Jest to zapis odczytu nagrodzonego medalem Filca na XXXIX Szkole Matematyki Poglądowej *Domniemanie – Hipoteza – Dowód*, Sierpień 2007

Jak wygrać milion dolarów w Sapera

Tomasz BARTNICKI, Zielona Góra

Gra w Sapera

Grę w Sapera znają dziś niemal wszyscy użytkownicy komputerów. Nieznany jest jej twórca, ale jej początki sięgają około 1950 roku, kiedy to była dość popularna w wielu krajach jako planszowa gra hazardowa. W późniejszych czasach, wraz z rozwojem informatyki, pojawiały się jej przeróżne komputerowe implementacje. Obecna klasyczna wersja gry, która dostępna jest jako akcesorium we wszystkich wersjach systemu Microsoft Windows oraz w wielu innych systemach operacyjnych, napisana została w 1981 roku przez ówczesnych pracowników Microsoftu, Roberta Donnera i Curta Johnsona.

Planszą do gry w Sapera jest prostokąt podzielony prostopadłymi, pionowymi i poziomymi liniami na kwadratowe pola. Na niektórych losowo wybranych polach umieszczone zostały miny. Na początku gry gracz (Saper) posiada informację o łącznej liczbie min, lecz nie zna ich rozmieszczenia, gdyż wszystkie pola są zakryte. Zadaniem gracza jest zlokalizowanie min i odkrycie wszystkich niezaminowanych pól. W przypadku odkrycia pola, na którym znajduje się mina, gra kończy się porażką. Jeżeli gracz odkryje pole, na którym nie ma miny, to pojawia się na nim liczba (od 1 do 8) informująca o tym, z iloma zaminowanymi polami ono sąsiaduje (w pionie, poziomie lub na ukos). Jeśli odkryte pole nie sąsiaduje z żadną miną, to zamiast liczby 0 pojawia się pusta kratka, a dla ułatwienia wszystkie sąsiednie pola odkrywają się automatycznie. Jeżeli w trakcie gry gracz nabierze pewności, że na którymś z nieodsłoniętych jeszcze pól znajduje się mina, to ma je prawo oznaczyć specjalną chorągiewką.

Milion dolarów

W 1998 roku London T. Clay i jego żona Lavinia D. Clay, małżeństwo milionerów z Bostonu, przy merytorycznym współudziale matematyka Arthura Jaffa z Uniwersytetu Harvarda, utworzyli organizację o charakterze non-profit – Instytut Matematyczny Claya. Celem jego działalności jest zwiększanie i pogłębianie szeroko pojętej wiedzy matematycznej. Jest on realizowany przez popularyzację wśród środowiska naukowego najnowszych osiągnięć matematycznych, rozpoznawanie i formułowanie kluczowych dla tej dziedziny problemów, jak również poprzez system nagród i szkoleń dla wyróżniających się matematyków.

Najgłośniejszym przedsięwzięciem Instytutu Claya było zorganizowane w maju 2000 roku w Paryżu spotkanie milenijne, na którym zaprezentowano siedem otwartych problemów, których rozwiązanie uznano za kluczowe dla rozwoju matematyki w XXI wieku. Czas, miejsce i cel tego spotkania nawiązywały do zorganizowanego 100 lat wcześniej (również w Paryżu) Międzynarodowego Kongresu Matematyków, na którym David Hilbert wygłosił słynny wykład, podczas którego przedstawił 23 zagadnienia obrazujące stan ówczesnej wiedzy (a raczej niewiedzy, gdyż były to problemy nierozwiązane) matematycznej.

Hilbert przedstawił tylko część z nich, pełna lista ukazała się drukiem po Kongresie.

W celu zmobilizowania matematyków do pracy nad problemami milenijnymi Instytut Claya, za rozwiązanie każdego z nich, ufundował nagrodę w wysokości okrągłego miliona dolarów. Można się oczywiście spierać, czy gratyfikacja finansowa jest wystarczającym bodźcem dla matematyka do zajęcia się jakimś problemem, niemniej jednak nagrody te zyskały już znaczny rozgłos, również w środowiskach pozanaukowych. Jako argument dla przeciwników całego przedsięwzięcia może posłużyć fakt, że jeden z problemów milenijnych (hipoteza Poincarégo) został niedawno rozstrzygnięty, lecz autor tego wyniku – Grigori Perelman, odmówił przyjęcia, zarówno Medalu Fieldsa, jaki i należnego mu miliona dolarów.

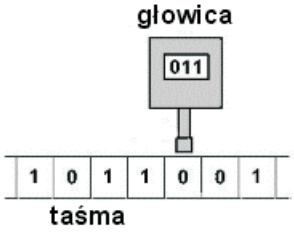
Nie będziemy tu wymieniać wszystkich problemów milenijnych. Warto tylko wspomnieć, że jednym z nich jest hipoteza Riemanna, która sto lat wcześniej

znalazła się także na liście problemów Hilberta (razem z hipotezą Goldbacha stanowiły 8. problem Hilberta). Do sformułowania i zrozumienia istoty każdego z problemów niezbędny jest przeważnie bardzo zaawansowany aparat matematyczny, którego przyswojenie wymaga dość sporego wysiłku od przeciętnego „użytkownika" matematyki. Wyjątkiem od tej reguły jest *problem P kontra NP*, którego istotę postaramy się tu przedstawić w sposób poglądowy, niewymagający żadnej zaawansowanej wiedzy matematycznej i z użyciem jak najmniejszej ilości formalizmów. Na zakończenie pokażemy, że tytuł niniejszego artykułu nie jest wcale tanim chwytem reklamowym, gdyż droga od prostej gry komputerowej do poważnego problemu za milion dolarów jest krótsza niż mogłoby się w pierwszej chwili wydawać.

Alan Turing i jego maszyny

Nie sposób w artykule na temat hipotezy P/NP nie wspomnieć o Alanie Turingu (1912–1954). Jest on słusznie uznawany za ojca współczesnej informatyki teoretycznej. W roku 1937, będąc na ostatnim roku studiów, napisał przełomową pracę *O liczbach obliczalnych*, w której udowodnił istnienie problemów, których maszyna cyfrowa nie jest w stanie rozwiązać za pomocą żadnego algorytmu. Ponieważ Turing nie mógł wówczas dysponować komputerem (gdyż takowe jeszcze nie istniały), to stworzył własny, czysto teoretyczny model znany do dziś jako maszyna Turinga.

Maszyna Turinga składa się z nieograniczonej, podzielonej na komórki taśmy oraz z ruchomej głowicy zapisująco-odczytującej (rys. 1). Na taśmie mogą być zapisywane symbole z ustalonego, skończonego alfabetu (np. 0, 1 oraz symbol pusty). Na początku taśma maszyny jest częściowo zapisana danymi wejściowymi, a głowica znajduje się w pewnym położeniu startowym. W każdym kroku głowica może odczytać i/lub zapisać wartość w pojedynczej komórce, a następnie przesunąć się o dokładnie jedną pozycję w lewo lub w prawo. Po skończonej liczbie kroków maszyna kończy pracę, a symbole zapisane na taśmie stanowią dane wyjściowe. Wydawać by się mogło, że tak prymitywne urządzenie posiada bardzo ograniczone możliwości obliczeniowe, jednak okazuje się, że maszyna Turinga niemal idealnie modeluje działanie współczesnych komputerów.



Rys. 1. Maszyna Turinga

Po wojnie Turing brał udział w projektowaniu pierwszych maszyn cyfrowych. W roku 1966 ustanowiono i nazwano jego imieniem nagrodę, przyznawaną corocznie za wybitne dokonania w dziedzinie informatyki. Obecnie prestiż tej nagrody jest bardzo duży, gdyż uznawana jest ona powszechnie za informatyczny odpowiednik Nagrody Nobla.

Klasa P (polynomial)

Głównym zadaniem informatyki teoretycznej jest badanie złożoności problemów obliczeniowych. Chcemy wiedzieć, jakie problemy dadzą się efektywnie rozwiązać za pomocą odpowiednich algorytmów, a jakie są algorytmicznie trudne. W dalszych rozważaniach ograniczymy się wyłącznie do tzw. *problemów decyzyjnych*, czyli takich, które dają się sformułować w postaci pytania, na które odpowiedź brzmi „tak" lub „nie". Należy dodać, że problem decyzyjny jest zawsze sformułowany w sposób ogólny, a więc niezależny od danych wejściowych. Natomiast jeżeli konkretne dane podstawimy do problemu, to mamy wówczas do czynienia z *instancją* problemu. Na przykład problemem decyzyjnym jest pytanie: *„Czy dana liczba naturalna jest pierwsza?"*, jego zaś instancją jest zdanie: *„Czy liczba 403 jest pierwsza?"*. Każdej z nieskończenie wielu instancji danego problemu możemy przypisać liczbę naturalną n, która określa rozmiar danych wejściowych (liczony np. w bitach).

$403 = 13 \cdot 31$

Algorytmem będziemy nazywać pewną skończoną procedurę (możemy ją utożsamiać z konkretnym programem komputerowym), której zastosowanie do każdej instancji danego problemu prowadzi do jego rozwiązania. Można spodziewać się, że czas działania algorytmu będzie zależał od rozmiaru danych wejściowych poszczególnych instancji. Oznaczamy go jako funkcję $T(n)$

i nazywamy *czasową złożonością algorytmu*. Czasu tego nie mierzymy w tradycyjny, zegarowy sposób, gdyż zależałby on od zbyt wielu nieprzewidywalnych czynników, jak, na przykład, implementacja algorytmu, rodzaj kompilatora i maszyny, na której algorytm wykonano czy umiejętności programisty. Zamiast tego zliczamy zwykle liczbę pewnych podstawowych operacji, co w najbardziej teoretycznym ujęciu może oznaczać policzenie liczby przesunięć głowicy maszyny Turinga.

Mówimy, że problem decyzyjny należy do klasy P, jeżeli istnieje algorytm rozwiązujący go w czasie wielomianowym, czyli gdy istnieje taki wielomian $W(n)$, że $T(n) < W(n)$. Algorytmy działające w czasie wielomianowym często nazywane są szybkimi lub efektywnymi.

Oto kilka przykładów problemów z klasy P.

- **Znajdowanie wzorca w tekście.** Dany jest ciąg znaków o łącznej długości m. Należy stwierdzić, czy zawiera on spójny podciąg, który jest identyczny z zadanym wzorcem o długości n. Istnieją algorytmy rozwiązujące ten problem w czasie liniowym względem jego rozmiaru (czyli $m + n$). Jako empiryczne potwierdzenie szybkości takich algorytmów mogą posłużyć znane wszystkim wyszukiwarki internetowe, które w ułamku sekundy odnajdują zadany wzorzec (często nawet w kilku milionach miejsc).
- **Dwukolorowanie grafu.** Dany jest spójny graf G o n wierzchołkach. Chcemy wiedzieć, czy możemy tak je pomalować używając tylko dwóch kolorów, aby każda para wierzchołków połączonych krawędzią otrzymała różne kolory (grafy, dla których jest to możliwe nazywamy dwukolorowalnymi lub dwudzielnymi). Rozmiar tego problemu jest rzędu n^2, bo graf możemy reprezentować przez zerojedynkową kwadratową macierz sąsiedztw jego wierzchołków. Jeżeli graf G jest dwudzielny, to możemy w czasie wielomianowym znaleźć jego poprawne dwukolorowanie, ponieważ jest ono jednoznaczne z dokładnością do zamiany kolorów. W przeciwnym wypadku algorytm kolorujący szybko stwierdzi, że żądane kolorowanie nie istnieje.
- **Cykl Eulera w grafie.** *Cyklem Eulera* w grafie nazywamy drogę przechodzącą przez każdą jego krawędź dokładnie jeden raz i kończącą się w punkcie wyjścia. Łatwo zauważyć warunek konieczny na to, by graf posiadał cykl Eulera (grafy takie nazywamy eulerowskimi). Musi on być spójny oraz w każdym wierzchołku musi „spotykać się" parzysta liczba krawędzi (mówimy, że stopień takiego wierzchołka jest parzysty). Leonard Euler udowodnił, że warunek ten jest również wystarczający, co uznawane jest za pierwsze twierdzenie teorii grafów. Opierając się na tym twierdzeniu łatwo podać szybki algorytm, który sprawdza, czy dany graf jest eulerowski. Warto zauważyć, że algorytm ten w przypadku, gdy graf jest eulerowski wcale nie znajduje cyklu Eulera (choć i na to znane są szybkie algorytmy), a jedynie stwierdza jego istnienie.
- **Test pierwszości.** Jednym z ważniejszych i przełomowych osiągnięć ostatnich lat było znalezienie (dokonali tego M. Agrawal, N. Kayal i, N. Saxena, *Annals of Mathematics*, 2004) wielomianowego algorytmu stwierdzającego, czy dana liczba naturalna jest pierwsza.

Klasa NP (nondeterministic-polynomial)

Do klasy NP należą problemy decyzyjne, które niedeterministyczna maszyna Turinga potrafi rozwiązać w czasie wielomianowym. Tę formalną definicję przytaczamy, aby wyjaśnić pochodzenie skrótu NP, często błędnie tłumaczonego jako non-polynomial. Niedeterministyczna maszyna Turinga tym się różni od deterministycznej, że jej głowica może się przesuwać w lewo i w prawo o dowolną (a nie tylko o jedną) liczbę pozycji. Zatem każdy algorytm, który jest możliwy do zrealizowania na maszynie deterministycznej może być w niedłuższym czasie zrealizowany na maszynie niederministycznej, co prowadzi do bardzo ważnego spostrzeżenia, że $P \subseteq NP$.

Powyższa definicja klasy NP jest dość abstrakcyjna i trudno w praktyce dostrzec, jakie problemy (oprócz tych z klasy P) do niej należą. Klasę NP możemy równoważnie zdefiniować w nieco inny sposób. Mamy pewien problem decyzyjny, dla którego rozmiar danych wejściowych poszczególnych instancji wyraża się liczbą naturalną n. Mamy również podane potencjalne rozwiązanie tego problemu, które zostało znalezione w sposób niederministyczny, czyli po prostu odgadnięte. Chcemy stwierdzić, czy rozwiązanie to jest poprawne. Mówimy, że problem decyzyjny należy do klasy NP, jeżeli istnieje algorytm weryfikujący rozwiązanie w czasie wielomianowym. Podamy teraz kilka przykładów problemów z klasy NP.

- **Puzzle.** Każdy, kto próbował układać puzzle, wie, że trudność tej układanki rośnie bardzo szybko wraz ze wzrostem liczby jej elementów. Tymczasem sprawdzenie, że nawet bardzo duże puzzle są poprawnie ułożone wymaga zaledwie kilku chwil. Wystarczy spojrzeć na układankę, ewentualnie dotknąć jej ręką, aby przekonać się, że wszystkie jej elementy pasują do siebie idealnie, bądź też stwierdzić, że ktoś popełnił błąd podczas układania. Mamy więc świetny przykład problemu z życia wziętego, o którym możemy intuicyjnie powiedzieć, że jest łatwo weryfikowalny, a więc należy do klasy NP.
- **Trójkolorowanie grafu.** Chcemy wiedzieć, czy wierzchołki danego grafu możemy tak pomalować używając trzech kolorów, aby każda para wierzchołków połączonych krawędzią otrzymała różne kolory. Mając dane potencjalne rozwiązanie tego problemu, czyli kolorowanie wierzchołków grafu, bardzo szybko można zweryfikować jego poprawność. Wystarczy sprawdzić, czy końce każdej krawędzi grafu otrzymały różne kolory oraz czy liczba kodów nie przekracza trzech.
- **Cykl Hamiltona w grafie.** *Cyklem Hamiltona* w grafie nazywamy drogę składającą się z kolejnych, sąsiednich wierzchołków grafu, zawierającą każdy z nich dokładnie jeden raz i kończącą się w punkcie wyjścia. Nie jest w ogólności znany warunek konieczny i wystarczający na to, aby graf był hamiltonowski, jeśli jednak mamy dane pewne uporządkowanie wierzchołków grafu, to bardzo szybko możemy sprawdzić, czy tworzy ono cykl Hamiltona.
- **Problem pirata komputerowego.** Początkujący pirat ma do dyspozycji n identycznych nośników (np. płyt CD), każdy o pojemności V i chce na nie skopiować k plików o rozmiarach $v_1, v_2 \ldots, v_k$, przy czym każdy z plików musi być skopiowany w całości. Załóżmy, że pirat nie wie, jak tego dokonać, ba, nie ma nawet pewności, czy jest to w ogóle możliwe, bo przecież sam warunek $nV \geqslant v_1 + v_2 + \cdots + v_k$ nie daje takiej gwarancji. Jeśli jednak zleci on wykonanie tego zadania innemu, bardziej zaawansowanemu piratowi, to sprawdzenie, czy zostało ono poprawnie wykonane, nie powinno mu już nastręczyć trudności ani zająć zbyt wiele czasu.

Sformułowanie problemu

Wiemy już, że $P \subseteq NP$, co oznacza, że każdy problem decyzyjny, który jest łatwo rozwiązywalny jest również łatwo weryfikowalny. Analiza różnych problemów, których potencjalne rozwiązania dają się szybko zweryfikować (jak te z poprzedniego rozdziału) zrodziła **domniemanie**, że nie każdy problem łatwo weryfikowalny musi być również łatwo rozwiązywalny. I tak sformułowano hipotezę za 1000000$.
Hipoteza.

$$P \neq NP.$$

Gdzie leżą trudności, które powodują, że **dowód** tak łatwej do sformułowania hipotezy nie został dotąd znaleziony? Po pierwsze należy zaznaczyć, że precyzyjne matematyczne sformułowanie hipotezy P kontra NP jest dużo bardziej skomplikowane niż to, które do tej pory przedstawiliśmy. Aby pokazać, że hipoteza jest prawdziwa musielibyśmy wziąć jakiś problem z klasy NP i udowodnić, że nie można go rozwiązać w czasie wielomianowym. Cała trudność

polega na tym, że nie możemy rozpatrywać jakiegoś konkretnego algorytmu, ale musimy wziąć pod uwagę wszystkie możliwe algorytmy, również takie, których nie znamy. Jeszcze trudniejszym zadaniem wydaje się być obalenie hipotezy, gdyż należałoby pokazać, że każdy problem, który jest szybko weryfikowalny da się również szybko rozwiązać. Bardzo ważnym pojęciem, które prowadzi do znacznego uproszczenia sformułowania hipotezy P/NP, jest NP-zupełność. Jak się za chwilę okaże, może ono również odegrać główną rolę w ewentualnym rozstrzygnięciu tej hipotezy.

NP-zupełność

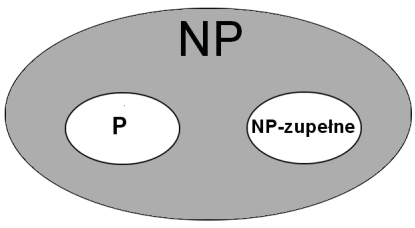
Wszyscy chyba znają dowcip o matematyku chcącym zagotować wodę.

> Co robi matematyk, gdy mając do dyspozycji czajnik, kran z wodą, palnik gazowy i zapałki chce zagotować wodę? Odkręca kran, nalewa wody, zapala gaz, stawia nań czajnik i gotuje. A co zrobi matematyk, gdy ma do dyspozycji te same akcesoria, ale woda znajduje się w czajniku, a gaz został już wcześniej zapalony? Odpowiedź jest prosta... wylewa wodę, zakręca gaz i sprowadza zadanie do poprzedniego przypadku.

W matematyce czasami zdarza się, że gdy mamy udowodnić nowe twierdzenie, to staramy się je sprowadzić do jakiegoś innego znanego twierdzenia bądź lematu. Podobnie możemy postępować z problemami decyzyjnymi, przy czym istotną rolę będzie tu pełnić szybkość sprowadzania jednych problemów do innych.

Załóżmy, że znamy wielomianowy algorytm na rozwiązanie pewnego problemu decyzyjnego Q_1 oraz mamy dany inny problem decyzyjny Q_2. Jeżeli potrafimy w czasie wielomianowym sprowadzić problem Q_2 do problemu Q_1 (lub jego szczególnego przypadku), to wówczas również problem Q_2 możemy rozwiązać w czasie wielomianowym. Sprowadzenie takie polega na przyporządkowaniu każdej instancji problemu Q_2 instancji problemu Q_1 w taki sposób, że odpowiedź „tak/nie" dla instancji problemu Q_1 pociąga za sobą identyczną odpowiedź w odpowiadających jej instancjach problemu Q_2. Procedurę tę nazywamy *wielomianową redukcją* problemu Q_2 do problemu Q_1.

Mówimy, że problem decyzyjny Q z klasy NP jest *NP-zupełny*, jeżeli możemy przeprowadzić wielomianową redukcję każdego problemu z klasy NP do problemu Q. Procedura wielomianowej redukcji wyznacza w całej klasie NP pewien qasi-porządek, według którego możemy porównywać stopień trudności poszczególnych problemów. Jeżeli Q_2 redukuje się wielomianowo do Q_1, to Q_2 jest nietrudniejszy niż Q_1. Problemy NP-zupełne są więc względem tego porządku najtrudniejszymi w całej klasie NP.



Rys. 2. P≠NP

Aby pokazać, że P=NP wystarczyłoby wziąć dowolny problem NP-zupełny i pokazać, że da się on rozwiązać w czasie wielomianowym. Jeśli zaś P≠NP to naturalnym wydaje się pomysł, aby poszukiwanie jakiegoś trudnego problemu z klasy NP zacząć od tych najtrudniejszych, czyli NP-zupełnych (patrz rysunek 2). Nasuwa się jedno zasadnicze pytanie: czy istnieją i czy są znane jakiekolwiek problemy NP-zupełne?

Problemy NP-zupełne

Problemy spełnialności formuł logicznych (SAT, 3SAT). Mamy daną formułę złożoną z pewnej liczby zmiennych logicznych, nawiasów oraz operatorów koniunkcji, alternatywy i negacji (problem SAT). Możemy dodatkowo założyć, że formuła ta zapisana została w bardzo regularnej postaci, mianowicie składa się ona z dowolnej liczby nawiasów przedzielonych znakami koniunkcji, zaś w każdym nawiasie występuje alternatywa dokładnie trzech (niekoniecznie różnych) zmiennych logicznych z możliwym znakiem negacji na dowolnej z nich (problem 3SAT). Rozwiązania problemów SAT i 3SAT polegają na stwierdzeniu, czy istnieje takie podstawienie wartości logicznych 0 i 1 pod zmienne występujące w formule, aby, zgodnie z zasadami rachunku zdań, wartość logiczna

całej formuły wynosiła 1. Łatwo zauważyć, że problemy te należą do klasy NP, gdyż mając podstawienie konkretnych wartości za zmienne możemy szybko określić, jaka jest wartość logiczna całej formuły. W wyrażeniu

$$(p \vee q \vee s) \wedge (\bar{p} \vee \bar{q} \vee \bar{r}) \wedge (\bar{s} \vee q \vee r) \wedge (p \vee \bar{q} \vee \bar{r})$$

podstawienie $p = 1$, $q = 0$, $r = 1$, $s = 1$ daje

$$(1 \vee 0 \vee 1) \wedge (0 \vee 1 \vee 0) \wedge (0 \vee 0 \vee 1) \wedge (1 \vee 1 \vee 0) = 1.$$

W 1971 roku Stephen Cook pokazał, że SAT jest problemem NP-zupełnym, co było jednym z ważniejszych osiągnięć w teorii złożoności obliczeniowej, gdyż otworzyło drogę do dalszych badań nad NP-zupełnością. Rok później Richard Karp udowodnił, że NP-zupełny jest również 3SAT oraz 20 innych problemów kombinatorycznych. Obaj naukowcy za swoje wyniki zostali wyróżnieni prestiżową Nagrodą Turinga. Największą trudnością w dowodzie Cooka było to, iż musiał on zostać przeprowadzony poniekąd od podstaw, z wykorzystaniem takich teoretycznych modeli, jak maszyny Turinga.

Obecnie znanych jest ponad trzy tysiące problemów NP-zupełnych i ich lista jest ciągle poszerzana. Prezentowane w poprzednim rozdziale problemy trójkolorowania grafu, cyklu Hamiltona i pirata komputerowego są również NP-zupełne. W celu pokazania, że jakiś nowy problem Q jest NP-zupełny, wystarczy stwierdzić, że należy on do klasy NP (co przeważnie jest bardzo łatwe), a następnie wziąć jakiś problem z bogatej kolekcji znanych problemów NP-zupełnych i zredukować go wielomianowo do problemu Q lub jego szczególnego przypadku. Przedstawimy teraz w zarysie jak może wyglądać taka redukcja.

Problem kliki w grafie. *Kliką* w grafie nazywamy taki jego podgraf, w którym każde dwa wierzchołki są połączone krawędzią. Liczbę wierzchołków w klice nazywamy jej *rozmiarem*. Mamy stwierdzić, czy w danym grafie G istnieje klika rozmiaru k. Jasne jest, że problem kliki należy do klasy NP, gdyż mając danych k wierzchołków grafu możemy w wielomianowym czasie sprawdzić, czy tworzą one klikę. Ponadto, jeżeli w problemie tym przyjmiemy, że k jest stałe, to będzie on również w klasie P, gdyż stwierdzenie istnienia kliki rozmiaru k w n-wierzchołkowym grafie wymaga, w najgorszym wypadku, sprawdzenia wszystkich k-elementowych podzbiorów wierzchołków, a tych jest $\binom{n}{k}$, co jest wielomianem względem n. Jeżeli jednak k będzie odpowiednio (np. liniowo) związane z rozmiarem grafu, to problem kliki staje się NP-zupełny.

Dowód. Sprowadzimy problem 3SAT do następującego przypadku problemu kliki: „Czy w danym n-wierzchołkowym grafie G istnieje klika rozmiaru $k = \lfloor \frac{n}{3} \rfloor$?" Daną mamy formułę logiczną złożoną z k nawiasów i z dokładnie trzech (niekoniecznie różnych) zmiennych lub ich negacji w każdym nawiasie. Mamy, zatem w całej formule łącznie $n = 3k$ wystąpień zmiennych lub ich negacji. Dla każdej takiej formuły konstruujmy graf G na n wierzchołkach, w następujący sposób:

(i) każdemu wystąpieniu zmiennej w formule odpowiada dokładnie jeden wierzchołek w grafie (zauważmy, że jeżeli zmienna występowała wiele razy, to utworzymy dla niej wiele wierzchołków),
(ii) krawędzią łączymy wszystkie pary wierzchołków, które odpowiadają zmiennym z różnych nawiasów z wyjątkiem par postaci: zmienna i jej negacja.

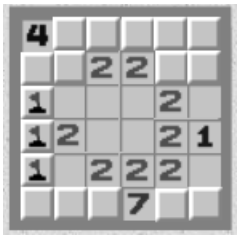
Załóżmy, że w tak skonstruowanym grafie istnieje klika rozmiaru k. Z uwagi na to, że w obrębie wierzchołków z jednego nawiasu nie było krawędzi, to klika taka musi zawierać dokładnie po jednym wierzchołku z każdego nawiasu. Jeżeli teraz za zmienne odpowiadające wierzchołkom z kliki podstawimy wartość logiczną 1 (a za pozostałe zmienne dowolną wartość), to w każdym nawiasie pojawi się przynajmniej jedna 1, a więc całe wyrażenie będzie miało również wartość logiczną 1. Zauważmy, że podstawienie takie jest zawsze możliwe, gdyż z uwagi na konstrukcję grafu G w jednej klice nie mogły pojawić się jednocześnie

zmienna wraz ze swoją negacją. Analogicznie postępujemy w drugą stronę – mianowicie mając niezerowe podstawienie możemy z każdego nawiasu wybrać po jednej zmiennej przyjmującej wartość 1, a wierzchołki im odpowiadające utworzą w grafie G klikę rozmiaru k. Ostatnim etapem dowodu jest stwierdzenie, że opisana na samym początku procedura redukcji odbywa się w czasie wielomianowym względem rozmiaru instancji problemu wyjściowego (3SAT). Ale tu wystarczy zauważyć, że dla formuły o n zmiennych (a więc również tego rozmiaru) skonstruowaliśmy graf na n wierzchołkach, który może być zapamiętany jako zerojedynkowa macierz kwadratowa wymiaru n, czyli o n^2 wyrazach. □

Rys. 3. Gdzie są miny?

Saper jest trudny

Przyszła wreszcie pora, aby zaprezentować, jak dzięki grze w Sapera można rozstrzygnąć hipotezę P/NP i zdobyć należny za to milion dolarów. Zacznijmy od prostego zadania. Czy układ przedstawiony na rysunku 3 ma rozwiązanie, czyli czy istnieje takie rozmieszczenie min na nieodsłoniętych polach, które będzie zgodne z informacjami na polach już odsłoniętych? Drogą szybkiej dedukcji i eliminacji nietrudno znaleźć żądane rozwiązanie i na dodatek można pokazać, że jest ono jednoznaczne (pozostawiamy to jako łamigłówkę dla czytelnika). A co będzie, jeśli analogiczny problem postawimy dla układu przedstawionego na rysunku 4? W tym przypadku możemy łatwo stwierdzić, że rozwiązanie nie istnieje. Powyższe dwa zadania były dość proste, gdyż rozmiar planszy, którą mieliśmy przeanalizować nie był zbyt duży. Można się jednak spodziewać, iż to samo (sformułowane poniżej) zadanie na planszy o znacznych rozmiarach nie będzie już takie łatwe.



Rys. 4. Coś tu nie gra?

Problem Sapera (Minesweeper Consistency Problem (MCP)). Czy zadany układ początkowy z Sapera posiada rozwiązanie?

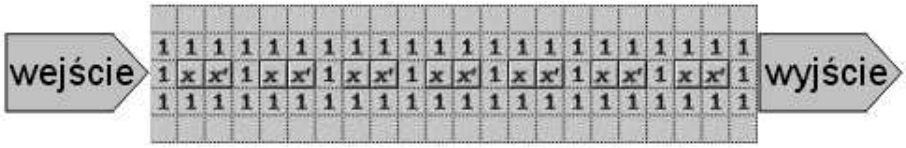
Jest to dobrze postawiony problem decyzyjny, więc możemy pytać o jego złożoność obliczeniową. Zauważmy, że mając dane potencjalne rozwiązanie problemu Sapera możemy w krótkim czasie (czyli wielomianowo zależnym od rozmiaru planszy) sprawdzić, czy jest ono poprawne, a więc MCP z pewnością należy do klasy NP. W roku 2000 Richard Kaye pokazał, że Problem Sapera jest NP-zupełny ([1]).

NP-zupełność Sapera

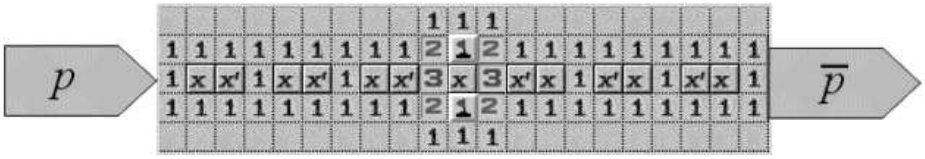
Aby pokazać, że problem Sapera jest NP-zupełny zredukujemy do niego problem SAT. Odbędzie się to w dwóch krokach według poniższego schematu:

formuła logiczna
⇓
obwód logiczny
⇓
konfiguracja Sapera

Pierwszy krok jest powszechnie znaną procedurą zapisywania danej formuły logicznej w postaci pewnego schematu zwanego obwodem logicznym. Składa się on z pewnej liczby wejść, które odpowiadają zmiennym użytym w formule, jednego wyjścia, które odpowiada wartości logicznej całej formuły oraz odpowiedniej liczby bramek logicznych (AND, OR i NOT), które odpowiadają operatorom koniunkcji, alternatywy i negacji. Całość obwodu jest tak połączona przewodami, aby realizował on działanie odpowiadającej mu formuły.



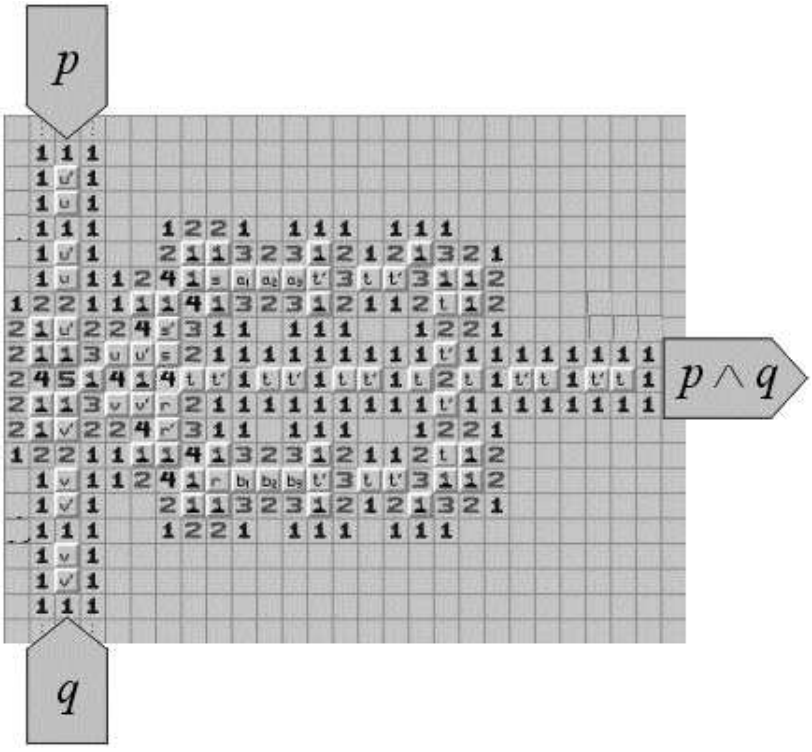
Rys. 5. Przewód logiczny



Rys. 6. Bramka NOT

Drugim krokiem, który w zarysie przedstawimy, będzie zastąpienie obwodu logicznego odpowiednią konfiguracją Sapera w taki sposób, aby istnienie rozwiązania dla niej (w sensie MCP) pociągało istnienie pozytywnego rozwiązania w problemie SAT. Konfigurację tę będziemy w systematyczny sposób budować z odpowiednich modułów różnego typu. Pierwszym z nich jest przewód logiczny przedstawiony na rysunku 5. Jego zadaniem jest łączenie poszczególnych modułów całego układu i przenoszenie pomiędzy nimi sygnału cyfrowego o wartości 0 lub 1. Zauważmy, że w przewodzie takim miny mogą znajdować się albo na wszystkich polach oznaczonych x, bądź też na polach oznaczonych x'. Ustalmy, że jeżeli na polu x znajduje się mina, to przewodem płynie sygnał 0, jeśli zaś mina znajduje się na polu x', to płynie nim sygnał 1. Należy również ustalić, że każdym takim przewodem sygnał płynie w określonym kierunku, w tym przypadku z lewej do prawej.

Konfiguracja na rysunku 6 przedstawia bramkę NOT. Łatwo sprawdzić, że sygnał wchodzący przewodem wejściowym (z lewej strony) zostanie na wyjściu zamieniony na przeciwny. Nieco dłuższej i dokładniejszej analizy wymaga układ z rysunku 7, który jest bramką AND.



Rys. 7. Bramka AND

Bramkę OR możemy zbudować z bramki AND i bramek NOT (można zbudować ją też nieco łatwiej w inny sposób), a więc mamy już główne moduły niezbędne do zbudowania każdego obwodu logicznego. Oprócz nich konieczne będzie jeszcze kilka pomocniczych konfiguracji odpowiedzialnych np. za rozszczepianie przewodów logicznych, łączenie ich, zmianę kierunku (z poziomego na pionowy i odwrotnie) lub krzyżowanie. Richard Kaye przewidział, zaprojektował i przeanalizował wszystkie te układy, następnie wskazał algorytm budowania z nich całego żądanego obwodu logicznego oraz udowodnił, że czas działania tego

algorytmu zależy wielomianowo od rozmiaru instancji problemu wyjściowego (SAT), a tym samym, że problem Sapera jest NP-zupełny.

Zamiast zakończenia

Możemy już udzielić odpowiedzi na tytułowe pytanie. Aby zdobyć milion dolarów nie wystarczy być dobrym w grze w Sapera, ale trzeba jeszcze rozwiązać trudny matematyczny problem z tą grą związany. Są zatem dwie możliwości: pierwsza to wymyślenie wielomianowego algorytmu, który będzie potrafił rozstrzygać, czy zadany układ z Sapera ma rozwiązanie, druga to pokazanie, że algorytm taki nie istnieje. W pierwszą możliwość prawie nikt dzisiaj nie wierzy, a nawet ci nieliczni, którzy uważają, że P=NP twierdzą, iż jeśli nawet uda się to kiedykolwiek udowodnić, to raczej w sposób niekonstruktywny, bez podania konkretnego algorytmu na któryś problem NP-zupełny. Przekonanie, że P≠NP jest dziś tak silne, że w potocznym rozumieniu problemy NP-zupełne uważa się za bardzo trudne, a na tym domyśle opiera się nawet bezpieczeństwo wielu poważnych systemów informatycznych.

Napisaliśmy powyżej o dwóch możliwych rozstrzygnięciach hipotezy P/NP nie wspominając nic o trzeciej, o której zaczyna być ostatnimi czasy coraz głośniej. Chcąc rozstrzygnąć jakikolwiek problem powinniśmy najpierw ustalić, na gruncie jakiej teorii mamy tego dokonać, a co za tym idzie, z jakich aksjomatów możemy korzystać. Skoro problem P/NP dotyczy równości dwóch zbiorów, to chcąc go rozstrzygnąć, prędzej czy później, będziemy musieli skorzystać z któregoś aksjomatu teorii mnogości (ZFC). Może się jednak okazać, że hipoteza P kontra NP, podobnie jak hipoteza continuum, jest od tych aksjomatów niezależna.

Czyżby więc drogą do zdobycia miliona dolarów było pokazanie, że hipoteza P kontra NP jest nierozstrzygalna?!
Sapere aude, drogi Czytelniku!

Literatura

[1] R. Kaye, *Minesweeper is NP-complete*, Mathematical Intelligencer **22**, (2000, 2), 9-15

[2] I. Stewart, *Million-dollar Minesweeper*, Scientific American **283/4**, (October 2000), 94-95

[3] `http://www.claymath.org/millennium/P_vs_NP/`