

Czy zadania z Olimpiady Informatycznej są trudne?

OI a „prawdziwa nauka”

Jakub Radoszewski

finał XVII Olimpiady Informatycznej

13 kwietnia 2010

Zadania na OI

200 chomików

Zadania na OI

200 chomików
20 000 owiec

Zadania na OI

200 chomików

20 000 owiec

100 000 wagonów do przetoczenia

Zadania na OI

200 chomików

20 000 owiec

100 000 wagonów do przetoczenia

1 000 000 słońi

Zadania na OI

200 chomików

20 000 owiec

100 000 wagonów do przetoczenia

1 000 000 słońi

1 000 000 000 pięter

Zadania na OI

200 chomików

20 000 owiec

100 000 wagonów do przetoczenia

1 000 000 słońi

1 000 000 000 pięter

10^{500} kafelków...

$P=NP?$

P – klasa problemów, dla których istnieją algorytmy wielomianowe, np. najkrótsze ścieżki w grafie, wyszukiwanie wzorca w tekście, wypukła otoczka, cykl Eulera, skojarzenia w grafie dowolnym...

P=NP?

P – klasa problemów, dla których istnieją algorytmy wielomianowe, np. najkrótsze ścieżki w grafie, wyszukiwanie wzorca w tekście, wypukła otoczka, cykl Eulera, skojarzenia w grafie dowolnym...

NP-zupełne – klasa problemów, dla których istnieje rozwiązanie o złożoności czasowej $O(c^{poly(n)})$, co do których nie wiadomo, czy istnieje algorytm wielomianowy, ale jeżeli choć dla *jednego* z nich istnieje, to i dla wszystkich. Przykłady: cykl Hamiltona, najliczniejsza klika, problem plecakowy, pokrycie zbioru, wiele problemów praktycznych...

P=NP?

P – klasa problemów, dla których istnieją algorytmy wielomianowe, np. najkrótsze ścieżki w grafie, wyszukiwanie wzorca w tekście, wypukła otoczka, cykl Eulera, skojarzenia w grafie dowolnym...

NP-zupełne – klasa problemów, dla których istnieje rozwiązanie o złożoności czasowej $O(c^{poly(n)})$, co do których nie wiadomo, czy istnieje algorytm wielomianowy, ale jeżeli choć dla *jednego* z nich istnieje, to i dla wszystkich. Przykłady: cykl Hamiltona, najliczniejsza klika, problem plecakowy, pokrycie zbioru, wiele problemów praktycznych...

Są problemy, co do których nie wiadomo, czy są w P czy NP-zupełne, np. *faktoryzacja liczb*, albo takie, o których dowiedziano się bardzo niedawno, np. *testowanie pierwszości* jest w P (2002 r.).

Co robić z problemami NP-zupełnymi?

- **optymalizacja algorytmów dokładnych** – poprawiamy (wykładniczą) złożoność obliczeniową. Przykład: zadanie *Przeprawa* z XI OI, $O^*(4^n) \rightarrow O^*(3^n)$.

Co robić z problemami NP-zupełnymi?

- **optymalizacja algorytmów dokładnych** – poprawiamy (wykładniczą) złożoność obliczeniową. Przykład: zadanie *Przeprawa* z XI OI, $O^*(4^n) \rightarrow O^*(3^n)$.
- **algorytmy parametryzowane** – rozwiązania o złożoności czasowej $O(f(k) \cdot \text{poly}(n))$, gdzie $f(k)$ to praktycznie dowolna funkcja („bardzo wykładnicza”). Przykład: zadanie *Zosia* z XIII OI, rozwiązanie $O(n + m + k^2 2^k)$.

Co robić z problemami NP-zupełnymi?

- **optymalizacja algorytmów dokładnych** – poprawiamy (wykładniczą) złożoność obliczeniową. Przykład: zadanie *Przeprawa* z XI OI, $O^*(4^n) \rightarrow O^*(3^n)$.
- **algorytmy parametryzowane** – rozwiązania o złożoności czasowej $O(f(k) \cdot \text{poly}(n))$, gdzie $f(k)$ to praktycznie dowolna funkcja („bardzo wykładnicza”). Przykład: zadanie *Zosia* z XIII OI, rozwiązanie $O(n + m + k^2 2^k)$.
- **algorytmy aproksymacyjne** – algorytm o lepszej złożoności czasowej, który zawsze znajduje rozwiązanie co najwyżej p razy gorsze (p -aproksymacyjny). Przykład: zadanie *Koleje* z XIV OI, „Sumaryczny koszt utrzymania odcinków torów może być co najwyżej dwukrotnie większy od najmniejszego kosztu [...]”.

Co robić z problemami NP-zupełnymi?

- **specjalne klasy wejść** – problem zbioru niezależnego w ogólnym grafie jest NP-zupełny, dla drzew jest całkiem łatwy.

Co robić z problemami NP-zupełnymi?

- **specjalne klasy wejść** – problem zbioru niezależnego w ogólnym grafie jest NP-zupełny, dla drzew jest całkiem łatwy.
- **inne modele** – np. komputery kwantowe (algorytm faktoryzacji Shora, złożoność czasowa $O((\log n)^3)$), obliczenia na kartach graficznych CUDA.

Co robić z problemami NP-zupełnymi?

- **specjalne klasy wejść** – problem zbioru niezależnego w ogólnym grafie jest NP-zupełny, dla drzew jest całkiem łatwy.
- **inne modele** – np. komputery kwantowe (algorytm faktoryzacji Shora, złożoność czasowa $O((\log n)^3)$), obliczenia na kartach graficznych CUDA.
- **sztuczna inteligencja** – hill climbing, symulowane wyżarzanie, algorytmy genetyczne. . . Przykład: Marathon Match na TopCoderze, zadanie *Kwadraty* z CPSPC 2009.

Co robić z problemami NP-zupełnymi?

- **specjalne klasy wejść** – problem zbioru niezależnego w ogólnym grafie jest NP-zupełny, dla drzew jest całkiem łatwy.
- **inne modele** – np. komputery kwantowe (algorytm faktoryzacji Shora, złożoność czasowa $O((\log n)^3)$), obliczenia na kartach graficznych CUDA.
- **sztuczna inteligencja** – hill climbing, symulowane wyżarzanie, algorytmy genetyczne. . . Przykład: Marathon Match na TopCoderze, zadanie *Kwadraty* z CPSPC 2009.
- **wykorzystanie NP-zupełności** – algorytm RSA (no, prawie), dowody z wiedzą zerową (polecam!).

Wnioski?

Algorytmy wielomianowe poza OI to przeżytek?

OI nie ma nic wspólnego z pracą naukową
algorytmików?

Zadanie Chomiki z XVII OI

$\#occ(u, v)$ – liczba wystąpień słowa u w słowie v

Dane są słowa s_1, s_2, \dots, s_k , $k \leq 200$,
 $|s_1| + \dots + |s_k| \leq 100\,000$. Znaleźć najkrótsze u , że
 $\sum_{i=1}^k \#occ(s_i, u) \geq m$ ($m \leq 10^9$).

Standardowe założenie: zbiór $\{s_1, \dots, s_k\}$ jest *factor-free*
(żadne słowo nie jest pod słowem innego).

Problem SCS

Dane są słowa s_1, s_2, \dots, s_k , $|s_1| + \dots + |s_k| \leq n$. Znaleźć najkrótsze słowo u , w którym każde s_i występuje jako pod słowo.

Przykład:

$s_1 = \text{abaab}$, $s_2 = \text{baba}$, $s_3 = \text{aab}$, $s_4 = \text{bb}$.
 $s = \text{babaabb}$, gdyż:

Problem SCS

Dane są słowa s_1, s_2, \dots, s_k , $|s_1| + \dots + |s_k| \leq n$. Znaleźć najkrótsze słowo u , w którym każde s_i występuje jako pod słowo.

Przykład:

$s_1 = \text{abaab}$, $s_2 = \text{baba}$, $s_3 = \text{aab}$, $s_4 = \text{bb}$.

$s = \text{babaabb}$, gdyż:

b abaab b

Problem SCS

Dane są słowa s_1, s_2, \dots, s_k , $|s_1| + \dots + |s_k| \leq n$. Znaleźć najkrótsze słowo u , w którym każde s_i występuje jako pod słowo.

Przykład:

$s_1 = \text{abaab}$, $s_2 = \text{baba}$, $s_3 = \text{aab}$, $s_4 = \text{bb}$.

$s = \text{babaabb}$, gdyż:

b abaab b

baba abb

Problem SCS

Dane są słowa s_1, s_2, \dots, s_k , $|s_1| + \dots + |s_k| \leq n$. Znaleźć najkrótsze słowo u , w którym każde s_i występuje jako pod słowo.

Przykład:

$s_1 = \text{abaab}$, $s_2 = \text{baba}$, $s_3 = \text{aab}$, $s_4 = \text{bb}$.

$s = \text{babaabb}$, gdyż:

b abaab b

baba abb

bab aab b

Problem SCS

Dane są słowa s_1, s_2, \dots, s_k , $|s_1| + \dots + |s_k| \leq n$. Znaleźć najkrótsze słowo u , w którym każde s_i występuje jako pod słowo.

Przykład:

$s_1 = \text{abaab}$, $s_2 = \text{baba}$, $s_3 = \text{aab}$, $s_4 = \text{bb}$.

$s = \text{babaabb}$, gdyż:

b abaab b

baba abb

bab aab b

babaa bb

Znane wyniki dotyczące SCS

- Problem jest NP-zupełny.
- Algorytm dokładny: złożoność czasowa $O^*(2^k)$.
- Zachłanny algorytm aproksymacyjny:
 - 4-aproksymacja (Blum i inni, 1991)
 - 3,5-aproksymacja (Kaplan, Shafrir, 2005)
 - 2-aproksymacja? (hipoteza, Blum i inni).
- Algorytm 2,5-aproksymacyjny (Breslauer i inni, 1997 + Kaplan i inni, 2005).

Wielokrotne wystąpienia...

Problem SUM-SCS(k):

Dane: s_i oraz liczba m .

Znaleźć najkrótsze u , że $\sum_{i=1}^k \#occ(s_i, u) \geq m$.

Problem MULTI-SCS(k):

Dane: s_i oraz liczby m_i .

Znaleźć najkrótsze u , że $\#occ(s_i, u) \geq m_i$ dla $i = 1, 2, \dots, k$.

... wielomianowe algorytmy!

Maxime Crochemore, Marek Cygan, Costas Iliopoulos, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, Tomasz Waleń, *Algorithms for Three Versions of the Shortest Common Superstring Problem*, CPM 2010, Nowy Jork, USA.

Rozwiązujemy problemy:

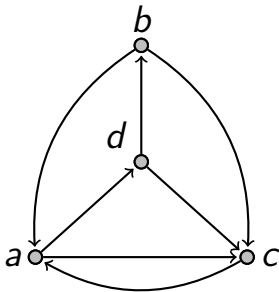
- **SUM-SCS(k)** i **MULTI-SCS(k)** dla słów długości 2: algorytmy $O(k)$
- **SUM-SCS(k)**: algorytm $O(n + k^3 \log m)$ – Chomiki
- **MULTI-SCS(k)**: algorytm wielomianowy dla $k = O(1)$

SUM-SCS(k), $|s_i| = 2$

Zbiór słów: $S = \{ad, ac, ba, bc, ca, db, dc\}$

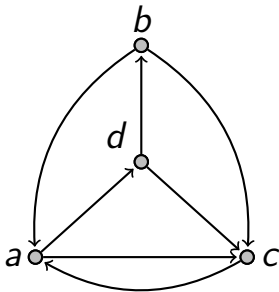
SUM-SCS(k), $|s_i| = 2$

Zbiór słów: $S = \{ad, ac, ba, bc, ca, db, dc\}$



SUM-SCS(k), $|s_i| = 2$

Zbiór słów: $S = \{ad, ac, ba, bc, ca, db, dc\}$



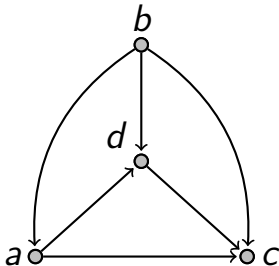
Bierzemy cykl, np. słowo $cadcadcadc \dots$

SUM-SCS(k), $|s_i| = 2$

Zbiór słów: $S = \{ad, ac, ba, bc, bd, dc\}$

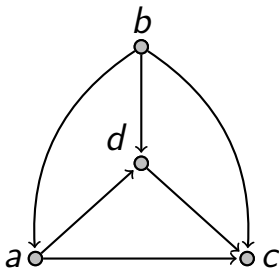
SUM-SCS(k), $|s_i| = 2$

Zbiór słów: $S = \{ad, ac, ba, bc, bd, dc\}$



SUM-SCS(k), $|s_i| = 2$

Zbiór słów: $S = \{ad, ac, ba, bc, bd, dc\}$



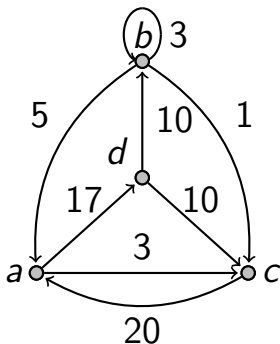
Brak cyklu – chodzimy w kółko po najdłuższej ścieżce:
badcbadcbadc...

MULTI-SCS(k), $|s_i| = 2$

Zbiór słów: $S = \{ad, ac, ba, bb, bc, ca, db, dc\}$ z krotnościami:
 $(m_i)_{i=1}^8 = (17, 3, 5, 3, 1, 20, 10, 10)$.

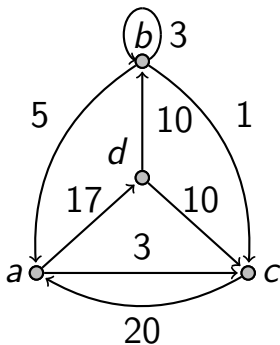
MULTI-SCS(k), $|s_i| = 2$

Zbiór słów: $S = \{ad, ac, ba, bb, bc, ca, db, dc\}$ z krotnościami:
 $(m_i)_{i=1}^8 = (17, 3, 5, 3, 1, 20, 10, 10)$.



MULTI-SCS(k), $|s_i| = 2$

Zbiór słów: $S = \{ad, ac, ba, bb, bc, ca, db, dc\}$ z krotnościami:
 $(m_i)_{i=1}^8 = (17, 3, 5, 3, 1, 20, 10, 10)$.



Chcemy dodać jak najmniej krawędzi, żeby graf miał ścieżkę Eulera, albo dla wygody, cykl Eulera.

MULTI-SCS(k), $|s_i| = 2$

Dany: multigraf nieeulerowski $G = (V, E)$.

C_1, C_2, \dots, C_q – słabo spójne składowe G , $C_i \subseteq V$.

Definiujemy *zapotrzebowania* składowych C_i jako:

$$D_i = \sum_{v \in C_i} \max(\text{indeg}(v) - \text{outdeg}(v), 0).$$

Pokażemy, że minimalny zbiór krawędzi E' , jakie musimy dołożyć, ma rozmiar:

$$|E'| = \sum_{i=1}^q \max(1, D_i).$$

MULTI-SCS(k), $|s_i| = 2$

Jak nam może pomóc dołożenie krawędzi (u, v) ?

- Jeżeli $u, v \in C_i$, to D_i maleje o 1.

Uwaga: jeżeli $D_i > 0$, to w składowej C_i znajduje się i jakiś wierzchołek nadmiarowy ($\text{indeg}(v) > \text{outdeg}(v)$), i jakiś deficytowy ($\text{indeg}(v) < \text{outdeg}(v)$).

MULTI-SCS(k), $|s_i| = 2$

Jak nam może pomóc dołożenie krawędzi (u, v) ?

- Jeżeli $u, v \in C_i$, to D_i maleje o 1.
- Jeżeli $u \in C_i$ oraz $v \in C_j$ ($i \neq j$) oraz $D_i, D_j > 0$, to C_i oraz C_j zostają sklejone w jedną składową o zapotrzebowaniu co najmniej $D_i + D_j - 1$.

Uwaga: jeżeli $D_i > 0$, to w składowej C_i znajduje się i jakiś wierzchołek nadmiarowy ($\text{indeg}(v) > \text{outdeg}(v)$), i jakiś deficytowy ($\text{indeg}(v) < \text{outdeg}(v)$).

MULTI-SCS(k), $|s_i| = 2$

Jak nam może pomóc dołożenie krawędzi (u, v) ?

- Jeżeli $u, v \in C_i$, to D_i maleje o 1.
- Jeżeli $u \in C_i$ oraz $v \in C_j$ ($i \neq j$) oraz $D_i, D_j > 0$, to C_i oraz C_j zostają sklejone w jedną składową o zapotrzebowaniu co najmniej $D_i + D_j - 1$.
- Jeżeli $u \in C_i$ oraz $v \in C_j$ ($i \neq j$) oraz $D_i = 0$ lub $D_j = 0$, to C_i oraz C_j zostają sklejone w jedną składową o zapotrzebowaniu co najmniej $D_i + D_j$.

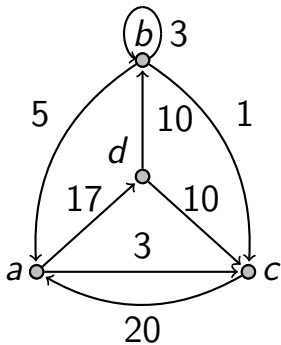
Uwaga: jeżeli $D_i > 0$, to w składowej C_i znajduje się i jakiś wierzchołek nadmiarowy ($\text{indeg}(v) > \text{outdeg}(v)$), i jakiś deficytowy ($\text{indeg}(v) < \text{outdeg}(v)$).

MULTI-SCS(k), $|s_i| = 2$

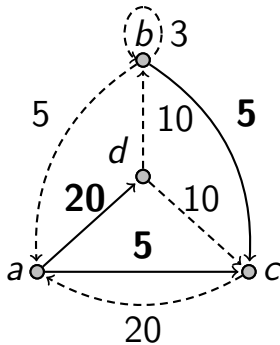
```
1:  $E' := \emptyset$ 
2: najpierw łączymy składowe:
3: while liczba składowych  $> 1$  do
4:   niech  $C_i, C_j$  będą dwiema różnymi składowymi
5:    $u :=$ wierzchołek nadmiarowy (lub jakikolwiek, jeśli  $D_i = 0$ )
6:    $v :=$ wierzchołek deficytowy (lub jakikolwiek, jeśli  $D_j = 0$ )
7:    $E' := E' \cup \{(u, v)\}$ 
8: mamy jedną składową  $C$  o zapotrzebowaniu  $D$ , zbijamy  $D$  do 0:
9: while  $D \neq 0$  do
10:   niech  $v^+$  – wierzchołek nadmiarowy,
         $v^-$  – wierzchołek deficytowy
11:    $E' := E' \cup \{(v^+, v^-) \cdot \min(d(v^+), d(v^-))\}$ 
12: return  $E'$ 
```

MULTI-SCS(k), $|s_i| = 2$

Przykładowy multigraf i jego minimalny multizbiór dodanych krawędzi: $E' = \{(a, d) \cdot 3, (a, c) \cdot 2, (b, c) \cdot 4\}$.



(a)



(b)

Zadanie *Ciągi bez zająknięć*, X OI:

Wyznaczyć n -elementowy ciąg bez zająknięć, złożony z najmniejszej liczby liter, $n \leq 10\,000\,000$. Zająknięcie: wystąpienie podstawa postaci vv , tzn. *kwadratu*.

Przykład: *abcacbcab*.

Zadanie *Ciągi bez zająknięć*, X OI:

Wyznaczyć n -elementowy ciąg bez zająknięć, złożony z najmniejszej liczby liter, $n \leq 10\,000\,000$. Zająknięcie: wystąpienie podstawa postaci vv , tzn. *kwadratu*.

Przykład: *abcacbcab*.

Jedna litera: ciąg długości 1.

Dwie litery: ciąg *aba*.

Trzy litery: dowolnej długości ciągi! Ale czy czytaliście uzasadnienie z niebieskiej książeczki?

Rozwiązanie

```
1: char s[107 + 1], int n;  
2: s[0] := 'a';  
3: for i := 1 to n - 1 do  
4:   s[i] := 'a';  
5:   while s[i] == s[i - 1] or s[i] == s[⌊i/2⌋] do  
6:     s[i] ++;  
7: return s;
```

Ale dlaczego to działa?

Rozwiązanie

```
1: char s[107 + 1], int n;  
2: s[0] := 'a';  
3: for i := 1 to n - 1 do  
4:   s[i] := 'a';  
5:   while s[i] == s[i - 1] or s[i] == s[ $\lfloor i/2 \rfloor$ ] do  
6:     s[i] ++;  
7: return s;
```

Ale dlaczego to działa?

Nie wiem 😊

Jak sprawdzać rozwiązania?

Jak sprawdzać rozwiązania?

Dany: napis s o długości n .

Obliczamy słownik podstów bazowych: $\lfloor \log n \rfloor$ tablic postaci $Name_t[]$ dla $0 \leq t \leq \log n$.

$Name_t[i]$ to identyfikator dla napisu $s[i..i + 2^t - 1]$.

Jak sprawdzać rozwiązania?

```
ALGORITHM Znajdź-Kwadrat(s, Name);  
  ( czy s zawiera jakiś kwadrat? )  
  
  for  $t = 0$  to  $\lfloor \log n \rfloor$  do  
    begin  
      RESET(Prev);  
      for  $j = 1$  to  $n$  do  
        if JestKwadrat(Prev[ Namet[ j ] ], j)  
          then return TRUE else  
            Prev[ Namet[ j ] ] := j;  
        end  
  return FALSE;
```

JestKwadrat(*i*, *j*) sprawdza, czy $s[i..j-1] = s[j..2j-i-1]$.

Jak sprawdzać rozwiązania?

```
ALGORITHM Znajdź-Kwadrat(s, Name);  
  ( czy s zawiera jakiś kwadrat? )  
for t = 0 to  $\lfloor \log n \rfloor$  do  
  begin  
    RESET(Prev);  
    for j = 1 to n do  
      if JestKwadrat(Prev[ Namet[ j ] ], j)  
      then return TRUE else  
        Prev[ Namet[ j ] ] := j;  
    end  
return FALSE;
```

Hint: jeśli $s[i..i + \ell - 1] = s[i + \ell..i + 2\ell - 1]$, a 2^t spełnia $2^t \leq \ell < 2^{t+1}$, to $Prev[Name_t[i + \ell]] = i$.

Inne przykłady

- Zadanie *Jaskinie* z XI OI to szczególny przypadek wyniku Pawła Parysa i Krzyszka Onaka, opublikowanego na prestiżowej konferencji FOCS 2006.

Inne przykłady

- Zadanie *Jaskinie* z XI OI to szczególny przypadek wyniku Pawła Parysa i Krzyśka Onaka, opublikowanego na prestiżowej konferencji FOCS 2006.
- *Przyspieszenie algorytmu z XVI OI* to tak naprawdę implementacja sprawdzania równoważności słów względem relacji $uu \approx u$ (Jakub Radoszewski, Wojciech Rytter, SOFSEM 2010).

Inne przykłady

- Zadanie *Jaskinie* z XI OI to szczególny przypadek wyniku Pawła Parysa i Krzyśka Onaka, opublikowanego na prestiżowej konferencji FOCS 2006.
- *Przyspieszenie algorytmu* z XVI OI to tak naprawdę implementacja sprawdzania równoważności słów względem relacji $uu \approx u$ (Jakub Radoszewski, Wojciech Rytter, SOFSEM 2010).
- trudniejsza wersja zadania *Słowa* z XVI OI (czy $Fib_{n_1} Fib_{n_2} \dots Fib_{n_k}$ jest podstwowem Fib_m , dla $k \leq 10^6$, $n_i \leq 10^9$?) została zgłoszona do publikacji przez Bartka Walczaka (czasopismo IPL).

Inne przykłady

- Zadanie *Jaskinie* z XI OI to szczególny przypadek wyniku Pawła Parysa i Krzyśka Onaka, opublikowanego na prestiżowej konferencji FOCS 2006.
- *Przyspieszenie algorytmu* z XVI OI to tak naprawdę implementacja sprawdzania równoważności słów względem relacji $uu \approx u$ (Jakub Radoszewski, Wojciech Rytter, SOFSEM 2010).
- trudniejsza wersja zadania *Słowa* z XVI OI (czy $Fib_{n_1} Fib_{n_2} \dots Fib_{n_k}$ jest podstwowem Fib_m , dla $k \leq 10^6$, $n_i \leq 10^9$?) została zgłoszona do publikacji przez Bartka Walczaka (czasopismo IPL).
- Pomysł na zadanie *Kod* z XVI OI został zapożyczony z pracy doktorskiej Marka Biskupa.

Wyjściowe pytania

Czy Olimpiada Informatyczna ma jakiś związek z „prawdziwą nauką”?

Czy zadania z Olimpiady Informatycznej są trudne?

Wyjściowe pytania

Czy Olimpiada Informatyczna ma jakiś związek z „prawdziwą nauką”?

Czy zadania z Olimpiady Informatycznej są trudne?

Odpowiedzcie sami!