

AKADEMIA NAUK STOSOWANYCH W NOWYM SĄCZU

Wydział Nauk Inżynierskich
Katedra Informatyki

DOKUMENTACJA PROJEKTOWA ZAAWANSOWANE PROGRAMOWANIE

Algorytm listy dwukierunkowej z zastosowaniem GitHub

Autor:
Mateusz Stanek
Dawid Szoldra

Prowadzący:
mgr inż. Dawid Kotlarski

Nowy Sącz 2024

Spis treści

1. Ogólne określenie wymagań	3
2. Analiza problemu	4
2.1. Drzewo BTS	4
2.2. Git	4
2.3. Doxygen	6
3. Projektowanie	7
3.1. Implementacja drzewa BTS	7
3.2. Git	7
3.3. Doxygen	7
4. Implementacja	9
4.1. Ogólne informacje o implementacji klasy	9
4.2. Ciekawe fragmenty kodu	10
5. Wnioski	12
Literatura	13
Spis rysunków	14
Spis tabel	15
Spis listingów	16

1. Ogólne określenie wymagań

Celem projektu jest stworzenie programu tworzące drzewo BTS działające na stercie oraz kontrolowanie jego wersji za pomocą narzędzia git. Do zadań programu będzie należało: dodawanie elementu, usuwanie elementu, ęcie całego drzewa, szukanie drogi do wskazanego elementu, wyświetlenie drzewa graficznego, zapis do pliku tekstowego wygenerowanego drzewa.

Program będzie podzielony na 3 pliki: plik main, plik z drzewem oraz plik który będzie zaczytywał drzewo to pliku tekstowego.

Wynikiem projektu powinno być działające drzewo BTS, repozytorium git z kodem oraz dokumentacją w doxygenie.

2. Analiza problemu

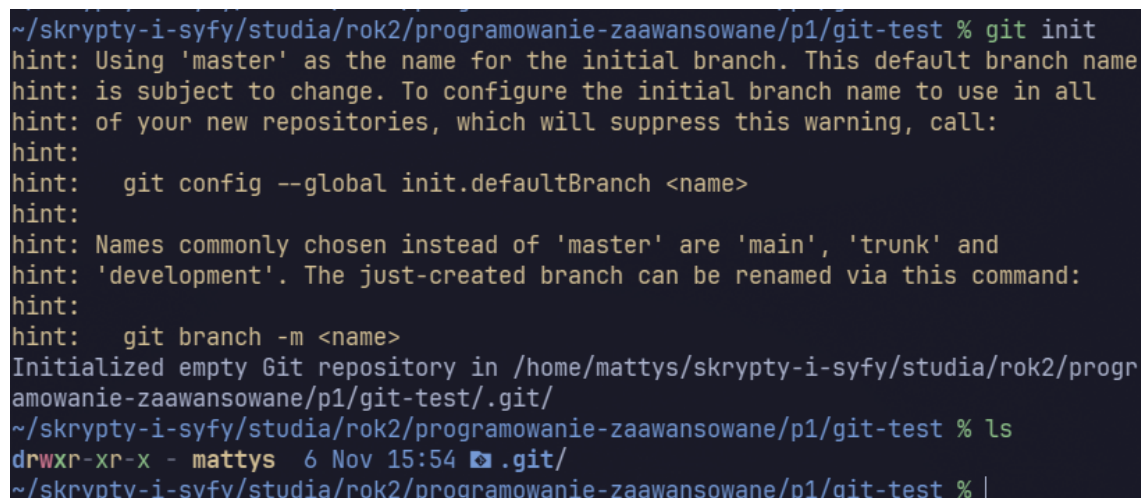
2.1. Drzewo BTS

Tutaj będzie info o drzewie BTS

2.2. Git

Kolejnym konceptem, którym zajmuje się projekt jest narzędzie git[1]. Pozwala ono zarządzać poszczególnymi wersjami projektów. Głównym korzeniem gita jest system commitów, czyli zapisania zmian w pliku w stosunku do commita starszego. To, w połączeniu z jego innymi możliwościami pozwala na tworzenie długich i skomplikowanych osi czasu danych projektów.

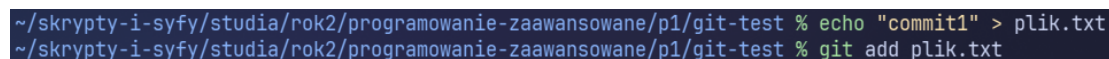
Użycie gita można zademonstrować na prostym przykładzie. Tworzymy katalog a w nim repozytorium, używając komendy `git init`, jak widać na rys. 2.1.



```
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /home/mattys/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test/.git/
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % ls
drwxr-xr-x - mattys  6 Nov 15:54 .git/
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % |
```

Rys. 2.1. Puste repozytorium git

Stwórzmy jakiś plik i dodajmy go do repozytorium. Plik można dodać do repozytorium komendą `git add`



```
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % echo "commit1" > plik.txt
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % git add plik.txt
```

Rys. 2.2. Stworzenie pliku w repozytorium

Następnie należy scommitować zmiany.

```
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % git commit -am "utworzenie plik.txt"
[master (root-commit) bb3b898] utworzenie plik.txt
 1 file changed, 1 insertion(+)
 create mode 100644 plik.txt
```

Rys. 2.3. Commit nr. 1

Na rysunku 2.3 użyta komenda `git commit` commituje wszystkie dodane pliki (-a) z jakimś komunikatem (-m).

```
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % echo "commit2" >> plik.txt
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % git commit -am "uzupelnienie plik.txt"
[master 40694c2] uzupelnienie plik.txt
 1 file changed, 1 insertion(+)
```

Rys. 2.4. Commit nr. 2

Na rys. 2.4, został utworzony kolejny commit, dodający zmiany do `plik.txt`.

```
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % git log
commit 40694c2e73758368445cb817f4524ee5c5afbece (HEAD -> master)
Author: mattys1 <mattys0082@gmail.com>
Date:   Wed Nov 6 16:26:07 2024 +0100

    uzupelnienie plik.txt

commit bb3b898df760395b5763575e204c3c43b513d2f6
Author: mattys1 <mattys0082@gmail.com>
Date:   Wed Nov 6 16:12:31 2024 +0100

    utworzenie plik.txt
```

Rys. 2.5. Log gita

Jak na rys. 2.5 jest pokazane, używając komendy `git log`, można wyświetlić log commitów w repozytorium.

```
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % git checkout bb3b898df760395b5763575e204c3c43b513d2f6
Note: switching to 'bb3b898df760395b5763575e204c3c43b513d2f6'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

    git switch -c <new-branch-name>

Or undo this operation with:

    git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at bb3b898 utworzenie plik.txt
~/skrypty-i-syfy/studia/rok2/programowanie-zaawansowane/p1/git-test % echo plik.txt
plik.txt
```

Rys. 2.6. Demonstracja checkout

Jak widać na rys. 2.6, komenda `git checkout`, pozwala na przejście repozytorium w inny stan, w tym przypadku przechodzi się do commita o danym ID, pokazanym na rys. 2.5. Jako, że jest to pierwszy commit, nie ma w nim zmian z drugiego.

2.3. Doxygen

Doxygen[2] jest narzędziem automatycznie generującym dokumentację programu z komentarzy w kodzie źródłowym. Potrafi on generować strony HTML, gdzie można dynamicznie nawigować się między różnymi częściami kodu oraz pliki $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$, które można konwertować na różne, statyczne formaty.

3. Projektowanie

3.1. Implementacja drzewa BTS

Do zaimplementowania drzewa BTS zostanie użyty Język C++ z kompilatorem g++. Wersja standardu C++ to C++23. Wersja ta została użyta, ze względu na zawartą w niej funkcję `std::print()`. Jako, że projekt ma być rozdzielony na dwa pliki, zostanie zastosowany CMake w celu automatyzacji procesu budowania. CMake pozwala na generowanie plików budujących dany projekt, zgodnie z określoną konfiguracją. Oszczędza to programiście, szczególnie przy większych projektach, manualne pisanie Makefile'ów.

Edytorem będzie program Neovim oraz Visual Studio 2022. Jest to terminalowy edytor tekstu z możliwością poszerzenia funkcjonalności przy użyciu wszelkiego rodzaju pluginów. Wybrany został, dlatego że jest on już skonfigurowany na moim komputerze zgodnie z moimi preferencjami.

3.2. Git

Dla ułatwienia pracy, zastosowany został front-end dla gita o nazwie lazygit. Jest to terminalowy program, którego główną zaletą jest łatwa nawigacja przy użyciu klawiatury. Ponadto, jest on lekki i szybki.

3.3. Doxygen

Konfiguracja dla Doxygena jest wygenerowana przy użyciu programu doxywizard, pokazany na rys. 3.1, pozwalającego na graficzne zmienianie ustawień. Po wygenerowaniu konfiguracji, Doxygen wywoływany jest przy użyciu komendy.

Specify the working directory from which doxygen will run

Configure doxygen using the Wizard and/or Expert tab, then switch to the Run tab to generate the documentation

Wizard Expert Run

Topics

- Project
- Mode
- Output
- Diagrams

Provide some information about the project you are documenting

Project name: My Project

Project synopsis:

Project version or id:

Project logo: Select... No Project logo selected.

Specify the directory to scan for source code

Source code directory: Select...

☐ Scan recursively

Specify the directory where doxygen should put the generated documentation

Destination directory: Select...

Previous Next

Rys. 3.1. Interfejs programu doxywizard

4. Implementacja

4.1. Ogólne informacje o implementacji klasy

Lista jest zaimplementowana jako jeden plik `.hpp`. Nie jest podzielona na plik implementacji oraz nagłówek, ponieważ jest ona szablonem. Deklaracja klasy oraz prywatne elementy wyglądają następująco:

```
1  template<typename T>
2  class DoubleLinkedList {
3  private:
4      struct Node {
5          Node* previous;
6          Node* next;
7          T contents;
8
9          Node(Node* oPrevious, Node* oNext, T oContents):
10             previous { oPrevious },
11             next { oNext },
12             contents { oContents } {}
13
14         ~Node(void) {
15             delete next;
16         }
17     };
18
19     Node* head;
20     Node* tail;
21
22     Node* get_node_at(size_t index) {
23         Node* currentNode { head };
24         for(size_t i {}; i < index; i++) {
25             currentNode = currentNode->next;
26         }
27
28         return currentNode;
29     }
30
31 public:
32     ...
```

Listing 1. Deklaracja szblonu listy

Jak widać w kodzie 1., Klasa jest wrapperem dla structa `Node`. Struct ten ma dwa wskaźniki - `next` dla elementu następnego i `previous` dla elementu poprzedniego.

Jego konstruktor pozwala od razu ustawiać zawartość oraz te wskaźniki. Metoda `get_node_at()` jest pomocniczą metodą pozwalającą uzyskać wskaźnik do `Node` o danym indeksie, podążając w przód od wskaźnika `head`, `index` razy.

Manipulacje strukturą listy odbywają się poprzez szereg metod publicznych. Wiele z nich posiada podobną strukturę. Za przykład jednej z nich można wziąć `prepend()`:

```

1  void prepend(const T& item) {
2      if(head == nullptr) {
3          head = new Node { nullptr, nullptr, item };
4          tail = head;
5          return;
6      }
7
8      head = new Node { nullptr, head, item };
9      head->next->previous = head;
10 }
```

Listing 2. Kod `prepend()`

Metoda z fragmentu nr. 2 ma na celu wstawienie elementu, którego wartość jest zawarta w parametrze `item` na początku listy. Na początku metody, sprawdzane jest czy lista jest pusta - wtedy `head == nullptr`. Jeżeli jest, trzeba stworzyć nowego `Node` na miejscu `heada` z zawartością będącą parametrem `item`. Jeżeli `head` już istnieje, to też tworzy się nowego `Nodea` na jego miejscu, lecz jako wskaźnik `next` ustawia się adres starego `heada`. Potem we wskaźniku `previous` starego `heada` ustawia się adres nowego `heada`. Ten zabieg efektywnie sprawił że stary `head` jest drugi w kolejności listy.

4.2. Ciekawe fragmenty kodu

W metodzie `pop_at()`, mającej na celu usunięcie `Nodea` o danym indeksie, wywołany jest destruktork danego `Nodea` w taki sposób, aby - ze względu na jego rekursywny charakter - nie usuwać następnych elementów listy. Przy czym ciągłość listy musi być zachowana. Kod metody wygląda następująco:

```

1
2  void pop_at(size_t index) {
3      Node* toPop { get_node_at(index) };
4      if(toPop == head) {
5          rpop();
6          return;
7      } else if(toPop == tail) {
```

```
8     pop();
9     return;
10  }
11
12  toPop->previous->next = toPop->next;
13  toPop->next->previous = toPop->previous;
14  toPop->next = nullptr;
15
16  delete toPop;
17 }
```

Listing 3. Kod pop_at()

Jak widać na fragmencie nr. 3, takie zabiegi wymagają niezłej zabawy ze wskaźnikami.

5. Wnioski

- Przy rebaseowaniu, należy zwrócić uwagę, jakie pliki zostaną zmienione.
- Stashe w git są zbawieniem.

Bibliografia

- [1] *Strona Gita*. URL: <https://git-scm.com/>.
- [2] *Strona Doxygena*. URL: <https://www.doxygen.nl/>.

Spis rysunków

2.1. Puste repozytorium git	4
2.2. Stworzenie pliku w repozytorium	4
2.3. Commit nr. 1	5
2.4. Commit nr. 2	5
2.5. Log gita	5
2.6. Demonstracja checkout	6
3.1. Interfejs programu doxywizard	8

Spis tabel

Spis listingów

1.	Deklaracja szblonu listy	9
2.	Kod <code>prepend()</code>	10
3.	Kod <code>pop_at()</code>	10