

SPRAWOZDANIE

Autor: Michał Deć

Numer indeksu: 169771

Grupa projektowa numer 1

Zadanie projektowe numer 3

Opiekun pracy: **dr inż. prof. PRz Mariusz Borkowski**

Rzeszów, rok 2022

1. Opis problemu

Program ma za zadanie dokonać implementacji struktury danych typu drzewo binarne wraz z wszelkimi potrzebnymi operacjami charakterystycznymi dla tej struktury, takich jak: inicjowanie struktury, dodawanie lub usuwanie elementów, wyświetlanie elementów oraz zliczanie elementów i wyszukiwanie zadanego elementu.

2. Opis teoretyczny zagadnienia

2.1 Czym jest drzewo binarne

Drzewo binarne jest hierarchiczną strukturą danych, którego elementy nazywa się węzłami (nodes) lub wierzchołkami. W drzewie binarnym każdy węzeł posiada co najwyżej dwa następniki (stąd jego nazwa, bo binarny = dwójkowy, zawierający dwa elementy). Następniki te nazywamy potomkami, dziećmi lub węzłami potomnymi (child node) danego węzła – ojca (parent node). Węzeł nie posiadający rodzica nazywany jest korzeniem drzewa binarnego (root node).

2.2 Reguła drzewa binarnego

Wszystkie węzły drzewa spełniają tak zwaną regułę drzewa binarnego. Elementy znajdujące się w lewym poddrzewie są mniejsze od swojego ojca, natomiast elementy prawego poddrzewa są większe od ojca. Reguła ta obowiązuje wszystkie poddrzewa.

2.3 Zasada działania drzewa binarnego

Dzięki przestrzeganiu reguły drzewa binarnego można użyć metody dziel i zwyciężaj (divide and conquer). Biorąc pod uwagę dowolny węzeł to w jego lewym poddrzewie zawsze są wartości mniejsze od rodzica, zaś w prawym poddrzewie wartości większe od rodzica. Jeśli szukamy dowolnej liczby w drzewie, to najpierw sprawdzimy wartość korzenia i dzięki temu ocenimy, w którym poddrzewie należy tej wartości szukać, przez co odrzucimy połowę węzłów do przeszukania.

Drzewo binarne najlepiej jest wykonać na implementacji wskaźnikowej. Do lewego potomka węzła k-tego dostajemy się lewym wskaźnikiem ojca, zaś do prawego potomka węzła k-tego dostajemy się prawym wskaźnikiem ojca. Jeśli węzeł potomny nie istnieje, to odpowiedni wskaźnik ojca wskazuje na NULL.

Funkcje obsługujące drzewo:

- `add_node()` - dodaje element do drzewa.
- `if_node_exists()` - znajduje element w drzewie.
- `del_node_key()` - usuwa dany element.
- `display()` - wydruk elementów drzewa na ekranie.
- `count_all_nodes()` - zlicza ilość węzłów

2.4 Diagram drzewa binarnego

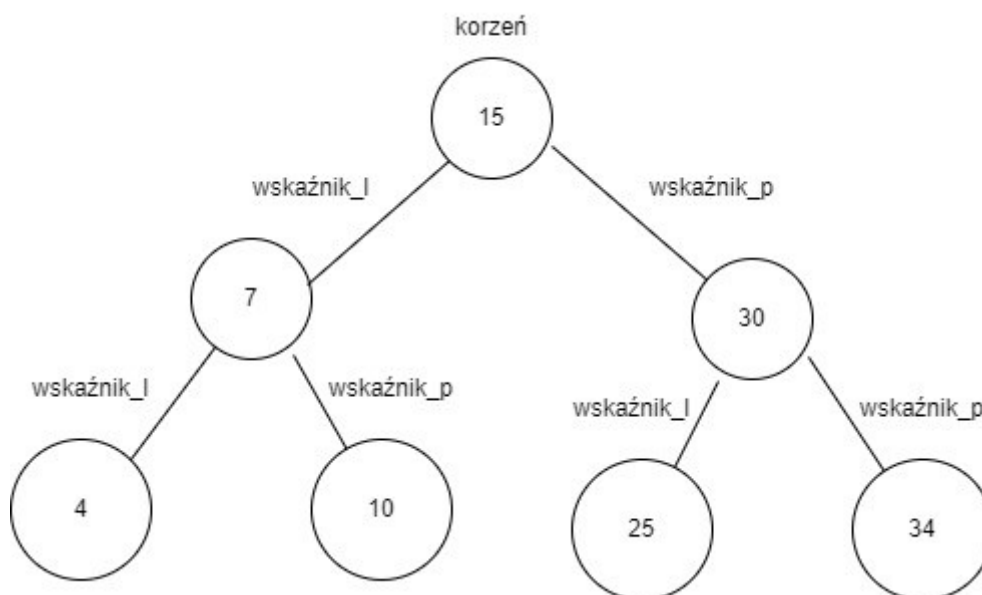
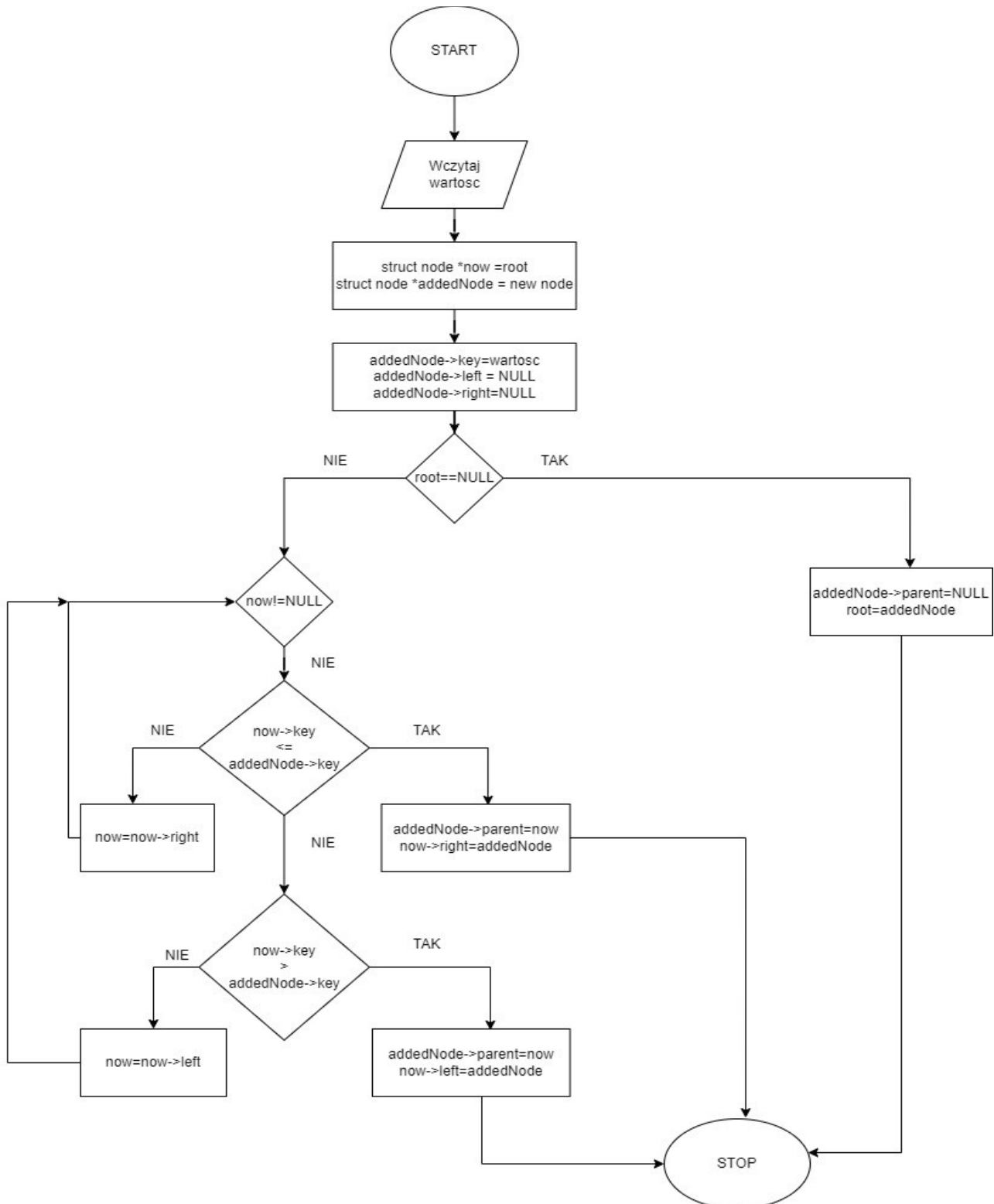


Diagram 1: Graficzne zinterpretowanie drzewa binarnego

Na powyższym diagramie możemy zauważyć zachowanie reguły drzewa binarnego. Dzięki temu, jeśli chcemy wyszukać dowolną liczbę, na przykład 10, to algorytm na początku sprawdzi wartość korzenia i dzięki temu będzie wiedział, w którym poddrzewie ma szukać liczby. Na starcie zaoszczędzimy czasu, ponieważ $10 < 15$, więc przy wyszukiwaniu liczby algorytm odrzuci całą prawą stronę poddrzewa od korzenia, ponieważ wie, że tam jej nie znajdzie. Następnie program sprawdzi wartość w lewym poddrzewie. Skoro $10 > 7$, to algorytm odrzuci z kolei lewą stronę poddrzewa od wartości 7. Przekłada się to na metodę dziel i zwyciężaj.

3. Schemat blokowy funkcji dodającej węzeł



Schemat 1: Schemat blokowy funkcji dodającej węzeł

4. Pseudokod funkcji dodającej węzeł

Wczytaj wprowadzoną wartość.

Utwórz wskaźniki wskazujące na aktualnie rozpatrywany węzeł i nowy węzeł.

Przypisz wartości do węzła.

Sprawdź czy korzeń ma wartość NULL, jeśli tak to przypisz wprowadzoną wartość do korzenia.

W przeciwnym wypadku wykonuj dopóki aktualny węzeł ma wartość inną, niż NULL:

Jeśli wartość nowego węzła jest większa bądź równa od wskazywanego, to sprawdź czy po prawej stronie jest miejsce i dodaj w nie węzeł. W innym wypadku przejdź w głąb drzewa.

Jeśli wartość nowego węzła jest mniejsza od wskazywanego, to sprawdź czy po lewej stronie jest miejsce. Jeśli tak to dodaj w nie aktualny węzeł. W przeciwnym wypadku przejdź w głąb drzewa.

Zakończ działanie funkcji.

5. Przedstawienie możliwości programu

5.1 Prezentacja menu głównego

```
#####MENU#####  
[1] Dodaj wezel  
[2] Usun wezel  
[3] Policz ilosc wezlow  
[4] Znajdz zadana wartosc  
[5] Wyszwietl wartosci wezlow  
[6] Wyjdz  
#####
```

Obraz 1: Wygląd menu głównego

5.2 Inicjowanie struktury

Według zasady drzewa binarnego każdy z węzłów musi przechowywać przynajmniej: zmienną wartości węzła, wskaźnik na rodzica, wskaźnik na lewe dziecko i wskaźnik na prawe dziecko. Jeśli dany węzeł nie ma dziecka po danej stronie, to wskaźnik należy ustawić na wartość NULL. Dzięki temu jesteśmy w stanie zweryfikować czy dany wskaźnik wskazuje na dziecko czy na pustą przestrzeń. Przy dodawaniu węzłów, jeśli wskaźnik na korzeń ma wartość NULL to nowy węzeł staje się automatycznie korzeniem.

```
7  
8 struct node {  
9     int key;  
10    struct node *left;  
11    struct node *right;  
12    struct node *parent;  
13 };  
14  
15 struct node *root = NULL;  
16
```

Obraz 2: Inicjowanie struktury.

5.3 Opis działania menu głównego

W programie mamy zdefiniowany panel sterowania. W zależności od wyboru jaki się wprowadzi program będzie wykonywał różne funkcje. Dla wyboru o numerze program doda nowy węzeł. Dla wyboru o numerze dwa program usunie dany węzeł z drzewa. Dla wyboru o numerze trzy program policzy ilość węzłów w drzewie. Dla wyboru o numerze cztery program wyszuka zadaną wartość w drzewie. Dla wyboru o numerze pięć program wyświetli aktualne wartości w drzewie. Dla wyboru o numerze sześć program zakończy swoje działanie.

```
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243

switch (answ) {
case 1: {
    cout << "Wprowadz wartosc wezla: ";
    cin >> inpu;
    add_node(inpu);
    system("PAUSE");
    break;
}
case 2: {
    cout << "Wprowadz wartosc do usuniecia: ";
    cin >> inpu;
    if (root == NULL) {
        cout << "Brak korzenia - drzewo jest puste" << endl;
    }
    else{
        del_node_key(root, inpu);
    }
    system("PAUSE");
    break;
}
case 3: {
    cout << "Ilosc wezlow = " << count_all_nodes() << endl;
    system("PAUSE");
    break;
}
case 4: {
    cout<<"Wprowadz wartosc do wyszukania: "<<endl;
    cin>>inpu;

    if(if_node_exsists(root, inpu))
    {
        cout<<"Odnaleziono poszukiwana wartosc"<<endl;
    }
    else
    {
        cout<<"Nie odnaleziono poszukiwanej wartosci"<<endl;
    }
}
```

```
244         }
245         system("PAUSE");
246         break;
247     }
248     case 5: {
249         display(root);
250         system("PAUSE");
251         break;
252     }
253     case 6: {
254         isRunning = false;
255         break;
256     }
257     default: {
258         cout << "Zły wybór!" << endl;
259         system("PAUSE");
260         break;
261     }
262 }
263 }
```

Obraz 3: Kod źródłowy panelu sterowania

6. Kod źródłowy funkcji działających w drzewie

6.1 Funkcja dodająca węzeł

```
18  /*
19  Funkcja dodawania węzła - funkcja ta przechodzi przez drzewo
20  w poszukiwaniu wolnego miejsca na węzeł. Jeśli wartość nowego węzła
21  jest mniejsza od aktualnego, przechodzi w lewo, w przeciwnym przypadku
22  przechodzimy w prawo. Gdy natkniemy się na wartość null wstawiamy tam
23  nowy węzeł pamiętając o przypisaniu wskaźników.
24  */
25  void add_node(int val) {
26      //tworzymy wskaźniki wskazujące na aktualnie rozpatrywany węzeł i nowy węzeł
27      struct node *now = root;
28      struct node *addedNode = new node;
29
30      //Przypisujemy wartości do węzła
31      addedNode->key = val;
32      addedNode->left = NULL;
33      addedNode->right = NULL;
34
35      //Jeśli korzeń ma wartość null, nowy węzeł staje się korzeniem
36      if (root == NULL) {
37          addedNode->parent = NULL;
38          root = addedNode;
39          return;
40      }
41      else {
42
43          while (now != NULL) {
44              //wartość nowego węzła jest większa bądź równa od wskazywanego
45              if (now->key <= addedNode->key) {
46                  //jeśli na prawej jest wolne miejsce dodajemy w nie węzeł
47                  if (now->right == NULL) {
48                      addedNode->parent = now;
49                      now->right = addedNode;
50                      return;
51                  }
52                  //w innym przypadku przechodzimy w głąb drzewa
53                  else {
54                      now = now->right;
```

```

55     }
56 }
57 //wartosc nowego wezla jest mniejsza od wskazywanego
58 if (now->key > addedNode->key) {
59     //jesli po lewej jest wolne miejsce dodajemy w nie wezel
60     if (now->left == NULL) {
61         addedNode->parent = now;
62         now->left = addedNode;
63         return;
64     }
65     //w innym przypadku przechodzimy w glab drzewa
66     else {
67         now = now->left;
68     }
69 }
70 }
71 }
72 }

```

Kod 1: Dodanie węzła

6.2 Funkcja usuwająca węzeł

```

74 /*
75 Funkcja usuwania wezla - funkcja przechodzi po drzewie w poszukiwaniu
76 wezla o zadanej wartosci. Jesli szukana wartosc jest mniejsza od wartosci
77 aktualnie wskazywanego wezla, przechodzi w lewo, w przeciwnym przypadku
78 przechodzimy w prawo. Jesli znajdziemy wezel o zadanej wartosci, usuwamy go.
79 W przypadku gdy posiadal jakiegokolwiek dzieci, wstawiamy ja w wolne miejsce.
80 Jesli zadana wartosc nie znajduje sie w drzewie, zwracamy NULL.
81 */
82 struct node* del_node_key(struct node* node, int find) {
83
84     if (node == NULL)
85         return node;
86
87     if (find < node->key) {
88         return node->left = del_node_key(node->left, find); //Rekurencyjne przejscie do szukanej wartosci
89     }
90
91     else if (find > node->key) {
92         return node->right = del_node_key(node->right, find);
93     }
94
95     if (find == node->key) {
96         //Jezeli drzewo sklada sie z samego korzenia
97         if (root->left == NULL && root->right == NULL) {
98             delete(root);
99             root = NULL;
100             return NULL;
101         }
102         //Wstawienie prawego syna na miejsce aktualnie wskazywanego
103         if (node->left == NULL) {
104             struct node *tmp = node->right;
105             delete(node);
106             return tmp;
107         }
108         //Wstawienie lewego syna na miejsce aktualnie wskazywanego
109         else if (node->right == NULL) {
110             struct node *tmp = node->left;
111             delete(node);

```

```

112         return tmp;
113     }
114
115     //Przypadek gdy wskazwany wezel ma dwóch synów
116     struct node* tmp = node->right;
117
118     node->key = tmp->key;
119     while (tmp->left != NULL) {
120         tmp = tmp->left;
121     }
122
123     //Znalezienie następcy
124     node->right = del_node_key(node->right, tmp->key);
125 }
126 return node;
127 }

```

Kod 2: Usuwanie węzła

6.3 Funkcja zliczająca ilość węzłów

```

129 //Zlicz węzły w sposób rekurencyjny
130 int count_node(struct node *now)
131 {
132     int count = 1;
133     if (now->left != NULL) {
134         count += count_node(now->left);
135     }
136     if (now->right != NULL) {
137         count += count_node(now->right);
138     }
139     return count;
140 }
141
142 //Zlicz wszystkie węzły
143 int count_all_nodes()
144 {
145     int count = 0;
146     if (root != NULL) {
147         count = count_node(root);
148     }
149     return count;
150 }

```

Kod 3: Zliczanie węzłów

6.4 Funkcja wyszukująca wartość

```
152 //Wyszukaj zadana wartosc w drzewie
153 bool if_node_exsists(struct node *now, int find)
154 {
155     if (now == NULL)
156         return false;
157
158     if (now->key == find)
159         return true;
160
161     // przejdź do lewego wezła
162     bool res1 = if_node_exsists(now->left, find);
163     // odnaleziono wezel z poszukiwana wartoscia
164     if(res1) return true;
165
166     //jesli nie odnaleziono wartosci po lewej stronie, to sprawdź prawy wezel
167     bool res2 = if_node_exsists(now->right, find);
168
169     return res2;
170 }
```

Kod 4: Wyszukiwanie elementu

6.5 Funkcja wyświetlająca wartości

```
172 //wyświetl elementy drzewa
173 void display(struct node *now) {
174
175     if(now != NULL){
176
177         //przejdź do wezła na lewo
178         display(now->left);
179
180         //wyświetl wartosc wezła
181         cout<<"Wartosc wezła wynosi: "<< now->key<<endl;
182
183         //przejdź do wezła na prawo
184         display(now->right);
185     }
186 }
```

Kod 5: Wyświetlanie elementu

7. Testy działania programu

7.1 Test pierwszy

```
#####MENU#####
[1] Dodaj wezel
[2] Usun wezel
[3] Policz ilosc wezlow
[4] Znajdz zadana wartosc
[5] Wyszwietl wartosci wezlow
[6] Wyjdz
#####
1

Wprowadz wartosc wezla: 5
```

Test 1: Dodawanie węzła

7.2 Test drugi

```
#####MENU#####
[1] Dodaj wezel
[2] Usun wezel
[3] Policz ilosc wezlow
[4] Znajdz zadana wartosc
[5] Wyszwietl wartosci wezlow
[6] Wyjdz
#####
5

Wartosc wezla wynosi: 2
Wartosc wezla wynosi: 3
Wartosc wezla wynosi: 4
Wartosc wezla wynosi: 5
Wartosc wezla wynosi: 6
Wartosc wezla wynosi: 7
Wartosc wezla wynosi: 9
Press any key to continue . . .
```

Test 2: Wyświetlenie węzłów

7.3 Test trzeci

```
#####MENU#####
[1] Dodaj wezel
[2] Usun wezel
[3] Policz ilosc wezlow
[4] Znajdz zadana wartosc
[5] Wyswietl wartosci wezlow
[6] Wyjdz
#####
2

Wprowadz wartosc do usuniecia: 9
Press any key to continue . . .
```

Test 3: Usuń węzeł

7.4 Test czwarty

```
#####MENU#####
[1] Dodaj wezel
[2] Usun wezel
[3] Policz ilosc wezlow
[4] Znajdz zadana wartosc
[5] Wyswietl wartosci wezlow
[6] Wyjdz
#####
4

Wprowadz wartosc do wyszukania:
9
Nie odnaleziono poszukiwanej wartosci
Press any key to continue . . .
```

Test 4: Wyszukanie wartości

7.5 Test piąty

```
#####MENU#####
[1] Dodaj wezel
[2] Usun wezel
[3] Policz ilosc wezlow
[4] Znajdz zadana wartosc
[5] Wyszwietl wartosci wezlow
[6] Wyjdz
#####
3

Ilosc wezlow = 6
Press any key to continue . . .
```

Test 5: Zliczenie węzłów

8. Podsumowanie

Drzewa binarne w informatyce to niezwykle popularny rodzaj struktur danych. Ułatwiają one i przyspieszają wyszukiwanie, a także pozwalają w łatwy sposób operować na posortowanych danych. Dzięki zachowaniu reguły drzewa binarnego można skorzystać z tak zwanej metody dziel i zwyciężaj (divide and conquer), która pozwala w bardzo krótkim czasie wyszukiwać dane. Struktura katalogów w systemie operacyjnym jest właśnie strukturą drzewiastą o określonej hierarchii. Istnieje bardzo wiele zastosowań drzew w przemyśle informatycznym.

Spis treści

1. Opis problemu.....	2
2. Opis teoretyczny zagadnienia.....	2
2.1 Czym jest drzewo binarne.....	2
2.2 Reguła drzewa binarnego.....	2
2.3 Zasada działania drzewa binarnego.....	2
2.4 Diagram drzewa binarnego.....	3
3. Schemat blokowy funkcji dodającej węzeł.....	4
4. Pseudokod funkcji dodającej węzeł.....	5
5. Przedstawienie możliwości programu.....	6
5.1 Prezentacja menu głównego.....	6
5.2 Inicjowanie struktury.....	6
5.3 Opis działania menu głównego.....	7
6. Kod źródłowy funkcji działających w drzewie.....	9
6.1 Funkcja dodająca węzeł.....	9
6.2 Funkcja usuwająca węzeł.....	10
6.3 Funkcja zliczająca ilość węzłów.....	11
6.4 Funkcja wyszukująca wartość.....	12
6.5 Funkcja wyświetlająca wartości.....	13
7. Testy działania programu.....	13
7.1 Test pierwszy.....	13
7.2 Test drugi.....	14
7.3 Test trzeci.....	14
7.4 Test czwarty.....	15
7.5 Test piąty.....	15
8. Podsumowanie.....	16

Diagram

Diagram 1: Graficzne zinterpretowanie drzewa binarnego.....	3
---	---

Schemat blokowy

Schemat 1: Schemat blokowy <i>funkcji dodającej węzeł</i>	4
---	---

Spis obrazów

Obraz 1: Wygląd menu głównego.....	6
Obraz 2: Inicjowanie struktury.....	6
Obraz 3: Kod źródłowy panelu sterowania.....	8

Spis kodu

Kod 1: Dodanie węzła.....	10
Kod 2: Usuwanie węzła.....	11
Kod 3: Zliczanie węzłów.....	11
Kod 4: Wyszukanie elementu.....	12
Kod 5: Wyświetlanie elementu.....	12

Spis testów

Test 1: Dodawanie węzła.....	13
Test 2: Wyświetlenie węzłów.....	13
Test 3: Usuń węzeł.....	14
Test 4: Wyszukanie wartości.....	14
Test 5: Zliczenie węzłów.....	15