Omówienie zadań z Turnieju nr 29 (dla początkujących)

Pola Marciniak

Palindromiczny Żabuś

W tym zadaniu chcemy sprawdzić, czy wczytane słowo s jest anagramem z nie więcej niż k błędami.

Mówimy, że anagram ma błąd, jeśli i-ta litera od lewej strony słowa s jest różna od i-tej litery od prawej strony (czyli gdy $s[i] \neq s[n-i-1]$, przy czym pozycje numerujemy od 0). Do błędów nie liczymy sytuacji, w których jedna z porównywanych liter jest znakiem pasującym do wszystkiego $*$, ani przypadków, gdy porównujemy litery S i Z.

Wystarczy przejść pętlą po pierwszej połowie słowa (aby każdą parę odpowiadających sobie pozycji sprawdzić dokładnie jeden raz), sprawdzając, czy litery na pozycjach i oraz $n-i-1$ pasują do siebie, i zliczać błędy. Na koniec należy porównać liczbę błędów z wartością k.

Aby uwzględnić porównywanie liter S i Z oraz specjalnego znaku $*$, warto napisać funkcję, która zwróci wartość $false$, jeśli na porównywanych pozycjach występuje błąd, oraz $true$ w przeciwnym przypadku. Wówczas otrzymamy przejrzystą implementację pętli zliczającej błędy.

Przykładowa funkcja, która to implementuje może wyglądać tak:

```
bool same(char a, char b) {
    if (a == 'Z')
        a = 'S';
    if (b == 'Z')
        b = 'S';
    if (a == '*' || b == '*')
        return true;
    if (a == b)
        return true;
    return false;
}
```

Wtedy do zliczenia błędów wystarczy prosta pętla korzystająca z napisanej funkcji:

```
for(int i = 0; i < n/2; i++) if(!same(s[i], s[n-i-1])) count++;
```

Sopelkowo

W tym zadaniu chcielibyśmy wyrównać poziom gór do pewnej całkowitej wartości H tak, aby koszt wyrównania był jak najmniejszy.

Pierwszym podejściem, za które można było otrzymać część punktów, było sprawdzenie wyniku dla każdej możliwej wartości H i wybranie minimalnej z otrzymanych wartości.

Dość intuicyjne jest, że wartością minimalizującą wynik jest **średnia arytmetyczna ciągu** podanego na wejściu - liczba, która jest "najbliżej" do każdej wartości ciągu. W zadaniu należało wyrównać góry do wysokości, która jest liczbą całkowitą, więc należało sprawdzić wynik dla wysokości $\lfloor \sum_{i=1}^{n} h_i \rfloor$ (zwykłe dzielenie w C++ zaokrągla w dół) oraz $\lfloor \sum_{i=1}^{n} h_i \rfloor + 1$ i wypisać mniejszy z nich.

Obserwację o średniej można udowodnić. Zapiszmy koszt jako sumę kwadratów różnic elementów z ciągu oraz pewnej wartości H, która będzie tę sumę minimalizować. Niech $\overline{h} := \frac{\sum_{i=1}^{n} h_i}{n}$ będzie średnią arytmetyczną wysokości gór.

$$\sum_{i=1}^{n}(h_i - H)^2 = \sum_{i=1}^{n}(h_i - H - \overline{h} + \overline{h})^2 = \sum_{i=1}^{n}((h_i - \overline{h}) + (\overline{h} - H))^2$$

Do sumy dodaliśmy i odjeliśmy średnią, więc wynik się nie zmienił. Teraz możemy podnieść sumę do kwadratu.

$$\sum_{i=1}^{n}((h_i - \overline{h})^2 + 2(h_i - \overline{h})(\overline{h} - H) + (\overline{h} - H)^2) = \sum_{i=1}^{n}(h_i - \overline{h})^2 + \sum_{i=1}^{n}2(h_i - \overline{h})(\overline{h} - H) + \sum_{i=1}^{n}(\overline{h} - H)^2$$

Pierwsza suma jest niezależna od wyboru wartości H, więc możemy ją na chwilę pominąć, w drugiej możemy wyjąć przed sumę $2(\overline{h} - H)$ przed sumę. Trzeci czynnik upraszcza się do $\sum_{i=1}^{n}(\overline{h} - H)^2 = n \cdot (\overline{h} - H)^2$. Minimalizujemy więc wartość:

$$2(\overline{h} - H)\sum_{i=1}^{n}(h_i - \overline{h}) + n(\overline{h} - H)^2$$

Patrzymy na składnik pierwszej sumy, czyli suma różnic $(h_i - \overline{h})$ upraszcza się do 0:

$$\sum_{i=1}^{n}(h_i - \overline{h}) = \sum_{i=1}^{n} h_i - \sum_{i=1}^{n} \frac{\sum_{i=1}^{n} h_i}{n} = \sum_{i=1}^{n} h_i - n \cdot \frac{\sum_{i=1}^{n} h_i}{n} = \sum_{i=1}^{n} h_i - \sum_{i=1}^{n} h_i = 0$$

Minimalizowana wartośc to w takim razie:

$$2(\overline{h} - H) \cdot 0 + n(\overline{h} - H)^2 = n(\overline{h} - H)^2$$

Ta wartość jest najmniejsza dla $H = \overline{h}$, czyli optymalną wartością minimalizującą sumę jest średnia arymetyczna wysokości gór.

Lentilky

Dany jest stos kamieni w dwóch kolorach: czerwonych i niebieskich. Gracze na przemian zdejmują od 1 do k kamieni, wygrywa ten, który zdejmie ostatni czerwony kamień. Mamy zdecydować, który z nich wygra, jeśli obydwaj grają optymalnie.

Kluczowe obserwacje:

1. Na przebieg gry nie mają wpływu wszystkie niebieskie kamienie, które znajdują się **pod ostatnim czerwonym kamieniem** (czyli pod tym o którego zdjęcie walczą Żwirek i Muchomorek, jest to czerwony kamień najdalej od szczytu stosu). Możemy je więc zignorować i patrzeć na stos, na którego dnie leży czerwony kamień.
2. Jak już zignorujemy wszystkie niebieskie kamienie pod ostatnim czerwonym kamieniem, to **kolory kamieni przestają mieć znaczenie** - gra redukuje się do następującej gry: *Gracze na przemian zdejmują od* 1 *do k kamieni ze stosu, wygrywa ten, który zdejmie ostatniego kamienia.* Jest to znany, najprostszy wariant gry *Nim* na jednym stosie.

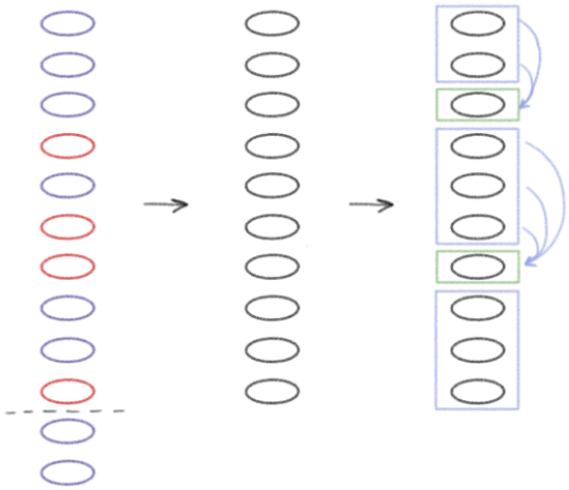
Niech N będzie wysokością stosu kamieni, a k maksymalną liczbą kamieni, które może zdjąć w jednym ruchu gracz (minimalna liczba to oczywiście 1). **Jak wygląda optymalna strategia gry?**

Jeśli $N \leqslant k$, wówczas pierwszy gracz może od razu wygrać. Nazwijmy te pozycje *wygrywającymi*. Jeśli $N = k + 1$, wówczas niezależnie jaki ruch wykona pierwszy gracz, drugi może skończyć grę w kolejnym ruchu (zabierając wszystkie pozostałe kamienie ze stosu). Nazwijmy tę pozycję *przegrywającą*.

Pozycje *wygrywające* to będą takie, z których zaczynając zawsze można wygrać, *przegrywające* to te, z których zaczynając przegramy (przy optymalnej strategii obu graczy).

Co dla większych wartości N? Zauważmy, że dla $k + 1 < N \leqslant 2 \cdot (k + 1)$ gracz pierwszy może zdjąć $N - (k + 1)$ kamieni, zostawiając $(k + 1)$ kamieni na stosie tak, aby w kolejnym ruchu gracz drugi zaczął od pozycji *przegrywającej*. Wszystkie te pozycje są więc *wygrywające*. Dla $N = 2(k + 1)$ zdejmując dowolną liczbę kamieni doprowadzimy do tego, że przeciwnik rozpocznie z pozycji wygrywającej.

Można więc wywnioskować, że jeśli początkowa liczba kamieni na stosie N jest podzielna przez $k + 1$, wygrywa gracz drugi, w przeciwnym przypadku gracz pierwszy zawsze wygrywa (oczywiście zakładając, że gra optymalnie). Przez *początkową liczbę kamieni* N mamy oczywiście na myśli liczbę kamieni powyżej ostatniego czerwonego kamienia.



Rysunek 1: Przykładowa gra dla k = 3 i narysowanego po lewej stronie stosu. Na ostatnim stosie niebieskim kolorem oznaczono pozycje wygrywające, a zielonymi - przegrywające.

Pomysłowy Dobromir

To jest zadanie implementacyjne. Należy w nim wczytać N równań chemicznych i sprawdzić, czy każde z nich jest zrównoważone.

Równanie chemiczne uznajemy za zrównoważone, jeśli dla każdego pierwiastka liczba jego atomów po lewej stronie równania jest równa liczbie atomów po stronie prawej.

Liczbę atomów danego pierwiastka obliczamy jako sumę iloczynów współczynnika przed cząsteczką i liczby atomów tego pierwiastka w danej cząsteczce.

Aby efektywnie zliczać wystąpienia atomów, należy przygotować tablicę zliczającą rozmiaru 26. W komórce 0 zliczamy wystąpienia literki 'A', w komórce 1 - literki 'B' i tak dalej, aż do 'Z'. Aby przeliczyć literę `lit` na jej indeks w tablicy, należy od `lit` odjąć literkę `'A'`. Wówczas otrzymamy ich odległość w reprezentacji ASCII (w której obie litery są przechowywane jako liczby, ułożone kolejno alfabetycznie).

Przykładowo, zwiększenie licznika literki `lit` o 1, to: `T[lit - 'A']++;`.

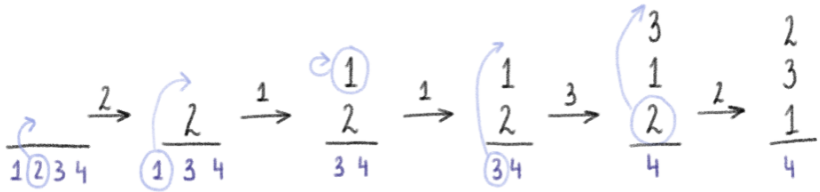
W zadaniu należało jeszcze uzwględnić cyfry. Aby dowiedzieć się, czy czytana literka `lit` jest cyfrą, możemy wykonać podobne przekształcenie: `int liczba = lit - '0';`, a następnie sprawdzając czy wynik jest liczbą: `if(liczba >= 0 && liczba <= 9)`.

Mogiłkowo

Wirt i Greg przenoszą na wierzch stosu zamawiane kolejno trumny, podnosząc przy tym wszystkie trumny znajdujące się powyżej aktualnie przenoszonej trumny. Celem jest zminimalizowanie sumy wag wszystkich trumien, które muszą zostać podniesione w trakcie realizacji zamówień.

Niech d będzie dowolnym dniem, w którym zamówiono trumnę t, a d' poprzednim dniem, w którym trumna t była zamówiona. W dniu d Wirt i Greg nie mogli uniknąć podniesienia wszystkich trumien zamówionych pomiędzy dniem d' i d. Jeśli trumna w ogóle nie była jeszcze zamówiona, wówczas musza ponieść wszystkie zamówione trumny. Tak więc łączny koszt jest co najmniej taki, jak suma wag trumien zamówionych pomiędzy wszystkimi takimi parami dni. Jeśli uda nam sie otrzymać taki koszt, to będzie on optymalny.

Aby ułożyć trumny w optymalnej konfiguracji i policzyć sumę podniesionych wag, można wykonać symulację stosu trumien. Na początku uznajemy, że wszystkie trumny są poza stosem (jeśli nie zostaną zamówione, to nie mają wpływu na końcowy wynik, można mysleć, jakby ich nie było). Następnie przy zamówieniu trumny sprawdzamy, która od szczytu jest ona na stosie, do wyniku dodajmy wszystkie wagi trumien, które leżą na stosie wyżej spychamy je o jedno miejsce dalej, a aktualnie zamawiana trumna ląduje na szczycie stosu. Jeśli trumna leżała poza stosem, to dodajemy do wyniku wagi wszystkich trumien leżących na stosie.



Rysunek 2: Symulacja zamówień dla testu przykładowego: 2 1 1 3 2. Niebieskie trumny nie zostały jeszcze nigdy zamówione, więc znajdują się pod stosem. Przy ostatnim zamówiniu należy podnieść wszystkie trumny powyżej trumny 2, po odstawieniu spadną o jedno miejsce na stosie, a zamówiona trumna znajdzie się na szczycie.

Wysokość na stosie każdej trumny możemy zapamiętać w tablicy. Sprawdzając trumnę t, patrzymy na jej wartość `stos[t]` w tablicy i dla każdej trumny i spełniającej `stos[i] < stos[t]` (pomijając te poza stosem) zwiększamy wartość w tablicy `order`, dodając wagę `waga[i]` do wyniku.

Zbanowany bez powodu

Dla danego słowa, przedziałów i permutacji p należy znaleźć najmniejszy indeks i, dla którego po zakryciu liter na pozycjach $p_1, \ldots, p_i$, w każdym z podanych przedziałów żadna litera nie występuje więcej niż raz.

Część punktów otrzymywała symulacja opisanego w treści zadania procesu. Złożoność czasowa tego rozwiązania wynosi $O(QN^2)$.

Troche więcej punktów trzymywała symulacja, korzystająca dodatkowo z sum prefiksowych. Dla każdej litery budujemy tablicę pref, w której element na pozycji i przechowuje liczbę wystąpień tej litery od początku słowa do i-tej litery (włącznie). Na przykład, dla słowa `abcaab` i litery `a`, tablica `pref` będzie miała postać: $[1, 1, 1, 2, 3, 3]$. W czasie $O(1)$ możemy określić liczbę wystąpień danej litery w przedziale od L-tej do R-tej litery w oryginalnym słowie, licząc: $\texttt{pref}[R] - \texttt{pref}[L-1]$.

Symulacja z szybkimi zapytaniami ma złożoność: $O\big((N^2 + Q) \cdot \Sigma\big)$ gdzie $\Sigma = 26$ to rozmiar alfabetu.

Aby zdobyć wszystkie punkty, można zauważyć następującą własność: jeśli lista kontaktów Johnny'ego (czyli dane słowo) spełnia warunki po i-tym zablokowaniu, będzie je również spełniać po każdym zablokowaniu. Dzięki temu możemy wykorzystać wyszukiwanie binarne, w poszukiwaniu najmniejszego takiego i, dla którego spełnione są warunki. W każdym kroku wyszukiwania binarnego w $O(N \cdot \Sigma)$ zakrywamy zablokowane kontakty (literki w słowie) i w $O(Q \cdot \Sigma)$ sprawdzamy poprawność przedziałów.

Złożoność czasowa tego rozwiązania wynosi: $O\big((N \log N + Q) \cdot \Sigma\big)$

Kombinatoryczny Żabuś

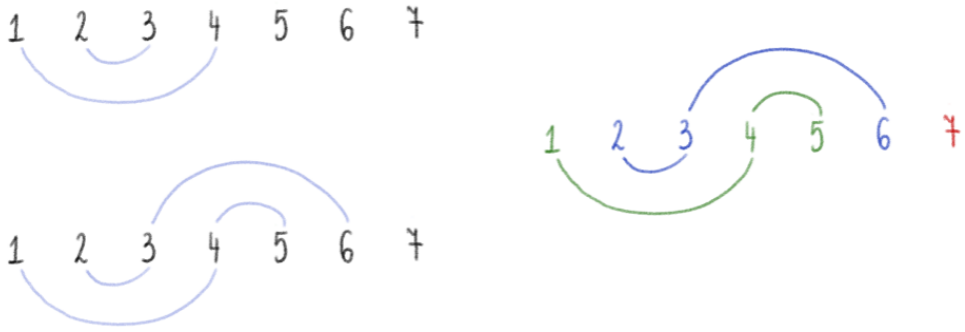
Zauważmy, że jeśli w słowie nie występuje żaden palindrom, to każda z N liter może zostać wybrana niezależnie na K sposobów, co daje K^N możliwych słów.

Jeśli na przedziale $[l, r]$ ma wystąpić palindrom, wówczas litery na symetrycznych pozycjach względem środka muszą być identyczne. Oznacza to, że litery na pozycjach l i r są jednakowe, podobnie jak litery na pozycjach $l + 1$ i $r - 1$, i tak dalej.

Stwórzmy zatem graf, w którym:

- Wierzchołki odpowiadają pozycjom od 1 do N w słowie.
- Jeśli podsłowo od pozycji l do r jest palindromem, łączymy krawędziami pary wierzchołków: (l, r), $(l + 1, r - 1)$, $(l + 2, r - 2)$, itd.

Krawędź między dwoma wierzchołkami oznacza, że na połączonych pozycjach musi wystąpić ta sama litera. W efekcie każda **spójna składowa** grafu reprezentuje zbiór pozycji, które muszą zawierać tę samą literę.



Rysunek 3: Przykładowy graf dla $N = 7$ z palindromami na podsłowach $[1, 4]$ i $[3, 6]$.

Możemy więc zliczyć ile jest spójnych składowych (BFSem albo DFSem) - niech to będzie C i zwrócić K^C - liczbę kolorowań spójnych grafu K kolorami.

Przykładowo, tak wyglądają spójne kolorów w grafie stworzonym dla przykładu podanego w treści zadania:

Rysunek 4: Przykładowy graf dla $N = 3$ z palindromami na podsłowach $[1, 3]$ i $[2, 5]$.