Marcin Kubica
Treść zadania, Opracowanie

Paweł Parys
Program

Dostępna pamięć: 32 MB. *OI, Etap II, dzień pierwszy, 11.02.2009*

Konduktor

Bajtazar pracuje jako konduktor w ekspresie Bajtockich Kolei Państwowych (BKP) relacji Bajtogród-Bitowice. Rozpoczął się trzeci etap reformy BKP[1]*, w ramach którego zmieniany jest system wynagrodzeń tak, aby lepiej motywował pracowników do efektywnej pracy. Wynagrodzenie Bajtazara będzie teraz zależało od liczby sprawdzonych przez niego biletów (pasażerów). Bajtazar jest w stanie sprawdzić bilety wszystkich pasażerów w trakcie przejazdu między dwiema kolejnymi stacjami. Nie ma jednak ani siły, ani chęci, żeby sprawdzać bilety między każdymi dwiema kolejnymi stacjami. Po namyśle zdecydował się, że będzie sprawdzał bilety k razy na trasie pociągu.*

Przed wyruszeniem w trasę Bajtazar otrzymuje zbiorcze zestawienie, ilu pasażerów będzie jechało z danej stacji do danej innej stacji. Na podstawie tych danych Bajtazar chciałby tak dobrać momenty kontroli biletów, aby sprawdzić jak najwięcej biletów. Niestety, powtórne sprawdzenie biletu pasażera, który już raz został skontrolowany, nie wpływa na wynagrodzenie Bajtazara. Napisz program, który pomoże ustalić Bajtazarowi, kiedy powinien kontrolować bilety, aby jego zarobki były jak największe.

Wejście

W pierwszym wierszu standardowego wejścia zapisane są dwie dodatnie liczby całkowite n i k ($1 \leqslant k < n \leqslant 600$, $k \leqslant 50$), oddzielone pojedynczym odstępem i oznaczające liczbę stacji na trasie pociągu oraz liczbę kontroli biletów, które chce przeprowadzić Bajtazar. Stacje są ponumerowane kolejno od 1 do n.

W kolejnych $n-1$ wierszach jest zapisane zbiorcze zestawienie pasażerów. Wiersz $(i+1)$-szy zawiera informacje o pasażerach, którzy wsiądą na stacji i — jest to ciąg $n-i$ nieujemnych liczb całkowitych pooddzielanych pojedynczymi odstępami: $x_{i,i+1}, x_{i,i+2}, \ldots, x_{i,n}$. Liczba $x_{i,j}$ (dla $1 \leqslant i < j \leqslant n$) oznacza liczbę pasażerów, którzy wsiedli na stacji i i jadą do stacji j. Łączna liczba pasażerów (czyli suma wszystkich $x_{i,j}$) wynosi nie więcej niż $2\,000\,000\,000$.

Wyjście

Twój program powinien wypisać na standardowe wyjście (w jednym wierszu) rosnący ciąg k liczb całkowitych z przedziału od 1 do $n-1$, pooddzielanych pojedynczymi odstępami. Liczby te mają być numerami stacji, po ruszeniu z których Bajtazar powinien sprawdzić bilety pasażerów.

[1] Historia reform BKP i stacji Bitowice została opisana w zadaniach **Koleje** z III etapu XIV OI oraz **Stacja** z III etapu XV OI. Znajomość tych zadań nie jest jednak w najmniejszym stopniu potrzebna do rozwiązania niniejszego zadania.

Przykład

Dla danych wejściowych:

```
7 2
2 1 8 2 1 0
3 5 1 0 1
3 1 2 2
3 5 6
3 2
1
```

poprawnym wynikiem jest:

```
2 5
```

lub

```
3 5
```

W obu przypadkach Bajtazar skontroluje 42 bilety.

Rozwiązanie

Analiza problemu

Główną część danych wejściowych stanowi informacja o liczbie pasażerów, którzy wsiadają na danej stacji, a wysiadają na innej danej stacji. Na rys. 1.a przedstawiono te dane (z przykładu z treści zadania) w postaci tabeli — wiersz określa numer stacji, na której pasażerowie wsiadają, a kolumna określa numer stacji, na której wysiadają. Zastanówmy się, jak z takiej tabeli odczytać, ile biletów sprawdzi Bajtazar w trakcie jednej kontroli. Jeżeli narysujemy prostokąt o dolnym lewym rogu na przekątnej (w miejscu odpowiadającym momentowi kontroli), a górnym prawym rogu w górnym prawym rogu tabeli, to będzie to suma liczb wewnątrz takiego prostokąta. Na rys. 1.a przedstawiono prostokąt odpowiadający kontroli biletów między stacjami nr 2 i 3.

	2	3	4	5	6	7
1	2	1	8	2	1	0
2		3	5	1	0	1
3			3	1	2	2
4				3	5	6
5					3	2
6						1

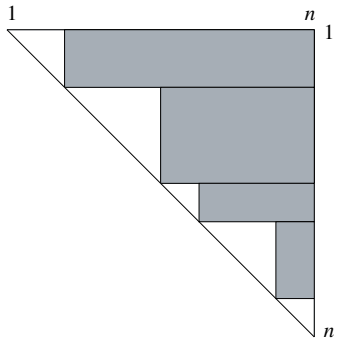
a)

	2	3	4	5	6	7
1	2	1	8	2	1	0
2		3	5	1	0	1
3			3	1	2	2
4				3	5	6
5					3	2
6						1

b)

Rys. 1: Przedstawienie danych o pasażerach w postaci tabeli. Prostokąt oznacza pasażerów, których bilety zostaną skontrolowane po tym, jak pociąg ruszy ze stacji nr 2 (a) oraz 2 i 5 (b).

Możemy więc sformułować nasze zadanie w następujący sposób: znaleźć takie k prostokątów (o dolnym lewym rogu na przekątnej i górnym prawym rogu w górnym prawym rogu tabeli), że pokrywają one liczby o maksymalnej sumie.

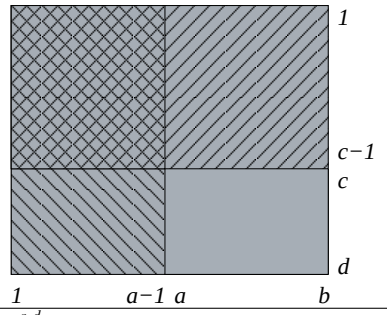


Rys. 2: Prostokąty obejmujące wszystkie sprawdzane bilety.

Jedno z rozwiązań przykładu z treści zadania, w którym Bajtazar sprawdza bilety między stacjami 2 i 3 oraz 5 i 6, jest przedstawione na rys. 1.b. Zwróćmy jednak uwagę, że liczby znajdujące się w obu prostokątach reprezentujących te dwie kontrole biletów powinny być liczone tylko raz. Byłoby najwygodniej, gdybyśmy rozważając konkretne momenty k kontroli biletów, sumowali liczby w prostokątach, które nie zachodzą na siebie. Możemy to zrobić, na przykład, w taki sposób, żeby prostokąty odpowiadające późniejszym kontrolom biletów nie zachodziły na prostokąty odpowiadające wcześniejszym kontrolom, jak pokazano na rys. 2.

Szybkie wyznaczanie sumy liczb w prostokącie

Widać już, że rozwiązując nasze zadanie, będziemy często obliczać sumy liczb zawartych w różnych prostokątach. Jest to dobry moment, żeby przypomnieć standardową technikę pozwalającą wyznaczać takie sumy w czasie stałym, po uprzednim wstępnym przetworzeniu tabeli z danymi o pasażerach (w czasie $O(n^2)$). Technika ta stanowiła sedno zadania *Mapa gęstości* z VIII OI [8] i pojawiała się już w innych zadaniach olimpijskich, np. w *Kupnie gruntu* z XV OI [15].



Rys. 3: Aby obliczyć $K_{a,b}^{c,d}$ (szary niezakreskowany prostokąt), musimy od $S[b,d]$ (cały prostokąt) odjąć $S[a-1,d]$ oraz $S[b,c-1]$ (obszary zakreskowane ukośnie), a do tego wszystkiego dodać, uprzednio odjęte dwukrotnie, $S[a-1,c-1]$ (podwójnie zakreskowany prostokąt).

Przypomnijmy, że $x_{i,j}$ dla $1 \leqslant i < j \leqslant n$ oznacza liczbę pasażerów podróżujących od stacji i do j, a dla $1 \leqslant j \leqslant i \leqslant n$ przyjmijmy $x_{i,j} = 0$. Oznaczmy przez $K_{a,b}^{c,d}$ sumę liczb $x_{i,j}$ zawartych w prostokącie, którego górny lewy róg ma współrzędne (a,c), a dolny prawy (b,d):

$$K_{a,b}^{c,d} = \sum_{i=a}^{b} \sum_{j=c}^{d} x_{i,j}.$$

Przyjmijmy oznaczenie $S[i,j] = K_{1,i}^{1,j}$ i załóżmy, że znamy wartości $S[i,j]$ dla wszystkich $0 \leqslant i,j \leqslant n$ (zakładamy, że $S[i,0] = S[0,j] = 0$). Chcąc wyznaczyć wartość $K_{a,b}^{c,d}$, korzystamy ze wzoru:

$$K_{a,b}^{c,d} = S[b,d] - S[a-1,d] - S[b,c-1] + S[a-1,c-1].$$

Ideę tej techniki przedstawia rysunek 3. Aby wyznaczyć wartości $S[i,j]$, wystarczy przejrzeć kolejne wiersze tabeli od góry do dołu (a w obrębie wiersza poruszać się od lewej do prawej) i skorzystać z tożsamości:

$$S[i,j] = x_{i,j} + S[i-1,j] + S[i,j-1] - S[i-1,j-1].$$

Rozwiązanie wzorcowe

Wiemy już, jak efektywnie wyznaczyć liczbę sprawdzanych biletów dla konkretnych k momentów kontroli biletów. Pozostaje problem, w jaki sposób rozpatrzyć wszystkie takie konfiguracje kontroli i wybrać z nich tę najkorzystniejszą. Możemy tu zastosować programowanie dynamiczne.

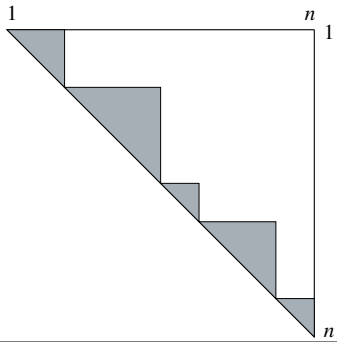
Tak jak w przypadku każdego zastosowania programowania dynamicznego, zanim rozwiążemy jeden konkretny problem, musimy go sparametryzować i rozwiązać wiele podobnych problemów. W tym przypadku, dla każdego $1 \leqslant l \leqslant n$ oraz $0 \leqslant h \leqslant k$ wyznaczymy maksymalną liczbę biletów należących do pasażerów, którzy wsiedli na stacji l lub później, które można sprawdzić, wykonując h kontroli. Oznaczmy tę liczbę biletów przez $T[l,h]$. Oczywiście $T[1,k]$ jest maksymalną liczbą biletów, jakie mogą zostać sprawdzone.

W przypadkach brzegowych, dla $h = 0$ (tzn. gdy bilety nie są kontrolowane) lub $n - l < h$ (tzn. gdy nie da się wykonać wszystkich kontroli), przyjmujemy $T[l,h] = 0$. W pozostałych przypadkach możemy skorzystać z następującej zależności rekurencyjnej:

$$T[l,h] = \max_{i=l,\ldots,n-h} \left(T[i+1,h-1] + K_{l,i}^{i+1,n} \right). \tag{1}$$

W powyższym wzorze i oznacza numer stacji, po której następuje pierwsza kontrola (wśród stacji o numerach nie mniejszych niż l). Obliczając $T[l,h]$, korzystamy wyłącznie z wartości $T[i,j]$ dla $i > l$ oraz $j < h$. Wystarczy więc obliczać wartości $T[l,h]$ w kolejności rosnących h i malejących l. Przypomnijmy, że wartości $K_{l,i}^{i+1,n}$ możemy wyznaczać w czasie stałym. Tak więc obliczenie maksimum w powyższym wzorze zajmuje czas $O(n)$. Mamy $O(n \cdot k)$ wartości $T[l,h]$ do obliczenia, tak więc wyznaczenie wszystkich wartości $T[l,h]$ wymaga czasu rzędu $O(n^2 \cdot k)$. Złożoność pamięciowa jest rzędu $O(n^2)$.

Jednak tym, czego szukamy, nie jest $T[1,k]$, ale numery stacji, po których mają nastąpić kontrole biletów. W tym celu możemy, wyznaczając wartości $T[l,h]$, zapamiętywać, dla



Rys. 4: Zaznaczone obszary trójkątne odpowiadają pasażerom, których bilety nie zostaną sprawdzone. Momenty kontroli są dokładnie takie same jak na rys. 2.

jakiego i osiągnęliśmy maksimum. Pozwoli nam to, po obliczeniu wszystkich $T[l,h]$, odtworzyć numery stacji, po których następują kontrole, w czasie $O(k)$.

Istnieje wiele analogicznych rozwiązań opartych na programowaniu dynamicznym — wszystkie one działają w tej samej złożoności czasowej i pamięciowej co opisane wcześniej rozwiązanie. Przykładowo, można inaczej wykonać pierwszą fazę rozwiązania wzorcowego, używając innego algorytmu wyznaczania wartości $S[i,j]$, albo też stosując mniej ogólne wstępne obliczenia — jako że w rozwiązaniu wzorcowym wykorzystywane są tylko wartości $K_{a,b}^{c,d}$ dla $d = n$, patrz wzór (1).

Można też, zamiast maksymalizować liczbę pasażerów, których bilety zostaną sprawdzone, minimalizować liczbę pasażerów, których bilety nie zostaną sprawdzone. Wówczas wystarczy skupić się na pasażerach, którzy zdążą wsiąść i wysiąść między dwiema kolejnymi kontrolami. Liczba takich pasażerów odpowiada sumie liczb w odpowiednim trójkącie, tak jak to zaznaczono na rys. 4. Przy tym, trójkąty te nie zachodzą na siebie, wystarczy więc sumować wyniki wyznaczone dla poszczególnych trójkątów.

Podobnie jak w poprzednim rozwiązaniu, zaczynamy od wstępnego przetworzenia danych, tak aby móc w czasie stałym wyznaczać sumy elementów w trójkątach (takich, jak na rys. 4). Oznaczmy przez $A[i,j]$ (dla $1 \leqslant i < j \leqslant n$) liczbę pasażerów, którzy wsiedli najwcześniej na stacji i, a wysiedli najpóźniej na stacji j. Łatwo wówczas zauważyć, że $A[i,j] = K_{i,j}^{i,j}$. Przyjmijmy, że $A[i,j] = 0$ dla $i \geqslant j$.

W dalszej części ponownie stosujemy programowanie dynamiczne. Dla każdej liczby kontroli h (dla $0 \leqslant h \leqslant k$) oraz dla każdej stacji l obliczamy minimalną liczbę pasażerów, którzy podróżują nie dalej niż do stacji l, a których bilety nie zostaną sprawdzone, przy założeniu, że na odcinku od stacji 1 do l jest h kontroli. Oznaczmy tę liczbę przez $B[l,h]$. Korzystamy tutaj z następujących wzorów:

$$\begin{aligned} B[l,0] &= A[1,l] \\ B[l,h] &= \min_{i=h,\ldots,l-1} (B[i,h-1] + A[i+1,l]) \text{ dla } l > h. \end{aligned}$$

W powyższym wzorze i oznacza numer stacji, po której następuje ostatnia kontrola (na odcinku do l-tej stacji). Wartości $B[l,h]$ obliczamy dla kolejnych, coraz większych wartości l oraz h (wynikiem końcowym jest $B[n,k]$). Obliczenie jednego minimum zajmuje czas $O(n)$, tak więc obliczenie wszystkich wartości $B[l,h]$ zajmuje czas rzędu $O(n^2 \cdot k)$. Aby móc

odtworzyć numery stacji, po których następują kontrole, możemy, podobnie jak poprzednio, zapamiętywać, dla jakich i otrzymaliśmy minimum. Tak więc, złożoności czasowa i pamięciowa są takie same jak w przypadku pierwszego rozwiązania. Rozwiązanie takie zostało zaimplementowane w pliku `kon1.cpp`, a z pewnymi drobnymi ulepszeniami również w plikach `kon.cpp`, `kon2.pas` i `kon4.java`.

Inne rozwiązania

Jeżeli nie zastosujemy wstępnego przetwarzania czy obliczania sum częściowych, które pozwolą na wyznaczanie wartości, z których bierzemy maksimum (lub minimum), w czasie stałym, to złożoność czasowa ulegnie pogorszeniu. Obliczenie jednej wartości $T[l,h]$ czy $B[l,h]$ będzie wymagać czasu rzędu $O(n^3)$. Takie rozwiązanie zostało zaimplementowane w pliku `kons6.cpp`. Możliwe jest też obliczanie sum częściowych, które przyspieszą rozwiązanie, ale tylko o czynnik n, dając złożoność czasową rzędu $O(n^3 \cdot k)$. Rozwiązania takie zostały zaimplementowane w plikach `kons5.cpp` i `kons7.cpp`.

Jeżeli nie zastosujemy w ogóle techniki programowania dynamicznego i będziemy przeglądać wszystkie możliwe kombinacje kontroli biletów, to uzyskamy rozwiązanie o złożoności wykładniczej. Zostało ono zaimplementowane w pliku `kons8.cpp`.

Testy

Większość testów została wygenerowana losowo. Jedynie trzy testy są inne. W teście 8b mamy $k = n-1$, czyli między każdymi dwiema stacjami muszą być sprawdzane bilety. W teście 9b wszyscy pasażerowie podróżują na dużej odległości, a w teście 10b każdy jedzie tylko do następnej stacji. Poniższa tabelka podsumowuje parametry użytych testów.

Nazwa	n	k	Opis
kon1.in	20	10	test losowy
kon2.in	50	25	test losowy
kon3.in	100	30	test losowy
kon4.in	250	50	test losowy
kon5.in	500	50	test losowy
kon6.in	600	40	test losowy
kon7.in	600	45	test losowy
kon8a.in	600	50	test losowy
kon8b.in	51	50	kontrola biletów między każdymi dwiema stacjami
kon9a.in	600	50	test losowy
kon9b.in	600	50	długie trasy przejazdu pasażerów
kon10a.in	600	50	test losowy
kon10b.in	600	50	przejazdy do następnej stacji