Zadania na konkurs

Piotr Zieliński

24 maja 2001 roku

Zadanie A – Garbage Collection

Wejście: `A.IN`
Wyjście: `A.OUT`
Program: `A.PAS`, `A.C` lub `A.CPP`

Większość wysokopoziomowych, nowoczesnych języków programowania (takich jak Java, C#, ...) zawiera zaawansowane mechanizmy przydzielania i zwalniania pamięci. Ich wspólną cechą jest to, że choć programista ciągle musi się martwić o przydzielenie pamięci (czyli de facto stworzenie obiektu) przez wywołanie instrukcji **new** lub podobnej, to instrukcja **delete** zniknęła. Programista nie musi jawnie usuwać danego obiektu z pamięci – wystarczy, że system stwierdzi, że do danego obiektu już nic się nie odwołuje – wtedy obiekt jest automatycznie usuwany.

Dla potrzeb niniejszego zadania zdefiniujmy nowy język *Pointer*. Jedynym dostępnym w tym języku typem danych jest *Obiekt*. Każdy obiekt ma taką samą strukturę i zawiera tylko i wyłącznie jeden wskaźnik *ptr* na inny obiekt (być może *null*). Jeśli chodzi o zmienne to mamy do dyspozycji tylko m predefiniowanych rejestrów wskaźnikowych (ponumerowanych od 1 do m) zawierających początkowo *null*. Dostępne są następujące operacje:

- **new** *reg1 reg2* — Tworzy nowy obiekt, który wskazuje na obiekt wskazywany przez rejestr o numerze *reg2* i umieszcza wskaźnik do niego w rejestrze o numerze *reg1*. Jeśli *reg2* jest równe 0, to nowo utworzony obiekt wskazuje na *null*.

- **let** *reg1 reg2* — Przepisuje zawartość rejestru o numerze *reg2* do rejestru o numerze *reg1*. Jeśli *reg2* jest równe 0, to do *reg1* wpisywany jest *null*

- **load** *reg1 reg2* — Przepisuje wartość wskaźnika *ptr* obiektu wskazywanego przez rejestr o numerze *reg2* do rejestru o numerze *reg1*. Zawartością rejestru o numerze *reg2* nie może być *null*.

- **save** *reg1 reg2* — Przepisuje wartość rejestru *reg2* do wskaźnika *ptr* obiektu wskazywanego przez rejestr o numerze *reg1*. Zawartością rejestru o numerze *reg1* nie może być *null*.

Poniższa tabelka obrazuje „odpowiedniki” w C oraz Pascalu:

Pointer	C/C++	Pascal
new *reg1 reg2*	`reg1 = new;` `reg->ptr = reg2;`	`new(reg1);` `reg1^.ptr := reg2;`
let *reg1 reg2*	`reg1 = reg2;`	`reg1 := reg2;`
load *reg1 reg2*	`reg1 = reg2->ptr;`	`reg1 := reg2^.ptr;`
save *reg1 reg2*	`reg1->ptr = reg2;`	`reg1^.ptr := reg2;`

Twoim zadaniem jest napisanie programu, który na podstawie tekstu programu w języku *Pointer* wyznaczy faktyczną liczbę obiektów w każdym momencie działania programu (zakładamy, że obiekt jest usuwany natychmiast w momencie, gdy nie można się do niego dostać przez rejestry wskaźnikowe bezpośrednio lub pośrednio).

Wejście. W pierwszej linii pliku wejściowego znajduje się liczba naturalna d ($1 \leqslant d \leqslant 10$), określająca liczbę zestawów danych, których opisy umieszczone są kolejno po sobie w następnych liniach pliku. Opis pojedynczego zestawu jest następujący:

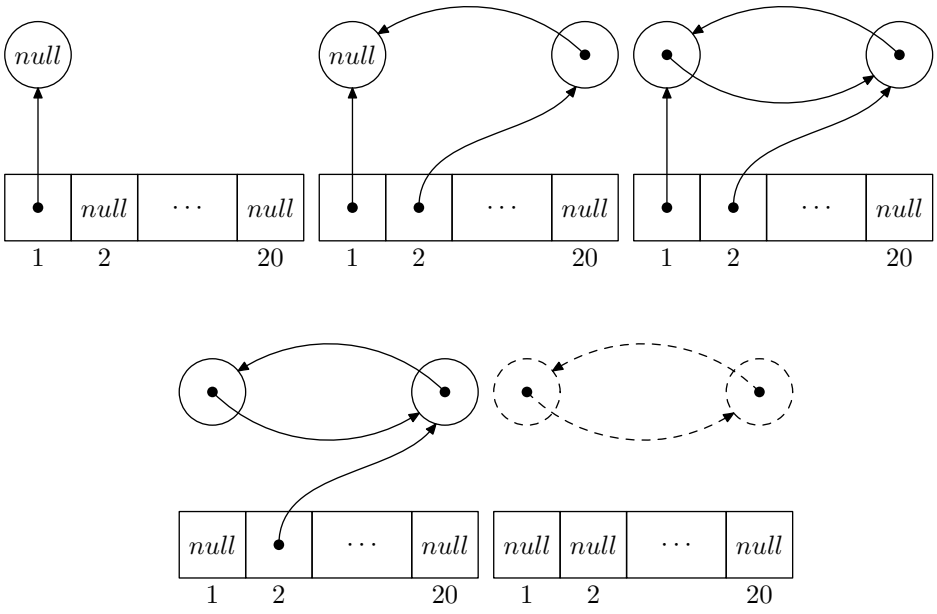
W pierwszej linii znajdują się dwie liczby naturalne n i m ($1 \leqslant n \leqslant 10000$, $1 \leqslant m \leqslant 100$), z których pierwsza określa liczbę instrukcji programu, a druga liczbę dostępnych rejestrów. W kolejnych n liniach znajdują się instrukcje programu (po jednej na linię).

Wyjście. Każdemu zestawowi w pliku wejściowym powinna odpowiadać n linii pliku wyjściowego. W i-tej linii k-tego bloku powinna znajdować się liczba całkowita określająca liczbę obiektów przechowywanych w pamięci po wykonaniu pierwszych i instrukcji programu z k-tego zestawu. Bloki w pliku wyjściowym powinny być oddzielone pustą linią.

Przykład

Wejście – `A.IN`	Wyjście – `A.OUT`
`3`	`1`
`2 2`	`0`
`new 1 0`	
`let 1 0`	`1`
`3 10`	`2`
`new 1 0`	`1`
`new 2 1`	
`load 2 1`	`1`
`5 20`	`2`
`new 1 0`	`2`
`new 2 1`	`2`
`save 1 2`	`0`
`let 1 0`	
`let 2 0`	

Kolejne stany rejestrów i obiektów z ostatniego przykładu przedstawiają poniższe rysunki:



Zadanie B – Obrazy i pokoje

Wejście: `B.IN`
Wyjście: `B.OUT`
Program: `B.PAS`, `B.C` lub `B.CPP`

Mamy N pokojów ponumerowanych liczbami 1, 2, ..., N. W każdym pokoju znajduje się obraz, na którym jest namalowana pewna liczba ze zbioru $\{1, 2, \ldots, N\}$. Ponadto w każdym pokoju znajduje się jeden człowiek.

Co minutę rozlega się dzwonek (słyszany jednocześnie w każdym pokoju) i na jego dźwięk każdy człowiek czyta liczbę z obrazu, który wisi w pokoju, w którym się znajduje i teleportuje się do pokoju oznaczonego tym numerem.

Chociaż na początku w każdym pokoju jest jedna osoba, to już po pierwszej iteracji tak wcale być nie musi – niektóre pokoje mogą być puste. W ogólności liczba zajętych pokojów może się zmieniać z minuty na minutę. Jednak, jak można udowodnić, w końcu liczba zajętych pokojów ustali się i nigdy więcej się już nie zmieni, mimo faktu, iż ludzie ciągle się będą przemieszczać.

Twoim zadaniem jest, na podstawie liczby pokojów N oraz liczb wymalowanych w poszczególnych pokojach, stwierdzić jaka będzie ta ostateczna liczba zajętych pokojów.

Wejście. W pierwszej linii pliku wejściowego znajduje się liczba naturalna d ($1 \leqslant d \leqslant 10$), określająca liczbę zestawów danych, których opisy umieszczone są kolejno po sobie w następnych liniach pliku. Opis pojedynczego zestawu jest następujący:

W pierwszej linii znajduje się liczba naturalna n ($2 \leqslant n \leqslant 10000$) określająca liczbę pokojów. W drugiej linii znajduje się n liczb naturalnych a_1, a_2, ..., a_n ($a_i \in \{1, 2, \ldots, n\}$) określających liczby wypisane w kolejnych pokojach.

Wyjście. Każdemu zestawowi w pliku wejściowym powinna odpowiadać jedna linia pliku wyjściowego. Ta linia powinna zawierać jedną liczbę naturalną określającą ostateczną liczbę zajętych pokojów.

Przykład

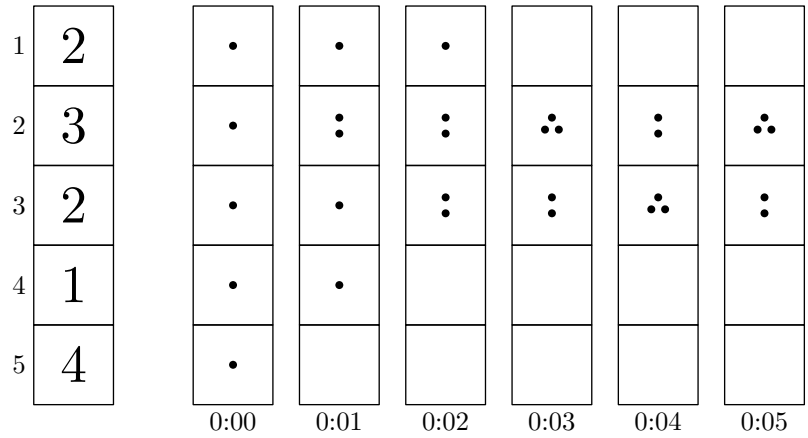
Wejście – `B.IN`

```
3
5
1 1 1 2 2
5
5 1 2 4 3
5
2 3 2 1 4
```

Wyjście – `B.OUT`

```
1
5
2
```

Pierwsze sześć minut symulacji dla ostatniego zestawu (`2 3 2 1 4`) zostało przedstawione na rysunku poniżej:



W pierwszej kolumnie przedstawione są liczby wymalowane w kolejnych pokojach. W kolejnych kolumnach przedstawiony jest stan pokojów w kolejnych „turach” – jedna kropka reprezentuje jednego człowieka. W dalszych turach (nie pokazanych na rysunku) liczba zajętych pokojów nie ulega zmianie – ludzie z pokojów nr 2 i 3 zamieniają się miejscami co minutę. Odpowiedzią jest więc 2.

Zadanie C – Wybieranie reprezentantów

Wejście: `C.IN`
Wyjście: `C.OUT`
Program: `C.PAS`, `C.C` lub `C.CPP`

Dwie równoliczne (tj. mające tyle samo członków) partie polityczne spotkały się w celu przeprowadzenia debaty. Szybko jednak zorientowano się, że gdy wszyscy mówią, to nikt nic nie rozumie. Postanowiono zatem wybrać reprezentanów, jednego z każdej partii.

Wybory przeprowadzono następująco. Na początku wszyscy usiedli w kółku, twarzami do środka. Jedna z osób (osoba nr 1) otrzymała pałeczkę. Teraz gra toczy się w rundach. W każdej rundzie osoba, która ma pałeczkę przekazuje ją pierwszej osobie na lewo (czyli od góry patrząc w kierunku ruchu wskazówek zegara), która należy do przeciwnej partii, po czym sama usuwa się z kółka. Pałeczka krąży tak długo, aż zostaną tylko dwie osoby (reprezentujące naturalnie przeciwne partie) i one toczą debatę.

Twoim zadaniem jest napisanie programu, który na podstawie liczby osób n w każdej partii oraz tego, gdzie usiadły w kółku, wyliczy, które dwie osoby będą toczyły debatę.

Wejście. W pierwszej linii pliku wejściowego znajduje się liczba naturalna d ($1 \leqslant d \leqslant 10$), określająca liczbę zestawów danych, których opisy umieszczone są kolejno po sobie w następnych liniach pliku. Opis pojedynczego zestawu jest następujący:

W pierwszej linii znajduje się liczba naturalna n ($2 \leqslant n \leqslant 3000$) określająca liczbę osób w każdej z partii. W kolejnej linii znajduje się ciąg $2n$ liczb ze zbioru $\{1, 2\}$, oddzielonych odstępami, określających przynależność partyjną kolejnych osób (licząc od osoby nr 1 w kierunku ruchu wskazówek zegara).

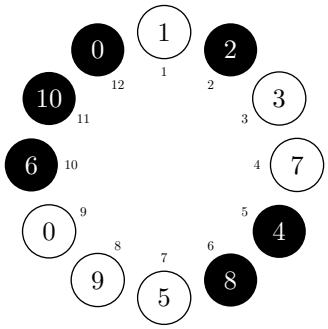
Wyjście. Każdemu zestawowi w pliku wejściowym powinna odpowiadać jedna linia pliku wyjściowego. Ta linia powinna zawierać dwie liczby n_1 i n_2 ($n_1, n_2 \in \{1, 2, \ldots, 2n\}$) określające numer osoby wybranej na reprezentanta odpowiednio pierwszej i drugiej partii. Numer osoby to początkowa pozycja w kółku licząc od rozpoczynającego w kierunku ruchu wskazówek zegara.

Przykład

Wejście – `C.IN`	Wyjście – `C.OUT`

`3`
`2`
`1 2 2 1`
`4`
`1 1 1 1 2 2 2 2`
`6`
`1 2 1 1 2 2 1 1 1 2 2 2`
	`4 3`
	`4 8`
	`9 12`

Ostatni zestaw został przedstawiony na poniższym rysunku:



Białe kółka reprezentują członków pierwszej partii, natomiast czarne – drugiej. Liczby w kółkach oznaczają kolejność odpadania (tj. najpierw odpada z jedynką, później z dwójką, ...). Liczba 0 oznacza, że dana osoba nie odpadnie i będzie toczyła debatę. Poprawną odpowiedzią jest zatem `9 12`.

Zadanie D – Prawoskrętna mrówka

Wejście: `D.IN`
Wyjście: `D.OUT`
Program: `D.PAS`, `D.C` lub `D.CPP`

Po prostokątnej szachownicy porusza się mrówka. Startuje z lewego górnego rogu i jest skierowana poziomo w prawo. Porusza się stosując bardzo prosty algorytm. Po pierwsze, patrzy na jakie pole za chwilę wejdzie (jest to następne pole w kierunku jej ruchu). Jeśli to pole jest wolne, leży na szachownicy oraz nigdy na tym polu jeszcze nie była to wchodzi na nie. Jeśli natomiast choć jeden z warunków z poprzedniego zdania nie jest spełniony, to obraca się o 90° w prawo. Mrówka kończy podróż jeśli wykona dwa obroty o 90° stojąc na tym samym polu.

Twoim zadaniem jest napisać program, który dla danej szachownicy (danej przez liczbę kolumn X oraz liczbę wierszy Y oraz przez współrzędne zajętych pól) wyliczy współrzędne pola, na którym mrówka zakończy swą podróż. Każde pole ma dwie współrzędne: numer kolumny (x) oraz numer wiersza (y). Pole leżące w lewym górnym rogu szachownicy ma współrzędne $(1, 1)$.

Wejście. W pierwszej linii pliku wejściowego znajduje się liczba naturalna d ($1 \leqslant d \leqslant 10$), określająca liczbę zestawów danych, których opisy umieszczone są kolejno po sobie w następnych liniach pliku. Opis pojedynczego zestawu jest następujący:

W pierwszej linii znajdują się trzy liczby całkowite: X, Y i n ($1 \leqslant X, Y \leqslant 5 \cdot 10^8, 1 \leqslant n \leqslant 100$). X i Y określają odpowiednio liczbę kolumn i wierszy szachownicy, natomiast n jest liczbą zajętych pól szachownicy. W następnej linii znajduje się $2n$ liczb całkowitych $x_1, y_1, \ldots, x_n, y_n$ ($1 \leqslant x_i \leqslant X, 1 \leqslant y_i \leqslant Y$) oznaczających współrzędne zajętych pól. Gwarantuje się, że pole (1,1) jest wolne.

Wyjście. Każdemu zestawowi w pliku wejściowym powinna odpowiadać jedna linia pliku wyjściowego. Ta linia powinna zawierać dwie liczby całkowite x_e i y_e określające współrzędne pola, w którym mrówka zakończy swą wędrówkę.

Przykład

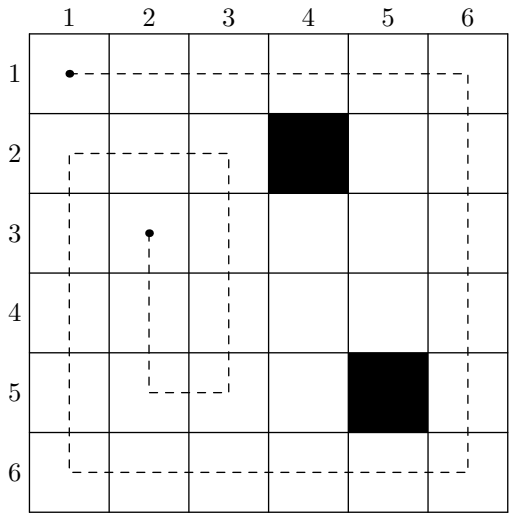
Wejście – `D.IN`

```
3
3 3 2
1 2 3 3
4 4 2
1 2 4 4
6 6 2
5 5 4 2
```

Wyjście – `D.OUT`

```
2 2
1 3
2 3
```

Sytuacja z ostatniego zestawu została przedstawiona na poniższym rysunku:



Zadanie E – Odwracanie kartek

Wejście: `E.IN`
Wyjście: `E.OUT`
Program: `E.PAS`, `E.C` lub `E.CPP`

Jadąc tramwajem nierzadko czytamy jakieś materiały. Niestety często zdarza się, że są one na pojedynczych kartkach. Sprawia to masę problemów. Pominąwszy fakt, że kartki mogą być nie po kolei, to w dodatku niektóre mogą być obrócone o 180° w stosunku do prawidłowej pozycji. Zakładamy, że wszystkie kartki są zadrukowane jednostronnie oraz że wszystkie są ułożone tak, że strona zadrukowana jest z góry.

Zestaw taki można czytać w następujący sposób. Po pierwsze, jeśli początkowa strona jest źle obrócona, to obracamy cały zestaw kartek o 180°. Później w każdym kroku wykonujemy następujące czynności:

- czytamy kartkę, która jest na wierzchu,
- podnosimy ją nad resztę
- pozostały zestaw (złożony z wszystkich kartek oprócz tej, którą podnieśliśmy) obracamy tak, by wierzchnia kartka tego zestawu była prawidłowo obrócona
- bez żadnych obrotów wkładamy kartkę, którą podnieśliśmy, na spód.

Czynności te można wykonywać w nieskończoność (zaniedbujemy fakt powtórnego czytania tych samych kartek). Powstaje naturalne pytanie, czy w którymś momencie wszystkie kartki będą poprawnie obrócone i jeśli tak, to po której rundzie. Runda polegająca na ewentualnym obróceniu całego zestawu ma numer 1, następne zaś 2, 3, ...

Twoim zadaniem jest napisanie programu, który dla danej liczby kartek n oraz początkowego ich ułożeń $a_i \in \{0, 1\}$ ($a_i = 0$ oznacza kartkę nieobróconą, $a_i = 1$ – obróconą o 180°) stwierdzi czy kiedykolwiek wszystkie kartki będą poprawnie obrócone, a jeśli tak to po której rundzie nastąpi to po raz pierwszy (jeśli wszystkie na początku są dobrze ułożone, tj. $a_1 = a_2 = \ldots = a_n = 0$, odpowiedzią jest 0).

Wejście. W pierwszej linii pliku wejściowego znajduje się liczba naturalna d ($1 \leqslant d \leqslant 10$), określająca liczbę zestawów danych, których opisy umieszczone są kolejno po sobie w następnych liniach pliku. Opis pojedynczego zestawu jest następujący:

W pierwszej linii znajduje się jedna liczba naturalna n ($1 \leqslant n \leqslant 10000$) określająca liczbę kartek. W kolejnej znajduje się ciąg n liczb a_1, a_2, ..., a_n ($a_i \in \{0, 1\}$) określających początkowe obrócenie kolejnych kartek zgodnie z regułą podaną w treści zadania.

Wyjście. Każdemu zestawowi w pliku wejściowym powinna odpowiadać jedna linia pliku wyjściowego. Ta linia powinna zawierać albo liczbę naturalną r określającą najmniejszy numer rundy, po której wszystkie kartki będą poprawnie obrócone (tzn. nieobrócone) lub słowo `NIGDY` jeśli taka sytuacja nigdy nie nastąpi.

Przykład

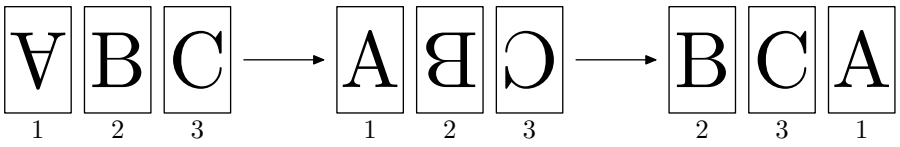
Wejście – `E.IN`

```
3
3
0 0 0
3
0 1 0
3
1 0 0
```

Wyjście – `E.OUT`

```
0
NIGDY
2
```

Pierwsze dwie rundy z sytuacji z ostatniego zestawu są przedstawione na poniższym rysunku:



Zadanie F – Kto zje więcej pizzy?

Wejście: `F.IN`
Wyjście: `F.OUT`
Program: `F.PAS`, `F.C` lub `F.CPP`

Często, gdy nie jesteśmy pewni własnych możliwości żołądkowych, zdarza się zamawiać jedną pizzę na dwie osoby. Pizza zostaje przywieziona i pojawia się następujący (często niewypowiedziany) problem – kto je który z ośmiu (z reguły) kawałków.

Ogólnie rzecz biorąc – przy założeniu, iż jedzący sięgają po kawałki na przemian – możliwe są dwa scenariusze. W obydwóch rozpoczynający wybiera dowolnych kawałek. W pierwszej metodzie każdy może brać dowolny (niezjedzony jeszcze, oczywiście) kawałek. Drugi wariant – bardziej tradycyjny – pozwala brać tylko te kawałki, które stykały się z tymi już wziętymi, dzięki czemu nie powstają nowe „dziury" w pizzy. Zakładamy, że przylegające kawałki mają kolejne numery (tj. i oraz $i+1$). Jedynym wyjątkiem są kawałki o numerach 1 i n (gdzie n to całkowita liczba kawałków), które też do siebie przylegają.

Każdej osobie zależy oczywiście na tym, aby zjeść jak najwięcej. Można udowodnić, że oba warianty gwarantują zaczynającemu wygraną (lub chociażby remis) jeśli będzie optymalnie wybierał kawałki do zjedzenia. Jednak zysk w obu przypadkach może być różny.

Twoim zadaniem jest napisanie programu, który dla danej liczby kawałków pizzy n oraz ich względnych wielkości $s_1, \ldots, s_n$ stwierdzi ile zje rozpoczynający w każdym z wariantów, jeśli zarówno on jak i jego oponent będą jedli optymalnie. Ilość zjedzonej pizzy jest sumą wielkości s_i zjedzonych kawałków.

Wejście. W pierwszej linii pliku wejściowego znajduje się liczba naturalna d ($1 \leqslant d \leqslant 10$), określająca liczbę zestawów danych, których opisy umieszczone są kolejno po sobie w następnych liniach pliku. Opis pojedynczego zestawu jest następujący:

W pierwszej linii znajduje się jedna parzysta liczba całkowita n ($2 \leqslant n \leqslant 100$) określająca liczbę kawałków, na które podzielona jest pizza. W drugiej linii znajduje się ciąg n liczb całkowitych s_i ($0 \leqslant s_i \leqslant 100$) określających wielkości kolejnych kawałków.

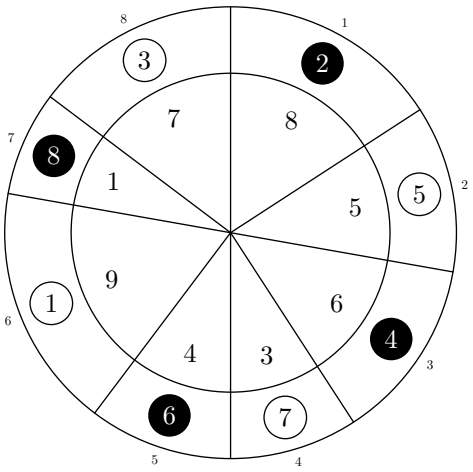
Wyjście. Każdemu zestawowi w pliku wejściowym powinna odpowiadać jedna linia pliku wyjściowego. Ta linia powinna zawierać dwie liczby całkowite a i b określające ilość pizzy jaką zje rozpoczynający odpowiednio w pierwszym i drugim wariancie.

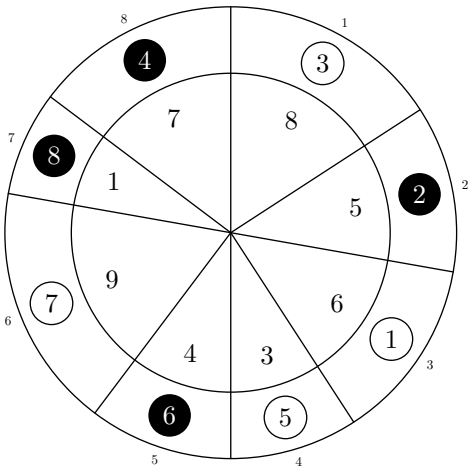
Przykład

Wejście – `F.IN`	Wyjście – `F.OUT`

`3`
`4`
`1 2 3 4`
`4`
`1 3 2 4`
`8`
`8 5 6 3 4 9 1 7`
	`6 6`
	`6 7`
	`24 26`

Sytuacja z ostatniego zestawu została przedstawiona na poniższych rysunkach:



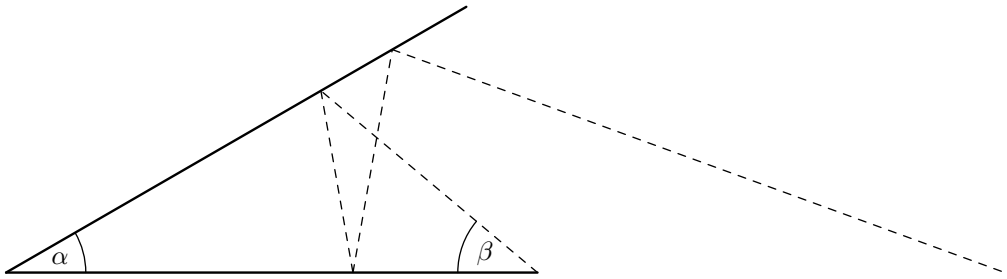


Lewy rysunek przedstawia pierwszy wariant, natomiast prawy – drugi. Liczby w wewnętrznych kołach oznaczają wielkości kawałków, natomiast liczby w kółkach oznaczają kolejność ich jedzenia. Białe kółka oznaczają kawałki zjedzone przez pierwszego zawodnika, natomiast czarne – przez drugiego.

Zadanie G – Kalibracja lasera

Wejście: `G.IN`
Wyjście: `G.OUT`
Program: `G.PAS`, `G.C` lub `G.CPP`

Aby przeprowadzić pomiary precyzji nowego lasera przeprowadza się następujący eksperyment. Na stole laboratoryjnym ustawia się pionowo dwa identyczne lustra, tak że tworzą one ze sobą kąt α. Zwierciadła stykają się końcami tworząc (patrząc od góry) kąt o równych ramionach. Dokładnie na końcu jednego takiego ramienia umieszczony jest laser, który emituje poziomy promień do „wnętrza" układu pod kątem β do zwierciadła, na którego końcu się znajduje. Całość układu (przykładowo dla $\alpha = 30°$ i $\beta = 35°$) jest przedstawiona na rysunku poniżej.



Liczba odbić promienia świetlnego zależy oczywiście od wartości kątów α i β. Okazuje się, że ta zależność, przy odpowiednio dobranych wartościach tych kątów może by wykorzystana do pomiaru precyzji lasera.

Twoim zadaniem jest napisanie programu, który dla danych kątów α i β wyliczy liczbę odbić promienia świetlnego od zwierciadeł. Dla uproszczenia zakładamy, że promień nie przechodzi (poza momentem startu) przez końcówkę żadnego zwierciadła.

Wejście. W pierwszej linii pliku wejściowego znajduje się liczba naturalna d ($1 \leqslant d \leqslant 10$), określająca liczbę zestawów danych, których opisy umieszczone są kolejno po sobie w następnych liniach pliku. Opis pojedynczego zestawu jest następujący:

W jedynej jego linii znajdują się dwie liczby rzeczywiste α i β ($0 < \alpha, \beta < 90$) oznaczające odpowiednio kąt pomiędzy zwierciadłami oraz kąt pomiędzy promieniem lasera a zwierciadłem. Liczby te podane są z dokładnością do co najwyżej szóstego miejsca po przecinku.

Wyjście. Każdemu zestawowi w pliku wejściowym powinna odpowiadać jedna linia pliku wyjściowego. Ta linia powinna zawierać pojedynczą nieujemną liczbę całkowitą określającą liczbę odbić promienia od zwierciadeł.

Przykład

Wejście – `G.IN`

```
3
30.000 35.000
45.000 10.000
0.300 17.000
```

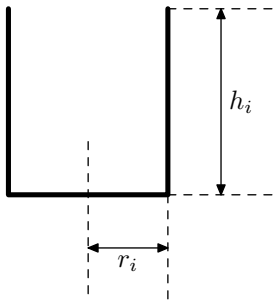
Wyjście – `G.OUT`

```
3
3
486
```

Zadanie H – Wieża z wiader

Wejście: `H.IN`
Wyjście: `H.OUT`
Program: `H.PAS`, `H.C` lub `H.CPP`

Przed nami stoi rząd wiader. Każde wiadro jest idealne – jest to powierzchnia walca bez górnej podstawy. Każdy walec jest więc w pełni scharakteryzowany przez swój promień r_i oraz wysokość h_i:



Nasze zadanie jest proste. Na początku bierzemy pierwsze wiadro i ustawiamy przed sobą. Później bierzemy drugie i ustawiamy je (na razie w powietrzu) tak, aby jego oś symetrii pokrywała się z osią symetrii pierwszego walca, po czym spuszczamy. Tak samo postępujemy z kolejnymi wiadrami.

Po skończeniu roboty przed nami stoi konstrukcja z wiader. Niektóre wiadra zmieściły się w innych, niektóre nie. Zakładamy, że jedno wiadro „wchodzi" w drugie, jeśli to pierwsze ma mniejszy promień. Zakładamy także, że wszystkie ścianki mają zerową grubość.

Twoim zadaniem jest napisanie programu, który dla danych wysokości i promieni kolejnych wiader wyliczy wysokość tak zbudowanej konstrukcji.

Wejście. W pierwszej linii pliku wejściowego znajduje się liczba naturalna d ($1 \leqslant d \leqslant 10$), określająca liczbę zestawów danych, których opisy umieszczone są kolejno po sobie w następnych liniach pliku. Opis pojedynczego zestawu jest następujący:

W pierwszej linii znajduje się liczba naturalna n ($1 \leqslant n \leqslant 10^6$) określająca liczbę wiader. W następnej linii znajduje się $2n$ liczb całkowitych: r_1, h_1, r_2, h_2, …, r_n, h_n ($1 \leqslant r_i \leqslant 10000$, $1 \leqslant h_i \leqslant 1000$). Liczba r_i określa promień i-tego wiadra, natomiast h_i – jego wysokość.

Wyjście. Każdemu zestawowi w pliku wejściowym powinna odpowiadać jedna linia pliku wyjściowego. Ta linia powinna zawierać pojedynczą liczbę całkowitą określającą wysokość konstrukcji zbudowanej z wiader podanych w zestawie.

Przykład

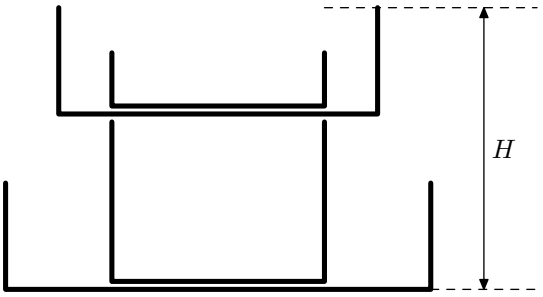
Wejście – `H.IN`

```
3
2
20 20 30 30
2
30 30 20 20
4
40 20 20 30 30 20 20 10
```

Wyjście – `H.OUT`

```
50
30
50
```

Układ wiader z ostatniego zestawu jest przedstawiony na poniższym rysunku:



Odstępy pionowe pomiędzy wiadrami zostały wprowadzone sztucznie, aby zwiększyć czytelność rysunku. W zadaniu ich nie uwzględniamy – przyjmujemy, że grubość materiału, z którego wykonano garnki to 0. Odpowiedzią jest wartość H z rysunku.