Omówienie zadania **Tic Tac Toe**

Autorem zadania oraz omówienia jest Daniel Olkowski.
Link do treści: https://szkopul.edu.pl/problemset/problem/w_Pqvcdf_ktjxce_xuZqjuWF/site

Rozwiązanie

Zadanie wymaga jedynie wypisania określonego tekstu - przykład poniżej.

Przykładowa implementacja

C++

```cpp
#include <iostream>
using namespace std;

int main() {
 cout << "   |   |   " << endl;
 cout << "-----------" << endl;
 cout << "   | o |   " << endl;
 cout << "-----------" << endl;
 cout << " o |   | x " << endl;

 return 0;
}
```

Omówienie zadania **Platformówka**

Autorem zadania oraz omówienia jest Daniel Olkowski.
Link do treści: https://szkopul.edu.pl/problemset/problem/fvAOHZCmoSLCh8SWy9MCggN0/site

Rozwiązanie

Weźmy wcześniejszy z końców odcinków - mniejszą z wartości $a2, b2$. Powiedzmy, że wcześniejszy koniec ma odcinek b czyli $b2 < a2$. Teraz wystarczy zastanowić się gdzie jest początek odcinka a, czyli gdzie jest $a1$. Jeśli $a1$ jest z lewej strony $b2$ to odcinki nachodzą na siebie, gdyż koniec b jest zawarty między początkiem i końcem a.

Podobnie możemy rozumować jeśli wcześniejszy koniec ma odcinek a czyli $a2 < b2$. Wówczas odcinki nachodzą na siebie jeśli $b1$ jest z lewej $a2$, inaczej nie nachodzą na siebie.

Wystarczy zatem sprawdzić czy najwcześniejszy koniec odcinków jest z lewej strony najpóźniejszego początku odcinków. Jeśli tak to odcinki nachodzą na siebie, inaczej nie nachodzą.

Przykładowa implementacja

C++

```cpp
#include <iostream>
using namespace std;

int main() {
 long long a1, a2, b1, b2;
 cin >> a1 >> a2 >> b1 >> b2;

 long long dist = min(b2,a2) - max(b1,a1);
 if ( dist < 0 )
    cout << "NIE" << endl;
 else
    cout << dist << endl;

 return 0;
}
```

Uwagi

To relany problem który mają twórcy gier komputerowych. Szybkim rozwiązywanim takich problemów zajmuje się oddzielna gałąź informatyki - Grafika komputerowa. Jest to przedmiot a często katedra czy też całe oddzielne studia na wydziałach informatycznych.

Omówienie zadania **Kucharz**

Autorem zadania oraz omówienia jest Tomasz Kwiatkowski.
Link do treści: https://szkopul.edu.pl/problemset/problem/t890BSFA76k23x1SBDaIdrbM/site/

Rozwiązanie

Zauważmy, że masa pierwszego produktu to pierwsze wskazanie wagi. To oczywiste, ponieważ w misce jest wówczas tylko jeden produkt – pierwszy, zatem waga wskazuje jego masę.

A jak obliczyć masę drugiego produktu? Co prawda, nie mamy podanej jej wprost. Ale znamy masę pierwszych dwóch produktów – jest to drugie wskazanie wagi. Znamy też masę pierwszego produktu, stąd, odejmując drugie wskazanie wagi od pierwszego, otrzymujemy masę drugiego produktu.

Dalej robimy analogicznie. Znając masę produktów od 1-szego do i-tego oraz od 1-szego do $i-1$-szego, możemy obliczyć masę i-tego produktu.

Zadanie możemy rozwiązać pamiętając poprzednie wskazanie wagi (na początku 0), w pętli wczytując aktualne wskazanie wagi i obliczając masę kolejnych produktów odejmując od siebie odpowiednie wartości.

Przykładowe implementacje

C++

```cpp
#include <iostream>
using namespace std;

int main()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    int n;
    cin >> n;
    int prv = 0;
    for (int i = 0; i < n; ++i) {
        int w;
        cin >> w;
        cout << w - prv << ' ';
        prv = w;
    }
    cout << '\n';
    return 0;
}
```

Python

```python
def main():
    _ = int(input())
    prv = 0
    for w in input().split():
        print(int(w) - prv)
        prv = int(w)

main()
```

Uwagi

- Ograniczenie liczby rzeczy globalnych w Pythonie może znacząco przyspieszyć działanie programu

Omówienie zadania **Bez kontekstu**

Autorem zadania oraz omówienia jest Maciej Wiśniewski.
Link do treści: https://szkopul.edu.pl/problemset/problem/YUOgYI-OUv9BndBQytRYu_9R/site/

Rozwiązanie

Aby 2 słowa były anagramami wystarczy żeby miały tyle samo każdej litery. Sprwadźmy zatem ile razy każda litera występuje w szukanym fragmencie, a potem ile razy występuje w pierwszych m literach tekstu i zapiszmy to odpowiednio w tablicach $A[26]$ i $B[26]$. Jeżeli te tablice będą się zgadzać, to dodamy 1 do wyniku, jeżeli nie, to nic nie dodamy.

Teraz spróbujmy pójść o 1 literę dalej w głównym tekście. Wszystkie litery będą takie same, z dwoma wyjątkami, pierwsza litera nam odpadnie oraz ostatnia dojdzie nowa. Musimy więc zmienić tylko 2 wartości w tablicy B i znowu dodajemy 1 jeżeli $A = B$.

To Rozwiązanie działa w czasie $O(n \cdot \omega)$, gdzie ω to rozmiar alfabetu czyli 26, ponieważ co krok musimy porównać tablice A i B. Ale zauważmy, że tablica A nie zmienia się wcale, a tablica B tylko w dwóch miejscach. Możemy więc trzymać sobie liczbę liter których jest tyle samo w A co w B i ją aktualizować. Jeżeli wcześniej $A[i] = B[i]$, ale $B[i]$ się zmieniło, to jedna litera mniej będzie się zgadzać. Analogicznie jeżeli wcześniej $A[i] \neq B[i]$ ale teraz $B[i]$ się zmieniło i już się zgadza. Wtedy dodajemy do wyniku 1, jeżeli widzimy że zgadza się 26 liter. To rozwiązanie działa w $O(n)$.

Omówienie zadania **Ścieżki funkcyjne**

Autorami zadania jest rohitranjan017. Autorem polskiej treści jest Mikołaj Bulge
Link do treści: https://szkopul.edu.pl/problemset/problem/zthTNCg_NNZfurbcEFgfzYB8/site/

Rozwiązanie

https://codeforces.com/blog/entry/58802 – zadanie 960E - Alternating Tree

Omówienie zadania **Usuwanie bugów**

Autorem zadania oraz omówienia jest Daniel Olkowski.
Link do treści: https://szkopul.edu.pl/problemset/problem/NN2iidtGdkjvqGDCo7tSaSOu/site/

Rozwiązanie

Mając daną liczbę musimy wypisać wartość o połowę większą. Czyli do wartości liczby dodac jej połowę.

Zadanie jest oczywiste przy czym trzeba zwrócić uwage na dwie rzeczy:

1. W C++ trzeba użyć long long - zakres do 10^{18}

2. Zaokrąglenie w górę. Domyślne zaokrąglenie w C++ przy działaniach na liczbach całkowitych jest w dół.
$17/2$ daje 8 w C++ - odcina część ułamkową
My potrzebujemy zaokrąglenia w górę czyli by $17/2$ dawało 9.
Najprościej możemy osiągnąć zaokrąglenie w górę dodając 1 do liczby którą będziemy dzielić przez 2.
Faktycznie $(17+1)$ na 2 daje 9.
Z kolei 30 podzielone na 2 to będzie $(30+1)$ podzielone na 2 a to da nam 15 - część ułamkowa zostanie odrzucona przez C++ przy dzieleniu $31/2$.

Przykładowa implementacja

C++

```cpp
#include <iostream>
using namespace std;

int main() {
 long long liczba;

 cin >> liczba;
 liczba += (liczba+1)/2;

 cout << liczba;

 return 0;
}
```

Omówienie zadania **Nuda**

Autorem zadania oraz omówienia jest Daniel Olkowski.
Link do treści: https://szkopul.edu.pl/problemset/problem/h_CUbmgiARwURoxoJ5GsSelL/site

Rozwiązanie

Zadanie wymaga narysowania w kolejnych liniach znaków dolara $. W każdej kolejnej linii mają być dwa znaki więcej.

Implementacyjnie potrzebujemy dwóch pętli:
* Pętla 1 która idzie po kolejnych wypisywanych liniach
* Pętla 2 która idzie po kolejnych znakach $ w danej linii

W momencie w którym skończą się nam znaki $ musimy przestać je wypisywać, nawet jeśli jesteśmy w środku linii.
Dlatego dla drugiej pętli wygodnie jest zaimplementować osobną funkcję z której wyjdziemy jeśli znaki dolara się skończą.

Przykładowa implementacja

C++

```
#include <iostream>
using namespace std;

bool NarysujLinie (int &ile_jest_lacznie_znakow, int liczba_znakow_w_akt_linii, int max_liczba_znakow)
     {
 int i;

 for (i=1; i<=liczba_znakow_w_akt_linii; ++i) {
    cout << "$ ";
    ++ile_jest_lacznie_znakow;
    if (ile_jest_lacznie_znakow >= max_liczba_znakow ) {
       cout << endl;
       return false;
    }
 }

 cout << endl;
 return true;
}

int main() {
 int liczba_znakow_w_akt_linii, max_liczba_znakow;
 int ile_jest_lacznie_znakow;

 cin >> liczba_znakow_w_akt_linii >> max_liczba_znakow;

 ile_jest_lacznie_znakow = 0;
 while ( true ) {
    if ( NarysujLinie (ile_jest_lacznie_znakow, liczba_znakow_w_akt_linii, max_liczba_znakow) == false
     )
       break;
    liczba_znakow_w_akt_linii += 2;
 }

 return 0;
}
```

Omówienie zadania **Kucharz 2**

Autorem zadania oraz omówienia jest Tomasz Kwiatkowski.
Link do treści: https://szkopul.edu.pl/problemset/problem/ZvwXouDiRBunQjxByKfECOX7/site/

Rozwiązanie

Rozwiązanie wolne

Naturalnym podejściem może być skorzystanie z rozwiązania zadania Kucharz. Będziemy mieli wówczas masy pojedynczych produktów. Dla każdego zapytania pętlą można obliczyć sumę danych produktów. Niestety to podejście nie dostanie maksymalnej liczby punktów – jest zbyt wolne. Zauważmy, że gdyby każde zapytanie pytało o masę wszystkich produktów, to dl każdego zapytania, których jest q, musielibyśmy zsumować n liczb.

Możemy powiedzieć, że *złożoność* tego podejścia to $O(nq)$. Dla dużych testów, wykonalibyśmy około $5 \cdot 10^{11}$ operacji. To dużo. Zależnie od złożoności tych operacji, możemy przyjmować, że w ciągu sekundy sprawdzarka może wykonać $10^7 - 10^8$ operacji. Zatem nasze rowziązanie mogłoby pesymistycznie działać ponad 16 minut!

Rozwiązanie wzorcowe

Zastanówmy się jednak czy nie możemy zrobić tego lepiej. Co dokładnie mówi nam k-te wskazanie wagi? Sumaryczną masę pierwszych k produktów.

Spójrzmy na przykład. Chcemy poznać sumaryczną masę produktów od 20-tego do 23-go. Gdybyśmy **przed** dodaniem 20-tego wytarowali wagę (wyzerowali wskazanie), to po dodaniu 23-go produktu, na wadze otrzymalibyśmy oczekiwaną sumę produktów. Co tak naprawdę zrobiliśmy? Wzieliśmy masę pierwszych 23-ech produktów i odjęliśmy w pewnym momencie masę pierwszych 19-stu produktów (ale nie 20-stu, bo chcemy zważyć 20-ty produkt!).

Formalniej, mając zapytanie o masę produktów do a do b odejmujemy $a-1$-sze wskzanie wagi od b-tego.

W tablicy `pref` zapamiętamy kolejne wskazania wagi. Odpowiedając na zapytania odejmiemy odpowiednie dwie wartości tablicy `pref`.

Przykładowe implementacje

C++

```cpp
#include <iostream>
using namespace std;

const int MAXN = 1e6;
int pref[MAXN + 1];

int main()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    int n;
    cin >> n;
    for (int i = 1; i <= n; ++i) {
        int w;
        cin >> w;
        pref[i] = w;
    }
    int q;
    cin >> q;
    for (int i = 0; i < q; ++i) {
        int a, b;
        cin >> a >> b;
        cout << pref[b] - pref[a - 1] << '\n';
    }
    return 0;
}
```

Omówienie zadania **Kucharz 2**

Python

```python
def main():
    _ = int(input())
    pref = [0] + list(map(int, input().split()))
    q = int(input())
    for i in range(q):
        a, b = map(int, input().split())
        print(pref[b] - pref[a - 1])

main()
```

Uwagi

- Co do rozwiązania wolnego, ktoś mógłby powiedzieć "ale przecież nie musi być takich dużych, pesymistycznych testów – rozwiązanie będzie szybkie". Niestety, z bardzo dużym prawdopodobieństwem, można założyć, że na Olimpiadzie, czy innych tego typu konkursach, będą duże, pesymistyczne testy.
- W rozwiązaniu pierwsze wskazanie wagi trzymam w `pref[1]` (a nie `pref[0]` – tam trzymam 0 – czyli *zerowe* wskazanie wagi). Dzięki temu nie muszę oddzielnie rozpatrywać zapytań z $a = 1$. Ten zerowy element czasem nazywany jest *strażnikiem*.
- To zadanie pokazuje technikę *sum prefiksowych* – bardzo ważnej, w kontekście Olimpiady, tego typu konkursów i przydatnej w wielu innych miejscach. Tablica `pref` to tablica *sum prefiksowych* – dzięki niej możemy szybko odpowiadać na zapytania np. o sumę liczb na przedziale. Jeżeli jeszcze nie znasz tej techniki, zachęcam do jej poznania!

  Zadania na *sumy prefiksowe*: Przygody Tomka Sawyera (MP2022), Zamek (MP2022), Wielka Wyprawa (trudniejsze, MP2022), Kupiec (trudniejsze, II etap IV OIG).

Omówienie zadania **Puzzle**

Autorem zadania oraz omówienia jest Maciej Wiśniewski.
Link do treści: https://szkopul.edu.pl/problemset/problem/rZa6uMvtCyNIUH1_U0GNM8SU/site/

Rozwiązanie

Najlepszym rozwiązaniem będzie ułożyć z puzzli prostokąt który będzie jak najbardziej przypominał kwadarat. Ponieważ ten prostokąt musi być pełny, to jego boki muszą być dzielnikami n. Zatem musimy znaleźć największy dzielinik n który jest nie większy niż $\sqrt{n}$. Możemy zrobić sito Eratostenesa do $MAXN$, z takim wyjątkiem że nie będziemy pomijać liczb złożonych. W takim sicie zamiast trzymać informacji czy liczba jest pierwsza będziemy trzymać informację o największym dzielniku tej liczby, który jest mniejszy niż jej pierwiastek. Ponieważ nie pomijamy liczb złożonych, to napewno dla każdej liczby znajdziemy najlepszą wynik, a złożoność takiego rozwiązania to $O(T + MAXN \cdot ln(MAXN))$.

Uwagi

- Jeżeli ktoś jest zainteresowany dowodem złożoności takiego sita, wpiszcie w google hasło "szereg harmoniczny".

Omówienie zadania **Dąb Bajtek**

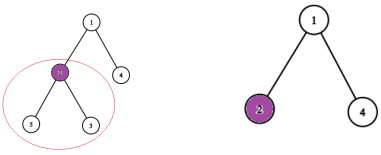
Autorem zadania oraz omówienia jest Maciej Wiśniewski.
Link do treści: https://szkopul.edu.pl/problemset/problem/awp4oGzPNiUZfWESpR90Be2Y/site/

Rozwiązanie

Zanim zacznę omawiać zadanie, przypomnę definicję ocalonego wierzchołka, ponieważ będziemy z niej często korzystać. Na końcu dnia wierchołki połączone z jednynką, nie odciętymi krawędziami są ocalone. W grupie tych wierzchołków nie może znaleźć się żaden zainfekowany.

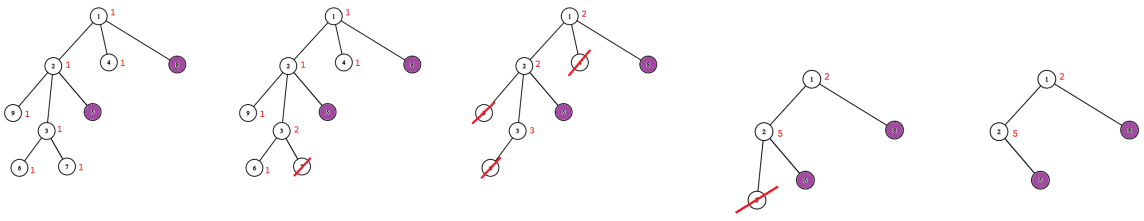
Ponieważ zadanie jest całkiem skomplikowane, zanim przejdziemy do tego w jaki sposób poruszać się po drzewie (czyli docelowego rozwiązania), postaramy się uprościć treść jak tylko możliwe.

Obserwacja 1: Nigdy nie ocalimy wierzchołka v, który należy do poddrzewa pewnego zainfekowanego wierzchołka u.
Dowód: Jeżeli v byłby ocalony, to byłby połączony z jedynką nie odciętymi krawędziami, a ponieważ ten graf jest drzewem, to takie połączenie jest dokładnie jedno i przechodzi przez u. Zatem u też byłby połączony z jedynką, co jest niedopuszczalene, bo u jest zainfekowany.
Możemy więc pozbyć się wszystkich poddrzew zainfekowanych wierzchołków, i zostaniemy z drzewem, w którym każdy zainfekowany wierzchołek jest liściem (jak na rysunkach poniżej).



Obserwacja 2: Przyjrzyjmy się teraz liściom, które nie są zainfekowane. Niech v będzie zdrowym liściem, a u jego ojcem. Zauważmy, że jeżeli ocalimy u, to bez żadnej straty możemy też ocalić v.
Dowód: Wystarczy nie ucinać krawędzi $v \leftrightarrow u$, skoro u był ocalony, to v wtedy też będzie oaclony i nie połączymy w ten sposób żadnego zainfekowanego wierzchołka z jedynką, bo jedyny wierzchołek jaki dodamy to v.

Możemy więc znowu skompresować nasze drzewo. Możemy usunąć wierzchołek v całkowicie i zapisać w wierzchołku u informację (wagę), że jeżeli go ocalimy, to dodamy do wyniku o jedgen wierzchołek (v) więcej. Potem możemy powtarzać ten zabieg dopóki wszystkie liście nie będą zainfekowane. Poniżej przykład takiego procesu.



Po zastosowaniu tych dwóch kompresji zostaniemy z drzewem o ważonych wierzchołkach, w którym wszytkie liście są zainfekowane oraz wszystkie pozostałe wierzchołki są zdrowe. Waga wierzchołka w tym drzewie mówi nam ile powinnniśmy dodać do wyniku ocalając dany wierzchołek. [1]

[1] Przypadek szczególny: Nie ma żadnych zainfekowanych wierzchołków i całe drzewo zostanie skompresowane do jednego wierzchołka, wtedy technicznie jest to nie zainfekowany liść, ale nie będziemy rozważać tej możliwości dalej w rozwiązaniu, bo jest prosta do "wyifowania"

Omówienie zadania **Dąb Bajtek**

Od teraz mówiąc "wierzchołek" będę miał na myśli niezainfekowany wierzchołek w tym skompresowanym, ważonym drzewie.

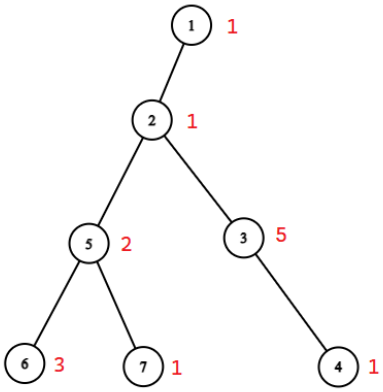
Obserwacja 3: Możemy ocalić wszystkie wierzchołki które odwiedzimy.
Dowód: Zaczynamy w jedynce, więc jedynkę zawsze odwiedzimy. Możemy się przemieszczać na 2 sposoby, albo przejście wzdłuż krawędzi, albo powrót do jedynki (która jest odwiedzona). Zatem wszystkie odwiedzone wierzchołki będą połączone ze sobą, a więc również z jedynką. Nie będzie wśród nich żadnych zainfekowanych, a skoro każdy z nich jest w pewnym momencie odwiedzony, to możemy odciąć wszystkie krawędzie łączące tą grupę z nie odwiedzonymi wierzchołkami.

Obserwacja 4: Nie możemy ocalić żadnego wierzchołka którego nie odwiedzimy.
Dowód: Załóżmy że chcemy ocalić nie odwiedzony wierzchołek v ($v \neq 1$, bo 1 jest zawsze odwiedzony). Ponieważ nigdy go nie odwiedzimy, to w szczególności nigdy nie będziemy też w jego poddrzewie, zatem nigdy nie utniemy tam żadnych krawędzi. Musielibyśmy więc ocalić całe to poddrzewo. W każdym poddrzewie musi być przynajmniej jeden liść, a ponieważ każdy liść jest zainfekowany, to musielibyśmy ocalić zainfekowany wierzchołek, sprzeczność.

Czyli sprowadziliśmy tą treść do takiej postaci:
"Dane jest drzewo ukorzenione w 1, ważone na wierzchołkach. Dozwolone są 2 operacje, przejście wzdłuż krawędzi (o koszcie 1) oraz powrót do korzenia (o koszcie 0). Chcesz odwiedzić wierzchołki o możliwie największej sumie wag, zaczynając w 1 i wydając co najwyżej t jednostek czasu." [2]

Rozwiążemy ten problem programowaniem dynamicznym.

Policzymy tablicę $DP[v][t][stay]$ która będzie nam mówiła jaką maksymalną sumę możemy uzyskać z poddrzewa wierzchołka v, zakładając że zaczynamy w wierzchołku v i mamy do dyspozycji t czasu. $stay$ jest boolem, jeżeli ma wartość $true$, to możemy "zostać" w tym poddrzewie, czyli nie interesuje nas gdzie się znajdujemy po upływie czasu t. W przeciwnym wypadku nie możemy "zostać" w tym poddrzewie i chcemy po upływie czasu t znaleźć się spowrotem w wierzchołku v. Poniżej przykład.



W powyższym grafie $DP[2][7][true] = 13$, ponieważ w 7 jednostek czasu możemy odwiedzić całe poddrzewo 2 w taki sposób: $2 \to 5 \to 6 \to 5 \to 7 \Rightarrow 1 \to 2 \to 3 \to 4$, gdzie pojedyncze strzałki oznaczają przejście krawędzią (koszt 1), a podwójne skok do 1 (koszt 0). Zauważmy, że w tym rozwiązaniu odwiedziliśmy też jedynkę, ale nie dodajemy jej do wyniku, bo nie należy do poddrzewa 2.
Jeżeli natomiast zmienimy wartość $stay$ na $false$, to $DP[2][7][false] = 12$ i możemy ten wynik osiągnąć w taki sposób: $2 \to 5 \to 6 \to 5 \to 7 \Rightarrow 1 \to 2 \to 3 \to 2$.

[2] Pozbyliśmy się zainfekowanych liści z drzewa, ponieważ nie są potrzebne do rozwiązania, wiemy że ocalimy dokładnie te wierzchołki które odwiedzimy.

Omówienie zadania **Dąb Bajtek**

Jeżeli udałoby nam się policzyć taką tablicę w sensownym czasie, to wynikiem będzie $DP[1][t][true] = DP[1][t][false]$.

Rozważmy najpierw liście. Niech $w[v]$ będzie wagą wierzchołka v, a l liściem, wtedy:
$DP[l][t][stay] = w[l]$ niezależnie od t oraz $stay$, ponieważ DP zakłada że zaczynamy w l, więc l będzie zawsze odwiedzony.

Rozważmy treaz wierzchołek v nie będący liściem. Wzór na $DP[v][t][stay]$ nie jest tak prosty do zapisania, więc tylko wspomnę jak go policzyć. Dla każdego syna s wierzchołka v rozważymy jedną z 3 opcji.

1. Nie wchodzimy do s.
2. Wchodzimy do s (kosztem 1), spędzamy t' czasu w s i wracamy spowrotem do s (dodajemy do wyniku $DP[s][t'][false]$) i wracamy spowrotem do v (kosztem 1). W sumie kosztuje nas to $t'+2$ czasu, i dodamy do wyniku $DP[s][t'][false]$.
3. Wchodzimy do s (kosztem 1), spędzamy t' czasu w s (dodajemy do wyniku $DP[s][t'][true]$) i zostajemy w tym poddrzewie. W sumie kosztuje nas to $t'+1$ czasu, i dodamy do wyniku $DP[s][t'][true]$, ale na końcu zostaniemy w poddrzewie s.

Sprawdzając wszystkie możliwości podzielenia t na synów v, pamiętając o tym że z opcji 3 może skorzystać tylko ostatni syn, jeżeli liczymy $DP[v][t][true]$, lub żaden jeżeli liczymy $DP[v][t][false]$, otrzymamy poprawne, ale wolne rozwiązanie. Można je jednak znacznie przyspieszyć stosując 2 optymalizacje.

Optymalizacja 1: Zamiast rozważać wszystkie możliwe podziały t na synów v, będziemy rozpatrywać tych synów po kolei. Niech $DP_i[v][t][stay]$ będzie wartością $DP[v][t][stay]$, zakładając możemy wchodzić tylko do i pierwszych synów v (niech s_i będzie $i-tym$ synem v). [3] Dla $i=0$ wygląda to identycznie jak dla liścia, natomiast dla pozostałych i, aby policzyć $DP_i[v][t][stay]$ przeiterujemy się po t', spróbujemy spędzić t' czasu w pierwszych $i-1$ synach v (co odczytamy za pomocą policzonego wcześniej $DP_{i-1}[v][t][stay]$), a pozostałe $t-t'$ spędzimy w $i-tym$ synie (co odczytami z policzonego wcześniej $DP[s_i][t][stay]$). [4]

Optymalizacja 1 daje nam już złożoność $O(n \cdot t^2)$, ponieważ dla każdego wierzchołka v który nie jest 1 ani pierwszym synem innego wierzchołka i każdego t przeiterujemy się po wszystkich t' aby policzyć DP dla ojca v.

Optymalizacja 2: Nie musimy dla każdego wierzchołka sprawdzać wszystkich możliwych t. W szczególności niezależnie od struktury drzewa, dla t rzędu dwukrotności rozmiaru poddrzewa da się odwiedzić całe drzewo i wrócić spowrotem do wierzchołka startowego.

Przy poprawnej implementacji Optymalizacji 2 złożoność amortyzuje się do $O(n \cdot min(n,t))$.
Dowód: https://ceoi2017.acm.si/files/ceoi2017-solutions-practice.pdf - omówienie zadania Muzeum, na samym dole po "Further optimization".

Uwagi

- Warto wspomnieć, że przy kompresji drzewa nigdy nie usuwamy wierzchołków które nie są liściami, więc odległości pomiędzy wierzchołkami pozostają równe 1.

[3] Celowo nie podałem i jako kolejnego wymiaru DP, ponieważ nie chcemy tego fizycznie pamiętać w kodzie ze względu na pamięć. Wartości będziemy trzymać bezpośrednio w $DP[v][t][stay]$, będą się one poprostu zmieniać wraz ze wzrostem i.

[4] Uwaga: niekoniecznie w tej kolejności. Możemy najpierw pójść do s_i, a potem wrócić do poprzednich $i-1$ synów, żeby na końcu zostać w jednym z nich.

Omówienie zadania **Slider**

Autorem zadania oraz omówienia jest Daniel Olkowski.
Link do treści: https://szkopul.edu.pl/problemset/problem/aglr789jaEUgz2pk9BWToAIW/site

Rozwiązanie

Kluczową obserwacją jest to, że minimalna liczba pól między sliderami - tak by nie się wzajemnie nie biły - to dwa pola.
Zatem możemy zachłannie, umieszczać slidery:
Slider numer 1 na pierwszym polu, slider numer 2 na polu 4, slider numer 3 na polu 7, itd.

Wniosek: Na każdej trójce pól możemy umieścić jeden slider.
Wystarczy zatem podaną liczbę pól podzielić przez 3, by otrzymać maksymalną możliwę liczbę sliderów które możemy umieścić by wzajemnie się nie biły.

Musimy jeszcze zauważyć, że
a. dla liczby pól 1, 2, 3 możemy umieścić jeden slider podczas gdy podzielenie 1/3 daje zero po obcięciu reszty, 2/3 daje 0 po obcięciu reszty, 3/3 daje 1 po obcięciu reszty
b. dla liczby pól 4, 5, 6 możemy umieścić dwa slidery podczas gdy podzielenie 4/3 daje 1 po obcięciu reszty, 5/3 daje 1 po obcięciu reszty, 6/3 daje 2 po obcięciu reszty

Wystarczy, że do liczby pól dodamy 2. Wówczas podzielenie przez 3 - w C++ dzielenie na liczbach całkowitych obcina część ułamkową - da nam odpowiednią ilość sliderów. Dla przypadku a. będzie to 1 slider, dla przypadku b. będą to 2 slidery, i tak dalej.

Przykładowa implementacja

C++

```cpp
#include <iostream>
using namespace std;

int main() {
 long long liczba_pol, wynik;

 cin >> liczba_pol;
 wynik = (liczba_pol+2)/3;
 cout << wynik << endl;

 return 0;
}
```

Omówienie zadania **Bez walki**

Autorem zadania jest Mikołaj Bulge - omówienie Daniel Olkowski.
Link do treści: https://szkopul.edu.pl/problemset/problem/haschIaQ6AzXw4HwUewSJEOt/site

Rozwiązanie

Jeśli w dniu numer 10 chcemy sprzedać akcje to kupić powinniśmy w tym dniu od 1 do 9 w którym były najtańsze.

Wystarczy zatem przejść po wszystkich dniach pamiętając cenę po której mogliśmy najtaniej kupić akcje do tej pory. Jeśli dla danego dnia otrzymujemy większy zysk ze sprzedaży akcji niż w przypadku wcześniejszych dni, to aktualizujemy maksymalny możliwy zysk.

Przykładowa implementacja

C++

```
#include <iostream>
#include <algorithm>
using namespace std;

int main() {
 ios_base::sync_with_stdio(0);
 cin.tie(0);
 cout.tie(0);

 int liczba_dni, i;
 long long cena_akt,cena_min;
 long long max_zysk;

 cin >> liczba_dni;

 cin >> cena_min;
 max_zysk = 0;
 for (i=2; i<=liczba_dni; ++i) {
    cin >> cena_akt;
    max_zysk = max (max_zysk, cena_akt-cena_min);
    cena_min = min (cena_min, cena_akt);
 }

 if ( max_zysk > 0)
    cout << max_zysk << endl;
 else
    cout << "Nie ma zysku, to ci sie nie oplaca" << endl;

 return 0;
}
```

Omówienie zadania **BANANA**

Autorem zadania oraz omówienia jest Maciej Wiśniewski.
Link do treści: https://szkopul.edu.pl/problemset/problem/4Buls0IA8e3DFOXtt63ShAbt/site/

Rozwiązanie

Zadanie polega na zapisaniu liczby w systemie 26-stkowym, z drobną modyfikacją że w tym systemie nie ma zera. Będziemy chcieli odczytać tą liczbę po jednej cyfrze (literze) od najmniej znaczących.

To co byśmy zrobili w normalnym systemie 26-stkowym, to powiedzieli że ostatnią cyfrą (literą) jest $n \, mod \, 26$ (reszta z dzielenia $n/26$), i potem podzielili n przez 26 zaokrąglając w dół, żeby sprawdzić następne cyfry(litery). Ze względu na drobną modyfikację systemu w naszym zadaniu, musimy przed odczytaniem ostatniej cyfry najpierw zmniejszyć n o 1. Czyli całe rozwiązanie można zapisać w 4 krokach:

1. zmniejsz n o 1.
2. dopisz jako kolejną cyfrę (literę) $n \, mod \, 26$.
3. podziel $n/26$ zaokrąglając w dół.
4. jeżeli n jest większe od 0, to powtórz wszystkie kroki.

W zależności od implementacji, być może słowo trzeba będzie na końcu odwrócić.

Przykładowe implementacje

C++

```cpp
#include <bits/stdc++.h>
using namespace std;


string solve(long long nr) {
    if (nr == 1) return "A";

    string s = "";

    while (nr) {
        nr--;
        char c = (nr % 26) + 'A';
        s += c;
        nr /= 26;
    }

    reverse(s.begin(), s.end());
    return s;
}

int main()
{
    ios_base::sync_with_stdio(0), cin.tie(0);

    int tcase;
    cin >> tcase;

    while (tcase--) {
        long long nr;
        cin >> nr;
        cout << solve(nr) << "\n";
    }
    return 0;
}
```

Omówienie zadania **Nieudany szyfr**

Zadanie pochodzi z Facebook Interview, autorem omówienia jest Tomasz Kwiatkowski.
Link do treści: https://szkopul.edu.pl/problemset/problem/sarFyLVwJIS1CC9V7YKV50As/site/

Rozwiązanie

Zadanie rozwiążemy dynamicznie. Niech `dp[i]` oznacza ile słów daje szyfr złożony z pierwszych i cyfr wejścia. Dla zeru znaków mamy 1 sposób – puste słowo, `dp[0] = 1`.

Dalej zastanówmy się, jak obliczyć `dp[i]`. Mamy dwie opcje albo ostatni znak ma kod jednocyfrowy, albo dwucyfrowy. Zatem jeżeli ostatnia cyfra nie jest zerem, to do `dp[i]` dodajemy `dp[i - 1]`. Jeżeli ostatnie dwie cyfry tworzą liczbę z zakresu $[10, 26]$, to do `dp[i]` dodajemy `dp[i - 2]` (oczywiście, jeżeli $i > 1$). Wynik będzie w ostatniej komórce `dp`.

Zauważmy, że tak naprawdę nie potrzebujemy pamiętać wszystkich wartości tablicy `dp`. Wystarczą nam dwie ostatnie. W rozwiązaniu używam dwóch zmiennych – `a` oznaczającej `dp[i - 2]` oraz `b` oznaczającej `dp[i - 1]`. Funkcja `ok_char` zwraca wartość $1/0$ i mówi odpowiednio, czy ostatnie $d+1$ cyfr odpowiada jakiemuś znakowi lub nie.

Przykładowe implementacje

C++

```cpp
#include <iostream>
#include <string>
using namespace std;

const int MOD = 1e9 + 7;

int main()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    string s;
    cin >> s;
    int a = 1, b = 1;
    auto ok_char = [&](int i, int d) {
        return (d == 0 && s[i] > '0') ||
               (d == 1 && s[i - 1] == '1') ||
               (d == 1 && s[i - 1] == '2' && s[i] < '7');
    };
    for (size_t i = 1; i < s.size(); ++i) {
        int c = (ok_char(i, 1) * a + ok_char(i, 0) * b) % MOD;
        a = b;
        b = c;
    }
    cout << b << '\n';
    return 0;
}
```

Python

```python
MOD = 1000000007

def main():
    n = input()
    a = b = 1
    for c1, c2 in zip(n[:-1], n[1:]):
        a, b = b, (a * (c1 == '1' or (c1 == '2' and c2 < '7')) + b * (c2 > '0')) % MOD
    print(b)

main()
```

Omówienie zadania **Nieudany szyfr**

Uwagi

- W C++ znak to też liczba (tzn. np. `'0'` odpowiada jej kodowi ASCII – 48). Dzięki temu nie musimy pamiętać kodów ASCII znaków, wystarczy, że porównujemy zmienne po prostu z danym znakiem.
- W Pythonie też można tak porównywać znaki.

Omówienie zadania **Platforma**

Autorem zadania oraz omówienia jest Mikołaj Bulge.
Link do treści: https://szkopul.edu.pl/problemset/problem/ap1YUQsxZoA9zDP62ps0ufdO/site/

Rozwiązanie

Rozwiązaniem założonym przez autora było utworzenie drzewa przedziałowego punkt–przedział, które umożliwia wykonanie operacji dodania w punkcie oraz którego każdy węzeł zna sumę swojego poddrzewa. W węzłach drzewa początkowo znajduje się wartość 0. Po pojawieniu się na platformie wartości x, należy dodać 1 do x-tego liścia. Teraz, aby znaleźć k-tą najmniejszą wartość spośród dotychczas dodanych do drzewa, należy wykonać poniższy algorytm:

- Ustaw `cnt` $= k$
- Zaczynając od korzenia powtarzaj: jeśli w lewym poddrzewie suma wynosi co najmniej `cnt`, to przejdź do niego. W przeciwnym wypadku odejmij od `cnt` sumę lewego poddrzewa i przejdź do prawego poddrzewa.
- Algorytm zakończ po dotarciu do liścia. Jeśli jest to x-ty liść od lewej, to znaczy, że x jest k-tą najmniejszą wartością z dotychczas wstawionych do drzewa wartości.

W ten sposób po dodaniu każdego elementu można łatwo znaleźć $\frac{n}{2}$-tą najmniejszą wartość (oraz $\frac{n}{2}+1$, jeśli potrzeba).

Alternatywnie do rozwiązania zadania można użyć struktury `ordered_set`, która jest opisana w tym linku.

Oba rozwiązania mają jednakową złożoność obliczeniową, która wynosi $\Theta(n \cdot log_2(n))$.

Omówienie zadania **Czas**

Autorem zadania oraz omówienia jest Daniel Olkowski.
Link do treści: https://szkopul.edu.pl/problemset/problem/vtWmO3zoqWjYMtjjvApkMjsL/site

Rozwiązanie

Suma czasów wykonywanych czynności jest większa od 24 - czasu trwania doby. Ten naddatek ponad 24h to czas który musi być wspólny. Wystarczy zatem zsumować czas wszystkich czynności i odjąć 24.

Przykładowa implementacja

C++

```cpp
#include <iostream>
using namespace std;

int main() {

 int czas_szkoly, czas_snu, czas_dla_siebie;
 int zysk_na_czasie_szkoly_i_snu;

 cin >> czas_szkoly >> czas_snu >> czas_dla_siebie;
 zysk_na_czasie_szkoly_i_snu = czas_szkoly + czas_snu + czas_dla_siebie - 24;

 cout << zysk_na_czasie_szkoly_i_snu << endl;

 return 0;
}
```

Omówienie zadania **Potęgi klucz**

Autorem zadania oraz omówienia jest Daniel Olkowski.
Link do treści: https://szkopul.edu.pl/problemset/problem/d_AZMQVbGBFVqZX0Oz1aArVI/site/

Rozwiązanie

W zadaniu mamy podane dwa kolejne wyrazy pewnego ciągu a^0, a^1, a^2, a^3, a^4, ... Musimy powiedzieć jaki będzie trzeci kolejny.
Na przykład weźmy ciąg 5^0, 5^1, 5^2, 5^3, 5^4, ... czyli 1, 5, 25, 125, 625, ...
Jeśli mamy dane 25 i 125 to jaki będzie kolejny wyraz ciągu?

Kolejne wyrazy są mnożone przez ten sam czynnik. Wystarczy podzielić 125 przez 5 by dowiedzieć się, że wyrazy w naszym ciągu są mnożone przez 5. Zatem w naszym przykładzie kolejny wyraz to będzie $125 * 5$ czyli 625.

Przykładowa implementacja

C++

```cpp
#include <iostream>
using namespace std;

int main() {

 long long pierwsza_liczba, druga_liczba;
 long long iloraz, wynik;

 cin >> pierwsza_liczba >> druga_liczba;
 iloraz = druga_liczba / pierwsza_liczba;
 wynik = druga_liczba*iloraz;

 cout << wynik << endl;

 return 0;
}
```

Omówienie zadania **Asystent**

Autorem zadania i omówienia jest Mikołaj Bulge
Link do treści: https://szkopul.edu.pl/problemset/problem/cn5EG4LiOODQup2AflwBmbQx/site/

Rozwiązanie

Zauważmy, że iloczyn wszystkich elementów poza i-tym, to jest iloczyn wszystkich elementów z przedziału $[1, i-1]$ oraz $[i+1, n]$. Dzięki temu, aby uzyskać wszystkie odpowiedzi wystarczy policzyć zawczasu iloczyny każdego prefiksu oraz każdego sufiksu ciągu z wejścia. Warto pamiętać, aby wszystkie obliczenia wykonywać modulo $10^9 + 7$, aby uniknąć problemów z overflow.

Przykładowe implementacje

C++

```cpp
#include<bits/stdc++.h>
using namespace std;

constexpr int M = 5e5 + 7;
constexpr int mod = 1e9 + 7;

long long t[M];
long long px[M];
long long sx[M];

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);

    int n;

    cin >> n;
    for(int i = 1; i <= n; i++) cin >> t[i];

    px[0] = sx[n+1] = 1;
    for(int i = 1; i <= n; i++) px[i] = (px[i - 1] * t[i]) % mod;
    for(int i = n; i >= 1; i--) sx[i] = (sx[i + 1] * t[i]) % mod;

    for(int i = 1; i <= n; i++) cout << (px[i - 1] * sx[i + 1]) % mod << ' ';

    return 0;
}
```

Python

```python
MOD = 10**9 + 7

def main():
    n = int(input())
    arr = list(map(int, input().split()))
    suf = [1]
    for i in range(n - 1):
        suf.append((suf[-1] * arr[-i - 1]) % MOD)
    pref = 1
    for i in range(n):
        print((suf[-i - 1] * pref) % MOD, end=' ')
        pref = (pref * arr[i]) % MOD
    print()
main()
```

Omówienie zadania **Pomidor**

Zadanie pochodzi z Twitter Interview, autorem omówienia jest Tomasz Kwiatkowski.
Link do treści: https://szkopul.edu.pl/problemset/problem/CWnRF4m5y2WR591HPrIaUcCR/site/

Rozwiązanie

Do rozwiązania zadania pomocne może okazać się drzewo Trie.

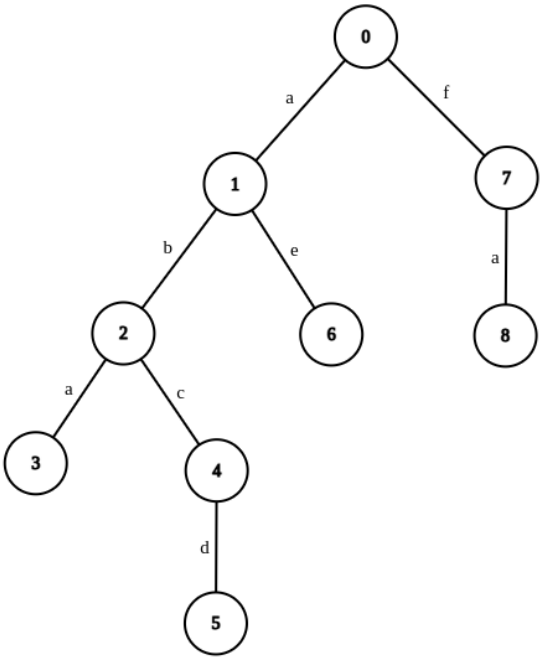
Drzewo Trie – idea

Będziemy konstruować drzewo. Każdy wierzchołek odpowiada pewnemu słowu (lub jego prefiksowi). Jeżeli słowo kończy się w danym wierzchołku, to chcemy to w nim pamiętać. Krawędzie natomiast odpowiadają literom w słowie.

Korzeń drzewa to słowo puste. Bezpośredni sąsiedzi korzenia to słowa długości 1. Kolejni to słowa długości 2, itd.

Idąc krawędzią, do słowa "dodajemy" literkę powiązaną z przechodzoną krawędzią.

Przykładowe drzewo dla słów {`ab`, `aba`, `abcd`, `ae`, `fa`}, może wyglądać następująco:



Wierzchołki 2, 3, 5, 6 oraz 8 powinny pamiętać, że w nich kończą się słowa.

Rozwiązanie wzorcowe

Tworzymy drzewo Trie na bazie słów ze słownika. Odpowiadanie na zapytanie, to odpowiednie zejście w dół drzewa.

Dla danego zapytania, najpierw schodzimy zgodnie z literami zapytania. Jeżeli w pewnym momencie chcemy zejść w dół odpowiednią literą, ale nie ma krawędzi dla tej litery, to znaczy, że nie ma żadnego słowa, które zaczyna się danym zapytaniem – wypisujemy `Pomidor`.

W przeciwnym wypadku, jeżeli dojdziemy do końca słowa, to dalej schodzimy w dół, zawsze wybierając najwcześniejszą (tzn. o najwcześniejszej alfabetycznie literze) krawędź aż nie dojdziemy do wierzchołka, w którym kończy się jakieś słowo.

Dodawanie słowa o długości m do drzewia Trie ma złożoność $O(m \cdot \alpha)$, gdzie α to rozmiar alfabetu. Znalezienie pasującego do zapytania słowa ma złożoność $O(M \cdot \alpha)$, gdzie M to długość znalezionego słowa.

Przykładowe implementacje

C++

```cpp
#include <bits/stdc++.h>
using namespace std;

const int A = 26;

struct Node {
    int next[A];
    bool is_end;
    Node() : is_end(false) { fill(next, next + A, -1); }
};

vector<Node> trie(1);

void add_word(string &s) {
    int v = 0;
    for (auto c : s) {
        if (trie[v].next[c - 'a'] == -1) {
            trie[v].next[c - 'a'] = trie.size();
            trie.push_back(Node());
        }
        v = trie[v].next[c - 'a'];
    }
    trie[v].is_end = true;
}

string find(string &s) {
    int v = 0;
    for (auto c : s) {
        if (trie[v].next[c - 'a'] == -1)
            return "Pomidor";
        v = trie[v].next[c - 'a'];
    }
    string res = s;
    while (!trie[v].is_end) {
        int i = 0;
        while (trie[v].next[i] == -1)
            ++i;
        v = trie[v].next[i];
        res += char('a' + i);
    }
    return res;
}

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    int n, q;
    cin >> n;
    for (int i = 0; i < n; ++i) {
        string s;
        cin >> s;
        add_word(s);
    }
    cin >> q;
    for (int i = 0; i < q; ++i) {
        string s;
        cin >> s;
        cout << find(s) << '\n';
    }
    return 0;
}
```

Python

```python
class TrieNode:
    def __init__(self):
        self.is_end = False
        self.next = {}

class Trie:
    def __init__(self):
        self.root = TrieNode()

    def add_word(self, s):
        v = self.root
        for char in s:
            if not char in v.next:
                v.next[char] = TrieNode()
            v = v.next[char]

        v.is_end = True

    def find(self, s):
        v = self.root
        for char in s:
            if not char in v.next:
                return "Pomidor"
            v = v.next[char]

        res = s
        while not v.is_end:
            char = min(v.next.keys())
            v = v.next[char]
            res += char

        return res

def main():
    n = int(input())
    t = Trie()
    for _ in range(n):
        t.add_word(input())
    q = int(input())
    for _ in range(q):
        print(t.find(input()))

main()
```

Uwagi

- Co prawda, test przykładowy w treści miał posortowane słowa ze słownika, jednak nigdzie w treści nie było powiedziane, że słowa na wejściu są jakkolwiek uporządkowane. Pamiętajmy, by **nie zakładać** rzeczy, których nie ma w treści.

Omówienie zadania **Bajtek nie lubi się powtarzać**

Zadanie pochodzi z VK Cup 2021, autorem treści i omówienia jest Maciej Wiśniewski.
Link do treści: https://szkopul.edu.pl/problemset/problem/X8O2Ff_d5gx9_3LaO8OVheVB/site/

Rozwiązanie

Zauważmy, że są 3 możliwe najlepsze powtarzalności:

1. jeżeli wszystkie litery są takie same, to słowo będzie miało powtarzalność $n-1$ i nie możemy nic zmienić. (np. "aaaaaaaaa")
2. jeżeli istnieje chociaż jedna litera która występuje w tym słowie dokładnie raz, to możemy ją dać na pierwsze miejsce i powtarzalność będzie równa 0. Możemy więc posortować leksykograficznie resztę słowa. (np. "caabbefg")
3. w przeciwnym wypadku istnieje przynajmniej jedna litera która występuje conajmniej 2, ale nie więcej niż $n/2+1$ razy. Możemy więc dać 2 te litery na początek, a potem na zmianę tą literę i jakąś inną, otrzymując powtarzalność 1.

Jedyny nietrywialny przypadek to ten trzeci, możemy go rozbić na 3 mniejsze przypadki:

1. jeżeli najmniejsza leksykograficznie litera występująca w słowie nie występuje więcej niż $n/2+1$ razy to dajemy 2 te litery na początek, resztę sortujemy leksykograficznie i dajemy na zmianę inną literę i tą najmniejszą aż się skończą. (np. "aabacadadefg")
2. w przeciwnym wypadkum, jeżeli w całym słowie występują dokładnie 2 różne litery, dajemy jedną najmniejszą leksykograficznie literę na początek, potem wszystkie drugiego rodzaju i na końcu resztę tego pierwszego. (np. "abbbbbaaaaaaaa").
3. w przeciwnym wypadku, dajemy jedną najmniejszą leksykograficznie literę na początek, potem jedną drugą najmniejszą, potem resztę pierwszej i na końcu całą resztę posortowaną leksykograficznie. (np. "abbbbbaaaaaaaa").

Omówienie zadania **Ramka**

Autorem zadania oraz omówienia jest Tomasz Kwiatkowski.
Link do treści: https://szkopul.edu.pl/problemset/problem/cRyyI_qzWJBH0DWzSsejOWkV/site/

Rozwiązanie

Stworzymy funkcję, która przyjmuje parametr n i wypisuje na przemian $n+1$ plusów oraz n minusów. Wywołamy ją, by uzyskać pierwszą i trzecią linijkę wyjścia.

Do wypisania drugiej linijki wykorzystamy pętlę `for`. Dla każdego znaku wypiszemy najpierw `|`, a następnie dany znak. Na koniec dopiszemy brakujący, ostatni znak `|`.

Do wczytania napisu w C++ skorzystamy z typu `string`.

Przykładowe implementacje

C++

```cpp
#include <iostream>
using namespace std;

void print_border(int n)
{
    for (int i = 0; i < n; ++i)
        cout << "+-";
    cout << "+" << "\n";
}

int main()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    string s;
    cin >> s;

    print_border(s.size());

    for (auto c : s)
        cout << "|" << c;
    cout << "|" << "\n";

    print_border(s.size());
    return 0;
}
```

Python

```python
def print_border(n):
    print("+-" * n + "+")

def solve():
    s = input()

    print_border(len(s))

    for c in s:
        print("|" + c, end = "")
    print("|")

    print_border(len(s))

solve()
```

Uwagi

- W tym przypadku można by dyskutować, czy funkcja `print_border` jest potrzebna. Ale w ogólności wydzielanie fragmentów kodu, który powtarzamy wielokrotnie do oddzielnych funkcji jest dobrą praktyką

Omówienie zadania **Dłuższy dzień**

Autorem zadania oraz omówienia jest Daniel Olkowski.
Link do treści: https://szkopul.edu.pl/problemset/problem/Kz45VU4hb8V8ofkbm8aQtEJ/site

Rozwiązanie

Przede wszystkim musimy sprawdzić w jakiej fazie roku jesteśmy:
1. Mamy najkrótszy dzień w roku 2. Kolejny dzień będzie dłuższy
3. Kolejny dzień będzie krótszy

Dla 1.
Wypisujemy odpowiedni komunikat: "NOC SWIETOJANSKA"

Dla 2.
Wypisujemy kolejny dzień.
Dla 3.
Koljeny najkrótszy dzień będzie symetryczny względem 21 grudnia. Musimy obliczyć ile jest dni do 21 grudnia. Jeśli do 31 grudnia jest 5 dni, to najbliższy dłuższy dzień będzie szóstym dniem po 21 grudnia.
Jak to zrobić? Najwygodniej operować na numerach dniach w roku zamiast na miesiącach i dniach miesiąca.

Przykładowe implementacje

C++

```cpp
#include <iostream>
#include <string>
using namespace std;

int days[12] = {31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
string month_labels[12] = {
    "styczen",
    "luty",
    "marzec",
    "kwiecien",
    "maj",
    "czerwiec",
    "lipiec",
    "sierpien",
    "wrzesien",
    "pazdziernik",
    "listopad",
    "grudzien",
};

int date_to_num(int day, string month)
{
    int res = day;
    for (int i = 0; month_labels[i] != month; ++i)
        res += days[i];
    return res;
}

string num_to_date(int num)
{
    num = (num - 1) % 366 + 1;
    int i = 0, sum = 0;
    for (; sum + days[i] < num; ++i)
        sum += days[i];
    return to_string(num - sum) + " " + month_labels[i];
}
```

```
const int LONGEST = date_to_num(21, "czerwiec");
const int SHORTEST = date_to_num(21, "grudzien");

int main()
{
    int day;
    string month;
    cin >> day >> month;
    int num = date_to_num(day, month);
    if (LONGEST == num) {
        cout << "NOC SWIETOJANSKA!\n";
    } else if (LONGEST < num && num < SHORTEST) {
        cout << num_to_date(2 * SHORTEST - num + 1) << '\n';
    } else {
        cout << num_to_date(num + 1) << '\n';
    }
    return 0;
}
```

Python

```
days = [31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
month_labels = [
    "styczen",
    "luty",
    "marzec",
    "kwiecien",
    "maj",
    "czerwiec",
    "lipiec",
    "sierpien",
    "wrzesien",
    "pazdziernik",
    "listopad",
    "grudzien",
]


def date_to_num(date):
    day, month = date.split()
    return sum(days[: month_labels.index(month)]) + int(day)


def num_to_date(num):
    num = (num - 1) % sum(days) + 1
    month_nr = 0
    while sum(days[: month_nr + 1]) < num:
        month_nr += 1
    return f"{str(num - sum(days[: month_nr]))} {month_labels[month_nr]}"


LONGEST = date_to_num("21 czerwiec")
SHORTEST = date_to_num("21 grudzien")


def main():
    date = input()
    num = date_to_num(date)
    if LONGEST == num:
        print("NOC SWIETOJANSKA!")
    elif LONGEST < num < SHORTEST:
        print(num_to_date(2 * SHORTEST - num + 1))
    else:
        print(num_to_date(num + 1))


main()
```

Omówienie zadania **Mistrz Programowania 2024**

Autorem zadania jest Jerzy Fronczyk, omówienie - Daniel Olkowski.
Link do treści: https://szkopul.edu.pl/problemset/problem/Pa5p1j1wDlfh53BYYBRYxNuk/site

Rozwiązanie

Zadanie ma 3 wyzwania: Sposób przechowywania danych, Niestandardowe posortowanie oraz Wypisanie rankingu tak by osoby z tą samą oceną miały identyczny numer. Omówiona zostanie implementacja C++

Sposób przechowywania danych

Najwygodniej dane dotyczące pojedynczego użytkownika (identyfikator, punkty, ...) trzymać we strukturze (struct) C++. Następnie towrzymy vector (tablicę) której pojedynczym elementem jest właśnie zdefiniowany struct. Teraz sortując vector sortujemy jednocześnie całość informacji dotyczących pojedynczego użtykownika.

Niestandardowe sortowanie

C++ posiada możliwośc podanie funkcji definiującej który z obiektów sortowanego vectora jest mniejszy - poiwnien być wcześniej w tablicy. U nas poniżej to funkcja *CzyZawdodnikAZLewejStrony*

Wypisanie rankingu

Generowanie fair rankingu - osoby z tą samą oceną mają ten sam numer rankingu - najłatwiej wykonać poprzez 2 zmienne. Globalny numer uczestnika oraz pozycja w rankingu. Wypisując posortowanych uczestników, sprawdzamy czy aktualny użytkownik ma tą samą ocenę co poprzedni. Jeśli TAK jego wypisywana pozycja nie ulega zmianie. Jeśli NIE to wypisywana pozycja ma globalny numer uczestnika.

Przykładowe implementacje

C++

```
#include <bits/stdc++.h>
using namespace std;

const int LICZBA_RUND = 5;
const int MAX_PUNKTOW_RUNDA = 500;

struct typ_zawodnik {
 string identyfikator;
 vector<int> wyniki_rund;
 int w_ilu_rundach;
 int ile_maksow;
 int suma_punktow;
};

bool CzyZawdodnikAZLewejStrony (const typ_zawodnik &zawodnikA, const typ_zawodnik &zawodnikB) {
 if ( zawodnikA.w_ilu_rundach != zawodnikB.w_ilu_rundach )
    return zawodnikA.w_ilu_rundach > zawodnikB.w_ilu_rundach;
 if ( zawodnikA.ile_maksow != zawodnikB.ile_maksow )
    return zawodnikA.ile_maksow > zawodnikB.ile_maksow;
 if ( zawodnikA.suma_punktow != zawodnikB.suma_punktow )
    return zawodnikA.suma_punktow > zawodnikB.suma_punktow;
 return zawodnikA.identyfikator < zawodnikB.identyfikator;
}

bool CzyZawodnicyTacySami (const typ_zawodnik &zawodnikA, const typ_zawodnik &zawodnikB) {
 if ( zawodnikA.w_ilu_rundach != zawodnikB.w_ilu_rundach )
    return false;
 if ( zawodnikA.ile_maksow != zawodnikB.ile_maksow )
    return false;
 if ( zawodnikA.suma_punktow != zawodnikB.suma_punktow )
    return false;
 return true;
}
int main() {
```

```
 ios_base::sync_with_stdio(0);
 cin.tie(0);
 cout.tie(0);

 int ile_uczestnikow, i, j;
 int akt_numer_zawodnika, numer_globalny_zawodnika;
 vector <typ_zawodnik> zawodnicy;

 cin >> ile_uczestnikow;
 zawodnicy.resize(ile_uczestnikow);

 for (i=0; i<ile_uczestnikow; ++i) {
    cin >> zawodnicy[i].identyfikator;
    zawodnicy[i].wyniki_rund.resize(LICZBA_RUND);
    zawodnicy[i].suma_punktow = zawodnicy[i].w_ilu_rundach = zawodnicy[i].ile_maksow = 0;
    for (j=0; j<LICZBA_RUND; ++j) {
       cin >> zawodnicy[i].wyniki_rund[j];
       zawodnicy[i].suma_punktow += zawodnicy[i].wyniki_rund[j];
       if ( zawodnicy[i].wyniki_rund[j] == MAX_PUNKTOW_RUNDA )
          ++zawodnicy[i].ile_maksow;
       if ( zawodnicy[i].wyniki_rund[j] > 0 )
          ++zawodnicy[i].w_ilu_rundach;
    }
 }

 sort (zawodnicy.begin(), zawodnicy.end(), CzyZawdodnikAZLewejStrony );

 akt_numer_zawodnika = numer_globalny_zawodnika = 1;
 for (i=0; i<ile_uczestnikow; ++i) {
    cout << akt_numer_zawodnika << " " << zawodnicy[i].identyfikator << " " << zawodnicy[i].
    suma_punktow << " ";
    for (j=0; j<LICZBA_RUND; ++j)
       cout << zawodnicy[i].wyniki_rund[j] << " ";
    cout << endl;
    ++numer_globalny_zawodnika;
    if ( numer_globalny_zawodnika > ile_uczestnikow )
       break;
    if ( CzyZawodnicyTacySami (zawodnicy[i], zawodnicy[i+1]) == false )
       akt_numer_zawodnika = numer_globalny_zawodnika;
 }


 return 0;
}
```

Python

```python
def main():
    n = int(input())
    to_sort = []
    for _ in range(n):
        row = input().split()
        """
        sort order:
        a. number of non-zero rounds
        b. number of maxed round
        c. total points
        d. lexicographically
        """
        to_sort.append(
            (
                -(5 - row.count("0")),
                -row.count("500"),
                -sum(map(int, row[1:])),
                row[0],
                row[1:],
            )
        )

    to_sort.sort()
    nr = 1
    for i in range(n):
        print(
            nr,
            to_sort[i][3],
            sum(list(map(int, to_sort[i][4]))),
            " ".join(to_sort[i][4]),
        )
        if i < n - 1 and to_sort[i + 1][:3] != to_sort[i][:3]:
            nr = i + 2

main()
```

Uwagi

Własne sortowanie to coś co jest bardzo potrzebne w realnym programowaniu - na przykład sklepy internetowe gdzie sortujemy po cenie, popularności, itp. Ale też przydaje się w konkursach i olimpidach informatycznych - zadanie Metro z Olimpiady Informatycznej Juniorów.

Omówienie zadania **W trosce o środowisko**

Autorem zadania oraz omówienia jest Tomasz Kwiatkowski.
Link do treści: https://szkopul.edu.pl/problemset/problem/Ro8p1Q32GyFmOc2wLF6GaCV5/site/

Rozwiązanie

Rozwiązanie wolne

Dla danego słowa-zapytania będziemy przechodzić się po tekście i utrzymywać, której aktualnie literki ze słowa-zapytania szukamy. Jeżeli po przejściu całego tekstu zmienna będzie wskazywała poza słowo-zapytanie, to odpowiedź dla danego słowa-zapytania to `TAK`.

Złożoność tego podejścia to $O(nA+S)$, gdzie A to długość tesktu, S to suma długości słów-zapytań. Takie rozwiązanie przechodziło dwa pierwsze podzadania.

Rozwiązanie wzorcowe

Aby przyśpieszyć rozwiązanie, będziemy równolegle wyszukiwać wszystkie słowa-zapytania. Dla każdego trzymamy indeks znaku, który aktualnie chcemy z niego znaleźć (na początku 0). Dla każdej literki trzymamy również listę indeksów słów-zapytań, które szukają danej literki. Przechodząc tekst znamy, które słowa-zapytania aktualnie szukają danej literki z tekstu. Zwiększamy indeks szukanego znaku dla tych słów-zapytań oraz aktualizujemy listę wrzucając do list odpowiednich znaków indeksy tych słów. Aby odpowiedzieć na zapytania, podobnie jak w rozwiązaniu wolnym, będziemy patrzyli, czy szukany indeks wskazuje poza słowo.

Złożoność tego podejścia to $O(A+S)$. Każdy znak z tekstu przejdziemy raz oraz każdy znak z każdego słowa przejrzymy dokładnie raz.

Przykładowe implementacje

Python

```python
def main():
    s = input()
    n = int(input())

    message = [""] * n
    curr_pos = [0] * n
    events = [[] for _ in range(60)]
    for i in range(n):
        message[i] = input()
        events[ord(message[i][0]) - 65].append(i)

    for c in s:
        tmp = events[ord(c) - 65]
        events[ord(c) - 65] = []
        for msg_ind in tmp:
            curr_pos[msg_ind] += 1
            if curr_pos[msg_ind] >= len(message[msg_ind]):
                continue
            next_c = message[msg_ind][curr_pos[msg_ind]]
            events[ord(next_c) - 65].append(msg_ind)

    for pos, msg in zip(curr_pos, message):
        if pos == len(msg):
            print("TAK")
        else:
            print("NIE")

main()
```

C++

```cpp
#include <iostream>
#include <string>
#include <vector>
using namespace std;

const int MAXN = 5e5;
const int ALPHA = 60;

string message[MAXN];
size_t curr_pos[MAXN];
vector<int> events[ALPHA];

int main()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    string s;
    cin >> s;
    int n;
    cin >> n;
    for (int i = 0; i < n; ++i) {
        cin >> message[i];
        events[message[i][0] - 'A'].push_back(i);
    }

    for (auto c : s) {
        vector<int> tmp = events[c - 'A'];
        events[c - 'A'].clear();
        for (auto msg_ind : tmp) {
            ++curr_pos[msg_ind];
            if (curr_pos[msg_ind] >= message[msg_ind].size())
                continue;
            char next_c = message[msg_ind][curr_pos[msg_ind]];
            events[next_c - 'A'].push_back(msg_ind);
        }
    }

    for (int i = 0; i < n; ++i)
        cout << (curr_pos[i] == message[i].size() ? "TAK" : "NIE") << '\n';
    return 0;
}
```

Omówienie zadania **Staw**

Autorem zadania oraz omówienia jest BledDest
Autorem polskiej treści jest Maciej Wiśniewski.
Link do treści: https://szkopul.edu.pl/problemset/problem/TVKTLymAkl3LuB3zZVHkBqqZ/site/

Rozwiązanie

https://codeforces.com/blog/entry/101790 – zadanie 1661E Narrow Components