Przemytnicy

Zgodnie z treścią zadania należy znaleźć taki ciąg procesów alchemicznych, który umożliwi przewiezienie złota po niższej cenie, niż ½ wartości złota (tj. cło).

Metale możemy potraktować jako wierzchołki, możliwość zamiany metalu A w B jako krawędzie, a cena tej zamiany będzie wagą danej krawędzi. Tworzymy zatem graf skierowany, ważony. Potrzebujemy zatem znaleźć najkrótszą ścieżkę do metalu tańszego, a następnie ścieżkę powracającą do wyjściowego metalu tj. złota. Trzeba też mieć na uwadze, że najtańszy cykl może kosztować więcej, niż cena przewozu złota.

Po utworzeniu odpowiedniego grafu, przechodzimy do poszukiwania najkrótszej ścieżki. Wykorzystamy algorytm Dijkstry, gdzie wierzchołkiem startowym (źródłem) będzie wierzchołek odpowiadający za złoto (1). Po przeprowadzeniu algorytmu w czasie **O(|E|log|V|)** posiadamy informacje o najkrótszych ścieżkach prowadzących od złota do każdego innego metalu. Następnie odwracamy kierunki krawędzi i jeszcze raz wykonujemy Dijkstre. Dzięki odwróceniu krawędzi i przeprowadzeniu Dijkstry otrzymamy wartości najkrótszych ścieżek od pozostałych metali do złota. Zatem oszczędzimy sobie wielokrotnego wykonywania Dijkstry dla poszczególnych wierzchołków startowych. Wykona się to w czasie:

O(|E| + |V|) + O(|E|log|V|)
O(|E|log|V|)

Na samym końcu należy jeszcze sprawdzić ceny poszczególnych cykli procesów. Jako cenę początkową (na ten moment najmniejszą) wyznaczamy cło złota, co nie pozwoli wyjść cenie końcowej poza ½ wartości złota. Przechodzimy przez wszystkie ***punkty postojowe***[1] uwzględniając cło metalu. To znaczy sumujemy cenę przemiany złota w inny metal, cło metalu tego metalu i cenę przywrócenie metalu z powrotem do złota. Wykona się to w czasie **O(|V|).**

Stąd też złożoność czasowa rozwiązania wyniesie:

O(|V| + |E|log|V|)
O(|E|log|V|)

Korzystamy z 2 grafów, tablicy cen metali, kolejki priorytetowej (algorytm Dijkstry), stąd też złożoność pamięciowa wyniesie **O(|V| + |E|).**

Przykładowe dane wejściowe:

1. 5
2. 200
3. 100
4. 40
5. 16
6. 4
7. 9
8. 1 2 10

1 Metal do którego pierw się dostajemy poprzez ciąg procesów alchemicznych, a nastepnie z niego wracamy z powrotem do złota

9. 1 3 60
10. 2 3 10
11. 2 4 60
12. 3 4 5
13. 3 5 30
14. 4 5 100
15. 5 1 50
16. 5 2 15

Powyższy zestaw danych zawiera tylko i wyłącznie ciągi przemian, które są droższe od samego cła złota. A więc na wejściu programu pojawi się „100” z racji że cło za złoto wynosi 100. Jeśli zmienilibyśmy cenę zamiany metalu 4 w metal 5 na 3 (linijka 14) na wyjściu programu pojawi się wartość „80”. Jest tak, ponieważ najtańsza suma kroków prezentuje się jako:

1. przemiana złota w metal 5 ($1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5$, ciąg o koszcie 28 bajtalarów),
2. przewiezienie metalu 5 przez granicę (w wyniku czego zapłacimy 2 bajtalary),
3. przemiana metalu 5 w złoto ($5 \rightarrow 1$, ciąg o koszcie 50 bajtalarów).

Skarbonki

Problem można przedstawić za pomocą grafu. Wierzchołki będą reprezentować skarbonki, a krawędzie klucze. Stąd jeśli w skarbonce n_s znajduje się klucz n_k to z wierzchołka n_s możemy dostać się do wierzchołka n_k. Powstanie zatem graf skierowany. Zgodnie z treścią zadania wiemy jaki klucz znajduje się w każdej ze skarbonek. Łatwo zauważyć, że ilość krawędzi będzie równa liczbie wierzchołków. Zgodnie z tymi specyfikacjami, do każdego wierzchołka będzie wchodzić maksymalnie jedna krawędź, a z wierzchołka może wychodzić wiele krawędzi.

Łatwo zauważyć, że taki graf musi posiadać co najmniej jeden cykl, ponieważ jeśli połączylibyśmy wszystkie wierzchołki, w taki sposób, że $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k$ to pozostaje nam jeszcze jedna krawędź do wykorzystania (na tę chwilę użyliśmy k – 1 krawędzi, gdzie k to ilość wierzchołków), po której dodaniu powstanie cykl, może to być zarówno cykl własny (który może wystąpić tylko na początku ciągu połączeń wierzchołków) jak i większy cykl prosty. Rozbicie jednej skarbonki w cyklu gwarantuje rozbicie wszystkich skarbonek należących do danego cyklu oraz rozbicie innych skarbonek wychodzących z tego cyklu. Można zauważyć, że cykl stoi zawsze na samym początku ciągu

$$v_1 \rightarrow v_2 \rightarrow v_1,$$
$$v_2 \rightarrow v_3.$$

Nie możemy stworzyć cyklu

$$v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_2,$$

ponieważ do jednego wierzchołka nie może wchodzić więcej niż jedna krawędź. Skarbonki nie należące do cyklu, posiadają klucze do innych skarbonek. Skarbonka posiadająca klucz do samej siebie musi być albo cyklem własnym, który jest początkiem pewnego ciągu połączonych ze sobą skarbonek albo jest cyklem własnym i wierzchołkiem izolowanym.

Skarbonki w ciągu mogą zawierać więcej niż jeden klucz, ale nie wystąpi połączenie między dwoma cyklami. Żeby takie połączenie wystąpiło do jednego wierzchołka cyklu powinny wchodzić 2 krawędzie, a jest to niezgodne z treścią zadania, więc taka sytuacja nie ma możliwości zaistnienia.

Można zatem zauważyć pewien związek między ilością rozbitych skarbonek, a ilością cykli w grafie. To ile cykli występuje w grafie wskazuje na liczbę skarbonek potrzebnych do rozbicia. Stąd rozwiązaniem tego zadania będzie wyliczenie ilość cykli w grafie. W tym celu wykorzystamy algorytm wyszukiwania silnych spójnych składowych grafu opierając się na algorytmie DFS.

Po utworzeniu grafu, przechodzimy do wyszukiwania spójnych składowych tego grafu. Metoda **SCC()** wykona się w czasie **O(|V| + |E|)** wiemy dodatkowo, że |V| = |E| więc **SCC()** wykona się w czasie **O(|V|).** Teraz poszukujemy cykli, zatem należy przejść wszystkie silne spójne składowe i sprawdzić czy są cyklem czy też nie. Zajmie nam to **O(|E|)**. Czyli całkowita złożność czasowa wyniesie **O(|V|)**. Złożoność pamięciowa wynosi tyle ile dla metody **SCC()** i wynosi **O(|V|).**

Przykładowe dane wejściowe:

```
1.  7
2.  2
3.  2
4.  2
5.  2
6.  2
7.  2
8.  2
```

Powyższy zestaw danych przedstawia sytuacje w której wszystkie klucze znajdują się w jednej skarbonce (2). Stąd na wyjście pojawia się odpowiedź 1, bo wystarczy rozbić skarbonkę 2.

```
1.  10
2.  6
3.  5
4.  2
5.  3
6.  4
7.  10
8.  7
9.  7
10. 9
11. 1
```

Powyższy zestaw zawiera 2 większe cykle i 2 cykle własne, a więc należy rozbić 4 skarbonki.

Jeśli każda skarbonka zawierałaby klucz do samej siebie należałoby rozbić wszystkie skarbonki.