­­МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

­­­

Лабораторна робота №1-6

з курсу «Теорія Алгоритмів»

для студентів базового напрямку 6.08.04 "Комп’ютерні науки"

(заочна форма навчання)

Варіант 14

Виконав студент гр. КНз-2

Чалий Михайло

­­

Львів 2014

# Лабораторна №1

## Теоретичні відомості

Однією із фундаментальних статей, результати якої лежать в основі сучасної теорії алгоритмів, є стаття польсько-американського математика і логіка Еміля Леона Поста (1897-1954), «Finite combinatory processes - Formulation 1» [1], опубліковану у 1936 році. У ній він запропонував абстрактну обчислювальну конструкцію, яка дозволила вперше уточнити поняття алгоритму і яку згодом назвали машиною Поста. При розробці обчислювальної конструкції Пост керувався принципом створення максимально простої абстракції: мінімум операцій під час обробки інформації, вхідна інформація повинна бути закодованою з використанням мінімального набору символів.

Не дивлячись на примітивність машини Поста, будь-який існуючий алгоритм може бути записаний у вигляді програми для машини Поста. У теорії алгоритмів існує так звана «теза Поста»: «Будь-який алгоритм можна представити у вигляді машини Поста». Ця теза одночасно є і формальним визначенням алгоритму. Алгоритм (за Постом) - програма для машини Поста, що приводить до вирішення поставленої задачі.

Теза Поста є гіпотезою. Її неможливо строго довести (так само, як і теза Тюрінга), тому що у ній фігурують, з одного боку, інтуїтивне поняття   
«будь-який алгоритм», а з іншого боку - точне поняття «машина Поста». Для того, щоб спростувати гіпотезу Поста, необхідно придумати алгоритм, який неможливо записати у вигляді програми для машини Поста. На сьогоднішній день такого алгоритму не існує.

Машина Поста — це абстрактна (тобто така, що не існує в арсеналі техніки), але дуже проста обчислювальна машина. Машина Поста, не дивлячись на зовнішню простоту, може здійснювати різні обчислення, для чого потрібно задати початковий стан каретки і програму, яка виконає ці обчислення. Машиною ця математична конструкція названа тому, що при її побудові використовуються деякі поняття реальних машин (елемент пам’яті, команда тощо). Машину Поста можна розглядати як спрощену модель комп’ютера. Насправді, як комп’ютер, так і машина Поста мають:

* неподільні носії інформації (комірки — біти), які можуть бути заповненими або незаповненими;
* обмежений набір елементарних дій — команд, кожна з яких виконується за один такт (крок).

Обидві машини працюють на основі програми. Проте у машині Поста інформація розташовується лінійно і читається підряд, а в комп’ютері можна читати інформацію за адресою; набір команд комп’ютера значно ширший і виразніший за команди машини Поста.

## Завдання

Задано масив міток. Каретка оглядає 1-шу порожню комірку перед початком масиву. Розсунути масив таким чином, щоб після кожної мітки була порожня комірка.

## Рішення

1. → 2

2. ? 1 : 3

3. → 4

4. X 5

5. → 6

6. ? 7 : 5

7. V 8

8. ← 9

9. ? 10 : 11

10. !

11. ← 12

12. ? 13 : 11

13. → 14

14. → 4

## Результат

Start:Tape [>0<1111]

1 -> Tape [0>1<111]

2 -> Tape [0>1<111]

3 -> Tape [01>1<11]

4 -> Tape [01>0<11]

5 -> Tape [010>1<1]

6 -> Tape [010>1<1]

5 -> Tape [0101>1<]

6 -> Tape [0101>1<]

5 -> Tape [01011>0<]

6 -> Tape [01011>0<]

7 -> Tape [01011>1<]

8 -> Tape [0101>1<1]

9 -> Tape [0101>1<1]

11 -> Tape [010>1<11]

12 -> Tape [010>1<11]

11 -> Tape [01>0<111]

12 -> Tape [01>0<111]

13 -> Tape [010>1<11]

14 -> Tape [0101>1<1]

4 -> Tape [0101>0<1]

5 -> Tape [01010>1<]

6 -> Tape [01010>1<]

5 -> Tape [010101>0<]

6 -> Tape [010101>0<]

7 -> Tape [010101>1<]

8 -> Tape [01010>1<1]

9 -> Tape [01010>1<1]

11 -> Tape [0101>0<11]

12 -> Tape [0101>0<11]

13 -> Tape [01010>1<1]

14 -> Tape [010101>1<]

4 -> Tape [010101>0<]

5 -> Tape [0101010>0<]

6 -> Tape [0101010>0<]

7 -> Tape [0101010>1<]

8 -> Tape [010101>0<1]

9 -> Tape [010101>0<1]

End:Tape [010101>0<1]

# Лабораторна №2

## Теоретичні відомості

Машина Тюрінґа є розширенням моделі кінцевого автомата, що включає потенційно нескінченну пам’ять з можливістю переходу (рух) від поточної в даний момент комірки до її лівого або правого сусіда.

Алан Тьюрінґ висловив припущення, що будь-який алгоритм в інтуїтивному значенні цього слова може бути представлений еквівалентною машиною Тюрінґа. Це припущення відоме як теза Чорча–Тюрінґа. Кожний комп’ютер може моделювати машину Тюрінґа (операції перезапису комірок, порівняння і переходу до іншої сусідньої комірки з урахуванням зміни стану машини). Отже, він може моделювати алгоритми в будь-якому формалізмі, і всі комп’ютери (незалежно від потужності, архітектури тощо) еквівалентні з погляду принципової можливості рішення алгоритмічних задач.

## Завдання

## На стрічці машини Тьюрінга знаходиться ціле додатне число, записане в десятковій системі числення. Знайти добуток цього числа на число 11. Каретка знаходиться над крайньою правою цифрою числа.

## Рішення

<movetoend, "0", →, "0", movetoend>

<movetoend, "1", →, "1", movetoend>

<movetoend, "2", →, "2", movetoend>

<movetoend, "3", →, "3", movetoend>

<movetoend, "4", →, "4", movetoend>

<movetoend, "5", →, "5", movetoend>

<movetoend, "6", →, "6", movetoend>

<movetoend, "7", →, "7", movetoend>

<movetoend, "8", →, "8", movetoend>

<movetoend, "9", →, "9", movetoend>

<movetoend, "λ", -, "0", moveprev>

<moveprev, "λ", -, "λ", exit>

<moveprev, "0", ←, "0", getprev>

<moveprev, "1", ←, "1", getprev>

<moveprev, "2", ←, "3", getprev>

<moveprev, "3", ←, "4", getprev>

<moveprev, "4", ←, "4", getprev>

<moveprev, "5", ←, "5", getprev>

<moveprev, "6", ←, "6", getprev>

<moveprev, "7", ←, "7", getprev>

<moveprev, "8", ←, "8", getprev>

<moveprev, "9", ←, "9", getprev>

<getprev, "0", →, "0", add0>

<getprev, "1", →, "1", add1>

<getprev, "2", →, "2", add2>

<getprev, "3", →, "3", add3>

<getprev, "4", →, "4", add4>

<getprev, "5", →, "5", add5>

<getprev, "6", →, "6", add6>

<getprev, "7", →, "7", add7>

<getprev, "8", →, "8", add8>

<getprev, "9", →, "9", add9>

<add1, "λ", ←, "1", moveprev>

<add1, "0", ←, "1", moveprev>

<add1, "1", ←, "2", moveprev>

<add1, "2", ←, "3", moveprev>

<add1, "3", ←, "4", moveprev>

<add1, "4", ←, "5", moveprev>

<add1, "5", ←, "6", moveprev>

<add1, "6", ←, "7", moveprev>

<add1, "7", ←, "8", moveprev>

<add1, "8", ←, "9", moveprev>

<add1, "9", ←, "0", moveprev>

<add4, "λ", ←, "4", moveprev>

<add4, "0", ←, "4", moveprev>

<add4, "1", ←, "5", moveprev>

<add4, "2", ←, "6", moveprev>

<add4, "3", ←, "7", moveprev>

<add4, "4", ←, "8", moveprev>

<add4, "5", ←, "9", moveprev>

<add4, "6", ←, "0", moveprev>

<add4, "7", ←, "1", moveprev>

<add4, "8", ←, "2", moveprev>

<add4, "9", ←, "3", moveprev>

## Результат

State movetoend; Tape at 1 of [None, '1', '4']

State movetoend; Tape at 2 of [None, '1', '4']

State movetoend; Tape at 3 of [None, '1', '4', None]

State moveprev; Tape at 3 of [None, '1', '4', '0']

State getprev; Tape at 2 of [None, '1', '4', '0']

State add4; Tape at 3 of [None, '1', '4', '0']

State moveprev; Tape at 2 of [None, '1', '4', '4']

State getprev; Tape at 1 of [None, '1', '4', '4']

State add1; Tape at 2 of [None, '1', '4', '4']

State moveprev; Tape at 1 of [None, '1', '5', '4']

State getprev; Tape at 0 of [None, '1', '5', '4']

State getprev; Tape at 0 of [None, '1', '5', '4']

[Finished in 0.1s]

# Лабораторна №3

## Теоретичні відомості

Нормальним алгоритмом Маркова (НАМ) називається непорожній скінчений впорядкований набір операторів підстановки:



У цих операторах можуть використовуватися два види стрілок: звичайна стрілка (→) і стрілка «з хвостиком» (). Оператор із звичайною стрілкою називається звичайним оператором, а оператор із стрілкою «з хвостиком» − завершальний оператором. Записати алгоритм у вигляді НАМ − означає представити такий набір операторів.

## Завдання

## Задано алфавіт A={а,b,c}. Перетворити слово р так, щоб спочатку йшли всі символи а, потім − всі символи b і в кінці − всі символи с.

## Рішення

ca -> ac

ba -> ab

cb -> bc

## Результат

Start from: acbacccabbc

Apply ca -> ac: acbaccacbbc

Apply ca -> ac: acbacaccbbc

Apply ca -> ac: acbaacccbbc

Apply ba -> ab: acabacccbbc

Apply ca -> ac: aacbacccbbc

Apply ba -> ab: aacabcccbbc

Apply ca -> ac: aaacbcccbbc

Apply cb -> bc: aaabcccbcbc

Apply cb -> bc: aaabccbcbcc

Apply cb -> bc: aaabcbcbccc

Apply cb -> bc: aaabbcbcccc

Apply cb -> bc: aaabbbccccc

End with: aaabbbccccc

[Finished in 0.1s]

# Лабораторна №4

## Теоретичні відомості

Сортуванням називають впорядковування по ключах (тобто за якою-небудь ознакою) елементів деякої структури даних на якій визначено відношення порядку. У залежності від того, чи знаходяться елементи структур даних у внутрішній (оперативній) пам’яті або у зовнішній пам’яті (на зовнішніх пристроях), розрізняють внутрішнє та зовнішнє сортування.

## Завжання

Відсортувати 23, 55, 47, 35, 10, 90, 84, 30 за допомогою сортування вставкою, і сортуванням злиттям.

## Рішення

def insertion\_sort(data):

for i in range(1,len(data)):

j = i

while j > 0 and data[j] < data[j - 1]:

data[j],data[j - 1] = data[j - 1], data[j]

j -= 1

print(str(data[0:i]) + "|" + str(data[i:len(data)]))

return data

def merge\_sort(data):

print("Split " +str(data))

l = len(data)

if l == 1:

return data

data1 = merge\_sort(data[:l//2])

data2 = merge\_sort(data[l//2:])

sorted\_data = []

data1\_index = 0

data2\_index = 0

print("Merge: " + str(data1) + " and " + str(data2))

# Merge

while data1\_index < len(data1) and data2\_index < len(data2):

if data1[data1\_index] >= data2[data2\_index]:

sorted\_data.append(data2[data2\_index])

data2\_index += 1

else:

sorted\_data.append(data1[data1\_index])

data1\_index += 1

# Put rest of it

if data1\_index < len(data1):

sorted\_data.extend(data1[data1\_index:])

else:

sorted\_data.extend(data2[data2\_index:])

return sorted\_data

data = [23, 55, 47, 35, 10, 90, 84, 30]

print("Insertsion sort: " + str(data))

result = insertion\_sort(data)

print("Result: " + str(result))

print("Merge sort: " + str(data))

result = merge\_sort(data)

print("Result: " + str(result))

## Результат

Insertsion sort: [23, 55, 47, 35, 10, 90, 84, 30]

[23]|[55, 47, 35, 10, 90, 84, 30]

[23, 47]|[55, 35, 10, 90, 84, 30]

[23, 35, 47]|[55, 10, 90, 84, 30]

[10, 23, 35, 47]|[55, 90, 84, 30]

[10, 23, 35, 47, 55]|[90, 84, 30]

[10, 23, 35, 47, 55, 84]|[90, 30]

[10, 23, 30, 35, 47, 55, 84]|[90]

Result: [10, 23, 30, 35, 47, 55, 84, 90]

Merge sort: [23, 55, 47, 35, 10, 90, 84, 30]

Split [23, 55, 47, 35, 10, 90, 84, 30]

Split [23, 55, 47, 35]

Split [23, 55]

Split [23]

Split [55]

Merge: [23] and [55]

Split [47, 35]

Split [47]

Split [35]

Merge: [47] and [35]

Merge: [23, 55] and [35, 47]

Split [10, 90, 84, 30]

Split [10, 90]

Split [10]

Split [90]

Merge: [10] and [90]

Split [84, 30]

Split [84]

Split [30]

Merge: [84] and [30]

Merge: [10, 90] and [30, 84]

Merge: [23, 35, 47, 55] and [10, 30, 84, 90]

Result: [10, 23, 30, 35, 47, 55, 84, 90]

[Finished in 0.1s]

# Лабораторна №5

## Теоретичні відомості

## Зручним методом пошуку у таблицi, якщо вона не надто велика, є метод дiлення пополам. Як приклад розглядається пошук у таблицi, в якiй наведена кiлькiсть населення в окремих областях. Пошук здiйснюється за кодом ключа, що є скороченою назвою областi. Якщо зобразити скiнченну множину у виглядi таблицi, впорядкованої за зростанням ключiв, то з'явиться можливiсть здiйснювати пошук як у словнику.

У загальному випадку бiнарне дерево з довiльною кiлькiстю вузлiв n називається повнiстю збалансованим деревом, якщо кiлькiсть вузлiв у лiвому i правому пiддеревах вiдрiзняються не бiльше нiж на одиницю.

Кiлькiсть варiантiв структур бiнарних дерев можна приблизно оцiнити за допомогою формули Стiрлiнга:

**4n/n\*3/2**

Якщо кiлькiсть ключiв n=15, серед 9694845 бiнарних дерев лише одне буде повнiстю збалансованим.

Якщо кiлькiсть ключiв n=16, серед 35357670 бiнарних дерев лише вiсiм будуть повнiстю збалансованими. Це переконливо свiдчить про те, що повнiстю збалансованi дерева трапляються рiдко. Кiлькiсть порiвнянь до збiгу аргумента iз ключем вузла:

**Cn=1/n\*(log2(k+1))**

**1<k<n,**

де Cn - середня кiлькiсть порiвнянь при вдалому пошуку, k - вузол збiгу.

Середня кiлькiсть порiвнянь для невдалого пошуку виражається формулою:

**Cnn=[n/(n+1)]\*(Cn+1)+1**

Бiнарне дерево пошуку можна подати для даних, що вводяться у випадковiй послiдовностi:

а) у послiдовностi географiчного розмiщення iз заходу на схiд;

б) у послiдовностi зростання кiлькостi населення i тому подiбне.

Доведено, що кiлькiсть порiвнянь для випадкового дерева у середньому всього лише на 39% перевищує кiлькiсть порiвнянь для повнiстю збалансованого дерева.

Використовуючи відносно прості засоби і послабивши умови збалансованості, можна побудувати майже збалансоване дерево. Одне iз рішень проблеми запропонували Адельсон-Вельський і Ландис. Критерій такий:

|  |
| --- |
| Дерево є збалансованим тоді і тільки тоді |
| коли для кожного вузла висота двох піддерев відрізняється  не більше ніж на одиницю. |

Дерева, що задовольняють цю умову, називаються ***АВЛ-деревами***. Усі ідеально збалансовані дерева також є АВЛ-деревами. Визначення приводить до легко виконуваного збалансування, а середня довжина пошуку залишається практично такою ж, як в ідеальнольному збалансованому деревi.

## Завдання

Побудувати AVL-дерево з елементів 23, 55, 47, 35, 10, 90, 84, 30.

## Рішення

|  |  |
| --- | --- |
| C:\Users\Mike\AppData\Local\Microsoft\Windows\INetCache\Content.Word\l5_14_1.png | Вставка 47 призводить до розбалансування, 23 тепер має показник збалансованості +2  C:\Users\Mike\AppData\Local\Microsoft\Windows\INetCache\Content.Word\l5_14_2.png |
| Повертаємо 47 та 55 вправо  C:\Users\Mike\AppData\Local\Microsoft\Windows\INetCache\Content.Word\l5_14_3.png | Повертаємо 23, 47, 55 вліво  C:\Users\Mike\AppData\Local\Microsoft\Windows\INetCache\Content.Word\l5_14_4.png |
| Ставка 35  C:\Users\Mike\AppData\Local\Microsoft\Windows\INetCache\Content.Word\l5_14_5.png | Ставка 10  C:\Users\Mike\AppData\Local\Microsoft\Windows\INetCache\Content.Word\l5_14_6.png |
| C:\Users\Mike\AppData\Local\Microsoft\Windows\INetCache\Content.Word\l5_14_7.png | C:\Users\Mike\AppData\Local\Microsoft\Windows\INetCache\Content.Word\l5_14_8.png |
| C:\Users\Mike\AppData\Local\Microsoft\Windows\INetCache\Content.Word\l5_14_9.png |  |

# Лабораторна №6

## Теоретичні відомості

## 2-3-деревом називається дерево, у якому кожний вузол, що не є листом, має двох чи трьох нащадків, а довжини всіх шляхів із кореня в листки однакові. Дерево, що складається з 1 вузла, є 2-3-деревом.

## Лема.

## Нехай Т буде 2-3-деревом висоти h. Кiлькiсть вузлів дерева Т міститься між 2^(h+1)-1 i (3^(h+1))/2, а кiлькiсть листків між 2^h i 3^h.

## Лінійно впорядковану множину S можна зобразити 2-3-деревом, приписавши всі елементи листкам дерева.

## Елемент, приписаний листку l, позначається Е[l]. У кожному вузлі V, що не є листком, вимагаються два типи даних: L[V] i M[V]. L[V]- найбільший елемент множини S у піддереві, коренем якого служить найлівіший син вузла V; M[V] - найбільший елемент множини S у піддереві, коренем якого є другий син вузла V. Значення L i M, приписані вузлам, дають змогу шукати елемент, починаючи з кореня способом, аналогічним двійковому пошуку. Операцію НАЛЕЖАТИ можна виконати на множині з n елементів за час О (log n).

## Завдання

Побудувати 2:3-дерево з елементів 23, 55, 47, 35, 10, 90, 84, 30.

## Рішення

|  |  |
| --- | --- |
| C:\Users\Mike\AppData\Local\Microsoft\Windows\INetCache\Content.Word\l6_14_1.png | Вставка 47 розщеплює 23:55  C:\Users\Mike\AppData\Local\Microsoft\Windows\INetCache\Content.Word\l6_14_2.png |
| C:\Users\Mike\AppData\Local\Microsoft\Windows\INetCache\Content.Word\l6_14_3.png | C:\Users\Mike\AppData\Local\Microsoft\Windows\INetCache\Content.Word\l6_14_4.png |
| C:\Users\Mike\AppData\Local\Microsoft\Windows\INetCache\Content.Word\l6_14_5.png | |
| C:\Users\Mike\AppData\Local\Microsoft\Windows\INetCache\Content.Word\l6_14_6.png | |
| C:\Users\Mike\AppData\Local\Microsoft\Windows\INetCache\Content.Word\l6_14_7.png | |