Próg

Będę musiał coś wymyślić. Warto odsyłać rozwiązania częściowe, zwłaszcza jeśli zadanie ma podpunkty. Wysoce pożądane jest, by odesłane rozwiązania zostały złożone na komputerze (np przez LaTeX-a). Nie jest to jednak wymagane.

Spis treści/lorem ipsum

Na samym końcu znajduje się dodatek na temat notacji dużego-O. Jeśli nie spotkałaś/nie spotkałeś się z nią wcześniej, lub w trakcie czytania stwierdzenie że coś *działa w czasie wielomianowym* wzbudzi w Tobie niepokój spowodowany niezrozumieniem jego znaczenia, to polecam tam zajrzeć.

Jest wysoce prawdopodobne, że w którymś miejscu tego skryptu znajduje się błąd. Jeśli uważasz, że taki błąd znalazłeś/znalazłaś, to możesz wysłać mi mejla[1]

Problemy decyzyjne

Na zajęciach będziemy zajmować się problemami decyzyjnymi, to znaczy takimi, na które odpowiedzią jest *TAK* lub *NIE*
Przykłady:

- Czy liczba n jest pierwsza?
- Czy w grafie istnieje cykl Hamiltona?
- Czy najkrótsza ścieżka, pomiędzy wierzchołkami v i u w nieważonym grafie G, ma długość k?

Czytelnik ma prawo poczuć się w tym momencie zażenowany przyjętymi uproszczeniami. Znane są w końcu problemy rozwiązywalne za pomocą komputera, na które odpowiedzią jest coś więcej, niż tylko *TAK*, lub *NIE* (np *Jaką długość ma najkrótsza ścieżka pomiędzy wierzchołkami v i u*).

Ograniczenie się tylko do problemów decyzyjnych nie jest aż tak *dużym kłamstwem*. Podobnie jak w przykładzie 3, problemy obliczeniowe (w tym przypakdu *Jaką długość ma najkrótsza ścieżka?*) dają się opisywać przez problemy decyzyjne. Jeśli n jest rozmiarem grafu, to rozwiązanie zadanego problemu obliczeniowego sprowadza się do rozwiązania n instancji odpowiedniego problemu decyzyjnego, dla każdej z liczb od 1 do n [2].

---

[1] igor.hanczaruk@gmail.com

[2] Można też zachować się odrobinę mądrzej i pytać o to, czy ścieżka ma długość przynajmniej k. Wtedy wystarczy $\log n$ instancji

Zadanie 1. *Za pomocą problemu decyzyjengo opisz problem obliczeniowy znajdowania rozkładu liczby n na iloczyn liczb pierwszych (tzn. zdefiniuj jakiś problem decyzyjny, a następnie opisz jak za jego pomocą można rozwiązać problem rozkładu). Liczba wywołań zdefiniowanego przez Ciebie problemu decyzyjnego, w rozwiązaniu problemu obliczeniowego, powinna być niewiększa niż $(\log n)^k$ dla pewnego $k \in \mathbb{N}$.*

Automaty skończone

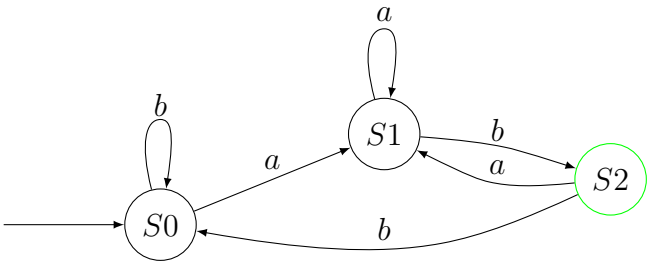
Teraz naszym celem jest wymyślenie matematycznego modelu dla maszyny, która:

- Wczytuje pewien ciąg znaków. [3] Ponieważ maszyny kończące pracę po skończonym czasie mogą wczytać tylko skończenie wiele znaków, to możemy ograniczyć się do skończonych wejść
- Ściśle wykonuje zadane jej instrukcje, po czym mówi *TAK*, lub *NIE*

Rozważanym do końca tej listy zadań modelem będzie deterministyczny automat skończony. Jest to maszyna składająca się z

- *Taśmy*, z której wczytuje znaki.
- *Alfabetu*, czyli zbioru znaków które potrafi wczytać
- Grafu skierowanego, którego krawędzie są etykietowane znakami.
- **Dokładnie jednego** wyróżnionego wierzchołka, nazywanego *startowym*
- **Pewnej ilości** (być może większej niż jeden, byc może równej zero) wyróżnionych wierzchołków, nazywanych *akceptującymi*

Maszyna ta wczytuje wejście z taśmy znak po znaku i po każdym przeczytanym znaku przechodzi odpowiednio zaetykietowaną krawędzią. Odpowiedzią jest *TAK* wtw po wczytaniu całego wejścia skończmy w wierzchołku akceptującym.



[3] Czytelnik mógłby się słusznie zastanowić dlaczego ograniczamy się do problemów w których wejściem jest ciąg znaków, skoro w rozważanych wyżej problemach wejściem mógł być np graf. Odpowiedź jest prosta: graf można opisać ciągiem znaków, tak samo jak ciągiem znaków można np opisać zadania kwalifikacyjne. Zresztą w prawdziwym komputerze wszystko jest słowem binarnym

Powyżej narysowany jest pewien automat dla słów nad dwuliterowym alfabetem $\{a, b\}$. Wierzchołkiem startowym jest S_0, ponadto jest jeden wierzchołek akceptujacy S_2. Przykładowo: dla wejścia *aba* najpierw przejdziemy do S_1, potem do S_2, a na końcu wrócimy do S_1. Nie jest to wierzchołek akceptujący, więc odpowiedzią będzie *NIE*. Z kolei dla wyjścia *bab* najpierw nigdzie nie pójdziemy (zostaniemy w S_0), potem do S_1, potem do S_2, więc odpowiedzią jest *TAK*.

Ogólnie o ile niczego nie zepsułem, to powższy automat rozwiązuje problem *Czy słowo kończy się na ab?*.

Formalnie mówimy, że deterministyczny automat skończony (DFA) to krotka $(\Sigma, Q, \delta, q_0, F)$, gdzie

- Σ jest alfabetem.
- Q jest zbiorem stanów (wierzchołków grafu)
- δ to funkcja $Q \times \Sigma \to Q$, nazywaną funkcją przejścia, która odpowiada za krawędzie naszego grafu
- $q_0 \in Q$ to stan początkowy
- $F \subseteq Q$ to zbiór stanów akceptujących

Czytelnik mógłby zauważyć, że w opisie tym pominęliśmy taśmę. Ale w końcu każdy automat ma taką samą.

Zwyczajowo, ścieżkę przebiegającą słowo w nazywa się *przebiegiem na słowie* w.

Zadanie 2. *Skonstruuj automat opisujący język nad alfabetem $\{a, b\}$, który ma parzystą liczbę liter a i parzystą liczbę liter b.*

Zadanie 3. *Skonstruuj automat rozpoznający język tych słów nad alfabetem $\Sigma = \{a, b\}$, w których żadne dwa a nie mają między sobą dokładnie jednej b.*

Zadanie 4. *Skonstruuj automat rozpoznający język tych słów nad alfabetem $\Sigma = \{0, 1\}$, które są zapisem binarnym liczby podzielnej przez 3.*

Zadanie 5. *Ile co najmniej wierzchołków musi mieć automat rozpoznający język tych słów nad alfabetem $\Sigma = \{1\}$, których długość jest podzielna przez n?*

Ograniczenia automatów skończonych

Okazuje się, że nie wszystkie problemy decyzyjne można rozwiązać za pomocą automatów skończonych. Czytelnik mógłby się na moment zatrzymać i zastanowić nad tym jak powinien wyglądać automat rozpoznający słowa, które mają tyle samo liter a, co b.

Problem powinien być intuicyjnie zrozumiały - automatowi brakuje pamięci na rozstrzyganie ile liter już przeczytał. W istocie, każdy automat skończony można zasymulować programem komputerowym, który zużywa stałą ilość pamięci tzn zużycie pamięci dla dowolnie dużych danych można ograniczyć przez stałą.

Poniższe zadanie, jak i dalsza część tego rozdziału, składają się z rozważań (momentami nieścisłych) na temat tego jak przyjęty model obliczeń ma się do prawdziwych programów komputerowych. Jeśli nie interesuje Cię informatyka, możesz to pominąć. Na zajęciach będzie mniej o komputerach.

Zadanie 6. *Udowodnij powyższą obserwację. Tzn: dla danego automatu, napisz jakiś opis programu (może być to jakiś istniejący lub nieistniejący język programowania, lista kroków, schemat blokowy, cokolwiek), który na wejściu przyjmuje słowo, a na wyjściu wypisuje TAK, lub NIE, w zależności od tego, czy słowo to jest akceptowane przez automat. Załóż, że słowo wejściowe jest wczytywane z innego nośnika i samo z siebie (no, dopóki się go nie wczyta), nie zużywa żadnej pamięci.* [4]*. Konkretny automat jest częścią specyfikacji zadania, możesz więc reprezentować go w komputerze w wygodny dla Ciebie sposób. Przykładowo, w C++ można zrobić*

```
const int ALPHABET_SIZE = 3;
const int STATES = 1000;
int to[STATES][ALPHABET_SIZE];
```

Użyte stałe są zupełnie przypadkowe. Tablica to przechowuje informacje o krawędziach (to[v][c] - do jakiego stanu przejść, jeśli będąc w stanie v wczytaliśmy znak c.

Zachodzi również odwrotność powyższej zależności. Jeśli jakiś problem można rozwiązać programem komputerowym w skończonej pamięci, to da się też automatem skończonym. Żeby to udowodnić, należy wziąć dowolny taki program i skonstruować automat skończony, który będzie zwracał takie same wyniki.

W tym momencie wygodnie będzie pozbyć się abstrakcji i popatrzeć na pamięć komputera jak na tablicę zer i jedynek (tym w istocie pamięć komputera jest). Ponieważ zużycie pamięci jest stałe, to istnieje taki indeks n w tej tablicy, że dla dowolnych danych nasz program zmienia tylko wartości na indeksach od 1 do n.[5] Ale to znaczy, że wszystkich możliwych ustawień tej tablicy jest 2^n - też skończenie wiele. Ponadto, to jak zachowuje się program zależy wyłącznie od tego co ma wpisane w pamięci. Można więc skonstruować automat o 2^n stanach, z których to każdy reprezentuje jakieś ustawienie pamięci. Przejścia pomiędzy nimi będą odpowiadać przejściom pomiędzy ustawieniami pamięci. Stany akceptujące to takie ustawienia pamięci, w których program wypisuje *TAK*.

---

[4] Ścisła definicja zużycia pamięci pojawi się na zajęciach, po wprowadzeniu maszyny turinga

[5] Czytelnik może nie czuć się przekonany. Słusznie, w końcu nie zdefiniowaliśmy ściśle zużycia pamięci

Lemat o pompowaniu

Pozostaje pytanie w jaki sposób przekonać się, że rzeczywiście w stałej pamięci nie da się stwierdzić, czy w słowie występuje tyle samo liter a, co b. Istnieje bardzo poręczny lemat *o pompowaniu*, o którym można poczytać na wikipedii: link (koniecznie z dowodem). Krótki glosariusz:

- Język regularny to taki, który da się opisać automatem skończonym
- Σ^* to zbiór wszystkich słów nad alfabetem Σ
- xy to sklejenie dwóch słow, x^k to sklejenie k razy słowa x.

Na wikipedii jest też dowód nieregularności języka $\{a^n b^n\}$, czyli składającego się z takich słów, w których jest tyle samo liter a i b, oraz wszystkie litery a występują przed wszystkimi literami b.

Zadanie 7. *Udowodnij, że język palidromów nad alfabetem $\Sigma = \{a, b\}$ nie jest językiem regularnym.*

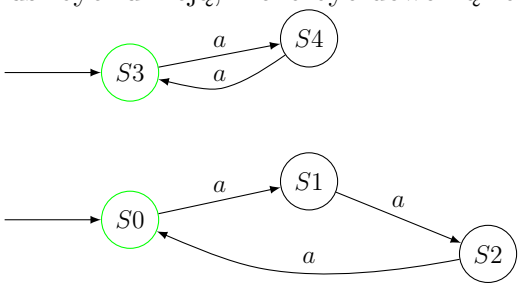
Zadanie 8. *Czy istnieje automat rozpoznający język tych słów nad alfabetem $\Sigma = \{a, b, c\}$, w których na parzystych pozycjach jest więcej liter a niż b?*

Niedeterminizm

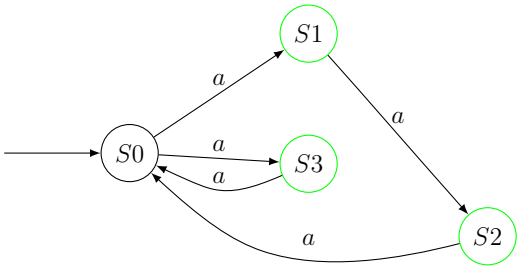
Niedeterministyczny automat skończony (NFA) różni się od DFA tym, że

- W DFA z wierzchołka musi wyjść dokładnie jedna krawędź na każdą literę alfabetu. W NFA może być tak, że z wierzchołka wychodzi wiele krawędzi z tą samą etykietą, lub że nie ma w ogóle krawędzi z jakąś etykietą
- Może być wiele stanów początkowych

Formalnie, zamiast jednego stanu $q_0 \in Q$, mamy zbiór stanów $Q_0 \subseteq Q$, oraz $\delta \subseteq Q \times \Sigma \times Q$ nie musi być funkcją, może być dowolną relacją.



Słowa o długości podzielnej prez 2 lub 3



Słowa o długości niepodzielnej przez 6

Powyżej narysowane są przykłady NFA. Tutaj mówimy, że automat akceptuje słowo, jeśli istnieje w nim przebieg na tym słowie, który zaczyna się w jakimś stanie początkowym

i kończy się w jakimś stanie akceptującym. Mogłoby się wydawać, że to duża różnica. DFA w naturalny sposób reprezentowało obliczenia komputerowe, w przypadku NFA na pierwszy rzut oka analogie mogą być niewidoczne. Okazuje się jednak, co jest treścią zadania z gwiazdką, że dla dowolnego NFA, można skonstruować DFA, które akcpetuje dokładnie te same słowa. Jest natomiast prawdą, że automaty niedeterministyczne mogą być mniejsze niż odpowiadające im automaty skończone, co pokazuje drugi przykład.

Zadanie 9. *(z gwiazdką) udowodnij, że NFA potrafią rozpoznawać tylko języki regularne*

W poniższym zadaniu można skorzystać z treści zadania z gwiazdką, nawet jeśli się go nie zrobiło.

Zadanie 10. *Niech L będzie językiem, a L^R będzie językiem odwróconych słów z L (np odwrócenie abc, to cba). Pokaż, że jeśli L jest regularny, to L^R też jest regularny.*

Reszta zadań

Zadanie 11. *Pokaż, że jeśli języki L_1 i L_2 (nad tym samym alfabetem) są regularne, to regularny jest też język $L_1 \cap L_2$ (słów które są zarówno w L_1, jak i w L_2).*

Zadanie 12. *Czy istnieją dwa języki, które nie są regularne, a których przecięcie jest regularne?*

Dodatek: Notacja duże O

Badając czas działania jakiegoś programu, bądź jego zużycie pamięci, na ogół nie będą interesować nas dokładnie wartości, a raczej tempo ich wzrostu. Przydatna jest tu notacja duże-O. Jeśli nie wiesz czym jest notacja duże-O, to poczytaj o niej w internecie. Możesz też przeczytać poniższą notatkę. W razie problemów, możesz napisać do mnie maila.

Załóżmy, że f i g to funkcje $\mathbb{N} \to \mathbb{R}^+$. Powiemy, że $f = O(g)$[6], gdy istnieją takie stałe n_0 i $c > 0$, że dla wszystkich $n > n_0$, $f(n) \leqslant cg(n)$. To znaczy, że dla dużych n, funkcja f rośnie co najwyżej tak samo szybko, jak g (z dokładnością do stałej). Odwrotną rzeczą jest duże-Ω. Powiemy, że $f = \Omega(g)$, gdy f rośnie co najmniej tak samo szybko, jak g. Jeśli obie zależności zachodzą, to mówimy, że f rośnie tak samo szybko jak g, co oznaczamy przez $f = \Theta(g)$.

Można zauważyć, że

$$a_k n^k + a_{k-1} n^{k-1} + \ldots + a_0 = \Theta(n^k)$$

Wiemy, że jeśli $i \geqslant j$, to (działając w liczbach naturalnych) $x^i \geqslant x^j$. To znaczy, że $a_k n^k + a_{k-1} n^{k-1} + \ldots + a_0 \leqslant \max\{a_i\} k n^k$, skąd wynika relacja duże-O. Relacja duże-Ω wynika stąd, że $a_k n^k + a_{k-1} n^{k-1} + \ldots + a_0 \geqslant a_k n^k$.

[6]Lepszym oznaczeniem byłoby $f \in O(g)$, ale tak już się przyjęło

Niech $T(n)$ będzie funkcją określającą czas działania jakiegoś algorytmu, w zależności od wielkości wprowadzonych danych. Powiemy, że algorytm ten działa w czasie wielomianowym, jeśli $T(n) = O(n^k)$, dla pewnego $k \in \mathbb{N}$.

Zachodzą również następujące zależności:

- $\lim_n \frac{f(n)}{g(n)} < \infty \implies f = O(g)$
- $\lim_n \frac{f(n)}{g(n)} > 0 \implies f = \Omega(g)$

Skąd wynika również

- $0 < \lim_n \frac{f(n)}{g(n)} < \infty \implies f = \Theta(g)$

Odwrotne implikacje jednak nie zachodzą, bo granice te wcale nie muszą istnieć (np ciąg $a_n = 2 + (-1)^n$ jest $\Theta(1)$, ale $\lim_n a_n$ nie istnieje). Ogólniej można zapisać

- $\exists_{c>0,n_0} \forall_{n>n_0} \frac{f(n)}{g(n)} < c \Leftrightarrow f = O(g)$
- $\exists_{c>0,n_0} \forall_{n>n_0} \frac{f(n)}{g(n)} > c \Leftrightarrow f = \Omega(g)$

Zadania z tej sekcji nie są bardzo ważne. W obu zadaniach przyjmuję $a, b \in \mathbb{R}$

Zadanie 13. *Wykazać, że dla dowolnych* $a, b > 1$, $\log_a(n) = \Theta(\log_b(n))$

Zadanie 14. *a) Wykazać, że jeśli* $a > b > 1$, *to* $b^n = O(a^n)$, *ale nie* $b^n = \Omega(a^n)$.
b) Wywnioskować stąd, że jeśli dla pewnej liczby $n_0 \in \mathbb{N}$, *wszystkich* $n > n_0$ *i pewnej stałej* c, $\frac{f(n)}{g(n)} < c < 1$, *to* $2^{f(n)} = O(2^{g(n)})$), *ale nie* $2^{f(n)} = \Omega(2^{g(n)})$).
c) Wywnioskować stąd w jakich relacjach są funkcje $(\log n)^n$, *oraz* $n^{\log n}$.[7]

[7] Wskazówka: $n \log\log n = O(n \log^2 n)$, ale nie $n \log\log n = \Omega(n \log^2 n)$. Wskazówki nie trzeba udowadniać.