МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
"КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО"

АЛГОРИТМИ ТА СТРУКТУРИ ДАНИХ

МЕТОДИЧНІ ВКАЗІВКИ ДО КОМП'ЮТЕРНОГО ПРАКТИКУМУ

**ДЛЯ СТУДЕНТІВ ФІЗИКО-ТЕХНІЧНОГО ІНСТИТУТУ
НТУУ "КПІ імені ІГОРЯ СІКОРСЬКОГО"**

Київ
НТУУ «КПІ імені ІГОРЯ СІКОРСЬКОГО»
2019

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
"КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО"

АЛГОРИТМИ ТА СТРУКТУРИ ДАНИХ

МЕТОДИЧНІ ВКАЗІВКИ ДО КОМП'ЮТЕРНОГО ПРАКТИКУМУ

для студентів спеціальностей
125 Кібербезпека
113 Прикладна математика

Рекомендовано Вченою радою Фізико-технічного інституту

Київ
НТУУ «КПІ імені ІГОРЯ СІКОРСЬКОГО»
2019

Алгоритми та структури даних. Методичні вказівки до комп’ютерного практикуму. Для студентів Фізико-технічного інституту НТУУ "КПІ імені ІГОРЯ СІКОРСЬКОГО". / Уклад.: А. М. Лавренюк, Н. М. Куссуль, А. Ю. Шелестов. — НТУУ «КПІ імені ІГОРЯ СІКОРСЬКОГО», 2019. — 24 с.

Електронне навчальне видання

Затверджено Вченою радою Фізико-технічного інституту НТУУ «КПІ імені Ігоря Сікорського»,
протокол № 8/2019 від 29 серпня 2019 р.

АЛГОРИТМИ ТА СТРУКТУРИ ДАНИХ

МЕТОДИЧНІ ВКАЗІВКИ ДО КОМП’ЮТЕРНОГО ПРАКТИКУМУ

для студентів спеціальностей
125 Кібербезпека
113 Прикладна математика

Укладачі: Алла Миколаївна Лавренюк, канд. техн. наук, доц.
Наталія Миколаївна Куссуль, д. техн. наук, проф.
Андрій Юрійович Шелестов, д. техн. наук, проф.

Рецензент: А. В. Колотій, канд. техн. наук

Під редакцією укладачів

ЗМІСТ

Лабораторна робота № 1
РЕКУРСІЯ

Мета роботи: отримати навички роботи з рекурсивними алгоритмами.

1.1. Теоретичні відомості

Рекурсія відноситься до одного з фундаментальних понять у математичних та комп'ютерних науках. В мовах програмування рекурсивною програмою називають програму, яка звертається сама до себе (сама себе викликає). Рекурсивна програма не може викликати себе до нескінченності, оскільки у цьому випадку вона ніколи не завершиться. Другою важливою особливістю рекурсивної програми є умова завершення, що дозволяє програмі припинити себе викликати.

Принципові властивості рекурсивного визначення:

1. **визначення об'єкту самого через себе, але з іншими параметрами;**
2. **завершеність ланцюжка визначень на деякому значенні аргументів.**

Необхідність використання рекурсії виникає при реалізації динамічних структур даних, таких як дерева, стеки, черги. Для реалізації рекурсивних алгоритмів у C++ передбачена можливість створення рекурсивних функцій. Рекурсивна функція являє собою функцію, у тілі якої здійснюється виклик цієї ж функції.

1.2. Приклади

1. Використання рекурсивної функції для обчислення факторіала.

Нехай потрібно скласти програму для обчислення факторіала будь-якого додатного числа.

```
#include <iostream>
using namespace std;

int fact(int n);
int main()
{
int m;
cout << "\nBВедіть ціле число:";
cin >> m;
cout << "\n Факторіал числа " << m << " дорівнює " <<
fact (m) ;
return 0; }
```

```
int fact(int n)
{
int a;
if (n<0) return 0;
if (n==0) return 1;
a =n * fact(n-1);
return a;
}
```

Для від'ємного аргументу факторіала не існує, тому функція в цьому випадку повертає нульове значення. Так як факторіал 0 дорівнює 1 за означенням, то в тілі функції передбачений і цей варіант. У випадку коли аргумент функції **fact()** відмінний від 0 та 1, то викликаємо функцію **fact()** із зменшеним на одиницю значенням параметра і результат множиться на значення поточного параметра. Таким чином, у результаті вбудованих викликів функцій повертається наступний результат:

```
n * (n-l) * (n-2) * . . . * 2 * 1 * 1
```

Остання одиниця при формуванні результату належить виклику **fact(0)**.

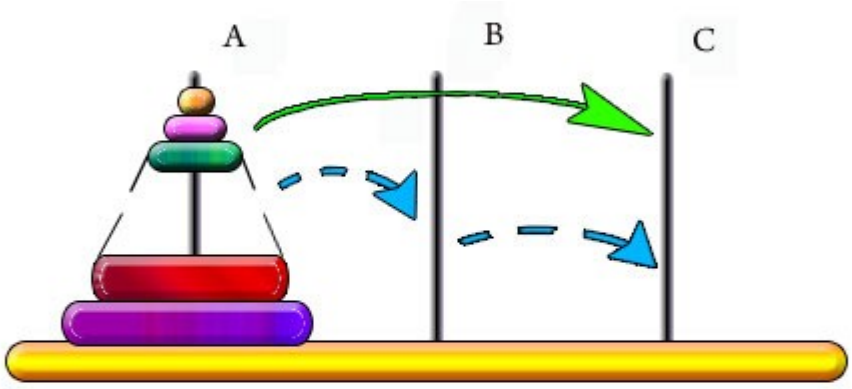
Розглянемо і іншу реалізацію функції **fact()** для невід'ємного числа n.

```
int fact (int n)
{
return n ? n*fact(n-1):1;
}
```

2. Задача «Ханойські вежі»

Ні один розгляд поняття рекурсії не був би повним без розгляду старовинної задачі про Ханойські вежі.

Принц **Шак'я-Муні** (623 - 544 р.р. до Р.Хр.), якого ще називали **Буддою**, що означає "просвітлений", під час однієї зі своїх подорожей заснував у Ханої (В'єтнам) монастир. У головному храмі монастиря стоїть три стержня (дерев'яні вісі). На одну з них Будда надягнув **64** дерев'яні диски, усі різного діаметру, причому найширший поклав униз, а решту впорядкував за зменшенням розміру:



Слід переставити піраміду з вісі **A** на вісь **C** у тому ж порядку, користуючись віссю **B**, як допоміжною, та додержуючись таких правил:

a. за один хід переставляти лише один диск з довільної осі на довільну (а не кілька);
b. забороняється класти більший диск на менший, тобто впорядкованість на кожній осі має зберігатися.

Ченці монастиря перекладають, не зупиняючись ні на мить, щосекунди по одному диску і досі. Коли піраміду буде складено на осі C, наступить кінець світу.

Рекурсивний підхід до програмування можна застосувати, якщо розуміти вежу з **n** дисків, як вежу з **(n-1)** диску, що стоїть ще на одному. Як переставити вежу з двох дисків з **A** на **C**?

1. перекласти диск з **A** на **B**;
2. перекласти диск з **A** на **C**;
3. перекласти диск з **B** на **C**;

Аналогічно програмується повний алгоритм.

Щоб переставити вежу з **n** дисків (позначимо її **P(n)**) з **A** на **C**, слід:

1. **P(n-1)** переставити з **A** на **B**;
2. **P(1)** перекласти з **A** на **C**;
3. **P(n-1)** переставити з **B** на **C**;

1.3. Порядок виконання роботи

1.3.1. Проаналізувати умову задачі.
1.3.2. Розробити алгоритм та створити програму розв’язання задачі згідно з номером варіанту.
1.3.3. Результати роботи оформити протоколом.

1.4. Варіанти завдань

З використанням рекурсії розв’язати наступні задачі.

1. Піднести до додатного цілого степеня дійсне ненульове число.
2. Знайти НСД двох цілих чисел за алгоритмом Евкліда.
3. Числа Фібоначчі f_n обчислюються за формулами $f_0=f_1=1$; $f_n=f_{n-1}+f_{n-2}$ при n=2,3,... Реалізувати функцію, яка за заданим номером *n* обчислюватиме значення f_n.
4. Реалізувати алгоритм для розв’язання задачі «Ханойські вежі». Виписати послідовність ходів для перекладання n дисків вежі (n = 2; 3; 4; 5 дисків).

1. – «задовільно»
2. або 3. – «добре»
2. і 4. або 3. і 4. – «відмінно»

Лабораторна робота № 2
МЕТОДИ СОРТУВАННЯ МАСИВІВ

Мета роботи: познайомитися з роботою поширених методів сортування, з критеріями та методикою їх порівняння.

2.1 Теоретичні відомості

ПРОСТІ МЕТОДИ СОРТУВАННЯ

Розглянемо декілька методів впорядкування елементів масиву, які широко використовуються у практичному програмуванні.

Сортування вставками

Даний метод сортування називається сортування вставками, так як на і-му етапі відбувається "вставка" і-ого елемента a[i] в потрібну позицію серед елементів a[1], a[2], …, a[i-1], які вже впорядковані. Після цієї вставки перші і елементів будуть впорядковані.

Масив до впорядкування	**22**	**20**	**-1**	**-40**	**88**	**-75**	**-22**
Перший перегляд масиву	20	22	-1	-40	88	-75	-22
Другий перегляд масиву	-1	20	22	-40	88	-75	-22
Третій перегляд масиву	-40	-1	20	22	88	-75	-22
Четвертий перегляд масиву	-40	-1	20	22	88	-75	-22
П'ятий перегляд масиву	-75	-40	-1	20	22	88	-22
Шостий перегляд масиву	-75	-40	-22	-1	20	22	88

Рис 3. Сортування вставками

Серед критеріїв, за якими порівнюються методи сортування масивів, розрізняють:

- **C** (від **compare**) - кількість порівнянь ключів між собою;
- **M** (від **move**) - кількість операцій перезапису елементів з місця на місце у оперативній пам'яті або кількість обмінів;
- **T** (від **time**) - загальний час роботи процедури.

Аналіз. Кількість перевірок на **i**-му кроці дорівнює щонайбільше **i-1**, щонайменше **1**, тому в середньому - **i/2**. Тому в середньому загальна кількість перевірок

$$C_{сер} = \sum_{i=2}^{n} \frac{i}{2} = \frac{(2+3+...+n)}{2} = \frac{(2+n)(n-1)}{4} = O(n^2)$$

При цьому

$$C_{\min} = n-1; \quad C_{\max} = \frac{(n-1)n}{2}.$$

Кількість обмінів **M** дорівнює щонайбільше **i**, щонайменше **0** на **i**-му кроці, тобто **i/2** у середньому. Тому

$$M_{\min} = 0; \quad M_{\max} = \frac{(n-1)n}{2}; \quad M_{сер} = \frac{(n-1)n}{4}.$$

Метод бульбашкового сортування

Метод "бульбашкового сортування" ґрунтується на перестановці сусідніх елементів. Для впорядкування елементів масиву здійснюються повторні проходи по масиву. Переміщення елементів масиву здійснюється таким чином: масив переглядається зліва направо, порівнюються два сусідніх елементи; якщо елементи в парі розміщені в порядку зростання, вони лишаються без змін, а якщо ні - міняються місцями.

В результаті першого проходу найбільше число буде поставлене в кінець масиву. У другому проході такі операції виконуються над елементами з першого до (N-1)-ого, у третьому - від першого до (N-2)-ого і т.д. Впорядкування масиву буде закінчено, якщо при проході масиву не виконається жодної перестановки елементів масиву. Факт перестановки фіксується за допомогою деякої змінної, яка на початку має значення 0 і набуває значення 1 тоді, коли виконається перестановка в якій-небудь парі.

Масив до впорядкування	**22**	**20**	**-1**	**-40**	**88**	**-75**	**-22**
Перший перегляд масиву	20	-1	-40	22	-75	-22	88
Другий перегляд масиву	-1	-40	20	-75	-22	22	88
Третій перегляд масиву	-40	-1	-75	-22	20	22	88
Четвертий перегляд масиву	-40	-75	-22	-1	20	22	88
П'ятий перегляд масиву	-75	-40	-22	-1	20	22	88

Рис 1. Бульбашкове сортування

Кількість порівнянь для "булькового методу":

$$C = \frac{(n-1)n}{2};$$

обмінів:

$$M_{min} = 0; \quad M_{max} = \frac{(n-1)n}{2}; \quad M_{сер} = \frac{(n-1)n}{4}.$$

Сортування методом вибору

Даний метод сортування передбачає наступні дії: масив переглядається перший раз, знаходиться мінімальний елемент цього масиву, який міняється місцями з першим елементом. Другий раз масив переглядається, починаючи з другого елементу. Знову знаходиться мінімальний елемент, який міняється місцями з другим елементом масиву.

Даний процес виконується до тих пір, поки не буде поставлено на місце N-1 елемент.

Масив до впорядкування	**22**	**20**	**-1**	**-40**	**88**	**-75**	**-22**
Перший перегляд масиву	-75	20	-1	-40	88	22	-22
Другий перегляд масиву	-75	-40	-1	20	88	22	-22
Третій перегляд масиву	-75	-40	-22	20	88	22	-1
Четвертий перегляд масиву	-75	-40	-22	-1	88	22	20
П'ятий перегляд масиву	-75	-40	-22	-1	20	22	88
Шостий перегляд масиву	-75	-40	-22	-1	20	22	88

Рис 2. Сортування методом вибору

Оцінимо якісні характеристики програми. На **i**-му кроці робиться **(n-i)** перевірок з елементами **i+1, i+2, ..., n**.
Тому

$$C = (n-1) + (n-2) + \ldots + 2 + 1 = \frac{(n-1)n}{2}$$

$$M_{сер} = \frac{0 + (n-1)}{2} = \frac{n-1}{2}$$

МОДИФІКОВАНІ МЕТОДИ СОРТУВАННЯ

Головний недолік простих методів - обмін ведеться в основному між сусідними елементами. Тому бажано робити якомога ширші обміни.

Метод Шелла

Метод Шелла (Shell D.L., 1959) - метод сортування вставками (включеннями) з відстанями, що зменшуються.

Візьмемо для прикладу масив:

60	16	41	06	59	79	34	15

На першому етапі масив уявно ділиться на підмасиви з елементами, що, скажімо, відстоять один від одного на 4 елементи:

60				59			

2.

	16				79		
		41				34	

4.

			06				15

Кожен з них впорядковується окремо:

1.

59				60			

2.

	16				79		

3.

		34				41	

4.

			06				15

Сортування робиться у кожному підмасиві вставками (методом простого включення).

На другому етапі підмасиви утворюються елементами через один:

1.

59		34		60		41	

2.

	16		06		79		15

Одержуємо

1.

34		41		59		60	

2.

	06		15		16		79

Після цього весь масив сортується разом. За рахунок попередніх етапів він виявляється вже близьким до відсортованого, тому обмінів необхідно вже не так багато.

Аналіз та експериментальні дослідження методу показують, що цей метод дає кращі результати, якщо розподіл на підмасиви роблять кроками, що не є степенями двійки, а, навпаки, не є множниками один одного.

Рекомендованими є такі послідовності кроків:

$$h_{k-1} = 3h_k + 1, \quad h_K = 1, \quad K = [\log_3 n] - 1 \qquad або$$
$$h_{k-1} = 2h_k + 1, \quad h_K = 1, \quad K = [\log_2 n] - 1.$$

У останньому випадку кількість необхідних операцій пропорційна $\mathbf{n^{6/5} = n^{1,2}}$.

ШВИДКЕ СОРТУВАННЯ

В загальному алгоритм швидкого сортування можна описати так:

quickSort

- Вибрати опорний елемент p
- Розділити масив по цьому елементу (реорганізувати масив таким чином, щоб всі елементи, менші або рівні опорному, виявилися зліва від нього, а всі елементи, більші опорного, - справа від нього)
- Якщо підмасив зліва від p містить більше одного елемента, викликати quickSort для нього (тобто повторити рекурсивно для підмасиву зліва від p)
- Якщо підмасив справа від p містить більше одного елемента, викликати quickSort для нього (тобто повторити рекурсивно для підмасиву справа від p)

Часто в якості опорного елемента пропонується вибрати медіану (середину масиву). Однак можна підібрати приклад, при якому алгоритм з вибором медіани в якості опорного елемента буде видавати неправильну відповідь. Відомі стратегії: вибирати постійно один і той самий елемент, наприклад, перший або останній; вибрати елемент випадковим чином.

Недолік вибору в якості опорного одного із крайніх елементів масиву — при передачі параметром уже відсортованого масиву такий вибір призводить до найгіршого випадку.

Недолік вибору опорного елемента випадковим чином — залежність швидкості алгоритму від реалізації генератора псевдовипадкових чисел. Якщо генератор працює повільно і видає погані послідовності псевдовипадкових чисел, можлива затримка роботи алгоритму.

Оцінювання середньостатистичних значень **M** та **C** є нелегкою задачею з огляду на необхідність використання апарату теорії ймовірностей, але обидві величини будуть порядку

$$\sim N \log_2 N.$$

2.2. Порядок виконання роботи

1. Написати програму, що реалізує один з простих методів сортування (згідно з номером варіанту).
2. Написати програму, що реалізує метод Шелла або швидкого сортування (згідно з номером варіанту).
3. Згенерувати три масиви з випадковими елементами типу **Integer** довжиною 100, 1000 та 10000 елементів, відповідно.
4. Відсортувати одержані масиви за збільшенням елементів, визначивши при цьому такі параметри:
   - кількість порівнянь;
   - кількість обмінів;
   - фактичний час роботи,

   Швидкодія сучасних ЕОМ може привести до того, що час роботи процедури буде дорівнювати нулеві з точністю, яку забезпечує системний таймер. Тоді варто запустити її багато разів у циклі для різних масивів, а потім усереднити результат.

5. Оформити звіт, навівши одержані експериментальні дані та теоретичні оцінки у вигляді таблиць.

Звіт має містити:

a. текст програми, яку Ви написали;
b. надрукований масив на 100 елементів до сортування та після нього;
c. теоретичні оцінки для кількостей операцій для методів Вашого варіанту;
d. результати експериментального дослідження та - обов`язково - їхній аналіз.

Останні два пункти можна викласти у таблиці:

Результати порівняння методів сортування

	Назва методу					**Назва методу**				
N	**К-ть копіювань (М)**		**К-ть порівнянь (С)**		**Час (Т)**	**К-ть копіювань (М)**		**К-ть порівнянь (С)**		**Час (Т)**
	Теорет.	**Експерим.**	**Теорет.**	**Експерим.**		**Теорет.**	**Експерим.**	**Теорет.**	**Експерим.**	
100										
1000										
10000										

Для методу Шелла відома лише порядкова оцінка: M = O($N^{1,2}$); C = O($N^{1,2}$). Це означає, що експериментальні величини можуть відрізнятися від теоретичних у багато разів, але відношення Мексп / Мтеор та Сексп / Стеор мають бути приблизно постійними незалежно від значення N. Порахуйте ці відношення, занесіть до таблиці та проаналізуйте.

2.3. Варіанти завдань

№ варіанту	Метод	
	"простий"	"складний"
1	"Вставками"	метод Шелла
2	"Вибору"	метод Шелла
3	"Бульбашки"	метод Шелла
4	"Вставками"	метод швидкого сортування
5	"Вибору"	метод швидкого сортування
6	"Бульбашки"	метод швидкого сортування

Лабораторна робота № 3
Структури даних: масиви, зв’язні списки

Мета роботи: обробити текст без використання стандартних бібліотек для роботи з даними символьного типу; порівняти ефективність обробки тексту з використанням символьних масивів і динамічних списків, відпрацювати навики роботи зі списками.

3.1 Теоретичні відомості

Персональний комп’ютер використовується не лише для виконання обчислень, а й для аналізу та обробки тексту. Текст при цьому подається як сукупність символьних даних, які об’єднуються в структури типу символьного масиву, динамічного списку чи рядка (конструкція String в Паскалі або C++). Ці структури розрізняються формою свого внутрішнього представлення в пам’яті комп’ютера.

При роботі з текстом програміст повинен, насамперед, вміти правильно обрати структуру даних для зберігання та обробки символьної інформації.

Отже, текст фізично можна представити як масивом символів, так і списком символів. При цьому символьний масив може бути як статичний, так і динамічний.

При використанні статичного масиву весь об’єм необхідної пам’яті виділяється на етапі компіляції. В процесі заповнення масиву можна забезпечити запам’ятовування кількості елементів в ньому, організовуючи таким чином роботу тільки із заповненою областю пам’яті. Частина пам’яті,

що залишатиметься незаповненою, в цьому випадку використовується не продуктивно.

Для оптимізації витрат пам’яті можна використовувати динамічний масив. Його перевагою є те, що пам'ять під динамічний масив виділяється під час виконання програми за відповідним запитом. Але слід враховувати, що при необхідності додати в масив ще один елемент, доведеться кожного разу відсилати запит в пам'ять для створення динамічного масиву нової довжини, що потребує часу на пошук неперервної ділянки пам’яті (концепція масиву потребує, щоб сусідні елементи розташувались в послідовних байтах пам’яті), а також виділення цієї пам’яті, перепису в неї даних зі «старого» масиву і збереження нової довжини. Таким чином, уникнувши втрат пам’яті, ми прийдемо до додаткових витрат часу. Відповідно, зміна розміру динамічного масиву на одиницю – дуже ресурсоємна та недоцільна дія. Пам’ять під динамічний масив необхідно запитувати «розумними» частинами, щоб мінімізувати можливі втрати ресурсів. Зазвичай новий масив подвоює використовуваний масив пам’яті.

Список – це динамічна структура даних, кожен елемент якої, містить дані і адресу наступного елемента (вказівник на наступний елемент). Ця адреса у останнього елемента – нульова. Пам’ять під кожен елемент списку виділяється динамічно по мірі необхідності. При цьому в статичній пам’яті зберігається адреса першого елемента або нульовий вказівник, якщо список порожній.

Для доступу до елемента списку необхідно переглянути список з «голови». Такий доступ називається послідовним. В елементі списку можна зберігати адресу не лише наступного, а й попереднього елемента. В цьому випадку список називається двозв’язним. Адреса попереднього у першого елемента і наступного у останнього елемента – нульова. Двозв’язні списки можна переглядати у двох напрямках, тобто, доступ до елементів двозв’язного списку можливий з будь-якого кінця.

Головними операціями над списками є вставка елемента в список, видалення елемента зі списку, пошук елемента в списку. При цьому слід забезпечити виведення відповідних повідомлень при спробі видалення елемента з порожнього списку і не забувати звільняти пам’ять при видаленні елемента із списку.

Порівняльний аналіз реалізації структур даних у формі масивів і списків показує наступне:

- якщо заздалегідь неможливо визначити кількість елементів в структурі, то її краще реалізовувати списком;
- якщо необхідно реалізувати операції включення або виключення елемента за номером (за позицією), то краще використовувати масиви, тому що в масивах можливий прямий доступ до елемента, а при використанні вказівників позиція взагалі не відслідковується;

- операції включення або виключення елемента із масиву виконуються важче і довше, ніж в списку;
- використання масивів передбачає неекономне виділення пам’яті (відразу під весь масив), однак у списку потрібна додаткова пам’ять в кожному елементі для зберігання вказівника на наступний (попередній) елемент.

Таким чином, кожна реалізація має свої переваги і недоліки, тому при розв’язанні конкретної задачі програміст повинен вміти оцінювати переваги та втрати, щоб свідомо надати перевагу одній із реалізацій.

3.2. Порядок виконання роботи

Слова тексту із малих латинських літер записані не менше, ніж через один пробіл; текст закінчується крапкою. БЕЗ ВИКОРИСТАННЯ конструкції String і стандартних процедур роботи з рядками написати програму введення такого тексту з клавіатури та його обробки, використовуючи: а) масив символів та б) список символів. Виконати завдання відповідно до свого варіанту.

Загальний алгоритм розв’язання поставленої задачі, як правило, не залежить від конкретної реалізації структури даних. Але всі дії над текстом (пошук слів, виділення будь-якого із слів, перестановка літер в слові тощо) повинні бути описані функціями, «налаштованими» на конкретну реалізацію з допомогою формальних параметрів.

Інтерфейс програми має бути зрозумілим непідготовленому користувачеві. При розробці інтерфейсу програми слід передбачити:

- задання формату і діапазону даних, що вводяться;
- блокування введення невірних за типом і форматом даних;
- задання операції, яка виконується програмою;
- наявність пояснень при виведенні результату.

Потрібно також вивести на екран інформацію про час виконання програми при використанні масиву і списку та про об’єм пам’яті, необхідний у кожному з цих випадків.

При тестуванні програми необхідно:

- перевірити правильність введення та виведення даних (зокрема, відстежити спроби введення даних, неправильних за типом і форматом);
- забезпечити виведення повідомлень за відсутності вхідних даних («пусте введення»);
- перевірити правильність виконання операцій, зокрема, при повністю заповненому масиві;
- відстежити вихід за межі масиву;
- забезпечити виведення відповідних повідомлень при спробі видалення елемента з пустого списку або масиву;
- відстежити переповнення масиву.

При представленні тексту у вигляді списку необхідно:

- перевірити можливість вставки елемента в початок, в кінець і в середину списку;
- проконтролювати правильність видалення елемента з кінця, середини, початку списку;
- відстежити видалення єдиного елемента і видалення елемента з порожнього списку;
- перевірити, як звільняється пам’ять при видаленні елемента зі списку.

3.3. Варіанти завдань

1. Надрукувати всі слова, у яких непарна кількість літер. Перед друком видалити першу та останню літеру кожного слова.

2. Надрукувати всі слова, які відрізняються від першого слова і співпадають з початковим відрізком алфавіту (a, ab, abc тощо). Видалити першу літеру в цих словах. До кожного слова дописати крапку.

3. Надрукувати всі слова, які починаються на літеру, відмінну від літери, з якої починається перше слово тексту. Перед друком видалити зі слів всі літери ’а’ та ’о’.

4. Надрукувати всі слова, які відрізняються від першого слова. Перед друком подвоїти першу літеру, якщо в слові парна кількість літер, та видалити останню літеру, якщо в слові непарна кількість літер. Якщо слів менше, ніж два, видати повідомлення.

Лабораторна робота № 4
Структури даних: стеки, черги

Мета роботи: отримати навики роботи зі стеком та чергою, реалізованими у вигляді одновимірного масиву та зв’язного лінійного списку.

4.1 Теоретичні відомості

Стеки та черги — це динамічні множини, елементи з яких видаляються за допомогою певним чином визначених операцій Delete.

Першим зі **стеку** (stack) видаляється елемент, який був туди доданий останнім: в стеку реалізується стратегія **"останнім зайшов — першим вийшов"** (last-in, first-out — LIFO).

Отже, всі операції (наприклад, видалення елементу) в стеку можна проводити тільки з одним елементом, що знаходиться на вершині стеку і був введений в стек останнім.

Стек можна представити, як сукупність однотипних елементів, в якій ми маємо доступ тільки до верхнього елемента.

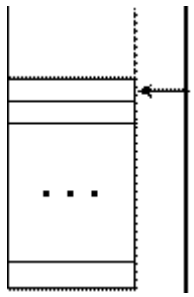


Рисунок. Стек

У **черзі** (queue) завжди видаляється елемент, який міститься у множині довше від інших: в черзі реалізується стратегія **"першим зайшов – першим вийшов"** (first-in, first-out — FIFO).

У черги є голова (англ. head) та хвіст (англ. tail). Елемент, що додається до черги, опиняється в її хвості. Елемент, що видаляється з черги, знаходиться в її голові.

Чергу можна представити, як сукупність однотипних елементів, в якій маємо доступ до кінця черги при додаванні елементів та до початку черги при взятті елементів.

Рисунок. Черга

Операції зі стеком та чергою (додавання елемента, видалення елемента) представлені у лекції на псевдокоді.

4.2. Порядок виконання роботи

4.2.1. Реалізувати стек двома способами: за допомогою масиву та списку (на оцінку **«добре»**).

4.2.2.а) Реалізувати чергу двома способами: за допомогою масиву (кільцеву чергу) та списку (на оцінку **«відмінно»**).

б) Перетворити вираз у постфіксну форму і реалізувати його за допомогою стеку (на оцінку **«відмінно»**).

4.2.3. Додаткове завдання: З використанням бібліотеки стандартних шаблонів **STL** (Standard Template Library) реалізувати стек або чергу (додаткові бали).

4.3. Варіанти завдань

І частина

1. Розробити програму роботи зі стеком, яка реалізує операції додавання, видалення елементів зі стеку і відображення поточного стану стеку. Реалізувати стек: а) масивом; б) списком (на оцінку **«добре»**).

2. Розробити програму роботи з чергою, яка реалізує операції додавання, видалення елементів з черги і відображення поточного стану черги.

Реалізувати чергу: а) масивом (використовувати кільцеву чергу); б) списком (на оцінку **«відмінно»**).

3. При реалізації стеку масивом забезпечити розміщення двох стеків в одному масиві. Один стек розміщується на початку масиву і зростає до кінця, а другий розміщується в кінці масиву і росте до початку. Заповнити і звільнити стеки довільним чином, вводячи значення з клавіатури. Елементами стеку є дійсні числа. У вигляді списку реалізувати один стек (на оцінку **«відмінно»**).

ІІ частина (на оцінку **«відмінно»**)

Існують три способи запису складних виразів. Для двомісних операцій (тут - ⊕) існують три способи запису:

префіксний (функціональний): **⊕ x y;**

інфіксний (шкільний): **x ⊕ y;**

постфіксний (польський): **x y ⊕;**

Всі три означають, що треба взяти операнди **x** та **y** (які можуть бути числами, тобто константами, змінними або іншими виразами) та виконати з ними операцію ⊕.

Приклад: вираз

(((A - B) * C) + (D / (E + F)))

Звичайний (шкільний) спосіб - це інфіксний.

Для даного прикладу її префіксна та постфіксна форми мають вигляд:

префіксна:

+ * - ABC / D + EF

постфіксна:

AB - C * DEF + / +

Обидві форми однозначно визначають порядок обчислення та не вимагають дужок. Префіксна структура дуже зручно реалізується за допомогою рекурсивних процедур. Постфіксна структура краще за все реалізується за допомогою стеку.

Розглянемо реалізацію постфіксної форми за допомогою стеку. Вираз 1 + 2 у постфіксній формі записується як 1 2 +, а вираз (1 + 2)*4 — як 1 2 + 4 *. В постфіксній формі порядок виконання операцій визначається виключно їх розташуванням в рядку; відпадає необхідність у використанні дужок і у такому понятті, як пріоритет операцій.

Для обчислення представлених у постфіксній формі виразів достатньо скористатися наступним простим алгоритмом: операнди, які зустрічаються у вхідному рядку, поміщаються в стек, а операції, які зустрічаються, виконуються над двома верхніми значеннями зі стеку з розміщенням результату в стек. Таким чином, при коректному записі виразу в результаті введення рядка на вершині стеку міститиметься результат обчислень.

Вх. рядок	1	1 2	1 2 +	1 2 + 4	1 2 + 4 *
Стек	1	2 1	3	4 3	12

Перетворіть наведені нижче вирази у постфіксну форму і реалізуйте за допомогою стеку:

1. (A-B-C)/D-E*F
2. (A+B)*C-(D+E)/F
3. A/(B-C)+D*(E-F)
4. (A*B+C)/D-F/E

Додаткове завдання (додаткові бали)

З використанням бібліотеки стандартних шаблонів **STL** (Standard Template Library) реалізувати:

1. Стек.
2. Чергу.

Лабораторна робота № 5
Дерева

Мета роботи: отримати навики застосування двійкових дерев, реалізувати основні операції над деревами: обхід дерев, включення, виключення та пошук вузлів.

5.1 Теоретичні відомості

Дерево - це нелінійна структура даних, що використовується для представлення ієрархічних зв'язків, що мають відношення «один до багатьох».

Дерево з базовим типом T визначається рекурсивно або як порожня структура (порожнє дерево), або як вузол типу T з кінцевою кількістю деревовидних структур цього ж типу, які називаються піддеревами.

Дерева використовуються при побудові організаційних діаграм, аналізі електричних ланцюгів, для представлення синтаксичних структур у компіляторах програм, для представлення структур математичних формул, організації інформації в СУБД тощо.

Самий верхній вузол дерева називається коренем. Верхній вузол для нижнього вузла називається предком, а нижній вузол для верхнього - нащадком. Вершини (вузли), що не мають нащадків, називаються листками дерева. Вершини, що мають нащадків, називаються внутрішніми. Дві вершини дерева з'єднуються гілкою. Кількість гілок від кореня до вершини є довжиною шляху до цієї вершини.

Довжина шляху від кореня до будь-якої вершини називається глибиною цієї вершини. Максимальна глибина вершин дерева називається висотою дерева.

Кількість безпосередніх нащадків у вершини (вузла) дерева називається степенем вершини (вузла). Максимальний степінь всіх вершин є степенем дерева.

Кожному вузлу дерева можна співставити ім'я вузла і значення вузла, тобто дані, які зберігаються в цьому вузлі. Причому, якщо значенням є різнорідні дані (записи або об'єднання), то значенням вузла можна вважати значення одного з полів цих даних, яке називається ключем.

Наприклад, у дерева на рис. 1 номер вузла може бути як його ім'ям, так і його значенням.

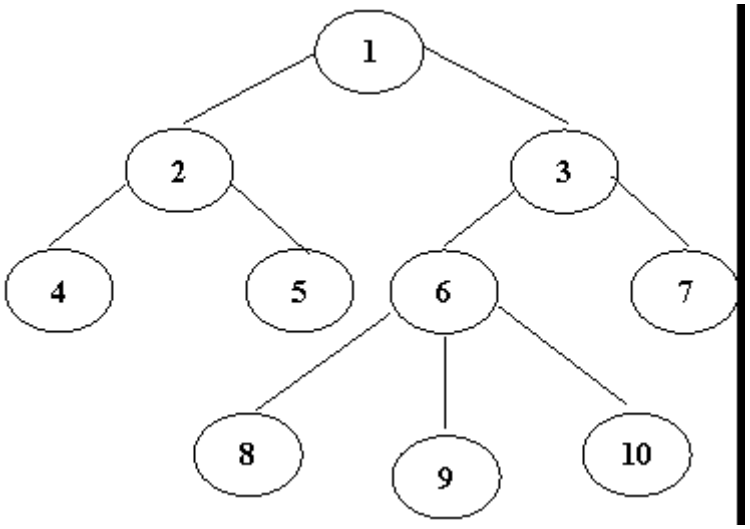


Рис. 1

У пам'яті дерева можна представити у вигляді зв'язків з предками (батьками); зв'язного списку нащадків (дітей) або структури даних.

Подання зазначеного дерева (див. рис. 1) у вигляді зв'язків з предками:

№ вершина	1	2	3	4	5	6	7	8	9	10
Батько	0	1	1	2	2	3	3	6	6	6

Приклад подання цього ж дерева у вигляді зв'язного списку нащадків наведено на рис 2:

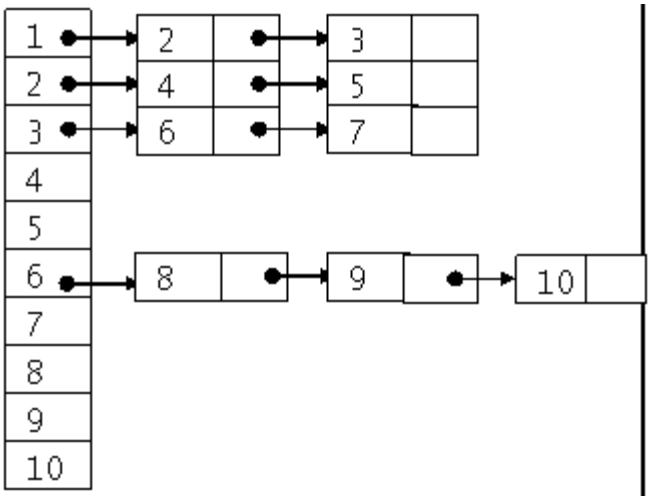


Рис. 2.

Якщо у кожної вершини дерева є не більше двох нащадків (ліві і праві піддерева), то таке дерево називається двійковим або бінарним.

Двійкові дерева широко використовуються у програмуванні.

Основні операції з деревами: обхід дерева, пошук по дереву, включення в дерево, виключення з дерева.

Обхід (відвідування) вершин дерева можна здійснити наступним чином (рис. 3):

- Зліва направо: A, R, B (інфіксний обхід, симетричний обхід) .
- Зверху вниз: R, A, B (префіксній обхід, обхід в прямому порядку).
- Знизу вверх: A, B, R (постфіксний обхід, обхід в зворотному порядку).

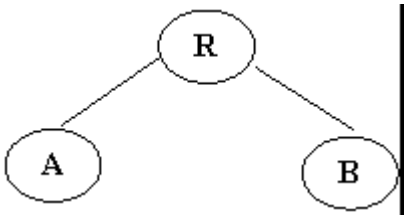


Рис. 3

Для реалізації алгоритмів пошуку використовуються дерева двійкового пошуку. Дерево двійкового пошуку - це таке дерево, в якому всі ліві нащадки молодші за предка, а всі праві - старші. Ця властивість називається характеристичною властивістю дерева двійкового пошуку і виконується для будь-якого вузла, включаючи корінь. З урахуванням цієї властивості пошук вузла в двійковому дереві пошуку можна здійснити, рухаючись від кореня в ліве або праве піддерево в залежності від значення ключа піддерева.

Якщо при побудові дерева по черзі розташовувати вузли зліва та справа, то вийде дерево, у якого кількість вершин у лівому та правому піддеревах відрізняється не більше ніж на одиницю. Таке дерево називається ідеально збалансованим.

N елементів можна організувати в бінарне дерево з висотою не більше $\log_2(N)$, тому для пошуку серед N елементів може знадобитись не більше $\log_2(N)$ порівнянь, якщо дерево ідеально збалансоване. Звідси випливає, що дерево - це одна з найкращих структур для організації пошуку.

Операція включення елемента у дерево розбивається на три етапи: включення вузла в порожнє дерево, пошук кореня для додавання нового вузла, включення вузла в ліве або праве піддерево.

Для видалення вузла з зазначеним ключем спочатку відбувається його пошук. У випадку, якщо вузол знайдений, то він видаляється. При цьому розглядаються наступні три випадки. Якщо видаляється вузол, у якого немає нащадків, то просто вказівнику на нього присвоюється значення NULL. Якщо видаляється вузол, який має одного нащадка, в цьому випадку переадресується вказівник на цього нащадка. Якщо видаляється вузол, який має двох нащадків, то на його місце ставиться самий лівий нащадок з правого піддерева.

5.2. Порядок виконання роботи

Побудувати дерево відповідно до варіанту. Вивести його на екран у вигляді дерева. Реалізувати основні операції роботи з деревом: обхід дерева, включення, виключення та пошук вузлів. Оформити їх у вигляді функцій.

При розробці інтерфейсу програми слід передбачити:

- вказати тип, формат і діапазон даних, що вводяться;
- вказати дії, що здійснює програма;
- наявність пояснень при виведенні результату;
- відображення дерева візуалізувати.

Під час тестування програми необхідно:

- перевірити правильність введення і виведення даних (тобто їх відповідність необхідному типу і формату). Забезпечити адекватну реакцію програми на невірне введення даних;
- забезпечити виведення повідомлень за відсутності вхідних даних («порожнє введення»);
- перевірити правильність виконання операцій;
- забезпечити можливість додавання вузла в порожнє дерево;
- передбачити відображення повідомлення при спробі видалити вузол з порожнього дерева;
- перевірити різні випадки включення і виключення вузла в існуючому дереві;

- перевірити пошук існуючого вузла та пошук неіснуючого вузла в дереві.

5.3. Варіанти завдань

На оцінку **«добре»:**

1. Побудувати двійкове дерево пошуку з цілих чисел, що вводиться. Вивести його на екран у вигляді дерева. Знайти вершину, яка містить задане число. Визначити максимальний елемент в цьому дереві.

На оцінку **«відмінно»:**

1. Побудувати двійкове дерево пошуку з букв рядка, що вводиться. Вивести його на екран у вигляді дерева. Знайти букви, що зустрічаються більше одного разу. Видалити з дерева ці літери. Вивести елементи дерева, що залишилися, при його постфіксному обході.
2. Побудувати двійкове дерево пошуку, в вершинах якого знаходяться слова з текстового файлу. Вивести його на екран у вигляді дерева. Визначити кількість вершин дерева, що містять слова, які починаються на зазначену букву. Видалити з дерева ці вершини.
3. Побудувати і вивести на екран бінарне дерево наступного виразу:

9 + 8 * (7 + (6 * (5 + 4) – (3 – 2)) +1) . Написати функції постфіксного, інфіксного та префіксного обходу дерева і вивести відповідні вирази на екран.

4. Побудувати словник зі слів текстового файлу у вигляді дерева двійкового пошуку. Вивести його на екран у вигляді дерева. Здійснити пошук вказаного слова у дереві і у файлі. Якщо слова немає, додати його при необхідності в дерево і у відповідний файл. Видалити вказане слово з дерева та файлу. Порівняти час пошуку в дереві та у файлі.

СПИСОК ЛІТЕРАТУРИ

1. Кормен Т. Х., Лейзерсон Ч. И., Ривест Р.Л., Штайн К. Алгоритмы: построение и анализ. - М.: Вильямс, 2013. - 1328 с.
2. Седжвик Р. Фундаментальные алгоритмы на C++. Части 1-4. Анализ/Структуры данных/Сортировка/Поиск. - СПб.: ООО "ДиаСофтЮП", 2002. - 688 с.
3. Седжвик Р. Фундаментальные алгоритмы на C++. Ч 5. Алгоритмы на графах. - СПб.: ООО "ДиаСофтЮП", 2002. - 496 с.
4. Кнут Д.Э. Искусство программирования, том 1. Основные алгоритмы. - М.: "Вильямс", 2000. - 720 с.
5. Кнут Д.Э. Искусство программирования, том 2. Получисленные методы. - М.: "Вильямс", 2000. - 832 с.

6. Кнут Д.Э. Искусство программирования, том 3. Сортировка и поиск. - М.: "Вильямс", 2000. - 832 с.
7. Вирт Н. Алгоритмы + структуры данных = программы. - М.: Мир, 1985. - 406с.
8. Вирт Н. Алгоритмы и структуры данных. - М.: Мир, 1989. - 360с.
9. Мальцев А.И. Алгоритмы и рекурсивные функции. - М.: Наука, 1986. - 368с.
10. Селін О.М. Алгоритми та структури даних. Методичні вказівки до лабораторних робіт, 1998
11. Селін О.М. Конспект лекцій з дисципліни «Алгоритми та структури даних», 1998.