Wyszukiwanie Binarne i Koszt Zamortyzowany

Autor: Franciszek Pietrusiak

Wyszukiwanie Binarne elementu w ciągu

Mając **uporządkowany monotonicznie** ciąg n elementów jesteśmy w stanie w czasie $O(\log n)$ odpowiadać na zapytania czy dany element należy do ciągu.

Kod:

```
// zalozenie: tab[1] <= tab[2] <= ... <= tab[n]
bool binsearch(int x) {
        int low = 1;
        int high = n;
        while (low <= high) {
                int mid = (low + high) / 2;
                if (tab[mid] == x)
                        return true;
                else if (tab[mid] < x)
                        low = mid + 1;
                else high = mid - 1;
        }
        return false;
}
```

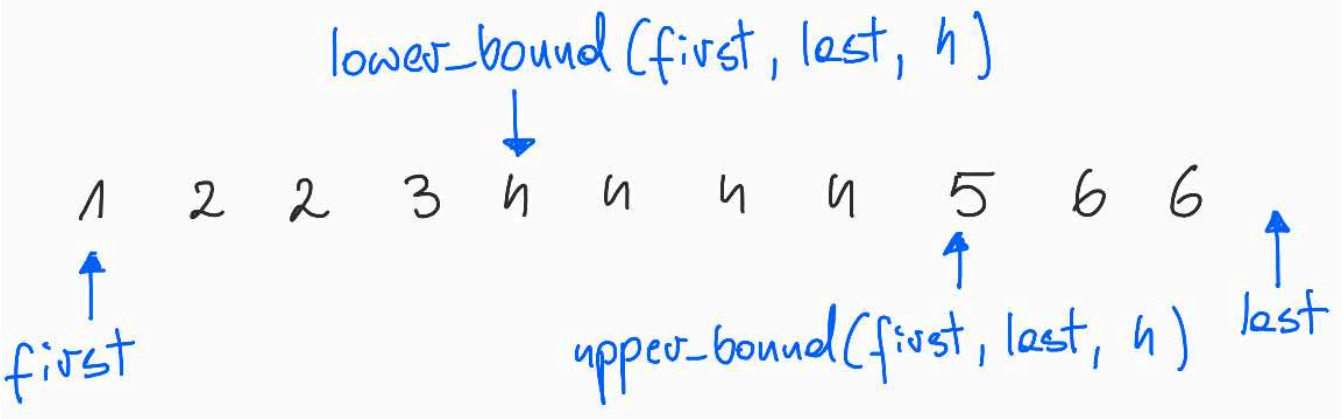
`lower_bound`, `upper_bound`

Używając `vector`'ów w C++ możemy skorzystać z wbudowanych funkcji w STL'u i nie pisać swojego binsearcha.

```
vector<int> v;
int x; \\ liczba której szukamy
auto lower = lower_bound(v.begin(), v.end(), x);
auto upper = upper_bound(v.begin(), v.end(), x);
```

`lower_bound(iterator first, iterator last, const val)` zwraca iterator do pierwszego elementu w przedziale `[first, last)` którego wartość jest **nie mniejsza niż** `val`.

`upper_bound(iterator first, iterator last, const val)` zwraca iterator do pierwszego elementu w przedziale `[first, last)` którego wartość jest większa niż `val`.



Zadanie Test na Inteligencje

Skrót Treści

Dostajemy ciąg A liczb całkowitych w przedziale $[1, 10^6]$. Ciąg A może być długości do 10^6. Mamy odpowiedzieć na dużo zapytań, czy pewien inny ciąg B jest **podciągiem** ciągu A. To znaczy, że jesteśmy w stanie usunąć niektóre elementy z ciągu A (ale nie zmieniać ich kolejności) i otrzymać ciąg B.

Rozwiązanie

Wystarczy dla każdej liczby z ciągu A zapamiętać wszystkie jej indeksy w ciągu posortowane rosnąco. Teraz dla każdego ciągu B idziemy wyraz po wyrazie. Załóżmy, że natrafiamy na liczbę x. Teraz szukamy takiego indeksu liczby x w ciągu A, który jest: 1) większy niż ostatni wzięty przez nas indeks (początkowo jest to 0), 2) najmniejszy możliwy. Jeśli w pewnym momencie okaże się, że nie istnieje taki indeks to B nie jest podciągiem A.

Wyszukiwanie Binarne po wyniku

Zadanie Sieć Wifi

Skrót Treści

Jest m sal ustawionych wzdłuż jednego długiego korytarza. Te sale są od siebie oddalone o pewne odległości. Mając do dyspozycji n access pointów mamy tak je ustawić, aby w każdej sali był możliwie najlepszy zasięg. Należy wypisać jedną liczbę rzeczywistą - największą odległość pomiędzy salą, a access pointem znajdującym się najbliżej tej klasy.

Rozwiązanie

Okazuje się, że o zadaniu możemy myśleć w inny, trochę prostszy sposób. Niech salom odpowiadają punkty na prostej, a access pointy będą odcinkami równej długości D. Teraz naszym zadaniem jest sprawdzić, jak małe D możemy dobrać, tak aby wszystkie m punktów było przykryte przez n odcinków. Zastanówmy się na początku jak mając ustalone D, w optymalny sposób ułożyć odcinki na prostej, tak, żeby zakryć wszystkie punkty. Możemy to robić **zachłannie**. Przeglądamy punkty od lewej do prawej (trzeba je wcześniej posortować) i jeśli dany punkt o współrzędnej x nie jest przykryty przez żaden odcinek, to kładziemy nowy odcinek na przedziale $[x, x + D]$. Jeśli po przejrzeniu wszystkich punktów użyliśmy nie więcej niż n odcinków to D nazywamy *dobrą*. Teraz kluczowa obserwacja: jeśli długość D jest *dobra*, to wszystkie długości większe od D także są *dobre*. A zatem możemy zastosować wyszukiwanie binarne po wyniku i znaleźć najmniejsze D, które jest *dobre.* Dostajemy rozwiązanie w złożoności $O(n \log K)$, gdzie K to maksymalna długość odcinków.

Zadanie Krążki

Skrót Treści

Dostajemy pewną nietypowo wyciętą rurkę oraz ileś krążków. Następnie wrzucamy te krążki w pewnej ustalonej kolejności. Jako odpowiedź mamy podać głębokość na jakiej zatrzyma się ostatni wrzucony do rurki krążek.

Rozwiązanie w złożoności $O(n + m \log n)$

Zauważmy dwa ważne fakty:

1. jeśli wrzuciliśmy do rurki krążek, który zatrzymał się na pewnej głębokości d, to teraz każdy z kolejnych krążków jakie wrzucimy do rurki zatrzyma się na **głębokości mniejszej** niż d.
2. jeśli rurka ma na pewnej głębokości d szerokość x to żaden z krążków o szerokości większej niż x **nie może** znaleźć się na głębokości większej niż d.

A zatem z perspektywy wrzucania krążków rurkę podaną w zadaniu możemy zastąpić równoważną rurką, w której szerokość coraz głębszych poziomów jest nierosnąca.
Dzięki temu, że teraz szerokość rurki jest posortowana, to w prosty sposób jesteśmy w stanie binsearchować głębokość na której zatrzyma się wrzucony do rurki krążek. Gdy tego dokonamy to na mocy faktu 1) możemy ograniczyć przeszukiwanie do poziomu na którym zatrzymał się krążek.

✓ **Rozwiązanie w złożoności** $O(n+m)$

Podobnie jak wcześniej zastępujemy rurkę podaną na wejściu przez "zwężającą się" rurkę. Jednak tym razem rurkę przechodzić będziemy od dna do góry nie korzystając z wyszukiwania binarnego. Dla każdego krążka będziemy liniowo szukać poziomu na który może on opaść. Zauważmy, że każdy z poziomów rozpatrywać będziemy **co najwyżej jeden raz**, ponieważ krążek może zatrzymać się na nim lub powyżej tego poziomu. Teraz wrzucając kolejne krążki z faktu 1) możemy "zapomnieć" o poziomach które już przeglądaliśmy. Zatem nasz algorytm z krokiem którego pesymistyczna złożoność to $O(n)$ okazał się o bardzo szybki. Jego **całkowita** złożoność jest znacznie lepsza niż iloczyn pesymistycznej złożoności jednego kroku i liczby wszystkich kroków. Jest to przykład **Kosztu Zamortyzowanego**.

Koszt Zamortyzowany

Analiza kosztu zamortyzowanego skupia się na oszacowaniu czasu niezbędnego do wykonania **ciągu** operacji, a nie pojedynczej operacji. Program może wykonywać się znacznie szybciej niż iloczyn pesymistycznego czasu operacji i ilości operacji do wykonania, ponieważ tylko niektóre ciągi operacji są możliwe.