Algorytmika Praktyczna. Dynamiczne struktury danych.

Artur Laskowski

10 marca 2022, Poznań

Struktury Danych

Przechowywanie danych

Wykonywanie na danych operacji:

- Wstawiania
- Usuwania
- Wyszukiwania
- ...

Umożliwienie sprawnego działania algorytmów

Najprostsze struktury danych

Zmienna:

- Wstawianie: O(1)
- Wyszukiwanie: O(1)

Tablica:

- Wstawianie: O(1)
- Wyszukiwanie: O(n)

Posortowana tablica:

- Wstawianie: O(n)
- Wyszukiwanie: O(logn)
- Inicjalizacja: O(nlogn)

Stos

Dodawanie i zdejmowanie elementów tylko ze szczytu stosu

Kolejka LIFO - Last In First Out

Uzywana np. do obsługi stosu wywołań funkcji (m.in. w rekurencji)

Stos

https://www.cplusplus.com/reference/stack/

```cpp
#include <iostream>
#include <stack>

using namespace std;

int main() {
    stack <int> s;

    for(auto el: {1, 2, 3, 4}) {
        s.push(el);
    }

    cout << s.top() << endl;
    s.push(7);
    cout << s.top() << endl;
    s.pop();

    while(!s.empty()) {
        cout << s.top() << " ";
        s.pop();
    } cout << endl;

    return 0;
}
```

```
apilaskowski@LAPTOP-J4S1ABQ6:/mnt/c/Users/User/Downloads$ g++ -std=c++2a main.cpp
apilaskowski@LAPTOP-J4S1ABQ6:/mnt/c/Users/User/Downloads$ ./a.out
4
7
4 3 2 1
apilaskowski@LAPTOP-J4S1ABQ6:/mnt/c/Users/User/Downloads$
```

Kolejka

Wstawianie elementu na końcu, zdejmowanie z początku

Kolejka FIFO - First In First Out

Uzywana np. do obsługi zadań o równym priorytecie

Kolejka

https://www.cplusplus.com/reference/queue/

```cpp
#include <iostream>
#include <queue>

using namespace std;

int main() {
    queue <int> s;

    for(auto el: {1, 2, 3, 4}) {
        s.push(el);
    }

    cout << s.front() << endl;
    s.push(7);
    cout << s.front() << endl;
    s.pop();

    while(!s.empty()) {
        cout << s.front() << " ";
        s.pop();
    } cout << endl;

    return 0;
}
```

```
apilaskowski@LAPTOP-J4S1ABQ6:/mnt/c/Users/User/Downloads$ g++ -std=c++2a main2.cpp
apilaskowski@LAPTOP-J4S1ABQ6:/mnt/c/Users/User/Downloads$ ./a.out
1
1
2 3 4 7
apilaskowski@LAPTOP-J4S1ABQ6:/mnt/c/Users/User/Downloads$
```

Lista

Elementy ustawione jeden za drugim

Każdy element wskazuje na swojego

- następnika - lista jednokierunkowa
- opcjonalnie poprzednika - lista dwukierunkowa

`https://www.cplusplus.com/reference/forward_list/`

`https://www.cplusplus.com/reference/list/`

```
3   #include <forward_list>
4   #include <list>
```

Wskaźniki

```
 5   int main() {
 6       int x = 5;              // deklaracja zmiennej typu int
 7       int* px = &x;           // przypisanie zmiennej do wskaźnika
 8
 9       cout << x << endl;      // wypisanie wartości zmiennej
10       cout << px << endl;     // wypisanie ADRESU wskaźnika
11       cout << *px << endl;    // wypisanie wartości pod zadanym ADRESEM
12   }
```

```
apilaskowski@LAPTOP-J4S1ABQ6:/mnt/c/Users/User/Downloads$ g++ -std=c++2a main3.cpp
apilaskowski@LAPTOP-J4S1ABQ6:/mnt/c/Users/User/Downloads$ ./a.out
5
0x7ffd20cb228c
5
apilaskowski@LAPTOP-J4S1ABQ6:/mnt/c/Users/User/Downloads$ ./a.out
5
0x7ffdb0d5de4c
5
```

Czas życia zmiennej

```
void dodaj1(int a, int b) {
    int c;          // Tworzenie zmiennej c (przydzielenie jej miejsca w pamięci)
    c = a + b;      // W miejscu przydzielonym zmiennej c zapisywana jest suma
}
                    // Po wyjściu z funkcji wartość zmiennej c jest bezpowrotnie utracona
```

Czas życia zmiennej

```
void dodaj2(int a, int b) {
    int *pc;         // tworzony jest wskaźnik na zmienne typu int o nazwie pc
    pc = new int;    // przydzielana jest pamięć dla zmiennej typu int,
                     //      a jej adres zapamiętujemy w pc

    *pc = a + b;     // w miejscu w pamięci przydzielonej zmiennej na którą wskazuję pc
                     //      przypisujemy sumę
}
                     // Po wyjściu z funkcji w miejscu wskazywanym przez pc pozostaje suma,
                     //      ale jest bezpowrotnie utracona, ponieważ utraciliśmy adres tej zmiennej.
```

Czas życia zmiennej

```
int* dodaj3(int a, int b) {
    int *pc;
    pc = new int;
    *pc = a + b;

    return pc;      // Wychodząc zwracamy wskaźnik na miejsce w pamięci, gdzie jest zapisana suma
}
                    // Po wyjściu z funkcji możemy odczytać wskaźni na miejsce w pamięci,
                    //      gdzie zapisana jest suma
```

Pamiętaj!

Przy deklaracji:

int *pc;

pc jest typu ***int****

****pc*** jest typu ***int***

Struktury

Struktury służą do zapisywania danych opisanych wieloma wartościami.

```
struct adres {
    int numer, mieszkanie;
    string ulica;
};
```

Do ***pól*** odwołujemy się za pomocą kropki:

```
adres a;
a.numer = 30;
a.mieszkanie = 3;
a.ulica = "Niepodleglosci";
```

Struktury

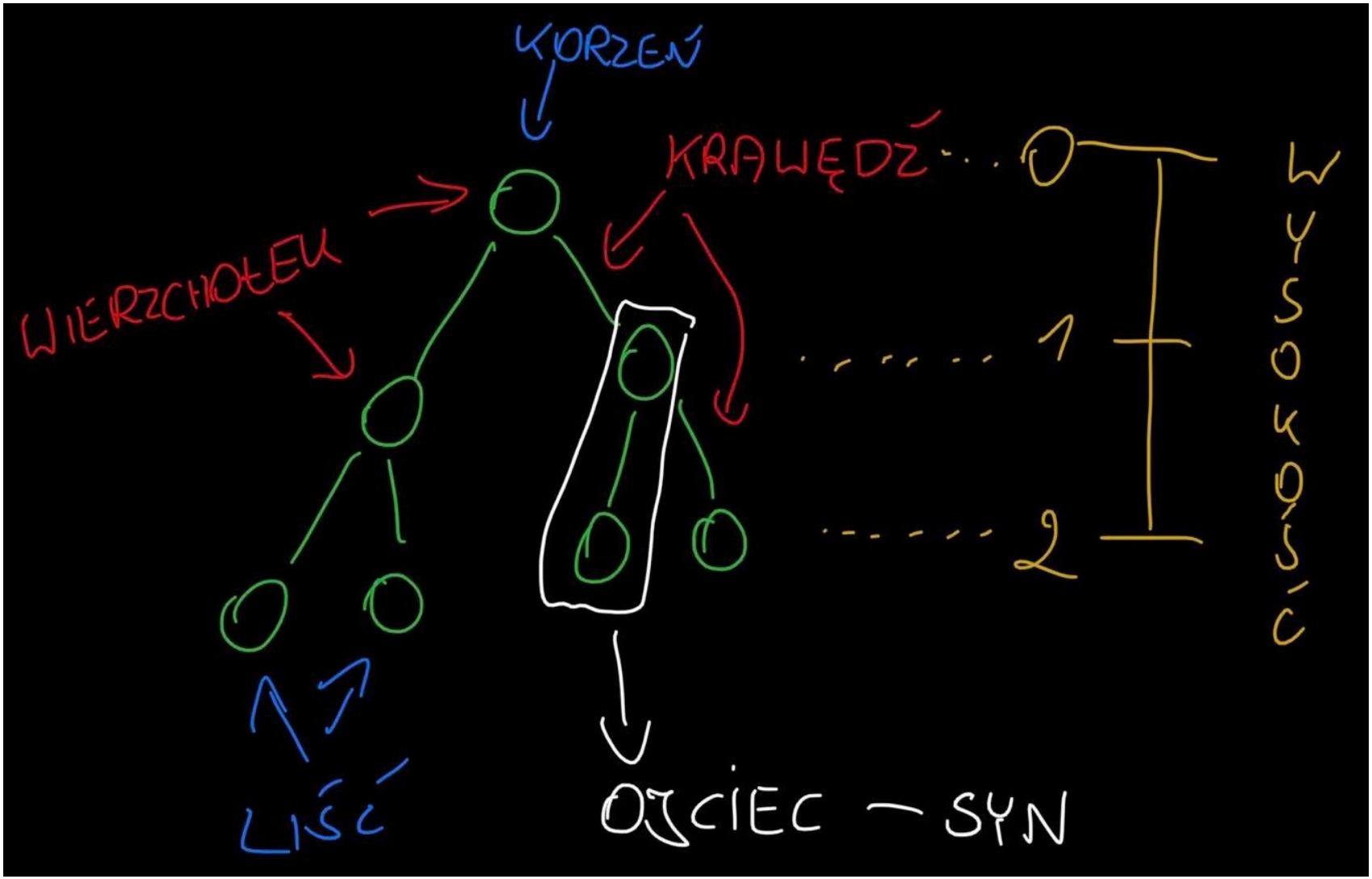
Możliwe jest również dynamiczne przydzielenie pamięci strukturom:

```
adres *a;
a = new adres;
(*a).numer = 30;
```

Zamiast gwiazdki można użyć strzałki, aby zapis był bardziej czytelny:

```
a->numer = 30;
```

Przypomnienie drzewo



Drzewa BST

Drzewo binarne o następującej własności:

Lewe poddrzewo zawiera wartości mniejsze

Prawe poddrzewo zawiera wartości większe

Drzewa BST - wstawianie

Schodzimy w dół, aż znajdziemy puste miejsce dla elementu

Złożoność obliczeniowa?

- ***O(h)***, czyli pesymistycznie ***O(n)***

Drzewa BST - wyszukiwanie

Analogicznie do wstawiania elementu

Złożoność obliczeniowa?

- ***O(h)***, czyli pesymistycznie ***O(n)***

Drzewa BST - przeszukiwanie całego drzewa

Przeszukujemy drzewo w kolejności Lewe, a następnie Prawe poddrzewo.
Wykorzystujemy trzy sposoby na wypisanie elementów drzewa.
Wypisujemy element:

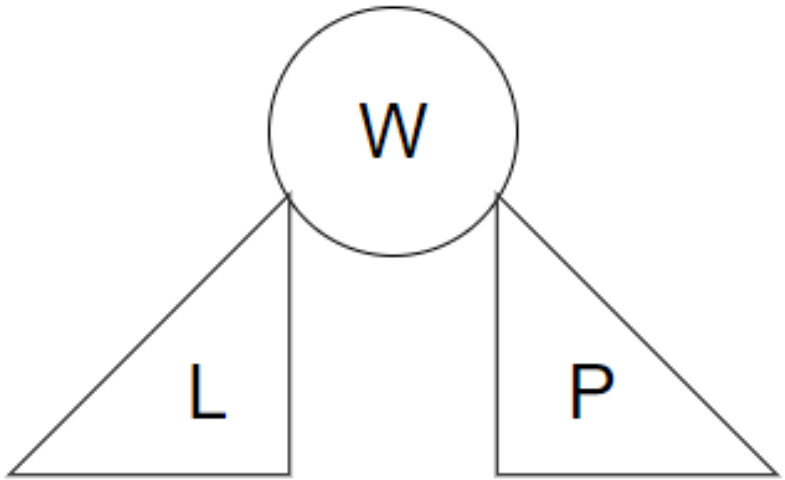
- Przed wypisaniem lewego poddrzewa - ***pre-order***,
- Między wypisaniem prawego, a lewego poddrzewa - ***in-order***,
- Po wypisaniu prawego podddrzewa - ***post-order***.

Drzewa BST - przeszukiwanie całego drzewa

Pre-order: W, L, P

In-order: L, W, P

Post-order: L, P, W



Drzewa BST - implementacja

```
struct Node {
    int value;
    Node *left, *right;
};

Node *root;
```

Drzewa BST - implementacja

```
void insert(int val) {
    Node *new_node = new Node;
    new_node->value = val;
    new_node->left = nullptr;
    new_node->right = nullptr;

    if(root == nullptr) {
        root = new_node;
        return;
    }

    Node *parent, *son;
    son = root;
    parent = nullptr;
    do {
        parent = son;
        if(val < parent->value) {
            son = parent->left;
        } else {
            son = parent->right;
        }
    } while(son != nullptr);

    if(val < parent->value) {
        parent->left = new_node;
    } else {
        parent->right = new_node;
    }
}
```

Drzewa BST - implementacja

```
int main() {
    root = nullptr;

    insert(5);
    insert(7);
    insert(2);
}
```

```
apilaskowski@LAPTOP-J4S1ABQ6:/mnt/c/Users/User/Downloads$ g++ -std=c++2a main5.cpp
apilaskowski@LAPTOP-J4S1ABQ6:/mnt/c/Users/User/Downloads$ ./a.out
apilaskowski@LAPTOP-J4S1ABQ6:/mnt/c/Users/User/Downloads$
```

Drzewa BST - Pre-order - implementacja

```
void pre_order(Node *starting = root) {
    cout << starting->value << " ";
    if(starting->left != nullptr) {
        pre_order(starting->left);
    }
    if(starting->right != nullptr) {
        pre_order(starting->right);
    }
}
```

Drzewa BST - In-order - implementacja

```
void in_order(Node *starting = root) {
    if(starting->left != nullptr) {
        in_order(starting->left);
    }
    cout << starting->value << " ";
    if(starting->right != nullptr) {
        in_order(starting->right);
    }
}
```

Drzewa BST - Post-order - implementacja

```
void post_order(Node *starting = root) {
    if(starting->left != nullptr) {
        post_order(starting->left);
    }
    if(starting->right != nullptr) {
        post_order(starting->right);
    }
    cout << starting->value << " ";
}
```

Drzewa BST - implementacja

```
int main() {
    root = nullptr;

    for(auto el: {6, 3, 5, 2, 8}) {
        insert(el);
    }

    pre_order(); cout << endl;
    in_order(); cout << endl;
    post_order(); cout << endl;
}
```

Drzewa BST - implementacja

```
apilaskowski@LAPTOP-J4S1ABQ6:/mnt/c/Users/User/Downloads$ g++ -std=c++2a main5
apilaskowski@LAPTOP-J4S1ABQ6:/mnt/c/Users/User/Downloads$ ./a.out
6 3 2 5 8
2 3 5 6 8
2 5 3 8 6
apilaskowski@LAPTOP-J4S1ABQ6:/mnt/c/Users/User/Downloads$
```

Laboratoria

https://www.hackerrank.com/ap-01-2022