Jakub Wojtaszczyk
Treść zadania, Opracowanie

Jakub Łącki
Program

Dostępna pamięć: 32 MB.

OI, Etap III, dzień pierwszy, 14.04.2010

Gra w minima

Asia i Basia poznały niedawno grę w minima, która bardzo im się spodobała. Zasady tej gry są następujące. Na stole leży pewna liczba kart, na których napisane są dodatnie liczby całkowite. Gracze wykonują ruchy na przemian (pierwszy ruch należy do Asi). W swoim ruchu gracz wybiera dowolną liczbę kart (ale co najmniej jedną) i zabiera je ze stołu. Za taki ruch gracz otrzymuje liczbę punktów równą najmniejszej z liczb zapisanych na zabranych przez niego kartach. Gra kończy się, gdy ze stołu zostanie wzięta ostatnia karta. Każdemu graczowi zależy na tym, aby różnica między jego końcowym wynikiem a wynikiem przeciwnika była jak największa.

Asia i Basia szybko spostrzegły, że w tej grze musi istnieć strategia optymalna. Poprosiły Cię więc o napisanie programu, który dla danego zestawu kart wyznaczy, jaki będzie wynik gry, przy założeniu, że obaj gracze grają optymalnie.

Wejście

W pierwszym wierszu standardowego wejścia znajduje się jedna liczba całkowita n ($1 \leqslant n \leqslant 1\,000\,000$), oznaczająca liczbę kart. W drugim wierszu znajduje się n dodatnich liczb całkowitych $k_1, k_2, \ldots, k_n$ ($1 \leqslant k_i \leqslant 10^9$), pooddzielanych pojedynczymi odstępami, oznaczających liczby zapisane na kartach.

Wyjście

Twój program powinien wypisać na standardowe wyjście jeden wiersz zawierający jedną liczbę całkowitą — liczbę punktów, o które Asia wygra z Basią, pod warunkiem, że obie będą grać optymalnie (jeżeli więcej punktów uzyska Basia, to należy wypisać liczbę ujemną).

Przykład

Dla danych wejściowych:	*poprawnym wynikiem jest:*
`3` `1 3 1`	`2`

Wyjaśnienie do przykładu: *Asia zabiera ze stołu kartę z liczbą 3, za co otrzymuje trzy punkty. Basia bierze obie pozostałe karty, za co otrzymuje jeden punkt. Gra skończy się wynikiem trzy do jednego, a zatem Asia wygra dwoma punktami.*

Rozwiązanie

Wprowadzenie

Zastanówmy się na początek, w jaki sposób opisać dowolną sytuację, która może pojawić się na stole w trakcie rozgrywki. Nietrudno dostrzec, że to, ile punktów ma aktualnie każdy z graczy, nie wpływa na ich strategie działania. Liczy się jedynie aktualny zestaw kart na stole.

Możemy więc dla każdego układu kart K określić wartość $wynik(K)$ jako największą możliwą różnicę punktów pomiędzy graczem wykonującym pierwszy ruch a jego (optymalnie grającym) przeciwnikiem. Na K można patrzeć jak na zbiór liczb, a dokładniej mówiąc na *multizbiór*, czyli strukturę podobną do zbioru, w której dopuszczamy powtórzenia elementów.

W przypadku, gdy na stole jest tylko jedna karta, na której zapisana jest liczba n, gracz nie ma wyboru — musi ją zabrać, tym samym osiągając n punktów przewagi nad przeciwnikiem. Tę własność można zapisać jako $wynik(\{n\}) = n$.

W przypadku, gdy kart jest więcej, wzór na najlepszy możliwy do osiągnięcia wynik przedstawia się następująco:

$$wynik(K) = \max_{S \subseteq K,\ S \neq \emptyset} (\min(S) - wynik(K \setminus S)). \tag{1}$$

W danej sytuacji gracz może zdjąć ze stołu dowolny niepusty podzbiór kart (oznaczony powyżej jako S). Za ten ruch otrzymuje liczbę punktów równą minimum tego multizbioru. Następnie rozpoczyna się kolejka jego przeciwnika, o którym wiemy, że gra optymalnie. Zatem, napotkawszy na stole sytuację $K \setminus S$, zdobędzie on $wynik(K \setminus S)$ punktów przewagi nad aktualnym graczem, co wyjaśnia drugą część powyższego wzoru.

Wzór ten tłumaczy się wprost na algorytm działający w czasie wykładniczym. Używa on techniki programowania dynamicznego i oblicza wszystkie wartości $wynik(K)$, korzystając z przedstawionej przed chwilą własności. Musi przejrzeć wszystkie podzbiory każdego z podzbiorów kart z wejścia, co przy użyciu masek bitowych i kilku sztuczek programistycznych daje się zrealizować w czasie $O(3^n)$. Po szczegóły odsyłamy do programów `grab4.cpp` i `grab5.pas`. Tego typu rozwiązania uzyskiwały na zawodach 20 punktów. Działanie tych programów jest niezdefiniowane dla dużych danych wejściowych, w szczególności czasem udzielają błędnych odpowiedzi, przez co muszą być zakwalifikowane do rozwiązań niepoprawnych.

Rozwiązanie wzorcowe

Rozwiązanie wzorcowe jest prostsze aniżeli rozwiązanie przedstawione powyżej, ale wyprowadzenie go wymaga zrozumienia, jak wyglądają optymalne strategie w grze w minima. Do wniosków przedstawionych poniżej najprościej dojść, rozgrywając (chociażby samemu) kilka partii gry i badając, które posunięcia działają, a które są ewidentnie bez sensu. Zacznijmy od następującej obserwacji:

Obserwacja 1. Istnieje optymalna strategia gry, która w każdym ruchu wybiera multizbiór zawierający kartę o największej wartości spośród dostępnych.

Dowód: Niech K będzie aktualnym układem kart, a S będzie podzbiorem kart wybranych przy tym układzie w pewnej optymalnej strategii. Niech dalej $x = \min(K)$. Jeśli $x \in S$, to nie ma czego dowodzić. Przyjmijmy zatem, że $x \notin S$. Będziemy porównywali sytuację po ruchach polegających na wzięciu odpowiednio S i $S \cup \{x\}$. Zauważmy, że obydwa dają nam tyle samo punktów. Jeżeli na nasz ruch polegający na wzięciu $S \cup \{x\}$ nasz przeciwnik reaguje, biorąc pewien podzbiór T, to po ruchu S nasz przeciwnik jest w stanie doprowadzić do identycznej sytuacji, biorąc $T \cup \{x\}$ (i zyskując identyczną liczbę punktów). Zatem przeciwnik ma po ruchu S strategię, która kończy się nie gorzej niż jego strategia po ruchu $S \cup \{x\}$ — to oznacza, że ruch S nie jest lepszy od ruchu $S \cup \{x\}$. ■

Załóżmy teraz, że mamy przed sobą karty o wartościach $k_1 \leqslant k_2 \leqslant \ldots \leqslant k_m$. Wiemy już, że na pewno wśród kart, które wybierzemy, będzie karta k_m. Jedną z opcji jest, że będzie to jedyna karta, którą weźmiemy. Wtedy przeciwnik będzie wykonywał ruch, mając do dyspozycji karty $k_1, k_2, \ldots, k_{m-1}$, i — z definicji — końcowy wynik będzie równy $k_m - wynik(\{k_1, k_2, \ldots, k_{m-1}\})$. Drugą opcją jest, że wybierzemy jeszcze jakieś karty.

Obserwacja 2. Optymalna spośród strategii, w których w pierwszym ruchu wybieramy przynajmniej dwie karty, w tym k_m, daje dokładnie $wynik(\{k_1, k_2, \ldots, k_{m-1}\})$ punktów.

Dowód: Wpierw pokażmy, jak uzyskać taką liczbę punktów. Rozważmy strategię, która daje tyle punktów w grze z kartami $k_1, k_2, \ldots, k_{m-1}$. Ta strategia każe w pierwszym ruchu wziąć multizbiór S. Zatem w oryginalnej grze możemy w pierwszym ruchu wziąć $S \cup \{k_m\}$, a potem kontynuować wedle naszej strategii, jako że sytuacja po pierwszym ruchu jest identyczna.

Analogicznie rozważmy sytuację, w której w oryginalnej grze pierwszy gracz wziął multizbiór $S \cup \{k_m\}$, przy czym S to niepusty podzbiór $\{k_1, k_2, \ldots, k_{m-1}\}$, i dostał za to $\min(S)$ punktów. Sytuacja jest więc taka sama, jak gdyby w grze na kartach $\{k_1, k_2, \ldots, k_{m-1}\}$ gracz pierwszy wybrał multizbiór S. Gracz drugi może zatem grać tak, żeby łączny wynik gry nie przekroczył $wynik(\{k_1, k_2, \ldots, k_{m-1}\})$. ■

Oczywiście, optymalną strategią dla m kart jest wybranie lepszej z dwóch powyższych strategii. To pozwala nam usprawnić wzór (1). Niech $k_1 \leqslant k_2 \leqslant \ldots \leqslant k_n$ będzie uporządkowanym, wyjściowym układem kart. Wprowadźmy oznaczenie

$$wynik(m) \stackrel{\text{def}}{=} wynik(\{k_1, k_2, \ldots, k_m\}).$$

Na mocy powyższych obserwacji otrzymujemy następującą zależność, która jest podstawą działania rozwiązań wzorcowych:

$$wynik(m) = \max\{wynik(m-1),\ k_m - wynik(m-1)\} \quad \text{dla } m \geqslant 2. \tag{2}$$

Wszystkie wartości $wynik(m)$ można obliczyć w złożoności $O(n)$, jednak konieczne jest uprzednie posortowanie danych wejściowych. Z tego powodu czas działania programu wzorcowego to $O(n \log n)$. Znaleźć go można w plikach `gra.cpp`, `gra1.cpp` (wersja autora) oraz `gra2.pas`.

Warto zauważyć jeszcze, że choć na wejściu znajduje się do miliona liczb nieprzekraczających 10^9, końcowy wynik będzie nie większy niż największa z tych liczb, co można pokazać za pomocą prostego argumentu indukcyjnego. Co więcej, wynik nigdy nie będzie ujemny, gdyż pierwszy gracz może w pierwszym ruchu zdjąć ze stołu wszystkie karty i zagwarantować sobie dodatni rezultat. Dzięki tym spostrzeżeniom zbędne staje się korzystanie z typów całkowitych 64-bitowych.

Rozwiązania niepoprawne

Jak zwykle w grach, można znaleźć wiele niepoprawnych strategii, które na pierwszy rzut oka mogą wydawać się ciekawe — np. „weź wszystkie karty", „zawsze bierz największą", „weź wszystkie karty o największej wartości", czy szereg innych. Przykładowe błędne rozwiązania są zaimplementowane w plikach `gra[b1-b8].[cpp|pas]`. Na zawodach zdobywały one od 0 do 30 punktów.

Testy

Dla każdego testu określamy dwa parametry: n oznacza liczbę kart, a m maksymalną wartość zapisaną na kartach. Przygotowane testy można podzielić na 5 kategorii:

- **losowy:** losowe wartości kart.
- **pierwiastek:** wartości kart rosną od 1 do m proporcjonalnie do pierwiastka z numeru karty.
- **wykładniczy:** wartości kart rosną proporcjonalnie do funkcji wykładniczej od numeru karty.
- **hiperbola:** wartość na i-tej karcie to $\lfloor \frac{m}{i} \rfloor$.
- **grupy:** ustalamy pewną liczbę rozłącznych, oddalonych od siebie przedziałów i następnie losujemy wartości kart należące do tych przedziałów.

Ponadto, w niektórych testach do wartości kart dodawane były małe, losowe liczby całkowite.

Nazwa	n	m	Opis
gra1a.in	1	1	jedna karta
gra1b.in	2	3	dwie karty
gra1c.in	2	5	dwie karty
gra1d.in	6	86	test wygenerowany ręcznie
gra2.in	10	9	pierwiastek
gra3.in	343	322	pierwiastek
gra4.in	3 843	8 582	grupy

Nazwa	n	m	Opis
gra5.in	81 231	999 955 097	losowy
gra6.in	103 243	9 998 828	wykładniczy
gra7.in	213 243	1 000 000	hiperbola
gra8.in	871 843	502 119 250	grupy
gra9.in	1 000 000	999 984 457	wykładniczy
gra10.in	1 000 000	1 000 000 000	hiperbola