ZŁOŻONOŚĆ PROBLEMÓW ALGORYTMICZNYCH

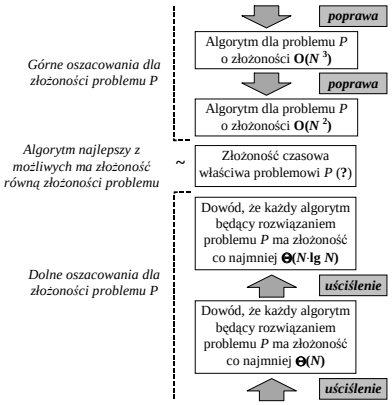
Dolne i górne oszacowania złożoności problemu

- Złożoność każdego poprawnego algorytmu znajdującego rozwiązanie danego problemu ustanawia **górne oszacowanie** złożoności dla tego problemu.

Czy można skonstruować algorytm o niższej złożoności?

Jeśli się uda, to górne oszacowanie złożoności problemu zostaje poprawione.

- **Dolne oszacowanie** złożoności problemu (otrzymane w wyniku analizy samego problemu) określa zakres dalszej poprawy rzędu złożoności algorytmów rozwiązujących ten problem.



Problemy zamknięte i luki algorytmiczne

Jeśli dysponujemy dla danego problemu algorytmem o złożoności O($g(N)$), to możemy powiedzieć, że złożoność problemu wynosi O($g(N)$), bo na pewno nie jest wyższego rzędu niż $g(N)$ – algorytm daje górne oszacowanie, a notacja O(·) ma dokładnie taki sam sens oszacowania od góry.

Jeśli górne i dolne oszacowania złożoności problemu algorytmicznego spotkają się w klasie złożoności tego samego rzędu, to możemy stwierdzić, że złożoność problemu wynosi dokładnie Θ($g(N)$) i taki problem nazywamy **zamkniętym** (z punktu widzenia określania jego złożoności).

Jeśli dla problemu algorytmicznego najlepsze znane górne i dolne oszacowania różnią się rzędem złożoności, to taką sytuację nazywamy **luką algorytmiczną**.

problem	*dolne oszacowanie*	*górne oszacowanie*
Wyszukiwanie z listy nieuporządkowanej	**Θ(N)**	**O(N)**
Wyszukiwanie z listy uporządkowanej	**Θ(lg$N$)**	**O(lg$N$)**
Sortowanie (bez ograniczania wartości)	**Θ(N·lg$N$)**	**O(N·lg$N$)**
Wyznaczanie „najkrótszej sieci kolejowej”	**Θ(N)**	**O($s(N)$·N)** [1)]

[1)] $s(N)$ - bardzo wolno rosnąca funkcja, np. dla N = 64 000 ma wartość 4

Wyszukiwanie z listy, sortowanie – problemy zamknięte

Wyznaczanie „najkrótszej sieci kolejowej” – luka algorytmiczna

Przykład – problem wież Hanoi

Problem jest zamknięty
(dolne ograniczenie złożoności = złożoność algorytmu rekurencyjnego lub iteracyjnego) i ma złożoność **Θ(2^N)**.

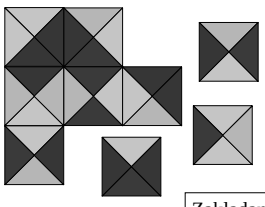
Podobno mnisi tybetańscy rozwiązują ten problem w pewnym klasztorze dla N = 64 i kiedy skończą, to także nasz świat się skończy!

1 ruch na sekundę ⇒ czas wykonania ok. **586 146 828 647 lat**

1 mln ruchów na sekundę ⇒ czas wykonania ok. **586 147 lat**

2^{64} = 18 446 744 073 709 551 616

Przykład – problem ułożenia płaskiej układanki



Algorytm ma rozstrzygać, czy istnieje takie ułożenie kwadratu M x M z $N = M^2$ kafelków, które zachowuje zgodność kolorów na przyległych bokach?

Zakładamy, że kafelków nie można obracać

Jest to przykład tzw. **problemu decyzyjnego** – problemu algorytmicznego, który polega na znalezieniu prawidłowej odpowiedzi „tak” lub „nie” na postawione pytanie (często jest to pytanie o istnienie rozwiązania problemu)

W problemie ułożenia płaskiej układanki występuje luka algorytmiczna, a górne oszacowanie jego złożoności jest rzędu **O(*N*!)**,

bo nie wymyślono lepszego algorytmu, jak tylko przeglądanie wszystkich możliwych ułożeń, których jest właśnie *N*!

Dla układanki 5 x 5 oznacza to:

1 mln układów na sekundę ⇒ czas sprawdzenia wynosi ok. **492 869 990 446 lat**

25! = 15 511 210 043 330 985 984 000 000

Wartości wybranych funkcji złożoności

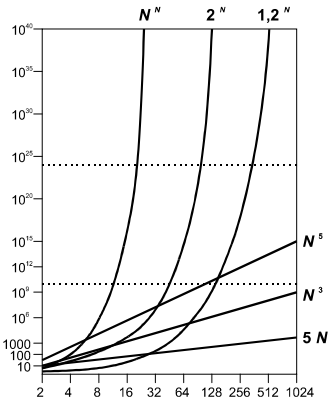
Funkcja	*N*				
	10	**50**	**100**	**300**	**1000**
lg *N*	**3**	**5**	**6**	**8**	**9**
N	**10**	**50**	**100**	**300**	**1000**
N* * lg *N	**33**	**282**	**664**	**2468**	**9965**
N^2	**100**	**2500**	**10 000**	**90 000**	**1 000 000**
N^3	**1000**	**125 000**	**1 000 000**	**27 mln (8 cyfr)**	**1 mld (10 cyfr)**
2^N	**1024**	**Liczba 16 cyfrowa**	**Liczba 31 cyfrowa**	**Liczba 91 cyfrowa**	**Liczba 302 cyfrowa**
***N*!**	**3,6 mln (7 cyfr)**	**Liczba 65 cyfrowa**	**Liczba 161 cyfrowa**	**Liczba 623 cyfrowa**	**∞**
N^N	**10 mld (11 cyfr)**	**Liczba 85 cyfrowa**	**Liczba 201 cyfrowa**	**Liczba 744 cyfrowa**	**∞**

Dla porównania:
liczba protonów w widocznym wszechświecie ma 126 cyfr,
liczba mikrosekund od „wielkiego wybuchu" ma 24 cyfry.

Zapotrzebowanie na czas
(jeśli wykonanie jednej operacji trwa mikrosekundę)

Funkcja	*N*				
	10	**20**	**50**	**100**	**300**
N^2	**1/10 000 sek.**	**1/2500 sek.**	**1/400 sek.**	**1/100 sek.**	**9/100 sek.**
2^N	**1/1000 sek.**	**1 sek.**	**35,7 lat**	**400 bilionów stuleci**	**75 cyfrowa liczba stuleci**
N^N	**2,8 godziny**	**3,3 biliona lat**	**70 cyfrowa liczba stuleci**	**185 cyfrowa liczba stuleci**	**728 cyfrowa liczba stuleci**

Tempo wzrostu wybranych funkcji złożoności



Funkcja $f(N)$ jest (asymptotycznie) **ograniczona z góry** przez funkcję $g(N)$, jeśli $\exists\, N_0$ i $C\ \forall\ N \geq N_0 :\ f(N) \leq C \cdot g(N)$

Jeśli $f(N) \prec g(N)$, to $f(N)$ jest ograniczona z góry przez $g(N)$

Funkcje złożoności dzielimy generalnie na:

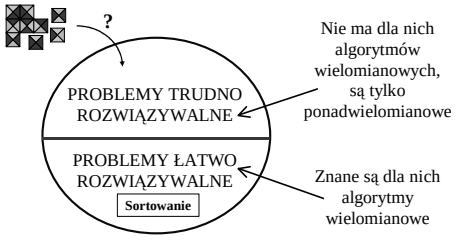
- **wielomianowe**, dla których istnieje takie $k < \infty$, że są one ograniczone z góry przez funkcję N^k,
- **ponadwielomianowe**, dla których takie k nie istnieje (np. 2^N).

$$1 \prec \lg\lg N \prec \lg N \prec \sqrt{N} \prec N \prec N \cdot \lg N \prec N^2 \prec N^3 \prec N^{\lg N} \prec 2^N \prec N! \prec N^N \prec 2^{2^N}$$

wielomianowe ⟷ ponadwielom.

Algorytm wielomianowy, to algorytm o złożoności $O(N^k)$ dla $k < \infty$

Dwie z klas problemów algorytmicznych:



Jak sobie poradzić z problemem płaskiej układanki?

1. Może po prostu poczekać na zbudowanie dostatecznie szybkiego komputera?
2. Może brak algorytmu wielomianowego dla tego problemu wynika z braku wiedzy i inwencji u informatyków?
3. Może udało by się wykazać, że dolne oszacowanie złożoności dla tego problemu jest wykładnicze i stwierdzić, że problem jest za trudny?
4. Może jest on tak szczególnym przypadkiem, że można go pominąć, bo wszystkie ważne problemy są łatwo rozwiązywalne?

Rozważmy 1. propozycję:

Funkcja złożoności	Maksymalny rozmiar zadania, które dla którego można znaleźć rozwiązanie w godzinę		
	Najszybszy współ-czesny komputer	Komputer 100 razy szybszy	Komputer 1000 razy szybszy
N^2	K	$10 * K$	$31{,}6 * K$
2^N	L	$L + 6$	$L + 10$

Rozważmy 4. propozycję:

Układanie płaskiej układanki należy do klasy problemów NPC (**NP-zupełnych**), która obejmuje ponad **1000** problemów algorytmicznych o jednakowych cechach:

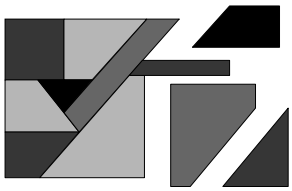
- dla wszystkich tych problemów istnieją ponadwielomianowe algorytmy
- dla żadnego jak dotąd nie znaleziono algorytmu wielomianowego, czyli górne oszacowanie złożoności jest ponadwielomianowe (wykładnicze)
- dla żadnego nie udowodniono, że nie może istnieć dla niego algorytm wielomianowy
- najlepsze wyznaczone dolne oszacowania złożoności są liniowe, tzn. $\Theta(N)$

Zatem dla problemu płaskiej układanki i pozostałych 1000 problemów NPC nie wiadomo nawet czy są to problemy trudno, czy łatwo rozwiązywalne!

W wielu dziedzinach zastosowań informatyki wciąż formułowane są nowe problemy, które okazują się problemami NPC.

Przykłady problemów algorytmicznych z klasy NPC (NP-zupełnych)

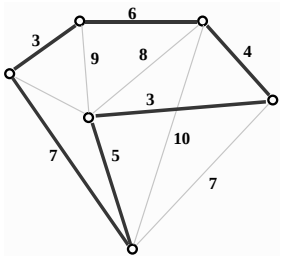
Inne płaskie układanki



Problem komiwojażera

Problem polega na znajdowaniu w sieci połączeń pomiędzy miastami najkrótszej drogi zamkniętej (cyklu), która pozwala odwiedzić każde z miast i powrócić do miasta wyjściowego.

Sieć z podanymi długościami połączeń



Minimalny cykl o długości 28

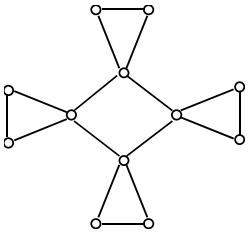
W wersji decyzyjnej problem komiwojażera polega na stwierdzaniu czy istnieje cykl o długości nie większej niż podana wartość L.

Algorytmy rozwiązywania problemu komiwojażera mają duże znaczenie np. przy:

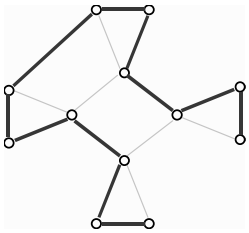
- projektowaniu sieci telekomunikacyjnych,
- projektowaniu układów scalonych,
- planowaniu linii montażowych,
- programowaniu robotów przemysłowych .

Problem wyznaczania drogi Hamiltona

Problem polega na sprawdzaniu, czy w grafie istnieje droga, która przez każdy wierzchołek przechodzi dokładnie raz.



W tym grafie nie istnieje,



ale po uzupełnieniu grafu jedną krawędzią można ją wyznaczyć

Problemy przydziału i układania harmonogramu

Na przykład:

- Należy przydzielić prace do wykonania pracownikom z uwzględnieniem różnych ograniczeń
- Należy wypełnić kontenery pojemnikami o różnych rozmiarach
- Należy ułożyć plan zajęć dopasowujący nauczycieli, klasy i godziny lekcyjne w taki sposób, aby dwie klasy nie miały jednocześnie zajęć z tym samym nauczycielem, nauczyciel nie prowadził w tym samym czasie lekcji w dwóch różnych klasach, dwaj nauczyciele nie prowadzili jednocześnie lekcji w tej samej klasie itd.

Problem spełnialności zdania logicznego

Problem polega na algorytmicznym rozstrzyganiu, czy istnieje zestaw takich prostych zdań logicznych o określonych wartościach prawda lub fałsz, które wstawione do podanego złożonego zdania logicznego, spowodują, że całe to zdanie stanie się prawdziwe.

Np. zdanie

$$\neg(E \Rightarrow F) \wedge (F \vee (D \Rightarrow E))$$

będzie prawdziwe po wstawieniu zdań o następujących wartościach

$E \leftarrow$ PRAWDA, $F \leftarrow$ FAŁSZ, $D \leftarrow$ FAŁSZ

i zatem **jest spełnialne**.

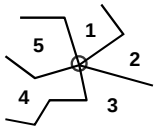
A zdanie

$$\neg((D \wedge E) \Rightarrow F) \wedge (F \vee (D \Rightarrow \neg E))$$

nie jest spełnialne.

Kolorowanie mapy płaskiej 3 kolorami lub *szukanie tzw. liczby chromatycznej grafu*

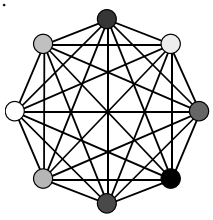
Problem polega na algorytmicznym rozstrzyganiu, czy podana mapa może być pokolorowana **3 kolorami** tak, aby sąsiednie obszary nie miały tego samego koloru.



- dla **2 kolorów** problem jest **łatwo rozwiązywalny** - wystarczy sprawdzić, czy mapa nie zawiera punktów, w których styka się nieparzysta liczba obszarów, np.
- dla **4 kolorów** problem jest banalny, bo udowodniono twierdzenie o czterech kolorach.

Problem polega na algorytmicznym znajdowaniu **najmniejszej liczby kolorów**, którymi można pokolorować wierzchołki podanego grafu tak, aby każde dwa wierzchołki bezpośrednio połączone krawędzią miały różne kolory.

Łatwo można skonstruować graf wymagający dowolnie dużej liczby kolorów:



Klika - zbiór wierzchołków w grafie połączonych krawędziami każdy z każdym

Problem załadunku plecaka

Problem polega na algorytmicznym wyznaczaniu takiego upakowania podanych przedmiotów do plecaka, aby łączna ich wartość była maksymalna, ale nie została przekroczona pojemności plecaka.

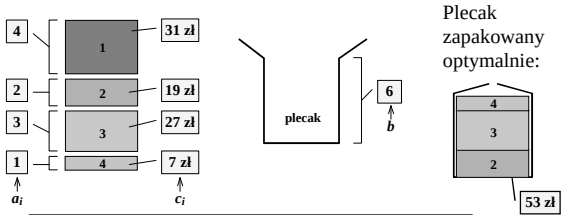
Dane wejściowe: dla ponumerowanych przedmiotów 1, 2, ..., N są podane ich wartości $\boldsymbol{c_1}, \boldsymbol{c_2}, ..., \boldsymbol{c_N}$ i objętości $\boldsymbol{a_1}, \boldsymbol{a_2}, ..., \boldsymbol{a_N}$ oraz podana jest objętość plecaka $\boldsymbol{b}$ i zachodzi

$$b < \sum_{i=1,...,N} a_i$$

Należy wyznaczyć wartość zmiennych decyzyjnych $\boldsymbol{x_1}, \boldsymbol{x_2}, ..., \boldsymbol{x_N}$, dla kórych $\boldsymbol{x_i = 1}$ oznacza „zapakowanie" przedmiotu o numerze $\boldsymbol{i}$, a $\boldsymbol{x_i = 0}$ pominięcie tego przedmiotu, tak aby były spełnione warunki:

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

W wersji decyzyjnej problem polega na rozstrzyganiu,
czy dla podanej wartości T istnieje takie upakowanie plecaka,
w którym: $\sum_{i=1,\ldots,N} c_i \cdot x_i \geq T$ i $\sum_{i=1,\ldots,N} a_i \cdot x_i \leq b$



Plecak zapakowany optymalnie:

53 zł

Rozwiązanie optymalne: $x_1 = \mathbf{0}$, $x_2 = \mathbf{1}$, $x_3 = \mathbf{1}$, $x_4 = \mathbf{1}$