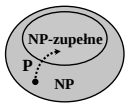
Ogólna charakterystyka problemów z klasy NP

- Algorytmy służące do ich rozwiązywania wymagają ciągłego sprawdzania dopuszczalności rozwiązań częściowych i stopniowego ich rozszerzania w celu znalezienia rozwiązania całego problemu; jeśli rozwiązanie częściowe nie da się dalej rozszerzyć w dopuszczalny sposób, to trzeba powrócić na jakiś wcześniejszy etap i po dokonaniu wymiany części składników rozwiązania rozpocząć od nowa próby rozszerzenia go.
- Postępowanie polegające na systematycznym przeglądzie wszystkich możliwych rozwiązań ma czasową złożoność ponadwielomianową.
- Jeśli rozwiązanie oznaczające rozstrzygnięcie problemu na „tak" (np. dla płaskiej układanki) jest podpowiedziane, to potwierdzenie, że taka odpowiedź jest poprawna ma złożoność wielomianową.

- Wiadomo, że dla każdego z problemów istnieje niedeterministyczny („magiczny") algorytm o czasowej złożoności wielomianowej;
  NP skrót od ang. *Nondeterministic Polynomial-time*;
  „magiczność" takiego algorytmu polega na założeniu, że w każdym kroku, w którym jest rozstrzygane, czy kolejny element dołożyć do rozwiązania, można skorzystać z „wyroczni" podpowiadającej, czy w dalszym postępowaniu ten wybór będzie prowadził do osiągnięcia poszukiwanego rozwiązania całego problemu, czy też nie.
- Biorąc pod uwagę brak, jak dotąd, deterministycznych algorytmów wielomianowych dla tych problemów, trzeba by stwierdzić, że są one w praktyce trudno rozwiązywalne, ale stałyby się łatwo rozwiązywalne, jeśli można by było korzystać przy budowie algorytmów z niedeterministycznej „wyroczni" (por. punkt poprzedni).

Ogólna charakterystyka problemów z klasy NP-zupełne

- Problemy NP-zupełne są szczególnymi problemami NP.
- Dla każdego problemu NP-zupełnego istnieją algorytmy o złożoności wielomianowej, którymi można przekształcić (zredukować) do niego każdy z problemów NP (jest to *definicja* „NP-zupełności problemu");



  NPC skrót od ang. *Nondeterministic Polynomial-time Complete*;
  redukcja wielomianowa polega na możliwości przekształcenia danych wejściowych zadanego problemu z klasy NP do postaci danych wejściowych któregokolwiek z problemów NP-zupełnych i po uzyskaniu dla niego rozwiązania wyznaczenia na jego podstawie rozwiązania dla pierwotnie rozważanego problemu NP. – złożoność obu tych przekształceń „dane w dane" i „rozwiązanie w rozwiązanie" musi być wielomianowa.

- Każdy problem z klasy NP-zupełne można zredukować wielomianowo (przekształcić w czasie wielomianowym) do każdego innego (*wniosek z definicji NP-zupełności*).
- Znalezienie algorytmu wielomianowego dla jakiegokolwiek z problemów NP-zupełnych (NPC) oznacza możliwość rozwiązywania w czasie wielomianowym, po pierwsze wszystkich pozostałych NP-zupełnych, a po drugie, wszystkich problemów NP (*dalszy wniosek*) .
- udowodnienie wykładniczego dolnego oszacowania dla jakiegokolwiek problemu NP-zupełnego oznacza, że po pierwsze dla żadnego problemu NP-zupełnego nie może istnieć algorytm wielomianowy, a po drugie, że także dla żadnego z problemów NP. (*jeszcze jeden wniosek*).
- albo wszystkie problemy NP-zupełne są łatwo rozwiązywalne, albo wszystkie trudno (*konkluzja*).

Udowodnienie, że nowozdefiniowany problem jest NP-zupełny przebiega w dwóch etapach:

1. trzeba udowodnić, że nowy problem należy do klasy NP (czyli, że można skonstruować dla niego niedeterministyczny algorytm wielomianowy),
2. trzeba skonstruować wielomianowy algorytmu, który redukuje do niego jakikolwiek ze znanych problemów NP-zupełnych.

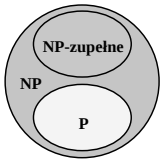
Np. wykazano, że problem komiwojażera jest NP-zupełny pokazując jak można redukować wielomianowo do niego problem wyznaczania drogi Hamiltona.

Pierwszą NP-zupełność udowodnił wprost Cook w 1971 r. dla problemu spełnialności zdania logicznego.

Klasa NP obejmuje problemy algorytmiczne, dla których istnieją niedeterministyczne algorytmy o złożoności wielomianowej.

Klasa P obejmuje problemy posiadające zwykłe (deterministyczne) algorytmy o złożoności wielomianowej, czyli problemy łatwo rozwiązywalne.

Klasa NP-zupełne obejmuje „wzorcowe" problemy z klasy NP „szybko" (wielomianowo) redukowalne jeden do drugiego.



Zawieranie się na rysunku klasy P w klasie NP wynika ze stwierdzenia, że algorytm deterministyczny jest szczególnym przypadkiem algorytmu niedeterministycznego, w którym nie trzeba korzystać z „wyroczni".

Czy P = NP ?

Algorytmy przybliżone

(czyli jak radzić sobie w praktyce z problemami NP i brakiem dla nich dokładnych algorytmów wielomianowych)

Na przykład dla problemu komiwojażera, który jest NP-zupełny (praktycznie trudno rozwiązywalny), istnieją algorytmy o złożoności wielomianowej wyznaczające „niezłe" co do długości cykle pozwalające odwiedzić każde z miast (te algorytmy nie gwarantują wyznaczenia najkrótszego cyklu – nie są w ścisłym znaczeniu rozwiązaniami tego problemu algorytmicznego – i dlatego nazywane są algorytmami przybliżonymi).

Dla algorytmów przybliżonych istotne znaczenia ma ocena różnicy pomiędzy wyznaczonym przez nie rozwiązaniem, a tym najlepszym, którego w czasie wielomianowym nie można było wyznaczyć.

W przypadku problemu komiwojażera ocena tego, jak dobry jest dany algorytm przybliżony może polegać na znalezieniu dla niego oszacowania ilorazu

$$s_A = \frac{L_{APR}}{L_{OPT}}$$

gdzie

L_{APR} oznacza długość cyklu, który można wyznaczyć rozważanym algorytmem przybliżonym, a

L_{OPT} oznacza długość najkrótszego cyklu, który byłby wyznaczony algorytmem dokładnym, gdyby ten potrafił zakończyć działanie w rozsądnym czasie.

Dla problemu komiwojażera istnieje algorytm przybliżony o złożoności $\mathbf{O(N^3)}$ wyznaczający w najgorszym przypadku cykl, dla którego $s_A \leq 1{,}5$

Przykład algorytmu przybliżonego dla problemu załadunku plecaka

Problem: znajdź wartości $x_1, x_2, ..., x_N$, dla których

$$\textbf{max} \sum_{i=1,...,N} c_i \cdot x_i \quad \text{i} \quad \sum_{i=1,...,N} a_i \cdot x_i \leq b$$

Dane: $c_1, c_2, ..., c_N$ i $a_1, a_2, ..., a_N$ oraz b

Algorytm:

1. Uporządkuj pakowane przedmioty według wartości $\frac{c_i}{a_i}$ posortowanych od największej do najmniejszej;
2. W wyznaczonym porządku sprawdzaj kolejno, czy przedmiot zmieści się jeszcze w plecaku – jeśli tak, to go zapakuj, a jeśli nie, to go pomiń i przejdź do następnego – aż do końca listy.

W przykładowym zadaniu: $c_1 = 31$, $c_2 = 19$, $c_3 = 27$, $c_4 = 7$

$a_1 = 4$, $a_2 = 2$, $a_3 = 3$, $a_4 = 1$ i $b = 6$

Etap 1:

$\frac{c_1}{a_1} = \frac{31}{4} = 7\frac{3}{4}$ $\quad \frac{c_2}{a_2} = \frac{19}{2} = 9\frac{1}{2}$ $\quad \frac{c_3}{a_3} = \frac{27}{3} = 9$ $\quad \frac{c_4}{a_4} = \frac{7}{1} = 7$

zatem $\frac{c_2}{a_2} \geq \frac{c_3}{a_3} \geq \frac{c_1}{a_1} \geq \frac{c_4}{a_4}$ i to jest kolejność pakowania

Etap 2:

$a_2 = 2 \leq 6 \quad \rightarrow \mathbf{x_2 = 1}$

$a_2 + a_3 = 5 \leq 6 \quad \rightarrow \mathbf{x_3 = 1}$

$a_2 + a_3 + a_1 = 9 > 6 \quad \rightarrow \mathbf{x_1 = 0}$

$a_2 + a_3 + a_4 = 6 \leq 6 \quad \rightarrow \mathbf{x_4 = 1}$

Wartość plecaka $c_2 + c_3 + c_4 = \mathbf{53}$

W tym przykładzie $s_A = 1$

W najgorszym przypadku algorytm daje upakowanie o $s_A \leq 2$

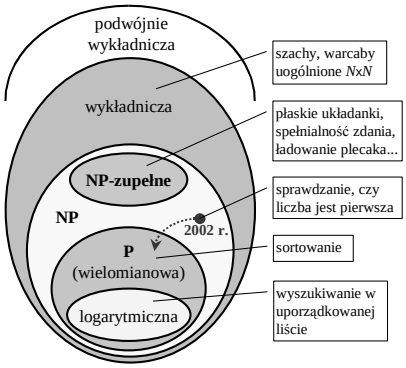
Złożoność algorytmu = złożoność alg. sortowania = $\mathbf{O(N \cdot \lg N)}$

Kilka dodatkowych uwag:

- Są problemy, które zmieniały przynależność do klas: sprawdzenie czy dana liczba jest liczbą pierwszą było przez długi czas przykładem problemu należącego do klasy NP, który nie był zupełny (tzn. nie należał do klasy NP-zupełne), ale w 2002 r. R. Agrawal przedstawił deterministyczny algorytm o złożoności $\mathbf{O(N^{12})}$, który jest całkowicie poprawnym rozwiązaniem dla tego problemu (N oznacza liczbę bitów potrzebną do zakodowania badanej liczby), czyli obecnie problem należy już do klasy P.
- Są problemy, których czasową złożoność ponadwielomianową można udowodnić przez podanie dolnych teoretycznych oszacowań, np. sprawdzenie czy dla danej konfiguracji w uogólnionych szachach N x N istnieje strategia wygrywająca dla wskazanego gracza.

- Są problemy, dla których udowodniono podwójnie wykładnicze dolne oszacowanie złożoności czasowej: $\Theta(2^{2^N})$
- Są problemy, dla których udowodniono ponadwielomianowe dolne oszacowanie złożoności pamięciowej.
- Są ciekawe przypadki problemów, które w praktyce rozwiązywane są algorytmami o pesymistycznej złożoności ponadwielomianowej, bo wprawdzie opracowano dla nich algorytm wielomianowy, ale dla większości zadań praktycznych sprawuje się on wyraźnie gorzej, np. problem programowania liniowego rozwiązywany algorytmem *sympleksowym* (o złożoności wykładniczej), mimo, że istnieje dla niego algorytm *elipsoidalny* (o złożoności wielomianowej), który wskazuje na wielomianową złożoność samego problemu.

Klasy problemów o różnej złożoności czasowej:



Zawieranie się jednych klas w drugich na tym rysunku wynika z notacji O(·) dla górnych oszacowań ich złożoności

Problemy nierozstrzygalne

Problem „nieograniczonego domina”

Dane wejściowe:
skończony katalog rodzajów kafelków i dostępna nieskończona liczba kafelków każdego rodzaju.

Rezultat:
rozstrzygnięcie, czy kafelkami z podanego katalogu można pokryć dowolny nieskończony płaski obszar zachowując zgodność kolorów na styku kafelków?

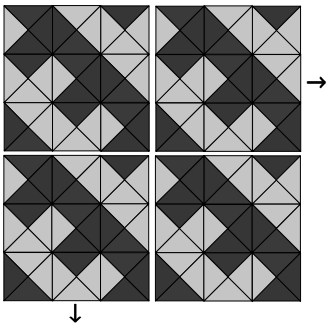
Szukamy algorytmu, który potrafiłby podawać takie rozstrzygnięcie

Np. dla katalogu:

Zakładamy, że kafelków nie można obracać

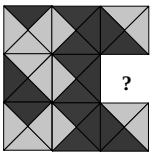
Dla katalogu:

Odpowiedź: ***TAK***



Dla katalogu:

Odpowiedź: ***NIE***



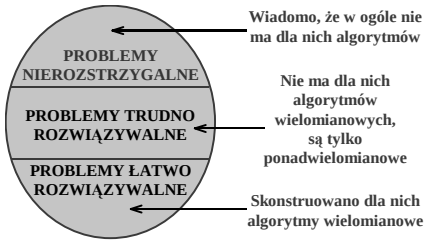
Twierdzenie

Dla każdego algorytmu (zapisanego w dającym się efektywnie wykonać języku programowania), który byłby przeznaczony do rozstrzygania problemu nieograniczonego domina, istnieje nieskończenie wiele dopuszczalnych zestawów danych wejściowych, dla których algorytm ten będzie działał w nieskończoność lub poda błędną odpowiedź, czyli nie będzie on całkowicie poprawnym rozwiązaniem tego problemu algorytmicznego.

Wniosek

Problem nieograniczonego domina jest problemem algorytmicznie **nierozstrzygalnym**

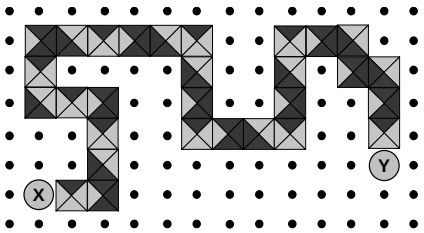
Trzy klasy problemów algorytmicznych:



- nieograniczona liczba przypadków do sprawdzenia nie jest dostatecznym warunkiem nierozstrzygalności problemu
- nierozstrzygalność wynika z wewnętrznej struktury problemu i jest często sprzeczna z intuicją

Problem „węża domino”

Czy dysponując skończonym katalogiem rodzajów kafelków można dane dwa punkty na nieskończonej siatce całkowitoliczbowej połączyć „wężem domino” z zachowaniem zgodności kolorów na styku kafelków?



Jeżeli sformułujemy problem „węża domino” zawartego w pewnym obszarze *R*, to:

- dla *R* ograniczonego (np. prostokąta) problem jest oczywiście rozstrzygalny, bo istnieje skończona liczba „węży”,
- dla *R* będącego całą płaszczyzną problem jest **rozstrzygalny**,
- dla *R* będącego półpłaszczyzną problem jest **nierozstrzygalny**.

Problem „zgodności słowników”

Dane wejściowe:
dwa zestawy słów (słowniki): $(X_1, X_2, ..., X_N)$ i $(Y_1, Y_2, ..., Y_N)$

Wszystkie słowa w obu zestawach składają się z liter tego samego alfabetu.

Czy możliwe jest ułożenie tego samego ciągu liter przez wybieranie słów w tej samej kolejności równolegle z obu zestawów?

Wybierane słowa mogą się powtarzać

Dla słowników:

	1	2	3	4	5
X	*abb*	*a*	*bab*	*baba*	*aba*
Y	*bbab*	*aa*	*ab*	*aa*	*a*

Odpowiedź: ***TAK***

Ułożenie słów w kolejności (2, 1, 1, 4, 1, 5) daje dla obu słowników ten sam ciąg ***aabbabbbabaabbaba***

Dla słowników:

	1	2	3	4	5
X	*bb*	*a*	*bab*	*baba*	*aba*
Y	*bab*	*aa*	*ab*	*aa*	*a*

Odpowiedź: ***NIE***

Problem „zgodności słowników” jest **nierozstrzygalny.**

Uwaga:
po zdjęciu ograniczenia, że z obu słowników słowa muszą być wybierane w tej samej kolejności problem staje się nie tylko rozstrzygalny, ale w dodatku łatwo rozwiązywalny!

Np.: w drugim zestawie słowników wybranie słów w kolejności (3, 2, 2) ze słownika *X* i w kolejności (1, 2) ze słownika *Y* daje ten sam ciąg ***babaa***

Problem równoważności składniowej dwóch języków programowania

Czy reguły składniowe jednego języka są równoważne regułom drugiego, tzn. definiują tę samą klasę instrukcji (lub programów)?

Problem równoważności składniowej jest **nierozstrzygalny**.

Problem stopu algorytmu

Dane wejściowe:
tekst programu będący zapisem poprawnego algorytmu w pewnym języku programowania
oraz dopuszczalny zbiór danych początkowych dla tego algorytmu.

Rezultat:
rozstrzygnięcie, czy ten program dla podanych danych zatrzyma się po skończonej liczbie operacji.

Szukamy algorytmu, który potrafiłby podawać takie rozstrzygnięcie

Poszukiwanie takiego algorytmu jest elementem ogólniejszego problemu algorytmicznej weryfikacji programów w celu stwierdzania ich całkowitej poprawności.

W zmiennej *X* jest wstępnie podana liczba naturalna.

Rozważmy dwa przykładowe algorytmy:

Algorytm 1

1. dopóki $X \neq 1$, wykonuj $X \leftarrow X - 2$.

- jeśli podana liczba jest nieparzysta i większa od 2, to algorytm 1. po wykonaniu skończonej liczby iteracji zatrzyma się,
- jeśli podana liczba jest parzysta lub równa 1, to nie zatrzyma się.

Algorytm 2

1. dopóki $X \neq 1$, wykonuj:
   1.1. jeśli *X* jest parzyste, to $X \leftarrow X / 2$,
   1.2. jeśli *X* jest nieparzyste, to $X \leftarrow 3 \cdot X + 1$.

Algorytm 2

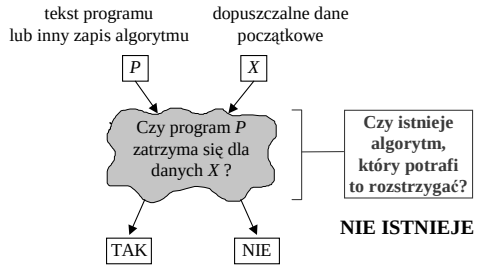
1. dopóki $X \neq 1$, wykonuj:
   1.1. jeśli *X* jest parzyste, to $X \leftarrow X / 2$,
   1.2. jeśli *X* jest nieparzyste, to $X \leftarrow 3 \cdot X + 1$.

- algorytm 2. sprawdzano wielokrotnie dla wielu liczb naturalnych i zawsze zatrzymywał się po wykonaniu skończonej liczby iteracji,
- do dzisiaj nie udało się udowodnić, że zatrzymuje się on dla dowolnej liczby naturalnej.

Np. dla początkowej wartości $X = 7$ ciąg wartości tej zmiennej w kolejnych iteracjach jest następujący:

7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, **1**

Problem stopu algorytmu



Problem stopu algorytmu jest **nierozstrzygalny**.

Problem „okresowego domina”

Czy kafelkami z podanego katalogu, który zawiera skończoną liczbę ich rodzajów, można pokryć dowolny płaski obszar zachowując zgodność kolorów na styku kafelków i wykorzystując nieskończoną liczbę razy kafelek wskazanego rodzaju?

Problem okresowego domina jest **wysoce nierozstrzygalny**.

Dla problemu wysoce nierozstrzygalnego nawet nie ma algorytmu dzielącego go na nieskończoną liczbę problemów nierozstrzygalnych.

W klasie problemów nierozstrzygalnych istnieje hierarchia ich trudności podobna do podziału na problemy NP-zupełne (raczej trudno rozwiązywalne) i problemy o złożoności ponadwielomianowej (na pewno trudno rozwiązywalne).

Odmiany problemu domina:

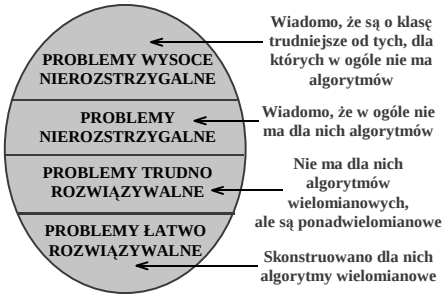
Dane wejściowe:
skończony katalog rodzajów kafelków i dostępna nieskończona liczba kafelków każdego rodzaju.

Rezultat:
rozstrzygnięcie, czy kafelkami z podanego katalogu można pokryć obszar ***T*** zachowując zgodność kolorów na styku kafelków?

- problem ograniczony ze stałą szerokością : *T* = prostokąt *C* x *N* (*C* jest ustaloną dla problemu liczbą naturalną, a *N* – dowolną liczbą podaną w danych wejściowych),
- problem ograniczony : *T* = kwadrat *N* x *N* ,
- problem nieograniczony : *T* = nieskończony płaski obszar,
- problem nieograniczony okresowy : *T* = nieskończony płaski obszar i kafelek wskazanego rodzaju powtarza się nieskończenie wiele razy.

Odmiana problemu domina	*Klasa trudności problemu*	*Jakie mamy w praktyce algorytmy?*
ograniczony ze stałą szerokością	łatwo rozwiązywalny	wielomianowe
ograniczony	trudno rozwiązywalny (NP-zupełny)	ponadwielomianowe
nieograniczony	nierozstrzygalny	żadnych nie ma i nie będzie
nieograniczony okresowy	wysoce nierozstrzygalny	nie ma algorytmów dzielących problem na nieskończenie wiele problemów nierozstrzygalnych

Cztery klasy problemów algorytmicznych:



Rozstrzygalność (obliczalność) problemów algorytmicznych