УДК 004.519.712+004.92

В.М. ТЕРЕЩЕНКО, В.І. ПУЗИРЕЙ

ЗАДАЧА ДИНАМІЧНОЇ ЛОКАЛІЗАЦІЇ ТОЧКИ НА НЕЗВ'ЯЗНОМУ ГРАФІ

Анотація. *У статті запропоновано розв'язок задачі динамічної локалізації точки на незв'язному графі за час* $O(\log N)$ *з використанням* $O(N)$ *пам'яті. Розроблено структуру даних на основі червоно-чорного дерева, що підтримує операції вставки і вилучення ребер за час* $O(\log N)$*, а також введено порядок над відрізками в середині смуги і знаходження сусіднього ребра.*
Ключові слова: *локалізація точки, незв'язний граф, вставка ребра, вилучення ребра, смуга, структура даних.*

Аннотация. *В статье предложено решение задачи динамической локализации точки на несвязном графе за время* $O(\log N)$ *с использованием* $O(N)$ *памяти. Разработано структуру данных на основе красно-черного дерева, которая поддерживает операции вставки и удаления ребер за время* $O(\log N)$*, а также введен порядок над отрезками внутри полосы и поиск соседнего ребра.*
Ключевые слова: *локализация точки. несвязный граф, вставка ребра, удаление ребра, полоса, структура данных.*

Abstract. *In this paper we propose solving a problem of dynamic point localization on a disconnected graph during* $O(\log N)$ *time and using* $O(N)$ *memory. The data structure of the base of red-and-black tree supporting an edge insert/delete operations using* $O(\log N)$ *time was developed. Segments order within a slab and finding neighbour edge clockwise was established.*
Keywords: *point localization, disconnected graph, edge insert/delete, slab, data structure.*

1. Вступ

Постановка проблеми. В роботі розглядається один із підходів до розв'язання задачі динамічної локалізації точки (далі ДЛТ) у двовимірному випадку. Як і для будь-якої динамічної задачі обчислювальної геометрії, побудова ефективних алгоритмів розв'язання задачі ДЛТ, навіть для площини, є не простим завданням. А тому актуальним є пошук підходів, які б давали водночас ефективні і прості рішення задачі. Розв'язки задачі ДЛТ мають широке практичне застосування, зокрема, в ГІС-системах, графіці і базах даних [1]. Одним із прикладів застосування задачі є задача визначення місцезнаходження, де графом є сітка доріг, а як запитна точка – координати GPS [2]. Крім того, задача ДЛТ являється будівничим блоком для цілої низки задач в середині самої обчислювальної геометрії [3].

Аналіз останніх досліджень. Першим, хто взявся за розв'язання задачі ДЛТ в установлених обмеженнях, був Б. Чазеле. У 1983 році для розв'язання задачі він запропонував громіздку структуру даних, яку назвав hive graph [4]. Застосуванню традиційних статичних підходів, що задовольняли установленим межам (методи трапецій, ланцюгів і Кіркпатріка), заважало те, що вся логіка в них зосереджена в передобробці, що неприпустимо для динамічного випадку. У 1986 році Коул розробив метод, який зменшував об'єм використаної пам'яті на базі традиційного методу смуг Ліптона до $O(N)$ [5]. З'явилася ідея модифікувати алгоритм для динамічного випадку, і у 1986 році вийшла праця Н. Сарнака і Р. Тар'яна [6], в якій пропонувалось застосувати до методу Добкіна-Ліптона таку динамічну структуру даних, як червоно-чорне дерево. Саме втілення ідей Сарнака-Тар'яна з певними модифікаціями є метою статті.

Як альтернативу складному в реалізації червоно-чорному дереву у 2004 році запропоновано метод перепозначення Дітца-Слетора [7, 8]. Але цей метод має вади, хоча в середньому він має хороші показники, в найгіршому випадку він не працює за логарифмічний час [9]. Тому в основі запропонованого методу лежить використання червоно-чорного дерева. Сучасні публікації пропонують як альтернативу методи, що засновуються на теорії ймовірності [1, 10]. Детально вивчається вузьке місце динамічної локалізації точки – проблема вводу-виводу [11, 12]. Крім того, стверджується, що при застосуванні суперкомп'ютерів можна подолати давнє обмеження на час запиту в $O(\log N)$ [13, 7].

Новизна та ідея. У роботі запропоновано ефективну модифікацію методу Сарнака-Тар'яна для випадку незв'язного графа на основі нової структури даних (модифікованого червоно-чорного дерева), що підтримує операції вставки і вилучення за час $O(\log M)$.

2. Постановка та побудова розв'язку задачі

Традиційна постановка задачі локалізації точки має такий вигляд.

Загальна постановка задачі. Нехай задані N – вершинне розбиття простору $G(V, E)$ і точка Z. Визначити область розбиття, яка містить точку Z. Оскільки розглядається лише двовимірний випадок, розбиття матиме вигляд планарного графа [3]. Динамічність задачі полягає у можливості додавати і вилучати ребра графа без повної перебудови структури пошуку. Таким чином, для площини матимемо таке формулювання задачі.

Постановка задачі. Розробити ефективний алгоритм динамічної локалізації точки на площині та структуру даних, яка підтримує локалізацію точки з операцію вставки та вилучення за час $O(\log N)$, з використанням $O(N)$ пам'яті.

Розглянемо існуючі підходи до розв'язання задачі локалізації точки. Одним із найбільш відомих є метод смуг, запропонований Ліптоном ще в 1973 році [3]. Суть його полягає у тому, що ми розбиваємо нашу площину на смуги у відповідності з вершинами графа. Будемо вважати, що смуга визначається своєю лівою абсцисою. При виконанні запиту на пошук здійснюється бінарний пошук у списку смуг, утворених вертикалями, які проходять через кожну вершину графа і упорядкованих по x-координаті. В середині кожної смуги формується список відрізків, упорядкованих по y-координаті. Таким чином, ми отримуємо верхній та нижній відрізки, які локалізують точку Z. Звичайний метод смуг у найгіршому випадку вимагає $O(N^2)$ пам'яті, проте Коул [5] дослідив можливість зменшення цієї оцінки до $O(N)$. Так, він розглядає таку проблему: нехай дана послідовність k-списків, кожен з яких довжиною N, і при цьому сусідні списки відрізняються не більше, як на h елементів. Запропонувати таку структуру даних, яка б дозволяла знаходити список елементів між i-м та j-м списками за час $O(j-i+\log N)$ [14].

Будемо вважати, що i-й список ставиться у відповідність i-му рядку матриці. Кожен список розглядається як бінарне дерево, в якому ліві і праві підсписки відносно медіани розглядаються як піддерева, що зберігаються рекурсивно. Якщо послідовність сусідніх дерев має ідентичні ліві або праві піддерева, ми зберігаємо лише один екземпляр піддерева, і всі дерева ділять однакові піддерева. Така процедура, яка зберігає місце, повторюється рекурсивно, окремо для лівих і правих піддерев. Після цього ми все ще можемо здійснити бінарний пошук за час $O(\log N)$ [5]. Це дозволяє зменшити використання пам'яті до $O(N+h\log(N))$ [5]. Оскільки у методі смуг сусідні смуги відрізняються лише на одну точку, в нашому випадку $h=1$. Отже, після редукції Коула ми маємо структуру, яка займає $O(N)$. Тепер повернемося до застосування цієї редукції для задачі локалізації точки.

Введемо відношення порядку над вершинами графа. Нехай e і f вершини графа. Введемо порядок $e \prec f$ на множині вершин графа, якщо існує лінія $x = c$, яка перетинає e і f так, що перетин e відбувається на меншому значенні y. Якщо за вісь абсцис взяти час, тоді, застосовуючи вищеописану структуру даних, отримаємо розв'язок задачі статичної локалізації точки з оцінками $O(N, \log N)$.

2.1. Червоно-чорне дерево

Майже повна відсутність логічної частини, винесеної у попередню обробку, максимально спрощує динамізацію статичного методу Ліптона [3]. Все, що необхідно, – це створити дерева, по яких ведеться пошук, використовуючи динамічні структури даних. Як варіант можна розглядати 2–3-дерева або АВЛ-дерева, але більш традиційним підходом є застосування червоно-чорного дерева, приклад якого подано на рис. 1 [6].

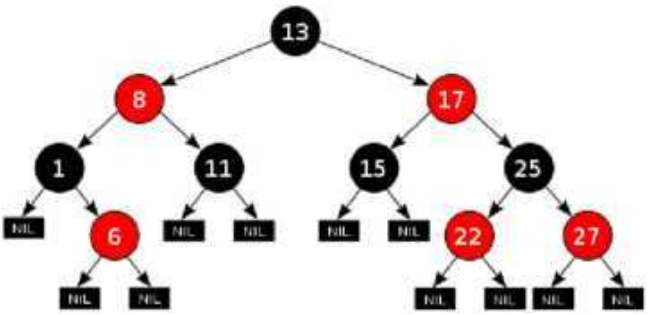

Рис. 1. Червоно-чорне дерево

Червоно-чорне дерево – це двійкове дерево пошуку, яке має такі властивості:

1. Кожна вершина має атрибут колір, який може бути або червоним, або чорним.
2. Кожен листок – NIL і чорний.
3. Якщо вершина червона, то обидва її сини чорні.
4. Всі шляхи, що йдуть вниз до листків, мають однакову кількість чорних вершин [15].

При додаванні і вилученні ці властивості можуть порушуватися. Для їх відновлення проводиться попередній аналіз із застосуванням зміщення, як показано на рис. 2.

Рис. 2. Зміщення в динамічному дереві

Застосовуючи це зміщення, при кожному додаванні або вилученні ми здійснюємо рух по дереву, виконуючи процедуру балансування, яка відновлює втрачені властивості червоно-чорного дерева. Балансування здійснюється за правилами, вказаними в роботі [6], де описані чотири правила балансування для додавання вершини і п'ять правил для вилучення. Червоні вершини позначено білим. Доведено, що при балансуванні після вставки виконується не більше двох зміщень, а при балансуванні після вилучення – не більше трьох [15]. Отже, процедура балансування проводиться за час $O(1)$.

2.2. Локалізація в червоно-чорному дереві

Для розв'язання поставленої нами задачі (динамічної локалізації точки) побудуємо структури даних для пошуку у вигляді червоно-чорного дерева, тобто представимо дерева смуг і відрізків червоно-чорними, і таким чином ми отримуємо динамічну структуру даних.

У контексті нашої задачі важливою функцією червоно-чорного дерева є не традиційний пошук, а локалізація – визначення правого та лівого сусідів. Саме тому на етапі ініціалізації ми додаватимемо в дерево праву та ліву межі (для пошуку по смугах) або верхню і нижню (для пошуку по відрізках). У цьому випадку алгоритм локалізації суттєво спрощу-

ється, а саме: рухаючись вниз по дереву праворуч, запам'ятовуємо даний елемент як лівого сусіда, ліворуч – запам'ятовуємо даний елемент як правого сусіда. На виході алгоритм видає пару <лівий сусід, правий сусід>.

2.3. Загальний алгоритм

Позначимо через $xTree$ дерево смуг для пошуку по осі абсцис, а через $yTree$ – дерево відрізків для пошуку по осі ординат при локалізації в середині локалізованої смуги.

1. Ініціалізація. На початку червоно-чорне дерево смуг $xTree$ складається з двох меж полотна: лівої і правої.

2. При кожному додаванні точки ми додаємо до $xTree$ нову смугу, що лежатиме праворуч від нової точки, до меж наступної смуги.

3. При кожному додаванні відрізка ми рухаємось по $xTree$ і знаходимо смугу, в яку потрапляють ліва і права вершини. Додаємо вершини відрізка у множини *insert* і *delete* відповідно.

4. Визначаємо точку Z.

5. Якщо Z визначена, виконуємо локалізацію. Спочатку описаним вище алгоритмом 2.2, взявши абсцису запитної точки xz, локалізуємо смугу S, в яку потрапляє запитна точка.

6. В середині S ініціалізуємо червоно-чорне дерево відрізків $yTree$, включивши до нього відрізки, що співпадають з нижніми і верхніми границями полотна.

7. Рухаючись по дереву з двох сторін, аналізуємо множини *insert* і *delete* згідно з алгоритмом пошуку редукції Коула [5]. Якщо знаходимо відрізок, ліва точка якого міститься в *insert* смуги ліворуч, а права – в *delete* смуги праворуч, додаємо цей відрізок до $yTree$.

8. Ще раз застосовуємо алгоритм 2.2 і знаходимо відрізки, що локалізують Z зверху і знизу.

9. Побудова області локалізації. Алгоритм детально описано нижче.

3. Підзадачі

У більшості робіт і, зокрема в [6], пропонуються лише принципи розв'язання проблеми, не вдаючись до низькорівневих проблем реалізації. Тому на етапі програмної реалізації виникли задачі, для розв'язання яких запропоновані оригінальні алгоритми. Розглянемо основні з них.

3.1. Введення порядку над відрізками

Нехай дано деяку смугу S, яка включає прямі $l_1, l_2 \in S$. Для організації пошуку, згідно з алгоритмом 3.2, необхідно встановити над даними прямими відношення порядку. Для цього візьмемо деяку точку $A \in l_2$ та визначимо її розташування відносно прямої l_1. У роботі [16] пропонується визначити порядок розташування за годинниковою стрілкою кінця і початку відрізку l_1, а також точки A. Існують й більш прості способи розв'язання цієї задачі. Наприклад, ми можемо скористатися рівнянням прямої

$$y(x) = \frac{(x - x_1)(y_2 - y_1)}{x_2 - x_1} + y_1,$$

де (x_1, y_1) і (x_2, y_2) – координати лівої і правої вершин відрізка l_1 відповідно. Тепер підставимо у це рівняння абсцису точки A і порівняємо отримане значення з ординатою точки

A. Тоді, коли необхідно порівняти відрізки, за точку A обираємо точку перетину прямої l_2 з лівою межею смуги S. Проте не завжди цей спосіб спрацьовує. Можна запропонувати приклад, де це має місце (рис. 3).

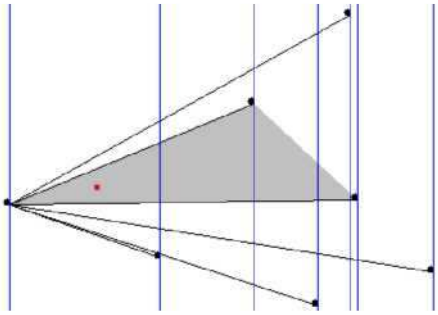

Рис. 3. Найгірший випадок для введення порядку над відрізками

У даному випадку ліва вершина l_1 співпадає з точкою A, а отже, порядок може бути визначений невірно. Це залежатиме від того, до якої півплощини ми відноситимемо точку, що належить прямій. Тому додатково необхідно перевірити, чи не співпадають ліві точки l_1 і l_2. Якщо це так, то за точку A обирається права вершина l_2.

3.2. Обведення області

Після того, як два відрізки локалізовано, постає задача виділення області, в якій лежить точка Z. У зв'язному графі це можна легко зробити, користуючись реберним списком з подвійними зв'язками: кожен відрізок містить посилання на наступний відрізок за годинниковою стрілкою. Тож рухаємось по списку, поки не отримаємо початковий відрізок.

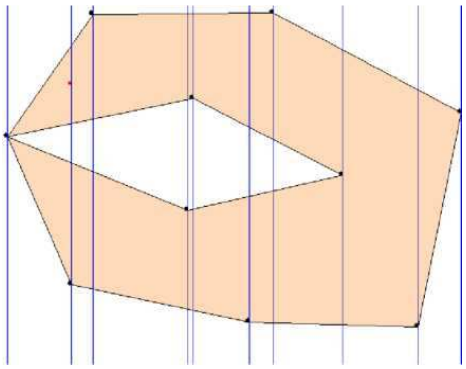

Рис. 4. Приклад локалізації з внутрішньою фігурою

У випадку ж незв'язного графа все не так просто. Необхідно придумати алгоритм, який би або повертав обведену область, або повідомляв про те, що область є незамкненою. Такий алгоритм було запропоновано шляхом введення обмеження на кількість внутрішніх фігур. Варто зазначити, що під внутрішніми фігурами розуміються фігури або ребра, які лежать в середині запитної області і мають з нею спільну точку (рис. 4).

У першу чергу, ми перевіряємо, чи не посилаються праві вершини локалізованих вузлів на самих себе. Якщо це так, очевидно, повертаємо FALSE. Інакше йдемо, починаючи з нижнього відрізка, як і у звичайному випадку. Якщо зустрічаємо точку, яку вже проходили, вводимо лічильник повторних входжень. Якщо цей лічильник переважає обмеження, ми вважаємо, що зациклились і повертаємо FALSE. Повертаємо TRUE, якщо зайшли у початковий відрізок і проходили через верхній локалізований відрізок.

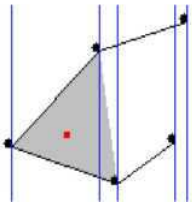

Рис. 5. Приклад проблемного встановлення обмеження на кількість внутрішніх фігур

На жаль, не завжди вдається встановити обмеження на кількість внутрішніх фігур більше, як на 1 (рис. 5). У даному випадку, починаючи з нижнього локалізованого відрізка, йдемо праворуч, одразу ж досягаємо кінця і йдемо у зворотному напрямку. За рахунок швидкого перевищення ліміту повторних входжень ребра алгоритм поверне FALSE. При більш слабких обмеженнях алгоритм працюватиме невірно.

4. Особливості реалізації

Особливістю даної реалізації є можливість роботи на незв'язному графі. Програма здатна виконувати два типи локалізації: для замкненої і незамкненої областей.

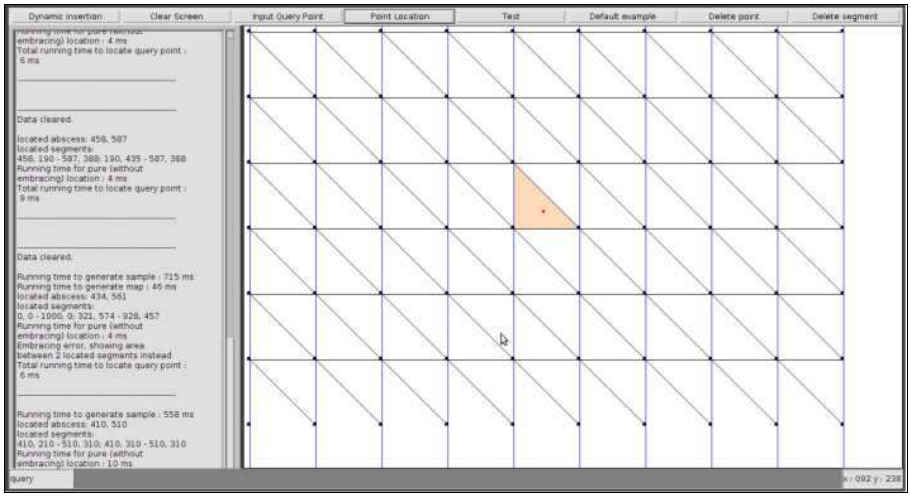

Рис. 6. Інтерфейс програми при стандартному тесті

На рис. 6 ілюструється інтерфейс програми для стандартного тесту.

Алгоритмічна частина програмної реалізації написана на мові Java, для графічного відображення використовується пакет java.awt. Програма спирається на графічний інтерфейс і деякі функції обчислювальної геометрії, запропоновані Лудманом і Депуаном [16]. Для реалізації червоно-чорного дерева частково використовувався програмний код з М. Вайса [17].

5. Обґрунтування оцінок складності

Оцінку складності обґрунтуємо, враховуючи час на основних кроках алгоритму за допомогою таких лем.

Лема 1. Червоно-чорне дерево з N внутрішніми вершинами має висоту не більше, як $2\log(N+1)$ (Кормен [15]).

Лема 2. Вставка і вилучення в червоно-чорному дереві проводиться за час $O(\log(N))$ (Кормен [15]).

З формальним доведенням цих фундаментальних фактів про червоно-чорне дерево можна ознайомитись у роботі [15].

Лема 3. Методом червоно-чорного дерева локалізувати верхній і нижній відрізки можна за час $O(\log(N))$.

Доведення. Оскільки, за лемою 1, висота червоно-чорного дерева не перевищує $2\log(N+1)$, алгоритм локалізації 2.5 працюватиме за час $O(\log(N))$. Під час запиту цей алгоритм застосовується двічі: в середині *xTree* і *yTree*. Звідси оцінка часу локалізації $O(\log(N))+O(\log(N))=O(\log(N))$. □

Лема 4. Метод червоно-чорного дерева використовує $O(n)$ пам'яті.

Доведення. Традиційний метод смуг використовує $O(N^2)$ пам'яті [3]. Використання методу Коула [5] дозволило зменшити об'єм пам'яті до $O(N)$, зберігаючи лише входження і виключення ребер. У загальному випадку маємо $O(N)$ пам'яті. □

6. Висновки

У роботі запропоновано розв'язання динамічної задачі локалізації точки методом червоно-чорного дерева Сарнака-Тар'яна для випадку незв'язного графа. Розроблено ефективну динамічну структуру даних, що дозволяє розв'язати задачу з часом $O(\log(N))$ і використанням $O(N)$ пам'яті. Сама ідея розв'язання задачі має перспективу застосування для

тривимірного випадку. Проблема просторової локалізації точки досліджена значно гірше [18]. Коул [5] передбачав використання редукції для просторової локалізації і наводив свій алгоритм для тривимірного випадку.

СПИСОК ЛІТЕРАТУРИ

1. Arge L. Improved Dynamic Planar Point Location / L. Arge, G.S. Brodai, L. Georgiadis // Proc. 47th Annual Symposium on Foundations of Computer Science. – USA, 2006. – P. 305 – 314.
2. Шашкин И. Локализация точки / Шашкин И. – СПб.: ЛЭТИ, 2008. – 214 с.
3. Препарата Ф. Вычислительная геометрия: Введение / Ф. Препарата, М. Шеймос. – М.: Мир, 1989. – 478 с.
4. Chazelle B. Filtering search: A new approach to query-answering / B. Chazelle // Proc. of 24fh Annual IEEE Symposium on Foundations of Computer Sc~e&, (Tucson. Ariz. Nov. 7–9, 1983). – Tucson. Ariz., 1983. – P. 122 – 132.
5. Cole R. Searching and storing similar lists / R. Cole // J. Algorithms. – 1986. – Vol. 7, N 2. – P. 202 – 220.
6. Sarnak N. Planar Point Location Using Persistent Search Trees / N. Sarnak, R.E. Tarjan // Programming Techniques and Data Structures. – 1986. – Vol. 29, N 7. – P. 669 – 679.
7. Dynamic Point Location via Self-Adjusting Computation / K. Tangwongsan, G. Blelloch, U. Acar [et al.] // Proc.of 23-th annual symposium on Computational geometry. – New-York, USA, 2007. – P. 129 – 130.
8. Dietz P. Two algorithms for maintaining order in a list / P. Dietz, D. Sleator // Proc. 19th Annual ACM Symp. on Theory of Computing. – New-York, USA, 1987. – P. 365 – 372.
9. Two simplified algorithms for maintaining order in a list / M.A. Bender, R. Cole, E.D. Demaine [et al.] // Proc. 10th ESA. – London, UK, 2002 – P. 152 – 164.
10. Arya S. A Simple Entropy-Based Algorithm for Planar Point Location / S. Arya, T. Malamatos, D.M. Mount // SIAM J. Comput. – Philadelphia,USA, 2007. – P. 262 – 268.
11. I/O-Efficient Dynamic Point Location in Monotone Planar Subdivisions / P.K. Agarwal, L. Arge, G.S. Brodai [et al.] // Proc. of Tenth Annual ACM-SIAM Symposium on Discrete Algorithms (SO-DA'99). – Philadelphia,USA, 1999. – P. 11 – 20.
12. 1/O-EfRcient map overlay and point location in low-density subdivisions. / M. de Berg, H. Haverkort, S. Thite [et al.] / / Tokuyama, editors, Proc. of the 18th Annual International Symposium on Algorithms and Computation (ISAAC 2007), LNCS. – 2007. – Vol. 4835. – P. 500 – 511.
13. Chan T.M. Transdichotomous Results in Computational Geometry, I: Point Location in Sublogarithmic Time / Chan T.M., Patrascu M. // SIAM Journal on Computing. – 2009. – Vol. 39, N 2. – P. 703 – 729.
14. Cole R. Geometric retrieval problems / R. Cole, C.K. Yap // Proc. of the 24th Annual Symposium on Foundations of Computer Science. – Washington, USA, 1983. – P. 112 – 121.
15. Кормен Т. Алгоритмы: построение и анализ / Кормен Т., Лейзерсон Ч., Рнвест Р. – М.: МЦМНО, 1999. – 960 с.
16. Brodal G.S. Planar point location using the DCEL / Brodal G.S., Depoyant G., Ludmann M. – Aarhus University, Denmark, 2010. – P. 456.
17. Weiss M.A. Data structures and algorithm analysis / Weiss M.A. – Addison. – Wesley, 2011. – P. 614.
18. Tan X.-H. Spatial Point Location and Its Applications / X.-H. Tan, T. Hirata, Y. Inagaki // Algorithms: International Symposium Sigal '90. – Tokyo, Japan, 1990. – P. 241 – 250.

Стаття надійшла до редакції 03.09.2012