Omówienie zadań z Turnieju 26 (dla początkujących)

Karol Pokorski

1 Wypisywanie drzewa poziomami

To było typowe zadanie grafowe (które można rozwiązać dowolnym algorytmem na przeszukiwanie grafu, np. DFS lub BFS), ale do rozwiązania można było dojść nie znając żadnego algorytmu grafowego. Przedstawimy poniżej właśnie takie rozwiązanie (nie używając nadmiernie języka teorii grafów).

Ciąg wszystkich wierzchołków odległych o dokładnie d od wierzchołka numer 1 (od korzenia) nazwiemy d-tą warstwą. Znamy zerową warstwę (jest na niej jedynie wierzchołek numer 1).

Załóżmy, że znamy listę wszystkich wierzchołków na d-tej warstwie. Chciałoby się na tej podstawie wyznaczyć listę wszystkich wierzchołków na kolejnej, $(d+1)$-szej warstwie. Są na niej dokładnie te wierzchołki, które są końcami krawędzi wychodzących z d-tej warstwy Konieczne jest jednak szybkie znalezienie tych wierzchołków, najlepiej w czasie proporcjonalnym do liczby przejrzanych krawędzi (a nie do liczby wszystkich wierzchołków) – wtedy łączny czas rozwiązania będzie liniowy, bo każda krawędź i każdy wierzchołek zostaną przejrzane jeden raz.

Wystarczy zatem utworzyć listy wierzchołków, które są końcami krawędzi wychodzących z każdego wierzchołka (tzw. listy sąsiedztwa). Może to być utrzymywane w postaci tablicy struktur `vector<int>` (po jednym vectorze na utrzymanie listy sąsiadów każdego wierzchołka, każda liczba na tym vectorze to po prostu numer wierzchołka, do którego prowadzi krawędź). Wtedy łączna zużyta pamięć jest proporcjonalna do liczby krawędzi, które są utrzymywane na vectorach plus niewielki narzut proporcjonalny do liczby wierzchołków na utrzymanie np. pustych vectorów.

W ten sposób otrzymaliśmy algorytm przeszukiwania wszerz (BFS).

2 Gra w życie

To typowe zadanie implementacyjne. Kluczowa uwaga w tym zadaniu była zawarta w sekcji wyjście: jeżeli odpowiedź jest większa niż 100, należy wypisać `NIE`.

Stać nas więc na zasymulowanie stu kroków przekształceń, zgodnie z treścią zadania (plansza ma przecież wymiary tylko 15×15). Planszę można utrzymywać w postaci tablicy napisów (tablicy zmiennych typu `string`). Próba używania jednej tablicy może skończyć się łatwą pomyłką – najlepiej mieć osobną tablicę napisów na poprzedni stan i osobną tablicę na następny stan planszy już po przekształceniu. Wtedy nie okaże się, że przy analizie liczby aktywnych sąsiednich komórek odczytamy już częściowo zmieniony stan.

Zamiast robić osiem osobnych instrukcji sprawdzających kolejny kierunek, dobrze jest zrobić sobie tablicę wektorów (w sensie matematycznym/fizycznym) określających możliwe ruchy (np. $(+1,+1)$ lub $(-1,0)$) względem sprawdzanego pola. Wtedy zliczanie aktywnych sąsiednich komórek wokół zadanej można zrealizować prostą pętlą, co pozwala uniknąć ewentualnych pomyłek i zachować czystość implementacji.

3 Okup

W tym zadaniu najlepiej stworzyć tablicę zliczającą rozmiaru 26. W komórce numer 0 utrzymujemy liczbę liter `a`, w komórce numer 1 liczbę liter `b`, ..., w komórce numer 25 liczbę liter `z`. W C++, żeby przeliczyć literę na jej pozycję w alfabecie wystarczy odjąć od znaku (który wewnętrznie jest przechowywany jako liczba – kod ASCII) kod znaku `a` (nie trzeba wiedzieć ile on wynosi): wystarczy napisać `int indeks_alfabetu = znak - 'a';`.

Każda potrzebna litera zwiększa odpowiednią komórkę tablicy zliczającej o 1, zaś każda posiadana litera zmniejsza tę komórkę o 1. Na końcu wystarczy zsumować (pętlą od 0 do 25) dodatnie wartości tablicy zliczającej (jeżeli wartość jest ujemna, to danej litery mamy więcej niż potrzeba, ale nie zmniejsza to naszych potrzeb w stosunku do innych liter).

4 Liczby rosnące

To zadanie można zrobić na co najmniej dwa proste sposoby.

Na początek zauważamy, że rozwiązanie, które po prostu sprawdza czy liczby $N+1$, $N+2$, $N+3$ itd. są rosnące, aż znajdzie jakąś, jest nieakceptowalne czasowo. W najgorszym razie, dziura między kolejnymi rosnącymi liczbami z dopuszczalnego zakresu jest rzędu 100 milionów, a sprawdzenie, czy liczba jest rosnąca nie jest darmowe (można tutaj systematycznie odczytywać ostatnią cyfrę używając operacji modulo 10 i odcinać ostatnią cyfrę dzieląc przez 10 z zaokrągleniem w dół).

Temu rozwiązaniu niewiele jednak brakuje do poprawnego rozwiązania. Zauważmy, że problem pojawia się dopiero przy liczbach ośmio- i dziewięciocyfrowych. A wśród nich są tylko trzy liczby rosnące: wystarczy sprawdzić je osobno (czyli zrobić hybrydę wyżej opisanego rozwiązania brutalnego ze sprawdzeniem czy liczba jest większa niż na przykład 10 milionów – jeżeli tak, to sprawdzić, która z odpowiedzi `12345678`, `23456789`, `123456789` lub `NIE`, powinna zostać wypisana na wyjście. W pozostałych przypadkach (dla liczb co najwyżej siedmiocyfrowych), zaproponowany wcześniej program będzie dostatecznie szybki.

Innym pomysłem jest zauważyć, że liczb rosnących jest tylko $2^9 = 512$. Istotnie, każda cyfra może wystąpić w liczbie albo zero albo jeden raz, a kolejność występujących cyfr jest dokładnie jedna (zgodnie z posortowaniem od najmniejszej do największej). Możliwe jest więc wygenerowanie wszystkich liczb rosnących i naiwne sprawdzenie, która z nich jest najmniejszą, większą niż N.

Wygenerowanie liczb rosnących można zrealizować brzydko (ale skutecznie) ciągiem dziewięciu pętli zagnieżdżonych od 0 do 1 włącznie (każda pętla ustala ile razy występuje w liczbie cyfra $1, 2, \ldots, 9$), rekurencją (przeszukiwaniem z nawrotami), która symuluje działanie tych dziewięciu pętli w dużo czystszy implementacyjnie sposób lub z użyciem masek bitowych (pętla od 0 do 511 i odczytanie zapisu dwójkowego każdej liczby: jeżeli na i-tej pozycji znajduje się jedynka, to cyfra i występuje w liczbie rosnącej, a jeżeli jest tam zero, to cyfry i w liczbie rosnącej nie ma – przykładowo $13_{(10)} = 1011_{(2)}$, a więc w liczbie rosnącej byłyby cyfry 1, 2 oraz 4, czyli byłaby to liczba 124).

5 Daltonista

Oczywiście zadanie polegało na przekonwertowaniu kodu koloru RGB w hex (tak jak podaje się np. stylizując jakiś element w CSS) na ludzki zapis słowny.

Zacznijmy od tego, co byłoby złe (i dałoby jedynie częściowe punkty, ale z bardzo mizerną szansą na doprowadzenie do pełnego działania). Raczej nie zadziała ciąg sprawdzeń typu: *jeżeli dużo czerwonego, a mało zielonego i mało niebieskiego to wypisz czerwony, w przeciwnym przypadku...*. W takim rozwiązaniu bardzo trudno upakować informacje o kolorach innych niż podstawowe.

Lepszym pomysłem jest wyobrazić sobie problem geometrycznie. Rozważmy sześcian o wymiarach $256 \times 256 \times 256$, każdemu kolorowi odpowiada kosteczka na pozycji (r, g, b), gdzie $0 \leqslant r, g, b \leqslant 255$

– liczby te odpowiadają odpowiednio przeliczonej na system dziesiętny wartości składowej czerwonej, zielonej i niebieskiej koloru (pierwsze dwa znaki na wejściu po haszu odpowiadają za składową czerwoną, kolejne dwa za składową zieloną i ostatnie dwa za składową niebieską).

Na sześcianie możemy teraz umieścić wzorcowe kolory – np. punkt o współrzędnych $(255, 0, 0)$ to wzorcowy kolor czerwony, zaś punkt $(255, 0, 255)$ to wzorcowy kolor różowy i tak dalej. Do konstrukcji tych wzorcowych kolorów najlepiej posłużyć się jakimś programem graficznym lub konwerterem kolorów online.

Teraz powinno już być chyba jasne, że kolor podany na wejściu wypada przybliżyć do któregoś punktu wzorcowego. Można zastosować wzór na odległość euklidesową

$$\sqrt{(r' - r)^2 + (g' - g)^2 + (b' - b)^2}$$

gdzie (r, g, b) to współrzędne koloru wczytanego na wejściu, zaś (r', g', b') to współrzędne sprawdzanego koloru wzorcowego (nie trzeba nawet liczyć pierwiastka, skoro i tak tylko porównujemy te liczby ze sobą). Każdy punkt należy przybliżyć do najbliższego punktu (tego, który daje najmniejszy wynik funkcji odległości zdefiniowanej powyżej). W razie problemów (np. jasny niebieski a'la morski i ciemny niebieski a'la granatowy, które mają bardzo różne współrzędne) można zdefiniować wiele różnych punktów wzorcowych dla tego samego koloru.

Testy zostały dobrane w taki sposób, żeby nie było kolorów „na granicy", o których ktoś mógłby zacząć dyskutować *czy to aby na pewno jest kolor różowy, a nie fioletowy?*.

6 Morfizm

Rozwiązanie zadania wprost, zgodnie z treścią zadania (wygeneruj całe J_N zgodnie z definicją, najpierw licząc J_0, potem J_1, potem J_2 itd., a następnie wyciągnij stosowny fragment napisu) prowadziło do poprawnego, ale nieakceptowalnie wolnego rozwiązania. Dawało to częściowe punkty, po które warto się schylić w razie braku pomysłów, lub problemów z implementacją rozwiązania wzorcowego.

Kluczowa obserwacja jest taka, że ponieważ każde przekształcenie zamienia jedną literę w trzy, długość napisu J_N wynosi 3^N. Chcąc więc wypisać k-ty znak napisu J_N, należałoby ustalić, $\lfloor \frac{k}{3} \rfloor$-ty znak napisu J_{N-1}, z którego powstał przecież napis J_N i wypisać (k mod 3)-cią literkę przekształcenia tego znaku: (k mod 3)-cią literę napisu `bab`, jeżeli $\lfloor \frac{k}{3} \rfloor$-ta literka napisu J_{N-1} to `a` lub po prostu literkę `b`, jeżeli $\lfloor \frac{k}{3} \rfloor$-ta literka napisu J_{N-1} to `b` (na potrzeby opisu rozwiązania przyjmujemy numerację znaków od 0). A jak ustalić tę literkę w napisie J_{N-1}? Tak samo, rekurencyjnie, odwołując się do odpowiedniej literki napisu J_{N-2} i tak dalej, aż dojdziemy do odpytania się do literek napisu J_0, który został nam dany na wejściu.

Każdą literę wypisujemy w czasie $O(N)$, używając tyle samo pamięci (poza przechowywaniem jeszcze oryginalnego napisu J_0), a ponieważ mamy do wypisania tylko $R - L + 1 \leqslant 100\,000$ literek, całe rozwiązanie mieści się bez problemu w założonym limicie czasu.

7 Skuteczny hacking

Jedyna, chyba niezbyt trudna, obserwacja, jaką należy poczynić w tym zadaniu jest taka, że nic nie ogranicza nas, żeby ciągle używać najwęższej literki. Wystarczy ją tylko znaleźć (nie potrzeba nawet tablicy, minimum ciągu liczb można znaleźć w locie, używając do tego pojedynczej zmiennej i pętli wczytującej kolejne szerokości literek oraz sprawdzając czy następna wczytana liczba jest mniejsza od bieżącego minimum).

Wiedząc jaka jest szerokość najwęższej literki, wystarczy podzielić dopuszczalną szerokość napisu przez nią i otrzymamy największą liczbę znaków, z których może składać się nazwa drużyny.