Równoważenie binarnych drzew przeszukiwania metodą drzew AVL, i miało być jeszcze metodą drzew wagowo zrównoważonych i metodą drzew czerwono-czarnych

Michał Krzysztof Feiler

Algorytmy i Struktury Danych gr. LJ*, 14 maja 2018 roku

Outline

1. Wstęp
   - Problem tablic asocjatywnych uporządkowanych
   - Problem implementacji typu zbiorowego i wielozbiorowego
   - Drzewa przeszukiwania
   - Binarne drzewa przeszukiwania
     - Działania na BST
   - Niezrównoważone BST są nieefektywne
2. Drzewa AVL
   - Definicja
   - Pojedyńcza rotacja w lewo
   - Podwójna rotacja prawo-lewo
3. B-drzewa pokrótce
   - Ogólnik

Wstęp — problem tablic asocjatywnych uporządkowanych

Problem implementacji tablic asocjatywnych, a więc, optymalnej struktury danych dla zbioru wartości typu V, mając funkcję klucza $k : V \to K$ (różnowartościową lub nie (wielozbiór kluczy)), gdzie K jest pewnym typem klucza, pod kątem efektywnego znajdowania wartości po kluczu, tzn. po odpowiadającej im wartości funkcji k.

Problem implementacji tablic asocjatywnych uporządkowanych, a więc rozszerzenie problemu tablicowania o zagadnienie optymalizacji pod kątem odwiedzania elementów w porządku, mając dodatkowo relację porządku liniowego $\leq$ na K — która gdy złożona z k tworzy relację na V:

- spójnego praporządku $\lesssim$ (przechodnią, spójną, i zwrotną) (k nie musi być injekcją, jest multizbiór kluczy),
- która, jeśli k injekcją, jest również porządkiem liniowym (dodatkowo słabo antysymetryczną) (tylko zbiór kluczy).

Wstęp — problem impl. typu zbiorowego i wielozbiorowego

Problem implementacji typu zbiorowego lub multizbiorowego uporządkowanego, a więc, optymalnej struktury danych dla zbioru wartości typu V, mając relację spójnego praporządku $\lesssim$ na V (przechodnią, spójną i zwrotną)
(w przypadku typu zbiorowego, $\lesssim$ jest również porządkiem liniowym — dodatkowo słabo antysymetryczną —),
pod kątem odwiedzania elementów w porządku
(a także, w przypadku typu zbiorowego, efektywnego pomijania duplikatów przy dodawaniu wartości do struktury).
Jeżeli mamy też pewną funkcję $k : V \to K$, gdzie K jest pewnym typem na którym mamy określony pewien porządek liniowy $\leq$, który złożony z k jest równoważny $\lesssim$, problem można sprowadzić do problemu implementacji tablic asocjatywnych uporządkowanych.

Wstęp — drzewa przeszukiwania

Drzewa przeszukiwania są jednym z rozwiązań problemu implementacji tablic asocjatywnych uporządkowanych lub typów (wielo)zbiorowych.

Są to struktury–drzewa
z parametryzacją rozmieszczenia poziomego (uporządkowane),

- która jest unikalna na zbiorze poddrzew danego węzła,
- dla której wartości ustalona spójna relacja porządku implikuje relację $\lesssim$ zarówno dla wierzchołków poddrzew po odpowiednich stronach jak i dla pozostałych wartości w danym węźle (czyli relację $\leq$ dla kluczy).

Wstęp — binarne drzewa przeszukiwania

Najprostszą odmianą drzew przeszukiwania są binarne drzewa przeszukiwania (BST). Są to drzewa przeszukiwania, w których

- węzły drzewa zawierają tylko jedną wartość,
- mają każdy od zera do dwóch poddrzew,
- ich parametryzacja rozmieszczenia poziomego jest
    - dwuwartościowa (boolowska)
    - o wartości spełniającej implikację zachodzenia relacji $\lesssim$ między wartościami danego węzła a wierzchołka danego jego poddrzewa
        - jeżeli $\lesssim$ jest porządkiem liniowym to równej zachodzeniu relacji $\lesssim$,
        - wówczas również, przez przechodniość tej relacji, zachodzeniu relacji $\lesssim$ również z pozostałym poddrzewem, jeżeli są dwa pod danym węzłem.

Działania na BST — struktury algebraiczne

Przedstawmy drzewo BST jako typ algebraiczny

$\texttt{class}\ \mathit{POrd}\ K\ \texttt{where}\ \ (\leq) : K \to K \to \mathit{Bool}$

$\texttt{class}\ \mathit{TPreOrd}\ V\ \texttt{where}\ \ (\lesssim) : V \to V \to \mathit{Bool}$

$\texttt{class}\ (\mathit{TPreOrd}\ V, \mathit{POrd}\ K) \Rightarrow \mathit{NodeVal}\ V\ \ K\ \texttt{where}\ k : V \to K$

$\texttt{type}\ \mathit{TwoSub}\ V\ \ K = (\mathit{Maybe}\,(\mathit{Node}\ V\ \ K)\ ,\ \mathit{Maybe}\,(\mathit{Node}\ V\ \ K))$

$\texttt{data}\ \mathit{Node}\ V\ \ K\ \texttt{where}$

$\quad \mathit{Node} : (\mathit{NodeVal}\ V\ \ K) \Rightarrow V \to \mathit{TwoSub}\ V\ \ K \to \mathit{Node}\ V\ \ K$

Działania na BST — znajdowanie po kluczu oraz traversal

$$
\begin{aligned}
&\mathit{get} : (\mathit{Eq}\ K) \Rightarrow \mathit{Maybe}\ (\mathit{Node}\ V\ \ K) \rightarrow K \rightarrow [V] \\
&\mathit{get}\ \mathit{Nothing}\ \_ = [] \\
&\mathit{get}\ (\mathit{Just}\ (\mathit{Node}\ t\ (l, p)))\ x = \\
&\quad [t \mid k\ \ t == x] +\!\!+ (\mathit{get}\ \ (\texttt{if}\ x \leq k\ \ t\ \ \texttt{then}\ l\ \ \texttt{else}\ p)\ \ x) \\
&\mathit{list} : \mathit{Maybe}\ (\mathit{Node}\ V\ \ K) \rightarrow [V] \\
&\mathit{list}\ \mathit{Nothing} = [] \\
&\mathit{list}\ (\mathit{Just}\ (\mathit{Node}\ t\ (l, p))) = \mathit{list}\ l +\!\!+ [t] +\!\!+ \mathit{list}\ p
\end{aligned}
$$

Działania na BST — wstawianie

$$
\begin{array}{l}
\mathit{insert} : (\mathit{NodeVal}\ V\ \ K) \Rightarrow \mathit{Maybe}\ (\mathit{Node}\ V\ \ K) \rightarrow v \rightarrow \mathit{Node}\ V\ \ K \\
\mathit{insert}\ \mathit{Nothing}\ x = \mathit{Node}\ x\ (\mathit{Nothing}, \mathit{Nothing}) \\
\mathit{insert}\ (\mathit{Just}\ \ (\mathit{Node}\ t\ (l, p)))\ x \\
\quad \texttt{--}\ |\quad x \lesssim t\ \wedge\ t \lesssim x\ \ = \mathit{Node}\ t\ (l, p) \quad \texttt{--}\ \text{el. duplikatów} \\
\quad\quad\ \ |\quad x \lesssim t\ \ = \mathit{Node}\ t\ (\mathit{Just}\ (\mathit{insert}\ l\ x)\ ,\ \ p) \\
\quad\quad\ \ |\quad \texttt{otherwise}\ \ = \mathit{Node}\ t\ (l,\ \mathit{Just}\ (\mathit{insert}\ p\ x))
\end{array}
$$

Działania na BST — popLeftmost

$\texttt{type}\ PopFun\ V\ \ K = Node\ V\ \ K \to (Maybe\,(Node\ V\ \ K)\ ,\ \ V)$

$popLeftmost : PopFun\ V\ \ K$

$popLeftmost\ (Node\ t\ (Nothing, p)) = (p, t)$

$popLeftmost\ (Node\ t\ (Just\ l,\ \ p)) =$

$\quad \texttt{let}\ (n, v) = popLeftmost\ l\ \texttt{in}\ (Just\ (Node\ t\ (l, n))\ ,\ \ v)$

$popRightmost : PopFun\ V\ \ K$

$popRightmost\ (Node\ t\ (l, Nothing)) = (l, t)$

$popRightmost\ (Node\ t\ (l, Just\ p)) =$

$\quad \texttt{let}\ (n, v) = popRightmost\ p\ \texttt{in}\ (Just\ (Node\ t\ (n, p))\ ,\ \ v)$

Działania na BST — popDeeperOneOffTop

$$popDeeperOneOffTop : PopFun\ V\ K$$
$$popDeeperOneOffTop\ (Node\ t\ Nothing\ Nothing) = (Nothing\ ,\ t)$$
$$popDeeperOneOffTop\ (Node\ t\ l\ p) =$$
$$\texttt{let}\ ((n_l, n_p), v) = \_popDeeperOneOff\ (l\ ,\ p)\ \texttt{in}\ (Just\ (Node\ t\ n_l\ n_p))$$
$$\_popDeeperOneOff : TwoSub\ V\ K \rightarrow (TwoSub\ V\ K\ ,\ V)$$
$$\_popDeeperOneOff\ (Just\ (Node\ t_l\ (o_l, l)), Just\ (Node\ t_p\ (p, o_p)) =$$
$$\quad \texttt{let}\ ((n_l, n_p), v) = \_popDeeperOneOff\ (l, p)\ \texttt{in}$$
$$\quad\ (Just\ (Node\ t_l\ (o_l, n_l)), Just\ (Node\ t_p\ (n_p, o_p))), v)$$
$$\_popDeeperOneOff\ (Just\ l\ , Nothing) =$$
$$\quad \texttt{let}\ (n, v) = popRightmost\ l\ \texttt{in}\ ((n, Nothing), v)$$
$$\_popDeeperOneOff\ (Nothing, Just\ p) =$$

Działania na BST — _popDeeperOneOff

$$\texttt{let}\ ((n_l, n_p), v) = \_popDeeperOneOff\ (l\ ,\ \ p)\ \texttt{in}\ (Just\ (Node\ t\ n_l\ \ n_p))$$

$$\_popDeeperOneOff : TwoSub\ V\ \ K \to (TwoSub\ V\ \ K\ ,\ \ V)$$

$$\_popDeeperOneOff\ (Just\ (Node\ t_l\ (o_l, l)), Just\ (Node\ t_p\ (p, o_p)) =$$
$$\quad \texttt{let}\ ((n_l, n_p), v) = \_popDeeperOneOff\ (l, p)\ \texttt{in}$$
$$\quad\quad (Just\ \$\ Node\ t_l\ (o_l, n_l), Just\ \$\ Node\ t_p\ (n_p, o_p)), v)$$

$$\_popDeeperOneOff\ (Just\ l\ , Nothing) =$$
$$\quad \texttt{let}\ (n, v) = popRightmost\ l\ \texttt{in}\ ((n, Nothing), v)$$

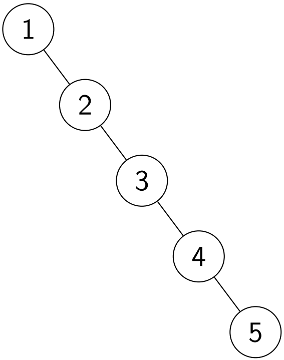
$$\_popDeeperOneOff\ (Nothing, Just\ p) =$$
$$\quad \texttt{let}\ (n, v) = popLeftmost\ p\ \texttt{in}\ ((Nothing, n), v)$$

$$\_popDeeperOneOff\ (Nothing, Nothing) = \texttt{undefined}$$

Działania na BST — usuwanie elementów po kluczu

$$
\begin{aligned}
&\mathit{delete} : (\mathit{Eq}\ K) \Rightarrow \mathit{Maybe}\ (\mathit{Node}\ V\ \ K) \to K \to \mathit{Maybe}\ (\mathit{Node}\ V\ \ K)\\
&\mathit{delete}\ \mathit{Nothing}\ \_ = \mathit{Nothing}\\
&\mathit{delete}\ (\mathit{Just}\ (\mathit{Node}\ t\ (l, p)\ )\ )\ \ x\\
&\qquad \mid\ \ k\ \ t == x\ \ \wedge\ \ \mathit{isNothing}\ l\ \ = \mathit{delete}\ p\ x\\
&\qquad \mid\ \ k\ \ t == x\ \ \wedge\ \ \mathit{isNothing}\ p\ \ = \mathit{delete}\ l\ x\\
&\qquad \mid\ \ k\ \ t == x\ \ = \mathit{flip}\ \mathit{delete}\ x\ \$\ \mathit{Just}\ \$\\
&\qquad\quad\ \ \texttt{let}\ ((n_l, n_p), v) = \_\mathit{popDeeperOneOff}\ (l, p)\\
&\qquad\quad\ \ \ \ \texttt{in}\ \mathit{Node}\ v\ (n_l, n_p)
\end{aligned}
$$

Niezrównoważone BST są nieefektywne



Rysunek 1: Niezbalansowane BST sekwencji kilku kolejnych liczb nat.

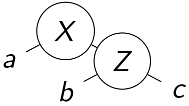
Drzewa AVL — definicja

- Drzewo AVL, od nazwisk Gieorgia Adelson-Wielskiego oraz Jewgienija Łandisa, którzy wynaleźli algorytm i opublikowali go w 1962.
- Miarą zrównoważenia drzewa jest tzw. współczynnik wyważenia drzewa, określany jako różnica wysokości prawego poddrzewa i lewego poddrzewa.
- Drzewo BST jest drzewem AVL, jeżeli współczynnik wyważenia drzewa nie jest oddalony od zera o więcej niż 1.
- Podtrzymywanie spełniania tej definicji polega na równoważeniu drzewa po każdym wstawieniu i usunięciu.

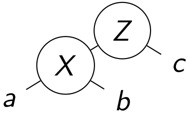
Pojedyńcza rotacja w lewo

- gdy mamy poddrzewo, którego prawe poddrzewo jest wyższe

1. niech $X =$ wartość wierzchołka naszego
2. niech $Z =$ wartość wierzchołka prawego



3. niech lewe poddrzewo naszego to będzie a
4. niech lewe poddrzewo prawego to będzie b
5. niech prawe poddrzewo prawego to będzie c



6. to wynik pojedyńczej rotacji w lewo to będzie
7. jeżeli było AVL i zostało coś usunięte z a, i b i c tak samo wysokie to zapiszmy że teraz dla X $+1$ a dla Z -1. Otherwise oba zero.

Podwójna rotacja prawo-lewo

- No ale mamy ddrzewo, którego prawe jest wyższe
- i tego prawego poddrzewa lewe poddrzewo jest wyższe

Podwójna rotacja prawo-lewo

- No ale mamy ddrzewo, którego prawe jest wyższe
- i tego prawego poddrzewa lewe poddrzewo jest wyższe

- jadąc od góry tylko obrócilibyśmy sobie sytuację

Podwójna rotacja prawo-lewo

- No ale mamy ddrzewo, którego prawe jest wyższe
- i tego prawego poddrzewa lewe poddrzewo jest wyższe

- jadąc od góry tylko obrócilibyśmy sobie sytuację
- dlatego trzeba obrócić piętro niżej
- i następnie obrócić nasze

Podwójna rotacja prawo-lewo

- No ale mamy ddrzewo, którego prawe jest wyższe
- i tego prawego poddrzewa lewe poddrzewo jest wyższe

- jadąc od góry tylko obrócilibyśmy sobie sytuację
- dlatego trzeba obrócić piętro niżej
- i następnie obrócić nasze

- teraz:
  - jeżeli podpod było prawo-wyższe to teraz to o wierzchołku naszego (które jest lewym pod) jest -1, a to o wierzchołku prawego pod (które jest prawego pod) jest 0
  - jeżeli podpod było na zero to teraz dwa ww są na zero oba
  - a jak było lewo-wyższe to jest odwrotnie co w prawo-wyższe i na plus a nie na minus

B-drzewa — ogólnik

B-drzewa (znaczenie B jest nieustalone) są poniekąd uogólnieniem BST. Są do drzewa przeszukiwania, w których

- węzły zawierają od d do $2d$ wartości, gdzie $d \in \mathbb{N}^+$ ustalone
- jeżeli dany węzeł zawiera n wartości, to ma $n + 1$ poddrzew
- ich parametryzacja rozmieszczenia poziomego jest
  - n-wartościowa, z ustalonym dla tych wartości ostrym porządkiem liniowym $<$, przy czym dla każdych dwóch elementów zbioru wartości parametryzacji $n + 1$-wartościowej wartość znajdująca się w zb. w. parametryzacji n-wartościowej będzie zawsze w relacji $<$ z wartością która się w nim nie znajduje

B-drzewa — definicja wg Knutha

- rząd B-drzewa w sensie Knutha określa się jako $2d$
- dla ułatwienia nazywa się je czasem drzewami d-$2d$

B-drzewa — definicja wg Knutha

- rząd B-drzewa w sensie Knutha określa się jako $2d$
- dla ułatwienia nazywa się je czasem drzewami d-$2d$
- termin *liść* w sensie ogólnym oznacza węzeł bez poddrzew, ale w sensie Knutha liście są dopiero poniżej najniższych węzłów
- bardziej precyzyjnie, węzłów wewnętrznych (nie mylić z podwęzłami)
- podwęzłami czyli poszczególnymi wartościami w węzłach

B-drzewa — definicja wg Knutha

- rząd B-drzewa w sensie Knutha określa się jako $2d$
- dla ułatwienia nazywa się je czasem drzewami d-$2d$
- termin *liść* w sensie ogólnym oznacza węzeł bez poddrzew, ale w sensie Knutha liście są dopiero poniżej najniższych węzłów
- bardziej precyzyjnie, węzłów wewnętrznych (nie mylić z podwęzłami)
- podwęzłami czyli poszczególnymi wartościami w węzłach
- wymaganie: wszystkie liście na tej samej głębokości

B-drzewa — rozróżnienie liści

- można zastosować podejście w którym z wartości nie da się wydobyć kluczy
- wówczas klucze znajdują się w węzłach wewnętrznych a wartości w liściach
- wówczas kluczem danej wartości jest klucz w podwęźle separującym z prawej (niejako traktuje się je razem jako wartość z wspomnianej wcześniej definicji)
- jeżeli to rightmost liść, to kluczem jest prawy podwęzeł separujący w węźle wyżej, itd., a w całym drzewie rightmost podwęzeł rightmost węzła ma po prawej liść `NIL`

B-drzewa — implementacja

```c
#include <stdlib.h>
#include <string.h>
#define E 1
#define EM (E + 1)
#define EMP (EM + 1)
#define D (1 << E)
#define DP (D + 1)
#define DM (1 << EM)
#define DMP (DM + 1)
typedef struct Val {
        short unsigned len;
        char *s;
} V;
typedef unsigned K;
```

```c
struct intnode {
        K ks[DM];
        unsigned kn : EM;
        void* s[DMP];
};
union either {
        struct intnode i;
        V l;
};
struct node {
        unsigned is_leaf : 1;
        union either n;
};
```

```
struct location {
        struct intnode *which;
        unsigned where : EM;
};
struct traversal {
        struct location *loc;
        size_t depth;
};
struct traversal new_traversal(struct intnode *w) {
        struct traversal r;
        r.depth = 1;
        r.loc = malloc(sizeof(struct location));
        r.loc->which = w;
        r.loc->where = 0;
        return r;
}
```

```
struct traversal begin_traversal(    struct traversal t,
                                     unsigned copy    ) {
    struct traversal r = t;
    size_t much = sizeof(struct location)*r.depth;
    struct location l = r.loc[r.depth-1];
    if(!((struct node*)(l.which->s[l.where]))->is_leaf) {
        if(copy) {    r.loc = malloc(much);
                      memcpy(r.loc, t.loc, much);    }
        much+=sizeof(struct location);
        r.loc = realloc(r.loc, much);
        r.loc[r.depth].which = l.which->s[l.where];
        r.loc[r.depth].where = 0;
        r.depth+=1;
        r = begin_traversal(r, 0);    }
    return r;
}
```

```
struct traversal walk(struct traversal t, unsigned copy) {
    struct traversal r = t;
    size_t much = sizeof(struct location)*r.depth;
    if(copy) {     r.loc = malloc(much);
                   memcpy(r.loc, t.loc, much);    }
    struct location l = r.loc[r.depth-1];
    do {    if(l.which->kn>l.where) {
                r.loc[r.depth-1].where++;
                break;              }
            r.depth--;
            l = r.loc[r.depth-1];
    } while (r.depth>0);
    r = begin_traversal(r, 0);
    return r;
}
```

B-drzewa — wstawianie

- .