Algorytmy i struktury danych

**Aldona Świrad**

Struktury danych: lista jednokierunkowa

Projekt

Rzeszów, 2022

Spis treści

**1.Wstęp5**

**2.Opis problemu6**

2.1.Opis podstaw teoretycznych zagadnienia 6

2.2.Opis szczegółów implementacji problemu6

**3.Badanie algorytmu7**

3.1. Dodawanie elementu na początku listy7

3.2. Usuwanie elementu z początku listy7

3.3. Dodawanie elementu na koniec listy8

3.4. Usuwanie ostatniego elementu listy8

3.5. Dodawanie nowego elementu przed wybranym elementem listy9

3.6. Dodawanie nowego elementu za elementem wybranym9

3.7. Usuwanie wybranego elementu listy10

**4.Pseudokod11**

**4.Wnioski18**

**Bibliografia19**

1. Wstęp

Projekt napisany w języku programowania C++, w środowisku Code::Blocks IDE, realizowany w ramach przedmiotu „Algorytmy i struktury danych” na kierunku Inżynieria i analiza danych, semestr I, grupa 8.

2. Opis problemu

Lista jednokierunkowa jest strukturą o dynamicznie zmieniającej się wielkości. Listę można opisać jako uszeregowany zbiór elementów. Każdy element zawiera jakieś dane oraz wskazuje na swojego następcę. Cechą listy jednokierunkowej jest to, że można przeglądać ją tylko w jedną stronę, od początku do końca.

2.1. Opis podstaw teoretycznych zagadnienia

Do rozwiązania problemu listy jednokierunkowej wykorzystano tablice statyczne i dynamiczne, pętle „for” oraz instrukcje warunkowe „if”, klasy, obiekty, struktury. W celu sprawnego testowania algorytmu, za pomocą instrukcji warunkowej „switch” stworzony został program z przejrzystym menu, gdzie użytkownik sam może zdecydować o źródle pochodzenia danych wejściowych.

2.2. Opis szczegółów implementacji problemu

Możliwości listy jednokierunkowej, jakie zostały zaimplementowane:

1. Zliczanie liczby elementów listy
2. Dodawanie elementu na początku listy
3. Usuwanie elementu z początku listy
4. Dodawanie elementu na koniec listy
5. Usuwanie ostatniego elementu listy
6. Dodawanie nowego elementu przed wybranym elementem listy
7. Dodawanie nowego elementu za elementem wybranym
8. Usuwanie wybranego elementu listy

W dalszych rozważaniach będziemy używali list jako obiektów ze względu na wygodę programowania. Obiekt jest strukturą danych, która posiada pola danych oraz funkcje i procedury obsługujące te pola. Do funkcji i procedur obiektu odwołujemy się identycznie jak do pól struktury.

3. Badanie algorytmu

3.1. Dodawanie elementu na początku listy

Obraz zawierający tekst

Opis wygenerowany automatycznie

3.2. Usuwanie elementu z początku listy

Obraz zawierający tekst

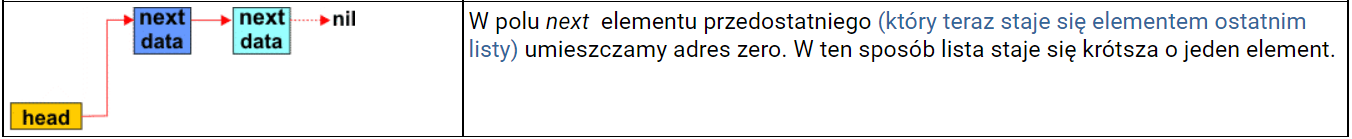
Opis wygenerowany automatycznie

3.3. Dodawanie elementu na koniec listy

Obraz zawierający tekst

Opis wygenerowany automatycznie

3.4. Usuwanie ostatniego elementu listy

Obraz zawierający tekst

Opis wygenerowany automatycznie

3.5. Dodawanie nowego elementu przed wybranym elementem listy

Obraz zawierający tekst

Opis wygenerowany automatycznieObraz zawierający tekst

Opis wygenerowany automatycznie

3.1. Dodawanie nowego elementu za elementem wybranym

3.6. Dodawanie nowego elementu za elementem wybranym

Obraz zawierający tekst

Opis wygenerowany automatycznie

3.6. Usuwanie wybranego elementu listy

Obraz zawierający stół

Opis wygenerowany automatycznie

4. Pseudokod

**Liczba elementów listy**

|  |  |  |
| --- | --- | --- |
| K01: | *c*  ← 0 | *zerujemy licznik* |
| K02: | **Dopóki** *p*  ≠ nil, **wykonuj** kroki K03...K04 | *w pętli przechodzimy przez kolejne elementy listy* |
| K03: | *c*  ← *c*  + 1 | *zwiększ licznik* |
| K04: | *p*  ← ( *p*→*next*  ) | *w p umieść zawartość pola next elementu wskazywanego przez p* |
| K05: | **Zakończ z wynikiem***c* | *koniec, wynik w c* |

Dodawanie elementu na początku listy

|  |  |  |
| --- | --- | --- |
| K01: | Utwórz nowy element listy |  |
| K02: | *p*  ← adres nowego elementu |  |
| K03: | ( *p*→*data*  ) ← *v* | *umieszczamy dane w elemencie* |
| K04: | ( *p*→*next*  ) ← *head* | *następnikiem będzie bieżący pierwszy element listy* |
| K05: | *head*  ← *p* | *ustawiamy początek listy na nowy element* |
| K06: | **Zakończ** |  |
|  |  |  |

**Usuwanie elementu z początku listy**

|  |  |  |
| --- | --- | --- |
| K01: | p ← *head* | *zapamiętaj pierwszy element* |
| K02 | **Jeśli** p = nil, **to zakończ** | *zakończ, jeśli lista jest pusta* |
| K03: | *head*  ← ( *p*→*next*  ) | *początkiem listy będzie element następny* |
| K04: | Usuń z pamięci element wskazany przez *p* |  |
| K05: | **Zakończ** |  |

**Dodawanie elementu na koniec listy**

|  |  |  |
| --- | --- | --- |
| K01: | Utwórz nowy element |  |
| K02: | *e*  ← adres nowego elementu |  |
| K03: | ( *e*→*next*  ) ← nil | *inicjujemy pola nowego elementu* |
| K04: | ( *e*→*data*  ) ← *v* |  |
| K05; | *p*  ← *head* | *w p ustawiamy początek listy* |
| K06: | **Jeśli** *p*  ≠ nil, **to idź do** kroku K09 | *czy lista jest pusta?* |
| K07: | *head*  ← *e* | *jeśli tak, to nowy element będzie pierwszym elementem listy* |
| K08: | **Zakończ** |  |
| K09: | **Dopóki** ( *p*→*next*  ) ≠ nil, **wykonuj** *p*  ← ( *p*→*next*  ) | *inaczej przechodzimy do ostatniego elementu listy* |
| K10: | ( *p*→*next*  ) ← *e* | *dołączamy nowy element za ostatnim na liście* |
| K11: | **Zakończ** |  |

**Usuwanie ostatniego elementu listy**

|  |  |  |
| --- | --- | --- |
| K01: | *p*  ← *head* | *pobieramy do p adres początku listy* |
| K02: | **Jeśli** *p*  = nil, **to zakończ** | *jeśli lista jest pusta, kończymy* |
| K03: | **Jeśli** ( *p*→*next*  ) ≠ nil, **to idź do** kroku K07 | *sprawdzamy, czy lista jest jednoelementowa* |
| K04: | Usuń z pamięci element wskazywany przez *p* |  |
| K05; | *head*  ← nil | *lista jednoelementowa staje się listą pustą* |
| K06: | **Zakończ** |  |
| K07: | **Dopóki** ( ( *p*→*next*  )→*next*  ) ≠ nil, **wykonuj** *p*  ← ( *p*→*next*  ) | *idziemy do przedostatniego elementu* |
| K08: | Usuń z pamięci element wskazywany przez ( *p*→*next*  ) | *usuwamy następny element, czyli ostatni* |
| K09: | ( *p*→*next*  ) ← nil | *przedostatni element nie ma już następnika* |
| K10: | **Zakończ** |  |

**Dodawanie nowego elementu przed wybranym elementem listy**

|  |  |  |
| --- | --- | --- |
| K01: | *p*  ← *head* | *pobieramy do p adres początku listy* |
| K02: | **Jeśli** *p*  ≠ *e*, **to idź do**kroku K05 | *sprawdzamy, czy e nie jest pierwszym elementem listy* |
| K03: | Wstaw nowy element na początku listy |  |
| K04: | **Zakończ** |  |
| K05; | **Dopóki** ( *p*→*next*  ) ≠ *e*, **wykonuj** *p*  ← ( *p*→*next*  ) | *przechodzimy do elementu poprzedzającego e* |
| K06: | Utwórz nowy element |  |
| K07: | ( *p*→*next*  ) ← adres nowego elementu | *nowy element wstawiamy za element poprzedzający* |
| K08: | ( ( *p*→*next*  )→*next*  )*← e* | *inicjujemy nowy element* |
| K09: | ( ( *p*→*next*  )→*data*  ) *← v* |  |
| K10: | **Zakończ** |  |

**Dodawanie nowego elementu za elementem wybranym**

|  |  |  |
| --- | --- | --- |
| K01: | Utwórz nowy element |  |
| K02: | *p*  ← adres nowego elementu |  |
| K03: | ( *p*→*next*  ) ← ( *e*→*next*  ) | *następnym elementem za nowym będzie następny element za e* |
| K04: | ( *p*→*data*  ) ← *v* |  |
| K05; | ( *e*→*next*  ) ← *p* | *nowy element wstawiamy za e* |
| K06: | **Zakończ** |  |

**Usuwanie wybranego elementu listy**

|  |  |  |
| --- | --- | --- |
| K01: | **Jeśli** *head*  ≠ *e*, **to idź do**kroku K04 | *sprawdzamy, czy usuwany element jest pierwszym na liście* |
| K02: | Usuń pierwszy element listy | *jeśli tak, usuwamy go z listy* |
| K03: | **Zakończ** |  |
| K04: | *p*  ← *head* | *w p ustawiamy początek listy* |
| K05: | **Dopóki** ( *p*→*next*  ) ≠ *e*, **wykonuj** *p*  ← ( *p*→*next*  ) | *w p ustawiamy adres elementu poprzedzającego e* |
| K06: | ( *p*→*next*  ) ← ( *e*→*next*  ) | *odłączamy e od listy* |
| K07: | Usuń z pamięci element wskazywany przez *e* |  |
| K08: | **Zakończ** |  |

5. Wnioski

Lista jednokierunkowa to lista "struktur danych" przy czym każda zawiera wskaźnik do następnego elementu. W liście jednokierunkowej poruszamy się od początku do końca listy. Każdy element listy składa się z co najmniej dwóch pól: klucza oraz pola wskazującego na następny element listy.

Lista jest strukturą danych, która wykorzystujemy, gdy mamy do czynienia z góry nieznaną ilością danych. Oczekuje się, że dane będą tego samego typu, ale wykorzystując unię można ograniczenie to pominąć, dzięki czemu ta struktura danych jest bardzo często wykorzystywana.

Bibliografia:

<https://eduinf.waw.pl/inf/alg/001_search/0086.php>

<https://home.agh.edu.pl/~pkleczek/dokuwiki/doku.php?id=dydaktyka:aisd:2016:data-structures_stack-queue-lists>

<https://www.samouczekprogramisty.pl/struktury-danych-lista-wiazana/>

<https://achilles.tu.kielce.pl/portal/Members/84df831b59534bdc88bef09b15e73c99/archive/semestr-ii-2019-2020/pdf/pp2/lecture/pp2_lecture_6.pdf>