Zadanie nr 1 na pracownię

Samobalansujące binarne drzewa przeszukiwań

Binarne drzewa przeszukiwań (ang. binary search trees, BST) w formie zaprezentowanej na wykładzie posiadają istotną praktyczną wadę. Mianowicie ciąg operacji wstawienia do drzewa może doprowadzić do szybszego niż logarytmicznego wzrostu długości najdłuższej ścieżki w drzewie, co przekłada się na gorszy niż logarytmiczny czas wyszukiwania elementu w drzewie (oraz innych operacji).

Przykładowo, wstawianie elementów w kolejności zgodnej z porządkiem przeszukiwania prowadzi do powstania zdegenerowanego drzewa rosnącego tylko w jedną stronę (w lewo lub w prawo), z najdłuższą ścieżką liniowej długości.

Aby temu zaradzić, stosuje się drzewa samobalansujące – tzn. takie, które przy wykonywaniu operacji (np. wstawień i usunięć) zmieniają swój kształt w taki sposób, aby czas wykonywania operacji na drzewie był logarytmiczny.

Znanych jest wiele rodzajów drzew samobalansujących. Najpopularniejsze z nich wymagają albo przechowywania dodatkowych informacji w węzłach (w drzewach czerwono-czarnych – jeden z dwóch kolorów węzła, w drzewach AVL – jeden z trzech współczynników wyważenia), albo przebudowywania drzewa również przy operacji przeszukiwania (drzewa splay – operacje na którym mają zamortyzowany koszt logarytmiczny, ale pesymistyczny liniowy). W tym zadaniu zajmiemy się implementacją innego rodzaju drzew przeszukiwania: drzewa te, podobnie do drzew splay, nie wymagają przechowywania dodatkowych informacji w węzłach, mają też logarytmiczny koszt zamortyzowany operacji wstawiania i usuwania; w przeciwieństwie do drzew splay, przeszukiwanie odbywa się w pesymistycznym czasie logarytmicznym i nie wymaga przebudowywania drzewa. Mowa o drzewach z kozłem ofiarnym (ang. scapegoat tree).[1]

[1] Galperin, Igal, and Ronald L. Rivest. “Scapegoat trees.” Proceedings of the fourth annual

Drzewa z kozłem ofiarnym – idea

Idea stojąca za operacją wstawiania do drzew z kozłem ofiarnym jest następująca. Razem z drzewem pamiętamy jedną wartość – liczbę wierzchołków w drzewie (`size`). Na jej podstawie można wyliczyć maksymalną dopuszczalną długość ścieżki w drzewie (jest ona równa podłodze logarytmu pewnego stopnia z liczby wierzchołków). Wstawianie nowego wierzchołka zaczyna się analogicznie, jak w drzewach BST z wykładu: znajdujemy w drzewie liść, który możemy zastąpić wierzchołkiem z wstawianą wartością. Jednocześnie liczymy też długość ścieżki, na której ten liść się znajduje. Gdy ścieżka ta jest krótsza od maksymalnej dopuszczalnej, wynik wstawiania jest taki sam, jak w drzewach BST z wykładu – wstawienie nie naruszyło zbalansowania drzewa. W przeciwnym wypadku wracając do korzenia poszukujemy pierwszego wierzchołka, w którym zakorzenione poddrzewo jest źle zbalansowane (w sensie opisanym w późniejszym, szczegółowym opisie). Wierzchołek ten nazwiemy kozłem ofiarnym. Całe poddrzewo którego korzeniem jest kozioł ofiarny przebudowujemy do drzewa kompletnego.[2] Operację przebudowania wykonujemy tylko raz – pozostała część drzewa pozostaje bez zmian.

Implementacja usuwania w drzewach z kozłem ofiarnym wymaga pamiętania razem z drzewem drugiej wartości – maksymalnej liczby wierzchołków, którą posiadało drzewo (`max_size`). Element usuwany jest z drzewa tak samo, jak ze zwykłych drzew BST; jednocześnie liczba wierzchołków w drzewie (`size`) jest zmniejszana o jeden, a maksymalna liczba wierzchołków (`max_size`) pozostaje bez zmian. Gdy liczba wierzchołków po usunięciu elementu okaże się zbyt mała (objaśnione później), należy przebudować całe drzewo do drzewa kompletnego oraz zastąpić `max_size` bieżącą liczbą wierzchołków.

Szczegóły implementacji

Drzewa z kozłem ofiarnym mają parametr α z zakresu $\frac{1}{2} < \alpha < 1$. Parametr ten wpływa na stopień zrównoważenia drzewa. Im α jest mniejsza, tym bardziej

ACM-SIAM Symposium on Discrete algorithms. 1993.

[2]Za drzewo kompletne uznamy takie, w którym długości ścieżek od korzenia do liści są równe n lub $n+1$ dla pewnego n.

zrównoważone będzie drzewo i tym samym szybsze będzie jego przeszukiwanie. Jednocześnie mniejsza α oznacza też, że wstawianie do drzewa będzie częściej je przebudowywać – co spowalnia wstawianie. W tym zadaniu arbitralnie przyjmiemy wartość $\alpha = \frac{3}{4}$.

Przyjmijmy następujące oznaczenia:

- $\mathit{size}(T)$ – liczba wierzchołków w drzewie T,
- $h(T)$ – wysokość drzewa T, tzn. długość najdłuższej ścieżki od korzenia do liścia.

Jako α-wysokość drzewa T wierzchołkach będziemy rozumieć wartość:

$$h_\alpha(T) = \lfloor \log_{1/\alpha} \mathit{size}(T) \rfloor$$

Drzewa z kozłem ofiarnym są zawsze α-wysokościowo-zbalansowane – co oznacza, że dla takich drzew T zachodzi warunek:

$$h(T) \leq h_\alpha(T) + 1$$

Wierzchołek drzewa T o głębokości[3] większej niż $h_\alpha(T)$ nazywamy *głębokim*. Wstawienie głębokiego wierzchołka jest warunkiem przebudowania drzewa. Przebudowując drzewo cofamy się po ścieżce, szukając kozła ofiarnego. Jest on pierwszym napotkanym (od strony wstawianego wierzchołka) wierzchołkiem, dla którego poddrzewo T' w nim zakorzenione nie spełnia warunku α-wagowego-zbalansowania:

$$\mathit{size}(\mathit{left}(T')) \leq \alpha \cdot \mathit{size}(T') \ \wedge \ \mathit{size}(\mathit{right}(T')) \leq \alpha \cdot \mathit{size}(T')$$

Przy usuwaniu wierzchołka przebudowujemy całe drzewo, jeśli rozmiar drzewa po usunięciu wierzchołka jest mniejszy niż maksymalny rozmiar `max_size` przemnożony przez α. Po przebudowaniu poddrzewa `max_size` jest ustawiany na rzeczywistą liczbę wierzchołków w drzewie.

[3]Głębokością wierzchołka nazywamy długość ścieżki od korzenia do tego wierzchołka.

Zadanie

Zaimplementuj drzewo z kozłem ofiarnym dla $\alpha = \frac{3}{4}$. Przyjmij następującą definicję drzewa:

```
type 'a tree = Leaf | Node of 'a tree * 'a * 'a tree
type 'a sgtree = { tree : 'a tree; size : int; max_size: int }
```

Zaimplementuj:

- `alpha_height : int -> int` – funkcja, która zaaplikowana do n oblicza $h_\alpha(T)$ dla drzewa T o rozmiarze $size(T) = n$,
- `rebuild_balanced : 'a tree -> 'a tree` – funkcja która dla zadanego drzewa BST zwraca kompletne drzewo BST zawierające te same elementy,
- `empty : 'a sgtree` – puste drzewo z kozłem ofiarnym,
- `find : 'a -> 'a sgtree -> bool` – przeszukiwanie drzewa,
- `insert : 'a -> 'a sgtree -> 'a sgtree` – wstawianie elementu do drzewa,
- `remove : 'a -> 'a sgtree -> 'a sgtree` – usuwanie elementu z drzewa.

Dodawanie elementu już istniejącego w drzewie oraz usuwanie elementu nie występującego w drzewie można uznać jako błąd.

Podpowiedź: Wstawianie elementu można zaimplementować przez użycie rekurencyjnej funkcji pomocniczej z odpowiednim typem zwracanych wartości – można użyć np. `'a sgtree * int option`. Inną opcją jest użycie struktury znanej jako *zipper*.[4]

Wykorzystaj szablon rozwiązania dostępny na SKOS. Plik z rozwiązaniem, nazwany `solution.ml`, zgłoś przez system Web-CAT (dostęp przez odnośnik na SKOS) do dnia 16 kwietnia 2025, godz. 6:00.

[4] Zobacz np. https://wiki.haskell.org/Zipper