Zadania z listy 9

Antoni Kościelski

1 Zadanie 2

1.1 Treść zadania

Napisz funkcję, która dla zadanej wartości n wyznaczy liczbę wszyskich możliwych ustawień n hetmanów na szachownicy o wymiarach $n \times n$, w których hetmany nie atakują się wzajemnie.

Najpierw zauważmy, że sfomułowanie tego zadania nie jest do końca jasne. Zacznijmy jednak od ustalenia jakiegoś sposobu opisywania ustawień.

1.1.1 Opis ustawienia

Interesują nas takie ustawieniach, w których hetmany nie atakują się wzajemnie. W takich ustawieniach na szachownicy w każdej poziomej linii (w każdym wierszu) znajduje się dokładnie jeden hetman, i to samo dotyczy linii pionowych (czyli kolumn).

Linie szachownicy zazwyczaj numerujemy począwszy od lewego górnego rogu. Informatycy lubią numerować od 0. Tak więc pole w lewym górnym rogu znajduje się w kolumnie o numerze 0 i w wierszu o tym samym numerze. Na prawo od tego pola znajduje się umieszczone w tym samym wierszu (o numerze 0) i w kolumnie o numerze 1. Ostatni od góry wiersz ma numer $n - 1$, podobnie ostatnia kolumna (licząc oczywiście od lewej), o ile mamy szachownicę o wymiarach $n \times n$.

Wiele ustawień, w tym wszystkie, które nas interesują, możemy opisać podając ciąg liczb $b_0, b_1, \ldots, b_{n-1}$. Taki ciąg oznacza, że mamy na myśli takie ustawienie hetmanów, w którym hetman z i-tej kolumny znajduje się w b_i-tym wierszu. Oczywiście, w ten sposób można opisać takie ustawienia, w których w każdej kolumnie znajduje się dokładnie jeden hetman. Możemy też dodatkowo przyjąć, że umieszczenie w opisie ustawienia liczby -1 oznacza, że w odpowiedniej kolumnie nie znajduje się żaden hetman, oraz że mamy prawo skrócić opis pomijając -1 znajdujące się na końcu ciągu.

1.1.2 Doprecyzowanie zadania

Nietrudno zauważyć, że dwie osoby obserwujące to samo ustawienie z różnych pozycji mogą opisać je w różny sposób. Na przykład ustawienie 0,2,4,1,3 hetmanów na szachownicy o wymiarach 5×5 zostanie opisane przez osobę stojącej z boku, z prawej strony jako 2,4,1,3,0. Powstaje więc pytanie, czy te dwa ciągi opisują dwa różne ustawienia, czy też jedno i to samo.

Na to pytanie nie można jednoznacznie odpowiedzieć. Najprościej przyjąć, że szachownicę obserwujemy z ustalonego miejsca, i jeżeli widzimy coś innego, to są to różne

ustawienia. Tak rozumiane ustawienie jest tożsame z jego opisem, licząc ustawienia wystarczy więc policzyć ich opisy. Tak też pojęcie ustawienia będziemy rozumieć w dalszym ciągu.

Oczywiście, możemy także przyjąć, że jeżeli do szachownicy podejdą z czterech stron różne osoby i opiszź, co widzą, to ich opisy będą opisywać to samo ustawienie. Takie podejście prowadzi do bardziej skomplikowanych rachunków.

A może nawet ciągi 0,2,4,3,1 oraz 2,0,3,1,4 opisują to samo ustawienie?

1.2 Rozwiązanie zadania

Jeżeli już wiemy, co chcemy policzyć, to możemy pokusić się o rozwiązanie zadania. Problem ustawienia hetmanów można rozwiązać także za pomocą następującego programu:

```
int hetmany(int n)
{ int k;
  b[0] = 0; //ustaw hetmana w lewym, górnym rogu
  k = 1; //ustaw hetmana nad drugą kolumną
  while ( k >= 0 ) //dopóki na szachownicy jest jakikolwiek hetman
  {  b[k]++; //przesuń ostatniego hetmana w dół
     if (b[k] < n) //jeżeli pozostał on na szachownicy
     {  if ( poprawne(k, b[k]) ) //jest poprawnie ustawiony
        {  if ( k = n-1 ) //i stoi w ostatniej kolumnie
             return k; //to zakończ
           else //a gdy nie stoi w ostatniej kolumnie
             k++; //weź i ustaw kolejnego hetmana nad następną kolumną
        }
     }
     else //a gdy przesuwany hetman wypadł poza szachownicę
      {b[k] = -1; k--;}//zabierz go i zajmij się poprzednim (jeżeli jest)
  }
}
```

Ma on trochę prostszą strukturę od programu podanego na wykładzie, łatwiej go modyfikować i analizować. Warto zastanowić się nad jego poprawnością. Zakładamy, że jest on uruchamiany i analizowany zawsze po uprzednim wypełnieniu tablicy `b` wartościami -1.

W podanym programie jest wywoływana funkcja `poprawne`. O tej funkcji musimy założyć, że wykorzystuje tablicę b i zmienną n, ale nie zmienia ich wartości. Przyjmujemy też, że zwraca wartości 0 i 1 (albo fałsz i prawda). Co więcej, po wywołaniu postaci `poprawne(k, a)` zwraca wartość 1 dokładnie wtedy, gdy ciąg liczb `(b[0], b[1], ..., b[k-1], a)` opisuje poprawne rozmieszczenie hetmanów na szachownicy o wymiarze $n \times n$.

Jeżeli usuniemy z wyżej przedstawionego programu instrukcję `return k;`, to powinien on przejrzeć wszystkie możliwe poprawne ustawienia hetmanów w początkowych kolumnach (pamiętajmy, że nie przegląda ustawień np. bez hetmana w pierwszej kolumnie), w szczególności wszystkie możliwe poprawne ustawienia hetmanów na szachownicy o danym wymiarze.

Jeżeli w tym programie dodatkowo umieścimy licznik (zmienną, której wartość początkowa jest równa 0), i zamiast instrukcji `return` będziemy zwiększać jego wartość o 1, końcowa wartość licznika będzie równa liczbie znalezionych poprawnych maksymalnych rozmieszczeń hetmanów na szachownicy o danym wymiarze.

Jeżeli z programu usuniemy instrukcję `return` i dodamy do niego licznik, ale instrukcję zwiększającą o 1 wartość licznika dodamy tuż przed instrukcją **if** `(poprawne(k, b[k]))`, to końcowa wartość licznika będzie równa liczbie wywołań funkcji `poprawne` wykonanych podczas wykonania programu. Możemy dodatkowo zwiększanie licznika uzależnić od wartości parametru k. W ten sposób uzyskamy informację o tym, ile razy program bada rozmieszczenie określonej liczby hetmanów.

1.3 Poprawność rozwiązania

Zawsze trzeba zastanawiać się nad poprawnością programu. Jedna wątpliwość już została zgłoszona. Być może licząc możliwe ustawienia hetmanów za pomocą podanego wyżej programu (z odpowiednią modyfikacją) niektóre ustawienia będą liczone wielokrotnie. Część wątpliwości została już jakoś rozstrzygnięta, ale może nadal to samo ustawienie jest liczone kilkakrotnie. Może niektóre ustawienia są pomijane. Może nawet program nic nie liczy, gdyż zapętla się. Aby odpowiedzieć na te pytania, trzeba program staranniej przeanalizować.

Program ten ma wiele niezmienników. Na przykład podczas działania programu nie ulega zmianie wartość zmiennej n. Dalej, wartość zmiennej k jest mniejsza od n i na ogół nieujemna, spadek wartości k poniżej 0 powoduje zatrzymanie programu.

W tablicy b elementy $b[0], b[1], \ldots, b[k-1]$ są nieujemne, a elementy $b[i]$ dla i spełniających $k < i < n$ są równe -1. Zdarza się, że element $b[k]$ ma wartość -1, ale prawie natychmiast staje się on nieujemny. Zawsze jest on nieujemny, w momencie wywołania funkcji $poprawne(k, b[k])$. Wymienione niezmienniki są dość oczywiste.

Następujący fakt podaje nieco ważniejszy niezmiennik:

Fakt 1.1 Niezmiennikiem pętli z rozważanego programu jest stwierdzenie, że ciąg ($b[0]$, $b[1], \ldots, b[k-1]$) opisuje poprawne rozmieszczenie hetmanów.

W dalszym ciągu będziemy rozważać zbiór ciągów liczb naturalnych $< n$ o długościach nie większych od n. Wykorzystamy również fakt, że elementy tego zbioru są uporządkowane leksykograficznie.

Zdefiniujmy sobie teraz pomocniczą funkcję U przyporządkowującą niektórym liczbom naturalnym ciągi liczb naturalnych, taką że

$$U(i) = (b[0], b[1], \ldots, b[k]),$$

która numerowi porządkowemu i wywołania funkcji *poprawne* w podanym wyżej programie, uruchomionym z ustaloną daną $n > 1$, przypisuje wskazany ciąg wartości elementów tablicy b wyliczony w tuż przed wykonaniem tego wywołania. Krótko mówiąc, $U(i)$ to ciąg opisujący ustawienie hetmanów badane podczas i-tego wywołania funkcji *poprawne*.

O funkcji U można wykazać następujące fakty[1]:

[1]Gdyby można było korzystać z dodatkowego elementu tablicy b, o indeksie -1 (taka możliwość jest w Pascalu), lub w sztuczny sposób zwolnić element $b[0]$ (na przykład przesuwając dane z tablicy b o jedno pole w prawo) lub też gdyby numerować wiersze i kolumny szachownicy od 1, to bez obawy, że wyjdziemy poza zakres indeksów, można by wykonać wstępne przypisanie `b[k]++` przed pętlą, a pozostałe przypisania przenieść na koniec pętli. Wtedy poniższe fakty stałyby się normalnymi niezmiennikami pętli z rozważanego programu.

Fakt 1.2 Jeżeli ciągi $U(i)$ oraz $U(i+1)$ są określone oraz $U(i)$ opisuje poprawne ustawienie hetmanów, to ciąg $U(i+1)$ jest następnikiem $U(i)$ w sensie porządku leksykograficznego.

Fakt 1.3 Jeżeli ciągi $U(i)$ oraz $U(i+1)$ są określone oraz $U(i)$ nie opisuje poprawnego ustawienie hetmanów, to ciąg $U(i+1)$ jest następnikiem w sensie porządku leksykograficznego ciągu $U(i)$ uzupełnionego do ciągu n elementowego o stosowną liczbę wyrazów $n-1$.

Dowody obu faktów nie są trudne, w większości przypadków wymagają prześledzenia jednorazowego wykonania instrukcji znajdujących się w pętli.

Z tych dwóch lematów właściwie wynikają wszystkie, interesujące nas fakty o programie. Tak więc zawsze kończy on pracę: podczas pracy stale rośnie pewien parametr przyjmujący skończenie wiele wartości. Dalej, program nie analizuje żadnego ustawienia dwukrotnie: każde ustawienie później analizowane jest większe w sensie porządku leksykograficznego. Co prawda, pewne ustawienia nie są analizowane przez program, ale są to wyłącznie ustawienia niepoprawne: jeżeli ustawienie $(b_0, b_1, \ldots, b_k)$ jest niepoprawne, a ustawienie $(a_0, a_1, \ldots, a_l)$ spełnia nierówności

$$(b_0, b_1, \ldots, b_k) <_{lex} (a_0, a_1, \ldots, a_l) \leqslant_{lex} (b_0, b_1, \ldots, b_k, n-1, \ldots, n-1),$$

to ciąg $(a_0, a_1, \ldots, a_l)$ jest wydłużeniem ciągu $(b_0, b_1, \ldots, b_k)$ i też opisuje ustawienie niepoprawne. Tak więc zliczając podczas pracy programu tylko ustawienia poprawne jakiegoś rodzaju powinniśmy uzyskać poprawne rezultaty.

2 Zadanie 1

2.1 Treść zadania

Mówimy, że algorytm przeszukiwania z nawrotami z wykładu sprawdza jakieś ustawienie n hetmanów, jeżeli jest wywoływana funkcja `isfree(n-1, y)` dla $0 \leqslant y < n$ (czyli sprawdzamy możliwość dostawienia n-tego hetmana do ustawionych już poprawnie $n-1$ hetmanów). Uzasadnij, że dla $n \geqslant 4$ algorytm sprawdza nie więcej niż $(n/2+1) \cdot n!$ różnych ustawień na szachownicy n hetmanów.

2.1.1 Kilka uwag

Po pierwsze, nie ma istotnych różnic między algorytmem, o którym mowa w zadaniu, podanym na wykładzie, i sformułowanym w tym tekście, w rozwiązaniu zadania 2. W tym tekście, rolę funkcji `isfree` przejmuje funkcja `poprawne`. Analiza obu algorytmów jest bardzo podobna.

Sformułowanie zadania zawiera mylące fragmenty. Nie jest jasno powiedziane, czym jest n. W przytoczonej definicji sprawdzania n może być dowolną liczbą. Przyjmujemy, że algorytm (przytoczony wyżej lub podany na wykładzie) sprawdza ustawienie m hetmanów, jeżeli wywołuje funkcję `poprawne` (lub jej odpowiednik) z pierwszym parametrem o wartości $m-1$.

Po starannym przeczytaniu należy dość do wniosku, że chodzi w nim o oszacowanie liczby maksymalnych ustawień hetmanów sprawdzanych przed znalezieniem pierwszego

poprawnego takiego ustawienia. Liczba n w poleceniu oznacza więc rozmiar szachownicy, czyli parametr wywołania algorytmu. Podane oszacowanie jest trochę zbyt duże.

Po pobieżnym przeczytaniu doszedłem do wniosku, że w zadaniu chodzi o oszacowanie liczby wszelkich badanych ustawień. Obie interpretacje zadania mają bardzo podobne rozwiązania i wymagają pokonania tych samych trudności.

2.2 Główny lemat

Przypuśćmy, że będziemy uruchamiać rozważany program z parametrem n.

Na razie będą nas interesowały zbiory

$$P_{a,m} = \{(b_0, b_1, \ldots, b_{m-1}) \ : \ b_0 = a \wedge \exists\, i \; U(i) = (b_0, b_1, \ldots, b_{m-1})\},$$

a więc zbiory tych ustawień m hetmanów, w których hetman z pierwszej kolumny znajduje się w wierszu o numerze a, i które są sprawdzane podczas działania algorytmu.

Pokażemy następujący

Lemat 2.1 *Dla liczb naturalnych $n \geqslant 2$ i $a < n$ oraz liczby m takiej, że $n \geqslant m \geqslant 2$, zbiór $P_{a,m}$ ma najwyżej*

$$\frac{n!}{(n-m+1)!}$$

elementów.

Dowód. Każdemu ustawieniu $(a, b_1, b_2, \ldots, b_{m-2}, b) \in P_{a,m}$ przyporządkowujemy ciąg $(b_1, b_2, \ldots, b_{m-2})$ oraz liczbę b. Przyporządkowanie to jest różnowartościowe.

Z Faktu 1.1, definicji U otrzymujemy, że ciąg $(a, b_1, b_2, \ldots, b_{m-2})$ opisuje poprawne rozmieszczenie hetmanów i jako taki, jest różnowartościowy. Stąd otrzymujemy, że ciąg $(b_1, b_2, \ldots, b_{m-2})$ jest różnowartościowym ciągiem długości $m-2$ elementów zbioru $\{i < n : i \neq a\}$, mającego $n-1$ elementów. Wiadomo z kombinatoryki, że takich ciągów jest najwyżej

$$(n-1) \cdot (n-2) \cdot \ldots \cdot (n-(m-2)) = \frac{(n-1)!}{(n-m+1)!}.$$

Tak więc każdemu elementowi zbioru $P_{a,m}$ przyporządkowaliśmy parę, której pierwsza współrzędna należy do zbioru o $\frac{(n-1)!}{(n-m+1)!}$ elementach, a druga – do zbioru n elementowego. Takich par jest najwyżej

$$\frac{n!}{(n-m+1)!}.$$

Ponieważ przyporządkowanie to jest różnowartościowe, zbiór $P_{a,m}$ ma najwyżej taką właśnie liczbę elementów. □

Wniosek 2.2 *Zbiór $\bigcup_{m=1}^{n} P_{a,m}$ ma najwyżej $(e-1)\cdot n! \leqslant 2\cdot n!$ elementów, a więc rozważany algorytm sprawdza najwyżej taką liczbę ustawień hetmanów, w których pierwszy hetman znajduje się w wierszu a.*

Dowód. Ponieważ wzór z poprzedniego lematu jest słuszny także dla $m = 1$, więc mamy

$$\left|\bigcup_{m=1}^{n} P_{a,m}\right| \leqslant \sum_{m=1}^{n} \frac{n!}{(n-m+1)!} = n! \cdot \sum_{m=1}^{n} \frac{1}{(n-m+1)!} = n! \cdot \sum_{m=1}^{n} \frac{1}{m!} \leqslant n! \cdot (e-1). \ \square$$

2.3 Kolejny krok

Mamy oczywisty

Lemat 2.3 *Dla dowolnych liczb naturalnych n i $a < n$, przynajmniej jedna z liczb a oraz $n - a - 1$ jest mniejsza od $n/2$.*

Dowód. Gdyby obie te liczby były przynajmniej równe połowie n, to ich suma, czyli $n - 1$, byłaby przynajmniej równa n. □

Natomiast szachiści doskonale wiedzą, że zachodzi

Lemat 2.4 *Jeżeli $(b_0, b_1, \ldots, b_m)$ opisuje poprawne rozmieszczenie hetmanów na szachownicy o wymiarach $n \times n$, to także $(n - 1 - b_0, n - 1 - b_1, \ldots, n - 1 - b_m)$ opisuje poprawne rozmieszczenie.* □

Stąd otrzymujemy

Wniosek 2.5 *Jeżeli na szachownicy o wymiarach $n \times n$ można poprawnie rozstawić n hetmanów, to można je tak poprawnie rozstawić, aby w pierwszej kolumnie hetman stał w wierszu o numerze $< n/2$.* □

Wiadomo też, że zachodzi

Lemat 2.6 *Jest najwyżej $(n + 1)/2$ liczb naturalnych $< n/2$.* □

Łącząc dotychczasowe ustalenia możemy też wykazać

Lemat 2.7 *Jeżeli na szachownicy o wymiarach $n \times n$ można poprawnie rozstawić n hetmanów, to program z rozdziału 1.2 zatrzymuje się bezpośrednio po sprawdzeniu pewnego poprawnego rozstawienia n hetmanów, w którym hetman z pierwszej kolumny znajduje się w wierszu o numerze $< n/2$. Opis tego rozstawienia należy więc do zbioru $\bigcup_{a<n/2} P_{a,n}$.*

Dowód. Ten lemat już ma trudniejszy dowód. Wymaga pokazania, że pewne szczególne ustawienie jest sprawdzane przez program. Próba bardziej szczegółowej analizy naszego programu została przedstawiona w rozdziale 1.3.

Po pierwsze musimy wiedzieć, że program zatrzyma się. Tak będzie, ponieważ przegląda kolejno, w porządku leksykograficznym, wszystkie poprawne rozstawienia. Więc w końcu znajdzie poprawne rozmieszczenie n hetmanów, jeżeli takie istnieje.

Powinniśmy ponadto pokazać, że dla takiego programu stwierdzenie „wszystkie rozmieszczenia n hetmanów mniejsze w sensie porządku leksykograficznego lub równe $(b[0], b[1], \ldots, b[k])$ nie są poprawne" jest niezmiennikiem pętli.

Teraz możemy wziąć dwa poprawne rozstawienia: $(b_0, b_1, \ldots, b_{n-1})$, czyli to, które spowodowało zatrzymanie programu, oraz $(a_0, a_1, \ldots, a_{n-1})$ takie, że $a_0 < n/2$, istniejące na mocy założenia. Nierówność

$$(a_0, a_1, \ldots, a_{n-1}) <_{lex} (b_0, b_1, \ldots, b_{n-1})$$

przeczy temu, że wyżej podane stwierdzenie jest niezmiennikiem programu. Nierówność przeciwna implikuje, że $b_0 \leqslant a_0 < n/2$. □

Lemat 2.8 *Jeżeli na szachownicy o wymiarach $n \times n$ można poprawnie rozstawić n hetmanów, to program z rozdziału 1.2 analizuje tylko te rozstawienia n hetmanów, które należą do $\bigcup_{a<n/2} P_{a,n}$.*

Dowód. Zbiór $\bigcup_{a<n/2} P_{a,n}$ wraz z każdym elementem zawiera wszystkie ciągi n elementowe mniejsze od niego w porządku leksykograficznym. Ponieważ program analizuje rozmieszczenia w porządku leksykograficznym, i ostatnie analizowane rozmieszczenie n hetmanów należy do tego zbioru, więc wszystkie wcześniej analizowane rozmieszczenia n hetmanów też należą do tego zbioru. □

Wniosek 2.9 *Jeżeli na szachownicy o wymiarach $n \times n$ można poprawnie rozstawić n hetmanów, to program z rozdziału 1.2 analizuje nie więcej niż $(n+1)!/2$ rozstawień n hetmanów.*

Dowód. Wystarczy policzyć liczbę elementów zbioru, o którym jest mowa w poprzednim lemacie, korzystając z lematów 2.1 oraz 2.6:

$$|\bigcup_{a<n/2} P_{a,n}| = \sum_{a<n/2} |P_{a,n}| \leqslant \sum_{a<n/2} n! \leqslant \frac{n+1}{2} \cdot n! = \frac{(n+1)!}{2}. \quad \square$$

Wniosek 2.10 *Jeżeli na szachownicy o wymiarach $n \times n$ można poprawnie rozstawić n hetmanów, to program z rozdziału 1.2 analizuje nie więcej niż $(n+1)!$ rozstawień hetmanów.*

Dowód. Najpierw trzeba zauważyć (podobnie jak w lemacie 2.8), że wszystkie rozstawienia hetmanów analizowane przez rozważany program należą do zbioru $\bigcup_{a<n/2} \bigcup_{m=1}^{n} P_{a,m}$.

Dalej liczymy jak w poprzednim lemacie, korzystając z wniosku 2.2 oraz lematu 2.6:

$$|\bigcup_{a<n/2} \bigcup_{m=1}^{n} P_{a,m}| \leqslant \sum_{a<n/2} |\bigcup_{m=1}^{n} P_{a,m}| \leqslant \sum_{a<n/2} 2 \cdot n! \leqslant \frac{n+1}{2} \cdot 2 \cdot n! = (n+1)!. \quad \square$$

Zauważmy jeszcze, że ostatni wniosek pozwala na dość dobre oszacowanie złożoności programu szukającego poprawnego rozstawienia hetmanów. Co prawda, podaje oszacowanie liczby sprawdzanych rozmieszczeń, ale – jak wiemy – jest to także oszacowanie liczby wywołań funkcji `poprawne` (każde wywołanie tej funkcji sprawdza inne rozstawienie hetmanów). Oszacowanie jest dobre w tym sensie, że jest widoczna zależność między liczbą wykonanych pętli, a liczbą sprawdzonych rozmieszczeń. Tym nie mniej, samo szacowanie można wyraźnie poprawić.