1. Модели на изчисленията – машина на Тюринг, машина с произволен достъп и език за програмиране.

Детерминистична машина на Тюринг ще наричаме осморка от вида

$$\mathcal{M} = \langle Q, \Sigma, \Gamma, \delta, \sqcup, q_{\mathtt{start}}, q_{\mathtt{accept}}, q_{\mathtt{reject}} \rangle,$$

където:

- Q - крайно множество от състояния;
- Σ - крайна азбука за входа;
- Γ - крайна азбука за лентата, $\Sigma \subseteq \Gamma$;
- $\sqcup$ - символ за празна клетка на лентата, $\sqcup \in \Gamma \setminus \Sigma$;
- $q_{\mathtt{start}} \in Q$ - начално състояние;
- $q_{\mathtt{accept}} \in Q$ - приемащо състояние;
- $q_{\mathtt{reject}} \in Q$ - отхвърлящо състояние, където $q_{\mathtt{accept}} \neq q_{\mathtt{reject}}$;
- $\delta : Q' \times \Gamma \to Q \times \Gamma \times \{\lhd, \rhd, \Box\}$ - тотална функция на преходите, където $Q' = Q \setminus \{q_{\mathtt{accept}}, q_{\mathtt{reject}}\}$.

Всяка машина на Тюринг разполага с неограничено количество памет, която е представена като безкрайна (и в двете посоки) лента, разделена на клетки. Всяка клетка съдържа елемент на Γ. Сега ще опишем как $\mathcal{M}$ работи върху вход думата $\alpha \in \Sigma^\star$. Първоначално безкрайната лента съдържа само думата α. Останалите клетки на лентата съдържат символа $\sqcup$.

Освен това, $\mathcal{M}$ се намира в началното състояние $q_{\mathtt{start}}$ и главата за четене е върху най-левия символ на α. Работата на $\mathcal{M}$ е описана от функцията на преходите δ.

Тук до голяма степен следваме [Sip12, Глава 3]. Понятието за машина на Тюринг има много еквивалентни дефиниции.

Тези две състояния ще наричаме заключителни

Това означава, че веднъж достигнем ли заключително състояние, не можем да правим повече преходи. Тук следваме [Sip12, стр. 169] и [HMU01, стр. 327].

- Формално, **моментната конфигурация** (или описание) на едно изчисление на машина на Тюринг е тройка от вида

$$(\alpha, q, \beta) \in \Gamma^\star \times Q \times \Gamma^+,$$

като интерпретацията на тази тройка е, че машината се намира в състояние q и лентата има вида

$$\cdots \sqcup \sqcup \sqcup \alpha \underline{x} \beta' \sqcup \sqcup \sqcup \cdots,$$

където $\beta = x\beta'$ и четящата глава на машината е поставена върху x.

- Макар и да имаме безкрайна лента, моментната конфигурация, която може да се представи като *крайна* дума, описва цялото моментно състояние на машината на Тюринг.

- **Началната конфигурация** за входната дума $\alpha \in \Sigma^\star$ представлява тройката

$$(\varepsilon, q_{\mathtt{start}}, \alpha\sqcup).$$

- **Приемаща конфигурация** представлява тройка от вида

$$(\beta, q_{\mathtt{accept}}, \gamma).$$

- **Отхвърляща конфигурация** представлява тройка от вида

$$(\beta, q_{\mathtt{reject}}, \gamma).$$

- Една конфигурация ще наричаме **заключителна**, ако тя е или приемаща или отхвърляща.

Както за автомати, удобно е да дефинираме бинарна релация $\vdash$ над $\Gamma^\star \times Q \times \Gamma^+$, която ще казва как моментната конфигурация на машината $\mathcal{M}$ се променя при изпълнение на една стъпка.

Една машина на Тюринг $\mathcal{N}$ се нарича недетерминирана, ако функцията на преходите има вида

$$\Delta : Q' \times \Gamma \to \mathscr{P}(Q \times \Gamma \times \{\lhd, \rhd, \square\}),$$

където да напомним, че $Q' = Q \setminus \{q_{\mathtt{accept}}, q_{\mathtt{reject}}\}$.

Отново можем да дефинираме бинарна релация $\vdash$ над $\Gamma^\star \times Q \times \Gamma^+$, която ще казва как моментното описание на машината $\mathcal{N}$ се променя при изпълнение на една стъпка.

2. Дефиниции на (машинно-зависима) сложност (по време и памет) в най-лошия и средния случай.

"Сложност на алгоритъм" е мярка за това, колко ресурси ползва този алгоритъм. "Сложен" в този смисъл означава "ползва много ресурси". За какви ресурси става дума?

- Ресурсът, който най-често имаме предвид, е времето. Естествено е, че искаме алгоритмите ни да работят бързо. В този смисъл, "качествен алгоритъм" е алгоритъм, който работи бързо върху всички входове. Но "бързо" е разговорен термин. Прецизните разсъждения се базират на понятието "сложност по време" (на английски е *time complexity*), което ще разгледаме след малко. Засега само казваме, че алгоритъм с висока сложност по време е алгоритъм, който е бавен.

- Следващият по важност ресурс е паметта, която ползва алгоритъмът. Естествено е, че искаме алгоритмите ни да ползват малко памет. В този смисъл, "качествен алгоритъм" е алгоритъм, който ползва малко памет върху всички входове. Прецизните разсъждения се базират на понятието "сложност по памет" (на английски е *space complexity*), което разгледаме след малко. Засега само казваме, че алгоритъм с висока сложност по памет е алгоритъм, който ползва много памет.

- по време

- Първото е, че се фокусираме именно върху алгоритмите, а не върху програмните им реализации. Разликата между бързодействието на две програми за една и съща задача, които реализират два различни алгоритъма, по правило са много по-драстични от разликите в бързодействието на две програми, реализиращи един и същи алгоритъм. **По отношение на сложните, нетривиални задачи, печелившата стратегия за бързодействието е подобрение на алгоритъма.** Печалбата от по-бързия алгоритъм е толкова по-видима, колкото по-голям е входът.

- Второто е допускането, че всички елементарни инструкции (стъпки) се изпълняват за единица време[†], така че времето за изпълнение на алгоритъма върху някакъв вход е точно броят на елементарните инструкции, които се изпълняват по време на работата му.

Ако в този опростен модел алгоритъм A е по-бърз от алгоритъм B за една и съща задача, най-вероятно в реалния свят програмната реализация на A ще е по-бърза от тази на B, като разликата ще е толкова по-очевидна, колкото по-голям е входът.

Определение 15: Сложност по време

Нека Π е изчислителна задача и A е алгоритъм за нея. За всяка големина на входа $n \in \mathbb{N}^+$, нека $\mathfrak{I}(n)$ е крайното множество от съществено различните входове с големина n. За всеки вход κ, нека $f(\kappa)$ е броят стъпки, които се изпълняват от $A(\kappa)$. Тогава, за всяко n, *сложността по време на* A *в най-лошия случай* е

$$T_A(n) = \max\{f(\kappa) \mid \kappa \in \mathfrak{I}(n)\}$$

сложността по време на A *в най-добрия случай* е

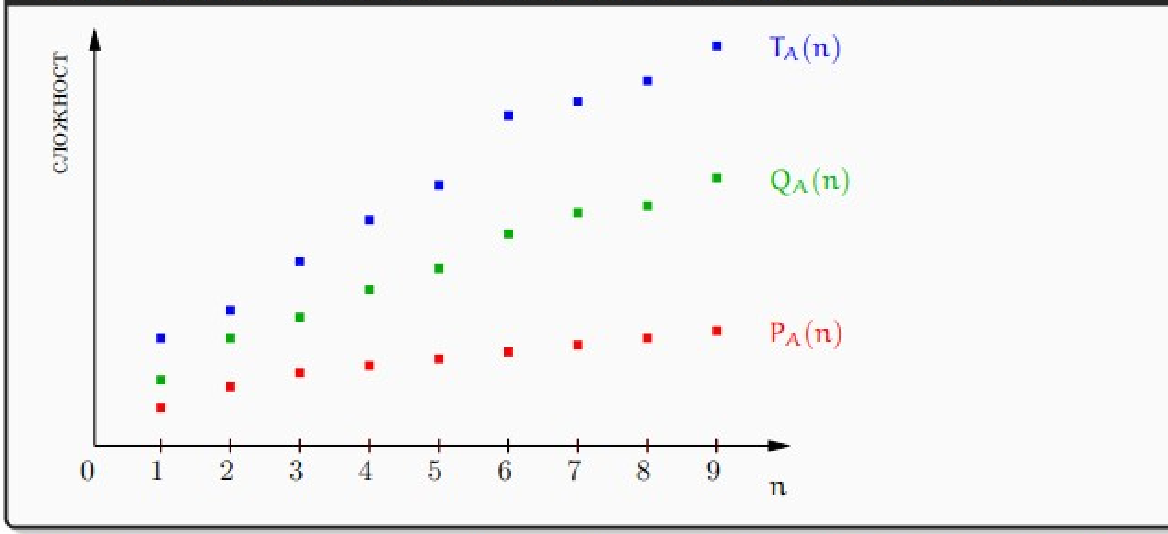
$$P_A(n) = \min\{f(\kappa) \mid \kappa \in \mathfrak{I}(n)\}$$

и *средната сложност по време на* A е

$$Q_A(n) = \frac{1}{|\mathfrak{I}(n)|} \sum_{\kappa \in \mathfrak{I}(n)} f(\kappa)$$

Забележка: използваната нотацията, например "$T_A(n)$" и т. н., не е общоприета.

Фигура 2.1 : Сложност по време в най-лошия случай $T_A(n)$, в най-добрия случай $P_A(n)$ и средна сложност $Q_A(n)$. Ординатата е сложността като брой стъпки.



Средната сложност винаги е между сложността в най-лошия и сложността в най-добрия случай. По правило функциите на сложността са строго нарастващи, защото (по правило) един и същи алгоритъм работи по-бавно върху по-голям вход, но има и изключения, както ще видим надолу в примера с Евклидовия алгоритъм.

На практика най-често използвана е сложността по време в най-лошия случай (*worst-case time complexity* на английски), по две основни причини. Първо, изследването на сложността в най-лошия случай е много по-лесно от изследването на средната сложност, в което ще се убедим в следваща лекция при анализа на QUICKSORT. Анализирането на средната сложност изисква значително по-задълбочени математически познания и значително по-сложни техники, дори при неявното допускане в определението на $Q_A(n)$, че всички входове с дадена големина са еднакво вероятни. Втората причина е, че резултатът за най-лошия случай е твърда гаранция, че по-лошо не може да бъде.

- по памет

Сложността по време е функция на големината на входа. Но има много входове с една и съща големина. По-лошо, в нашия опростен модел, в който всяко число има един и същи размер (единица), има **безброй много** входове за всяка големина на входа[†]. Да разгледаме отново сортиращ алгоритъм. Да кажем, че сортираме цели числа. Множеството от входовете с големина 1 е $\mathbb{Z}$. Множеството от входовете с големина 2 е $\mathbb{Z}^2$. Множеството от входовете с големина 3 е $\mathbb{Z}^3$. И така нататък. В общия случай, множеството от входовете с големина n е $\mathbb{Z}^n$. Виждаме, че множеството от входовете е (изброимо) безкрайно за всяко n. Следните разсъждения ни позволяват да разглеждаме само краен брой входове за този алгоритъм.

Сложността по памет на даден алгоритъм A върху даден вход е броят на елементите памет, които A ползва, без да броим паметта, в която се разполага входа и паметта, в която се разполага изходът. С други думи, гледаме само **работната памет** на алгоритъма; това е съвкупността от променливите му, които се ползват, за да се получи изходът от входа. Сложност по памет в най-лошия, средния и най-добрия случай се дефинира по начин, аналогичен на сложността по време.

Една значителна принципна разлика между сложността по време и сложността по памет е, че сложността по време—било в най-лошия случай, било средната—по правило е строго растяща функция на големината на входа, докато е напълно възможно даден алгоритъм да ползва само константна работна памет за **всяка** големина на входа. Алгоритми, които ползват константна работна памет, се наричат *in-place*.

3. Поведение на асимптотически положителни целочислени функции – O-, Ω-, Θ-, o- и ω-нотация.

Определение 16: Асимптотично положителна функция

Нека $f : \mathbb{R}^+ \to \mathbb{R}^+$. Казваме, че f е *асимптотично положителна*, ако

$$\exists n_0 \in \mathbb{R}^+ \, \forall n \geqslant n_0 : f(n) > 0$$

На прост български, иска се функцията да е строго положителна от някоя стойност на аргумента нататък.

Нотация 1: $\forall n \nearrow$

В тези лекционни записки, “$\forall n \nearrow$” е кратък запис за “$\exists n_0 \, \forall n \geqslant n_0$”. На прост български се казва “за всички достатъчно големи n”.

Определение 17: $\Theta(g(n))$

За всяка функция $g(n)$:

$$\Theta(g(n)) \stackrel{\text{def}}{=} \left\{ f(n) \mid \exists c_1, c_2 > 0 \; \forall n \nearrow \; : \; 0 \leqslant c_1 \cdot g(n) \leqslant f(n) \leqslant c_2 \cdot g(n) \right\}$$

Формално, $\Theta(g(n))$ е безкрайно множество от функции. Ако искаме да кажем, че $h(n)$ е една от тях, пишем $h(n) = \Theta(g(n))$ наместо формално коректното $h(n) \in \Theta(g(n))$. Причината е историческа – прието е да се използва знакът за равенство, а не формално коректният знак за принадлежност към множество. Изразът "$h(n) = \Theta(g(n))$" се чете: "$h(n)$ е Тета-голямо от $g(n)$".

Ето пример за педантично доказателство, че една функция е Тета-голямо от друга функция.

Определение 19: $O(g(n)), \Omega(g(n)), o(g(n)), \omega(g(n))$

За всяка функция $g(n)$:

$$O(g(n)) \stackrel{\text{def}}{=} \left\{ f(n) \mid \exists c > 0 \; \forall n \nearrow \; : \; 0 \leqslant f(n) \leqslant c \cdot g(n) \right\} \tag{2.17}$$

$$\Omega(g(n)) \stackrel{\text{def}}{=} \left\{ f(n) \mid \exists c > 0 \; \forall n \nearrow \; : \; 0 \leqslant c \cdot g(n) \leqslant f(n) \right\} \tag{2.18}$$

$$o(g(n)) \stackrel{\text{def}}{=} \left\{ f(n) \mid \forall c > 0 \; \forall n \nearrow \; : \; 0 \leqslant f(n) < c \cdot g(n) \right\} \tag{2.19}$$

$$\omega(g(n)) \stackrel{\text{def}}{=} \left\{ f(n) \mid \forall c > 0 \; \forall n \nearrow \; : \; 0 \leqslant c \cdot g(n) < f(n) \right\} \tag{2.20}$$

Както и при Тета-нотацията, принадлежността към тези множества означаваме не с "ϵ", а с "$=$", примерно пишем $f(n) = O(g(n))$, $h(n) = \omega(\phi(n))$ и така нататък.

Ако $f(n) = O(g(n))$, казваме, че $g(n)$ е *асимптотична горна граница* за $f(n)$. Примерно, вярно е, че:

$$n = O(n^2) \qquad n^2 = O(n^2) \qquad n^2 + 1000n + 10000 = O(n^2) \qquad 10n^2 = O(n^2) \qquad 1 = O(n^2)$$

Съответно, функцията n^2 е асимптотична горна граница за всяка от функциите n, n^2, $n^2 + 1000n + 10000$, $10n^2$ и 1.

От друга страна обаче, $\frac{1}{1\,000\,000}n^3 \neq O(n^2)$. Да видим защо. Да допуснем, че $\frac{1}{1\,000\,000}n^3 = O(n^2)$. Тогава от (2.17) знаем, че съществува положително c, такова че за всички достатъчно големи n е изпълнено:

$$\frac{1}{1\,000\,000}n^3 \leqslant c \cdot n^2$$

Разделяме на n^2 и получаваме еквивалентното неравенство

$$\frac{1}{1\,000\,000}n \leqslant c$$

което е същото като

$$n \leqslant 1\,000\,000\,c$$

Веднага се вижда, че това не може да е вярно. Колкото и голямо c да изберем, съществува n, което е по-голямо от $1\,000\,000c$. Ерго, функцията n^2 не е асимптотична горна граница за функцията $\frac{1}{1\,000\,000}n^3$.

Дуално, $\Omega(\,)$ и $\omega(\,)$ задават съответно *асимптотична долна граница* и *строга асимптотична долна граница*.

4. Свойства и гранични теореми (без доказателство).

?

5. Формулировка на теоремата за решенията на рекурентни отношения от вида T(1)=Θ(1), T(n)=a.T(n/b) + f(n), n>1.

Теорема 27: Мастър теорема (Theorem 4.1 от [31, стр. 94])

Нека $a \geqslant 1$ и $b > 1$ са константи и нека $f(n)$ е положителна функция. Нека

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) \tag{3.58}$$

където $\frac{n}{b}$ има смисъл или на $\left\lfloor \frac{n}{b} \right\rfloor$, или на $\left\lceil \frac{n}{b} \right\rceil$. Тогава асимптотиката на $T(n)$ е следната:

Случай 1 Ако $f(n) = O\left(n^{\log_b a}/n^{\epsilon}\right)$ за някоя положителна константа ϵ, то $T(n) = \Theta(n^{\log_b a})$.

Случай 2 Ако $f(n) = \Theta(n^{\log_b a})$, тогава $T(n) = \Theta\left(n^{\log_b a}. \lg n\right)$.
С други думи, $T(n) = \Theta(f(n). \lg n)$.

Случай 3 Ако са изпълнени следните условия:

1. $f(n) = \Omega\left(n^{\log_b a} \cdot n^{\epsilon}\right)$ за някоя положителна константа ϵ, и
2. съществува константа c, такава че $0 < c < 1$ и $\forall n \nearrow : a \cdot f\left(\frac{n}{b}\right) \leqslant c \cdot f(n)$,

то $T(n) = \Theta(f(n))$. $\square$