XII Międzynarodowa Olimpiada Informatyczna, Pekin 2000

XII Międzynarodowa Olimpiada Informatyczna — treści zadań

Palindrome

Zadanie

Palindrom to sumetryczny napis, tzn. taki napis, który czytany z lewa na prawo i z prawa na lewo jest taki sam. Napisz program, który dla zadanego napisu wyznaczy minimalną liczbę znaków, które należy do niego wstawić, aby stał się palindromem.
Np. napis `Ab3bd` *można zamienić na palindrom (*`dAb3bAd` *lub* `Adb3bdA`*) wstawiając do niego 2 znaki. Nie można jednak tego zrobić wstawiając mniej niż 2 znaki.*

Wejście

Plik wejściowy nazywa się `PALIN.IN`*. Pierwszy wiersz tego pliku zawiera jedną liczbę całkowitą: długość zadanego napisu* N*,* $3 \leq N \leq 5000$*. Drugi wiersz zawiera napis długości* N*. Napis ten składa się z wielkich liter od* `A` *do* `Z`*, małych liter od* `a` *do* `z` *oraz cyfr od* `0` *do* `9`*. Małe i wielkie litery są różnymi znakami.*

Wyjście

Plik wyjściowy nazwa się `PALIN.OUT`*. Pierwszy wiersz tego pliku powinien zawierać jedną liczbę całkowitą równą szukanej minimalnej liczbie wstawianych znaków.*

Przykład

`PALIN.IN:`

```
5
Ab3bd
```

`PALIN.OUT:`

```
2
```

Car Parking

Zadanie

Na centralnym parkingu pod Wielkim Murem znajduje się długi rząd miejsc parkingowych. Jeden z końców tego rzędu uważa się za lewy, a drugi za prawy. Cały rząd jest wypełniony parkującymi samochodami. Każdy samochód jest ustalonej marki, ale może być wiele samochodów tej samej marki. Marki identyfikujemy liczbami całkowitymi. Pracownicy parkingu postanowili ustawić auta w niemalejącej kolejności względem ich marek, od lewego do prawego końca. Do tego celu wykorzystają następującą metodę postępowania. W tak zwanej rundzie, każdy z pracowników może wyjechać jednym samochodem z miejsca jego postoju (pracownicy pracują jednocześnie), a następnie zaparkować na zwolnionym w tej samej rundzie miejscu postojowym. Może się zdarzyć, że w danej rundzie nie wszyscy pracują. Oczywiście, pracownicy chcieliby wykonać swoją pracę w jak najmniejszej liczbie rund.

Niech N *będzie liczbą samochodów, a* W *liczbą pracowników parkingu. Napisz program, który dla danej liczby marek samochodów na parkingu i danej liczby pracowników, znajduje sposób uporządkowania samochodów w co najwyżej* $\lceil \frac{N}{W-1} \rceil$ *rundach (zaokrąglenie w górę). Minimalna liczba rund nigdy nie jest większa niż* $\lceil \frac{N}{W-1} \rceil$.

Rozważmy następujący przykład. Na parkingu znajduje się 10 samochodów o markach 1, 2, 3 i 4, oraz 4 pracowników. Na poczatku kolejność samochodów na parkingu, od lewej do prawej i względem ich marek, jest taka: 2 3 3 4 4 2 1 1 3 1. Minimalna liczba rund wynosi 3, a rundy można wykonać w taki sposób, że rozmieszczenie samochodów po każdej z nich jest następujące:

- *2 1 1 4 4 2 3 3 3 1 — po rundzie 1,*
- *2 1 1 2 4 3 3 3 4 1 — po rundzie 2,*
- *1 1 1 2 2 3 3 3 4 4 — po rundzie 3.*

Wejście

Nazwą pliku wejściowego jest `CAR.IN`*. Pierwszy wiersz pliku wejściowego zawiera trzy liczby całkowite. Pierwszą liczbą jest* N *— liczba samochodów na parkingu,* $2 \leq N \leq 20000$*. Drugą liczbą jest liczba marek* M, $2 \leq M \leq 50$*. Marki są ponumerowane od 1 do* M*. Na parkingu znajduje sie co najmniej jeden samochód każdej marki. Trzecia liczba jest liczbą pracowników parkingu* W, $2 \leq W \leq M$*. Drugi wiersz zawiera* N *liczb całkowitych —* i*-ta liczba jest marką* i*-tego samochodu w rzędzie, patrząc od lewego do prawgo końca.*

Wyjście

Nazwą pliku wyjsciowego jest `CAR.OUT`*. Pierwszy wiersz pliku wyjściowego powinien zawierać jedną liczbę całkowitą* R *— liczbę rund w znalezionym rozwiązaniu. Kolejne* R *wierszy zawiera opisy kolejnych rund od 1 do* R*. Każdy taki wiersz rozpoczyna się od liczby całkowitej* C *— liczby samochodów, które należy ruszyć w tej rundzie. Następnie pojawia się* $2C$ *liczb całkowitych — identyfikatorów pozycji samochodów. Pozycje samochodów są ponumerowane od lewego do prawego końca, kolejnymi liczbami* $1, 2, \ldots N$*. Pierwsze dwie liczby tworzą parę opisującą ruch jednego z samochodów: pierwsza liczba jest pozycją tego samochodu (od lewego końca) sprzed opisywanej rundy, a druga liczba jest pozycją tego samochodu (od lewego końca) po tej rundzie. Trzecia i czwarta liczba tworzą parę opisującą ruch innego samochodu, itd. Dla każdego z tych* R *wierszy może być wiele różnych rozwiązań, ale Twój program powinien wypisać tylko jedno z nich.*

Przykład

`CAR.IN:`

```
10 4 4
2 3 3 4 4 2 1 1 3 1
```

`CAR.OUT:`

```
3
4 2 7 3 8 7 2 8 3
3 4 9 9 6 6 4
3 1 5 5 10 10 1
```

Częściowa punktacja

Przypuśćmy, że twój program wypisał liczę rund R*, a* $\left\lceil \frac{N}{W-1} \right\rceil$ *jest równe* Q*. Jeśli twój program wypisuje niepoprawne opisy rund lub w wyniku ich wykonania samochody nie zostaną uporządkowane, nie zdobywasz żadnych punktów. W przeciwnym przypadku otrzymasz liczbę punktów obliczoną w następujący sposób:*

- $R \leq Q$ *— 100% punktów*
- $R = Q + 1$ *— 50% punktów*
- $R = Q + 2$ *— 20% punktów*
- $R \geq Q + 3$ *— 0% punktów*

Median strength

Zadanie

W nowym eksperymencie kosmicznym wykorzystanych jest N obiektów ponumerowanych od 1 do N. Wiadomo, że N jest liczbą nieparzystą. Każdy obiekt charakteryzuje się inną, ale nieznaną wagą, wyrażaną liczbą naturalną. Dla każdej wag Y mamy $1 \leq Y \leq N$. Obiekt o środkowej wadze to taki obiekt, dla którego mamy tyle samo obiektów od niego cięższych, co lżejszych. Napisz program, który wyznaczy obiekt o środkowej wadze. Niestety jedyny sposób na porównywanie wagi obiektów, to użycie narzędzia, które dla zadanych trzech różnych obiektów wyznacza, który spośród tych trzech obiektów ma środkową wagę.

Biblioteka

Dana jest biblioteka `device` *udostępniająca trzy operacje:*

- `GetN`*, bez argumentów, którą należy wywołać raz na początku, dająca w wyniku wartość N,*
- `Med3`*, mająca 3 argumenty będące numerami trzech różnych obiektów, dająca w wyniku numer obiektu o środkowej wadze,*
- `Answer`*, o jednym argumencie będącym numerem obiektu; wywołując te funkcje przekazuje się numer obiektu o środkowej wadze, funkcja ta kończy (w poprawny sposób) wykonanie programu.*

Biblioteka `device` *tworzy dwa pliki tekstowe:* `MEDIAN.OUT` *i* `MEDIAN.LOG`*. Pierwszy wiersz pliku* `MEDIAN.OUT` *zawiera jedną liczbę całkowitą — numer obiektu, który był argumentem* `Answer`*. Drugi wiersz jedną liczbę całkowitą — liczbę wywołań funkcji* `Med3` *wykonywanych przez Twój program. Komunikacja pomiędzy Twoim programem i biblioteką jest opisana w pliku* `MEDIAN.LOG`*.*

Instrukcja dla programujących w Pascalu: *Wstaw do kodu źródłowego programu następujące polecenie dołączające bibliotekę:*

```
uses device;
```

Instrukcja dla programujących w C/C++: *W kodzie źródłowym Twojego programu użyj polecenia:*

```
#include <device.h>
```

stwórz projekt `MEDIAN.PRJ` *i dodaj do niego pliki* `MEDIAN.C` *(*`MEDIAN.CPP`*) oraz* `DEVICE.OBJ`*.*

Testowanie

Możesz przetestować swój program tworząc plik tekstowy `DEVICE.IN`*. Plik ten musi zawierać dwa wiersze. W pierwszym wierszu musi być zapisana jedna liczba całkowita: liczba obiektów N. Drugi wiersz musi zawierać liczby całkowite od 1 do N podane w pewnej kolejności — i-ta liczba reprezentuje wagę obiektu nr i.*

Przykład

`DEVICE.IN`:

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

Powyższy plik `DEVICE.IN` *opisuje 5 danych obiektów o następujących wagach:*

Numer	*1*	*2*	*3*	*4*	*5*
Waga	*2*	*5*	*4*	*3*	*1*

Oto poprawna sekwencja 5–ciu wywołań biblioteki:

1. `GetN` *(w Pascalu) lub* `GetN()` *(W C/C++), daje w wwyniku 5,*

2. `Med3(1,2,3)`, *daje w wyniki 3,*
3. `Med3(3,4,1)`, *daje w wyniki 4,*
4. `Med3(4,2,5)`, *daje w wyniki 4,*
5. `Answer(4)`

Ograniczenia

- *Liczba obiektów N jest nieparzysta oraz $5 \leq N \leq 1499$.*
- *Numery obiektów i spełniają $1 \leq i \leq N$.*
- *Wagi obiektów Y spełniają $1 \leq Y \leq N$. Wszystkie wagi są różne,*
- *Biblioteka dla Pascala znajduje się w pliku* `device.tpu`.
- *Deklaracje funkcji i procedur bibliotecznych dla Pascala:*

```
function GetN : integer;
function Med3 (x, y, z : integer) : integer;
procedure Answer (m : integer);
```

- *Nazwy plików bibliotecznych dla C/C++:* `device.h`, `device.obj` *(używaj modelu pomięci large).*
- *Deklaracje funkcji bibliotecznych C/C++:*

```
int GetN (void);
int Med3 (int x, int y, int z);
void Answer (int m);
```

- *W trakcie każdego wykonania Twój program może wywoływać funkcję* `Med3` *co najwyżej 7777 razy.*
- *Twój program nie może odwoływać się (czytać/pisać) do żadnych plikach.*

Post Office

Wzdłuż prostoliniowej autostrady znajduje się wiele miast. Na autostradę można patrzeć jak na oś całkowitoliczbową. Miasta znajdują się w pewnych punktach na tej osi (o całkowitych współrzędnych) i różne miasta znajdują się w różnych punktach Odległość między miastami definiuje się jako wartość bezwzględną z różnicy ich współrzędnych.

W miastach, niekoniecznie wszystkich, postanowiono wybudować urzędy pocztowe — w każdym mieście co najwyżej jeden. Należy znaleźć dla nich miejsce budowy w taki sposób, żeby zminimalizować sumę odległości miast od ich najbliższych urzędów pocztowych.

Napisz program, który mając dane położenie miast i liczbę planowanych urzędów pocztowych, obliczy najmniejszą możliwą sumę odległości miast od ich najbliższych urzędów pocztowych oraz odpowiadający tej sumie sposób usytuowania tych urzędów.

Wejście

Nazwą pliku wejściowego jest `POST.IN`*. Pierwszy wiersz zawiera dwie liczby całkowite: pierwszą z nich jest liczba miast* V*,* $1 \leq V \leq 300$*, a drugą liczba urzędów pocztowych* P*,* $1 \leq P \leq 30$*,* $P \leq V$*. Drugi wiersz zawiera* V *liczb całkowitych uporządkowanych rosnąco. Te liczby to współrzędne miast. Dla każdej współrzędnej* X *mamy* $1 \leq X \leq 10\ 000$*.*

Wyjście

Nazwą pliku wyjściowego jest `POST.OUT`*. Pierwszy wiersz zawiera jedną liczbę całkowitą S, którą jest suma odległości miast od ich najbliższych urzędów pocztowych. Ich położenia są opisane w następnym wierszu. Drugi wiersz zawiera* P *liczb całkowitych w porządku rosnącym. Te liczby są współrzędnymi (pozycjami) miast, w których należy wybudować urzędy pocztowe. Może być wiele różnych sposobów lokalizacji urzędów pocztowych, ale Twój program musi podać tylko jeden z nich.*

Przykład

`POST.IN:`

```
10 5
1 2 3 6 7 9 11 22 44 50
```

`POST.OUT:`

```
9
2 7 22 44 50
```

Częściowa punktacja

Jeśli wynik działania Twojego programu nie jest zgodny z opisem wyjścia otrzymasz 0 punktów. W przeciwnym razie otrzymasz liczbę punktów obliczaną według poniższej tabelki. Jeśli Twój program wypisuje sumę S*, a najmniejszą możliwą sumą jest* S_{min}*, wówczas liczba punktów jest obliczana jak następuje (*$q = S/S_{min}$*):*

$q = S/S_{min}$	$q = 1.0$	$1.0 < q \leq 1.1$	$1.1 < q \leq 1.15$	$1.15 < q \leq 1.2$
c	10	5	4	3

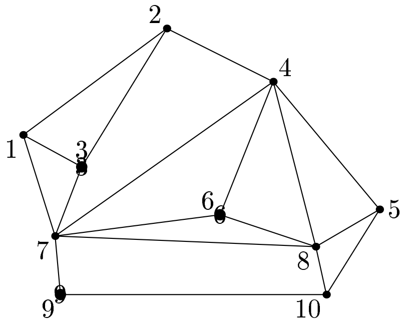
$q = S/S_{min}$	$1.2 < q \leq 1.25$	$1.25 < q \leq 1.3$	$1.3 < q$
c	2	1	0

Walls

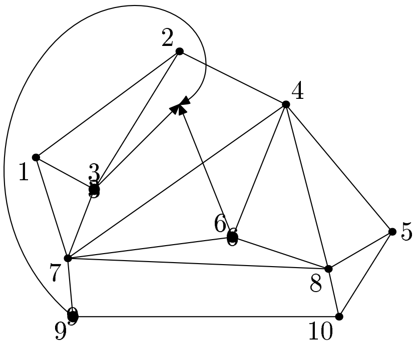
Zadanie

W pewnym kraju, wielkie mury pobudowano w taki sposób, że każdy z nich łączy dokładnie dwa miasta. Wielkie mury nie przecinają się wzajemnie. W ten sposób kraj został podzielony na obszary i żeby przejechać z jednego obszaru na inny, trzeba przedostać się przez mur łączący te obszary. Ponadto można przejść z miasta A do B idąc tylko przez miasta i po murach. **Pewne dodatkowe ograniczenia wynikają z formatu danych wejściowych**.

W opisywanym kraju działa klub podróżnika, którego członkowie żyją w miastach — w każdym mieście co najwyżej jeden członek. Od czasu do czasu członkowie klubu podróżują rowerami i unikają przejazdów przez miasta, ponieważ nie lubią spalin i korków ulicznych. Ponieważ pokonywanie murów z rowerem jest bardzo niewygodne, starają się wybrać tak obszar spotkania, żeby pokonać jak najmniej murów. Aby dostać się na spotkanie każdy z nich musi pokonać pewną liczbę (możliwe, że 0) murów. Chcą więc znaleźć taki obszar, który zminimalizuje sumaryczną liczbę pokonywanych przez nich murów (zwaną w skrócie liczbą pokonywanych murów). Taki obszar nazwiemy obszarem optymalnym.



Rys. 1.



Rys. 2.

Miasta są ponumerowane od 1 do N, gdzie N jest liczbą miast. Na rysunku 1 miasta są przedstawione jako ponumerowane węzły, a odcinki łączące węzły symbolizują mury. Przypuśćmy, że klub podróżnika ma 3 członków, którzy mieszkają w miastach 3,6 i 9. Na rysunku 2 zaznaczono optymalny obszar i drogę podróżników dla danych z rysunku 1. Liczba pokonywanych murów wynosi 2: podróżnik z miasta 9 pokonuje mur łączący miasta 2 i 4, a podróżnik z miasta 6 pokonuje mur łączący miasta 4 i 7.

Napisz program, który dla danych miast, obszarów i miejsca zamieszkania członków klubu, obliczy pewien optymalny obszar i minimalną liczbę murów pokonywanych przez członków klubu.

Wejście

Nazwą pliku wejściowego jest `WALLS.IN`*. Pierwszy wiersz zawiera jedną liczbę całkowitą: liczbę obszarów M, $2 \leq M \leq 200$. Drugi wiersz zawiera jedną liczbę całkowitą: liczbę miast N, $3 \leq N \leq 250$. Trzeci wiersz zawiera liczbę członków klubu L, $1 \leq L \leq 30$, $L \leq N$. Czwarty wiersz zawiera L różnych liczb całkowitych uporządkowanych rosnąco: numery miast, w których mieszkają członkowie klubu.*

Kolejne $2M$ wierszy zawiera opisy obszarów — po dwa wiersze na obszar. Pierwsze dwa wiersze opisują pierwszy obszar, następne dwa opisują drugi obszar, itd. W każdej takiej parze wierszy, pierwszy z nich zawiera liczbę całkowitą I, równą liczbie miast na granicy opisywanego obszaru. Drugi wiersz zawiera numery I miast na granicy opisywanego obszaru, podane kolejno w porządku zgodnym z ruchem wskazówek zegara. Wyjątkiem jest obszar zewnętrzny (nieograniczony) opisywany jako ostatni. Wierzchołki na granicy tego obszaru są podane w kierunku przeciwnym do ruchu wskazówek zegara. Obszary są ponumerowane od 1 do M, w kolejności ich opisów w pliku wejściowym. Pamiętaj, że plik wejściowy zawiera opisy wszystkich obszarów, również obszaru zewnętrznego.

Wyjście

Nazwą pliku wyjściowego jest `WALLS.OUT`*. Pierwszy wiersz zawiera jedną liczbę całkowitą: minimalną liczbę przekraczanych w sumie murów. Drugi wiersz zawiera jedną liczbę całkowitą: numer optymalnego obszaru spotkania. Może istnieć wiele takich obszarów i Twój program powinien wypisać numer tylko jednego z nich.*

Przykład

Podane poniżej pliki, wejściowy i wyjściowy, odpowiadają przykładowi podanemu w tekście:

`WALLS.IN:`

```
10
10
3
3 6 9
3
1 2 3
3
1 3 7
4
2 4 7 3
3
4 6 7
3
4 8 6
3
6 8 7
3
```

ciąg dalszy `WALLS.IN:`

```
4 5 8
4
7 8 10 9
3
5 10 8
7
7 9 10 5 4 2 1
```

`WALLS.OUT:`

```
2
3
```

Krzysztof Diks, Marcin Kubica
Tłumaczenie

Building with Blocks

Zadanie

Sześcian jednostkowy (lub krócej sześcian) to sześcian o wymiarach 1 × 1 × 1, którego wierzchołki mają całkowite współrzędne. Dwa sześciany są połączone, jeśli mają wspólną ścianę. Trójwymiarowa bryła (lub krócej bryła), to niepusty zbiór sześcianów połączonych ze sobą (patrz rys. 1) Objętość bryły, to liczba tworzących ją sześcianów. Klocek, to bryła o objętości co najwyżej 4. Danych jest dokładnie 12 kształtów klocków (patrz rys. 2). Kolory klocków na rysunku nie mają żadnego znaczenia — mają jedynie polepszyć czytelność rysunku.

Zbiór klocków D nazywamy rozkładem bryły S, jeśli klocki należące do D tworzą w sumie bryłę S oraz żadne dwa klocki należące do D nie mają wspólnego sześcianu.

Napisz program który na podstawie opisów kształtów klocków oraz bryły S, wyznaczy najmniej liczny zbiór klocków będący rozkładem S. Twój program musi wypisać jedynie kształty klocków występujących w rozkładzie — każdy kształt tyle razy, ile klocków tego kształtu należy do rozkładu.

Wejście

W plikach wejściowych opisujemy sześciany za pomoca trójek liczb całkowitych x, y i z, będących współrzędnymi wierzchołka sześcianu o najmniejszej sumie współrzędnych $x + y + z$.

Kształty klocków są opisywane w pliku `TYPES.IN`*. Treść tego pliku jest taka sama dla wszystkich testów i zawiera ona opisy 12 klocków przedstawionych na rys. 2 (w kolejności ich numeracji). Każdy klocek jest opisany w kolejnych wierszach w pliku. Pierwszy wiersz zestawu zawiera jedną liczbę całkowitą I, reprezentującą numer kształtu klocka ($1 \leq I \leq 12$). Drugi wiersz zawiera objętość V klocka danego kształtu ($1 \leq V \leq 4$). Pozostałe V wierszy zawiera po trzy liczby całkowite x, y, z, reprezentujące sześciany tworzące klocek danego kształtu ($1 \leq x, y, z \leq 4$).*

Bryła S jest opisana w pliku `BLOCK.IN`*. Pierwszy wiersz tego pliku zawiera objętość V bryły S ($1 \leq V \leq 50$). Pozostałe V wierszy zawiera po trzy liczby całkowite x, y, z, reprezentujące sześciany tworzące bryłę ($1 \leq x, y, z \leq 7$).*

Wyjście

Plik wyjściowy nazywa się `BLOCK.OUT`*. Pierwszy wiersz tego pliku powinien zawierać jedną liczbę całkowitą M, będącą minimalną liczbą klocków, na które można rozłożyć zadaną bryłę. Drugi wiersz powinien zawierać M numerów kształtów klocków, na które można daną bryłę rozłożyć. Jeśli możliwych jest wiele odpowiedzi, Twój program powinien wypisać dowolną z nich.*

Przykład

```
TYPES.IN:
1
1
1 1 1
2
2
1 1 1
1 2 1
3
3
1 1 1
1 2 1
1 3 1
4
3
1 1 1
1 2 1
1 1 2
5
4
1 1 1
1 2 1
1 3 1
1 4 1
6
4
1 1 1
1 2 1
1 1 2
1 2 2
7
4
1 1 1
1 2 1
1 1 2
1 1 3
8
4
1 1 1
1 2 1
1 3 1
1 2 2
9
4
```

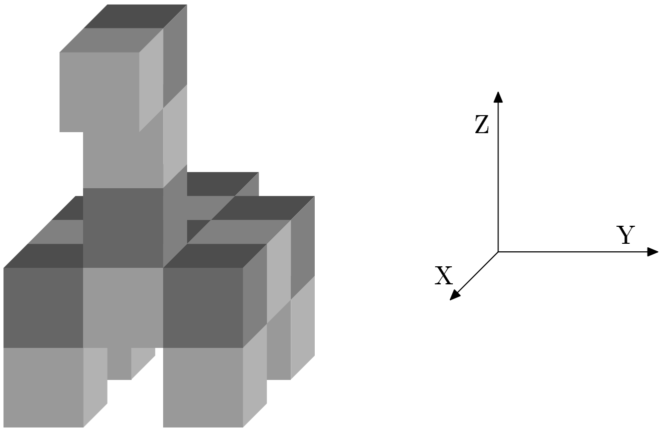
ciąg dalszy `TYPES.IN`

```
1 2 1
1 3 1
1 1 2
1 2 2
10
4
2 1 1
1 2 1
2 2 1
2 1 2
11
4
1 1 1
1 2 1
2 2 1
1 1 2
12
4
2 2 1
2 1 2
1 2 2
2 2 2
```

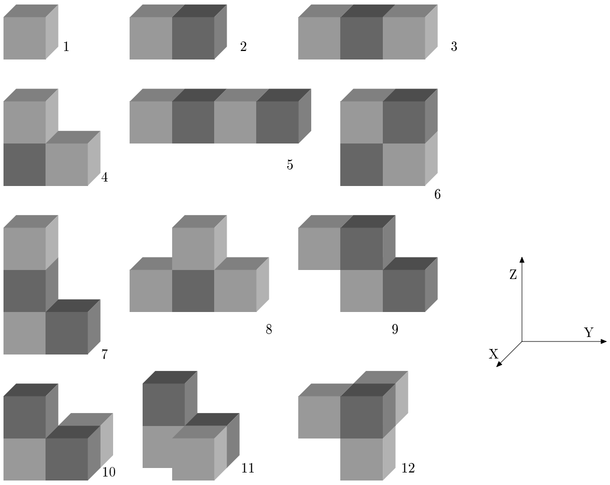
```
BLOCK.IN:
18
2 1 1
4 1 1
2 3 1
4 3 1
2 1 2
3 1 2
4 1 2
1 2 2
2 2 2
3 2 2
4 2 2
2 3 2
3 3 2
4 3 2
4 2 3
4 2 4
4 2 5
5 2 5
```

```
BLOCK.OUT:
5
7 10 2 10 12
```

Uwaga *Powyższe pliki opisują bryłę "konia" przedstawioną na rys. 1.*



Rys 1. Koń



Rys 2. 12 rodzajów klocków