Rozważmy Masterminda

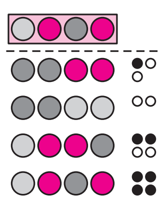
Dominika PAWLIK[*], *Aleksander ZABŁOCKI*[*]

Gra Mastermind jest rozrywką głównie dla gracza odgadującego. Przypomnijmy: musi on ustalić, jaki *kod* (ciąg czterech kolorowych szpilek) ułożył przeciwnik, posiłkując się odpowiedziami otrzymywanymi na zadawane pytania. Pytania muszą mieć postać „Jak bardzo kod przypomina ciąg X?”, zaś odpowiedzią są dwie liczby: trafień właściwych kolorów na właściwych pozycjach oraz trafień kolorów na pozycjach niewłaściwych (odpowiednio: *trafienia celne* i *niecelne*; te pierwsze będziemy też nazywać po prostu *trafieniami*). Oczywiście chodzi o to, by odgadnąć kod jak najszybciej.

Kiedy wymyślono Masterminda? Oficjalnie w roku 1970, jednak już wcześniej znano grę w wersji z czterema cyframi zamiast kolorowych szpilek. Roku powstania tej wersji nie znamy; w każdym razie „1970” ma przynajmniej jedną cyfrę trafioną na właściwej pozycji.

W Masterminda można grać, gimnastykując pomysłowość na bieżąco – ale matematyk chciałby też poszukać ogólnej metody. Jaka jest najmniejsza liczba pytań, która zawsze wystarczy do zwycięstwa? To niełatwy problem, ale możemy wspomóc się komputerem: właśnie w ten sposób Donald Knuth uzyskał w 1977 roku wynik czterech pytań. Komputery nie radzą sobie jednak z uogólnioną wersją gry, gdy kod zawiera n szpilek (zamiast 4), a do dyspozycji jest k kolorów (zamiast 6). Nie zdziwi nas to, gdy pomyślimy, jakich rachunków wymaga uzasadnienie, że do zwycięstwa potrzeba przynajmniej t ruchów. Teoretycznie należy przejrzeć drzewo *stanów gry* (możliwych ciągów pytań i odpowiedzi) do głębokości $t-1$ ruchów włącznie. Dla gry standardowej oraz $t=4$, jako że możliwych pytań w grze jest $6^4 = 1296$, zaś odpowiedzi 14, samych stanów na głębokości 3 mamy $(1296 \cdot 14)^3 \approx 5{,}9$ biliona! Na szczęście możemy zauważyć, że stany różniące się jedynie przemianowaniem kolorów i/lub pozycji są właściwie równoważne – jeśli przejrzeliśmy możliwości wygrania w jednym z nich, nie trzeba już zajmować się tym drugim. Zmniejsza to liczbę *istotnych* stanów do około 415 milionów (tj. ponad 10000-krotnie!), co – w połączeniu z pewnymi dodatkowymi usprawnieniami – czyni zadanie dostępnym dla domowego komputera. Zauważmy jednak, że w ogólności nasz problem wciąż zachowuje się wykładniczo względem n, k, t: powiększając te parametry choćby o 1, możemy zwiększyć trudność setki razy. Nic więc dziwnego, że nawet dla $k = 2$ udało się dotychczas (z wykorzystaniem dużo bardziej złożonych technik) znaleźć optymalne strategie jedynie dla $n \leqslant 8$.

Ściśle mówiąc, cztery pytania wystarczą do *poznania* kodu – ale wg zasad trzeba jeszcze uzyskać odpowiedź „zgadza się”, co wymaga dodatkowego pytania. My będziemy je konsekwentnie pomijać.



Rys. 1. Przykładowa rozgrywka Masterminda. Ramka na górze zawiera ukryty kod; poniżej znajdują się kolejne pytania (zgadnięcia) oraz uzyskiwane odpowiedzi: czarna szpilka oznacza trafienie celne, biała – niecelne. Gracz odgadujący wygrywa, gdy otrzyma odpowiedź ∷ w określonej liczbie ruchów. (My przyjmiemy, że wystarczy mu *pewność*, że otrzyma ją za chwilę.)

Skoro nie umiemy znaleźć najlepszej strategii dla gry z n szpilkami w k kolorach, poszukajmy (bez komputera) możliwie dobrej. Na początek załóżmy, że $k = 2$: szpilki są jedynie ciemnoszare i kolorowe. Łatwo wówczas wygrać, zadając $n+1$ pytań: w pierwszym wszystkie szpilki są ciemnoszare, w następnych jedna kolorowa szpilka „wędruje” po kolejnych pozycjach. (Właściwie ostatnie z tych pytań jest zbędne, zatem można wygrać w n ruchach.) A czy da się lepiej?

Aby odpowiedzieć na to pytanie, zaczniemy od nieco innego zadania. Danych jest n monet, z których każda waży 5 albo 7,5 grama, oraz zwykła waga, podająca łączną masę umieszczonych na niej przedmiotów. Należy odróżnić monety cięższe od lżejszych, wykonując jak najmniej ważeń. Oczywiście, można zważyć każdą monetę osobno, jednak istnieją lepsze rozwiązania. Na przykład, dla pięciu monet o wagach A, B, C, D, E wystarczą cztery ważenia:

Tyle ważą jednozłotówka i jedno euro.

$$(*) \qquad A+D+E, \quad B+D+E, \quad C+E, \quad A+B+C+D+E.$$

Istotnie: dodając pierwsze trzy wyniki i odejmując ostatni, otrzymujemy $D + 2E$; łatwo sprawdzić, że wyznacza to naraz D oraz E. Wtedy już łatwo znaleźć A, B, C.

Oryginalne prace o ważeniu monet:

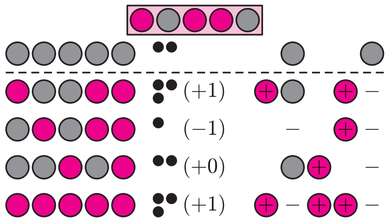
V. Grebinski, G. Kucherov, *Optimal reconstruction of graphs under the additive model*, Algorithmica 28 (1997).

N. Bshouty, *Optimal algorithms for the coin weighing problem with a spring scale*, Proc. Annual Conf. on Learning Theory, 2009.

Czy powyższy przykład pomaga zważyć n monet dla $n > 5$? Oczywiście tak: grupujemy monety w piątki i ważymy każdą piątkę w czterech ważeniach, co daje około 20-procentową oszczędność wobec „naiwnych” n ważeń. Okazuje się jednak, że im bardziej rośnie n, tym jest lepiej: w ogólności wystarczy wykonać około $\frac{2n}{\log n}$ ważeń, co daje np. pięciokrotną oszczędność dla $n \approx 1000$. Ba, jeśli rozpoczniemy od zważenia razem wszystkich monet, a następnie zaplanujemy ponad $\frac{2n}{\log n}$ *losowych* ważeń, to z dużym prawdopodobieństwem taki schemat ważeń pozwoli na odgadnięcie wag monet w dowolnym ich układzie.

W tym artykule sformułowanie „A wynosi około B” lub „$A \sim B$” oznacza, że iloraz $\frac{A}{B}$ jest dowolnie bliski jedynce, jeśli tylko n jest odpowiednio duże.

[*]doktoranci, Wydział Matematyki, Informatyki i Mechaniki, Uniwersytet Warszawski



Rys. 2. Przykładowa gra dwukolorowa. Lewa kolumna zawiera pytania, zbudowane na podstawie strategii ważenia monet (∗), środkowa – liczby (celnych) trafień oraz różnice trafności w stosunku do pierwszego pytania. Prawa kolumna pokazuje, gdzie konkretnie wystąpiły trafienia (oczywiście odgadujący początkowo nie ma tej wiedzy). Znaki + i − przedstawiają różnice w trafieniach w stosunku do pierwszego pytania. Pojawiają się one dokładnie tam, gdzie w pytaniu była kolorowa szpilka. W ten sposób kolejne odpowiedzi mówią nam o sumach wag szpilek na wybranych pozycjach w kodzie. (Wybór pozycji następuje poprzez umieszczenie na niej w pytaniu kolorowej szpilki.)

A jak ma się to do Masterminda? Za chwilę pokażemy strategię dla gry dwukolorowej, wykorzystującą ważenie monet. Najpierw zauważmy, że podane powyżej wartości 7,5 oraz 5 gramów nie są istotne dla rozwiązania zadania – równie dobrze monety ciężkie mogłyby ważyć 1 g, a lekkie (niech fizycy wybaczą) 0 g lub wręcz minus jeden gram. Umówmy się więc, że szpilka kolorowa waży gram, zaś ciemnoszara minus gram. Na początek zadajmy pytanie złożone z ciemnoszarych szpilek i oznaczmy liczbę trafień przez s. Jeżeli teraz zmienimy gdziekolwiek w pytaniu szpilkę na kolorową, spowodujemy zwiększenie liczby trafień o 1, jeśli odpowiednia szpilka w kodzie jest kolorowa, i jej zmniejszenie, jeśli jest ciemnoszara – zatem zmiana równa się wadze rozważanej szpilki w kodzie. W ogólności, możemy w ten sposób zważyć szpilki, wykorzystując w roli wagi zasady Masterminda! Na przykład, aby poznać sumę wag pierwszej i trzeciej monety, zadajemy pytanie ●●●●●●···● (z kolorowymi szpilkami na odpowiednich pozycjach) i odejmujemy s od liczby trafień (patrz też rys. 2).

Skoro umiemy odgadnąć kod po około $\frac{2n}{\log n}$ pytaniach, może da się jeszcze lepiej? Policzmy: możliwych kodów jest 2^n, zaś możliwych odpowiedzi na jedno pytanie jest $n+1$. Po t ruchach dysponujemy ciągiem t takich odpowiedzi; jeśli ma on pozwalać na jednoznaczne odtworzenie kodu, musi zachodzić $(n+1)^t \geqslant 2^n$. Nierówność ta staje się prawdziwa dla t równych mniej więcej $\frac{n}{\log n}$, co pokazuje, że opisaną powyżej strategię można poprawić w najlepszym razie około dwukrotnie. Można więc powiedzieć, że uzyskaliśmy „większość” osiągalnego usprawnienia w porównaniu z rozwiązaniem naiwnym wymagającym $n+1$ pytań. Co ciekawe, nasze rozwiązanie wykorzystuje jedynie informacje o trafieniach; trafienia niecelne nie są potrzebne.

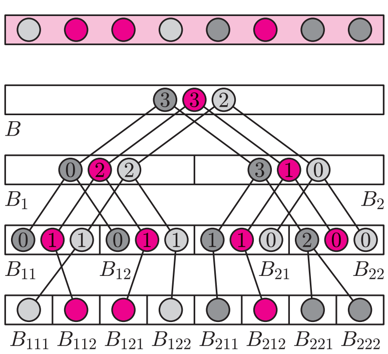
Oryginalna praca o grze wielokolorowej:

B. Doerr i in., *Playing Mastermind with many colors*, Proc. Sympos. on Discrete Algorithms, 2013.

Zrozumiawszy z grubsza przypadek $k=2$, przechodzimy do Masterminda z większą liczbą kolorów. Ponownie odwołamy się do zagadnienia ważenia monet, jednak w trudniejszej wersji: tym razem o n monetach wiadomo tylko tyle, że ich wagi (w gramach) są całkowite nieujemne oraz że suma wag wszystkich monet nie przekracza pewnej wartości M. Wówczas okazuje się, że wystarczy $\frac{8n}{\log n}$ ważeń, jeśli $M \leqslant n$, oraz $\frac{4(n+M)}{\log n}$ ważeń w przeciwnym razie. (Podobnie jak poprzednio, dobrych schematów ważeń jest wówczas bardzo wiele.)

Od obu tych założeń można się łatwo uwolnić kosztem dodatkowych pytań: jeśli n nie jest potęgą dwójki, liczba pytań może się podwoić, zaś jeśli k jest bardzo duże – powiększyć o około $\frac{k}{n}$. Tego drugiego nie da się zresztą uniknąć.

Dla wygody założymy, że liczba szpilek n jest potęgą dwójki, a liczba dostępnych kolorów wynosi również n. W takiej sytuacji opisana przez nas metoda będzie wymagać około $8n \log\log n$ pytań. We wstępnym opisie założymy dodatkowo, że istnieje kolor w ogóle niewystępujący w kodzie i jest on nam znany; nazwiemy go *zerowym* i oznaczymy przez ○. Później pokażemy, jak radzić sobie bez niego.



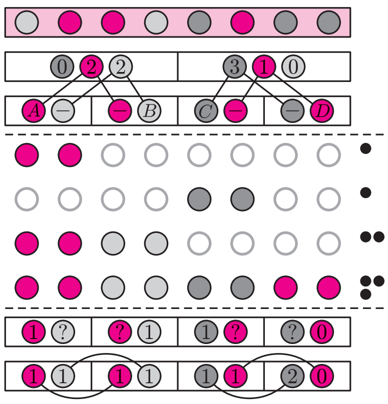
Rys. 3. Podział kodu na bloki oraz monety związane z tymi blokami; jest to zarazem schemat działania algorytmu. Odcinki łączą monety z ich pod-monetami; waga monety jest sumą wag jej pod-monet. Na ostatnim poziomie zaznaczono jedynie monety o wadze 1 (pozostałe mają wagę 0); wyznaczają one rozwiązanie zagadki.

W przypadku dwukolorowym rolę monet pełniły pojedyncze szpilki o wagach ± 1. Teraz będzie inaczej. Po pierwsze, pojedyncza rozgrywka Masterminda będzie odwoływać się do problemu monet wielokrotnie, za każdym razem dla innego układu monet (każde takie wydarzenie nazwiemy *grą w monety*). Po drugie, rolę monety będzie zawsze pełnił *zbiór* szpilek danego koloru, znajdujących się w kodzie w obrębie danego bloku pozycji; wagą takiej monety będzie liczba wspomnianych szpilek. Na przykład, dla kodu ○●●○●●●● moneta kolorowa dla bloku 1–4 ma wagę 2, zaś jasnoszara dla bloku 5–8 wagę 1. W pierwszym kroku algorytmu ustalimy wagi wszystkich monet dla bloku B zawierającego wszystkie pozycje. W drugim kroku ustalimy wagi wszystkich monet dla obu połów tego bloku, które oznaczymy przez B_1 i B_2. Następnie zważymy monety odpowiadające ćwiartkom $B_{11}, B_{12}, B_{21}, B_{22}$, i tak dalej, co da nam rozwiązanie (patrz rys. 3).

Ustalenie wag monet dla bloku B jest proste: np. waga ciemnoszarej jest równa trafności pytania ●●●···●. Podobnie dla mniejszych bloków: np. wagą monety ciemnoszarej dla bloku B_{21} jest trafność pytania ○○○○●●○○. Co więcej, sumą tejże wagi oraz wagi monety kolorowej dla bloku B_{11} jest trafność pytania ●●○○●●○○. Stąd już widać, że każdy zestaw monet związanych z *rozłącznymi* (nieprzecinającymi się) blokami można wydajnie zważyć za pomocą gry w monety (rys. 4).

Wykorzystując fakt, że waga monety jest sumą wag jej pod-monet (rys. 3), można uzyskać dodatkowo około dwukrotną oszczędność (rysunek 4 zawiera przykład tego usprawnienia).

Trzeba jeszcze wyjaśnić, które monety dobieramy do kolejnych gier. W pierwszym kroku (tj. dla bloku B) ważymy monety pojedynczo. W następnych krokach łączymy monety w grupy, tak by każda grupa zawierała po (maksymalnie) jednej monecie dla każdego z bloków odpowiedniego poziomu; każda grupa bierze udział w osobnej



Rys. 4. Przebieg jednej gry w monety podczas trzeciego kroku algorytmu. Wagi monet dla bloków B_1, B_2 są już znane; chcemy ustalić wagi ich pod-monet, których jest 8. Wybieramy 4 z nich (po jednej w każdym bloku) i znajdujemy ich wagi A, B, C, D, zadając pytania odpowiadające ważeniom

$$A, \quad C, \quad A+B, \quad A+B+C+D.$$

Wagi pozostałych 4 monet można by ustalić analogicznie, ale w tym przypadku można je od razu wyliczyć na podstawie już posiadanych informacji.

grze w monety. Kluczowy dla efektywności jest fakt, że jeśli pewna moneta ma wagę 0 (co dla wielu kolorów musi się zdarzać bardzo często), to można wyłączyć z ważeń obie jej pod-monety, ponieważ odpowiedni kolor w ogóle nie występuje w danym bloku (rys. 4).

Czas wytłumaczyć, jak poradzić sobie bez koloru zerowego. Trik polega na tym, żeby stworzyć sobie taki kolor, a konkretniej – rozpocząć algorytm od dodatkowych pytań, które pozwolą dla każdej z osobna pozycji ustalić pewien kolor, który na pewno tam nie występuje. Będzie to kolor zerowy *dla tej pozycji* (rys. 5). Zauważmy, że takie posunięcie jest bezpieczne dzięki temu, że cały czas interesują nas wyłącznie trafienia celne.

Jak więc znaleźć wydajnie *zestaw* kolorów zerowych? Okazuje się to proste, a zarazem podobne do gry w Masterminda dwukolorowego. Zadajmy najpierw pytanie ●●●···●, a następnie ●●●●···●. Jeśli trafność wzrosła, druga szpilka jest kolorowa; jeśli spadła, jest ciemnoszara; jeśli nie zmieniła się, nie może być kolorowa ani ciemnoszara. Zatem aby odrzucić po jednym kolorze na każdej pozycji, wystarczy „przespacerować się” kolorową szpilką po pozycjach; łącznie zadajemy $n+1$ pytań. Wygląda to podobnie do naiwnej strategii dla gry dwukolorowej, i nie bez konsekwencji: pozostawiamy Czytelnikowi jako ćwiczenie wykazanie, że za pomocą ważenia monet można kolory zerowe wyznaczyć znacznie szybciej.

Zakończyliśmy nareszcie opis strategii dla Masterminda o n szpilkach i n kolorach. Ilu pytań wymaga ta strategia? Najprościej znaleźć odpowiedź w sytuacji, gdy wszystkie szpilki w kodzie mają różny kolor: wówczas wszystkie monety ważą 0 lub 1, zatem z blokiem i-tego poziomu związanych jest dokładnie $\frac{n}{2^{i-1}}$ monet o niezerowej wadze. Wówczas i-ty krok algorytmu składa się z $\frac{n}{2^{i-1}}$ gier w monety, z których w każdej uczestniczy 2^{i-1} monet. Z rozwiązania problemu monet wynika, że jedna taka gra wymaga około

$$\frac{2 \cdot 2^{i-1}}{\log(2^{i-1})} = \frac{2^i}{i-1}$$

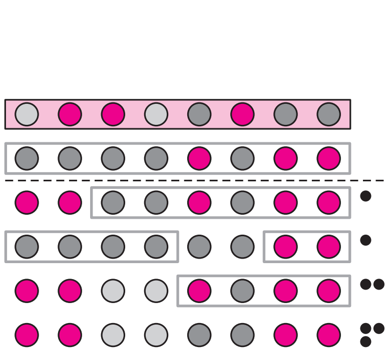
ważeń, zatem cały krok zajmuje około $\frac{2n}{i-1}$ pytań Masterminda. Czyli w sumie:

$$n \cdot \left(1 + \frac{2}{2} + \frac{2}{3} + \ldots + \frac{2}{\log n - 1}\right) \sim 2n \cdot \log\log n.$$

Dla $i = 1$ wzór $\frac{n}{i-1}$ nie ma sensu, ale skądinąd wiemy, że w oczywisty sposób pierwszy krok zajmuje n pytań.

Jeśli teraz dopuścimy powtarzające się kolory w kodzie, to podobne (choć nieco trudniejsze) rozumowanie – oczywiście z wykorzystaniem rozwiązania ogólniejszego problemu monet – pokazuje, że nasza metoda wymaga nie więcej niż około $16n \log\log n$ pytań.

Jak do tej pory, nie wykorzystaliśmy jeszcze nigdzie wiedzy o liczbie trafień niecelnych, która jest przecież dostępna. Czyżby była ona całkiem niepotrzebna? To zależy od liczby kolorów w grze. Jeśli $k > n$, wygodnie rozpocząć grę od ustalenia, jakie kolory w ogóle występują w kodzie (co umożliwi zastosowanie opisanej powyżej metody); trafienia niecelne są tu istotną pomocą. Dla odmiany, w przypadku $k = 2$ za ich pomocą można zaoszczędzić co najwyżej jeden ruch.



Rys. 5. Przykładowy zestaw kolorów zerowych dla kodu (nad linią) oraz praktyczna realizacja czterech pytań z rysunku 4 z wykorzystaniem tego zestawu (pod linią). Zauważmy, że trafność pytań pozostała niezmieniona. Szare ramki obejmują szpilki będące teoretycznie koloru zerowego.

Problem uogólnionego Masterminda oczywiście nie jest kluczowy dla współczesnej nauki; dlaczego w takim razie miałby nas interesować? Zanim odpowiemy, zastanówmy się, dlaczego naukowcy w ogóle zajmują się grami. Pierwszym argumentem jest na pewno atrakcyjna treść, jednak ich zaletą jest też kondensacja różnych problemów obliczeniowych oraz teoretycznych. Jeśli nauczymy się efektywnie grać w szachy bądź w go, albo znajdziemy dowód optymalności strategii dla Masterminda, być może uda się wykorzystać wypracowane metody w bardziej życiowych zagadnieniach. Tymczasem dla uogólnionego Masterminda wciąż nie znamy ani strategii lepszej od opisanej powyżej, ani dowodu, że nie wystarczy wykonać $C \cdot n$ ruchów (dla stałej C niezależnej od n). Kto wie, może rozstrzygnięcie tej niejasności okaże się – pośrednio – szerzej przydatne.

Mastermind ma też inne ciekawe uogólnienia. Jak grać, jeśli przeciwnik może dwukrotnie skłamać? Można ważyć monety w sposób odporny na dwa kłamstwa – co z kolei wiąże się z teorią kodów korekcyjnych, które „naprawiają” drobne błędy, np. w przesyle danych. A to przydaje się wszędzie: w Internecie, sieciach komórkowych, nośnikach danych...