

Algorytmy i struktury danych

Kolokwia (2)

struktury danych, algorytmy grafowe

2016/17

A

1. Student chce wypuścić n różnych pokémonów (numerowanych od 0 do $n - 1$) z klatek (pokéball'i). Wypuszczony Pokémon natychmiast atakuje swojego wybawiciela, chyba że (a) jest spokojny, lub (b) w okolicy znajdują się co najmniej dwa uwolnione pokémony, na które ten pokémon poluje. Proszę zaimplementować funkcję:

```
int* releaseThemAll( HuntingList* list, int n ),
```

gdzie `list` to lista z informacją, które pokémony polują na które (lista nie zawiera powtórzeń):

```
struct HuntingList {  
    HuntingList* next; // następny element listy  
    int predator;      // numer pokemona, który poluje  
    int prey; }       // numer pokemona, na którego poluje
```

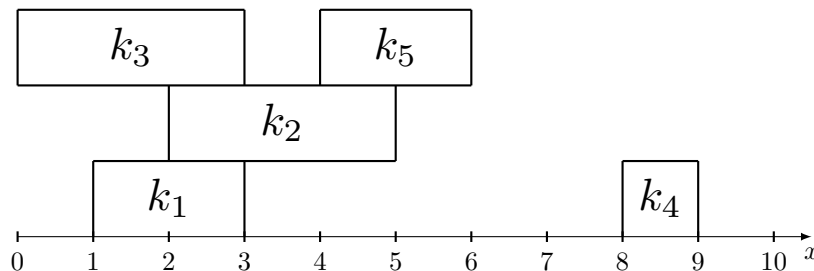
Funkcja powinna zwrócić n elementową tablicę z numerami pokémonów w kolejności wypuszczania (tak, żeby wypuszczający nie został zaatakowany) lub NULL jeśli taka kolejność nie istnieje. Każdy wypuszczony pokémon zostaje "w okolicy". Jeśli pokémon nie występuje na liście jako `predator` to znaczy, że jest spokojny. Zaimplementowana funkcja powinna być możliwie jak najszybsza. Proszę krótko oszacować jej złożoność.

2. Dana jest struktura `struct HT{ int* table; int size; }`, która opisuje tablicę haszującą rozmiaru `size`, przechowującą liczby nieujemne. Tablica korzysta z funkcji haszującej `int hash(int x)` i liniowego rozwiązywania konfliktów (ujemne wartości w tablicy `table` oznaczają wolne pola). Doskonałością takiej tablicy nazywamy liczbę elementów x takich, że pozycja x w tablicy to `hash(x) mod size` (a więc x jest na "swojej" pozycji). Proszę napisać funkcję:

```
void enlarge( HT* ht);
```

która powiększa tablicę dwukrotnie i wpisuje elementy w takiej kolejności, by doskonałość powstałej tablicy była jak największa. Funkcja powinna być możliwie jak najszybsza.

3. Dany jest ciąg klocków (k_1, \dots, k_n) . Kłoczek k_i zaczyna się na pozycji a_i i ciągnie się do pozycji b_i (wszystkie pozycje to liczby naturalne) oraz ma wysokość 1. Klocki układane są po kolei. Jeśli klocek nachodzi na któryś z poprzednich, to jest przymocowywany na szczycie poprzedzającego klocka. Na przykład dla klocków o pozycjach $(1,3)$, $(2,5)$, $(0,3)$, $(8,9)$, $(4,6)$ powstaje konstrukcja o wysokości trzech klocków (vide rysunek). Proszę podać możliwie jak najszybszy algorytm, który oblicza wysokość powstałej konstrukcji (bez implementacji) oraz oszacować jego złożoność obliczeniową.



Rysunek 1: Przykład konstrukcji.

1. Dane są struktury danych opisujące SkipListę:

```
struct SLNode {
    int      value; // wartość przechowywana w węźle
    int      level; // poziom węzła
    SLNode** next;  // level-elementowa tablica wskaźników na następniki
};
struct SkipList {
    SLNode* first; // wartownik przedni (pole value ma wartość  $-\infty$ )
    SLNode* last;  // wartownik tylny (pole value ma wartość  $+\infty$ )
};
const int MAX_LEVEL = ... ; // maksymalny poziom węzłów
```

Proszę zaimplementować funkcję `SkipList merge(SkipList A, SkipList B)`, która scala dwie otrzymane SkipListy (w efekcie powstaje nowa SkipLista składająca się z dokładnie tych samych węzłów co poprzednie; nie należy zmieniać poziomów węzłów). Węzły wartowników mają poziom `MAX_LEVEL`. Funkcja powinna działać możliwie jak najszybciej. Proszę oszacować złożoność czasową.

2. Proszę opisać (bez implementacji) algorytm, który otrzymuje na wejściu pewne drzewo BST T i tworzy nowe drzewo BST T' , które spełnia następujące warunki: (a) T' zawiera dokładnie te same wartości co T , oraz (b) drzewo T' jest drzewem czerwono-czarnym (w związku z tym powinno zawierać kolory węzłów).
3. W ramach obchodów międzynarodowego święta cyklistów organizatorzy przewidzieli górską wycieczkę rowerową. Będzie się ona odbywać po bardzo wąskiej ścieżce, na której rowery mogą jechać tylko jeden za drugim. W ramach zapewnienia bezpieczeństwa każdy rowerzysta będzie miał numer identyfikacyjny oraz małe urządzenie, przez które będzie mógł przekazać identyfikator rowerzysty przed nim (lub -1 jeśli nie widzi przed sobą nikogo). Należy napisać funkcję, która na wejściu otrzymuje informacje wysłane przez rowerzystów i oblicza rozmiar najmniejszej grupy (organizatorzy obawiają się, że na małe grupy mogłyby napadać dzikie zwierzęta). Dane są następujące struktury danych:

```
struct Cyclist {
    int id, n_id; // identyfikator rowerzysty oraz tego, którego widzi
};
```

Funkcja powinna mieć prototyp `int smallestGroup(Cyclist cyclist[], int n)`, gdzie `cyclist` to tablica n rowerzystów. Funkcja powinna być możliwie jak najszybsza. Można założyć, że identyfikatory rowerzystów to liczby z zakresu 0 do 10^8 (osiem cyfr napisanych w dwóch rzędach na koszulce rowerzysty) i że pamięć nie jest ograniczona. Rowerzystów jest jednak dużo mniej niż identyfikatorów (nie bez powodu boją się dzikich zwierząt). Należy zaimplementować wszystkie potrzebne struktury danych.

1. Pewna firma przechowuje dużo iczb pierwszych w postaci binarnej jako stringi "10101...". Zaimplementuj strukturę danych `Set` do przechowywania tych danych. Napisz funkcje: `Set createSet(string A[], int n)`, która tworzy `Set` z n -elementowej tablicy stringów oraz `bool contains(Set a, string s)`, która sprawdza czy dana liczba jest w `Secie`. Oszacuj złożoność czasową i pamięciową powyższych funkcji.
2. Dana jest struktura danych:


```
struct Edge {
    int u, v; // u < v
    Edge* next;
};
```

 Napisz funkcję `bool Euleran(Edge* E, int n)`, która sprawdza czy graf zadany przez listę krawędzi posiada cykl Eulera (n to liczba wierzchołków w grafie). Graf jest nieskierowany i spójny. Krawędzie w liście mogą się powtarzać. Funkcja powinna alokować nie więcej pamięci, niż liniowo proporcjonalnie do ilości krawędzi.
3. Zaproponuj algorytm, który policzy ile jest najkrótszych ścieżek w grafie z danego wierzchołka u do v . Wskazówka: Dla każdej najkrótszej ścieżki przechodzącej przez wierzchołek w , odległość w od startu jest taka sama jak odległość w do mety.

A

1. Proszę zaimplementować funkcję wstawiającą zadaną liczbę do SkipListy przechowującej dane typu `int`. Proszę zadeklarować wszystkie potrzebne struktury danych i krótko (2-3 zdania) opisać zaimplementowany algorytm.
2. Proszę opisać jak zmodyfikować drzewa czerwono-czarne (przechowujące elementy typu `int`) tak, by operacja `int sum(T, x, y)` obliczająca sumę elementów z drzewa o wartościach z przedziału $[x, y]$ działała w czasie $O(\log n)$ (gdzie n to rozmiar drzewa T). Pozostałe operacje na powstałym drzewie powinny mieć złożoność taką samą jak w standardowym drzewie czerwono-czarnym. (Podpowiedź: Warto w każdym węźle drzewa przechowywać pewną dodatkową informację, która upraszcza wykonanie operacji `sum` i którą można łatwo aktualizować).
3. Kapitan pewnego statku zastanawia się, czy może wpłynąć do portu mimo tego, że nastąpił odpływ. Do dyspozycji ma mapę zatoki w postaci tablicy:


```
int n = ...
int m = ...
int A[m][n];
```

gdzie wartość $A[y][x]$ to głębokość zatoki na pozycji (x, y) . Jeśli jest ona większa niż pewna wartość `int T` to statek może się tam znaleźć. Początkowo statek jest na pozycji $(0, 0)$ a port znajduje się na pozycji $(n - 1, m - 1)$. Z danej pozycji statek może przepłynąć bezpośrednio jedynie na pozycję bezpośrednio obok (to znaczy, na pozycję, której dokładnie jedna ze współrzędnych różni się o jeden). Proszę napisać funkcję rozwiązującą problem kapitana.

B

1. Proszę zaimplementować funkcję usuwającą zadaną liczbę ze SkipListy przechowującej dane typu `int`. Proszę zadeklarować wszystkie potrzebne struktury danych i krótko (2-3 zdania) opisać zaimplementowany algorytm.
2. Proszę opisać jak zmodyfikować drzewa AVL (przechowujące elementy typu `int`) tak, by operacja `int findRandom(T)` zwracająca losowo wybrany element z drzewa T działała w czasie $O(\log n)$. Funkcja `findRandom` powinna zwracać każdy element z drzewa z takim samym prawdopodobieństwem. Do dyspozycji mają Państwo funkcję `int random(int k)`, która zwraca liczbę ze zbioru $\{0, \dots, k - 1\}$ zgodnie z rozkładem jednostajnym. Pozostałe operacje na powstałym drzewie powinny mieć złożoność taką samą jak w standardowym drzewie AVL. (Podpowiedź: Warto w każdym węźle drzewa przechowywać pewną dodatkową informację, która upraszcza wykonanie operacji `findRandom` i którą można łatwo aktualizować).

3. Dana jest tablica:

```
int n = ...
int m = ...
bool A[m][n];
```

Gracz początkowo znajduje się na (zadanej) pozycji (x, y) , dla której zachodzi `A[y][x] == true`. Z danej pozycji wolno bezpośrednio przejść jedynie na pozycję, której dokładnie jedna współrzędna różni się o 1, oraz której wartość w tablicy A wynosi `true`. Proszę napisać program obliczający do ilu różnych pozycji może dojść gracz startując z zadanej pozycji (x, y) .