemž známe vzdálenosti mezi jednotlivými městy.
    - Najít **nejrychlejší** cestu mezi dvěma městy na mapě, přičemž známe vzdálenosti mezi jednotlivými městy.
    - Jaká bude **nejdelší** doba, za kterou zvládneme úkol (kritická cesta), pokud známe máme graf provázaných dílčích činností.
- **Dijkstrův algoritmus pro hledání minimální cesty**
    - Jednoduchý a rychlý algoritmus.
    - Existuje řada variant tohoto algoritmu.
    - Pracuje s hranově kladně ohodnoceným grafem (neohodnocený graf lze však na ohodnocený snadno převést).
- **Floydův-Warshallův algoritmus (Royův–Floydův algoritmus)** pro hledání minimální cesty
    - Jediný průchod algoritmu spočte nejkratší cestu mezi všemi dvojicemi vrcholů, tedy nejen mezi dvěma zadanými vrcholy.
    - Algoritmus je typickým příkladem dynamického programování.
    - Základní FW algoritmus vrací délky minimálních cest.
    - Modifikovaný FW algoritmus umožňuje zjistit hledané minimální cesty.
    - Algoritmus je iterační a konečný.
    - Principem algoritmu je nahrazování úseku cesty, která vede po jedné hraně, obchvatem (cestou) tvořenou dvěma hranami, pokud součet jejich ohodnocení je menší než u původní hrany.

Binární strom – struktura používaná k ukládání a vyhledávání dat, orientovaný graf s jedním kořenem, z něhož existuje cesta do všech vrcholů grafu, každý vrchol může mít maximálně dva potomky a jednoho předka s výjimkou kořene (nemá předka)

- Může být reprezentován dynamickými strukturami (ukazateli) – AVL stromy
- Existují různé druhy binárních stromů podle uspořádání a struktury

Binární vyhledávací strom (BST) – datová struktura založená na binárním stromu, s uspořádanými uzly, které umožňují rychlé vyhledávání, základní operace: MEMBER, INSERT, DELETE (v nejhorším případě je náročnost těchto operací O(n) nebo O(log n)

Vlastnosti:

- Každý uzel má nejvýše dva potomky
- Každý uzel má svůj klíč, podle hodnot klíčů jsou uzly uspořádány
- Levý podstrom uzlu obsahuje pouze klíče menší než je klíč tohoto uzlu
- Pravý podstrom uzlu obsahuje pouze klíče větší než je klíč tohoto uzlu

AVL strom – samovyvažovací BST, má stejné vlastnosti jako BST (navíc délka nejdelší větve levého a pravého podstromu se liší nejvýše o 1), stejné operace jako BST + vyvažování (O(log n)), při každém INSERT nebo DELETE se počítá koeficient vyváženosti všech uzlů z výšky obou podstromů, pokud je menší nebo roven 1 tak je vyvážený, vyvážení se děje pomocí rotace stromu, rotací je několik druhů

Halda – datová struktura, podmínka: klíč otce je větší klíč syna (max heap) nebo naopak (min heap) a strukturální podmínka na uspořádání stromu, využívá se v heapsortu, výběrový algoritmus, implementace Dijkstry (fibonacciho halda) nebo Jarník, hladové algoritmy

- **Fibonacciho halda**
    - Principiálně vychází z binomiální haldy.
    - Hlavní výhodou Fibonacciho haldy je nízká asymptotická složitost prováděných algoritmů.
    - Operace vložení, hledání minima, snížení hodnoty klíče a spojování stromů probíhají v konstantním čase, amortizovaně O(1).
    - Operace mazání a mazání minimálního prvku pracuje s časovou složitostí O(log n).
    - Oproti binomiálním haldám se liší tím, že povolujeme i jiné, než binomiální stromy ve struktuře. Spojování stromů pak provádíme jen mezi stromy stejného řádu.
    - Fibonacciho haldu tvoří skupina stromů vyhovující lokální podmínce na uspořádání haldy (Hodnota prvku je větší, než hodnota jeho otce).
    - Minimálním prvkem je vždy kořen jednoho ze stromů.
    - Vnitřní struktura Fibonacciho haldy je v porovnání s binomiální haldou daleko více flexibilní.
    - Jednotlivé stromy nemají pevně daný tvar a v extrémním případě může každý prvek haldy tvořit izolovaný strom nebo naopak všechny prvky mohou být součástí jediného stromu hloubky *N*.

Třídění a vyhledávání

Bubble sort $O(n^2)$

Porovnávání dvou sousedních prvků a pokud je nižší číslo nalevo od vyššího, pak se vymění, pokračuje se s touto logikou dále, pokud jsou ve správném pořadí, nevymění se

- **Heap sort $O(n \log n)$**

  Jedním z nejefektivnějších řadících algoritmů založených na porovnávání prvků, základem je binární halda, chová se jako prioritní fronta

  1. Postavme haldu nad zadaným polem.
  2. Utrhněme vrchol haldy (prvek s nejvyšší prioritou - nejvyšší nebo nejnižší prvek dle způsobu řazení).
  3. Prohoďme utržený prvek s posledním prvkem haldy.
  4. Zmenšeme haldu o 1 (prvky řazené dle priority na konci pole jsou již seřazené).
  5. Opravme haldu tak, aby splňovala požadované vlastnosti (přestaly platit v momentě prohození prvků).
  6. Dokud má halda prvky **GOTO: 2**.
  7. Pole je seřazené v opačném pořadí, než je priorita prvků.

Quick sort $O(n^2)$ – velmi rychlý nestabilní řadící algoritmus

Zvolme v zadaném poli libovolný prvek a říkejme mu *pivot*. Nyní můžeme pole přeházet tak, aby na jedné straně byly prvky větší než pivot, na druhé menší než pivot a pivot samotný byl umístěn přesně mezi těmito částmi. Tento postup můžeme zopakovat pro obě rozdělené části (bez pivota, ten je již umístěn na správném místě). Proceduru opakujeme tak dlouho, dokud nenarazíme na všechny triviálně řešitelné podproblémy (pole velikosti 1). V tento okamžik je celé pole seřazeno od nejvyššího prvku.

Merge sort O(n logn) – typ rozděl a panuj

1. Rozdělí neseřazenou množinu dat na dvě podmnožiny o přibližně stejné velikosti.
2. Seřadí obě podmnožiny.
3. Spojí seřazené podmnožiny do jedné seřazené množiny.

Hledání k-tého prvku –

Rozděl posloupnost délky n na n/5 pětic (poslední může být neúplná).

1. V každé pětici najdi její medián.
2. Rekurzivně najdi medián ze získané množiny □n/5□ mediánů.
3. Použij medián mediánů jako pivot k rozdělení vstupní posloupnosti.
4. Pokud medián mediánů není hledaným prvkem, tak rekurzivně hledej v množině

prvků menších, než je on nebo v množině prvků větších, než je on.

Hašování

Hašování je technika, která pomocí hašovací funkce převádí hodnotu (klíč) na index v tabulce (pole). Hašovací tabulka umožňuje rychlé vkládání, mazání i hledání průměrně v čase O(1), při správném rozložení a zamezení kolizí. Důležitá je kvalita hašovací funkce (MD5, SHA apod.), která zaručuje rovnoměrné rozložení klíčů.

Kódování

Kódování znamená převod dat do jiné, často standardizované podoby, např. převod znaků do kódování ASCII nebo UTF-8.

Cílem je jednoznačná a bezpečná reprezentace dat.

Komprese

Komprese zmenšuje velikost dat.

- **Bezztrátová komprese** (např. Huffmanovo kódování) zachovává původní data přesně, lze je obnovit do původní podoby.
- **Ztrátová komprese** (např. JPEG, MP3) některá data nenávratně odstraní pro větší úsporu místa.

Šifrování

Šifrování chrání data před neautorizovaným přístupem.

Symetrická šifra

Při **symetrickém šifrování** se pro šifrování i dešifrování používá stejný tajný klíč (například algoritmy AES nebo DES). Hlavní výhodou je rychlost, nevýhodou potřeba bezpečně sdílet klíč.

Asymetrická šifra (RSA)

Asymetrické šifrování využívá dvojici klíčů: veřejný klíč (pro šifrování) a soukromý klíč (pro dešifrování). Nejznámější algoritmus je **RSA**. Veřejný klíč lze bezpečně sdílet, soukromý zůstává skrytý. Výpočetně náročnější, často se používá v kombinaci se symetrickými šiframi (například pro výměnu klíčů).