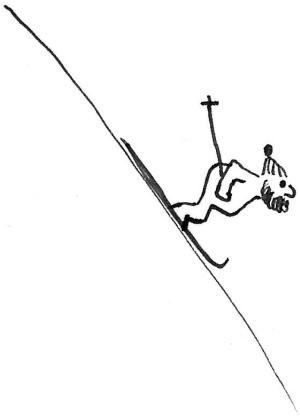
Gdy nie da się uniknąć ściągania rękawiczek

Piotr CHRZĄSTOWSKI-WACHTEL *



Poprzedniej zimy jeździłem na nartach w wyjątkowo sympatycznym dwudziestoosobowym towarzystwie. Wynajęliśmy wspólnie domek w Alpach, zrobiliśmy podstawowe zakupy w Polsce, resztę jedzenia kupowaliśmy na miejscu. Oczywiście, trzeba było rozwiązać problem bieżących zakupów. Ustaliliśmy, że każdy z nas co parę dni robi obiad dla pozostałych, dokupując w razie potrzeby konieczne produkty. W czasie całego pobytu zapisywaliśmy wydatki i pod koniec trzeba było zrobić rozliczenie i wyrównać rachunki. Po podliczeniu i obliczeniu średniej wydatków okazało się, że niektórzy zapłacili za dużo, inni za mało – inaczej mówiąc, niektórzy byli wierzycielami, inni dłużnikami. Wiadomo było, ile kto nadpłacił, a ile kto niedopłacił.

Pozostało stwierdzić, kto komu powinien przekazać ile pieniędzy. Żeby nie komplikować rozliczeń, polecono mi, jako informatykowi, opracowanie algorytmu, który doprowadziłby do wyrównania rachunków za pomocą minimalnej liczby transferów. Pojedynczy transfer powinien polegać na tym, że jedna osoba drugiej przekazuje jakąś sumę pieniędzy.

Z początku wydawało mi się, że może zadziałać następujący zachłanny algorytm. Największy dłużnik przekazuje tyle, ile powinien, największemu wierzycielowi. Tyle, ile powinien, czyli albo wszystko, co jest dłużny, albo całą nadpłatę wierzyciela, o ile jest ona mniejsza od długu. Po takiej transakcji ktoś przestaje być dłużnikiem lub ktoś przestaje być wierzycielem, wypada z puli i sprowadzamy problem do $n-1$ uczestników. Aktualizujemy wierzytelności i długi, określając znowu, kto ma największy dług i kto jest największym wierzycielem. Pozostały w grze uczestnik ostatniej transakcji może przestać już być największym w swojej klasie. Dla nowych danych, po uwzględnieniu efektu transakcji, powtarzamy opisany krok: wybieramy największego wierzyciela i największego dłużnika i każemy im dokonać możliwie największego sensownego transferu. I tak aż do wyczerpania dłużników i wierzycieli.

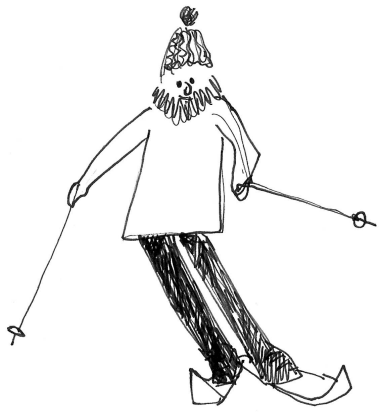
Okazało się szybko, że taka metoda nie jest optymalna. Spróbujmy wyobrazić sobie sytuację, w której jest 7 dłużników i 2 wierzycieli. Obaj wierzyciele spodziewają się dostać po 10 jednostek długu, a dłużnicy są winni kolejno 4, 4, 3, 3, 2, 2, 2. Teraz, jeśli zastosujemy nasz przepis, to 4 powędruje do 10, sprowadzając problem do szóstki $(4,3,3,2,2,2)$ po stronie dłużników i pary $(10,6)$ po stronie wierzycieli. Po kolejnym kroku dłużnikami będą $(3,3,2,2,2)$, a wierzycielami będą $(6,6)$. Dalej otrzymamy kolejno układy: $[(3,2,2,2),(6,3)]$, $[(2,2,2),(3,3)]$, $[(2,2),(3,1)]$, $[(2),(1,1)]$, $[(1),(1)]$, $[(),()]$. Razem 8 transferów. Tymczasem możliwe było dokonanie wyrównania za pomocą 7 transferów – wystarczyło, że jednemu wierzycielowi oddali całkowicie dług dłużnicy: 4, 4, 2, a drugiemu: 3, 3, 2, 2.

Wkrótce stało się dla mnie jasne, że nie ma prostego algorytmu rozwiązującego ten problem. Jest to bowiem problem NP-zupełny. W *Delcie* były już artykuły o problemach NP-zupełnych, ale przypomnijmy podstawowe sprawy. Po pierwsze, problemy NP-zupełne są problemami decyzyjnymi, czyli takimi, że wynikiem algorytmu jest odpowiedź TAK lub NIE. Zacznijmy nasze przypomnienie od określenia, które problemy są NP-trudne. Nieco upraszczając, powiedzmy, że problem jest NP-trudny, jeśli istnieje algorytm o złożoności wielomianowej, który na podstawie dostarczonego przykładu rozwiązania jest w stanie sprawdzić, czy przedstawione rozwiązanie faktycznie jest poprawne. Zbiór wszystkich problemów NP-trudnych tworzy klasę NP. Przykładem problemu NP-trudnego jest problem spełnialności formuły logicznej. Jeśli mamy daną formułę o n zmiennych, to pytanie o to, czy istnieje takie wartościowanie zmiennych, że ta formuła stanie się prawdziwa, może być rozwiązane pozytywnie przez podanie konkretnego wartościowania, które czyni ją prawdziwą. Sprawdzenie, czy takie konkretne wartościowanie wybrane spośród 2^n możliwych wartościowań czyni naszą formułę prawdziwą, wymaga jedynie obliczenia

Mówimy, że algorytm ma złożoność wielomianową, jeśli liczba kroków, którą wykonuje, wzrasta wielomianowo wraz ze wzrostem rozmiaru danych. Inaczej mówiąc, dla każdego n i dla każdych danych rozmiaru n wykona nie więcej niż $W(n)$ kroków dla pewnego wielomianu W zmiennej n. W szczególności, jeśli złożoność algorytmu wyraża się funkcją wykładniczą a^n dla $a>1$, to jego złożoność nie jest wielomianowa.

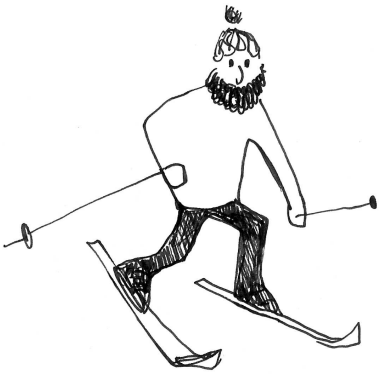
*Instytut Informatyki, Uniwersytet Warszawski

wartości tej formuły dla tego wartościowania, a to się robi elementarnie w czasie liniowym, czyli wielomianowym ze względu na długość formuły. Jeżeli zatem mielibyśmy cudowną wróżkę, która dla naszego problemu spełnialności dostarczałaby certyfikatów poprawności pozytywnej odpowiedzi (na przykład w postaci dostarczenia takiej formuły), to sprawdzenie, czy wróżka się nie pomyliła i upewnienie się, że dane wartościowanie jest prawidłowe, nie nastręczałoby problemów. Pytanie, skąd taką wróżkę wziąć? Niestety, takich wróżek nie ma i na razie nie pozostaje nam nic innego, niż przebadanie wszystkich 2^n (lub prawie wszystkich, ale też wykładniczo wielu) wartościowań. A nasza wróżka jest stworzeniem niedeterministycznym – stąd się bierze skrót: NP oznacza „Nondeterministic Polynomial", czyli wróżka losuje od razu dobry certyfikat, posługując się niedeterministycznym wyczuciem i w wielomianowym czasie generuje rozwiązanie – trochę przez przypadek, ale od czego są wróżki z ich niezawodną intuicją?



Przy okazji uwaga: nie wiadomo, czy problem tautologiczności formuły jest problemem w klasie NP, bo nie znamy sposobu na przedstawienie pozytywnego świadectwa tautologiczności, nieodwołującego się do *wszystkich* wartościowań. Za to problem nietautologiczności formuły jest w klasie NP, bo wystarczy znaleźć wartościowanie falsyfikujące formułę. Jeśli jakiś problem jest w klasie NP, to mówimy, że jego zaprzeczenie jest w klasie co-NP. Zatem problem tautologiczności formuły jest w klasie co-NP.

Teraz najciekawsze. Okazuje się, że w klasie NP istnieją problemy najtrudniejsze, czyli takie, że jeśli rozwiążemy którykolwiek z nich w czasie wielomianowym, to dowolny inny problem będziemy w stanie rozwiązać w czasie wielomianowym, sprowadzając go do któregoś z tych problemów najtrudniejszych. Te najtrudniejsze problemy w klasie NP tworzą klasę problemów NP-zupełnych. Należy do nich m.in. problem spełnialności formuły logicznej, ale i wiele, wiele innych. Nikt nie wie, czy istnieje choć jeden algorytm wielomianowy, który rozwiązałby przynajmniej jeden z problemów NP-zupełnych (gdyby rozwiązał jeden, to rozwiązałby wszystkie problemy z tej klasy wielomianowo). Za znalezienie takiego algorytmu lub za pokazanie, że takiego algorytmu nie ma, została wyznaczona nagroda miliona dolarów (`http://www.claymath.org/millennium/P_vs_NP/`) i to zagadnienie jest uznawane za największy otwarty problem informatyki teoretycznej.



Co to znaczy, że problem $\mathcal{A}$ jest trudniejszy niż problem $\mathcal{B}$? Przyjmijmy, że jest tak wtedy, kiedy mając konkretny przypadek problemu $\mathcal{B}$, możemy go rozwiązać, tłumacząc dane problemu $\mathcal{B}$ na dane problemu $\mathcal{A}$, a następnie rozwiązując problem $\mathcal{A}$ i uzyskując odpowiedź pozytywną wtedy i tylko wtedy, gdy odpowiedź na problem $\mathcal{B}$ jest pozytywna. Przyjmujemy tu, że takie tłumaczenie danych nie może być zbyt kosztowne (czyli powinno być też wielomianowe).

Żeby przybliżyć zagadnienie sprowadzalności, wyobraźmy sobie, że ktoś szuka dobrego algorytmu sprawdzającego, czy w nieuporządkowanej tablicy znajdują się choć dwa takie same elementy. Można to, oczywiście, zrobić w czasie kwadratowym, badając wszystkie możliwe pary. Ale można to zrobić szybciej, sortując dane. Wiadomo, że istnieją algorytmy sortujące n wartości wykonujące nie więcej niż $n \log n$ działań. Zatem sprowadźmy nasz problem do problemu sortowania: jeśli potrafimy sortować szybciej niż kwadratowo – a potrafimy, choćby stosując metodę kopcowania czy scalania – to problem istnienia powtarzających się wartości można rozwiązać następująco. Sortujemy dane, a potem sprawdzamy, czy jacyś dwaj sąsiedzi są sobie równi. Zatem problem istnienia dubletów nie jest bardziej złożony niż problem sortowania, czyli da się go rozwiązać w czasie proporcjonalnym do $n \log n$. Pokazanie, że nie da się go rozwiązać szybciej – a nie da się! – jest znacznie trudniejsze.

Pokażemy teraz, że nasze zadanie minimalizacji liczby transferów jest problemem o złożoności niemniejszej niż złożoność problemów NP-zupełnych. W naszym przypadku rozmiarem zadania jest liczba uczestników wyjazdu. Zagadnienie minimalizacji transferów można sprowadzić do problemu decyzyjnego w prosty sposób, modyfikując zadanie do takiego: *czy wystarczy k transferów, aby*

Rozwiązanie zadania M 1156.
Jeśli $547 \mid n$, to teza zadania jest oczywiście spełniona. Przyjmijmy więc, że liczba n nie jest podzielna przez 547. Ponieważ

$$n^{n^{n^4}} - n^{n^{n^2}} = n^{n^{n^2}}\left(n^{n^{n^4}-n^{n^2}} - 1\right),$$

więc wystarczy dowieść zależności

$$n^{n^{n^4}-n^{n^2}} \equiv 1 \mod 547.$$

Liczba 547 jest pierwsza, więc z małego twierdzenia Fermata wynika, że powyższa kongruencja będzie spełniona, jeśli wykażemy, że liczba

$$(*) \qquad n^{n^4} - n^{n^2} = n^{n^2}(n^{n^4-n^2} - 1)$$

jest podzielna przez $546 = 2 \cdot 3 \cdot 7 \cdot 13$.

Niech p będzie jedną z liczb $2, 3, 7$ lub 13. Jeśli $p \mid n$, to oczywiście wyrażenie $(*)$ jest podzielne przez p. Możemy więc przyjąć, że liczba n nie jest podzielna przez p. Korzystając ponownie z małego twierdzenia Fermata, stwierdzamy, że aby dowieść podzielności przez p wyrażenia $(*)$, wystarczy wykazać, że liczba $n^4 - n^2$ dzieli się przez $p - 1$.

Ale $p - 1$ jest dzielnikiem liczby 12, więc wystarczy dowieść, że liczba

$$n^4 - n^2 = n^2(n-1)(n+1)$$

jest podzielna przez 12. Istotnie: iloczyn $(n-1)n(n+1)$ trzech kolejnych liczb całkowitych jest podzielny przez 3, a ponadto wśród liczb n, n, $n-1$, $n+1$ są dokładnie dwie liczby parzyste.

wyrównać rachunki? Jeżeli dla każdego k rozwiążemy takie zadanie w czasie wielomianowym, to zapuszczając je kolejno dla $k = 1, 2, 3, \ldots$, uzyskamy szereg odpowiedzi NIE i w końcu pierwsze TAK dla pewnego $k < n$. To będzie minimalna liczba transferów. Ograniczenie $k < n$ wynika z przedstawionego nieoptymalnego algorytmu z początku artykułu. Tam na pewno po co najwyżej $n - 1$ transferach, w czasie których za każdym razem ubywa jedna osoba, a po ostatnim transferze nawet dwie, wyrównywanie długów zakończy się. Więc optymalna strategia nie może być gorsza. Jeśli zatem dla konkretnego k złożoność naszego algorytmu decyzyjnego wyniosłaby $W(n)$ dla pewnego wielomianu W, to rozwiązanie problemu minimalizacji transferów kosztowałoby co najwyżej $nW(n)$, czyli koszt też byłby wielomianowy.

W celu wykazania, że problem decyzyjny, czy wystarczy k transferów, aby wyrównać rachunki, jest NP-zupełny, wystarczy, po pierwsze, pokazać, że nasz problem jest w klasie NP, a po drugie, sprowadzić dowolny problem NP-zupełny do naszego problemu. Wykazanie, że nasz problem jest w klasie NP, jest proste. Jeśli ktoś nam przedstawia scenariusz dokonania k transferów, to sprawdzić, że faktycznie jest on poprawny, jest łatwo – po prostu wystarczy wykonać przedstawiony scenariusz, weryfikując, czy faktycznie działa. I zajmie to nam czas proporcjonalny do liczby transferów, czyli liniowy. Aby wykazać, że nasz problem jest pośród najtrudniejszych problemów w klasie NP, musimy znaleźć problem NP-zupełny, który sprowadzimy do naszego. Takim problemem może być np. problem połowienia zbioru. Jest to jeden ze znanych problemów NP-zupełnych. Dla zadanego zbioru liczb naturalnych $A = \{a_1, \ldots, a_n\}$ stwierdzić, czy istnieje taki podzbiór indeksów $J \subseteq \{1, \ldots, n\}$, że

$$\sum_{j \in J} a_j = \frac{1}{2} \sum_{j \in \{1,\ldots,n\}} a_j.$$

Czyli czy można tak wybrać podzbiór zbioru A, aby suma jego elementów była dokładnie połową sumy wszystkich elementów zbioru A. Oczywiście, można ten problem rozwiązać w czasie wykładniczym, badając wszystkie możliwe podzbiory zbioru A i sumując dla każdego z nich jego elementy. Dramat polega na tym, że takich podzbiorów jest wykładniczo dużo i już dla stosunkowo niewielkich n czas wykładniczy oznacza, że algorytm będzie działał zbyt długo, a ogólnego algorytmu działającego istotnie szybciej nie znamy, i pewnie takiego algorytmu nie ma.

Pokażemy teraz, jak problem połowienia zbioru sprowadzić do naszego problemu transferów. Załóżmy, że mamy konkretny przypadek problemu połowienia zbioru $A = \{a_1, \ldots, a_n\}$. Niech

$$s = \frac{1}{2} \sum_{j \in \{1,\ldots,n\}} a_j.$$

Tworzymy problem transferów w następujący sposób. Ustalamy n dłużników i dwóch wierzycieli następująco: $[(a_1, \ldots, a_n)(s, s)]$ i pytamy, czy n transferów wystarczy, aby uregulować należności. Jeśli wystarczy, to znaczy, że zbiór nasz da się podzielić na połowy, jeśli nie, to znaczy, że się nie da. Czyli w szczególności, gdybyśmy umieli w czasie wielomianowym rozwiązać nasze zadanie minimalizacji liczby transferów, to umielibyśmy w czasie wielomianowym rozwiązać problem połowienia zbioru. W każdym razie nasz problem transferów jest co najmniej tak trudny, jak problem połowienia zbioru. Koniec dowodu.

Oczywiście, przy naszych narciarskich rozliczeniach te wszystkie rozważania potraktowaliśmy jako zabawę i rozwiązaliśmy problem regulowania długów w następujący sposób: każdy dłużnik wyłożył to, co był winien, na stół, a każdy wierzyciel zabrał to, co nadpłacił i tym samym sprawa została załatwiona. Zauważmy, że to był jednak inny model. Problem powstaje, gdy takiego stołu nie ma, np. gdy jesteśmy na stoku narciarskim i wiatr dmie, a pieniędzy nie ma gdzie położyć, tylko trzeba przekazywać je z ręki do ręki, a ręce marzną i trzeba zdejmować rękawiczki. Albo gdy komputery, chcąc zrównoważyć obciążenie, wymieniają się przetwarzanymi zadaniami i chcą zminimalizować liczbę transmisji... Ale to jest nieco inna bajka.