PAMSI - projekt 1
Drzewo AVL

Aleksandra Rzeszowska (234780)
środa, 18:55-20:35

28 marca 2018

1 Sposób implementacji

Drzewo AVL jest binarnym drzewem poszukiwań (BST) o zrównoważonej wysokości poddrzew. Lewe poddrzewo może mieć wysokość taką samą jak prawe poddrzewo, lub różną o 1. Jeśli lewe poddrzewo jest wyższe niż prawe, wartość współczynnika zrównoważenia w węźle przyjmuje wartość 1, w przeciwnym razie -1.

1.1 Struktura ED

Każdy węzeł drzewa AVL ma typ struktury ED. Struktura ta ma trzy pola będące wskaźnikami na ten sam typ (ED*): wskazanie na rodzica, lewego syna oraz prawego syna; wartość przechowywaną w zmiennej typu int oraz współczynnik zrównoważenia węzła. Obrazowo strukturę ED przedstawia rysunek obok.

Rysunek 1: Struktura ED

1.2 Klasa Drzewo

Klasa Drzewo została stworzona, by umożliwić implementację drzewa AVL.

1.2.1 Atrybuty klasy Drzewo

Klasa Drzewo posiada wskaźnik na element ED, nazwany *korzen*. Jest to wskazanie elementu drzewa, ktorego pole *ojciec* nie wskazuje na nic. Kolejne węzły w drzewie są realizowane za pomocą struktury ED.

1.2.2 Metody klasy Drzewo

- **void printBT(string sp, string sn, ED* v)** - wyświetla zawartość drzewa. Została ona zapożyczona z implementacji udostępnionej w Internecie[1]
- **bool Dodaj(int element)** - dodaje element o podanej wartości do drzewa, zwraca $false$, gdy element już istnieje w drzewie lub *true*, gdy dodanie elementu powiodło się
- **ED* Usun(ED* wezel)** - usuwa zadany węzeł, a następnie zwraca go. Metoda ta została napisana po uprzednim dokładnym przestudiowaniu kodu odpowiadającej metody z przytaczanej poprzednio strony internetowej, dlatego kod jest podobny do internetowej wersji
- **ED* Usun(int element)** - usuwa węzeł o zadanej wartości, gdy taki nie istnieje, zwraca pusty wskaźnik. Wykorzystuje metodę **ED* Usun(ED* wezel)** oraz **ED* Znajdz(int element)**

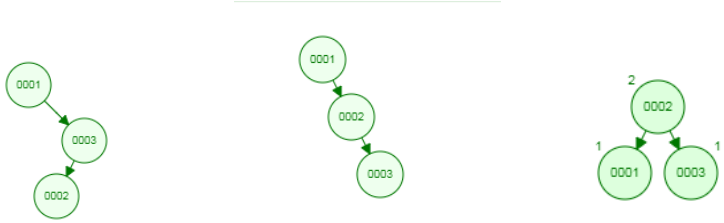
[1] $http://eduinf.waw.pl/inf/alg/001_search/0119.php$

- **int Wysokosc(ED* wezel)** - zwraca wysokosc drzewa, ktorego korzeniem jest podany wezel
- **int Korzen()** - zwraca wartość elementu, na który wskazuje *korzen*
- **int Najwiekszy()** - zwraca największą wartość elementu w drzewie
- **int Najmniejszy()** - zwraca najmniejszą wartość elementu w drzewie
- **ED* RotujLL(ED* wezel)** - realizuje rotację wezłów, które są połączone tylko prawymi gałęziami. Przykład realizacji przedstawia grafika poniżej[2]

(a) przed (b) po

Rysunek 2: Rotacja RR

- **ED* RotujRR(ED* wezel)** - realizuje rotację wezłów, które są połączone tylko lewymi gałęziami (odwrócona rotacja RR)
- **ED* RotujRL(ED* wezel)** - realizuje podwójną rotację wezłów, które są połączone najpierw prawą, a później lewą gałęzią. Jest złożeniem dwóch rotacji: LL, a następniei RR. Przykład realizacji przedstawia grafika poniżej[2]



(a) przed (b) w trakcie (c) po

Rysunek 3: Rotacja RL

- **ED* RotujLR(ED* wezel)**- realizuje podwójną rotację wezłów, które są połączone najpierw lewą, a później prawą gałęzią (odwrócona rotacja LR). Jest złożeniem najpierw rotacji RR, a później LL
- **ED* Poprzednik(ED* wezel)** - zwraca wskaźnik na węzeł, będący poprzednikiem danego. Poprzednik jest węzłem, którego wartość jest mniejsza od wartości danego węzła i jednocześnie jest największą możliwą
- **ED* Znajdz(int element)** - zwraca wskaźnik do węzła o danej wartości. Jeśli węzeł ten nie istnieje, zwraca pusty wskaźnik
- **void WyswietlMenu()** - wyświetla menu do obsługi programu przez użytkownika
- **void Obsluz()** - obsługuje menu, realizując komunikację z użytkownikiem. To jedyna publiczna metody klasy Drzewo

[2]Obrazki powstały dzięki stronie *https : //www.cs.usfca.edu/ galles/visualization/AVLtree.html*

- **void Preorder(ED* wezel)** - realizuje przejscie pre-order przez drzewo od wskazanego węzła, bazując na rekurencji, odwiedzając dany węzeł, wypisuje jego wartość na standardowe wyjście
- **void Postorder(ED* wezel)** - realizuje przejscie post-order przez drzewo od wskazanego węzła, bazując na rekurencji, odwiedzając dany węzeł, wypisuje jego wartość na standardowe wyjście
- **void Inorder(ED* wezel)** - realizuje przejscie in-order przez drzewo od wskazanego węzła, bazując na rekurencji, odwiedzając dany węzeł, wypisuje jego wartość na standardowe wyjście

2 Testy

Zaimplementowane w taki sposób drzewo AVL poddano testom działania. Opis i ich wyniki przedstawiono poniżej.

2.1 Niewielka ilość węzłów, $n = 30$

Do drzewa AVL dodano 40 węzłów o wartościach od 1 do 40, w losowej kolejności. Następnie usunięto z niego wszystkie elementy o wartościach będących wielokrotnościami liczby 4. Wydrukowano zawartość drzewa oraz sprawdzono działanie metod: **Wysokosc(korzen), Korzen(), Najwiekszy(), Najmniejszy(), Preorder(korzen), Postorder(korzen), Inorder(korzen)**. Ponowne przeprowadzenie tego testu wymaga wpisania wartości 111 po uruchomieniu programu. Efekt wypisania przedstawia grafika poniżej:

```
.           0 - koniec                                          .
*                                                               *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Twoj wybor:
111
     @39:0
   @36:1
   @ @ @35:0
   @ @34:0
   @   @33:0
 @32:-1
 @ @29:1
 @   @28:0
  27:1
 @     @26:0
 @   @25:1
 @   @ @ @24:0
 @   @ @22:-1
 @ @21:0
 @ @ @   @20:0
 @ @ @ @19:-1
 @ @ @17:0
 @ @   @ @16:0
 @ @   @15:-1
 @14:0
   @   @13:0
   @ @11:1
   @ @ @ @10:0
   @ @ @8:-1
   @6:0
     @ @5:1
     @ @ @4:0
     @3:0
       @ @2:0
       @1:-1
Wysokosc drzewa: 6
Wartosc w korzeniu: 27
Najwieksza wartosc na drzewie: 39
Najmniejsza wartosc na drzewie: 1

PRE-ORDER
27, 14, 6, 3, 1, 2, 5, 4, 11, 8, 10, 13, 21, 17, 15, 16, 19, 20, 25, 22, 24, 26, 32, 29, 28, 36, 34, 33, 35, 39,

IN-ORDER
1, 2, 3, 4, 5, 6, 8, 10, 11, 13, 14, 15, 16, 17, 19, 20, 21, 22, 24, 25, 26, 27, 28, 29, 32, 33, 34, 35, 36, 39,

POST-ORDER
2, 1, 4, 5, 3, 10, 8, 13, 11, 6, 16, 15, 20, 19, 17, 24, 22, 26, 25, 21, 14, 28, 29, 33, 35, 34, 39, 36, 32, 27,
Twoj wybor:
```

Rysunek 4: Test na malym drzewie

2.2 Duża ilość węzłów, $n = 100.000$

Do drzewa dodano 100.000 węzłów o wartościach od 1 do 100.000 w losowej kolejności. Wyświetlanie zawartości drzewa pozostaje tutaj zbędne, gdyż kontrola poprawności jest znacznie trudniejsza niż poprzednio, sprawdzono jednak działanie tych samych metod, co poprzednio. Efekt:

```
*           0 - koniec                                          *
*                                                               *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Twoj wybor:
112
Wysokosc drzewa: 20
Wartosc w korzeniu: 41930
Najwieksza wartosc na drzewie: 99999
Najmniejsza wartosc na drzewie: 0
Twoj wybor:
```

Rysunek 5: Test na średnim drzewie

Ponowne przeprowadznie tego testu jest możliwe po wpisaniu 112 po uruchomieniu programu.

2.3 Bardzo duża ilość węzłów $n = 1.000.000$

Do drzewa dodano 1.000.000 węzłów o wartościach od 1 do 1.000.000 w losowej kolejności. Wyświetlanie zawartości drzewa również pozostaje tutaj zbędne, gdyż kontrola poprawności jest właściwie niemożliwa, sprawdzono jednak działanie tych samych metod, co poprzednio. Efekt:

```
*           0 - koniec                                          *
*                                                               *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Twoj wybor:
113
Wysokosc drzewa: 24
Wartosc w korzeniu: 430565
Najwieksza wartosc na drzewie: 999999
Najmniejsza wartosc na drzewie: 0
Twoj wybor:
```

Rysunek 6: Test na dużym drzewie

Ponowne przeprowadznie tego testu jest możliwe po wpisaniu 113 po uruchomieniu programu.

3 Uwagi i wnioski

Menu programu nie jest odporne na wpisanie wartości innego typu niż *int*. Gdy użytkownik wpisze np. literę, program kończy swoje działanie.

Wynik testu przy niewielkiej ilości węzłów został poddany dokładnemu oglądowi. Wyrysowane drzewo zachowuje własności drzewa AVL - różnica wysokości obu poddrzew danego węzła nie różni się więcej niż o 1.

Jak widać z wrysowanej struktury, wartość w korzeniu drzewa wynosi 27. Taką wartość zwraca też **Korzen()**. Najmniejsza wartość w drzewie to 1, a największa 39 - te wartości również odpowiadają wartościom, które zwróciy odpowiednie metody.

Dla tego drzewa (ze względu na jego niewielkie rozmiary) można wypisac ręcznie wartości kolejno odwiedzanch węzłów w przejściach **Pre-order, Post-order, In-order**. Efekt tego wypisania pokrywa się z tym, co zwróciły odpowiednie metody.

W drzewie AVL przejście in-order porządkuje elementy według wartości. W teście na małym drzewie AVL wartości te ustawione są w kolejności rosnącej, a najmniejszy i największy element zgadzadzają się co do wartości.

Na niewielkim drzewie można było sprawdzić dokładnie jego działanie. Wpisanie wartości 111 w konsoli po uruchomieniu programu za każdym razem generuje nam nieznacznie różne poddrzewo. Dla kilku takich poddrzew sprawdzono czy program działa poprawnie. Efekt był zadowalający.

Dla testów na dużym i bardzo dużym drzewie, nie można tak łatwo sprawdzić poprawności działania wszystkich metod. W tych drzewach nie było realizowane usuwanie węzłów, dlatego wartość najmniejszego i największego elementu drzewa jest nam znana i wynosi 0 dla najmniejszego elementu, niezależnie od drzewa i 99.999 oraz 999.999 dla odpowiednich drzew. Wartości te zgadzają się.

Wysokość każdego drzewa powinna być nie większa niż $1,44 \cdot log_2(n+2)$.

$$1,44 \cdot log_2(30+2) \approx 7,07 \geqslant 6$$

$$1,44 \cdot log_2(100.000+2) \approx 23,92 \geqslant 20$$

$$1,44 \cdot log_2(1.000.000+2) \approx 28,70 \geqslant 24$$

Widać, że wysokość każdego z drzew jest prawidłowa, mniejsza niż wysokość w najgorszym przypadku.

Przeprowadzone testy pozwalają wydedukować, że program działa zgodnie z założeniami.

Przy tworzeniu drzewa korzystano z zasobów dostępnych w Internecie:

$$http://eduinf.waw.pl/inf/alg/001_search/0119.php$$

$$https://pl.wikipedia.org/wiki/Drzewo_AVL$$

$$http://smurf.mimuw.edu.pl/node/341$$

$$https://www.cs.usfca.edu/\ galles/visualization/AVLtree.html$$

.

Kod programu udostępny pod adresem:

$$https://github.com/234780/PAMSI1$$