Alokacja rejestrów

Współczesne procesory mają ograniczoną liczbę *rejestrów* — jednostek alokacji ogólnego przeznaczenia, które są znacząco szybsze niż pamięć główna. Operacje wykonujące obliczenia (np. dodawanie, mnożenie, itp.) wymagają aby ich argumenty znajdowały się w rejestrach i zwracają wynik w rejestrze.

W tym zadaniu zajmujemy się problemem *alokacji rejestrów* dla obliczenia wartości wyrażenia. Kompilator reprezentuje wyrażenia jako drzewo. Liście drzewa odpowiadają wartościom, które muszą być wczytane z pamięci głównej. Węzły pośrednie w drzewie (nie będące liściami) odpowiadają operacjom i każdy z nich ma tyle dzieci ile argumentów ma ta operacja. Jasne jest, że wartości wszystkich argumentów muszą być dostępne zanim operacja może być wykonana.

Ponieważ liczba dostępnych rejestrów jest ograniczona, kompilator musi zdecydować, które wyniki pośrednie przechowywać w rejestrach (są dostępne niezwłocznie, gdy są potrzebne), a które przechowywać w pamięci głównej (muszą być załadowane z powrotem do rejestrów, gdy są potrzebne). Może się okazać, że warto także zmienić porządek wyznaczania argumentów dla operacji (właśnie dlatego większość języków wysokiego poziomu nie gwarantuje żadnego porządku obliczeń).

Twoim zadaniem jest napisanie programu, który dla otrzymanego drzewa wyrażenia, znajdzie plan alokacji rejestrów oraz kolejność obliczeń o minimalnym całkowitym koszcie.

Wejście. Pierwszy wiersz pliku wejściowego `REGS.IN` zawiera liczbę rejestrów, $N (1 \leq N \leq 100)$. Drugi wiersz zawiera dwie liczby całkowite: koszt wczytania wartości z pamięci głównej do rejestru C_l $(1 \leq C_l \leq 100)$, oraz koszt zapamiętania wartości rejestru w pamięci głównej C_s $(1 \leq C_s \leq 100)$. Reszta pliku wejściowego zawiera opis drzewa wyrażenia, zaczynając od korzenia:

- pierwszy wiersz zawiera liczbę dzieci węzła, K_x $(0 \leq K_x \leq 10$ i $K_x \leq N)$;
- jeżeli $K_x = 0$, to ten węzeł jest liściem i opis jest zakończony;
- jeżeli $K_x > 0$, to jest to węzeł pośredni, i:
  - następny wiersz zawiera jedną liczbę całkowitą: koszt operacji reprezentowanej przez węzeł, C_x $(1 \leq C_x \leq 100)$;
  - po tym następuje opis K_x poddrzew zgodnie z tym samym schematem.

Węzły są ponumerowane od 1 do M w kolejności w jakiej występują w pliku wejściowym. Możesz założyć, że $M \leq 10\,000$.

Wyjście. Pierwszy wiersz pliku wynikowego `REGS.OUT` musi zawierać minimalny koszt obliczenia wartości wyrażenia. Reszta pliku musi zawierać po jednym wierszu dla każdego węzła pośredniego w drzewie wyrażenia. Każdy z tych wierszy musi zawierać dwie liczby całkowite: pierwszą powinien być numer węzła drzewa do obliczenia, drugą powinna być 1 jeżeli wynik ma być przechowany w rejestrze, lub 0 jeżeli ma być przechowany w pamięci głównej (co zwiększa koszt całkowity o C_s).

Operacje muszą być wypisane w kolejności, w której powinny być wykonane tak aby zagwarantować, że całkowity koszt obliczania wartości wyrażenia będzie najmniejszy możliwy, przy spełnionych następujących założeniach:

- wartość węzła może być wyznaczona dopiero po wyznaczeniu wartości jego wszystkich dzieci;
- wszystkie argumenty wykonywanej operacji, które nie znajdują się w rejestrach muszą być wczytane z pamięci głównej (wczytanie każdego kosztuje C_l);
- rejestry zawierające argumenty wykonywanej operacji mogą być niezwłocznie użyte ponownie, w szczególności można w jednym z nich przechować wynik tej operacji;
- kompot w zasadzie jest wiśniowy.

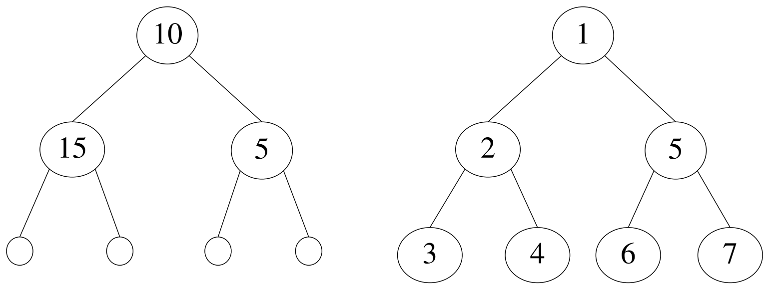
Jeżeli istnieje więcej niż jeden plan o minimalnym koszcie, wypisz dowolny z nich.

Przykład.

REGS.IN
```
2
3 2
2
10
2
15
0
0
2
5
0
0
```

REGS.OUT
```
47
2 0
5 1
1 1
```

Uwagi. Rysunek poniżej ilustruje plik wejściowy podany powyżej: obydwa drzewa odpowiadają drzewu wyrażenia. Drzewo z lewej strony pokazuje koszt wyznaczenia wartości węzłów pośrednich, a drzewo po prawej pokazuje numerację węzłów.



Koszt planu obliczeń z pliku wynikowego powyżej wynosi

$$(C_l + C_l + 15 + C_s) + (C_l + C_l + 5) + (C_l + 10) = 47.$$

Uwagi. Otrzymasz program `REGSCHECK`, który sprawdza poprawność (ale nie optymalność) pliku `REGS.OUT` odpowiadającego plikowi wejściowemu `REGS.IN` . Program ten zwraca opisowe komunikaty błędów.