МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Київський національний університет будівництва і архітектури

ТЕОРІЯ АЛГОРИТМІВ

Методичні вказівки
до виконання практичних та лабораторних робіт
для студентів спеціальності
122 Комп’ютерні науки

Київ 2020

УДК 004.421
ББК 22.18
А45

Укладачі: С.В. Білощицька, канд. техн. наук, доцент
Н.Д. Федоренко, канд. техн. наук, професор
О.І. Баліна, канд. техн. наук, доцент
І.С. Безклубенко, канд. техн. наук, доцент
О.В. Доля, канд. фіз.-мат. наук, доцент
А.О. Білощицький, д.т.н., професор

Рецензент Є.В. Бородавка, д.т.н., професор

Відповідальний за випуск В.М. Міхайленко, д.т.н., професор, завідувач кафедри інформаційних технологій проектування та прикладної математики

Затверджено на засіданні кафедри прикладної математики, протокол № від року.

Видається в авторській редакції.

Теорія алгоритмів: Методичні вказівки до виконання практичних та лабораторних робіт / Уклад. С.В. Білощицька, Н.Д. Федоренко, О.І. Баліна, І.С. Безклубенко, О.В. Доля, А.О. Білощицький. – К.: КНУБА, 2020. – 41с.

Містять теоретичні відомості і рекомендації щодо виконання лабораторних робіт з дисципліни та вимоги до оформлення звіту. Спрямовані на організацію самостійної роботи студентів.

Призначені для студентів спеціальності 122 Комп’ютерні науки для практичного використання при виконанні лабораторних робіт.

Зміст

Вступ

Лабораторні роботи є логічним продовженням лекційного курсу з дисципліни “Теорія алгоритмів” і є перехідною ланкою від теоретичного курсу до набуття практичних навичок всіх етапів розробки надійної програми для розв’язку задачі на ЕОМ, починаючи від аналізу умови задачі і закінчуючи складанням звіту по написаній роботі.

Кожна лабораторна робота містить наступні види робіт:

- аналіз умови задачі і розробка підходу до її розв’язку.
- покрокову розробку алгоритму розв’язку і його опис.
- обґрунтування алгоритму.
- складання схеми алгоритму.
- оцінку О-складності складеного алгоритму.
- вибір і обґрунтування представлення вхідних, вихідних і проміжних даних.
- написання програми, що реалізує цей алгоритм.
- обчислення часу виконання програми.
- вибір наборів тестів, на яких буде перевірятися програма.
- демонстрація правильної роботи програми на обраному наборі тестів.
- складання і захист звіту.

В результаті виконання лабораторних робіт студенти повинні навчитись: самостійно проводити аналіз простих задач; складати алгоритми їх розв’язання; вміти аналізувати алгоритми і визначати їх складність; мати практичні навички у написанні програм; знати правила діагностики та відлагоджування програм; вміти тестувати програми.

Вказівки до виконання і оформлення звіту з лабораторних робіт.

Кожен студент отримує індивідуальне завдання, відповідно до варіанту за списком. Підготовка до лабораторної роботи здійснюється в позааудиторний час. Виконавши лабораторну роботу, студент складає звіт, який містить такі розділи:

- Тема і мета роботи.
- Умова завдання.
- Алгоритм розв’язку задачі: математична модель та схема алгоритму.

– Аналіз алгоритму.
– Обґрунтування вибору типу даних.
– Текст програми.
– Результати виконання програми на тестовому прикладі.
– Обґрунтування правильності розробленої програми.
– Висновки.

Титульний лист оформлюється згідно з дод. 1.

Робота має бути виконана у визначені викладачем терміни з урахуванням перерахованих вимог. При захисті звіту необхідно відповідати на контрольні питання і вміти пояснювати роботу програми.

Лабораторна робота №1.
Стеки. Реалізація стеків за допомогою масивів та покажчиків.

Мета роботи: Закріпити техніку роботи з напівстатичними структурами даних на прикладі стеку. Розглянути основні операції зі стеком і познайомитись з типовими прикладами застосування стеків.

Теоретичні відомості.

Стек – це спеціальний тип списку, в якому всі операції включення і виключення виконуються на одному кінці, що називається *вершиною*, тому стек називають списком типу LIFO (last-in-first-out – останнім включається – першим виключається).

Стек можна організувати на базі будь-якої структури даних, де можливе зберігання декількох однотипних елементів і де можна реалізувати визначення стека: лінійний масив, типізований файл, однонаправлений або двонаправленный список. Над стеком і його елементами виконуються такі типові операції: додавання елемента в стек; видалення елемента зі стеку; перевірка, чи порожній стек; перегляд елемента у вершині стека без видалення; очищення стека.

Реалізація стеку за допомогою масиву.

Існує декілька фізичних представлень стеків за допомогою масивів (рис.1, рис.2, рис.3).

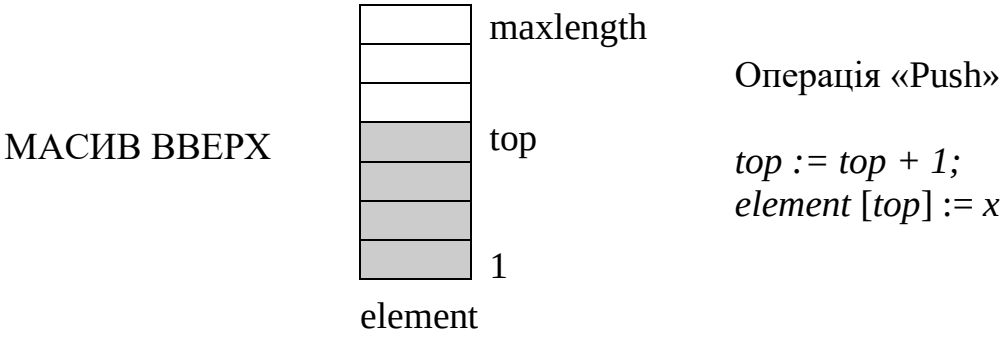


Рис.1. Представлення стеків за допомогою масиву вгору

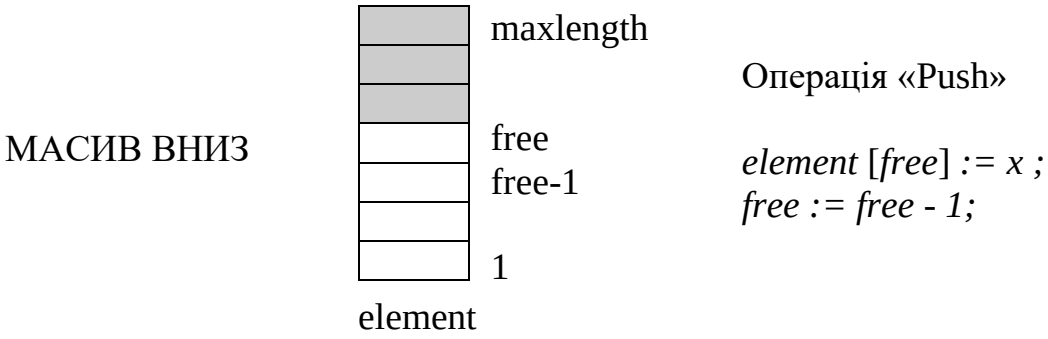


Рис. 2. Представлення стеків за допомогою масиву вниз

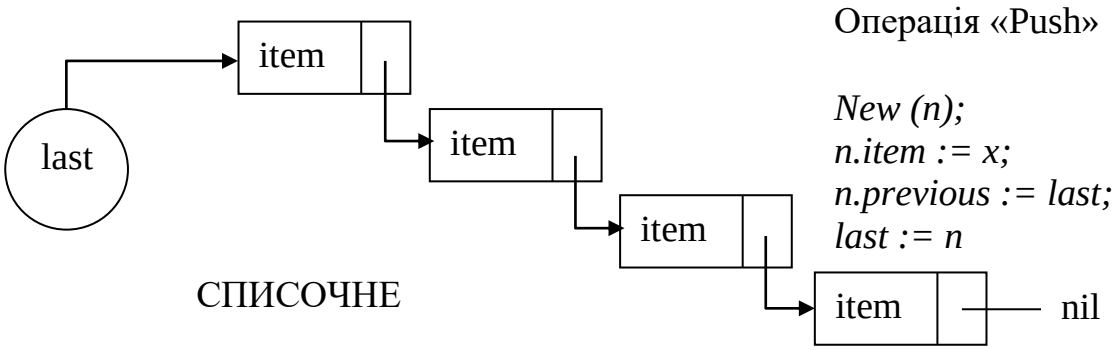


Рис. 3. Списочне представлення стеків

1. МАСИВ ВВЕРХ – представлення стеку за допомогою масиву *element* і цілого числа *top* з діапазоном значень від 0 (для пустого стеку) до *maxlength* (для заповненого стеку). Елементи стеку зберігаються в масиві та індексуються від 1 до *top*.

2. МАСИВ ВНИЗ схожий на МАСИВ ВВЕРХ, але елементи розміщуються в кінець стеку, а не в початок. Число *free* є індексом верхньої вільної позиції в стеку або 0, якщо всі позиції в масиві зайняті, і змінюється в діапазоні від *maxlength* (для порожнього стеку) до 0 (для

заповненого). Елементи стеку зберігаються в масиві й індексуються від *maxlength* до *free*+1.

3. При СПИСОЧНОМУ представленні кожний елемент стеку зберігається у вузлі з двома полями: *item*, що вміщує сам елемент, і *previous*, що вміщує покажчик на вузол з попереднім елементом. Для такого представлення потрібен покажчик *last* на вузол, що вміщує вершину стека.

Поряд з кожним представленням на рисунку наведено фрагмент програми з відповідною реалізацією основної стекової операції *push* (включення елементу *x* у вершину стеку).

Для представлення за допомогою масивів команди збільшують або зменшують покажчик на вершину (*top* або *free*) і записують *x* у відповідний елемент масиву. Так як ці представлення підтримують стеки з не більше чим *maxlength* елементами, то коректні реалізації повинні містити тести, які захищають від переповнення, виду:

if *top* = maxlength **then** ...

if *free* = 0 **then** ...

СПИСОЧНЕ представлення для включення елемента потребує чотирьох дій: створення нового вузла *n*; присвоювання *x* полю *item* нового вузла; приєднання нового вузла до вершини стеку шляхом надання його полю *previous* поточного значення покажчика *last*; змінювання *last* так, щоб він посилався на щойно створений вузол.

Хоча ці представлення зустрічаються найчастіше, існує і багато інших представлень стеків. Наприклад, якщо потрібні два стеки з однотипними елементами і пам’ять для них обмежена, то можна використовувати один масив з двома мітками вершин *top* як в представленні МАССИВ ВВЕРХ і *free* як в МАССИВ ВНИЗ. При цьому один стек буде рости вгору, а інший вниз (рис.4). Умовою повного заповнення цього представлення є рівність *top = free.*

Перевага такого представлення полягає у зменшенні ризику переповнити пам’ять: при двох масивах розміру *n*, що представляють стеки способом МАССИВ ВВЕРХ або МАССИВ ВНИЗ, пам’ять вичерпується, коли лише будь який зі стеків досягне *n* елементів. А у випадку одного масиву розміру 2*n*, що містить два стеки один напроти другого, робота

продовжується до тих пір, доки їх спільна довжина не перевищить 2*n*, що є малоймовірним, якщо стеки ростуть незалежно один від одного.

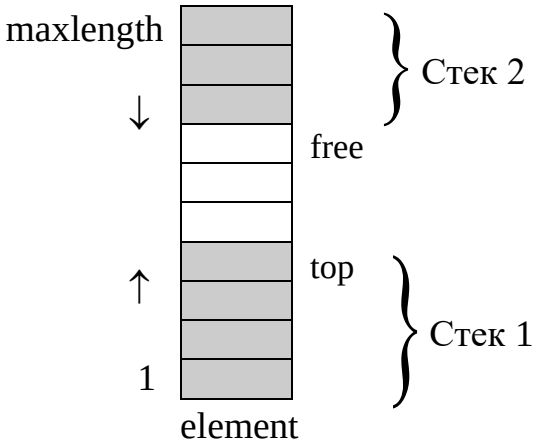


Рис.4. Представлення двох стеків один до одного

Абстрактні типи даних сімейства Stack (Стек) зазвичай використовують наступні п’ять операторів:

1. MAKENULL (S). Робить стек S порожнім.
2. TOP (S). Повертає елемент з вершини стека S.
3. POP (S). Видаляє елемент з вершини стека (виштовхує зі стеку).
4. PUSH(*x*,S) Включає елемент *x* у вершину стека S (заштовхує елемент в стек). Елемент, що раніше знаходився в вершині стека, стає елементом наступним за вершиною.
5. EMPTY(S). Ця функція повертає значення *true* (істина), якщо стек S порожній і значення *false* в протилежному випадку.

Для реалізації стеків за допомогою масивів абстрактний тип Stack можна визначити наступним чином:

```
Type
    Stack=Record
        top: integer;
        element: array [1..maxlength] of elementtype
    end;
```

Наведемо реалізацію стеку представленням МАСИВ ВНИЗ. В цій реалізації стек складається з послідовності елементів element[*free*+1], element[*free*+2], …, element[*maxlength*]. Розглянемо п’ять типових операторів: MAKENULL, EMPTY, TOP, POP та PUSH.

```
procedure MAKENULL (var S: Stack);
begin
        S. free := maxlength
end;
```

```
procedure POP (var S : Stack);
begin
    if EMPTY (S) then
        error (`Стек порожній`)
    else  S. free := S. free +1
end;
```

```
function TOP (var S: Stack): elementtype;
begin
    if EMPTY (S) then
        error (`Стек порожній`)
    else TOP := S.element[S.free +1]
end;
```

```
procedure PUSH (x : elementtype; var S : Stack);
begin
    if S. free = 0 then
        error (`Стек повний`)
    else
  begin  S.element[S. free] := x;
        S. free:=S. free -1;
  end
end;
```

```
function EMPTY (S: Stack): boolean;
begin
     if S.free > maxlength  then
        EMPTY := true
     else
        EMPTY := false
end;
```

Реалізація стеку за допомогою покажчиків.

У цій реалізації стек складається з вузлів, кожен з яких містить елемент стеку і покажчик на наступний вузол. Вузол, що містить останній елемент, має покажчик *nil* (нуль). Вузол *header* (заголовок) не містить елементів стеку, а тільки показує на перший елемент списку. Якщо стек порожній, заголовок містить покажчик *nil.* При цьому потрібна додаткова пам’ять для зберігання покажчиків. На рис. 5 наведено представлення стеку за допомогою зв’язного списку описаного виду.

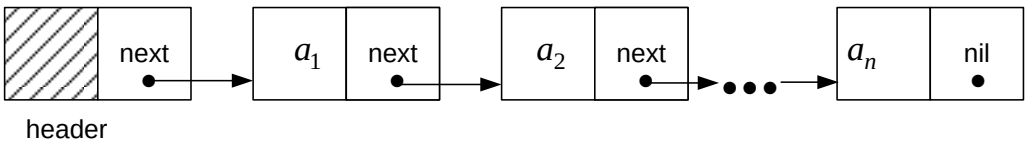


Рис. 5. Представлення стеку за допомогою зв’язного списку

Таким чином, описати стек, який містить цілі числа, можна наступним чином:

```
Type
        PStack = ^TStack;
        TStack = Record
                Data : Integer;
                Next : PStack;
        end;
Var     Stack : PStack;
```

Якщо стек порожній, то значення змінної Stack дорівнює nil.

Включення елемента в стек відбувається аналогічно до вставки нового елемента в початок списку. При виключенні елемента зі стеку деякій змінній N присвоюється значення першого елементу стека і змінюється значення покажчика на початок стеку.

Включення елемента в стек	Виключення елемента зі стеку
Var x : PStack; …… New(x); x^.Data := …….; x^.Next := Stack; Stack := x;	**Var** N : Integer; x : PStack; ……. N := Stack^.Data; x:= Stack; Stack := Stack^.Next; Dispose(x);

Завдання.

Написати програму: а) реалізації стеків за допомогою списків;
б) реалізації стеків за допомогою масивів, використовуючи МАСИВ ВГОРУ. Виконати: додавання елемента в стек; видалення елемента зі стеку; перевірка, чи порожній стек; перегляд елемента у вершині стека без видалення; очищення стека.

Контрольні запитання.

1. Що таке структура даних? Що таке стек?
2. Чим відрізняються статичні та динамічні структури даних?
3. У чому полягає схожість та відмінність структур даних список та стек?

4. Чи можна дістатися середини або кінця стеку, обійшовши його початок (вершину)?
5. Наведіть приклади з життя де зустрічається «принцип стеку».
6. У чому відмінність реалізацій стеку МАСИВ ВНИЗ та МАСИВ ВГОРУ?
7. У чому переваги представлення двох стеків один до одного?

Лабораторна робота №2.
Черги. Реалізація черг за допомогою циклічних масивів та покажчиків.

Мета роботи: Закріпити техніку роботи з напівстатичними структурами даних на прикладі черги. Розглянути основні операції з чергою і познайомитись з типовими прикладами застосування черг.

Теоретичні відомості.

Черга – це спеціальний тип списку, де елементи включаються на одному кінці, який називається хвостом (tail), а вилучаються на іншому – голові (head). Чергу називають списком типу FIFO (first-in-first-out – першим включається – першим виключається).

Оператори, що виконуються над чергами, аналогічні операторам стеків:

1. MAKENULL (Q). Робить чергу Q порожньою.
2. HEAD (Q) – функція повертає перший елемент черги Q. Цю функцію можна реалізувати за допомогою операторів списку як RETRIEVE (FIRST(Q),Q).
3. ENQUEUE (x,Q) вставляє елемент *x* в хвіст черги Q. За допомогою операторів списку можна реалізувати цю функцію наступним чином INSERT(Q),Q).
4. DEQUEUE (Q) видаляє перший елемент черги Q. За допомогою операторів списку реалізується як DELETE (FIRST(Q),Q).
5. EMPTY(Q). Повертає значення true тоді і тільки тоді, коли Q є порожньою чергою.

Реалізація черг за допомогою покажчиків.

Для черг більш ефективним є їх представлення основане на масивах з безпосереднім використанням покажчиків.

Для початку об’явимо комірки наступним чином:

```
Type
    celltype = Record
        element: elementtype;
        next: ^ celltype
    end;
```

Тепер можна визначити список, який вміщує покажчики на початок і кінець черги. Першим елементом черги буде елемент заголовку, в якому поле element не враховується.

```
Type
    Queue = Record
        head, tail: ^celltype
    end;
```

Наведемо програми для п’яти операторів, які виконуються над чергами:

```
procedure MAKENULL (var Q : Queue);
begin
    new(Q.head);  {створення комірки заголовку}
    Q.head^.next := nil;
    Q.tail := Q.head
end;
```

```
procedure DEQUEUE (var Q: Queue);
begin
    if EMPTY (Q) then
    error (`Черга порожня`)
    else
    Q.head := Q.head^.next
end;
```

```
function HEAD (Q: Queue): elementtype;
begin
    if EMPTY (Q) then
    error (`Черга порожня`)
    else
    HEAD := Q.head^.next^.element
end;
```

```
procedure ENQUEUE (x: elementtype; var Q: Queue);
begin
    new(Q.tail^.next);
    Q.tail := Q.tail^.next;
    Q.tail^.element := x;
    Q.tail^.next := nil
end;
```

```
function EMPTY (Q: Queue): boolean;
begin
        if Q.head = Q.tail then
                EMPTY := true
        else    EMPTY := false
end;
```

Реалізація черг за допомогою циклічних масивів.

Можлива реалізація черги на основі циклічного масиву. Представимо масив у вигляді циклічної структури, де перший елемент масиву розташований за останнім (рис.6).

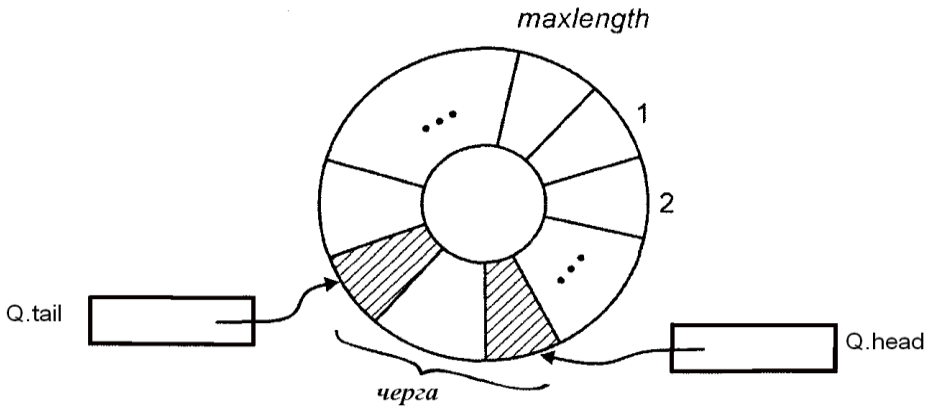


Рис. 6. Реалізація черги на основі циклічного масиву

Кінець черги знаходиться на певній відстані від її початку за годинниковою стрілкою. Для включення нового елемента в чергу досить перемістити *Q.tail* (покажчик на кінець черги) на одну позицію за годинниковою стрілкою і записати елемент у цю позицію. При виключенні елемента з черги досить просто перемістити *Q.head* (покажчик на початок черги) на одну позицію за годинниковою стрілкою.

Треба відзначити, що при такій реалізації час виконання операцій включення і виключення елементів не залежить від довжини черги.

Наведемо п’ять команд, що виконуються над чергами, які використовують описану реалізацію. Формально черги визначаються наступним чином:

```
Type
    Queue = Record
        elements : array [1..maxlength] of elementtype;
        head, tail : integer
    end;
```

Відповідні процедури наведені в лістингу. Функція addone (*i*) додає одиницю до позиції *i* в «циклічному» розумінні.

```
function addone (i: integer): integer;
begin
    addone := (i mod maxlength) + 1
end;
```

```
procedure MAKENULL (var Q : Queue);
begin
    Q.head := 1;Q.tail := maxlength
end;
```

```
function EMPTY (var Q: Queue): boolean;
begin    if addone (Q.tail) = Q.head
         then EMPTY := true
    else EMPTY := false
end;
```

```
function HEAD (var Q: Queue): elementtype;
begin    if EMPTY (Q) then
         error (`Черга порожня`)
else HEAD := Q.elements[Q.head]
end;
```

```
procedure ENQUEUE (x: elementtype; var Q: Queue);
begin
    if addone(addone(Q.tail)) = Q.head
    then error (`Черга повна`)
else begin Q.tail := addone(Q.tail);
           Q.elements[Q.tail] := x;
     end; end;
```

```
procedure DEQUEUE (var Q: Queue);
begin
    if EMPTY (Q) then
    error (`Черга порожня`)
    else Q.head:= addone(Q.head)
end;
```

Завдання.

Написати програму: а) реалізації черги за допомогою покажчиків; б) реалізації черги за допомогою циклічного масиву. Виконати: додавання елемента в чергу; видалення елемента з черги; перевірка, чи порожня черга; перегляд елемента в голові черги; очищення черги.

Контрольні запитання.

1. Що таке структура даних? Що таке черга?

2. Чим відрізняються статичні, напівстатичні та динамічні структури даних?
3. У чому полягає схожість та відмінність структур даних список, стек та черга?
4. Чи можна дістатися середини або початку черги, обійшовши її кінець (хвіст)?
5. Наведіть приклади з життя, де зустрічається «принцип черги».
6. У чому переваги реалізації черги за допомогою циклічного масиву від реалізації черги за допомогою звичайного масиву?

Лабораторна робота №3. Бінарні дерева.

Мета роботи: Вивчити способи ефективного зберігання та обробки інформації на прикладі бінарних дерев.

Теоретичні відомості.

Дерево – це сукупність елементів, що називаються вузлами (один з яких корінь), та відношень („батьківських”), що утворюють ієрархічну структуру вузлів. Вузли можуть бути елементами будь-якого типу (літерами, рядками, числами). До основних операцій з деревами належать: внесення елемента в дерево, обхід дерев та вилучення елемента з дерева. Під *обходом бінарного дерева* розуміють визначений порядок проходження усіх вершин дерева. Розрізняють декілька способів обходу дерев: прямий, зворотній та симетричний порядки обходу.

1. *Прямий порядок обходу* бінарного дерева можна визначити в такий спосіб (рис. 7): потрапити в корінь R; пройти в прямому порядку ліве піддерево A; пройти в прямому порядку праве піддерево B.

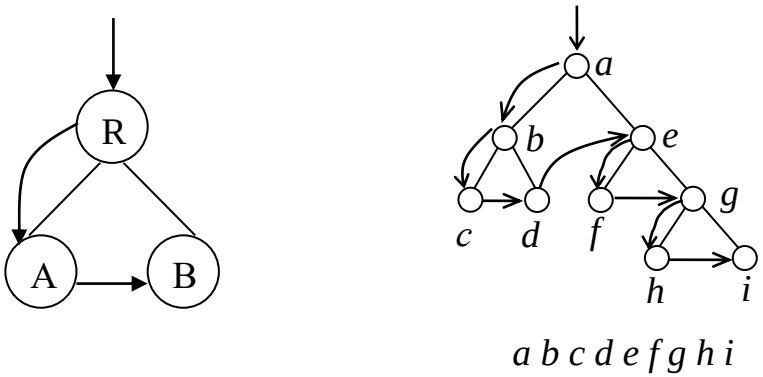


Рис. 7. Прямий порядок обходу бінарного дерева

2. Обхід бінарного дерева в *зворотному порядку* можна визначити в аналогічній формі (рис. 8): пройти в зворотному порядку ліве піддерево A; пройти в зворотному порядку праве піддерево B; потрапити в корінь R.

3. Визначимо ще один порядок обходу бінарного дерева, називаний *симетричним* (рис. 9): пройти в симетричному порядку ліве піддерево A; потрапити в корінь R; пройти в симетричному порядку праве піддерево B.

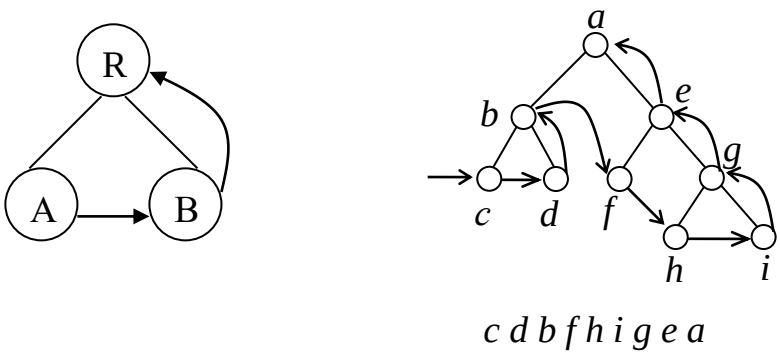


Рис. 8. Зворотний порядок обходу бінарного дерева

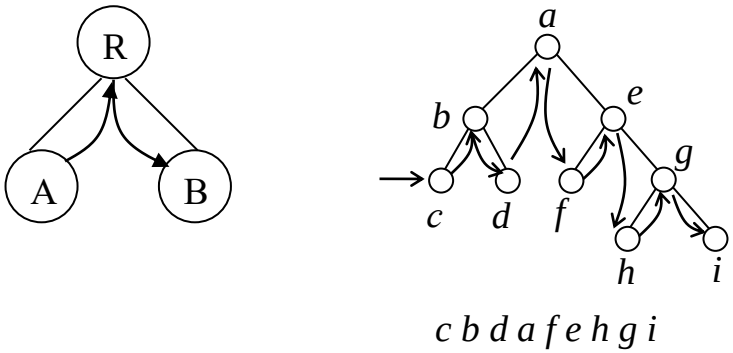


Рис. 9. Симетричний порядок обходу бінарного дерева

Розглянемо реалізацію дерева за допомогою покажчиків. Об’явимо дерево як:

```
Type
  TreeLink = ^Tree;
  Tree = Record
      Data : <тип даних>;
      Left, Right : TreeLink;
  End;
```

Корінь дерева опишемо в розділі опису змінних:

```
Var   kd : TreeLink;
```

Опишемо процедуру включення в дерево нового вузла. При включенні в дерево вузол вставляється або як піддерево вже існуючого вузла або як єдиний вузол дерева. Тому й лівий і правий зв'язки нового вузла повинні бути Nil. Коли дерево порожнє, значення передане у вигляді параметра покажчика дорівнює Nil. У цьому випадку потрібно змінити його так, щоб він вказував на новий вузол, що був включений як кореневий. При включенні наступного елемента переданий з основної програми параметр *t* уже не буде дорівнює nil, і треба приймати рішення щодо того, в яке піддерево необхідно включити новий вузол.

```
Procedure InsTree(n : integer; var t : TreeLink);
 Begin
  if t = nil     then
     begin new(t);
       with t^ do
         begin
           Left := nil;
           Right := nil;
           Data := n;
         end;
     end;
    else
     if n<=t^.data   then
        InsTree(n, t^.Left)
       else
        InsTree(n, t^.Right)
 End;
```

Опишемо процедуру виводу значень елементів бінарного дерева на екран. Для цього необхідно виконати повний обхід дерева. При обході дерева його окремі вузли відвідуються в окремому порядку: прямому, зворотному або симетричному. Процедура виводу дерева при зворотному обході має такий вигляд:

```
Procedure PrintTree(t : TreeLink);
 Begin
  if t<>nil
```

```
    then
      begin
        PrintTree(t^.Left);
        Write(t^.Data:3);
        PrintTree(t^.Right);
      end;
  End;
```

Основна програма здійснює введення чисел з клавіатури. Використаються: змінна nd типу TreeLink - значення покажчика на корінь дерева; змінна Digit типу integer для зберігання чергового введеного числа.

```
Begin
 writeln('Введіть вузли дерева. Закінчення введення – 0');
  kd := nil;
  read(Digit);
  while Digit <> 0 do
    begin
      InsTree(Digit, kd);
      writeln(' Введіть чергове число ');
      read(Digit);
    end;  PrintTree(kd);
End.
```

Завдання.

1. Створити й вивести на екран дерево, елементи якого вводяться із клавіатури й мають цілий тип. Причому для кожного вузла дерева у всіх лівих вузлах повинні перебувати числа менші, а в правих більші, ніж числа, що зберігаються в цьому вузлі. Таке дерево називається деревом пошуку.
2. Здійснити обхід дерева в прямому, зворотньому та симетричному порядках.
3. Написати функцію, яка знаходить найбільший та найменший елементи дерева.
4. Написати процедуру, яка видаляє з дерева всі парні елементи.

5. Написати процедуру, яка визначає число входжень заданого елемента в дерево.

6. Написати функцію, яка підраховує суму всіх елементів дерева.

Контрольні запитання.

1. Дати визначення: дерево, бінарне дерево, корінь та листи дерева, впорядковані дерева, дерево пошуку, висота та глибина дерева.

2. Схеми обходу дерев.

3. Яким є принцип динамічної побудови дерева?

4. У чому полягає схожість та відмінність таких структур даних як зв'язний список, стек, дерево?

Лабораторна робота №4.
Реалізація алгоритмів Прима та Крускала.

Мета роботи: Дослідити задачу знаходження остовного підграфа мінімальної ваги, використовуючи алгоритм Прима та алгоритм Крускала .

Теоретичні відомості.

Алгоритм Прима.

Для заданого графа *G* (рис.10) побудуємо остовне дерево мінімальної ваги, використовуючи алгоритм Прима.

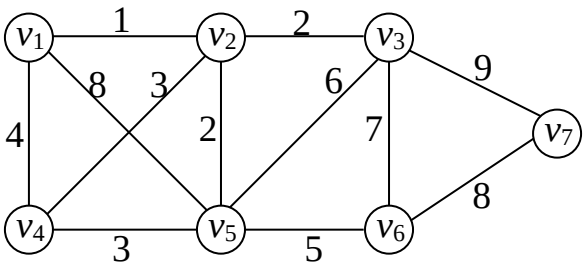


Рис. 10. Неорієнтований граф *G*

Побудуємо матрицю суміжності ваг:

	v_1	v_2	v_3	v_4	v_5	v_6	v_7
v_1	∞	1	∞	4	8	∞	∞
v_2	1	∞	2	3	2	∞	∞
v_3	∞	2	∞	∞	6	7	9
v_4	4	3	∞	∞	3	∞	∞
v_5	8	2	6	3	∞	5	∞
v_6	∞	∞	7	∞	5	∞	8
v_7	∞	∞	9	∞	∞	8	∞

За початкову обираємо довільну вершину, нехай це буде вершина v_1, і для неї шукаємо найближчого сусіда, тобто вершину відстань до якої найменша:

	v_2	v_3	v_4	v_5	v_6	v_7
v_1	1	∞	4	8	∞	∞

Це буде вершина v_2. Отримали перший фрагмент (рис.11 а). Вершина v_2 вилучається з рядка і переноситься у стовпчик.

Розширюємо фрагмент, переглядаючи всі можливі відстані від вершин, що знаходяться у стовпчику до ще неприєднаних вершин, які знаходяться в рядочку:

	v_3	v_4	v_5	v_6	v_7
v_1	∞	4	8	∞	∞
v_2	2	3	2	∞	∞

Існує дві вершини відстань до яких найменша і однакова. Це вершини v_3 та v_5. Обираємо будь-яку з них, нехай це буде вершина v_3. Отримуємо новий фрагмент (рис. 11 б).

Розширюємо фрагмент:

	v_4	v_5	v_6	v_7
v_1	4	8	∞	∞
v_2	3	2	∞	∞
v_3	∞	6	7	9

Найменша відстань від вершини v_2 до вершини v_5, тому фрагмент розширюємо на вершину v_5 (рис. 11 в).

Розширюємо фрагмент:

	v_4	v_6	v_7
v_1	4	∞	∞
v_2	3	∞	∞
v_3	∞	7	9
v_5	3	5	∞

Існує дві вершини відстань до яких найменша і однакова. Це вершини v_2 та v_5. Обираємо будь-яку з них, нехай це буде вершина v_2. Отримуємо новий фрагмент (рис. 11 г).

Розширюємо фрагмент:

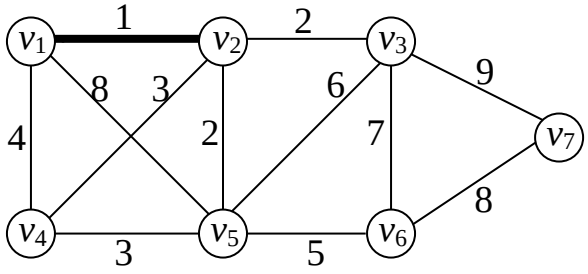
	v_6	v_7
v_1	∞	∞
v_2	∞	∞
v_3	7	9
v_4	∞	∞
v_5	5	∞

Найменша відстань від вершини v_5 до вершини v_6, тому фрагмент розширюємо на вершину v_6 (рис. 11 д).

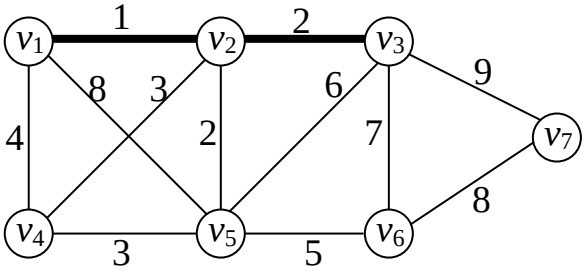
Розширюємо фрагмент:

	v_7
v_1	∞
v_2	∞
v_3	9
v_4	∞
v_5	∞
v_6	8

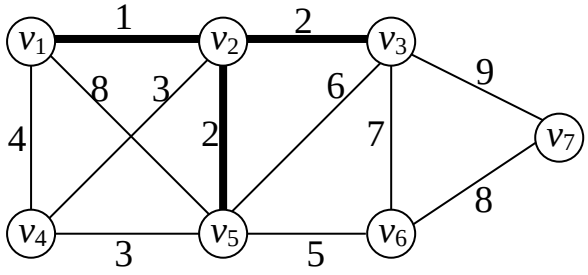
Остання неприєднана вершина v_7 приєднується до вершини v_6. Ортимуємо мінімальне остовне дерево T (рис. 11 е) з вагою w = 1+2+2+3+5+8 = 21.



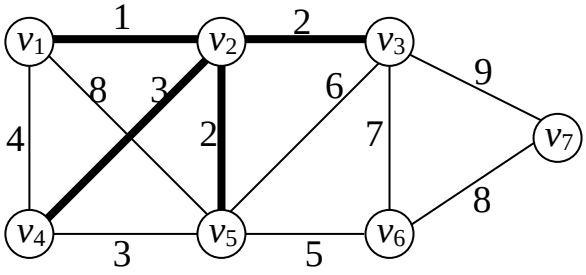
а)



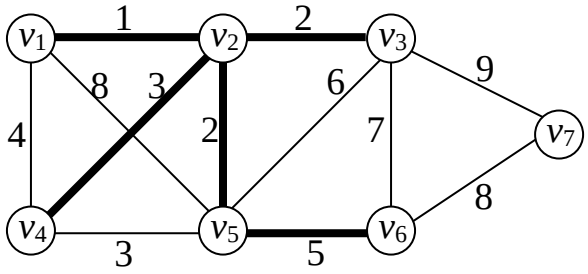
б)



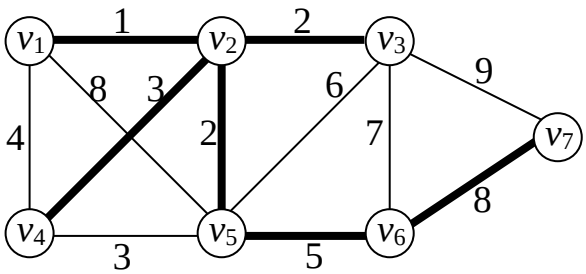
в)



г)



д)



е)

Рис. 11. Реалізація алгоритму Прима

Алгоритм Крускала.

Для заданого графа G (рис. 10) побудуємо остовне дерево мінімальної ваги, використовуючи алгоритм Крускала.

Спочатку впорядковуємо ребра (табл.1, стовпчик 1). Виконуємо розбиття множини вершин $\rho_0=\{\{v_1\},\{v_2\},\{v_3\},\{v_4\},\{v_5\},\{v_6\},\{v_7\}\}$. Переглядаємо кожне ребро в порядку зростання, якщо інцидентні йому вершини знаходяться в різних множинах розбиття, ребро додаємо до остовного дерева мінімальної ваги, а множини, в яких знаходяться ці вершини, об'єднуємо в одну. Процес вибору ребер наведено у табл.1.

Таблиця 1

Ребра впорядковані за зростанням	Розбиття множини вершин	Вибір ребра y мінімальний остов T
$e_1(v_1, v_2) = 1$	$\rho_1=\{\{v_1,v_2\},\{v_3\},\{v_4\},\{v_5\},\{v_6\}, \{v_7\}\}$	$E_T=\{ e_1 \}$
$e_2(v_2, v_3) = 2$	$\rho_2=\{\{v_1,v_2,v_3\},\{v_4\},\{v_5\},\{v_6\}, \{v_7\}\}$	$E_T=\{ e_1, e_2\}$
$e_3(v_5, v_6) = 2$	$\rho_3=\{\{v_1,v_2,v_3\},\{v_4\},\{v_5,v_6\}, \{v_7\}\}$	$E_T=\{ e_1, e_2, e_3\}$
$e_4(v_2, v_4) = 3$	$\rho_4=\{\{v_1,v_2,v_3,v_4\},\{v_5,v_6\}, \{v_7\}\}$	$E_T=\{ e_1, e_2, e_3, e_4\}$
$e_5(v_4, v_5) = 3$	$\rho_5=\{\{v_1,v_2,v_3,v_4,v_5,v_6\}, \{v_7\}\}$	$E_T=\{ e_1, e_2, e_3, e_4, e_5\}$
$e_6(v_1, v_4) = 4$	ρ_5	$e_6 \notin E_T$
$e_7(v_2, v_5) = 5$	ρ_5	$e_7 \notin E_T$
$e_8(v_3, v_5) = 6$	ρ_5	$e_8 \notin E_T$
$e_9(v_3, v_6) = 7$	ρ_5	$e_9 \notin E_T$
$e_{10}(v_1, v_5) = 8$	ρ_5	$e_{10} \notin E_T$
$e_{11}(v_6, v_7) = 8$	$\rho_6=\{\{v_1,v_2,v_3,v_4,v_5,v_6,v_7\}\}$	$E_T=\{ e_1, e_2, e_3, e_4, e_5, e_{11}\}$
$e_{12}(v_3, v_7) = 9$	-	-

При приєднанні ребра e_{11} робота алгоритму закінчується, так як вже приєднано $n-1= 7-1=6$ ребер і всі підмножини розбиття об'єдналися в одну $\rho_6=\{\{v_1,v_2,v_3,v_4,v_5,v_6,v_7\}\}$. Остовне дерево мінімальної ваги T утворюють ребра e_1, e_2, e_3, e_4, e_5, e_{11} (рис. 12).

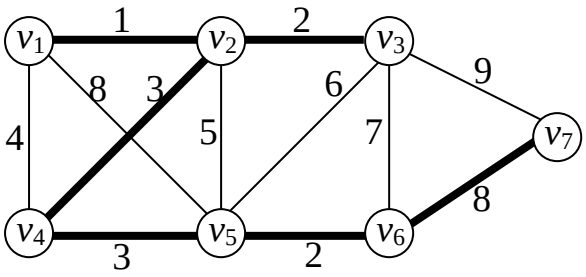


Рис. 12. Алгоритм Крускала

Завдання.

1. Навести нерієнтований граф G(V,E), V=8, E=15.
2. Для заданого графа побудувати остовне дерево мінімальної вартості:

а) за допомогою алгоритму Прима;

б) за допомогою алгоритму Крускала.

3. Написати програми, що реалізують ці алгоритми.
4. Написати процедуру обчислення вартості побудованого остовного дерева.

Контрольні запитання.

1. Дайте визначення: граф, орієнтований граф, степінь вершини, простий, псевдо, дводольний, ізоморфний, граф, мультиграф, остовний підграф, зважений граф.
2. Способи задання графів.
3. Постановка задачі пошуку остовного підграфу мінімальної ваги.
4. Чи будуть коректно працювати алгоритми Крускала та Прима з графом, у якого є ребра з від'ємною вагою?
5. Дайте визначення: фрагмент, ізольований фрагмент, ізольована вершина.
6. В чому полягає ідея алгоритмів Крускала та Прима?

Лабораторна робота №5
Реалізація алгоритмів Дейкстри та Флойда.

Мета роботи: Дослідити задачу знаходження найкоротших шляхів:
а) від джерела до всіх інших вершин заданого графа (алгоритм Дейкстри);
б) між усіма парами вершин заданого графа (алгоритм Флойда).

Теоретичні відомості.

Алгоритм Дейкстри.

Для заданого графа *G* (рис. 13) знайти найкоротші шляхи від джерела v_1 до всіх інших вершин, використовуючи алгоритм Дейкстри.

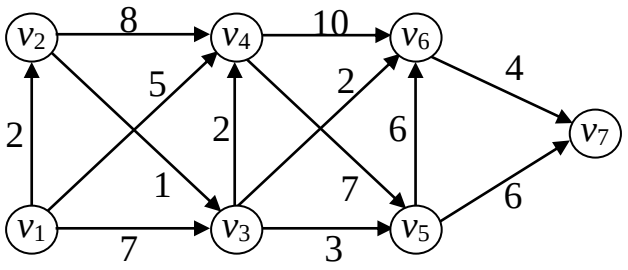


Рис. 13. Орієнтований граф *G*

Крок 1. Призначимо всім вершинам, окрім початкової мітки $(\infty,-)$, а початковій $v_1(0,-)$ (перше значення – це відстань від початкової вершини, друге – попередня вершина). Задамо дві множини: множину постійних міток *П* та множину тимчасових міток *Т*. $П_0 = \{v_1(0,-)\}$, $Т_0=\varnothing$.

Крок 2. Переглядаємо всі вершини, в які є шлях з початкової, змінюємо значення їх міток і заносимо їх в множину тимчасових міток $T_1 = \{v_2(2,v_1), v_3(7,v_1), v_4(5,v_1)\}$.

З множини T_1 обираємо вершину, в яку з v_1 найменша відстань і переносимо її з T_1 в множину $П_1$: $П_1 = \{v_1(\infty,-), v_2(2,v_1)\}$.

Крок 3. Від всіх вершин з множини *П* переглядаємо шляхи до суміжних з ними вершин, змінюємо значення їх міток і заносимо їх в множину тимчасових міток, якщо в множині *Т* вже є така вершина, обираємо значення з меншою відстанню

$T_2 = \{v_3(7, v_1), v_3(3, v_2), v_4(10, v_2), v_4(5, v_1)\} \Rightarrow T_2 = \{v_3(3, v_2), v_4(5, v_1)\}$.

$П_2 = \{v_1(\infty, -), v_2(2, v_1), v_3(3, v_2)\}$.

Крок 4. Перевіряємо, чи всі вершини мають постійні мітки, тобто знаходяться у множині $П$. Якщо так, то найкоротші шляхи від джерела v_1 до всіх інших вершин знайдено, якщо ні переходимо на крок 3.

$T_3 = \{v_4(5, v_1), v_6(5, v_3), v_5(6, v_3)\}$, $П_3 = \{v_1(\infty, -), v_2(2, v_1), v_3(3, v_2), v_4(5, v_1)\}$.

$T_4 = \{v_6(5, v_3), v_5(6, v_3)\}$, $П_4 = \{v_1(\infty, -), v_2(2, v_1), v_3(3, v_2), v_4(5, v_1), v_6(5, v_3)\}$.

$T_5 = \{v_5(6, v_3), v_7(9, v_6)\}$, $П_5 = \{v_1(\infty, -), v_2(2, v_1), v_3(3, v_2), v_4(5, v_1), v_6(5, v_3), v_5(6, v_3)\}$.

$T_6 = \{v_7(9, v_6)\}$, $П_6 = \{v_1(\infty, -), v_2(2, v_1), v_3(3, v_2), v_4(5, v_1), v_6(5, v_3), v_5(6, v_3), v_7(9, v_6)\}$.

Отже, щоб потрапити з вершини v_1 у вершину v_7 треба пройти найменшу відстань 9: $v_1 \xrightarrow{2} v_2 \xrightarrow{1} v_3 \xrightarrow{2} v_6 \xrightarrow{4} v_7$.

Алгоритм Флойда.

Для заданого графа G (рис. 14) знайти найкоротші шляхи між усіма парами вершин, використовуючи алгоритм Флойда.

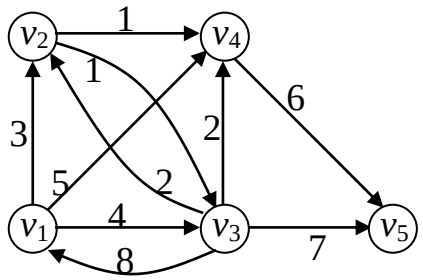


Рис. 14. Орієнтований граф G

Матриця ваг

	v_1	v_2	v_3	v_4	v_5
v_1	∞	3	4	5	∞
v_2	∞	∞	1	1	∞
v_3	8	2	∞	2	7
v_4	∞	∞	∞	∞	6
v_5	∞	∞	∞	∞	∞

$k = 1$

Матриця маршруів

	v_1	v_2	v_3	v_4	v_5
v_1	1	1	1	1	1
v_2	2	2	2	2	2
v_3	3	3	3	3	3
v_4	4	4	4	4	4
v_5	5	5	5	5	5

$$\begin{pmatrix} \infty & 3 & 4 & 5 & \infty \\ \infty & \infty & 1 & 1 & \infty \\ 8 & 2 & 2 & 2 & 7 \\ \infty & \infty & \infty & \infty & 6 \\ \infty & \infty & \infty & \infty & \infty \end{pmatrix} \quad k=2 \qquad \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 1 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 \\ 5 & 5 & 5 & 5 & 5 \end{pmatrix}$$

$$\begin{pmatrix} \infty & 3 & 4 & 5 & \infty \\ \infty & \infty & 1 & 1 & \infty \\ 8 & 2 & 3 & 2 & \infty \\ \infty & \infty & \infty & \infty & 6 \\ \infty & \infty & \infty & \infty & \infty \end{pmatrix} \quad k=3 \qquad \begin{pmatrix} 1 & 1 & 1 & 2 & 1 \\ 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 2 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 \\ 5 & 5 & 5 & 5 & 5 \end{pmatrix}$$

$$\begin{pmatrix} 12 & 3 & 4 & 4 & \infty \\ 9 & 3 & 1 & 1 & \infty \\ 8 & 2 & 3 & 2 & \infty \\ \infty & \infty & \infty & \infty & 6 \\ \infty & \infty & \infty & \infty & \infty \end{pmatrix} \quad k=4 \qquad \begin{pmatrix} 3 & 1 & 1 & 2 & 1 \\ 3 & 3 & 2 & 2 & 2 \\ 3 & 3 & 2 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 \\ 5 & 5 & 5 & 5 & 5 \end{pmatrix}$$

$$\begin{pmatrix} 12 & 3 & 4 & 4 & 10 \\ 9 & 3 & 1 & 1 & 7 \\ 8 & 2 & 3 & 2 & 8 \\ \infty & \infty & \infty & \infty & 6 \\ \infty & \infty & \infty & \infty & \infty \end{pmatrix} \quad k=5 \qquad \begin{pmatrix} 3 & 1 & 1 & 2 & 4 \\ 3 & 3 & 2 & 2 & 4 \\ 3 & 3 & 2 & 3 & 4 \\ 4 & 4 & 4 & 4 & 4 \\ 5 & 5 & 5 & 5 & 5 \end{pmatrix}$$

Відповідь: Матриця ваг Матриця маршруів

$$\begin{pmatrix} 12 & 3 & 4 & 4 & 10 \\ 9 & 3 & 1 & 1 & 7 \\ 8 & 2 & 3 & 2 & 8 \\ \infty & \infty & \infty & \infty & 6 \\ \infty & \infty & \infty & \infty & \infty \end{pmatrix} \qquad \begin{pmatrix} 3 & 1 & 1 & 2 & 4 \\ 3 & 3 & 2 & 2 & 4 \\ 3 & 3 & 2 & 3 & 4 \\ 4 & 4 & 4 & 4 & 4 \\ 5 & 5 & 5 & 5 & 5 \end{pmatrix}$$

Завдання.

1. Навести орієнтований граф G(V,E), V=8, E=15.
2. Для заданого графа знайти найкоротший шлях від джерела до всіх інших вершин заданого графа, використовуючи алгоритм Дейкстри.

3. Для заданого графа знайти найкоротші шляхи між усіма парами вершин за допомогою алгоритму Флойда.

4. Написати програму реалізації алгоритмів Дейкстри та Флойда.

Контрольні запитання.

1. Чи будуть коректно працювати алгоритми Дейкстри та Флойда з графом, у якого є ребра з від'ємною вагою?

2. В чому полягає ідея алгоритмів Дейкстри та Флойда?

Лабораторна робота № 6. Внутрішнє сортування.

Мета роботи: Дослідити сортування масиву різними методами (простого обміну, вставками, вибором, Шелла, Хоара), оцінити їх ефективність та порівняти між собою.

Теоретичні відомості.

Метод простого обміну.

Переглядається весь масив "знизу вверх " і міняються місцями поряд розташовані елементи, якщо нижній елемент менше за верхній. Таким чином виштовхується наверх самий легкий елемент масиву. Потім повторюється ця операцію для n -1 елементів, що залишилися.

Відсортувати масив методом бульбашки: 4, 9, 7, 6, 2, 3.

4	4	4	4	2
9	9	9	2	4
7	7	2	9	9
6	2	7	7	7
2	6	6	6	6
3	3	3	3	3

Нульовий прохід

4	2	2	2	2	2
9	4	3	3	3	3
7	9	4	4	4	4
6	7	9	9	6	6
2	6	7	7	9	7
3	3	6	6	7	9
i=0	i=1	i=2	i=3	i=4	i=5

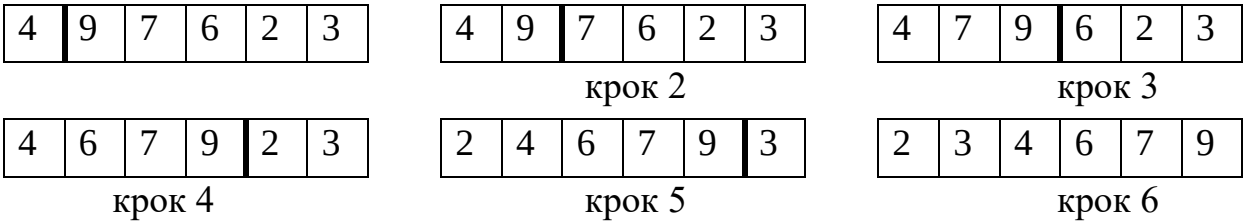
i – номер проходу

Сортування вставками.

При сортуванні вставками спочатку впорядковуються два перші елементи масиву. Потім робиться вставка третього елемента у відповідне місце по відношенню до перших двох елементів. На i-му кроці послідовність розділена на дві частини: впорядковану $a[0]...a[i]$ і невпорядковану $a[i+1]...a[n]$ на наступному (i+1)-му кроці алгоритму беремо $a[i+1]$ і встановлюємо в потрібне місце у впорядковану частину.

Пошук відповідного місця для наступного елемента здійснюється шляхом послідовних порівнянь з елементами, що розташовані перед ним. В залежності від результату порівняння елементи або залишаються на поточних місцях (вставку завершено), або міняються місцями і процес продовжується.

Відсортувати масив 4, 9, 7, 6, 2, 3 вставками.

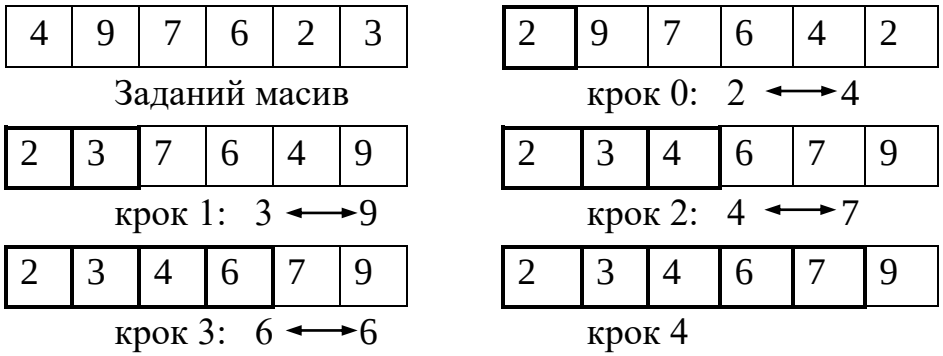


Так як масив з одного елемента можна вважати відсортовані, то алгоритм починається з k =2.

Сортування вибором.

Послідовність будується, починаючи з лівого кінця масиву. Алгоритм складається з n послідовних кроків від 0 до n-1. На i-му кроці обирається найменший з елементів $a[i]....a[n]$ і міняється місцем з $a[i]$.

Відсортувати масив методом вибору.

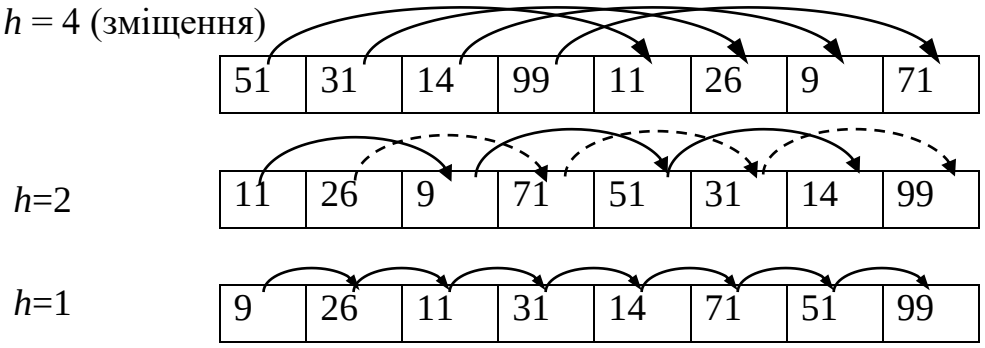


Таким чином на (n-1)-му кроці вся послідовність, крім $a[n]$ буде відсортованою , а $a[n]$ стоїть на останньому місці: всі менші елементи зліва від нього.

Сортування методом Шелла.

Спочатку прибирається «масовий безлад» в масиві, порівнюючи далеко розташовані один від одного елементи, поступово зменшуючи інтервал між ними до одиниці. Це означає, що на останніх стадіях алгоритм зводиться до перестановки сусідніх елементів.

Найчастіше використовується послідовність зміщень 8, 4, 2, 1 (для цього кількість елементів = 2^n), але застосовувати можна будь-яку послідовність (наприклад 7, 5, 3 ,1), головне щоб останнє зміщення було 1. Відсортувати масив 51, 31, 14, 99, 11, 26, 9, 71.



Масив після застосування алгоритму Шелла.

9	11	26	14	31	51	71	99

Перестановка сусідніх елементів.

9	11	14	26	31	51	71	99

Сортування методом Хоара (швидке сортування).

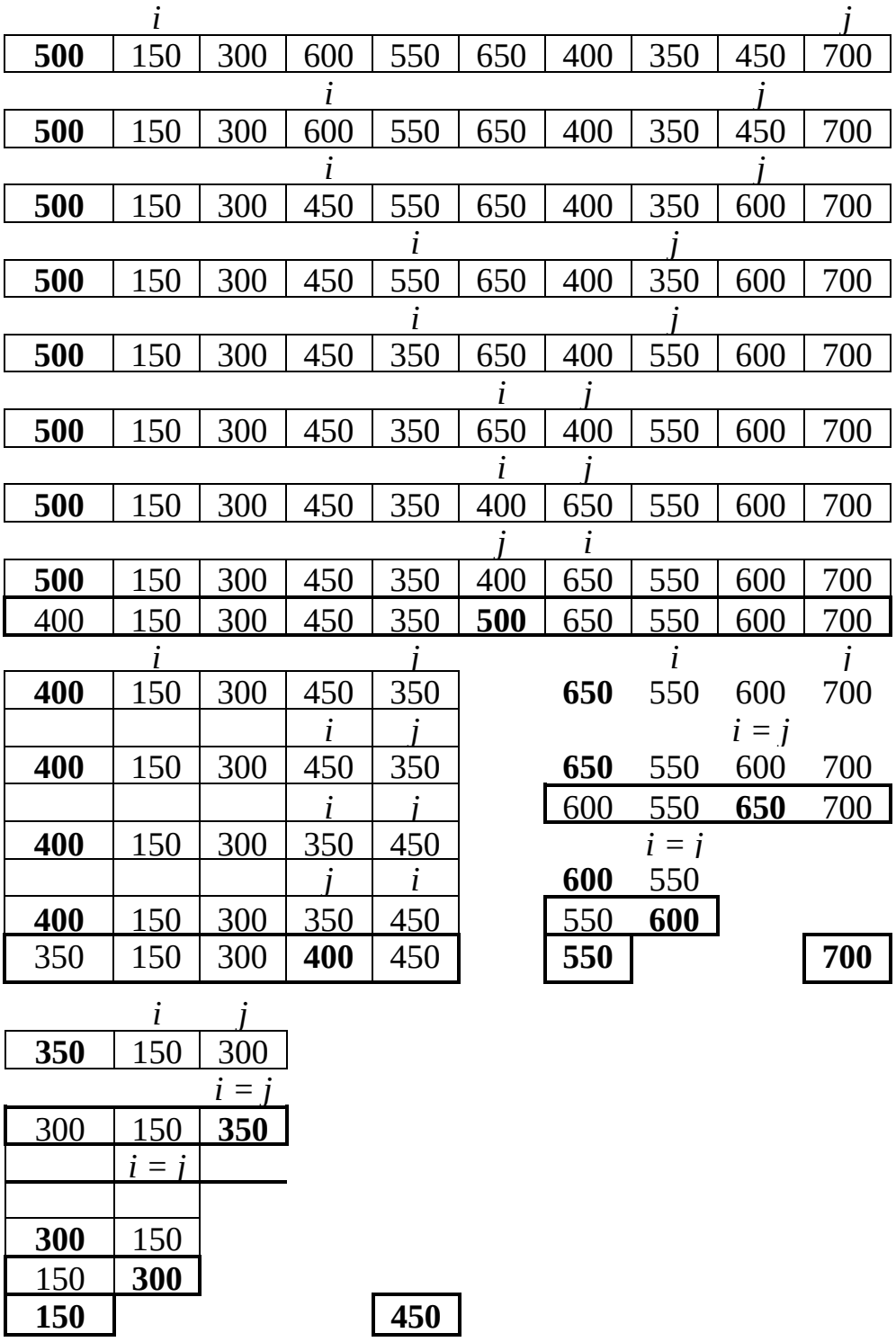
Швидке сортування – це алгоритм, заснований на методі декомпозиції. Швидке сортування розподіляє елементи масиву у відповідності з їх значеннями, тобто виконується перестановка елементів заданого масиву A[0..*n*] для отримання розбиття, при якому всі елементи до деякої позиції *s* не перевищують елемент A[*s*], а елементи після позиції *s* не менше A[*s*]:

$$\underbrace{A[0]...A[s-1]}_{\text{Всі елементи} \le A[s]} \qquad A[s] \qquad \underbrace{A[s+1]...A[n]}_{\text{Всі елементи} \ge A[s]}$$

Після розбиття елемент A[*s*] знаходиться на своїй кінцевій позиції, і необхідно сортувати два підмасиви до і після A[*s*] незалежно.

Розбиття масиву A[0..*n*] і його підмасиву (A[*l*..*r*]) ($0 \le l < r \le n$)можна виковувати наступним чином. Спочатку обирається елемент відносно якого буде виконуватися розбиття. Цей елемент називається *опорним*. Є декілька стратегій для вибору опорного елементу, наприклад за опорний елемент можна взяти перший елемент підмасиву, середній або останній. Розглянемо найпростішу стратегію і в якості опорного оберемо перший елемент підмасиву: p = A[*l*].

Задано масив із 10 цілих чисел: A=500, 150, 300, 600, 550, 650, 400, 350, 450, 700. Треба відсортувати його в порядку зростання.



Завдання.

Скласти схеми алгоритму для кожного методу і написати програми реалізації цих методів (по варіантах):

а) сортування бульбашкою;

б) сортування вставками;

в) сортування вибором;

г) швидке сортування Хоара;

д) сортування методом Шелла.

Контрольні запитання.

1. В чому полягає задача сортування даних?
2. Від чого залежить ефективність алгоритмів сортування?
3. Які параметри впливають на оцінку алгоритмів сортування?
4. В чому полягає ідея методів сортування: бульбашкою, вставкою, вибором, Шелла та Хоара?
5. Порівняти час виконання методів сортування: бульбашкою, вставкою, вибором, Шелла та Хоара.
6. Дати оцінку алгоритмам сортування внутрішнього сортування.

Лабораторна робота № 7. Зовнішнє сортування.

Мета роботи: Навчитись сортувати файли даних, вміст яких перевищує об'єм оперативної пам'яті.

Теоретичні відомості.

Сортування прямим злиттям.

Відсортувати файл A (8, 23, 5, 65, 44, 33, 1, 6), використовуючи зовнішнє сортування прямим злиттям. Наведемо розв'язок у таблиці:

Початковий стан файлу A	8, 23, 5,65, 44, 33, 1, 6
Крок 1. Розподіл Файл B Файл C Файл A	 8 5 44 1 (a_1, a_3, a_5, a_7) 23 65 33 6 (a_2, a_4, a_6, a_8) $\underbrace{8\ 23}_{a'_1}$ $\underbrace{5\ 65}_{a'_2}$ $\underbrace{33\ 44}_{a'_3}$ $\underbrace{1\ 6}_{a'_4}$ (a_1, a_2) (a_3, a_4) (a_5, a_6) (a_7, a_8)
Крок 2. Розподіл Файл B Файл C	 8 23 33 44 (a'_1, a'_3) 5 65 1 6 (a'_2, a'_4)

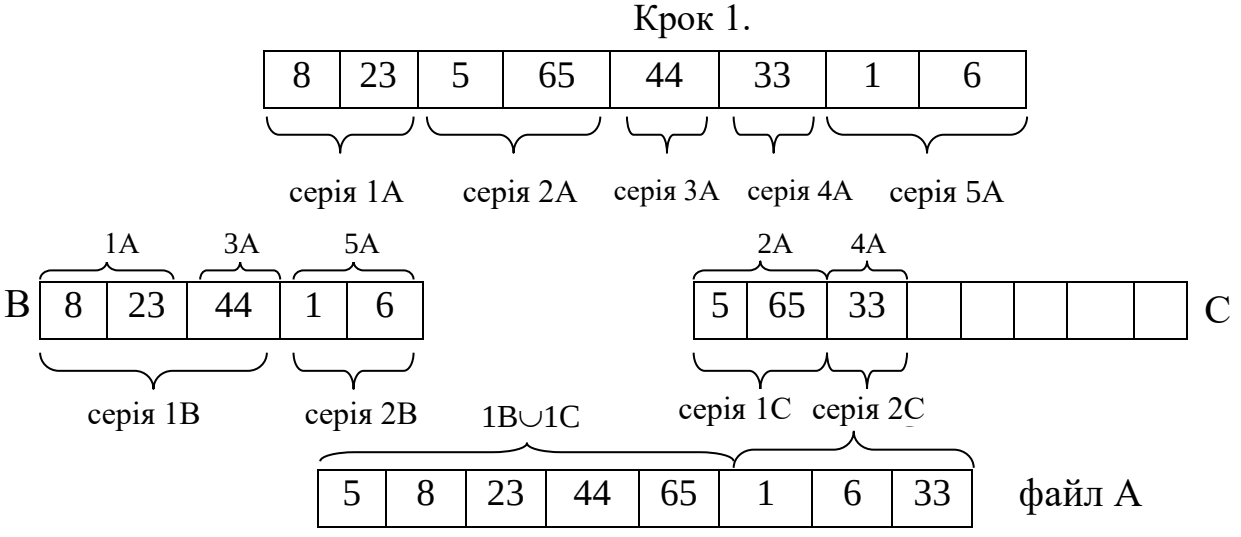
Файл А	$\underbrace{5\ \ 8\ 23\ 65}_{a''_1(a'_1,\ a'_2)}$ $\underbrace{1\ 6\ 33\ \ 44}_{a''_2(a'_3,\ a'_4)}$
Крок 3. Розподіл Файл В Файл С Файл А	 5 8 23 65 (a''_1) 1 6 33 44 (a''_2) 1 5 68 23 33 44 65

Для виконання зовнішнього сортування методом прямого злиття в основній пам’яті необхідно розташувати лише дві змінні – для розміщення записів з файлів В і С.

Природне злиття.

Сортування виконується за декілька кроків, в кожному з яких спочатку виконується розподілення файла А по файлах В і С, а потім злиття В і С в файл А. При розподілі розпізнається перша серія записів і переписується в файл В, друга – в файл С і т. д. При злитті перша серія записів файла В зливається з першою серією файлу С, друга серія В з другою серією С і т. д. Якщо перегляд одного файлу закінчується раніше ніж перегляд другого (з причини різної кількості серій), то залишок недопереглянутого файлу ціяко переноситься в кінець файлу А. Процес завершується коли в файлі А залишається лише одна серія (рис. 15).

Оскільки довжина серій може бути довільною, то максимальний розмір файлів В і С може бути близьким до розміру файла А.



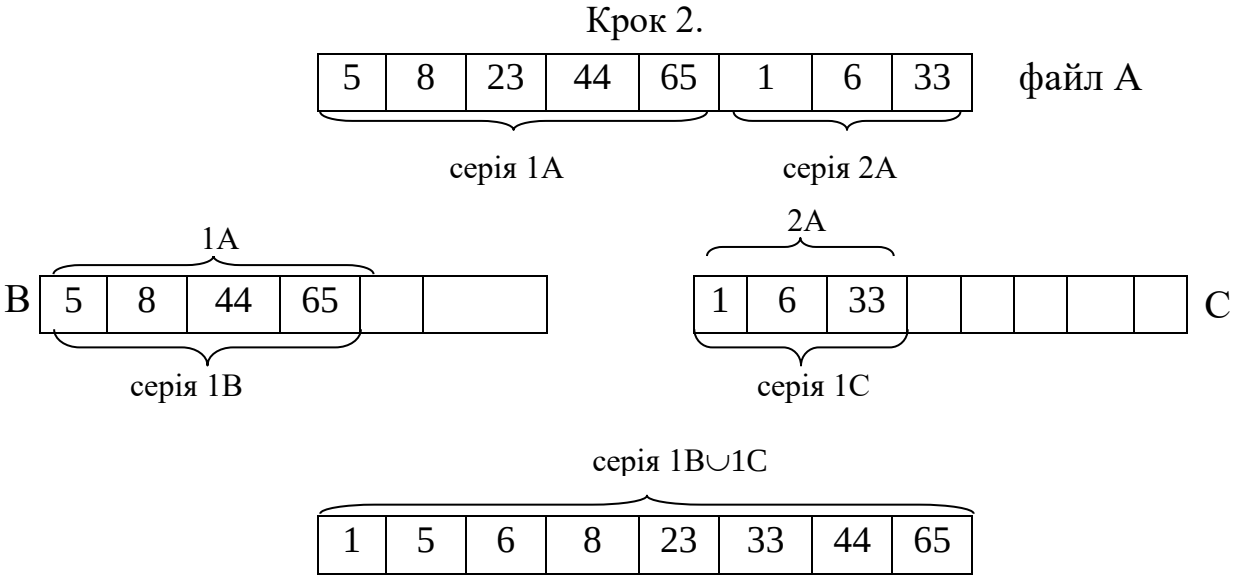


Рис. 15. Природне злиття

Збалансоване багатоканальне злиття.

В основі методу зовнішнього сортування збалансованим багатоканальним злиттям є розподіл серій вхідного файлу по m допоміжних файлів C_1, C_2, ..., Cm і т. д., доки в B_1 або C_1 не утвориться одна серія. Відсортувати файл, використовуючи трьохканальне злиття (рис. 16).

Як видно з рис. 16, по мірі, збільшення довжини серій допоміжні файли, з великими номерами (починаючи з номера *n*) перестають використовуватися, оскільки їм „не дістається” жодної серії.

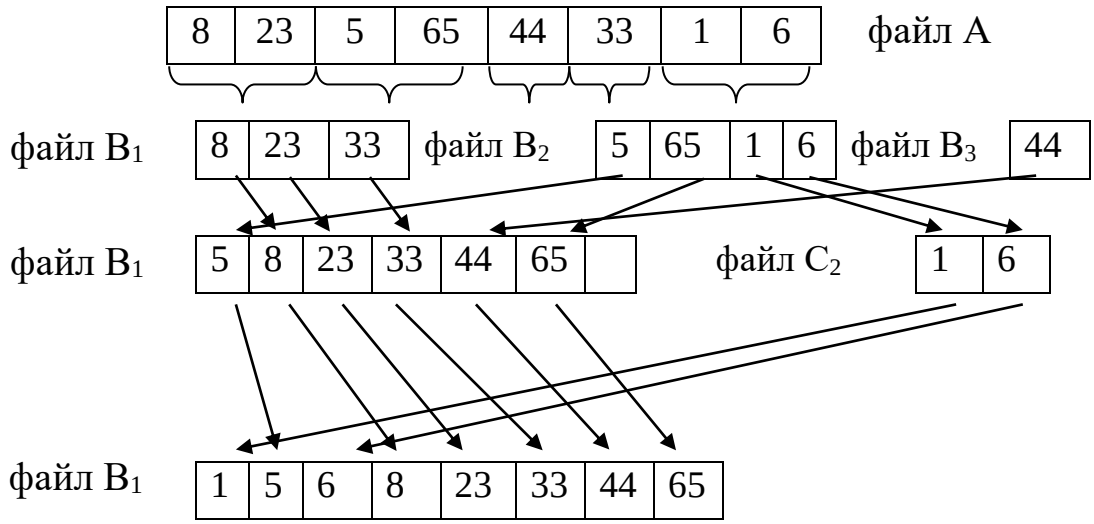


Рис. 16. Трьохканальне злиття

Багатофазне сортування.

Ідея багатофазного сортування полягає в тому, що з m допоміжних файлів (m-1) файл використовується для вводу послідовностей, що зливаються, а один для виводу утворених серій. Як тільки один з файлів вводу стає порожнім, його починають використовувати для виводу серій, отриманих при злитті серій нового набору (m-1) файлів. Таким чином, маємо перший крок, при якому серії початкового файлу А розподіляються по m-1 допоміжному файлу, а потім виконується багатофазне злиття серій з (m-1) файлів, доки в одному з них не утвориться одна серія.

Відсортувати файл, використовуючи трьохфазне злиття.

Крок	B_1	B_2	B_3
n	1	0	0
n-1	0	1	1
n-2	1	2	0
n-3	3	0	2
n-4	0	3	5
n-5	5	8	0
n-6	13	0	8
...	...	...	...
1	...	...	...

Цей приклад показує, що метод трьохфазного зовнішнього сортування дає бажаний результат і працює максимально ефективно (на кожному етапі відбувається злиття максимальної кількості серій), якщо початковий розподіл серій між допоміжними файлами описується сусідніми числами Фібоначчі.

Завдання.

Скласти схеми алгоритмів і написати програми реалізації алгоритмів зовнішнього сортування (по варіантах):

а) природне злиття;

б) пряме злиття;

в) багатоканальна злиття;

г) багатофазне злиття.

Контрольні питання.

1. Де використовується зовнішнє сортування?
2. Від чого залежить швидкість виконання зовнішнього сортування?
3. В чому полягає ідея прямого, природного, багатоканального та багатофазного злиття?
4. Яким повинен бути початковий розподіл серій між допоміжними файлами у багатофазному злитті, щоб алгоритм працював найефективніше?

Лабораторна робота № 8. Алгоритми пошуку.

Мета роботи: Навчитись реалізовувати різні алгоритми пошуку.

Теоретичні відомості.

Алгоритм Рабіна-Карпа.

Ідея, запропонована Рабіном і Карпом, полягає в тому, щоб поставити у відповідність кожному рядку деяке унікальне число, і замість того, щоб порівнювати самі рядки, порівнювати числа, що набагато швидше.

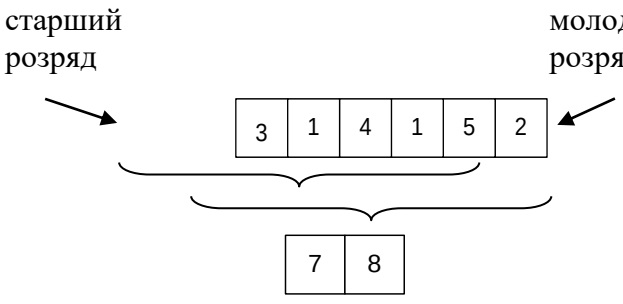
Задано рядок *T=abcdeabfgfcakmbddaf* і підрядок, який треба знайти *P=bfgfc* (*m*=5). {0,1…9} – алфавіт, *d*=10, $h=d^{m-1}=10^4=10000$; *q*=13.

b	f	g	f	c
3	1	4	1	5

P=31415.

P_q =(5+10(1+10(4+10(1+10·3)))) mod 13=31415 mod 13=7

Необхідно шукати підрядки, які дорівнюють 7 по модулю 13.



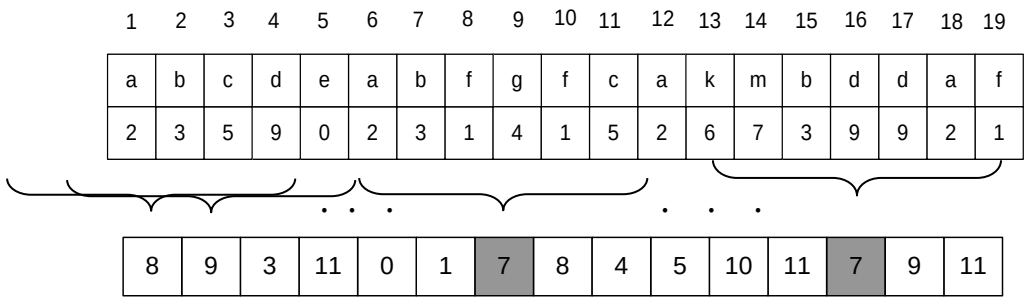
T_6=31415 , T_6 mod 13=7.

Обчислюємо T_7:

T_7=10(31415 –3·10000)+2=14152

T_7 mod 13=8

Обчислимо значення T_S для всього тексту:



Procedure RK (T,P,d,q)

Var …

Begin n:=length (T); //довжина тексту

m:= length (P); //довжина зразка

P:=0; //числовий покажчик зразка P

T_s :=0; //змінний числовий покажчик підрядка тексту T

h:=1:

for i:=1 **to** m-1 **do** h:=d*h mod q; //$h=d^{m-1}$ mod q

for i:=1 **to** m **do**

begin

P:=(d*P+ord(P[i])) mod q;

T_s:=(d*T_s +ord(T[i])) mod q;

end;

for s:=0 **to** n-m **do**

begin

if P=T_s **then** //якщо співпадання за модулем

begin

k:=1; //перевірка тексту зі зразком

while (k≤m) and (P[k]=T[s+k]) **do** inc(k);

if k>m **then** //якщо співпали P і T_s, то виводимо зсув

writeln (s);

end;

//обчислюємо наступне T_s

T_s:=(d*(T_s– ord T[s+1]*h)+ ord T[s+m+1]) mod q;

end

End.

Алгоритм Кнута-Морриса Пратта (КМП).

Алгоритм КМП складається з двох етапів: підготовчого (побудова префікс-функції) і основного (пошуку).

Підготовчий етап.

Для довільного слова *X* розглянемо всі його префікси, що одночасно є суфіксами і серед них оберемо найдовший (не враховуючи самого *X*). Його будемо позначати *n*(*X*) і називати найбільшим префікс-суфіксом.

Знайти повну префіксну функцію *f* для зразка *P=abababababca.*

n(*a*)=*L*, *f*(1)=0
n(*ab*)=*L*, *f*(2)=0
n(*aba*)=*a*, *f*(3)=1
n(*abab*)=*ab*, *f*(4)=2
n(*ababa*)=*aba*, *f*(5)=3
n(*abababab*)=*abab*, *f*(6)=4
n(*abababa*)=*ababa*, *f*(7)=5
n(*ababababab*)=*abababab*, *f*(8)=6
n(*abababababc*)=*L*, *f*(9)=0
n(*abababababca*)=*a*, *f*(10)=1

або

i	1	2	3	4	5	6	7	8	9	10
P[*i*]	a	b	a	b	a	b	a	b	c	a
f[*i*]	0	0	1	2	3	4	5	6	0	1

Процедура для обчислення f(*k*):

```
Procedure Prefix;
Begin
        f[1]:=0;    // префікс рядка з одного символу має нульову довжину
        m=length(P);
        k=0;
    for i:=2  to m do // обчислюється для префіксів рядків довжиною від 2 до m символів
            begin
    while (k>0) and (P[k+1]<>P[i]) do k:= f[k]; //значення функції може бути отримано з раніше зроблених обчислень
    if P[k+1]=P[i] then k:=k+1;
    f[i]:=k;                //присвоювання префікс - функції
            end
End;
```

Етап пошуку

```
Procedure KMP;
Begin
        Prefix;
        k:=0;                  // кількість співпавших символів
    for i:=1 to n do           //сканування тексту зліва направо
    begin
    while (k>0) and P[k +1]<>T[i] do k:=f[k]; //наступний символ не співпадає
    if P[k +1]=T[i] then k:= k +1;  // наступний символ співпадає
    if k =m then                    //співпали всі символи рядка P?
        begin writeln (“ зразок знайдено зі зсувом; i-m);
        k:=f(k);                    //пошук наступного співпадання
        end;        end;
End.
```

Завдання.

Скласти схеми алгоритмів та написати програми реалізації алгоритмів Рабіна-Карпа та КМП. Провести порівняльний аналіз алгоритмів пошуку в рядках.

Контрольні запитання.

1. Ідея алгоритмів Рабіна-Карпа та КМП.
2. З яких етапів складається алгоритм КМП? Охарактеризувати кожний етап.
3. Що таке префіксна функція?

Список літератури:

1. *Ахо Альфред В., Хопкрофт Джон, Ульман Джефри Д.* Структуры данных и алгоритмы: Пер. с англ. – М.: Издательский дом «Вильямс», 2003. – 384 с.

2. *Ахо Альфред В., Хопкрофт Джон, Ульман Джефри Д.* Построение и анализ вычислительных алгоритмов. – М.: Мир, 1979.

3. *Вирт Н.* Алгоритмы и структуры данных: Пер. с англ. – 2-е изд., испр. – СПб.: Невский Диалект, 2001. – 352 с.

4. *Иванов Б.Н.* Дискретная математика. Алгоритмы и программы. Учебное пособие. – М.: Лаборатория Базовых Знаний, 2003. – 288 с.

5. *Кнут Д.* Искусство программирования для ЭВМ т.1. Основные алгоритмы: Пер. с англ.: – М.: Мир, 1976. – 734 с.

6. *Кнут Д.* Искусство программирования для ЭВМ т.2. Получисленные алгоритмы: Пер. с англ.: – М.: Мир, 1976. – 723 с.

7. *Кнут Д.* Искусство программирования для ЭВМ т.3. Сортировка и поиск: Пер. с англ.: – М.: Мир, 1976. – 843 с.

8. *Кормен Т., Лейзерсон Ч., Ривест Р.* Алгоритмы: построение и анализ. – М.: Центр непрер. матем. образ-я, 2000. – 960 с.

9. *Макконнелл Дж.* Анализ алгоритмов. Вводный курс. – М.: Техносфера, 2002. – 304 с.

10. *Міхайленко В.М., Федоренко Н.Д., Демченко В.В..* Дискретна математика: Підручник. – Київ, Європейський університет, 2003. – 320 с.

11. *Новиков Ф. А..* Дискретная математика для программистов: Учебник для вузов. 2-е изд. – СПБ.: Питер, 2004. – 364 с.

Додаток 1

Титульний лист до звіту з лабораторної роботи

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Київський національний університет будівництва і архітектури

Кафедра прикладної математики

Лабораторна робота №
з дисципліни “Алгоритми та структури даних”
Тема “(назва теми)”

Виконав: П.І.Б. студента
Спеціальність ІУСТ(ІТЕП)-(номер групи)
Перевірив: П.І.Б. викладача

(Місце для підпису і дати)

Київ – (поточний рік)

Навчально-методичне видання

АЛГОРИТМИ І СТРУКТУРИ ДАНИХ

Методичні вказівки
до виконання лабораторних робіт
для студентів спеціальності
122 Комп’ютерні науки

Укладачі: **Білощицька** Світлана Василівна
Федоренко Наталія Дмитрівна
Баліна Олена Іванівна
Безклубенко Ирина Сергіївна
Доля Олена Вікторівна
Білощицький Андрей Олександрович

Комп’ютерна верстка

Підписано до друку 2020 Формат 60x84 1/16.
Папір офсетний. Гарнітура Таймс. Друк на різографі.
Ум. друк. арк. 2,56. Обл.-вид. арк. 2,75.
Ум. фарбовідб. 23 Тираж 50 прим. Вид. № 35/III-06. Зам. №

КНУБА, Повітрофлотський проспект, 31, Київ, Україна, 03680

E-mail: red-isdat@knuba.edu.ua

Віддруковано в редакційно-видавничому відділі
Київського національного університету будівництва і архітектури

Свідоцтво про внесення до Державного реєстру суб’єктів видавничої справи
ДК № 808 від 13.02.2002 р.