Задачи по изчислителна сложност

Йоан Василев

Януари 2025

Задача 1 (**Теорема на Райс за полуразрешими езици**)**.** Нека S е свойство на полуразрешимите езици. $code(S)$ е полуразрешим точно когато са изпълнени следните:

1) Ако $L \in S$ и $L \subseteq L'$ за L' полуразрешим, то $L' \in S$;

2) Ако $L \in S$ е безкраен, то съществува краен подезик на L също в S;

3) Множеството от безкрайните езици в S (кодирани като низове) е полуразрешимо;

Доказателство (следва [2]). Ще докажем двете посоки поотделно.
(⇒) Правата посока следва директно от следните три леми:

Лема 1

Ако $code(S)$ е полуразрешим, то $(\forall L \subseteq S)(\forall$ полуразрешим $L')(L \subseteq L' \to L' \in S)$, тоест S има "containment property" .

Д-во: Ще докажем контрапозитивното. Да допуснем противното. Нека $code(S)$ е полуразрешим и $\exists L, L' : (L \in S) \wedge (L \subseteq L') \wedge (L' \notin S)$. Нека $f : \Sigma^* \to \Sigma^*, f(\omega) = \ulcorner M \urcorner$, като

$$\mathcal{L}(M) = \begin{cases} L', & \text{ако} \quad \omega \in \overline{L_{diag}} \\ L, & \text{ако} \quad \omega \notin \overline{L_{diag}} \end{cases}$$

Тази функция е изчислима. Да опишем какво трябва да е действието на M. Нека $\mathcal{M}_{Accept}, \mathcal{M}_L$ и $\mathcal{M}_{L'}$ са машини съответно приемащи $\overline{L_{diag}}, L$ и L' (полуразрешими). По вход α симулира $\mathcal{M}_L$ върху α, ако думата се приеме, M също приема. Успоредно с това има втори процес. Симулира се $\mathcal{M}_{Accept}$ върху ω, ако бъде приета, той процесът продължава със симулация на $\mathcal{M}_{L'}$ върху α, ако и тя завърши в приемащо състояние, α се приема от M.
Тогава $\omega \in L_{diag} \Leftrightarrow \omega \notin \overline{L_{diag}} \Leftrightarrow \mathcal{L}(M) = L \Leftrightarrow \mathcal{L}(M) \in S \Leftrightarrow f(\omega) = \ulcorner M \urcorner \in code(S)$, значи $L_{diag} \leq_M code(S)$, или L_{diag} излиза полуразрешим, противоречие. □

Лема 2

Ако $L \in S$ е безкраен език и никой негов краен подезик не е от S, то $code(S)$ не е полуразрешим.

Д-во: Да допуснем противното, при тези условия $code(S)$ все пак е полуразрешим. Дефинираме $f : \Sigma^* \to \Sigma^*, f(\omega) = \ulcorner M \urcorner$, като

$$\mathcal{L}(M) = \begin{cases} L, & \text{ако} \quad \omega \notin L_{accept} \\ \text{крайно подмножество на } L, & \text{ако} \quad \omega \in L_{accept} \end{cases}$$

Да опишем какво трябва да е действието на M.

I начин: По вход α M симулира $\mathcal{M}_L$ върху α. Ако думата бъде приета, започваме симулация на $\mathcal{M}_{accept}$ върху ω. Ако ω *не* бъде приета в рамките на $|\alpha|$ стъпки, M приема α. Така, ако $\omega \in L_{accept}$ и се приема за t стъпки, ще приеме само думите от L с дължина $\leq t$, т.е. крайно подмножество.

II начин: M симулира едновременно α върху $\mathcal{M}_L$ и ω върху $\mathcal{M}_{accept}$. Ако първо ω бъде приета, отхвърляме α, ако първо α бъде приета, приемаме я. Така при $\omega \in L_{accept}$, M приема само тези $\alpha \in L$, които се приемат за по-малко стъпки от ω, т.е отново получаваме крайно подмножество на L.
В крайна сметка $\omega \in L_{diag} \Leftrightarrow f(w) \in code(S)$, при $code(S)$ полуразрешим L_{diag} също излиза такъв, противоречие. □

Лема 3

Ако $code(S)$ е полуразрешимо, то множеството на кодовете на крайните езици в S е полуразрешимо[a].

Д-во: Строим машина $\mathcal{M}$, която приема код на краен език $\omega = \ulcorner\omega_1 \cdots \omega_n\urcorner$, по който генерира машина $\mathcal{M}_\omega$ с език $\{\omega_1, \omega_2, \cdots, \omega_n\}$. Накрая $\mathcal{M}$ симулира $\mathcal{M}_{code(S)}$ върху $\ulcorner\mathcal{M}_\omega\urcorner$, ако кодът се приеме, то и $\mathcal{M}$ приема $\ulcorner\omega_1 \cdots \omega_n\urcorner$. □

[a] В [2] вместо това се използва *изброимо* в смисъл, че съществува машина-"генератор", която (без вход) печата (номерира) думи във формата $\sharp\omega_1\sharp\omega_2\sharp\cdots$. Лесно се доказва, че език е полуразрешим точно когато съществува генератор, който номерира думите му.

(⇒) Сега обратната посока, нека 1), 2) и 3) са в сила, ще построим машина $\mathcal{M}$, която по вход ω определя дали $\omega \in code(S)$. $\mathcal{M}$ последователно генерира двойки $(code, j)$, където $code$ е код на краен език (а не машина) от S (това можем да направим, защото от *3)* знаем, че това множество е изброимо/полуразрешимо). Сега $\mathcal{M}$ започва parsing на думите от $code$ и проверява за j стъпки дали всяка от тях е в $\mathcal{L}(\mathcal{M}_\omega)$, ако това е така, приемаме ω.
Причината горното да работи коректно е, че всеки безкраен език от S има краен подезик пак там (така че няма да изпуснем никой) и че всеки безкраен полуразрешим надезик на краен такъв от S също е от S (така че няма как и да включим нещо, което не е в S). ■

Задача 2 (Универсална машина на Тюринг за $n.log(n)$). [Hennie and Stearns] Съществува универсална машина на Тюринг $\mathcal{U}$ такава, че по дадена двойка $\ulcorner\omega, \alpha\urcorner$, където $\omega, \alpha \in \{0,1\}^*$, $M_w(\alpha) = \mathcal{U}(\ulcorner\omega, \alpha\urcorner)$ и ако M спира върху α за стъпки, то $\mathcal{U}$ спира α за $cT.log(T)$ стъпки, където c е константа, зависеща само от M_ω.

Следствие. *Ако M_ω е ограничена по време от $T(n)$, то $\mathcal{U}$ симулира M_ω за време $O(T(n).logT(n))$.*

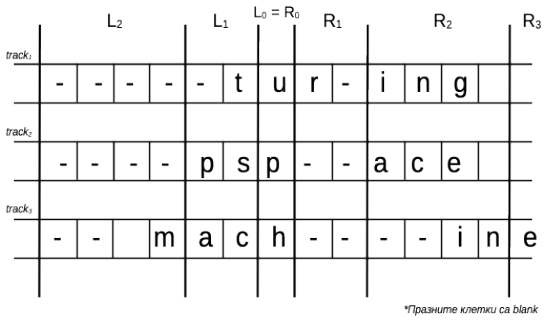
Доказателство (следва [1]). Да започнем с общо представяне начина на действие на $\mathcal{U}$. Правим машината с 3 ленти - на първата е входът (съответно и описанието на делта функцията на M_ω), втората лента помни в кое състояние сме (от тези на M_ω), а в третата става симулацията на изчислението. За да симулира една стъпка от изчислението на M_ω, универсалната машина прави няколко неща. Сканира текущото състояние и делта функцията и определя новото състояние, символите, които ще се напишат, и движенията на главите. Отделно от същинската симулация на M_ω, този процес отнема допълнително време K, зависещо от само от M_ω и кодирането ѝ.

Сега по същинската симулация на стъпката в M_ω: Понеже последната може да е с произволен брой ленти (нека да са k), а $\mathcal{U}$ е с константен брой такива, то ни трябва начин за кодиране, използваме идеята за разделяне на лентата на k различни потока, по един за всяка лента (в литературата "tracks"). Проблемът тук е, че обикновена симулация на движенията на k-те глави води до квадратично забавяне. Затова тук използваме друга идея - вместо да движим главите по потоците, ще движим (отместваме) потоците така, че винаги в клетка 0 да са тези клетки от тях, в които би била съответната глава. Така например вместо да придвижим главата на поток i с една позиция надясно, ще отместим самия поток с една наляво. Съответно истинската глава на лентата на $\mathcal{U}$, съдържаща потоците, ще бъде застопорена върху клетка 0, изключаю времето, в което ще се грижи за отместването на потоците.

За целта правим разделяне на лентата с потоците на блокове (както на фигурата по-долу). В

блок 0 са нулевите клетки на потоците. Вляво и вдясно от него са съответно блоковете L_1, R_1, които ще съдържат по 2 клетки от всеки поток. Всеки следващ блок съдържа 2 пъти повече клетки от предишния. Така блокове L_i, R_i съдържат съответно клетките с номера $(-2^{i+1}+1, -2^i+1]$ и $[2^i-1, 2^{i+1}-1)$. Целта на това е в блоковете да се държат празни буферни клетки, които ще премахнат нуждата от отместване на целия поток при всеки ход, като вместо това ще се отмества само част от него (за сметка на буфера). За простота на изложението ще се фокусираме върху клетките от само един поток (те са независими от тези на другите).
Ефективността на операциите по отместване гарантират следните инварианти:

- Всеки блок е пълен, празен или наполовина пълен, като за празни считаме клетките с маркиращ символ $- \notin \Gamma$ (клетките $\sqcup$ считаме запълнени),
- Общият брой символи, различни от $-$, в $L_i \cup R_i$ винаги е 2^i,
- Нулевият блок винаги е запълнен (със символ, различен от $-$);



Остава да обясним как работят самите отмествания на потоците (track shifts). Да кажем, че искаме да преместим главата на i-тата лента на M_ω надясно, т.е. ще отместваме поток i наляво.

1. $\mathcal{U}$ намира първото i, за което блок R_i не е празен (тоест е наполовина или изцяло пълно). Отбелязваме, че заради инварианта това е и първото i, за което блок L_i не е пълен.
2. Копираме съдържанието на първата непразна (но и различна от $\sqcup$) клетка от блока R_i на позиция 0 (върху нея трябва да се премести главата). Всички останали клетки от блока също "придвижваме" (по-точно копираме) вляво, в досега празните блокове $R_1, R_2, ...R_{i-1}$ (ако се R_i е изцяло пълен, първата му половина от клетки ще остане запълнена със символи), като важното е, че всеки блок $R_1, ..., R_{i-1}$ запълваме само наполовина (първата му половина) с цел ефективност. Това е възможно, понеже броят запълнени клетки в блок R_i (изключваме първата, преместена в блок 0) е $\leq 2^i - 1 = (\sum_{j=1}^{i-1} 2^{j-1}) + 2^{i-1}$, като последните 2^{i-1} в сметката са показват, че при нужда R_i ще остане наполовина пълен.
3. Остава да отместим и лявата част на потока, правим обратния на горния процес. Напомняме, че преди операцията блоковете $L_0, ..., L_{i-1}$ бяха пълни, а L_i не е пълен, така че има място да събере половината от тяхното съдържание. Придвижваме (копираме) наляво символите от блокове $L_0, ..., L_{i-1}, L_i$ в блокове $L_1, ..., L_{i-1}, L_i$, като и този път блокове $L_1, ..., L_{i-1}$ запълваме наполовина.

Важното е, че след приложените операции инвариантността остава в сила. Уточняваме, че за отместването ще е необходима допълнителна лента (или няколко такива) за временно копиране на информацията, това обаче не променя крайната сложност (добавя константен коефициент).
Колкото до сложността им, един такъв процес по отместване става за време $O(2^i)$. Да отбележим, че $i \leq log_2 T$, защото броят операции по преместване на главите на лентите е ограничен от T, а за толкова стъпки могат да се достъпят блокове с номера най-много $i \leq \lceil log_2 T \rceil$. Важното е, че за да се редактират отново L_i/R_i и да се стигне пак до такъв порядък на сложността, блоковете $R_1, ..., R_{i-1}$, които оставихме наполовина пълни, трябва преди това да са се изпразнили, което може да се случи след най-малко още 2^{i-1} операции по отместване. Тоест за всеки T стъпки от изчислението до сложност $O(2^i)$ на отместването можем да стигнем в $\frac{T}{2^{i-1}}$ от тях. Тогава изчислението в

$\mathcal{U}$ става за не повече от време

$$KT + \sum_{i=0}^{\lceil logT \rceil} \frac{T}{2^{i-1}} O(2^i) = O(T.logT) \quad \blacksquare$$

Задача 3 (**Теорема за линейна компресия, linear compression**). За всяка $k-$лентова *off-line* машина на Тюринг M_1, ограничена по памет от $S(n)$, и константа $c > 0$, съществува (еднолентова) *off-line* машина на Тюринг M_2, ограничена по памет от $S(n)$, такава, че $L(M_1) = L(M_2)$. При това, ако M_1 е детерминирана, то и M_2 също е.

Доказателство. Ще направим доказателството на две стъпки - първо ще покажем, че съществува *off-line* машина M_2 с произволен брой ленти, отговаряща на останалите условия, а после ще видим как тя може да бъде сведена до еднолентова такава.

Идеята за конструирането на M_2 е следната: съдържанието на i-тата работна лента на M_1 държим в i-тата работна лента на M_2, като всеки d символа обаче (конкретната стойност ще бъде уточнена по-долу) от M_1 групираме в един сложен символ/клетка в M_2, който пък е просто наредена d-орка от букви, потенциално с маркер $\widehat{a}$ за мястото на главата върху някой от тях. По-формално $\Gamma_2 = (\Gamma_1 \times \widehat{\Gamma_1})^d$ (с допълнителното изискване най-много един от символите да има маркер за глава на лента).

Самото действие на M_2 не се различава особено от това M_1 с изключение на движенията на главите. - Когато стъпката не предполага излизане от "групата" от d клетки, оставаме в същата M_2 клетка, като само сменяме d-орката от символи в клетката (съответно придружено с актуализиране на маркера на главата), реално движение в съседна клетка на M_2 има само когато се налага излизане от "групата". Колкото до делта функцията, за детерминирана МТ тя би изглеждала горе-долу така (аналогично се обобщава за недетерминирани):

- ако $\delta_1(q, b) = (p, x, \lhd)$, дефинираме $\delta_2(q, \cdots a\widehat{b} \cdots) = (p, \cdots \widehat{a}x \cdots, \square)$, а също $\delta_2(q, \widehat{b} \cdots) = (p', x \cdots, \lhd)$ и $\delta_2(p', \cdots a) = (p, \cdots \widehat{a}, \square)$,
- ако $\delta_1(q, b) = (p, x, \square)$, дефинираме $\delta_2(q, \cdots \widehat{b} \cdots) = (p, \cdots \widehat{x} \cdots, \square)$,
- ако $\delta_1(q, b) = (p, x, \rhd)$, дефинираме $\delta_2(q, \cdots \widehat{b}c \cdots) = (p, \cdots x\widehat{c} \cdots, \square)$, а също $\delta_2(q, \cdots \widehat{b}) = (p', \cdots x, \rhd)$ и $\delta_2(p', c \cdots) = (p, \widehat{c} \cdots, \square)$;

Важна за нас е паметта: така построената M_2 ползва не повече от $\lceil \frac{S(n)}{d} \rceil$ клетки на всяка работна лента. Оттук $S_2 \leq \lceil \frac{S(n)}{d} \rceil$. Тук е моментът да изберем d такова, че $\frac{2}{d} \leq c$. Ако $\frac{S(n)}{d} \leq 1$, то M_2 с нуждае от не повече от една клетка по всяка лента, откъдето $S_2 \leq 1$. В противен случай $\frac{S(n)}{d} > 1 \Rightarrow \lceil \frac{S(n)}{d} \rceil \leq 2\frac{S(n)}{d} \leq cS(n)$. И в двата случая $S_2 \leq max(1, cS(n))$, което и ни трябва. $\square$

Сега можем да сведем $k-$лентовата M_2 до еднолентова такава по познатия начин за свеждане (този от доказателството за еквивалентност между многолентови и еднолентови машини на Тюринг, всеки символ на еднолентовата машина е наредена k-орка). Въпросната конструкция запазва дължината на лентите (по-точно на най-дългата от лентите), като това не влияе на използваната памет, тоест оставаме в $cS(n)$. $\blacksquare$

Задача 4 (**Теорема за линейното ускорение, linear speedup**). Ако L се приема от $k-$лентова машина M_1, ограничена по време от $T(n)$, където $k > 1$ и $n \in o(T(n))$, тогава за всяко $c > 0 : L$ се приема от $k-$лентова машина M_2, ограничена по време от $cT(n)$. При това, ако M_1 е детерминирана, то M_2 също е.

Доказателство. Идеята отново е всяка клетка от M_2 да кодира група от m (конкретна стойност ще дадем по-късно) клетки от M_1, а това компресиране ни позволява с няколко хода в M_2 (в рамките на една клетка) да симулираме ходовете върху цяла група от клетки от M_1, което намалява и времето за работа.
За целта i-тата лента от M_1 ще се симулира от i-тата лента на M_2. Уточняваме, че в началото M_2

копира входа (некомпресиран) върху някоя от другите си ленти, като кодира по m символа в един. От този момент нататък считаме, че лентата с компресирания вход е входната лента, а старата входна вече е работна лента.

Да дадем повече детайли за работата на M_2. Понеже искаме да си гарантираме, че за константен брой стъпки в M_2 (в нашия случай ще бъдат 7) се симулират поне m хода от изчислението на M_1, разглеждаме всяка клетка на M_2 (припомняме, че тя съдържа информация от m клетки на M_1) заедно със съседите ѝ (ляв и десен). Причината е, че след m хода от изчислението на M_1, главата на съответната лента в M_2, ако първоначално е била в клетка j, ще бъде в $j-1$, j или $j+1$. Дефинираме $Q_{M_2} = (Q \cup \bigcup\{Q_i | 1 \leq i \leq 5\}) \times (\Gamma \cup \widehat{\Gamma})^3$, където Q, Γ са от M_1, а Q_i са допълнително индексирани копия на състоянията. Главни насоки за ламбда функцията:

- $\delta_2([q, \cdots, \cdots, \cdots], mid) = ([q_1, \cdots, mid, \cdots], mid, \lhd)$ - извлича информация за средния интервал
- $\delta_2([q_1, \cdots, mid, \cdots], l) = ([q_2, l, mid, \cdots], l, \rhd)$ - извлича информация за левия интервал
- $\delta_2([q_2, l, mid, \cdots], mid) = ([q_3, l, mid, \cdots], mid, \rhd)$ - връща се в средата
- $\delta_2([q_3, l, mid, \cdots], r) = ([q_4, l, mid, r], r', \lhd)$ - извлича информация за десния интервал, актуализира съдържанието му
- $\delta_2([q_4, l, mid, r], mid) = ([q_5, l, mid, r], mid', \lhd)$ - актуализира съдържанието на средния интервал
- $\delta_2([q_5, l, mid, r], l) = ([q_6, l, mid, r], l', \rhd)$ - - актуализира съдържанието на левия интервал
- $\delta_2([q_6, l, mid, r], \cdots) = ([p, \cdots, \cdots, \cdots], \cdots, \rhd)$ или $([p, \cdots, \cdots, \cdots], \cdots, \lhd)$ в зависимост случая;

Получаването на актуализираните състояния l', mid', r' (тоест какво трябва да има на мястото на клетките след m-те стъпки от симулацията на M_1) се закодира предварително в делта функцията и така не отнема време в M_2. Тук правим допълнителните уточнения, че ако преждевременно (някъде по средата на $m-$те стъпки) M_1 спре, трябва и M_2 да. Също така горедефинираната функция показва какво става върху една от лентите при детерминирана машина, но в общия случай се разглеждат вектори и множества състояния (за недетерминирани). От всичко казано M_2 е работи за време:

$$T_2 \leq \underbrace{n + \lceil \frac{n}{m} \rceil}_{\text{за входа}} + 7\lceil \frac{T(n)}{m} \rceil \leq n + \frac{n}{m} + 7\frac{T(n)}{m} + 8$$

По допускане $n \in o(T(n)) \Rightarrow lim_{n\to\infty} \frac{T(n)}{n} = \infty \Rightarrow \forall d > 0\ \exists N\ \forall n \geq N : d < \frac{T(n)}{n}$. Оттук за $n \geq max(N, 8) : T_2 \leq 2n + \frac{n}{m} + 7\frac{T(n)}{m} < 2\frac{T(n)}{d} + \frac{T(n)}{dm} + 7\frac{T(n)}{m} = T(n)[\frac{2}{d} + \frac{1}{dm} + \frac{7}{m}]$. При избор на $m \geq \frac{16}{c}$ и $d \geq \frac{4}{c} + \frac{1}{8}$: $T_2 < cT(n)$, което и искахме. ■

Задача 5 (**Blum's Speedup Theorem**)**.** Нека $r : \mathbb{N} \to \mathbb{N}$ е тотална изчислима функция. Тогава съществува разрешим език L такъв, че за всяка машина на Тюринг M_i с $\mathcal{L}(M_i) = L$ съществува друга машина на Тюринг M_j с $\mathcal{L}(M_j) = L$ така, че $r(S_j(n)) \leq S_i(n)$ за почти всяко n.

Доказателство (следва [2]). Ще започнем доказателството с две помощни леми:

Лема 4

Ако r е тотална изчислима функция, то съществува ненамаляваща функция r', напълно построима по памет ("fully time-constructible") такава, че: $r'(n) > r(n)$ и $r'(n) > n^2$ $(\forall n)$.

Д-во: Следваме модела, използван в [2], в който за кодирането на числа се ползва унарно представяне (0^i). Разглеждаме машина $\mathcal{B}$, която за вход с дължина n последователно пресмята $r(i)$ $\forall i \leq n$ върху една и съща лента (всяка нова стойност на функцията пише директно върху старата), а накрая, ако се налага, маркира допълнително клетки от въпросната лента така, че броят на посетените стане n^2. Последното ни гарантира,

че $S_{\ulcorner\mathcal{B}\urcorner}(n) \geq n^2$, изчислението на $r(n)$ гарантира, че $S_{\ulcorner\mathcal{B}\urcorner}(n) \geq r(n)$ (заради унарното кодиране на числа), а пресмятането на предните стойности $r(i)$ $\forall i < n$ гарантира, че $S_{\ulcorner\mathcal{B}\urcorner}(n) \geq S_{\ulcorner\mathcal{B}\urcorner}(n-1)$, т.е. е ненамаляваща. Така намерихме функция $r' = S_{\ulcorner\mathcal{B}\urcorner}$ с исканите свойства. □

Сега към основната ни задача. Можем да считаме, че функцията r от условието е ненамаляваща, напълно построима по памет и $r(n) \geq n^2$. Причината е, че ако това не е изпълнено, можем да вземем функцията, базирана на r, чието съществуване лемата предполага. Ако теоремата на Блум е в сила за нея (а ние ще покажем, че е), то тя ще е в сила и за оригиналната функция.

Лема 5

Ако r напълно построима по памет, то съществува напълно построима по памет функция h такава, че: $h(0) = 2$ и $h(n) = r(h(n-1)) = r^n(2)$.

Д-во: Нека $\mathcal{R}$ е машина свидетел за това, че r е напълно построима по памет. Построяваме машина $\mathcal{H}$ с 2 работни ленти, като в началото върху първата стои низът 11. Машината $\mathcal{H}$ симулира $\mathcal{R}$ върху втората работна лента точно n пъти, като всеки път приема за вход съдържанието на първата лента, а след всяка такава итерация върху първата лента записва низа 1^{cells}, където *cells* са използваните клетки от втората работна лента при последната симулация на $\mathcal{R}$. Един детайл е, че за да използва точно $r^n(2)$ клетки, не трябва при симулациите излишно да се достъпват клетки $\sqcup$ по краищата на думата, затова машината, след края на всяка итерация, ще сменя посоката на писане (дясно - ляво - дясно - $\cdots$). □

Разглеждаме такава h.
Нека $M_1, M_2, \cdots$ е наредба на машините на Тюринг и да приемем, че дължината на кода на M_i е $\lceil log_2 i \rceil$. Идеята ни ще е да конструираме разрешим език L със следните свойства.

1. Ако $L(M_i) = L$, то $S_i(n) \geq h(n-i)$ за почти всички n ($\overset{\infty}{\forall} n$)
2. $\forall k\ \exists j : (L(M_j) = L) \wedge (S_j(n) \leq h(n-k))$

Такъв език би доказал теоремата, защото за всяко i такова, че $L(M_i) = L$, за почти всички n е в сила $S_i(n) \geq h(n-i) = r(h(n-i-1)) \geq r(S_j(n))$, като съществуването на j се гарантира от 2. за $k = i+1$.

Ще дефинираме $L \subseteq 0^*$ с горните свойства чрез алгоритъм. За целта въвеждаме понятието "отписана" машина, с което ще определяме някои M_i (всъщност такива, които със сигурност не отговарят на условие 1). За всяко $n = 0, 1, 2, \cdots$ последователно определяме

$$\sigma(n) := min\{j \leq n |\ Mj \text{ все още не е } \textit{отписана} \text{ и } S_j(n) < h(n-j)\};$$

Ако такова $\sigma(n)$ съществува, *отписваме* и машина $M_{\sigma(n)}$, а думата 0^j приемаме точно когато $0^j \notin L(M_{\sigma(n)})$. Всъщност тук просто правим диагонализация.

(св-во 1): Ще докажем, че L изпълнява 1. чрез контрапозитивното твърдение. Нека за някое $i : S_i(n) < h(n-i)$ за безброй много n *(*)*. За всички машини с по-малки индекси $j < i$, които някога ще бъдат отписани, това ще се случи в краен момент, преди достигане на някое определено n_0. Заради *(*)* съществува $n > max\{n_0, i\} : S_i(n) < h(n-i)$. По дефиницията на алгоритъма и защото всички по-малки j, които някога ще бъдат отписани, вече са, то M_i следва да бъде отписана ($\sigma(n) = i$). Значи $0^n \in L \Leftrightarrow 0^j \notin L(M_i)$, откъдето $L(M_i) \neq L$. ✓

(св-во 2): Нека k е фиксирано. Искаме да построим $M : L(M) = L$ и $S_{\ulcorner M\urcorner}(n) \leq h(n-k)$. По построение произволна $0^n \in L \Leftrightarrow 0^n \notin L(M_{\sigma(n)})$, но за определяне на последното трябва да знаем кои машини вече са били отписани (при $l < n$). Отново отчитаме факта, че за всички машини с индекси $i \leq k$, които някога ще бъдат отписани, това ще се случи преди достигане на някое определено $n_1 = const$. С цел пестене на памет, което ще се окаже решаващо, в състоянията на M предварително записваме всички въпросни кодове на машини с индекси $i \leq k$, които някога ще бъдат отписани,

както и информация дали $0^n \in L$ ($\forall n \leq n_1$) (всички думи в езика с дължина $\leq n_1$). За проверка на думите $0^n, n > n_1$ първо трябва да допълним списъка с отхвърлени машини (те ще са с индекси i между k и n), така че последователно ще трябва да симулираме $\forall l, n_1 < l \leq n\ \forall i, k < i \leq n : M_i$ върху 0^l и да видим дали индекс i се отписва от l, като тази симулация има смисъл да трае, докато използваната памет $\leq h(l-i)$ (тук ползваме факта, че функцията е построима по памет, за да разберем точната стойност, пускаме машината $\mathcal{H}$, за която $S_{\ulcorner\mathcal{H}\urcorner} = h$, върху вход 1^{l-i}). Понеже $l - i \leq n - k - 1$, то една такава симулация ползва памет $h(l-i) \leq h(n-k-1)$ (отчитаме, че h е ненамаляваща), но понеже симулацията на M_i не може да е съвсем директна (има някаква допълнителна константа), това се умножава по паметта, използвана за кодиране на символите от лентата на M_i (тази памет е $\leq \lceil log_2 i \rceil$, понеже кодът на цялата машина заема толкова). В крайна сметка една симулация заема памет $\leq h(n-k-1).\lceil log_2 n \rceil \leq h(n-k-1)2^{n-k-1}$ $(\overset{\infty}{\forall} n) \leq h(n-k)$, като тук ползваме, че $h(x) \geq 2^{2^x}$, заради $r(x) \geq x^2$ и $h(x+1) = r(h(x))$. Допълнително M се нуждае от памет, в която да помни списък от отписаните машини с индекси между k и n. По предположение всяка такава има код с размер $\lceil log_2 i \rceil \leq \lceil log_2 n \rceil$, обща памет $nlog(n) \leq h(n-k)$ $\overset{\infty}{\forall} n$. От това и паметта, нужна за симулациите, използвана памет на машината е $\leq h(n-k)$. ✓
Да отбележим, че L е разрешим, защото дефинираната M е разрешител за него. ■

Задача 6 (**Теорема на Имерман-Селепчени**)**.** За всяка $S(n) \geq log(n)$ е в сила $NSPACE(S(n)) =$ co-$NSPACE(S(n))$

Доказателство. Ще използваме наготово, че $NL =$ co-NL, както и че $PATH$, $\overline{PATH} \in NL$ (доказано на лекции).

*I начин (**"padding argument"**):* Нека $L \in NSPACE(S(n))$, като M_1 свидетелствува за това. Дефинираме езика $L_{pad} := \{\omega\sharp^{2^{S(|\omega|)}-|\omega|} \mid \omega \in L\}$. Построяваме M_2, която да приема L_{pad} за памет $log(n)$. По вход α машината проверява дали той е от вида $z\sharp^{2^{S(|z|)}-|z|}$, ако не е, отхвърля (тази проверка става за памет $log(|\alpha|)$). Ако думата не е отхвърлена, M_2 симулира M_1 върху z (като се ползва входът на M_2 и символите $\sharp$ се възприемат като $\sqcup$), за това е необходима памет $\leq S(|z|) = log_2(2^{S(|z|)} - |z| + |z|) = log(|\alpha|)$. Следва, че M_2 е ограничена от $log(n)$, откъдето $L_{pad} \in NL \Rightarrow \overline{L_{pad}} \in$ co-$NL = NL$. Нека свидетел за това е M_3, където $L(M_3) = \overline{L_{pad}}$ и M_3 работи за памет $log(n)$.
Сега строим M_4, която по вход ω симулира M_3 върху $\omega\sharp^{2^{S(|\omega|)}-|\omega|}$, като за да не се копира цялата опашка от $\sharp$, което би заело много място, просто си представяме, че зад ω стоят съответните $\sharp$, това може да стане с допълнителен указател за позиция, необходимата памет за него е $log(|\omega\sharp^{2^{S(|\omega|)}-|\omega|}|) = log(2^{S(|\omega|)}) = S(|\omega|)$. За симулацията на M_3 също е необходима памет $log(|\omega\sharp^{2^{S(|\omega|)}-|\omega|}|) = S(|\omega|)$, значи M_4 е ограничена по памет от $S(n)$. При това M_4 приема ω точно когато M_3 приема $\omega\sharp^{2^{S(|\omega|)-|\omega|}}$ (т.е. $\omega\sharp^{2^{S(|\omega|)-|\omega|}} \in \overline{L_{pad}}$) точно когато $w \notin L$, откъдето $L(M_4) = \overline{L} \Rightarrow \overline{L} \in NSPACE(S(n)) \Rightarrow L \in$ co-$NSPACE(S(n)) \Rightarrow NSPACE(S(n)) \subseteq$ co-$NSPACE(S(n))$. За обратната посока бихме могли да приложим аналогичен аргумент (но обърнат), но от горното включване директно $L \notin$ co-$NSPACE(S(n)) \Rightarrow L \notin NSPACE(S(n))$ и $L \in$ co-$NSPACE(S(n)) \Rightarrow \overline{L} \in NSPACE(S(n)) \Rightarrow \overline{L} \in$ co-$NSPACE(S(n)) \Rightarrow L \in NSPACE(S(n))$, или $L \in NSPACE(S(n)) \Leftrightarrow L \in$ co-$NSPACE(S(n))$, откъдето $NSPACE(S(n)) =$ co-$NSPACE(S(n))$. ■

Задача 7 (**NTIME(f) $\subseteq$ DSPACE(f)**)**.** За функция f са в сила:

- $DTIME(f) \subseteq DSPACE(f)$
- $NTIME(f) \subseteq NSPACE(f)$
- $NTIME(f) \subseteq DSPACE(f)$
- $NP \subseteq PSPACE$

  Доказателство.

- Причината е, че t стъпки машина може да използва най-много толкова клетки t. ■

- Аналогично на горното. ■
- Нека $L(\mathcal{N}) = L$ и $\mathcal{N}$ е недетерминирана, ограничена по време от f. Въвеждаме наредба във всяко от множествата $\Delta(q, x)$. Нека максималната мощност на такова множество е r, това е горна граница за разклонеността на дървото на изчисленията, тоест всеки клон на изчислението може да бъде описан с редица от символите $x_1, \cdots, x_r$, като ако i-тият символ в редицата е x_j, трябва да се продължи по j-тия път от наредбата, въведена в съответното $\Delta(q_{current}, a)$, стига такъв да има. Строим детерминирана M, като на първата работна лента генерираме всички думи от $\{x_1, \cdots, x_r\}^*$ в каноничната им наредба, това ще показва пътя на изчислението (разбира се, не всички такива редици са валидни изчисления). Да отбележим, че заради ограничеността на $\mathcal{N}$ от f, височината на всеки клон, а оттам и на цялото дърво е не повече от $f(|\omega|)$ (при вход $\omega, |\omega| = n$), значи е достатъчно да генерираме редици с дължина не повече от $f(n)$, откъдето първата лента ползва памет $\leq f(n)$. Понеже не знаем точната стойност на $f(n)$, ще ни трябва брояч, който да отчита дали всички изчисления са завършили. Когато броячът се изравни с броя листа r^h (h е текущата височина на дървото, тоест дължината на редицата от първата лента), можем да спрем. Отчитаме, че за един такъв брояч е необходима памет $\leq log_r(r^h) = h \leq f(n)$ (броим в основа r).
  Остава самата симулация, нея правим върху друга лента. Понеже тя трае не повече от $f(n)$ хода, то не са сканирани повече от също толкова клетки. В крайна сметка всяка лента ползва $\leq f(n)$ памет, тоест $L \in DSPACE(f(n))$. ■
- Първо можем да покажем, че $SAT \in PSPACE$ (например директна проверка на всички валюации, като за целта преизползваме памет, би свършила работа). Ползваме, че SAT е NP-пълен $\Rightarrow \forall L \in NP : L \leq_P SAT$, т.е. имаме полиномиално свеждане по време, откъдето и по памет. Това, комбинирано със $SAT \in PSPACE$, влече $L \in PSPACE$. ■

Задача 8 (2-SAT принадлежни на класа P). $2\text{-}SAT \in PTIME$.

Доказателство. Разглеждаме произволна формула в 2-CNF (конюнктивна нормална форма, като във всеки дизюнкт има точно два литерала), нека тя е с общ вид $\phi = (a_1 \vee a_2) \wedge (a_3 \vee a_4) \cdots (a_{m-1} \vee a_m)$, където $a_j \in \{x_i \mid 1 \leq i \leq n\} \cup \{\overline{x_i} \mid 1 \leq i \leq n\}$, като последните две множества са реалните променливи от формулата и техните отрицания. Можем да запишем формулата алтернативно като $\phi = (\overline{a_1} \to a_2) \wedge (\overline{a_2} \to a_1) \wedge (\overline{a_3} \to a_4) \cdots (\overline{a_{m-1}} \to a_m) \wedge (\overline{a_m} \to a_{m-1})$, импликациите тук напомнят за ребра в граф. Разглеждаме насочен граф G с върхове $V = \{x_i \mid 1 \leq i \leq n\} \cup \{\overline{x_i} \mid 1 \leq i \leq n\}$ и ребра $E = \{(a_i, a_j) |\ a_i, a_j \in V \wedge (\overline{a_i} \vee a_j)$ е клауза във ϕ /с точност размяна на местата на променливите/$\}$.

Лема 6

Формулата ϕ е удовлетворима точно когато не съществува $1 \leq i \leq n$ такова, че в G между x_i и $\overline{x_i}$ има насочени пътища (и в двете посоки).

Д-во:
($\Rightarrow$) Ще докажем контрапозитивното. Нека между x_i и $\overline{x_i}$ има насочени пътища в G, тогава ϕ не е удовлетворима. Ако редиците от върхове в двата насочени пътя са съответно $x_i, v_1, \cdots v_d, \overline{x_i}$ и $\overline{x_i}, u_1, \cdots u_t, x_i$, то $(\overline{x_i} \vee v_1), (\overline{v_1} \vee v_2) \cdots (\overline{v_d} \vee \overline{x_i})$ и $(\overline{\overline{x_i}} \vee u_1), (\overline{u_1} \vee u_2) \cdots (\overline{u_t} \vee x_i)$ са клаузи във ϕ (според дефиницията на ребрата по-горе). Но да забележим, че $(\overline{x_i} \vee v_1), (\overline{v_1} \vee v_2) \cdots (\overline{v_d} \vee \overline{x_i})$ не могат да са едновременно изпълнени, ако $x_i = 1$, а $(x_i \vee u_1), (\overline{u_1} \vee u_2) \cdots (\overline{u_t} \vee x_i)$ не могат да са едновременно изпълнени, ако $x_i = 0$. Но тогава за да има решение ϕ, $x_i \neq 0$ и $x_i \neq 1$, невъзможно, откъдето ϕ не е удовлетворима.

($\Leftarrow$) Да уточним, че в G v, u са от една силно свързана компонента точно когато има насочен път от v към u и обратно. При това, заради дефиницията на графа, v, u са от една компонента точно когато $\overline{v}, \overline{u}$ също са от една (под $\overline{u}$ разбираме *върха*, съответстващ на противоположния литерал на този, на които u съответства, а не самата стойност), а ϕ е удовлетворима само ако всички променливи в компонента имат еднаква стойност. Изобщо забелязва, че за да има стойност 1 формулата, стойностите на променливите трябва да са такива, че ако има ребро $(a_i, a_j) \in E$, то $a_i \to a_j = 1$ (това следва просто от дефиницията на ребрата, а и естествената аналогия между ребро и импликация).

Нека G' е кондензационният граф на G (всички върхове от една силно свързана компонента се разглеждат като един, ребрата са снопове от ребрата между компонентите в G), той няма цикли. Абстрахираме се от това, че един връх в новия граф съответства на няколко променливи от стария (те така или иначе трябва да са с еднаква стойност в оценката). Разглеждаме топологично сортиране на върховете на G' (променливите), отзад напред. За всяка следваща променлива в редицата, ако още не ѝ е зададена стойност, заменяме я с 1, а обратната ѝ (с черта отгоре) съответно с 0.
Да допуснем, че този алгоритъм е некоректен, значи има клауза със стойност 0, респективно ребро (a_i, a_j), за което $a_i = 1, a_j = 0$. Оттук $\overline{a_i} = 0, \overline{a_j} = 1$. Отбелязваме, че според алгоритъма, променлива може да бъде остойностено с 0 точно когато нейната спрегната бъде остойностена с 1. Остойностяването може да е станало по два начина:

1. първо a_j е остойностено с 0, а на a_i впоследствие е дадена стойност 1. За да е било $a_j = 0$ обаче, трябва $\overline{a_j}$ вече да е било станало 1. Същевременно на $\overline{a_i}$ е дадена стойност 0 едва след като е дадена стойност на самото a_i, значи на $\overline{a_j}$ се дава стойност преди да се даде на $\overline{a_i}$, но това противоречи на посоката на реброто $(\overline{a_j}, \overline{a_i})$ и топологичната сортировка, от която се очаква $\overline{a_i}$ да е остойностено преди $\overline{a_j}$.

2. първо a_i е остойностено с 1, а на a_j впоследствие е дадена стойност 0. Този случай е аналогичен (огледален) на предния, отново стигаме до противоречие.

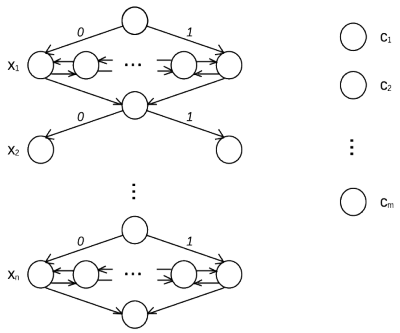
Това означава, че ребра от вида $(1, 0)$, а съответно и клаузи от $(1 \to 0)$ няма, значи е намерена валюация, за която ϕ е удовлетворима. □

Построяваме машина $\mathcal{M}$, която по дадена формула ϕ първо конструира графа G. Ясно е, че това става за полиномиално време (а даже и за линейно $O(n)$). Сега е достатъчно да използваме необходимото и достатъчно условие, което лемата ни дава. За всеки два върха x_i и $\overline{x_i}$ в графа симулираме $\mathcal{M}_{PATH}$ (разрешител за проблема $PATH \in P$, който работи за полиномиално време) първо върху вход $\ulcorner G, x_i, \overline{x_i} \urcorner$, а после и върху $\ulcorner G, \overline{x_i}, x_i \urcorner$. Ако $\mathcal{M}_{PATH}$ отхвърли наличието на поне единия път, добре, продължаваме със следваща двойка върхове, ако се окаже, че и двата пътя съществуват, $\mathcal{M}$ отхвърля ϕ. Така $\mathcal{L}(\mathcal{M}) = \{\phi \mid \phi \text{ е удовлетворима 2-CNF формула}\} = 2SAT$.
Всяка симулация на $\mathcal{M}_{PATH}$ работи за полиномиално време, а броят симулации съответства на броя на всички променливи, отново линеен по $|\phi|$. Следва, че $\mathcal{M}$ работи за полиномиално време и 2-$SAT \in P$. ■

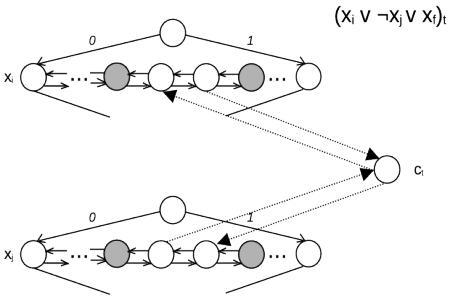
Задача 9 (**HAMPATH е NP-пълен**). Проблемът за съществуване на Хамилтонов път HAMPATH $=\{\ulcorner G \urcorner |$ в G има Хамилтонов път$\}$ е NP-пълен.

Доказателство. Принадлежността е се доказва тривиално. Достатъчно е недетерминистично да се генерират различните пермутации от върхове на графа (или по-точно всички последователности от върхове с дължина $|V|$). После детерминистично се проверява дали така генерираната последователност е валиден Хамилтонов път, ако е, приемаме.
Сега към пълнотата, ще сведем 3-$SAT \leq_P HAMPATH$. Понеже първият проблем е NP-пълен, то и вторият ще излезе такъв. Разглеждаме произволна булева формула ϕ в 3-CNF (конюнктивна нормална форма, като всяка дизюнктивна клауза има 3 литерала). Нека ϕ има n променливи и m клаузи. Строим граф G с n двузсвързани вериги от по $4m$ върха всяка, $n + 1$ помощни върха, съединяващи веригите m допълнителни върха, съответстващи на клаузите, както е на фигурата.

Идеята е всеки такъв ромбовиден блок да отговаря за една променлива от формулата и когато се мине през един такъв от ляво надясно, това да означава даване стойност 0 на променливата, а при минаване от дясно наляво по него да задава стойност 1 (конструкцията допуска точно едното). Дефинираме ребрата между върховете, отговарящи на клаузи $c_1, \cdots c_m$ и останалите върхове по следния начин. Ако клауза t съдържа променлива x_i, то построяваме ребрата (c_t, v_l) и (v_{l+1}, c_t), а ако съдържа отрицанието $\overline{x_i}$, то построяваме ребрата (v_l, c_t) и (c_t, v_{l+1}), където v_l и v_{l+1} са произволни два съседни върха от i-тата верига. Спазваме обаче следното правило - ако v_k вече е свързан с някой c_p, повече не го свързваме с други, а също и между всяка двойка такива съседи искаме да има поне един свободен връх разстояние (както са на фигура *(б)* върховете в по-тъмен цвят около свързаните). Такова свързване е възможно, защото всяка верига е с дължина $3m$. Накратко ще обясним защо графът по конструкция има Хамилтонов цикъл точно когато ϕ е удовлетворима.



(а) Графът G (липсват ребрата между клаузите и променливите)



(б) Ребра между клаузите и променливите (на фигурата между клауза t и две от променливите ѝ)

($\Rightarrow$) Нека графът има Хамилтонов цикъл. Имаме единствен връх, в който не влизат ребра (най-горният) и единствен, от който не излизат (най-отдолу), тези върхове ще са съответно начален и краен за пътя. Всяка верига е била обиколена, при това в точно една от двете посоки (в противен случай, при опит да се върнем назад, ще влезем във вече посетен връх). Както споменахме, остойностяваме x_i с 1 ако посоката на обхождане на верига i е от дясно наляво и 0 иначе. Всеки връх c_t, отговарящ на клауза, е посетен, като нека сме дошли от верига номер i, б.о.о x_i (без черта) участва в клауза t. По дефиницията за някое $l : (c_t, v_l)$ и $(v_{l+1}, c_t) \in E$. Тоест в c_t сме дошли от v_{l+1}. Ако допуснем, че движението по веригата е било ляво-дясно (т.е. $x_i = 0$), ще достигнем противоречие, защото връщане и дообхождане на блока по-късно е невъзможно (понеже v_l вече е посетен), остава $x_i = 1$, тоест дизюнктивната клауза t е 1 ($\forall t$). Оттук ϕ има решение.

($\Leftarrow$) Обратно, нека има оценка на $x_1, \cdots x_n$, за която $\phi = 1$. Дефинираме следния път:

- Започни от началния, най-горен връх
- Ако си в блок i и $x_i = 1$, обиколи блока от дясно наляво, в противен случай обратно (обиколката на блок преминава през всичките му върхове и завършва в началото на следващия блок)
- При обиколка на верига i, ако от връх v_l има ребро към някой необходен c_t и съответното му ребро в обратната посока (от c_t към блока) не е към вече обходен съсед на v_l, мини през c_l и се върни обратно.

Ясно, че тази процедура обхожда всички върхове от блоковете, остава въпросът дали обхожда всички c_t. Всъщност да, нека най-малкият индекс на литерал със стойност 1 в клауза с номер t е i. Заради дефиницията, когато минаваме през блок i, ще е в точно тази посока, която позволява отбиване към c_t, той е непосетен, свободно можем да минем през него и да се върнем тъкмо в следващия връх от веригата.

Имаме функция $f : \Sigma^* \to \Sigma^*$ такава, че $f(\phi) = \ulcorner G \urcorner$ и ϕ е удовлетворима $\Leftrightarrow f(\phi) \in HAMPATH$. Ясно е, че функцията е изчислима и работи за полиномиално време, т.е. имаме свеждането $3\text{-}SAT \leq_P HAMPATH$. Следва, че $HAMPATH$ е NP-пълен. ■

Други

Задача 10 (**Recursion Theorem**)**.** За всяка машина на Тюринг $\mathcal{T}$, която изчислява функция $t : \Sigma^* \times \Sigma^* \to \Sigma^*$, съществува друга машина $\mathcal{R}$, изчисляваща функция $r : \Sigma^* \to \Sigma^*$ така, че за всяка дума ω: $r(\omega) = t(\omega, \ulcorner \mathcal{R} \urcorner)$.

Задача 11 (**Fixed-point Theorem**)**.** За всяка изчислима функция $f : \Sigma^* \to \Sigma^*$ съществува машина на Тюринг $\mathcal{R}$ такава, че $\mathcal{L}(\mathcal{R}) = \mathcal{L}(\mathcal{M}_{f(\ulcorner \mathcal{R} \urcorner)})$.

Доказателство. Нека $\mathcal{R}$ работи по следния начин:

- $\mathcal{R}$ взима собствения си код $\ulcorner\mathcal{R}\urcorner$ (това е възможно благодарение на теоремата за рекурсията).
- $\mathcal{R}$ изчислява $f(\ulcorner\mathcal{R}\urcorner) = \omega$.
- по вход α $\mathcal{R}$ симулира $\mathcal{M}_\omega$ върху α

Така $\mathcal{L}(\mathcal{R}) = \mathcal{L}(\mathcal{M}_\omega)$ и $\omega = f(\ulcorner\mathcal{R}\urcorner)$, както и искахме. ■

Литература

[1] Sanjeev Arora and Boaz Barak (2007) *Computational Complexity: A Modern Approach*, *https://theory.cs.princeton.edu/complexity/book.pdf*

[2] John E. Hopcroft, Jeffrey D. Ullman (1977) *Introduction to Automata Theory, Languages, and Computation*

[3] Steven Homer, Alan L. Selman (2011) *Computability and Complexity Theory*

[4] Michael Sipser (2012) *Introduction to the Theory of Computation*