

Algoritmy a datové struktury

Studijní materiál ke státnicím (I2)

Jakub Klesa

Pokryté předměty: NTIN060 (ADS 1), NTIN061 (ADS 2)

Obsah

Úvod	2
1 Časová a prostorová složitost, asymptotická notace	2
1.1 Model výpočtu a měření složitosti	2
1.2 Asymptotická notace	3
2 Třídy složitosti: P, NP, převoditelnost, NP-úplnost	4
2.1 Rozhodovací problémy, třídy P a NP	4
2.2 Převoditelnost, NP-těžkost a NP-úplnost	5
3 Metoda rozděl a panuj (Master theorem)	7
3.1 Rekurentní rovnice a Master theorem	7
3.2 Aplikace I: třídění sléváním (Mergesort) a halda	8
3.3 Aplikace II: násobení dlouhých čísel (Karacuba)	9
3.4 Aplikace III: medián a Quickselect	10
4 Binární vyhledávací stromy (BST, AVL)	11
4.1 Binární vyhledávací strom a jeho operace	11
4.2 AVL stromy	11
5 Třídění: primitivní algoritmy, Quicksort, dolní odhad	14
5.1 Primitivní třídící algoritmy	14
5.2 Quicksort	14
5.3 Dolní odhad složitosti porovnávacího třídění	15
6 Grafové algoritmy	16
6.1 Prohledávání do šířky (BFS) a do hloubky (DFS)	16
6.2 Topologické třídění orientovaných grafů	17
6.3 Nejkratší cesty v ohodnocených grafech	17
6.4 Minimální kostra (Jarník, Borůvka)	19
6.5 Toky v sítích (Ford–Fulkerson)	19
Shrnutí	20
Zdroje	22

Úvod

Tento materiál pokrývá okruh **I2 – Algoritmy a datové struktury** (předměty NTIN060 *Algoritmy a datové struktury 1* a NTIN061 *Algoritmy a datové struktury 2*). Výklad stojí na jediném, ale úplném zdroji z MFF UK na úrovni obou kurzů: na učebnici M. Mareše a T. Vally *Průvodce labyrintem algoritmů* (2. vydání, edice CZ.NIC, 2022), kterou přednášející obou předmětů uvádějí jako hlavní studijní text a na kterou se odvolává i oficiální náčrt řešení státnicových otázek. Značení, hloubka i konvence (pseudokód, model RAM, asymptotická notace) odpovídají této učebnici.

Role důkazů. Požadavky okruhu kladou důraz na *porozumění a konstrukci*: navrhnout nebo odsimulovat algoritmus na konkrétním vstupu, sestavit a vyřešit rekurentní rovnici, určit asymptotickou složitost, případně dát protipříklad. Pojmenované věty proto uvádíme se zněním, intuicí a krátkým odvozením tam, kde pomáhá pochopení. Požadavky výslovně uvádějí jedinou větu *bez důkazu*, **Master theorem (kuchařkovou větu)**: stačí přesné znění, tři případy a aplikace na rekurenci. Naopak korektnost a složitost algoritmu typu *rozděl a panuj se odvozuje* (sestavit rekurenci a vyřešit ji), viz otázka léto 2024. **Dolní odhad** $\Omega(n \log n)$ pro porovnávací třídění je krátký argument přes rozhodovací strom a uvádíme ho celý (využívá ho otázka jaro 2023).

Do výkladu je zařazeno všech pět reálných otázek SZZ k okruhu z let 2022–2025 jako řešené příklady; jsou i v samostatném dokumentu *Příklady otázek*. Použité zdroje jsou uvedeny na konci.

Značení. Velikost vstupu značíme n (počet prvků posloupnosti, počet vrcholů grafu apod.), u grafů navíc m pro počet hran. *Časovou složitost* algoritmu chápeme jako počet elementárních operací modelu RAM v *nejhorším případě* jako funkci n ; *prostorovou složitost* jako počet použitých paměťových buněk. Logaritmus bez základu znamená $\log_2$ (na základu nezáleží, liší se jen konstanta-krát). Graf je dvojice $G = (V, E)$, $|V| = n$, $|E| = m$; orientovanou hranu z u do v píšeme uv . Strom kreslíme kořenem nahoru. *Hloubkou* vrcholu v rozumíme *délku nejdelší cesty z v dolů do listu* a značíme ji $h(v)$ (list má $h = 0$, prázdný strom $h = -1$); *hloubka stromu* je h jeho kořene. (Pozor na konvenci: tutéž veličinu jinde někdy najdete pod názvem „výška“, kdežto „hloubkou vrcholu“ se občas myslí vzdálenost *od* kořene – zde vždy měříme cestu *dolů k listu*.)

1 Časová a prostorová složitost, asymptotická notace

První otázkou u každého algoritmu je „jak je rychlý a kolik paměti spotřebuje“. Měřit to vteřinami na konkrétním počítači nedává smysl – výsledek by závisel na stroji, jazyce i operačním systému. Proto zavádíme abstraktní míru: počítáme *elementární operace* idealizovaného stroje a zajímá nás, jak jejich počet *roste* s velikostí vstupu.

1.1 Model výpočtu a měření složitosti

Definice – Výpočetní model RAM, elementární operace

RAM (Random Access Machine) je idealizovaný počítač s pamětí tvořenou očíslovanými buňkami, do nichž lze v konstantním čase číst i zapisovat podle adresy. *Elementární operací* rozumíme jednu instrukci procesoru: aritmetiku (+, −, ·, /, mod), porovnání, logické a bitové operace, řízení toku (skok, podmíněný skok) a přístup do paměti. Každá elementární operace trvá jednu časovou jednotku.

 **Intuice.** Konkrétní výběr sady elementárních operací změní dobu výpočtu nejvýše konstanta-krát, a konstanty stejně zanedbáváme. Proto nezáleží na detailech procesoru; podstatný je jen *řád růstu* počtu operací. (U čísel s mnoha ciframi je třeba být opatrný – pak se počítá s tzv. logaritmickou cenou podle počtu bitů; v běžných úlohách s čísly velikosti polynomiální v n má každá operace cenu konstantní.)

Definice – Časová a prostorová složitost, tři případy

Nechť algoritmus na vstupu velikosti n provede pro každý konkrétní vstup nějaký počet operací. Časová složitost v nejhorším případě $f(n)$ je maximum tohoto počtu přes všechny vstupy velikosti n ; v nejlepším případě minimum; v průměrném případě aritmetický průměr (resp. střední hodnota) přes všechny vstupy velikosti n . Prostorová složitost je maximální počet paměťových buněk použitých v jednom okamžiku výpočtu.

i Poznámka. Není-li řečeno jinak, „složitosti“ myslíme nejhorší případ – dává záruku „nejhůře tolik a tolik“. Průměrný případ je užitečný tam, kde se nejhorší případ vyskytuje vzácně (typicky Quicksort, sekce 5).

i Poznámka. Měření velikosti dat. Velikost vstupu n udáváme v přirozené míře dané úlohy – počet prvků posloupnosti, počet vrcholů a hran grafu, případně počet bitů (číslíc) zápisu čísla. Volba míry je podstatná: u číselných úloh se velikost měří délkou zápisu, nikoli hodnotou čísla. Jinak bychom dostali klamně příznivé odhady – např. „vyzkoušet všechny dělitele do x “ vypadá jako $O(x)$, ale vůči počtu číslíc $\approx \log_{10} x$ je to exponenciální (viz příklad o rozkladu čísla v sekci 2).

1.2 Asymptotická notace

Přesný počet operací (např. $2n^2 + 3n + 1$) je nepraktický a stejně závisí na konstantách, které neumíme měřit. Zajímá nás chování pro velká n a až na konstantní násobek. K tomu slouží trojice symbolů O , Ω , Θ . Obrat „pro skoro všechna n “ znamená „pro všechna n od nějakého n_0 výše“ (konečně mnoho výjimek nevadí).

Definice – Asymptotická notace O , Ω , Θ

Nechť $f, g : \mathbb{N} \rightarrow \mathbb{R}^+$. Pro „skoro všechna n “ (tj. pro všechna n od nějakého n_0 výše) definujeme:

$$f \in O(g) \iff \exists c > 0 : f(n) \leq c g(n) \quad (\text{horní odhad}),$$

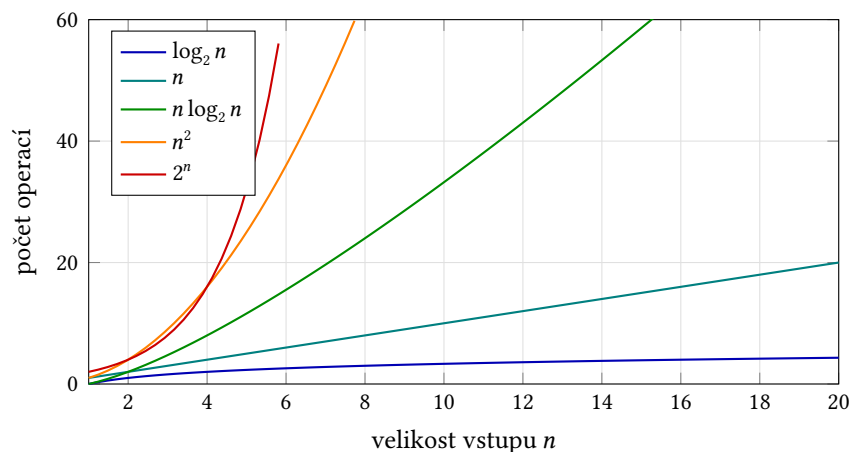
$$f \in \Omega(g) \iff \exists c > 0 : f(n) \geq c g(n) \quad (\text{dolní odhad}),$$

$$f \in \Theta(g) \iff f \in O(g) \wedge f \in \Omega(g) \quad (\text{těsný odhad}).$$

Ekvivalentně je $f \in \Theta(g)$ právě tehdy, když $c_1 g(n) \leq f(n) \leq c_2 g(n)$ pro vhodné konstanty $c_1, c_2 > 0$ a skoro všechna n .

💡 Intuice. O je obdoba „ $\leq$ “, Ω obdoba „ $\geq$ “ a Θ obdoba „ $=$ “ – vše ovšem až na konstanty a počáteční chování. Zápis $f = O(g)$ je zaužívaný, ale je to nerovnost, ne rovnost (nelze ji „obrátit“). Θ nese nejvíc informace; mnohdy ale umíme dokázat jen horní odhad O .

Praktická pravidla (plynou přímo z definice): v součtu rozhoduje nejrychleji rostoucí člen ($n^2 + n = \Theta(n^2)$), multiplikativní konstanty se zahazují ($5n = \Theta(n)$), polynom stupně d je $\Theta(n^d)$, a platí $\log n = O(n^\epsilon)$ pro každé $\epsilon > 0$ a $n^d = O(2^n)$. Následující obrázek ukazuje, proč na řádu růstu tolik záleží.



růst běžných složitostních funkcí. Pro $n = 20$ je $\log_2 n \approx 4,3$, $n \log_2 n \approx 86$, $n^2 = 400$, ale $2^n \approx 10^6$ (vyběhlo z grafu). Lineární a lineární algoritmy zvládnou obří vstupy; kvadratické jsou na hranici; exponenciální jsou nepoužitelné už pro několik desítek prvků. Zrychlení stroje tisíckrát posune zvládnutelnou velikost u exponenciálního algoritmu jen o ≈ 10 prvků.

Příklad – Určení složitosti dvojitého cyklu

Zadání. Určete časovou složitost fragmentu: pro $i = 1, \dots, n$ a uvnitř pro $j = i, \dots, n$ proved konstantní práci.

Řešení. Vnitřní tělo se provede $\sum_{i=1}^n (n - i + 1) = n + (n - 1) + \dots + 1 = \frac{n(n+1)}{2}$ krát. To je $\frac{1}{2}n^2 + \frac{1}{2}n$; nejrychleji rostoucí člen je n^2 , konstantu $\frac{1}{2}$ zahodíme, takže složitost je $\Theta(n^2)$. Prostorová složitost je $\Theta(1)$ (jen pár indexů).

2 Třídy složitosti: P, NP, převoditelnost, NP-úplnost

Dosud jsme měřili složitost konkrétních algoritmů. Teď se ptáme na složitost *problémů*: existuje pro daný problém efektivní algoritmus vůbec? Hranicí „efektivnosti“ je polynomiální čas. Klíčové je, že u mnoha praktických úloh efektivní algoritmus *neznáme*, ale ani neumíme dokázat, že neexistuje – a tyto úlohy jsou navzájem „stejně těžké“ ve smyslu převoditelnosti. To je obsah teorie NP-úplnosti.

2.1 Rozhodovací problémy, třídy P a NP

Definice – Rozhodovací problém / jazyk

Rozhodovací problém je úloha s odpovědí ano/ne. Vstup zakódujeme jako řetězec nad abecedou $\{0, 1\}$; problém pak ztotožníme s jazykem $L \subseteq \{0, 1\}^*$ všech vstupů, na něž je odpověď ano. *Velikostí vstupu* x je délka řetězce $|x|$.

Definice – Třída P

Problém L leží v třídě **P**, existuje-li algoritmus A a polynom p tak, že pro každý vstup x algoritmus A doběhne v čase nejvýše $p(|x|)$ a vydá správnou odpověď (tj. $A(x) = 1 \Leftrightarrow x \in L$). Neformálně: **P** jsou problémy *řešitelné v polynomiálním čase*.

Definice – Třída NP

Problém L leží v třídě **NP**, existuje-li problém $K \in \mathbf{P}$ (*verifikátor*) a polynom g tak, že pro každý vstup x platí

$$x \in L \Leftrightarrow \exists y, |y| \leq g(|x|) : K(x, y) = 1.$$

Řetězci y se říká *certifikát* (svědek). Neformálně: **NP** jsou problémy, u nichž lze *ověřit* předložené řešení v polynomiálním čase.

Intuice. Rozdíl P vs. NP je „najít řešení“ vs. „ověřit hotové řešení“. U SAT (splnitelnost formule) neumíme rychle splňující ohodnocení *najít*, ale dostaneme-li nějaké, snadno ho *ověříme*. Platí $P \subseteq NP$ (kdo umí řešit, umí i ověřit – certifikát ignoruje). Zda je inkluze ostrá, tedy zda $P \neq NP$, je nejslavnější otevřený problém informatiky.

2.2 Převoditelnost, NP-těžkost a NP-úplnost

Definice – Polynomiální převod (redukce)

Problém A se *polynomiálně převádí* na B (píšeme $A \leq B$, případně $A \rightarrow B$), existuje-li funkce $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ spočitatelná v polynomiálním čase taková, že pro každý vstup x platí

$$x \in A \iff f(x) \in B.$$

Intuice. $A \leq B$ znamená „ B je aspoň tak těžký jako A “ – obtížnost teče po šipce. Kdybychom uměli rychle řešit B , uměli bychom rychle řešit i A (vstup x přeložíme na $f(x)$ a zeptáme se na B). Převod je *reflexivní* a *tranzitivní* ($A \leq B \leq C \Rightarrow A \leq C$), tvoří tedy kvaziuspořádání „podle obtížnosti“.

★ Lemma – Dědičnost snadnosti

Je-li $A \leq B$ a $B \in P$, pak $A \in P$. Důsledkem (obměnou): je-li $A \leq B$ a $A \notin P$, pak ani B neleží v P , tj. $B \notin P$.

Definice – NP-těžkost a NP-úplnost

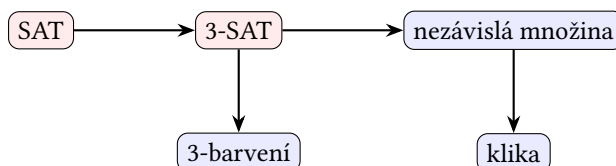
Problém L je **NP-těžký**, pokud se na něj polynomiálně převádí *každý* problém z **NP**. Je-li navíc $L \in NP$, je L **NP-úplný**. NP-úplné problémy jsou tedy „nejtěžší v NP“.

★ Věta – Cookova–Levinova, bez důkazu

Problém SAT (splnitelnost výrokové formule v CNF) je NP-úplný.

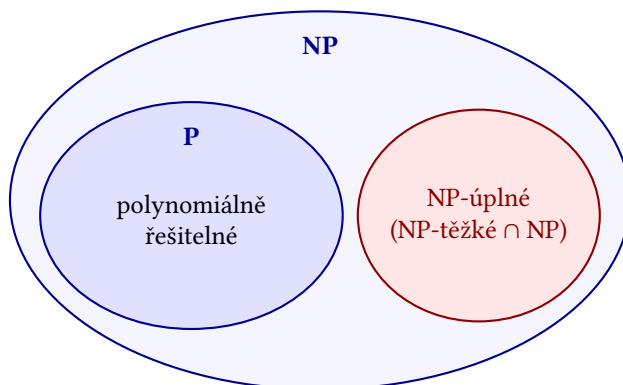
Intuice. Cookova věta dává „první“ NP-úplný problém. Další NP-úplnost se pak dokazuje *převodem*: chci-li ukázat, že $M \in NP$ je NP-úplný, stačí (1) ověřit $M \in NP$ a (2) převést nějaký už známý NP-úplný problém L na M ($L \leq M$). Z tranzitivity pak vše z $NP \leq L \leq M$.

Poznámka. Mezi klasické NP-úplné problémy patří SAT, 3-SAT, nezávislá množina, klika, k -barvení grafu ($k \geq 3$), Hamiltonovská kružnice, batoh a součet podmnožiny. Šipky níže ukazují směr typických *převodů* mezi nimi; jeden z nich, 3-SAT $\rightarrow$ nezávislá množina, si vzápětí rozebereme jako příklad:



směr převodů $A \rightarrow B$ čteme „obtížnost teče po šipce“ (B je aspoň tak těžký jako A). Cookova věta dá NP-úplnost SAT; každá další šipka od už NP-úplného problému přenáší dál NP-těžkost (tranzitivita převodu) a spolu s příslušností cíle do NP i NP-úplnost.

Vztahy tříd za předpokladu $P \neq NP$ shrnuje obrázek (kdyby $P = NP$, všechny tři oblasti by splynuly, až na triviální jazyky $\emptyset$ a $\{0, 1\}^*$, které NP-úplné být nemohou):



$P \subseteq NP$; NP-úplné problémy leží v NP, ale (za předpokladu $P \neq NP$) mimo P. NP-těžké problémy mohou ležet i mimo NP (nemusí být ověřitelné v polynomiálním čase).

Příklad — Konkrétní převod: 3-SAT $\rightarrow$ nezávislá množina

Zadání. Ukažte, že problém NEZÁVISLÁ MNOŽINA (existuje v grafu G množina aspoň k vrcholů, z nichž žádné dva nespojuje hrana?) je NP-úplný. Využijte NP-úplnosti 3-SAT.

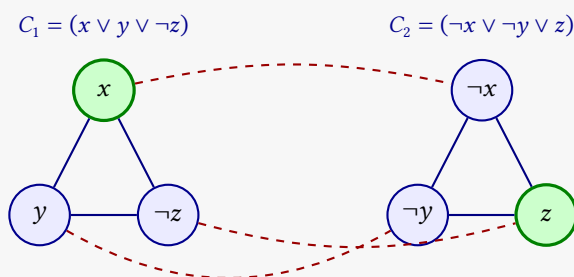
Řešení.

(1) **Příslušnost do NP.** Certifikátem je samotná k -prvková množina vrcholů; ověřit, že žádné dva z nich nespojuje hrana, zvládneme v polynomiálním čase. Tedy NEZÁVISLÁ MNOŽINA $\in NP$.

(2) **Převod 3-SAT $\leq$ NEZÁVISLÁ MNOŽINA.** Z formule φ v 3-CNF s m klauzulemi sestrojíme graf G :

- každou klauzuli nahradíme *trojúhelníkem* – tři vrcholy (po jednom za každý literál klauzule) navzájem pospojované hranami;
- mezi vrcholy z *různých* trojúhelníků přidáme hranu vždy, když jsou jejich literály *kontradiktorní* (tvaru x_i a $\neg x_i$);
- položíme $k = m$.

Konstrukce je zjevně polynomiální ($3m$ vrcholů, nejvýše $O(m^2)$ hran).



převod formule $\varphi = (x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z)$ na graf. Každá klauzule je trojúhelník (plné hrany); kontradiktorní literály z různých klauzulí spojuje *čárkovaná* hrana ($x-\neg x$, $y-\neg y$, $z-\neg z$). Hledáme nezávislou množinu velikosti $k = m = 2$: zelené vrcholy $\{x, z\}$ jí jsou (z každého trojúhelníku jeden, navzájem nespojené) a odpovídají splňujícímu ohodnocení $x = z = 1$.

Korektnost: φ je splnitelná $\Leftrightarrow G$ má nezávislou množinu velikosti m .

- ($\Rightarrow$) Ze splňujícího ohodnocení vyber v každé klauzuli jeden *pravdivý* literál a jeho vrchol. Z každého trojúhelníku tak bereme právě jeden vrchol (vnitřní hranu neporušíme) a dva vybrané vrcholy nemohou být kontradiktorní (oba literály jsou pod týmž ohodnocením pravdivé), takže neporušíme ani hranu mezi trojúhelníky. Vznikne nezávislá množina velikosti m .
- ($\Leftarrow$) Nezávislá množina velikosti m musí z každého trojúhelníku brát *právě jeden* vrchol (trojúhelník je úplný, dva by spojovala hrana). Vybrané literály nastav na *pravdu*; je to bezesporné, neboť dva kontradiktorní literály spojuje hrana, takže oba ve množině být nemohou. Každá klauzule má vybraný, a tedy pravdivý literál, takže φ je splněná.

Protože 3-SAT je NP-úplný a $3\text{-SAT} \leq \text{NEZÁVISLÁ MNOŽINA}$, je NEZÁVISLÁ MNOŽINA NP-těžká; spolu s bodem (1) je tedy **NP-úplná**.

Příklad — SZZ podzim 2025 č. 1: Třídy složitosti (P/NP, redukce, NP-úplnost)

(Reálná otázka SZZ: podzim 2025, č. 1 – Třídy složitosti (P/NP, redukce, NP-úplnost))

Zadání.

1. Definujte třídy složitosti **P** a **NP**.
2. Definujte převod (redukci) mezi rozhodovacími problémy.
3. Definujte **NP-úplnost** rozhodovacího problému.
4. Rozhodněte, zda následující problém leží v **NP**, a zdůvodněte: je dáno přirozené číslo x v desítkové soustavě; existují přirozená a, b s $1 < a, b < x$, $x = a \cdot b$ a desítkové zápisy a, b (bez počátečních nul) jsou stejně dlouhé?

Řešení.

(1)–(3) Přesně definice výše: **P** jsou problémy řešitelné v polynomiálním čase; **NP** problémy s polynomiálně ověřitelným certifikátem; převod $A \leq B$ je polynomiální f s $x \in A \Leftrightarrow f(x) \in B$; L je NP-úplný, je-li NP-těžký (vše z NP se na něj převádí) a zároveň $L \in \text{NP}$.

(4) **Velikost vstupu** je počet číslic x , tedy $\approx \log_{10} x$ (ne hodnota x !). Tvrdíme, že problém *leží v NP*. Jako **certifikát** vezmeme jeden z činitelů a . Verifikátor:

1. zkontroluje, že a je číslo s $1 < a < x$ a že $a \mid x$; spočte $b = x/a$ (dělení se zbytkem – při nenulovém zbytku zamítne),
2. ověří $1 < b < x$ a že desítkové zápisy a a b mají stejnou délku.

Délka certifikátu a je nejvýše délka x , tedy *lineární* ve velikosti vstupu. Dělení, porovnání a měření délky zápisů jsou polynomiální v počtu číslic. Tedy existuje polynomiální verifikátor a problém je v **NP**.

i Poznámka. Pozor na typickou past: kdybychom „velikost“ měřili hodnotou x , vyšel by zkoušení dělitelů zdánlivě „polynomiální“ čas $O(x)$. Vůči *počtu číslic* je ale x exponenciální, takže naivní faktorizace polynomiální není. To, že problém je v NP, plyne z existence krátkého certifikátu, ne z rychlého řešení.

3 Metoda rozděl a panuj (Master theorem)

Rozděl a panuj je návrhová technika: úlohu rozložíme na několik menších podúloh téhož typu, ty vyřešíme rekurzivně a jejich výsledky spojíme. Časová složitost se pak řídí *rekurentní rovnicí*, kterou typicky vyřeší Master theorem.

3.1 Rekurentní rovnice a Master theorem

Algoritmus, který vstup velikosti n rozdělí na a podproblémů velikosti n/b a rozdělení i spojení zvládne v čase $\Theta(n^c)$, má složitost danou rovnicí

$$T(n) = aT(n/b) + \Theta(n^c), \quad T(1) = \Theta(1).$$

★ Věta — Master theorem / Kuchařková věta, bez důkazu

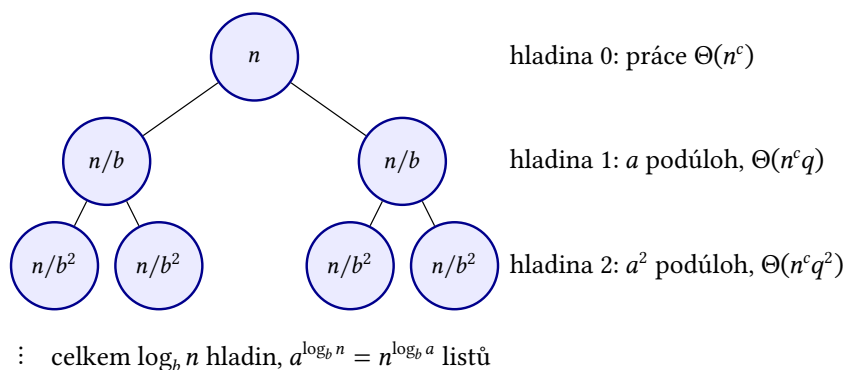
Nechť $a \geq 1$, $b > 1$, $c \geq 0$ jsou konstanty a $T(n) = aT(n/b) + \Theta(n^c)$. Označme $q = a/b^c$. Pak

$$T(n) = \begin{cases} \Theta(n^c), & q < 1 \quad (\text{tj. } c > \log_b a), \\ \Theta(n^c \log n), & q = 1 \quad (\text{tj. } c = \log_b a), \\ \Theta(n^{\log_b a}), & q > 1 \quad (\text{tj. } c < \log_b a). \end{cases}$$

Intuice. Strom rekurze má na každém vrcholu a synů a $\log_b n$ hladin. Na i -té hladině je a^i podproblémů velikosti n/b^i , dohromady práce $a^i \cdot \Theta((n/b^i)^c) = \Theta(n^c q^i)$. Sečteme geometrickou řadu s kvocientem $q = a/b^c$:

- $q < 1$: dominuje kořen (vrchní hladina), práce $\Theta(n^c)$;
- $q = 1$: všechny hladiny stejně, $\log_b n$ hladin krát $\Theta(n^c)$, tedy $\Theta(n^c \log n)$;
- $q > 1$: dominují listy, kterých je $a^{\log_b n} = n^{\log_b a}$.

Rozhoduje tedy souboj mezi „práce na spojení“ (n^c) a „počet listů“ ($n^{\log_b a}$).



strom rekurze pro $T(n) = aT(n/b) + \Theta(n^c)$ (zde $a = 2, b = 2$). Součet prací po hladinách je geometrická řada s kvocientem $q = a/b^c$; který konec řady dominuje, určuje tři případy Master theoremu.

3.2 Aplikace I: třídění sléváním (Mergesort) a halda

Algoritmus – MERGESORT – třídění sléváním

1. Pokud $n \leq 1$, vrať vstup. ▷ triviální případ
2. Rozděľ posloupnost na dvě poloviny délky $n/2$.
3. Každou polovinu setříd' rekurzivně voláním MERGESORT.
4. Slij dvě setříděné poloviny do jedné (procedura MERGE). ▷ $\Theta(n)$

Procedura MERGE porovnává první (nejmenší) prvky obou setříděných posloupností, menší z nich přesune na výstup a pokračuje, dokud jednu nevyprázdní; pak zbytek druhé doplní. Každý prvek se přesune právě jednou, takže slévání dvou posloupností o celkem n prvcích trvá $\Theta(n)$.

Rekurence je $T(n) = 2T(n/2) + \Theta(n)$, tj. $a = 2, b = 2, c = 1, q = 2/2^1 = 1$ – případ $q = 1$, takže

$$T(n) = \Theta(n \log n).$$

Prostorová složitost je $\Theta(n)$ (pomocné pole na slévání). Mergesort je *stabilní* (zachová pořadí stejných klíčů, slévá-li se při rovnosti zleva) a jeho $\Theta(n \log n)$ platí vždy, i v nejhorším případě.

Pro slévání více posloupností najednou se hodí *prioritní fronta* realizovaná *haldou*; využijeme ji i později u Dijkstry a Jarníka.

Definice – Binární halda (prioritní fronta)

Binární (minimová) halda je úplný binární strom uložený v poli, v němž platí *haldová podmínka*: klíč každého vrcholu je nejvýše klíč jeho synů (kořen je tedy globální minimum). Podporuje operace INSERT (vložit prvek) a EXTRACTMIN (odeber minimum), obě v čase $O(\log n)$, a MIN (přečti minimum) v $O(1)$.

Příklad — SZZ léto 2023 č. 1: Třídění sléváním (Mergesort, slévání k posloupností)

(Reálná otázka SZZ: léto 2023, č. 1 – Třídění sléváním (Mergesort, slévání k posloupností))

Zadání.

1. Formulujte Mergesort v pseudokódu.
2. Analyzujte jeho časovou a prostorovou složitost.
3. Navrhněte efektivní algoritmus pro slévání k setříděných posloupností o celkem n prvcích. Upravte Mergesort, aby toto slévání používal. Jak se změní časová složitost (jako funkce n a k)?

Řešení.

(1) Pseudokód viz algoritmus MERGESORT výše (rozděl na poloviny, seřaď rekurzivně, slij).

(2) Rekurence $T(n) = 2T(n/2) + \Theta(n)$ dává $\Theta(n \log n)$ času (Master, případ $q = 1$); prostor $\Theta(n)$ na pomocné pole.

(3) **Slévání k posloupností.** Udržujeme *minimovou haldu* s k prvky – po jednom „čele“ z každé posloupnosti. Opakovaně: vytáhni minimum z haldy (EXTRACTMIN), zapiš ho na výstup a na jeho místo vlož další prvek z téže posloupnosti (INSERT). Provedeme celkem n dvojic operací, každá $O(\log k)$ (halda má nejvýše k prvků), takže

$$\text{slévání } k \text{ posloupností} = \boxed{\Theta(n \log k)}.$$

k -cestný Mergesort. Místo dělení na 2 části dělíme na k částí a sléváme je výše uvedeným k -cestným sléváním. Rekurence je $T(n) = kT(n/k) + \Theta(n \log k)$. Strom rekurze má $\log_k n$ hladin, na každé je celková práce slévání $\Theta(n \log k)$, tedy

$$T(n) = \Theta(n \log k \cdot \log_k n) = \Theta\left(n \log k \cdot \frac{\log n}{\log k}\right) = \boxed{\Theta(n \log n)}.$$

Závěr: faktor k se vykrátí – asymptotická složitost zůstává $\Theta(n \log n)$ nezávisle na k . (Volba většího k může v praxi snížit počet průchodů paměti, ale řád nezmění.)

i Poznámka. Samotné slévání k posloupností je $\Theta(n \log k)$ a je to optimální: rozhodnout, ze které z k posloupností pochází každý další prvek, vyžaduje v porovnávacím modelu $\Omega(n \log k)$ porovnání (analogie dolního odhadu třídění, sekce 5).

3.3 Aplikace II: násobení dlouhých čísel (Karacuba)

Násobení dvou n -ciferných čísel „školním“ postupem stojí $\Theta(n^2)$. Karacubův algoritmus to zrychlí. Čísla X, Y rozdělíme na horní a dolní polovinu, $X = A \cdot 10^{n/2} + B$, $Y = C \cdot 10^{n/2} + D$. Pak

$$XY = AC \cdot 10^n + (AD + BC) \cdot 10^{n/2} + BD.$$

Naivně to jsou 4 násobení poloviční délky. Karacubův trik využije identitu

$$AD + BC = (A + B)(C + D) - AC - BD,$$

takže stačí jen *tři* násobení: AC , BD a $(A + B)(C + D)$. Rekurence je

$$T(n) = 3T(n/2) + \Theta(n), \quad a = 3, \quad b = 2, \quad c = 1, \quad q = \frac{3}{2^1} = \frac{3}{2} > 1,$$

tedy případ $q > 1$ a $T(n) = \Theta(n^{\log_2 3}) \approx \Theta(n^{1.585})$ – výrazně lepší než n^2 .

3.4 Aplikace III: medián a Quickselect

Příklad — SZZ léto 2024 č. 1: Rozděl a panuj (Master theorem, medián $2n$ čísel)

(Reálná otázka SZZ: léto 2024, č. 1 – Algoritmy rozděl a panuj (Master theorem, medián $2n$ čísel))

Zadání.

1. Rekurzivní algoritmus A na datech velikosti n ($a, c, x, y \in \mathbb{N}$, $a > 0$, $c > 1$): (a) v čase $\Theta(n^x)$ vybere a podmnožin velikosti n/c ; (b) rekurzivně se zavolá na každou; (c) v čase $\Theta(n^y)$ spojí výsledky. Určete (bez důkazu) těsný odhad $T(n)$ v závislosti na a, c, x, y .
2. Pole X, Y délky n obsahují seříděné posloupnosti n přirozených čísel. Navrhněte algoritmus s časem $O(\log n)$, který najde medián všech $2n$ čísel, a metodou z bodu 1) dokažte složitost.

Řešení.

(1) Společná režie kroků (a) a (c) je $\Theta(n^x + n^y) = \Theta(n^{\max\{x, y\}})$. Položíme $d = \max\{x, y\}$ a dostaneme rekurenci

$$T(n) = aT(n/c) + \Theta(n^d).$$

(Pozn. ke značení: oproti našemu znění Master theoremu zde hraje roli dělitele písmeno c , tedy $c \leftrightarrow b$, a exponent je $d \leftrightarrow$ naše c .) Podle Master theoremu s poměrem a/c^d :

$$T(n) = \begin{cases} \Theta(n^d), & a/c^d < 1, \\ \Theta(n^d \log n), & a/c^d = 1, \\ \Theta(n^{\log_c a}), & a/c^d > 1. \end{cases}$$

(2) **Algoritmus (binární vyhledávání mediánu).** Porovnáme „prostřední“ prvky obou polí, např. $x = X[\lfloor (n-1)/2 \rfloor]$ a $y = Y[\lfloor (n-1)/2 \rfloor]$ (indexujeme od 0):

- Je-li $x < y$: odřízneme prvky vlevo od x (je jich $\lfloor (n-1)/2 \rfloor$) a stejný počet prvků vpravo od y .
Klíčové: z X a z Y odebereme stejný počet prvků – odebrané z X nejsou větší než dolní medián, odebrané z Y nejsou menší než horní medián (zdůvodnění níže), takže prostřední dvojice zbylých prvků je též jako u původní množiny. Voláme rekurzi na zbytek.
- Je-li $x > y$: symetricky odřízneme prvky vlevo od y a stejný počet prvků vpravo od x .
- Je-li $x = y$: x je (jeden z) medián(ů), konec.

Pro $n < 3$ dopočítáme medián přímo v konstantním čase (pro $n = 2$ by krok nic neodebral).

Proč odřezání nemění medián (případ $x < y$, $k = \lfloor (n-1)/2 \rfloor$; seříděné všech $2n$ čísel označme $z_1 \leq \dots \leq z_{2n}$, mediány jsou z_n a z_{n+1}). Vezměme odebraný prvek $X[i]$, $i < k$. Čísla $\geq X[i]$ jsou aspoň všechna v $X[i..n-1]$ ($n-i \geq n-k+1$ čísel) a v $Y[k..n-1]$ ($n-k$ čísel, všechna $\geq y > x \geq X[i]$), celkem aspoň $2n - 2k + 1 \geq n + 2$. Proto $X[i] \leq z_{n-1} \leq z_n$. Symetricky pro odebraný $Y[j]$, $j \geq n-k$: čísla $\leq Y[j]$ jsou aspoň všechna v $X[0..k]$ a $Y[0..j]$, celkem aspoň $(k+1) + (n-k+1) = n+2$, tedy $Y[j] \geq z_{n+2} \geq z_{n+1}$. Odebereme-li k čísel „zdola“ a k „shora“, prostřední dvojice z_n, z_{n+1} zůstane uprostřed. (Ověřeno i náhodnými testy včetně opakovaných hodnot.)

Složitost metodou z bodu 1). V každém kroku uděláme konstantní práci (přístup k dvěma prvkům, porovnání, posun indexů) a zavoláme *jednu* rekurzi na poloviční vstup:

$$T(n) = 1 \cdot T(n/2) + \Theta(1), \quad a = 1, c = 2, d = 0.$$

Poměr $a/c^d = 1/2^0 = 1$ – případ $q = 1$, tedy $T(n) = \Theta(n^0 \log n) = \boxed{\Theta(\log n)}$.

i Poznámka. Příbuzná úloha je hledání k -tého nejmenšího prvku *nesetříděné* posloupnosti. **Quickselect** ji řeší jako Quicksort s jedinou rekurzí: rozdělí prvky podle pivotu a pokračuje jen v části, kde k -tý prvek leží; s náhodným pivotem má očekávaný čas $\Theta(n)$, v nejhorším případě $\Theta(n^2)$. **LinearSelect** (metoda „medián mediánů“) zaručí lineární čas i v nejhorším případě: rozdělí prvky do pětic, spočte medián každé, rekurzivně najde medián těchto mediánů a použije ho jako pivot. Ten zaručí,

že obě části mají zhruba nejvýše $\frac{7}{10}n$ prvků, takže $T(n) = T(n/5) + T(7n/10) + \Theta(n) = \Theta(n)$ (součet $\frac{1}{5} + \frac{7}{10} < 1$ dá geometrickou řadu; není to čistý tvar pro Master theorem).

4 Binární vyhledávací stromy (BST, AVL)

Vyhledávací strom je datová struktura pro *dynamickou množinu* klíčů s operacemi hledání, vkládání a mazání. Klíče drží uspořádané tak, že operace jdou po jediné cestě od kořene – jejich cena je tedy úměrná *hloubce* stromu. Celá hra je o to udržet hloubku malou ($\Theta(\log n)$), což řeší vyvážené stromy (AVL).

4.1 Binární vyhledávací strom a jeho operace

Definice – Binární vyhledávací strom (BST)

Binární vyhledávací strom je binární strom, jehož každý vrchol v nese klíč $k(v)$, a to tak, že pro každý vrchol platí *uspořádací vlastnost*: všechny klíče v levém podstromu v jsou $< k(v)$ a všechny klíče v pravém podstromu jsou $> k(v)$.

 **Intuice.** Vrchol „rozděluje“ zbytek na menší (vlevo) a větší (vpravo). Hledání klíče x proto jde jednoznačně: porovnej x s $k(v)$ a podle výsledku jdi doleva, doprava, nebo jsi našel. Symetrický (in-order) průchod – nejprve levý podstrom, pak vrchol, pak pravý – vypíše klíče *vzestupně seříděné*.

Operace FIND, INSERT i DELETE sledují od kořene cestu danou porovnáváním klíčů:

- FIND(x): v každém vrcholu jdi doleva/doprava podle porovnání; konec v nalezeném vrcholu, nebo v prázdném podstromu (nenalezeno).
- INSERT(x): jako FIND; na místě prázdného podstromu, kde hledání skončilo, vytvoř nový list.
- DELETE(x): tři případy – list smaž; vrchol s jedním synem „přemostí“ synem; vrchol se dvěma syny nahraď jeho *následníkem* (nejlevější vrchol pravého podstromu) a ten pak smaž (má nejvýše jednoho syna).

Všechny tři stojí $\Theta(h)$, kde h je hloubka stromu. V příznivém (vyváženém) případě je $h = \Theta(\log n)$, v nejhorším (degenerovaný strom, „lána“ – vznikne např. vkládáním už seříděných klíčů) je $h = \Theta(n)$.



vlevo vyvážený BST na 7 vrcholech, hloubka $2 = \lfloor \log_2 7 \rfloor$, operace $\Theta(\log n)$. Vpravo *degenerovaný* BST („lána“) ze stejných klíčů vložených v seříděném pořadí 2, 3, 5, 7, 9: hloubka $n - 1$, operace $\Theta(n)$. Oba splňují uspořádací vlastnost (levý $<$ vrchol $<$ pravý); liší se jen tvarem – proto chceme tvar řídit (AVL).

4.2 AVL stromy

Definice – AVL strom

AVL strom je binární vyhledávací strom, který je *hloubkově vyvážený*: pro každý vrchol v se hloubky jeho levého a pravého podstromu liší nejvýše o 1, tj.

$$|h(\ell(v)) - h(r(v))| \leq 1.$$

V každém vrcholu si pamatujeme *znaménko* (balanci) $\delta(v) = h(r(v)) - h(l(v)) \in \{-1, 0, +1\}$.

★ Věta – Hloubka AVL stromu

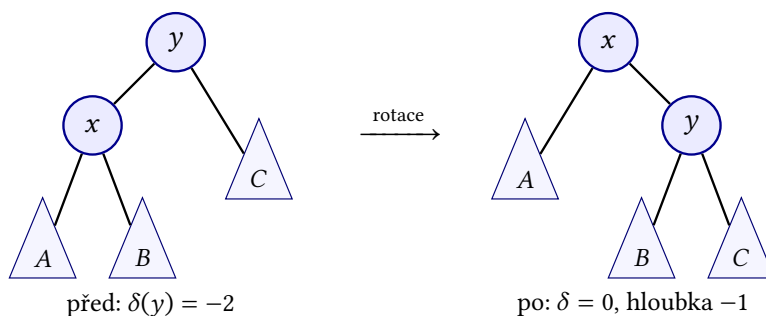
AVL strom o n vrcholech má hloubku $\Theta(\log n)$.

Intuice. Dolní mez $h \geq \log_2(n+1) - 1$ platí pro každý binární strom (strom hloubky h má nejvýše $2^{h+1} - 1$ vrcholů). Podstatná je *horní* mez hloubky, tedy *dolní* mez na počet vrcholů: označme A_h nejmenší možný počet vrcholů AVL stromu hloubky h . Kořen má dva podstromy, jeden hloubky $h-1$ a (aby byl strom co nejřidší) druhý hloubky $h-2$, takže

$$A_h = A_{h-1} + A_{h-2} + 1, \quad A_0 = 1, \quad A_1 = 2.$$

To je posloupnost Fibonacciho typu, $A_h = F_{h+3} - 1$, a roste exponenciálně: $A_h \geq \varphi^h$ pro $\varphi = \frac{1+\sqrt{5}}{2}$. Tedy $n \geq A_h \geq \varphi^h$, odkud $h \leq \log_\varphi n = \Theta(\log n)$. AVL strom je proto vždy „skoro vyvážený“ a všechny operace stojí $\Theta(\log n)$.

Vyváženost se po vložení nebo smazání obnovuje *rotacemi* – lokálními přepojeními, která zachovávají uspořádací vlastnost a zmenší hloubku rozhozeného místa. Stačí $O(1)$ rotací na jedné hladině, propagace nahoru je $O(\log n)$.



jednoduchá rotace hrany xy vyrovná stav $\delta(y) = -2$ s „rozhozením vlevo-vlevo“ (údaje „po“ platí pro typickou situaci po vložení, kdy $h(A) = h(B) + 1$). Podstrom B změní rodiče z x na y ; pořadí klíčů $A < x < B < y < C$ zůstává. (Pro „cik-cak“ rozhození vlevo-vpravo se použije *dvojitá* rotace = dvě jednoduché za sebou.)

🔧 Příklad – SZZ jaro 2023 č. 1: Třídění pomocí BST a AVL

(Reálná otázka SZZ: jaro 2023, č. 1 – Třídění (BST/AVL, složitost))

Zadání.

1. Popište, jak pomocí BST setřídít (vzestupně) posloupnost $x_1, \dots, x_n$ navzájem různých prvků.
2. Co se změní, mohou-li se prvky opakovat?
3. Co z toho plyne pro minimální možnou složitost operací s vyhledávacím stromem, lze-li klíče jen porovnávat?
4. Jakou složitost má algoritmus z bodu 1 s AVL stromem, jsou-li prvky řetězce délky L ?

Řešení.

(1) Postupně vlož $x_1, \dots, x_n$ do (zprvu prázdného) BST, pak proveď in-order průchod – ten vypíše klíče vzestupně. Vkládání jednoho prvku stojí $\Theta(h)$; při použití *vyváženého* stromu je $h = \Theta(\log n)$, takže n vložení trvá $\Theta(n \log n)$ a in-order průchod $\Theta(n)$. Celkem $\Theta(n \log n)$. (S nevyváženým BST hrozí degenerace na liánu a čas až $\Theta(n^2)$, např. pro už setříděný vstup.)

(2) Při opakování klíčů uspořádací vlastnost „< vlevo, > vpravo“ nestačí. Buď ve vrcholu držíme *počet výskytů* klíče (in-order pak vypíše klíč tolikrát), nebo povolíme rovnost na jedné straně ($\leq$),

případně klíče zjednodušíme dvojicí (x_i, i) porovnávanou lexikograficky (to navíc dá stabilní třídění). Setřídění zůstane $\Theta(n \log n)$; výstup už není množina, ale multimnožina.

(3) Třídění stromem je n -krát INSERT a jeden in-order výpis; výpis stojí $O(n)$ a žádná porovnání nepotřebuje. Kdyby nějaký vyhledávací strom (v porovnávacím modelu) zvládl n vložení v čase $o(n \log n)$, uměli bychom porovnáváním třdit v čase $o(n \log n)$. To zakazuje dolní odhad porovnávacího třídění (sekce 5): $\Omega(n \log n)$ porovnání v nejhorším případě. Proto INSERT nemůže mít (ani amortizovaně) složitost $o(\log n)$; potřebuje $\Omega(\log n)$ porovnání. Logaritmická cena operací AVL stromu je tedy v tomto smyslu *optimální*.

(4) U řetězců délky L není jedno porovnání konstantní – porovnání dvou řetězců stojí až $\Theta(L)$ (znak po znaku). Vkládání jednoho řetězce do AVL stromu projde $\Theta(\log n)$ vrcholů, v každém jedno porovnání řetězců za $O(L)$, tedy $O(L \log n)$ na prvek a $O(nL \log n)$ za n vložení. In-order výpis vypíše n řetězců o celkové délce $\leq nL$ za $O(nL)$. Celková složitost je

$$O(nL \log n).$$

i Poznámka. Plně rozkreslené stromy (vyvážený vs. liána, in-order pořadí, průběh rotace při vkládání) jsou v příklady/ u této otázky.

Příklad – SZZ léto 2022 č. 1: Binární vyhledávací stromy

(Reálná otázka SZZ: léto 2022, č. 1 – Binární vyhledávací stromy)

Zadání. (1) Definujte binární vyhledávací strom (BVS). (2) Pseudokódem implementujte *vložení* prvku (bez vyvažování) a rozeberte časovou složitost v nejlepším a nejhorším případě. (3) Pseudokódem implementujte *nalezení následníka* (pro klíč x vrchol s nejmenším klíčem $> x$) a rozeberte složitost v nejlepším a nejhorším případě.

Řešení.

(1) BVS je binární strom, jehož každý vrchol nese klíč a splňuje *uspořádací vlastnost*: všechny klíče v levém podstromu jsou $<$ klíč vrcholu a všechny v pravém podstromu jsou $>$ klíč vrcholu (viz definice výše).

(2) **Vložení.** Sestup od kořene doleva/doprava podle porovnání; nový prvek vlož jako list tam, kde sestup narazí na prázdný podstrom.

```
Insert(strom T, klic k):
  if T.koren == null:
    T.koren = new Uzel(k); return
  v = T.koren
  loop:
    if k == v.klic: return // klic uz ve strome je
    if k < v.klic:
      if v.levy == null: v.levy = new Uzel(k); return
      else: v = v.levy
    else: // k > v.klic
      if v.pravy == null: v.pravy = new Uzel(k); return
      else: v = v.pravy
```

Vložení projde jedinou cestu od kořene nejvýše k listu, tj. $O(h)$, a v nejhorším případě (nový list pod nejhlubším listem) $\Theta(h)$. Nejhorší případ: degenerovaný strom (liána, např. vzniklý vkládáním setříděné posloupnosti) má $h = n - 1$, tedy $\Theta(n)$. Nejlepší případ: sestup skončí hned pod kořenem, tj. $\Theta(1)$ (např. do liány $1, 2, \dots, n$ vkládáme 0). Chceme-li nejlepší *tvar stromu* při nejhorší volbě klíče, je to vyvážený strom s $h = \Theta(\log n)$, tedy $\Theta(\log n)$.

(3) **Následník.** Má-li x pravý podstrom, následník je jeho nejlevější vrchol; jinak stoupej nahoru, dokud nejsi levým synem – ten rodič je následník (jinak byl x maximem).

```

Successor(uzel x):
    if x.pravy != null: // nejmensi v pravem podstromu
        v = x.pravy
        while v.levy != null: v = v.levy
        return v
    v = x; r = x.rodic // nahoru, dokud nejsem levý syn
    while r != null and v == r.pravy:
        v = r; r = r.rodic
    return r

```

Obě větve projdou jednu cestu, takže $O(h)$. Nejhorší případ $\Theta(n)$: ve stromu vzniklém vkládáním $1, n, n-1, \dots, 2$ je následník kořene 1 klíč 2 na dně levé větve pod n , cesta má $n-1$ hran. Nejlepší případ $\Theta(1)$ (pravý syn x je list, nebo x je levý syn bez pravého podstromu). Ve vyváženém stromu je i nejhorší případ jen $\Theta(\log n)$.

5 Třídění: primitivní algoritmy, Quicksort, dolní odhad

Třídění je nejčastější podúloha vůbec. Projdeme jednoduché kvadratické algoritmy, praktický Quicksort a dokážeme, že porovnávací třídění nejde rychleji než $\Theta(n \log n)$.

5.1 Primitivní třídící algoritmy

Dvě pomocná pojmenování: *inverze* je dvojice indexů $i < j$ s $a_i > a_j$ (prvky ve špatném vzájemném pořadí); třídící algoritmus je *stabilní*, zachovává-li vzájemné pořadí prvků se *stejným* klíčem.

- **Selectsort (výběrem)**: opakovaně najdi minimum nesetříděné části a dej ho na konec setříděné. Vždy $\Theta(n^2)$ porovnání; prostor $\Theta(1)$; nestabilní.
- **Insertsort (vkládáním)**: ber prvky po jednom a vsouvěj je na správné místo do už setříděné přední části (posouváním). Nejhorší/průměrný $\Theta(n^2)$, *nejlepší* $\Theta(n)$ (téměř setříděný vstup); prostor $\Theta(1)$; stabilní.
- **Bubblesort (bublínkové)**: opakovaně procházej pole a prohazuj sousedy ve špatném pořadí; větší prvky „bublají“ dozadu. Nejhorší/průměrný $\Theta(n^2)$, *nejlepší* $\Theta(n)$ (s detekcí, že proběhl průchod beze změny); prostor $\Theta(1)$; stabilní.

💡 Intuice. Insertsort a Bubblesort jsou v nejhorším případě $\Theta(n^2)$, protože prohazují jen *sousední* prvky, a každé takové prohození opraví právě jednu inverzi; inverzí přitom může být až $\binom{n}{2} = \Theta(n^2)$ (úplně obrácená posloupnost). Selectsort prohazuje i vzdálené prvky, je ale $\Theta(n^2)$ kvůli počtu porovnání při hledání minim. Pro malá n nebo skoro setříděná data jsou ale rychlé a jednoduché; Insertsort je proto často „dokončovací“ fáze rychlejších algoritmů.

5.2 Quicksort

⚙️ Algoritmus – QUICKSORT

1. Pokud $n \leq 1$, vrať vstup.
2. Zvol *pivot* p (některý prvek).
3. Rozděl prvky na L (menší než p), S (rovné p), P (větší než p). ▷ $\Theta(n)$
4. Rekursivně setříd L a P .
5. Vrať zřetězení L, S, P .

Quicksort je rovněž typu rozděl a panuj, ale dělení je *datově závislé* (podle pivotu). Při „šťastné“ volbě pivotu (medián) je rekurence $T(n) = 2T(n/2) + \Theta(n) = \Theta(n \log n)$. Při „nešťastné“ volbě (pivot je vždy minimum/maximum – nastane např. na setříděném vstupu s pivotem na kraji) se odděluje po jednom

prvku a rekurence degeneruje na $T(n) = T(n-1) + \Theta(n) = \Theta(n^2)$.

nejhorší případ $\Theta(n^2)$, průměrný případ $\Theta(n \log n)$.

Náhodná volba pivotu dává očekávaný čas $\Theta(n \log n)$ a v praxi je Quicksort díky malým konstantám a třídění „na místě“ zpravidla nejrychlejší. (Verze v boxu výše pro názornost tvoří nová pole L, S, P ; praktická implementace dělí pole na místě a potřebuje jen zásobník rekurze, který má velikost $O(\log n)$, zanořuje-li se rekurze nejdřív do menší části a větší zpracuje cyklem.) Není stabilní.

Příklad – Simulace Quicksortu

Zadání. Odsimulujte Quicksort na poli $[5, 3, 8, 4, 2, 7, 1]$, pivot vždy první prvek.

Řešení. V každém kroku pivot p rozdělí pole na $L < p$, $S = p$, $P > p$; rekurze na L a P .

- $[5, 3, 8, 4, 2, 7, 1]$, $p = 5$: $L = [3, 4, 2, 1]$, $S = [5]$, $P = [8, 7]$.
- $[3, 4, 2, 1]$, $p = 3$: $L = [2, 1]$, $S = [3]$, $P = [4]$; $[2, 1]$, $p = 2$: $L = [1]$, $S = [2]$, $P = []$.
- $[8, 7]$, $p = 8$: $L = [7]$, $S = [8]$, $P = []$.

Zřetěžením setříděných částí zdola nahoru dostaneme $[1, 2, 3, 4, 5, 7, 8]$. (Vstup nebyl setříděný, takže pivoty dělily rozumně; na již setříděném vstupu by pivot „první prvek“ byl pokaždé minimum a strom rekurze by degeneroval na hloubku n , tedy $\Theta(n^2)$.)

5.3 Dolní odhad složitosti porovnávacího třídění

★ Věta – Dolní odhad porovnávacího třídění

Každý deterministický třídící algoritmus, který o prvcích zjišťuje informaci *pouze jejich vzájemným porovnáváním*, provede v nejhorším případě $\Omega(n \log n)$ porovnání.

 **Intuice.** Algoritmus „nevidí“ hodnoty, jen výsledky porovnání. Jeho běh popíšeme *rozhodovacím stromem*: vnitřní vrchol = jedno porovnání a_i vs. a_j se třemi výsledky ($<$, $=$, $>$), list = konečná odpověď (konkrétní setříděné pořadí). Délka cesty k listu = počet porovnání na daném vstupu, hloubka stromu = nejhorší případ.

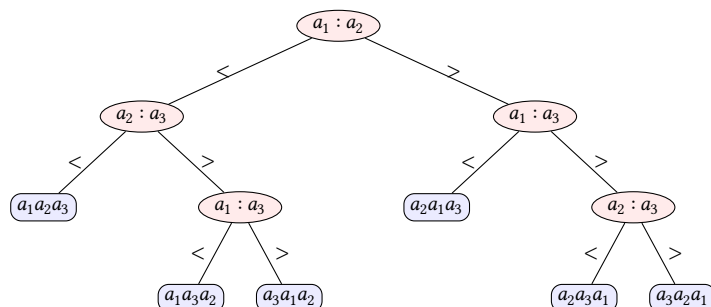
Důkaz (přes rozhodovací strom). (1) Uvažme n navzájem různých prvků. Různá vstupní pořadí (permutace) vyžadují různé posloupnosti přesunů, tedy musí skončit v *různých listech*. Permutací je $n!$, takže rozhodovací strom má aspoň $n!$ listů.

(2) Strom je *ternární* (každý vnitřní vrchol má nejvýše 3 syny dle výsledku porovnání), takže strom hloubky h má nejvýše 3^h listů. Z $3^h \geq n!$ plyne $h \geq \log_3(n!)$.

(3) Odhadneme $\log_3(n!)$ zdola. Z $n! \geq \left(\frac{n}{2}\right)^{n/2}$ (aspoň polovina činitelů je $\geq \frac{n}{2}$) dostaneme

$$h \geq \log_3(n!) \geq \frac{n}{2} \log_3 \frac{n}{2} = \Omega(n \log n).$$

Tedy v nejhorším případě je potřeba $\Omega(n \log n)$ porovnání. □



úplný rozhodovací strom pro třídění $n = 3$ navzájem různých prvků (výsledek „=“ proto nenastává). Vnitřní vrcholy jsou porovnání $a_i : a_j$, hrany jejich výsledky $<, >$ a listy všech $n! = 6$ setříděných pořadí (každé právě jednou). Nejdelší cesta má 3 porovnání, tj. hloubka stromu je 3. Obecně $\geq n!$ listů vynutí hloubku $\geq \log_3(n!) = \Omega(n \log n)$; *Mergesort této meze dosahuje*, je tedy asymptoticky optimální.

i Poznámka. Dolní odhad platí jen v *porovnávacím* modelu. Třídění, které o klíích ví víc (např. že jsou to malá celá čísla), může být rychlejší – přihrádkové (counting/radix) třídění běží až v $\Theta(n)$, protože klíče neporovnává, ale přímo indexuje.

6 Grafové algoritmy

Graf $G = (V, E)$ modeluje vztahy mezi objekty. Reprezentujeme ho buď *maticí sousednosti* ($n \times n$, test hrany $O(1)$, ale prostor $\Theta(n^2)$ a výpis sousedů $\Theta(n)$), nebo *seznamy sousedů* (prostor $\Theta(n + m)$, výpis sousedů úměrný jejich počtu) – pro řídké grafy volíme seznamy. Skoro všechny základní algoritmy pak běží v čase $O(n + m)$.

6.1 Prohledávání do šířky (BFS) a do hloubky (DFS)

Algoritmus – BFS – prohledávání do šířky z v_0

1. Všem vrcholům nastav stav *nenalezený*; $\text{stav}(v_0) \leftarrow \text{otevřený}$, $D(v_0) \leftarrow 0$, vlož v_0 do fronty Q .
2. Dokud $Q \neq \emptyset$: vyber v ze začátku Q .
3. Pro každého souseda w vrcholu v , je-li *nenalezený*: $\text{stav}(w) \leftarrow \text{otevřený}$, $D(w) \leftarrow D(v) + 1$, $P(w) \leftarrow v$, přidej w na konec Q . Pak $\text{stav}(v) \leftarrow \text{uzavřený}$. $\triangleright$ každý vrchol jde do fronty nejvýše jednou

BFS prochází vrcholy po *vrstvách* podle vzdálenosti od v_0 (počet hran) a používá *frontu* (FIFO). Vydá vzdálenosti $D(v)$ (nejkratší v počtu hran) a strom nejkratších cest přes ukazatele P .

DFS (prohledávání do hloubky) naopak používá *zásobník* / rekurzi: z vrcholu se zanoří co nejhlouběji, než se vrátí. Každému vrcholu přiřadí čas *otevření* $\text{in}(v)$ a *uzavření* $\text{out}(v)$; tyto časy se chovají jako *závorkování* (podstromy se do sebe vnořují, nekříží).

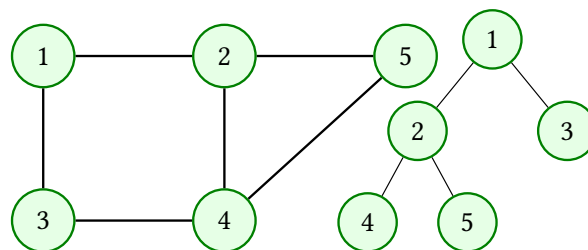
Algoritmus – DFS – prohledávání do hloubky, rekurzivně z v_0

Globální čítač $T \leftarrow 0$; všem vrcholům nastav stav *nenalezený*. Zavolej $\text{DFS-NAVŠTIV}(v_0)$.

$\text{DFS-NAVŠTIV}(v)$:

1. $T \leftarrow T + 1$; $\text{in}(v) \leftarrow T$; $\text{stav}(v) \leftarrow \text{otevřený}$.
2. Pro každého souseda w vrcholu v : je-li *nenalezený*, $P(w) \leftarrow v$ a zavolej $\text{DFS-NAVŠTIV}(w)$.
3. $T \leftarrow T + 1$; $\text{out}(v) \leftarrow T$; $\text{stav}(v) \leftarrow \text{uzavřený}$.

Obě prohledávání běží v čase $\Theta(n + m)$ (každý vrchol i hrana se zpracují konstanta-krát). Časy in/out z DFS využijeme vzápětí u topologického třídění.



graf G

BFS-strom z 1 (vrstvy 0, 1, 2)

BFS z vrcholu 1 dá vrstvy $V_0 = \{1\}$, $V_1 = \{2, 3\}$, $V_2 = \{4, 5\}$, tj. $D = (0, 1, 1, 2, 2)$. Stromové hrany (přes které se vrcholy poprvé objevily) tvoří strom nejkratších cest. DFS z 1 by naopak nejdříve sestoupilo $1 \rightarrow 2 \rightarrow 4 \rightarrow \dots$ a vrátilo se – pořadí návštěv i tvar stromu se liší.

6.2 Topologické třídění orientovaných grafů

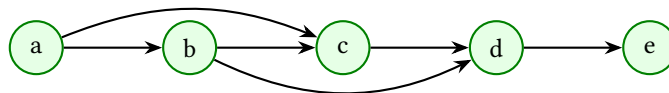
Definice – DAG a topologické uspořádání

Orientovaný acyklický graf (DAG) je orientovaný graf bez orientovaného cyklu. *Topologické uspořádání* je lineární uspořádání $<$ vrcholů takové, že pro každou hranu uv platí $u < v$ (každá hrana míří „dopředu“).

★ Věta – Existence topologického uspořádání

Orientovaný graf má topologické uspořádání právě tehdy, když je to DAG. Uspořádání lze najít v čase $\Theta(n + m)$.

Intuice. Cyklus zjevně topologicky uspořádat nelze (po šípkách bychom se vrátili). V DAGu naopak vždy existuje vrchol bez vstupních hran (zdroj); ten dáme první a odebereme – a indukcí pokračujeme. Algoritmicky elegantnější: spusť DFS; pořadí *klesajících* časů uzavření $\text{out}(v)$ je topologické. (Pro každou hranu uv v DAGu totiž $\text{out}(u) > \text{out}(v)$, protože zpětné hrany, které by to porušily, by tvořily cyklus.)



DAG srovnaný do řady $a < b < c < d < e$ – všechny hrany míří zleva doprava, což je přesně topologické uspořádání. Takové uspořádání umožní počítat na DAGu „po proudu“ (např. nejdelší cestu, počet cest) v lineárním čase.

6.3 Nejkratší cesty v ohodnocených grafech

Mějme graf s délkami hran $\ell(e)$. Vzdálenost $d(u, v)$ je nejmenší délka uv -sledu. Je dobře definovaná, pokud v grafu nejsou *záporné cykly* (obcházením by šlo sled libovolně zkracovat); pak se minimum nabývá na cestě. Oba následující algoritmy stojí na *relaxaci* hrany: je-li $h(w) > h(v) + \ell(vw)$, zlepši $h(w) \leftarrow h(v) + \ell(vw)$.

★ Věta – Dijkstrův algoritmus

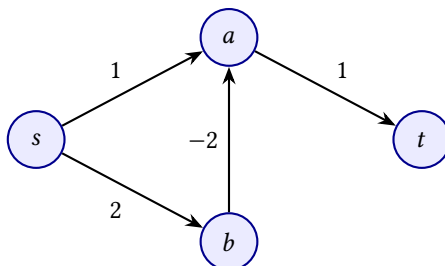
Pro graf s *nezápornými* délkami hran najde Dijkstrův algoritmus vzdálenosti z v_0 do všech vrcholů. S binární haldou běží v čase $O((n + m) \log n)$.

Intuice. Dijkstra uzavírá vrcholy v pořadí rostoucí vzdálenosti: vždy vytáhne dosud neuzavřený vrchol s nejmenším h (proto halda) a zrelaxuje jeho hrany. Klíčový invariant: jakmile vrchol uzavřeme, jeho h už je správná vzdálenost a vrchol se znovu neotevře – to platí právě proto, že hrany jsou nezáporné (objížďka přes pozdější, vzdálenější vrchol nemůže být kratší). Proto se každý vrchol uzavře jen jednou a platí uvedená složitost. Při záporných hranách invariant padá. Průvodce zapisuje Dijkstru tak, že relaxace smí uzavřený vrchol znovu otevřít; taková verze pak v grafu bez záporných cyklů sice spočte správné vzdálenosti, ale může vrcholy otevírat opakovaně a běžet až exponenciálně dlouho. Běžná implementace s haldou, která uzavřené vrcholy už nerelaxuje, vydá špatný výsledek (obrázek níže).

★ Věta – Bellmanův–Fordův algoritmus

Pro graf bez záporných cyklů (i se zápornými hranami) najde Bellmanův–Fordův algoritmus vzdálenosti z v_0 v čase $O(nm)$. Záporný cyklus dosažitelný z v_0 navíc umí detekovat.

Intuice. BF nepotřebuje haldy: provede $n - 1$ fází, v každé zrelaxuje všechny hrany. Invariant: po i fázích je $h(v)$ nejvýše délka nejkratšího v_0 -sledu o nejvýše i hranách. Nejkratší cesta má $\leq n - 1$ hran, takže po $n - 1$ fázích jsou všechna $h(v)$ správně. Pokud se v n -té fázi ještě něco zlepší, existuje záporný cyklus.



graf se zápornou hranou $b \rightarrow a$ (-2). Dijkstra uzavře a s $h(a) = 1$ (přímou hranou $s \rightarrow a$) a t dostane $h(t) = 2$. Levnější obchvat $s \rightarrow b \rightarrow a$ délky $2 - 2 = 0$ se objeví až po uzavření a ; verze, která uzavřené vrcholy nerelaxuje, vydá $d(s, t) = 2$ místo správného 1 (při obou pořadích uzavření b a t). Verze z Průvodce a znovu otevře a t opraví, zaplatí za to ale opakovaným zpracováním vrcholů. Bellman-Ford fázovou relaxací najde správně $d(s, a) = 0$ a $d(s, t) = 1$ (přes $s \rightarrow b \rightarrow a \rightarrow t$). Záporných cyklů zde není, vzdálenosti jsou dobře definované.

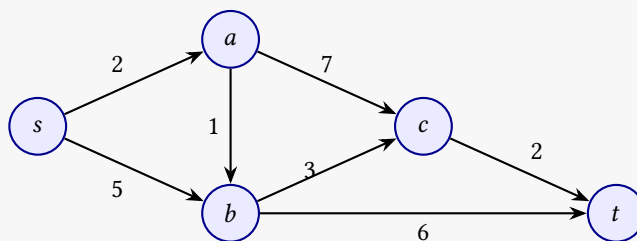
✎ Příklad – Simulace Dijkstrova algoritmu

Zadání. Najděte Dijkstrovým algoritmem vzdálenosti z s v grafu s nezápornými hranami $s \rightarrow a$ (2), $s \rightarrow b$ (5), $a \rightarrow b$ (1), $a \rightarrow c$ (7), $b \rightarrow c$ (3), $b \rightarrow t$ (6), $c \rightarrow t$ (2).

Řešení. Vždy uzavřeme dosud neuzavřený vrchol s nejmenším h a zrelaxujeme jeho hrany ($h(w) \leftarrow \min(h(w), h(v) + \ell(vw))$). Tabulka uzavírání:

- Uzavři s ($h=0$): relax $a \rightarrow 2$, $b \rightarrow 5$.
- Uzavři a ($h=2$): relax $b \rightarrow \min(5, 2+1)=3$, $c \rightarrow 2+7=9$.
- Uzavři b ($h=3$): relax $c \rightarrow \min(9, 3+3)=6$, $t \rightarrow 3+6=9$.
- Uzavři c ($h=6$): relax $t \rightarrow \min(9, 6+2)=8$.
- Uzavři t ($h=8$).

Výsledek $d(s, \cdot) = (s:0, a:2, b:3, c:6, t:8)$. Všimni si, že c i t se po prvním nalezení ještě zlepšily (z 9 na 6, resp. z 9 na 8) – proto Dijkstra uzavírá až dosud nejmenší h , ne první nalezenou hodnotu.



6.4 Minimální kostra (Jarník, Borůvka)

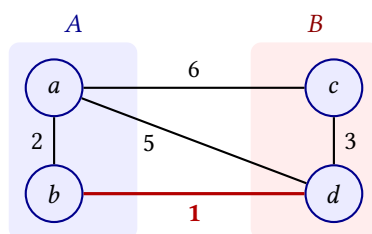
Definice – Kostra a minimální kostra

Kostra souvislého grafu G je podgraf, který je strom a obsahuje všechny vrcholy. *Minimální kostra* (MST) je kostra s nejmenším součtem vah hran. Mají-li hrany navzájem různé váhy, je MST určena jednoznačně.

★ Lemma – Řezové lemma

Nechť hrany mají navzájem různé váhy a nechť $A \dot{\cup} B = V$ je rozdělení vrcholů na dvě neprázdné části. Pak *nejlehčí hrana* vedoucí mezi A a B leží v každé minimální kostře.

Důkaz (výměnný argument). Nechť e je nejlehčí hrana řezu mezi A a B a nechť nějaká kostra T ji neobsahuje. Přidáním e do T vznikne cyklus, který řez překročí ještě v nějaké jiné hraně f (jdoucí mezi A a B). Protože e je v řezu nejlehčí a váhy jsou různé, je $w(f) > w(e)$. Záměnou $T' = T - f + e$ dostaneme kostru s menší vahou, takže T nebyla minimální. Tedy každá MST obsahuje e . $\square$



řez mezi $A = \{a, b\}$ a $B = \{c, d\}$. Hrany přes řez mají váhy 1, 5, 6; nejlehčí ($b-d$, váha 1, zvýrazněná) podle řezového lemmatu leží v každé MST. Toto lemma je společným základem korektnosti Jarníkova, Borůvkova i Kruskalova algoritmu – každý z nich přidává hrany, které jsou nejlehčí v nějakém řezu.

★ Věta – Jarníkův a Borůvkův algoritmus

Oba algoritmy najdou minimální kostru.

- **Jarník:** pěstuje jeden strom od počátečního vrcholu; opakovaně přidá nejlehčí hrana vedoucí ze stromu ven. S haldou běží v čase $O(m \log n)$.
- **Borůvka:** začne lesem izolovaných vrcholů; v každé fázi přidá pro každou komponentu její nejlehčí vnější hrana (všechny naráz). Počet komponent klesne aspoň na polovinu, takže fází je $\leq \lceil \log_2 n \rceil$ a celkový čas je $O(m \log n)$.

Intuice. Korektnost obou plyne z řezového lemmatu: Jarník bere nejlehčí hrana řezu „strom vs. zbytek“; Borůvka pro každou komponentu nejlehčí hrana řezu „komponenta vs. zbytek“. Obě tedy přidávají jen hrany, které musí být v MST. (Příbuzný **Kruskalův** algoritmus třídí hrany podle váhy a bere každou, která nespojí už spojené vrcholy; s datovou strukturou Union-Find běží také $O(m \log n)$.)

6.5 Toky v sítích (Ford–Fulkerson)

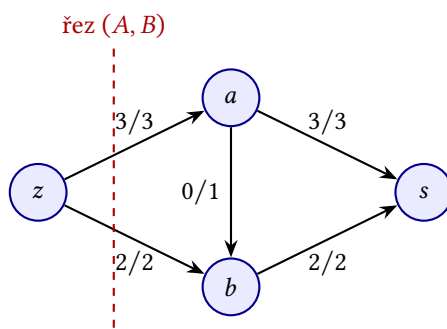
Definice – Síť, tok, řez

Síť je orientovaný graf s kapacitami $c(e) \geq 0$ na hranách a dvěma vyznačenými vrcholy: *zdrojem* z a *stokem* s . *Tok* je funkce $f(e)$ s $0 \leq f(e) \leq c(e)$ (kapacitní omezení), která v každém vrcholu kromě z, s splňuje *zachování* (Kirchhoffův zákon): co přiteče, to odeče. *Velikost toku* $|f|$ je čistý odtok ze zdroje. *Řez* (A, B) je rozdělení vrcholů s $z \in A$, $s \in B$; jeho *kapacita* je součet kapacit hran z A do B .

★ Věta – O maximálním toku a minimálním řezu

Velikost maximálního toku se rovná kapacitě minimálního řezu ($\text{max-flow} = \text{min-cut}$). Fordův–Fulkersonův algoritmus maximální tok najde: dokud v rezervní (reziduální) síti existuje zlepšující cesta ze z do s , pošle po ní co nejvíc toku.

Intuice. Rezerva hrany je volná kapacita ve směru hrany plus možnost „vrátit“ už poslaný tok proti směru. Zlepšující cesta je cesta ze z do s samými hranami s kladnou rezervou; po ní přidáme tok o velikosti minima rezerv na cestě. Když žádná taková cesta není, je tok maximální: množina A vrcholů dosažitelných ze z v rezervní síti dává řez (A, B) , jehož hrany z A do B jsou nasycené a hrany z B do A nevedou žádný tok, takže $|f| = c(A, B)$ – a protože $|f| \leq c(\text{libovolný řez})$, je tento řez minimální a tok maximální. (Pro celočíselné kapacity algoritmus skončí, každé zlepšení zvýší tok aspoň o 1.)



síť s popisky tok/kapacita. Maximální tok má velikost $|f| = 5$ (ze z odtéká $3 + 2$). Minimální řez $A = \{z\}$, $B = \{a, b, s\}$ má kapacitu $c = 3 + 2 = 5$. Rovnost $|f| = c(A, B) = 5$ je věta $\text{max-flow} = \text{min-cut}$; hrany řezu jsou nasycené ($3/3, 2/2$), takže tok už nelze zvětšit.

Shrnutí

Poznámka. Klíčové pojmy, fakta a složitosti (přehled k SZZ). Stručný výčet klíčových pojmů, algoritmů a jejich složitostí okruhu I2. Zkratky: n = počet prvků / vrcholů, m = počet hran.

Složitost algoritmů a asymptotická notace.

- Model RAM:** elementární operace (aritmetika, porovnání, přístup do paměti) každá $O(1)$; výsledky nezávisí na HW až na konstantní faktor. Složitost v *nejhorším případě* = maximum přes vstupy velikosti n ; *nejlepší* = minimum; *průměrný* = střední hodnota přes vstupy; bez poznámky myslíme nejhorší.
- $f \in O(g)$: $\exists c > 0$: $f(n) \leq c g(n)$ pro skoro všechna n (horní odhad); $\Omega(g)$ – dolní; $\Theta(g)$ – těsný. Pravidla: v součtu dominuje nejrychleji rostoucí člen; $\log n = O(n^\epsilon)$ pro každé $\epsilon > 0$; $n^d = O(2^n)$.
- Měření velikosti dat:** u číselných úloh je velikost *délka zápisu* (počet číslic/bitů), ne hodnota čísla. „Vyzkoušet dělitele do x “ je $O(x)$, ale $x \approx 10^{\text{délka}}$, tedy *exponenciální* v délce zápisu (viz rozklad čísla).

Třídy složitosti P a NP, NP-úplnost.

- $L \in \mathbf{P}$: $\exists$ algo. a polynom p : pro každý x doběhne v $p(|x|)$ se správnou odpovědí. $L \in \mathbf{NP}$: $\exists$ verifikátor $K \in \mathbf{P}$ a polynom g : $x \in L \Leftrightarrow \exists y, |y| \leq g(|x|) : K(x, y) = 1$ (y = *certifikát*; ověřit hotové řešení je v $\mathbf{P}$, najít ho nemusí být). Platí $\mathbf{P} \subseteq \mathbf{NP}$; zda $\mathbf{P} = \mathbf{NP}$, je otevřené.
- Polynomiální převod** $A \leq B$: $\exists$ polynomiálně spočitatelná f : $x \in A \Leftrightarrow f(x) \in B$. Obtížnost teče po šipce: $A \leq B \wedge B \in \mathbf{P} \Rightarrow A \in \mathbf{P}$; obměna: $A \leq B \wedge A \notin \mathbf{P} \Rightarrow B \notin \mathbf{P}$. Relace $\leq$ je reflexivní a tranzitivní.
- L je **NP-těžký**: každý $M \in \mathbf{NP}$ se polynomiálně převede na L . **NP-úplný** = $\text{NP-těžký} \wedge L \in \mathbf{NP}$ (nejtěžší v $\mathbf{NP}$). Cook–Levinova věta (bez důkazu): SAT je NP-úplný.

- Klasické NP-úplné problémy: SAT, 3-SAT, nezávislá množina, klika, k -barvení ($k \geq 3$), Hamiltonská kružnice, batoh, součet podmnožiny. Typický řetěz: SAT $\rightarrow$ 3-SAT $\rightarrow$ nezáv. množina $\rightarrow$ klika; 3-SAT $\rightarrow$ 3-barvení.
- Postup NP-úplnosti nového M : (1) $M \in \mathbf{NP}$ – popsat certifikát a polynomiální verifikátor; (2) $L \leq M$ pro nějaký NP-úplný L – zkonstruovat polynomiální převod f .

Metoda rozděl a panuj, Master theorem.

- Rekurence: $T(n) = aT(n/b) + \Theta(n^c)$, $T(1) = \Theta(1)$.
- **Master theorem** (bez důkazu): nechť $q = a/b^c$: $q < 1 \Rightarrow T(n) = \Theta(n^c)$ (dominuje kořen stromu rekurze); $q = 1 \Rightarrow T(n) = \Theta(n^c \log n)$ (všechny hladiny stejné); $q > 1 \Rightarrow T(n) = \Theta(n^{\log_b a})$ (dominují listy).
- Příklady: Mergesort $a=b=2, c=1$: $q=1 \Rightarrow \Theta(n \log n)$; Karacuba $a=3, b=2, c=1$: $q=3/2 > 1 \Rightarrow \Theta(n^{\log_2 3}) \approx \Theta(n^{1.585})$; medián 2 polí $a=1, b=2, c=0$: $q=1 \Rightarrow \Theta(\log n)$.
- **Mergesort**: dělí na 2 poloviny, slévá $\Theta(n)$; vždy $\Theta(n \log n)$; stabilní; prostor $\Theta(n)$. Slévání k posloupností s celkem n prvků přes haldu: $\Theta(n \log k)$ (každý prvek jednou do/z haldy velikosti k).
- **k -tý nejmenší prvek**: Quickselect (jedna rekurze podle pivotu) má s náhodným pivotem očekávaný čas $\Theta(n)$; LinearSelect (medián mediánů pětic jako pivot) zaručí části zhruba $\leq \frac{7}{10}n$, $T(n) = T(n/5) + T(7n/10) + \Theta(n) = \Theta(n)$ v nejhorším případě.

Binární vyhledávací stromy (BST, AVL) a halda.

- **BST – uspořádací vlastnost**: levý podstrom $< k(v) <$ pravý podstrom v každém vrcholu. Operace FIND/INSERT/DELETE v $\Theta(h)$; in-order průchod vydá vzestupně seřazený výpis.
- **Následník** (nejmenší klíč ostře větší než x): má-li vrchol pravý podstrom, jde do jeho nejlevějšího vrcholu; jinak stoupej nahoru, dokud nejsi levým synem – ten otec je následník. Složitost $\Theta(h)$.
- Degenerace (lána): seřazené vstupy $\Rightarrow h = \Theta(n) \Rightarrow$ operace $\Theta(n)$. Porovnávání klíčů délky L (řetězce) násobí cenu faktorem L : třídění n řetězců AVL stromem $= O(n \cdot L \cdot \log n)$.
- **AVL strom**: BST s podmínkou $|h(\ell) - h(r)| \leq 1$ v každém vrcholu (h = hloubka podstromu). Dolní mez: nejřidší AVL strom splňuje $A_h = A_{h-1} + A_{h-2} + 1$, $A_0 = 1$, $A_1 = 2$, tedy $A_h \geq \varphi^h$ – odkud $n \geq \varphi^h$, tj. $h = O(\log n)$. Operace $\Theta(\log n)$ s $O(\log n)$ rotacemi. Třídění přes AVL (vlození + in-order): $\Theta(n \log n)$; z dolního odhadu třídění $\Omega(n \log n)$ plyne dolní mez $\Omega(\log n)$ na BST operace.
- **Binární halda**: úplný bin. strom uložený v poli, haldová podmínka (kořen = globální minimum). INSERT a EXTRACTMIN $O(\log n)$; MIN $O(1)$. Využití: Dijkstra, Jarník, slévání k posloupností.

Třídění.

- **Primitivní** ($\Theta(n^2)$): Selectsort – vždy $\Theta(n^2)$, nestabilní; Insertsort – nejlepší $\Theta(n)$ (téměř seřazený vstup), stabilní; Bubblesort – nejlepší $\Theta(n)$ (detekcí průchodu beze změny), stabilní.
- **Quicksort**: průměrný $\Theta(n \log n)$, nejhorší $\Theta(n^2)$ (pivot = min/max, nastane např. seřazený vstup s pivotem na kraji); nestabilní; prostor $O(\log n)$ na zásobník (rekurze nejdřív do menší části). Náhodná volba pivotu $\Rightarrow \Theta(n \log n)$ v očekávání.
- **Mergesort**: vždy $\Theta(n \log n)$, stabilní, prostor $\Theta(n)$ (viz rozděl a panuj).
- **Dolní odhad** porovnávacího třídění: $\Omega(n \log n)$ v nejhorším případě. Argument: rozhodovací strom má $\geq n!$ listů (různá pořadí $\Rightarrow$ různé listy); hloubka $\geq \log_3(n!) = \Theta(n \log n)$ (z $n! \geq (n/2)^{n/2}$).
- **Inverze** = dvojice $(i < j)$ s $a_i > a_j$; Insertsort a Bubblesort prohazují jen sousedy, tj. opraví jednu inverzi na krok, takže $\Omega(n^2)$ na obrácené posloupnosti (Selectsort je $\Theta(n^2)$ kvůli porovnáním). **Stabilní** algoritmus zachovává pořadí prvků se stejným klíčem.

Grafové algoritmy.

- **Reprezentace**: matice sousednosti $\Theta(n^2)$, hrana $O(1)$; seznamy sousedů $\Theta(n+m)$, iterace sousedů rychlá; pro řídké grafy volíme seznamy.
- **BFS** $\Theta(n+m)$: fronta, vrstvy, vzdálenosti v počtu hran. **DFS** $\Theta(n+m)$: zásobník/rekurze, časy in/out (závorkování). **Topologické třídění** DAGu: DFS + klesající out(v) (nebo odebrání zdrojů);

$\Theta(n + m)$; existuje $\Leftrightarrow$ DAG.

- **Dijkstrův algoritmus:** nezáporné délky hran; relaxace v pořadí rostoucí h (haldy); $O((n + m) \log n)$. Při záporné hraně invariant (uzavřený $\Rightarrow$ správná vzdálenost) selže.
- **Bellmanův–Fordův algoritmus:** záporné hrany povoleny, bez záporných cyklů; $n - 1$ fází – v každé zrelaxuj všechny hrany; $O(nm)$. Záporný cyklus $\Leftrightarrow n$ -tá fáze stále mění h .
- **Minimální kostra (MST)** – řezové lemma: světlá (nejlehčí) hrana přes libovolný řez leží v každé MST. Jarník (Prim): z jednoho vrcholu hladově s haldou, $O(m \log n)$. Borůvka: paralelní přidávání nejlehčí vnější hrany každé komponenty, $\lceil \log_2 n \rceil$ fází, $O(m \log n)$. Kruskal: setřídít hrany + Union-Find, přidávat nejlehčí nespojující tutéž komponentu, $O(m \log n)$.
- **Toky v sítích:** tok f splňuje $0 \leq f(e) \leq c(e)$ (kapacita) a Kirchhoffův zákon. Ford–Fulkerson augmentuje po nenasyčených cestách v reziduální síti; pro celočíselné kapacity terminuje. **Věta max-flow = min-cut:** velikost max. toku = kapacita min. řezu.

Zdroje

- M. Mareš, T. Valla: *Průvodce labyrintem algoritmů*, 2. vydání, edice CZ.NIC, Praha 2022 (doporučená učebnice předmětů NTIN060 a NTIN061, jediný primární zdroj výkladu; licence CC BY-ND 4.0, volně ke stažení).
<https://pruvodce.ucw.cz/>
- MFF UK: požadavky ke státní závěrečné zkoušce (Informatika, Bc.) a zadání otázek z minulých termínů.
<https://www.mff.cuni.cz/cs/studenti/bakalarske-studium/statni-zaverecne-zkousky/bakalarske-statni-zkousky-studijniho-programu-informatika>