

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
ХЕРСОНСЬКИЙ ПОЛІТЕХНІЧНИЙ КОЛЕДЖ  
ОДЕСЬКОГО НАЦІОНАЛЬНОГО ПОЛІТЕХНІЧНОГО УНІВЕРСИТЕТУ  
ВІДДІЛЕННЯ КОМП'ЮТЕРНОЇ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

Курс лекцій  
з дисципліни

## **«Алгоритми та структури даних»**

# Зміст

<b>1</b>	<b>Систення інформація за допомогою бінарних дерев</b>	<b>9</b>
1.1	Вступ . . . . .	9
1.2	Ключові терміни . . . . .	9
1.3	Розширені теоретичні відомості . . . . .	10
1.4	Приклади обчислень . . . . .	12
<b>2</b>	<b>Представлення виразів за допомогою дерев</b>	<b>13</b>
2.1	Вступ . . . . .	13
2.2	Ключові терміни . . . . .	13
2.3	Розширені теоретичні відомості . . . . .	13
2.4	Приклади обчислень . . . . .	13
<b>3</b>	<b>Представлення багаторозгалужених дерев</b>	<b>14</b>
3.1	Вступ . . . . .	14
3.2	Ключові терміни . . . . .	14
3.3	Розширені теоретичні відомості . . . . .	14
3.4	Приклади обчислень . . . . .	14
<b>4</b>	<b>Представлення графів</b>	<b>15</b>
4.1	Вступ . . . . .	15
4.2	Ключові терміни . . . . .	15
4.3	Розширені теоретичні відомості . . . . .	15
4.4	Приклади обчислень . . . . .	15
<b>5</b>	<b>Алгоритми на графах</b>	<b>16</b>
5.1	Вступ . . . . .	16
5.2	Ключові терміни . . . . .	16
5.3	Розширені теоретичні відомості . . . . .	16
5.4	Приклади обчислень . . . . .	16
<b>6</b>	<b>Використання багаторозгалужених дерев</b>	<b>17</b>
6.1	Вступ . . . . .	17
6.2	Ключові терміни . . . . .	17
6.3	Розширені теоретичні відомості . . . . .	17

6.4	Приклади обчислень . . . . .	17
<b>7</b>	<b>Алгоритми сортування</b>	<b>18</b>
7.1	Вступ . . . . .	18
7.2	Ключові терміни . . . . .	18
7.3	Розширені теоретичні відомості . . . . .	18
7.3.1	Оцінка алгоритмів сортування . . . . .	18
7.3.2	Класифікація алгоритмів сортувань . . . . .	19
<b>8</b>	<b>Алгоритми внутрішнього сортування</b>	<b>20</b>
8.1	Вступ . . . . .	20
8.2	Ключові терміни . . . . .	20
8.3	Розширені теоретичні відомості . . . . .	20
8.3.1	Сортування Вибором . . . . .	20
8.3.2	Сортування вставкою . . . . .	21
8.3.3	Бульбашкове сортування . . . . .	22
8.4	Приклади обчислень . . . . .	22
<b>9</b>	<b>Алгоритми зовнішнього сортування</b>	<b>23</b>
9.1	Вступ . . . . .	23
9.2	Ключові терміни . . . . .	23
9.3	Розширені теоретичні відомості . . . . .	24
9.3.1	Сортування простим злиттям . . . . .	24
9.4	Приклади обчислень . . . . .	25
<b>10</b>	<b>Характеристика алгоритмів порівняння методів сортування</b>	<b>26</b>
10.1	Вступ . . . . .	26
10.2	Ключові терміни . . . . .	26
10.3	Розширені теоретичні відомості . . . . .	26
10.4	Приклади обчислень . . . . .	26
<b>11</b>	<b>Алгоритми розподілу обчислювального процесу</b>	<b>27</b>
11.1	Вступ . . . . .	27
11.2	Ключові терміни . . . . .	27
11.3	Розширені теоретичні відомості . . . . .	27
11.4	Приклади обчислень . . . . .	27
<b>12</b>	<b>Архітектура розподілених обчислень</b>	<b>28</b>
12.1	Вступ . . . . .	28
12.2	Ключові терміни . . . . .	28
12.3	Розширені теоретичні відомості . . . . .	28
12.4	Приклади обчислень . . . . .	28

<b>13</b>	<b>Процеси і потоки в обчисленні</b>	<b>29</b>
13.1	Вступ . . . . .	29
13.2	Ключові терміни . . . . .	29
13.3	Розширені теоретичні відомості . . . . .	29
13.4	Приклади обчислень . . . . .	29
<b>14</b>	<b>Реалізація багатозадачного середовища</b>	<b>30</b>
14.1	Вступ . . . . .	30
14.2	Ключові терміни . . . . .	30
14.3	Розширені теоретичні відомості . . . . .	30
14.4	Приклади обчислень . . . . .	30
<b>15</b>	<b>Бібліотеки організації розподілених обчислень</b>	<b>31</b>
15.1	Вступ . . . . .	31
15.2	Ключові терміни . . . . .	31
15.3	Розширені теоретичні відомості . . . . .	31
15.4	Приклади обчислень . . . . .	31
<b>16</b>	<b>Програмна модель OpenMP</b>	<b>32</b>
16.1	Вступ . . . . .	32
16.2	Ключові терміни . . . . .	32
16.3	Розширені теоретичні відомості . . . . .	32
16.4	Приклади обчислень . . . . .	32
<b>17</b>	<b>Конструкції OpenMP для розподілу робіт</b>	<b>33</b>
17.1	Вступ . . . . .	33
17.2	Ключові терміни . . . . .	33
17.3	Розширені теоретичні відомості . . . . .	33
17.4	Приклади обчислень . . . . .	33
<b>18</b>	<b>Умови виконання (планування)</b>	<b>34</b>
18.1	Вступ . . . . .	34
18.2	Ключові терміни . . . . .	34
18.3	Розширені теоретичні відомості . . . . .	34
18.4	Приклади обчислень . . . . .	34
<b>19</b>	<b>Бібліотечні функції OpenMP</b>	<b>35</b>
19.1	Вступ . . . . .	35
19.2	Ключові терміни . . . . .	35
19.3	Розширені теоретичні відомості . . . . .	35
19.4	Приклади обчислень . . . . .	35
<b>20</b>	<b>Бібліотека MPI</b>	<b>36</b>

20.1 Вступ . . . . .	36
20.2 Ключові терміни . . . . .	36
20.3 Розширені теоретичні відомості . . . . .	36
20.4 Приклади обчислень . . . . .	36
<b>21 Режими обліку повідомлень в MPI</b>	<b>37</b>
21.1 Вступ . . . . .	37
21.2 Ключові терміни . . . . .	37
21.3 Розширені теоретичні відомості . . . . .	37
21.4 Приклади обчислень . . . . .	37
<b>A Правила оформлення звіту</b>	<b>38</b>
A.1 Титульна сторінка лабораторної роботи . . . . .	38
A.2 Приклади блок-схем . . . . .	39
<b>Б Лістинги</b>	<b>40</b>
Б.1 Програмна реалізація алгоритму Хаффмана за допомогою кодового дерева . . .	40
Б.2 Опис функції сортування простим злиттям . . . . .	43

# Перелік ілюстрацій

1.1	Дерево . . . . .	9
1.2	Дерево . . . . .	11
9.1	Демонстрація сортування двоколіїному двофазним простим злиттям . . . . .	24

## Перелік таблиць

# Listings

8.1	Сортування вибором . . . . .	20
8.2	Сортування вставкою . . . . .	21
8.3	Бульбашкове сортування . . . . .	22
Б.1	Програмна реалізація алгоритму Хаффмана . . . . .	40
Б.2	Програмна реалізація функції сортування простим злиттям . . . . .	43



# Лекція № 1

## Систення інформація за допомогою бінарних дерев

### 1.1 Вступ

Дерево — це структура даних, що представляє собою сукупність елементів і відносин, що утворюють ієрархічну структуру цих елементів (рис. 1.1). Кожен елемент дерева називається вершиною (вузлом) дерева. Вершини дерева з'єднані спрямованими дугами, які називають гілками дерева. Початковий вузол дерева називають коренем дерева, йому відповідає нульовий рівень. Листям дерева називають вершини, в які входить одна гілка і не виходить жодної гілки.

Кожне дерево має такі властивості:

- ◇ існує вузол, в який не входить ні однієї дуги (корінь);
- ◇ в кожную вершину, крім кореня, входить одна дуга.

Дерева особливо часто використовують на практиці при зображенні різних ієрархій.

Алгоритм Хаффмана - адаптивний жадний алгоритм оптимального префіксного кодування алфавіту з мінімальною надмірністю. Був розроблений в 1952 році аспірантом Массачусетського технологічного інституту Девідом Хаффманом при написанні їм курсової роботи. В даний час використовується в багатьох програмах стиснення даних.

### 1.2 Ключові терміни

**Стиснення даних** - це процес, що забезпечує зменшення обсягу даних шляхом скорочення їх надмірності.

**Стиснення без втрат (повністю оборотне)** - це метод стиснення даних, при якому раніше закодована порція даних відновлюється після їх розпакування повністю без внесення змін.

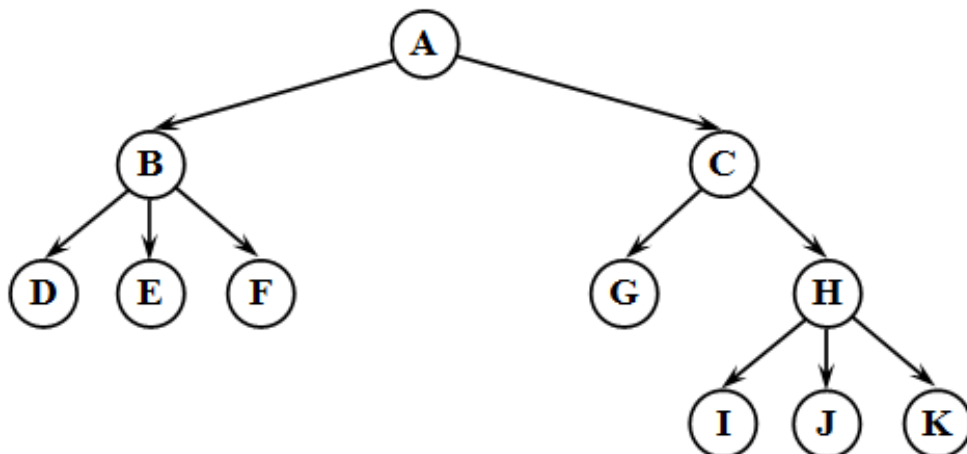


Рис. 1.1 – Дерево

**Стиснення з втратами** - це метод стиснення даних, при якому для забезпечення максимального ступеня стиску вихідного масиву даних частину містяться в ньому даних відкидається.

**Алгоритм стиснення даних (алгоритм архівації)** - це алгоритм, який усуває надмірність запису даних.

**Алфавіт коду** - це множина всіх символів вхідного потоку.

**Кодовий символ** - це найменша одиниця даних, що підлягає стисненню.

**Кодове слово** - це послідовність кодових символів з алфавіту коду.

**Токен** - це одиниця даних, записувана в стислий потік деяким алгоритмом стиснення.

**Фраза** - це фрагмент даних, що поміщається в словник для подальшого використання в стисненні.

**Кодування** - це процес стиснення даних.

**Декодування** - це зворотний кодуванню процес, при якому здійснюється відновлення даних.

**Ставлення стиснення** - це величина для позначення ефективності методу стиснення, що дорівнює відношенню розміру вихідного потоку до розміру вхідного потоку.

**Коефіцієнт стиснення** - це величина, зворотна відношенню стиснення.

**Середня довжина кодового слова** - це величина, яка обчислюється як зважена ймовірностями сума довжин всіх кодових слів.

**Статистичні методи** - це методи стиснення, що привласнюють коди змінної довжини символам вхідного потоку, причому більш короткі коди присвоюються символам або групам символів, що має більшу ймовірність появи у вхідному потоці.

**Словникове стиснення** - це методи стиснення, що зберігають фрагменти даних в деякій структурі даних, званої словником.

**Хаффманове кодування (стиснення)** - це метод стиснення, привласнює символам алфавіту коди змінної довжини ґрунтуючись на ймовірність появи цих символів.

**Префіксний код** - це код, в якому ніяке кодове слово не є префіксом будь-якого іншого кодового слова.

**Оптимальний префіксний код** - це префіксний код, що має мінімальну середню довжину.

**Кодова дерево (дерево кодування Хаффмана, Н-дерево)** - це бінарне дерево, у якого: листя позначені символами, для яких розробляється кодування; вузли (у тому числі корінь) позначені сумою ймовірностей появи всіх символів, відповідних листю піддерева, коренем якого є відповідний вузол.

## 1.3 Розширені теоретичні відомості

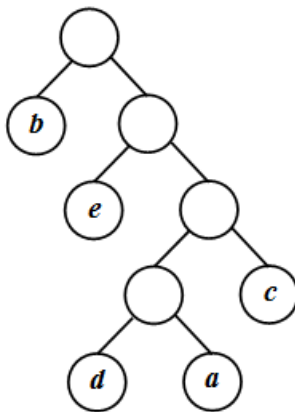
**Кодова дерево (дерево кодування Хаффмана, Н-дерево)** - це бінарне дерево, у якого:

- ◇ листя позначені символами, для яких розробляється кодування;

**Вероятности появления  
символов**

Символ	Вероятность
<i>a</i>	0,1
<i>b</i>	0,4
<i>c</i>	0,2
<i>d</i>	0,05
<i>e</i>	0,25

**Кодовое дерево**



**Оптимальные  
префиксные коды**

Символ	Код
<i>a</i>	1101
<i>b</i>	0
<i>c</i>	111
<i>d</i>	1100
<i>e</i>	10

Рис. 1.2 – Дерево

◇ вузли (у тому числі корінь) позначені сумою ймовірностей появи всіх символів, відповідних листю подерева, коренем якого є відповідний вузол.,

Метод Хаффмана на вході отримує таблицю частот зустрічальності символів у вихідному тексті. Далі на підставі цієї таблиці будується дерево кодування Хаффмана.

Алгоритм побудови дерева Хаффмана.

1. Символи вхідного алфавіту утворюють список вільних вузлів. Кожен лист має вагу, що може бути дорівнює або ймовірності, або кількістю входжень символу в стискається текст.
2. Вибираються два вільні вузла дерева з найменшими вагами.
3. Створюється їх батько з вагою, рівним їх сумарному вазі.
4. Батько додається в список вільних вузлів, а двоє його дітей видаляються з цього списку.
5. Однією дузі, що виходить з батька, ставиться у відповідність біт 1, інший - біт 0.
6. Повторюємо кроки, починаючи з другого, до тих пір, поки в списку вільних вузлів не залишиться тільки один вільний вузол. Він і буде вважатися коренем дерева.

Існує два підходи до побудови кодового дерева: від кореня до листя і від листя до кореня.

Приклад побудови кодового дерева. Нехай задана вихідна послідовність символів:

**aabbbbbbbccccdeeeee.**

Її вихідний обсяг дорівнює 20 байт (160 біт). Згідно з наведеними на рис. 1.2 даними (таблиця ймовірності появи символів, кодове дерево і таблиця оптимальних префіксних кодів) закодована вихідна послідовність символів буде виглядати наступним чином:

**110111010000000011111111111111001010101010.**

Отже, її обсяг буде дорівнює 42 біта. Коефіцієнт стиснення наближено дорівнює 3,8.

Класичний алгоритм Хаффмана має один істотний недолік. Для відновлення вмісту стислого тексту при декодуванні необхідно знати таблицю частот яку використовували при кодуванні. Отже, довжина стислого тексту збільшується на довжину таблиці частот, яка повинна

надсилатися попереду даних, що може звести нанівець всі зусилля зі стиснення даних. Крім того, необхідність наявності повної частотної статистики перед початком власне кодування вимагає двох проходів по тексті: одного для побудови моделі тексту (Таблиці частот і дерева Хаффмана), іншого для власне кодування.

У процесі роботи алгоритму стиснення вага вузлів в дереві кодування Хаффмана неухильно зростає. Перша проблема виникає тоді, коли вага кореня дерева починає перевершувати місткість комірки, в якій він зберігається. Як правило, це 16-бітове значення і, отже, не може бути більше, ніж 65535. Друга проблема, яка заслуговує ще більшої уваги, може виникнути значно раніше, коли розмір найдовшого коду Хаффмана перевершує місткість осередку, який використовується для того, щоб передати його у вихідний потік. Декодеру все одно, якої довжини код він декодує, оскільки він рухається зверху вниз по дереву кодування, вибираючи з вхідного потоку по одному біту. Кодер же повинен починати від листа дерева і рухатися вгору до кореня, збираючи біти, які потрібно передати. Зазвичай це відбувається зі змінною типу «ціле», і, коли довжина коду Хаффмана перевершує розмір типу «ціле» в бітах, настає переповнення.

## 1.4 Приклади обчислень

Програмну реалізацію алгоритму надано у додатку [Б.1](#).

## Лекція № 2

### Представлення виразів за допомогою дерев

#### Мета роботи

#### 2.1 Вступ

#### 2.2 Ключові терміни

#### 2.3 Розширені теоретичні відомості

#### 2.4 Приклади обчислень

## Лекція № 3

### Представлення багаторозгалужених дерев

#### Мета роботи

#### 3.1 Вступ

#### 3.2 Ключові терміни

#### 3.3 Розширені теоретичні відомості

#### 3.4 Приклади обчислень

#### Контрольні запитання

1. ?

# Лекція № 4

## Представлення графів

Мета роботи

4.1 Вступ

4.2 Ключові терміни

4.3 Розширені теоретичні відомості

4.4 Приклади обчислень

Контрольні запитання

1. ?

# Лекція № 5

## Алгоритми на графах

Мета роботи

5.1 Вступ

5.2 Ключові терміни

5.3 Розширені теоретичні відомості

5.4 Приклади обчислень

Контрольні запитання

1. ?



## Лекція № 6

### Використання багаторозгалужених дерев

#### Мета роботи

#### 6.1 Вступ

#### 6.2 Ключові терміни

#### 6.3 Розширені теоретичні відомості

#### 6.4 Приклади обчислень

#### Контрольні запитання

1. ?

# Лекція № 7

## Алгоритми сортування

### 7.1 Вступ

**Алгоритмом сортування** називається алгоритм для впорядкування деякого безлічі елементів. Зазвичай під алгоритмом сортування подразумевают алгоритм упорядкування безлічі елементів за зростанням або спаданням.

У разі наявності елементів з однаковими значеннями, у впорядкованій послідовності вони розташовуються поруч один за одним у будь-якому порядку. Однак іноді буває корисно зберігати первісний порядок елементів з однаковими значеннями.

Алгоритми сортування мають велике практичне застосування. Їх можна зустріти там, де мова йде про обробку та зберігання великих обсягів інформації. Деякі завдання обробки даних вирішуються простіше, якщо дані заздалегідь впорядкувати.

### 7.2 Ключові терміни

### 7.3 Розширені теоретичні відомості

В алгоритмах сортування лише частина даних використовується як ключ сортування. Ключем сортування називається атрибут (або декілька атрибутів), за значенням якого визначається порядок елементів. Таким чином, при написанні алгоритмів сортувань масивів слід врахувати, що ключ повністю або частково збігається з даними.

Практично кожен алгоритм сортування можна розбити на 3 частини:

1. порівняння, визначальне впорядкованість пари елементів;
2. перестановку, змінюють місце пару елементів;
3. власне сортують алгоритм, який здійснює порівняння і перестановку елементів до тих пір, поки всі елементи множини не будуть впорядковані.

#### 7.3.1 Оцінка алгоритмів сортування

Жодна інша проблема не породила такої кількості найрізноманітніших рішень, як завдання сортування. Універсального, найкращого алгоритму сортування на даний момент не існує. Проте, маючи приблизні характеристики вхідних даних, можна підібрати метод, який працює оптимальним чином. Для цього необхідно знати параметри, за якими буде проводитися оцінка алгоритмів.

**Час сортування** - основний параметр, що характеризує швидкодію алгоритму.

**Пам'ять** - один з параметрів, який характеризується тим, що ряд алгоритмів сортування вимагають виділення додаткової пам'яті під тимчасове зберігання даних. При оцінці використовуваної пам'яті не враховуватиметься місце, яке займає вихідний масив даних і незалежні від вхідної послідовності витрати, наприклад, на зберігання коду програми.

**Стійкість** - це параметр, який відповідає за те, що сортування не змінює взаємного розташування рівних елементів.

**Природність поведінки** - параметр, який вказує на ефективність методу при обробці вже відсортованих, або частково відсортованих даних. Алгоритм поводить себе природно, якщо враховує цю характеристику вхідної послідовності і працює краще.

### 7.3.2 Класифікація алгоритмів сортувань

Все розмаїття і різноманіття алгоритмів сортувань можна класифікувати за різними ознаками, наприклад, по стійкості, по поведінці, по використанню операцій порівняння, за потреби в додатковій пам'яті, по потреби в знаннях про структуру даних, що виходять за рамки операції порівняння, та інші.

Найбільш докладно розглянемо класифікацію алгоритмів сортування за сферою застосування. У даному випадку основні типи впорядкування діляться наступним чином.

**Внутрішнє сортування** - це алгоритм сортування, який в процесі упорядкування даних використовує тільки оперативну пам'ять (ОЗП) на комп'ютері. Тобто оперативної пам'яті достатньо для завантаження в неї сортованого масиву даних з довільним доступом до будь-якій комірки і власне для виконання алгоритму. Внутрішнє сортування застосовується у всіх випадках, за винятком однопрохідного зчитування даних і однопрохідної записи відсортованих даних. Залежно від конкретного алгоритму та його реалізації дані можуть сортуватися в тій же області пам'яті, або використовувати додаткову оперативну пам'ять.

**Зовнішнє сортування** - це алгоритм сортування, який при проведенні упорядкування даних використовує зовнішню пам'ять, як правило, жорсткі диски. Зовнішнє сортування розроблена для обробки великих списків даних, які не поміщаються в оперативну пам'ять. Звернення до різних носіїв накладає деякі додаткові обмеження на даний алгоритм: доступ до носія здійснюється послідовним чином, тобто в кожен момент часу можна вважати або записати тільки елемент, наступний за поточним; обсяг даних не дозволяє їм розміститися в ОЗУ.

Внутрішнє сортування є базовою для будь-якого алгоритму зовнішнього сортування - окремі частини масиву даних сортуються в оперативній пам'яті і за допомогою спеціального алгоритму зчіплюються в один масив, упорядкований по ключу.

## Лекція № 8

# Алгоритми внутрішнього сортування

## 8.1 Вступ

**Внутрішня сортування** - різновид алгоритмів сортування або їх реалізацій, при якій обсягу оперативної пам'яті достатньо для поміщення в неї сортованого масиву даних з довільним доступом до будь-якій комірки  $i$ , власне, для виконання алгоритму. У цьому випадку сортування відбувається максимально швидко, оскільки швидкість доступу до оперативної пам'яті значно вище, ніж до периферійних пристроїв (відповідно, час доступу значно менше). Залежно від конкретного алгоритму та його реалізації дані можуть сортуватися в тій же області пам'яті, або використовувати додаткову оперативну пам'ять. Внутрішня сортування є базовою для будь-якого алгоритму зовнішньої сортування - окремі частини масиву даних сортуються в оперативній пам'яті і за допомогою спеціального алгоритму зчіплюються в один масив, упорядкований по ключу.

## 8.2 Ключові терміни

## 8.3 Розширені теоретичні відомості

### 8.3.1 Сортування Вибором

Один з найпростіших методів сортування працює таким чином: знаходимо найменший елемент в масиві і обмінюємо його з елементом знаходяться на першому місці, потім повторюємо процес із другої позиції у файлі і знайдений елемент обмінюємо з другим елементом і так далі поки весь масив НЕ БУДЕ відсортований. Цей метод називається сортування вибором оскільки він працює циклічно вибираючи найменший з елементів, що залишилися.

Лістинг 8.1 – Сортування вибором

---

```
procedure selection;
var
i, j, min, t: integer;
begin
  for i: = 1 to N-1 do
    begin
      min: = i;
      for j: = i + 1 to N do
        if a [j] <a [min] then
min: = j;
      t: = a [min];
a [min]: = a [i];
```

```
a [i]: = t;  
    end;  
end;
```

---

У міру просування покажчика і зліва направо через файл, елементи зліва від покажчика знаходяться вже у своїй кінцевій позиції (і їх не більше вже не будуть чіпати), тому масив стає повністю відсортованим до того моменту, коли покажчик досягає правого краю.

Цей метод - один з найпростіших, і від працює дуже добре для невеликих файлів. Його "внутрішній цикл" складається з порівняння  $a[i] < a[\text{min}]$  (плюс код необхідний для збільшення  $j$  та перевірки на те, що він не перевищив  $N$ ), що навряд чи можна ще спростити. Нижче ми обговоримо те, скільки швидше за все раз ці інструкції будуть виконуватися.

Більше того, незважаючи на те, що цей метод очевидно є методом "грубої сили" він має дуже важливе застосування: оскільки кожен елемент пересувається не більше ніж раз, то він дуже хороший для великих записів з маленькими ключами. Це обговорюється нижче.

Сортування вставкою

### 8.3.2 Сортування вставкою

- це метод який майже настільки ж простий, що і сортування вибором, але набагато більш гнучкий. Цей метод часто використовують при сортуванні карт: беремо один елемент і вставляємо його в потрібне місце серед тих, що ми вже обробили (тим самим залишаючи їх відсортованими).

#### Лістинг 8.2 – Сортування вставкою

---

```
type  
Index = 0..n;  
var  
a: array [1..n] of elem;  
procedure Insert;  
    var i, j: index;  
    x: elem;  
    begin  
        for i: = 1 to n do  
            begin  
                x: = a [i]; a [0]: = x; j: = i-1;  
                while x.key < a [j] .key do  
                    begin  
                        a [j + 1]: = a [j]; j: = j-1;  
                    end;  
                a [j + 1]: = x;  
            end;  
        end;  
    end;
```

---

Також як і в сортуванні вибором, в процесі сортування елементи зліва від покажчика і знаходяться вже в сортованому порядку, але вони не обов'язково знаходяться у своїй останній позиції, оскільки їх ще можуть пересунути направо щоб вставити більш маленькі елементи

зустрінуті пізніше .. Однак масив стає повністю Сортувати коли показчик досягає правого краю. Бульбашкова Сортування

### 8.3.3 Бульбашкове сортування

Необхідно проходити через масив, обмінюючи якщо потрібно елементи; коли на якомусь кроці обмінів не буде потрібно - сортування закінчена. Реалізація цього методу дана нижче.

Лістинг 8.3 – Бульбашкове сортування

---

```
procedure bubble;  
var i, j, t: byte;  
begin  
  for i: = 2 to N do  
    for j: = N down to i do  
      if x [i-1]> x [j] then  
        begin t: = x [j-1]; x [j-1]: = x [j]; x [j]: = t; end;  
      end;  
    end;  
end;
```

---

Щоб повірити в те, що вона насправді працює, може знадобитися деякий час. Для цього зауважте, що коли під час першого проходу зустрічаємо максимальний елемент, обмінюємо його з кожним елементом праворуч від нього поки він не опиниться у вкрай правій позиції. На другому проході поміщаємо другу максимальний елемент в передостанню позицію і так далі. Бульбашкова сортування працює також як і сортування вибором, хоча вона і робить набагато більше роботи на те, щоб перемістити елемент у його кінцеву позицію. Характеристики Найпростіших сортувань

Властивість 1 Сортування вибором використовує близько  $N^2 / 2$  порівнянь і  $N$  обмінів.

Властивість 2 Сортування вставкою використовує близько  $N^2 / 4$  порівнянь і  $N^2 / 8$  обмінів у середньому, і в два рази більше в найгіршому випадку.

Властивість 3 Бульбашкова сортування використовує близько  $N^2 / 2$  порівнянь і  $N^2 / 2$  обмінів у середньому і найгіршому випадках.

Властивість 4 Сортування вставкою линейна для "майже сортованих" файлів.

Властивість 5 Сортування вибором линейна для файлів з великими записами і маленькими ключами.

## 8.4 Приклади обчислень

# Лекція № 9

## Алгоритми зовнішнього сортування

### 9.1 Вступ

**Зовнішнє сортування** - сортування даних, розташованих на периферійних пристроях і не вміщаються в оперативну пам'ять, тобто коли застосувати одну з внутрішніх сортувань неможливо. Варто відзначити, що внутрішнє сортування значно ефективніше зовнішнього, так як на звернення до оперативної пам'яті витрачається набагато менше часу, ніж до магнітних дисків, стрічок і т.п.

Найбільш часто зовнішнє сортування використовується в СУБД.

Дані, що зберігаються на зовнішніх пристроях, мають великий обсяг, що не дозволяє їх цілком перемістити в оперативну пам'ять, відсортувати з використанням одного з алгоритмів внутрішнього сортування, а потім повернути їх на зовнішній пристрій. У цьому випадку здійснювалося б мінімальну кількість проходів через файл, тобто було б однократне читання і однократна запис даних. Однак на практиці доводиться здійснювати читання, обробку і запис даних у файл по блоках, розмір яких залежить від операційної системи та наявного обсягу оперативної пам'яті, що призводить до збільшення числа проходів через файл і помітного зниження швидкості сортування.

### 9.2 Ключові терміни

**Зовнішнє сортування** - це сортування даних, які розташовані на зовнішніх пристроях і не вміщаються в оперативну пам'ять.

**Фаза** - це дії по одноразовій обробці всієї послідовності елементів.

**Серія (упорядкований відрізок)** - це послідовність елементів, що упорядкована по ключу. Кількість елементів в серії називається довжиною серії.

**Двофазне сортування** - це сортування, в якому окремо реалізується дві фази: розподіл і злиття.

**Однофазне сортування** - це сортування, в якому об'єднані фази розподілу і злиття в одну.

**Двоколіїним злиттям** називається сортування, в якому дані розподіляються на два допоміжних файли.

**Багатоколіїним злиттям** називається сортування, в якому дані розподіляються на  $N$  ( $N > 2$ ) допоміжних файлів.

<b>Исходный файл f: 5 7 3 2 8 4 1</b>		
	<b>Распределение</b>	<b>Слияние</b>
<b>1 проход</b>	f1: 5 3 8 1	f: 5 7 2 3 4 8 1
	f2: 7 2 4	
<b>2 проход</b>	f1: 5 7 4 8	f: 2 3 5 7 1 4 8
	f2: 2 3 1	
<b>3 проход</b>	f1: 2 3 5 7	f: 1 2 3 4 5 7 8
	f2: 1 4 8	

Рис. 9.1 – Демонстрація сортування двоколіїному двофазним простим злиттям

## 9.3 Розширені теоретичні відомості

### 9.3.1 Сортування простим злиттям

Одне з сортувань на основі злиття називається простим злиттям.

Алгоритм сортування простим злиття є найпростішим алгоритмом зовнішньої сортування, заснований на процедурі злиття серією.

У даному алгоритмі довжина серій фіксується на кожному кроці. У вихідному файлі всі серії мають довжину 1, після першого кроку вона дорівнює 2, після другого - 4, після третього - 8, після k-го кроку -  $2^k$ .

Алгоритм сортування простим злиттям

1. Вихідний файл f розбивається на два допоміжних файлу f1 і f2.
2. Допоміжні файли f1 і f2 зливаються в файл f, при цьому поодинокі елементи утворюють впорядковані пари.
3. Отриманий файл f знову обробляється, як зазначено в кроках 1 і 2. При цьому впорядковані пари переходять у впорядковані четвірки.
4. Повторюючи кроки, зливаємо четвірки в вісімки і т.д., кожен раз подвоюючи довжину злитих послідовностей до тих пір, поки не буде впорядкований цілком весь файл (рис. 9.1).

Після виконання i проходів отримуємо два файли, що складаються з серій довжини  $2^i$ . Закінчення процесу відбувається при виконанні умови  $2^i \geq n$ . Отже, процес сортування простим злиттям вимагає порядку  $O(\log n)$  проходів за даними.

Ознаками кінця сортування простим злиттям є наступні умови:

1. довжина серії не менша кількості елементів у файлі (визначається після фази злиття);
2. кількість серій рівно 1 (визначається на фазі злиття).
3. при однофазному сортуванні другий за рахунком допоміжний файл після розподілу серій залишився порожнім.



Приклад програмної реалізації надано у додатку [Б.2](#);

## 9.4 Приклади обчислень

## Лекція № 10

### Характеристика алгоритмів порівняння методів сортування

#### Мета роботи

#### 10.1 Вступ

#### 10.2 Ключові терміни

#### 10.3 Розширені теоретичні відомості

#### 10.4 Приклади обчислень

#### Контрольні запитання

1. ?

## Лекція № 11

### Алгоритми розподілу обчислювального процесу

#### Мета роботи

#### 11.1 Вступ

#### 11.2 Ключові терміни

#### 11.3 Розширені теоретичні відомості

#### 11.4 Приклади обчислень

#### Контрольні запитання

1. ?

## Лекція № 12

### Архітектура розподілених обчислень

#### Мета роботи

#### 12.1 Вступ

#### 12.2 Ключові терміни

#### 12.3 Розширені теоретичні відомості

#### 12.4 Приклади обчислень

#### Контрольні запитання

1. ?

## Лекція № 13

### Процеси і потоки в обчисленні

#### Мета роботи

#### 13.1 Вступ

#### 13.2 Ключові терміни

#### 13.3 Розширені теоретичні відомості

#### 13.4 Приклади обчислень

#### Контрольні запитання

1. ?

## Лекція № 14

### Реалізація багатозадачного середовища

#### Мета роботи

#### 14.1 Вступ

#### 14.2 Ключові терміни

#### 14.3 Розширені теоретичні відомості

#### 14.4 Приклади обчислень

#### Контрольні запитання

1. ?

## Лекція № 15

### Бібліотеки організації розподілених обчислень

#### Мета роботи

#### 15.1 Вступ

#### 15.2 Ключові терміни

#### 15.3 Розширені теоретичні відомості

#### 15.4 Приклади обчислень

#### Контрольні запитання

1. ?

## Лекція № 16

### Програмна модель OpenMP

#### Мета роботи

#### 16.1 Вступ

#### 16.2 Ключові терміни

#### 16.3 Розширені теоретичні відомості

#### 16.4 Приклади обчислень

#### Контрольні запитання

1. ?



## Лекція № 17

### Конструкції OpenMP для розподілу робіт

#### Мета роботи

#### 17.1 Вступ

#### 17.2 Ключові терміни

#### 17.3 Розширені теоретичні відомості

#### 17.4 Приклади обчислень

#### Контрольні запитання

1. ?

## Лекція № 18

### Умови виконання (планування)

### Мета роботи

#### 18.1 Вступ

#### 18.2 Ключові терміни

#### 18.3 Розширені теоретичні відомості

#### 18.4 Приклади обчислень

### Контрольні запитання

1. ?

# Лекція № 19

## Бібліотечні функції OpenMP

Мета роботи

19.1 Вступ

19.2 Ключові терміни

19.3 Розширені теоретичні відомості

19.4 Приклади обчислень

Контрольні запитання

1. ?

Лекція № 20  
Бібліотека МРІ

Мета роботи

20.1 Вступ

20.2 Ключові терміни

20.3 Розширені теоретичні відомості

20.4 Приклади обчислень

Контрольні запитання

1. ?

## Лекція № 21

### Режими обліку повідомлень в MPI

#### Мета роботи

#### 21.1 Вступ

#### 21.2 Ключові терміни

#### 21.3 Розширені теоретичні відомості

#### 21.4 Приклади обчислень

#### Контрольні запитання

1. ?

# Додаток А

## Правила оформлення звіту

### А.1 Титульна сторінка лабораторної роботи

---

МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ  
ХЕРСОНСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ  
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Звіт до лабораторної роботи №123  
з дисципліни «Web-програмування»

Тема: «Основи мережі Internet»

Виконав  
ст.групи хПР1

Пупкін А.А.

Перевірив  
ст.викладач

Іванов Б.Б.

Херсон 2012

## А.2 Приклади блок-схем

Правила виконання блок-схем задані наступними документами:

- ◇ ГОСТ 19.701-90. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения
- ◇ ГОСТ 19.002-80. Схемы алгоритмов и программ. Правила выполнения
- ◇ ГОСТ 19.003-80. Схемы алгоритмов и программ. Обозначения условные графические

## Додаток Б

### Лістинги

#### Б.1 Програмна реалізація алгоритму Хаффмана за допомогою кодового дерева

Лістинг Б.1 – Програмна реалізація алгоритму Хаффмана

---

```
#include "stdafx.h"
#include <iostream>
using namespace std;
struct sym {
    unsigned char ch;
    float freq;
    char code[255];
    sym *left;
    sym *right;
};
void Statistics(char *String);
sym *makeTree(sym *psym[],int k);
void makeCodes(sym *root);
void CodeHuffman(char *String,char *BinaryCode, sym *root);
void DecodeHuffman(char *BinaryCode,char *ReducedString,
                    sym *root);
int chh;//переменная для подсчета информации из строки
int k=0;
//счётчик количества различных букв, уникальных символов
int kk=0;//счётчик количества всех знаков в файле
int kolvo[256]={0};
//инициализируем массив количества уникальных символов
sym symbols[256]={0};//инициализируем массив записей
sym *psym[256];//инициализируем массив указателей на записи
float summir=0;//сумма частот встречаемости

int _tmain(int argc, _TCHAR* argv[]){
    char *String = new char[1000];
    char *BinaryCode = new char[1000];
    char *ReducedString = new char[1000];
    String[0] = BinaryCode[0] = ReducedString[0] = 0;
    cout << "Введите строку для кодирования: ";
    cin >> String;
    sym *symbols = new sym[k];
    //создание динамического массива структур symbols
    sym **psum = new sym*[k];
    //создание динамического массива указателей на symbols
```



```

Statistics(String);
sym *root = makeTree(psym,k);
//вызов функции создания дерева Хаффмана
makeCodes(root);//вызов функции получения кода
CodeHuffman(String,BinaryCode,root);
cout << "Закодированная строка: " << endl;
cout << BinaryCode << endl;
DecodeHuffman(BinaryCode,ReducedString, root);
cout << "Раскодированная строка: " << endl;
cout << ReducedString << endl;
delete psum;
delete String;
delete BinaryCode;
delete ReducedString;
system("pause");
return 0;
}

//рекурсивная функция создания дерева Хаффмана
sym *makeTree(sym *psym[],int k) {
    int i, j;
    sym *temp;
    temp = new sym;
    temp->freq = psym[k-1]->freq+psym[k-2]->freq;
    temp->code[0] = 0;
    temp->left = psym[k-1];
    temp->right = psym[k-2];
    if ( k == 2 )
        return temp;
    else {
        //внесение в нужное место массива элемента дерева Хаффмана
        for ( i = 0; i < k; i++)
            if ( temp->freq > psym[i]->freq ) {
                for( j = k - 1; j > i; j--)
                    psym[j] = psym[j-1];
                psym[i] = temp;
                break;
            }
    }
    return makeTree(psym,k-1);
}

//рекурсивная функция кодирования дерева
void makeCodes(sym *root) {
    if ( root->left ) {
        strcpy(root->left->code,root->code);
        strcat(root->left->code,"0");
        makeCodes(root->left);
    }
    if ( root->right ) {

```

```

    strcpy(root->right->code, root->code);
    strcat(root->right->code, "1");
    makeCodes(root->right);
}
}
/*функция подсчета количества каждого символа и его вероятности*/
void Statistics(char *String){
    int i, j;
    //побайтно считываем строку и составляем таблицу встречаемости
    for ( i = 0; i < strlen(String); i++){
        chh = String[i];
        for ( j = 0; j < 256; j++){
            if (chh==simbols[j].ch) {
                kolvo[j]++;
                kk++;
                break;
            }
            if (simbols[j].ch==0){
                simbols[j].ch=(unsigned char)chh;
                kolvo[j]=1;
                k++; kk++;
                break;
            }
        }
    }
    // расчет частоты встречаемости
    for ( i = 0; i < k; i++)
        simbols[i].freq = (float)kolvo[i] / kk;
    // в массив указателей заносим адреса записей
    for ( i = 0; i < k; i++)
        psym[i] = &simbols[i];
    //сортировка по убыванию
    sym tempp;
    for ( i = 1; i < k; i++)
        for ( j = 0; j < k - 1; j++)
            if ( simbols[j].freq < simbols[j+1].freq ){
                tempp = simbols[j];
                simbols[j] = simbols[j+1];
                simbols[j+1] = tempp;
            }
    for( i=0;i<k;i++) {
        summir+=simbols[i].freq;
        printf("Ch=%d\tFreq=%f\tPPP=%c\t\n", simbols[i].ch,
            simbols[i].freq, psym[i]->ch, i);
    }
    printf("\nSlova=%d\tSummir=%f\n", kk, summir);
}
//функция кодирования строки

```

```

void CodeHuffman(char *String,char *BinaryCode, sym *root){
    for (int i = 0; i < strlen(String); i++){
        chh = String[i];
        for (int j = 0; j < k; j++){
            if ( chh == simbols[j].ch ){
                strcat(BinaryCode,simbols[j].code);
            }
        }
    }
}

//функция декодирования строки
void DecodeHuffman(char *BinaryCode,char *ReducedString,
                    sym *root){
    sym *Current;// указатель в дереве
    char CurrentBit;// значение текущего бита кода
    int BitNumber;
    int CurrentSimbol;// индекс распаковываемого символа
    bool FlagOfEnd; // флаг конца битовой последовательности
    FlagOfEnd = false;
    CurrentSimbol = 0;
    BitNumber = 0;
    Current = root;
    //пока не закончилась битовая последовательность
    while ( BitNumber != strlen(BinaryCode) ) {
        //пока не пришли в лист дерева
        while (Current->left != NULL && Current->right != NULL &&
                BitNumber != strlen(BinaryCode) ) {
            //читаем значение очередного бита
            CurrentBit = BinaryCode[BitNumber++];
            //бит - 0, то идем налево, бит - 1, то направо
            if ( CurrentBit == '0' )
                Current = Current->left;
            else
                Current = Current->right;
        }
        //пришли в лист и формируем очередной символ
        ReducedString[CurrentSimbol++] = Current->ch;
        Current = root;
    }
    ReducedString[CurrentSimbol] = 0;
}

```

---

## Б.2 Опис функції сортування простим злиттям

Лістинг Б.2 – Програмна реалізація функції сортування простим злиттям

```

//Описание функции сортировки простым слиянием
void Simple_Merging_Sort (char *name){

```

```

int a1, a2, k, i, j, kol, tmp;
FILE *f, *f1, *f2;
kol = 0;
if ( (f = fopen(name,"r")) == NULL )
    printf("\u0418\u0441\u0445\u043e\u0434\u043d\u044b\u0439 \u0444\u0430\u0439\u043b \u043d\u0435 \u043c\u043e\u0436\u0435\u0442 \u0431\u044b\u0442\u044c \u043f\u0440\u043e\u0447\u0438\u0442\u0430\u043d...\");
else {
    while ( !feof(f) ) {
        fscanf(f,"%d",&a1);
        kol++;
    }
    fclose(f);
}
k = 1;
while ( k < kol ){
    f = fopen(name,"r");
    f1 = fopen("smsort_1","w");
    f2 = fopen("smsort_2","w");
    if ( !feof(f) ) fscanf(f,"%d",&a1);
    while ( !feof(f) ){
        for ( i = 0; i < k && !feof(f) ; i++ ){
            fprintf(f1,"%d_",a1);
            fscanf(f,"%d",&a1);
        }
        for ( j = 0; j < k && !feof(f) ; j++ ){
            fprintf(f2,"%d_",a1);
            fscanf(f,"%d",&a1);
        }
    }
    fclose(f2);
    fclose(f1);
    fclose(f);

    f = fopen(name,"w");
    f1 = fopen("smsort_1","r");
    f2 = fopen("smsort_2","r");
    if ( !feof(f1) ) fscanf(f1,"%d",&a1);
    if ( !feof(f2) ) fscanf(f2,"%d",&a2);
    while ( !feof(f1) && !feof(f2) ){
        i = 0;
        j = 0;
        while ( i < k && j < k && !feof(f1) && !feof(f2) ) {
            if ( a1 < a2 ) {
                fprintf(f,"%d_",a1);
                fscanf(f1,"%d",&a1);
                i++;
            }
            else {
                fprintf(f,"%d_",a2);

```

```

        fscanf(f2,"%d",&a2);
        j++;
    }
}
while ( i < k && !feof(f1) ) {
    fprintf(f,"%d_",a1);
    fscanf(f1,"%d",&a1);
    i++;
}
while ( j < k && !feof(f2) ) {
    fprintf(f,"%d_",a2);
    fscanf(f2,"%d",&a2);
    j++;
}
}
while ( !feof(f1) ) {
    fprintf(f,"%d_",a1);
    fscanf(f1,"%d",&a1);
}
while ( !feof(f2) ) {
    fprintf(f,"%d_",a2);
    fscanf(f2,"%d",&a2);
}
fclose(f2);
fclose(f1);
fclose(f);
k *= 2;
}
remove("smsort_1");
remove("smsort_2");
}

```

---