УДК 519.68

Терещенко В.М.[1], д.ф.-м.н., доц.

Модифікація методу ланцюгів для задачі локалізації точки

У роботі пропонується модифікація методу ланцюгів для розв'язання задачі локалізації точки на площині. Запропонована модифікація стосується побудови нової структури даних у вигляді орієнтованого ациклічного графа, яка дозволяє лакалізувати точку за O(logn) часу, використовуючи O(n) пам'яті.

Ключові слова: локалізація точки, монотонний ланцюг, дискримінація точки, структура даних.

[1] Київський національний університет імені Тараса Шевченка,03680, м. Київ, пр-т. Глушкова 4д, e-mail: vtereshch@gmail.com

V.M. Tereshchenko[1], Dr. Sci.(Phis.-Math.), Assoc. Prof.

A modificated chain method for point-location problem

The paper proposes a modification of chain method for solving the point-location problem on the plane. The modification concerns the construction of a new data structure as directed acyclic graph, which allows to locate a point in O(logn) time using O(n) memory.

Key Words: point-location, monotone chain, point discrimination, data structures.

[1] Taras Shevchenko National University of Kyiv, 03680, Kyiv, Glushkova av., 4d, e-mail: vtereshch@gmail.com

Статтю представив д.ф.-м.н., проф. Анісімов А.В.

1. Вступ

Постановка проблеми. Задача локалізація точки є однією з фундаментальних та важливих проблем в області обчислювальної геометрії. Для n- вершинного планарного розбиття важливим завданням є побудова алгоритму цієї задачі, який одночасно досягає $O(\log n)$ часу та $O(n)$ пам'яті.

Аналіз останніх досліджень. Перші оптимальні рішення були представлені Ліптоном і Тар'яном [6] та Кіркпатріком [2]. Метод Ліптона-Тар'яна заснований на теоремі сепаратора графу, і досі представляє лише теоретичний інтерес. Метод деталізації тріангуляції [2], який будує ієрархію тріангуляцій, можна реалізувати, але він має константи занадто великі константи. Окрім того, жоден з цих методів не поширюється на планарні розбиття з криволінійними ребрами.

Історично, Добкін і Ліптон [3] були першими в досягненні $O(\log n)$ оцінки часу запиту, при цьому використовуючи $O(n^2)$ пам'яті. Їх метод, що отримав назву метод смуг, був згодом покращений Препаратою [4], так що оцінка пам'яті стала $O(n\log n)$. Пізніше Білардо і Препарата [1] ще раз покращили оцінку складності пам'яті в середньому до $O(n\log n)$. Окрім цього, ці методи мають позитивну рису: вони можуть бути застосовані до розбиття з криволінійними ребрами.

Істотно інший підхід був розглянутий Шеймосом [7], що привів до відомої роботи про проблему локалізації точки Лі і Препарати [5], заснованої на побудові розділяючих ланцюгів. Час попередньої обробки побудови цієї структури даних складає $O(n\log n)$, з використанням пам'яті $O(n)$ в гіршому випадку, та часом пошуку $O(\log^2 n)$.

Мета роботи. Запропонувати модифікацію методу ланцюгів Лі і Препарати [5], яка покращує оцінки складності часу запиту до $O(\log n)$ та пам'яті до $O(n)$.

Новизна. В роботі пропонується модифікація методу ланцюгів, в основі якої лежить нова структура даних у вигляді орієнтованого ациклічного графу. В цій структурі, точка запиту локалізується не лише у батьківському ланцюгу, а й в одному із ланцюгів нащадків за $O(1)$ часу. Окрім цього, досягається зменшення значень констант та проста реалізація.

2. Побудова модифікованого алгоритму розв'язання задачі

Сподівання отримати швидший алгоритм полягають в тому, що є деякі очевидні втрати інформації в існуючому методі ланцюгів [5].

Зокрема, коли ми локалізуємо точку відносно ланцюга c_k, ми повинні локалізувати її *x*-координату в межах деякого ребра або так званої прогалини c_k. Проте, коли ми продовжуємо пошук вниз по одному із синів вершини k, ми починаємо цей процес локалізації заново. Було б добре, якщо б кожне ребро або прогалина у ланцюгу вказувала на місце, в кожному із ланцюгів нащадків, де пошук по *x*-координаті буде завершений. Проблема в тому, що ребро або прогалина батьківського ланцюга може покривати багато ребер або прогалин в його ланцюгах нащадках.

В роботі пропонується така модифікація ланцюгів, щоб локалізація p_x в одному ланцюгу дозволяла зробити те саме в одному з нащадків за $O(1)$ часу. Ідея розбити кожний ланцюг по *x*-координатам вершин в графі не підходить, адже зрозуміло, що така структура даних вимагатиме O(n^2) пам'яті. Замість цього, для кожного ланцюга c_k опишемо структуру даних L_k, яка визначає розбиття осі OX на інтервали. Кожен такий інтервал L_k покриває *x*-проекцію одного ребра або прогалини ланцюга c_k і не більше двох інтервалів списків $L_{l(k)}$ і $L_{r(k)}$.

L_k списки і їхні взаємозв'язки доцільно представити за допомогою орієнтованого ациклічного графу. Список L_k будується на основі ланцюга c_k і представлений послідовністю вузлів трьох типів: *x*-вузли, *e*-вузли, і *g*-вузли. *x*-вузол відповідає вершині ланцюга і складається із значення *x*-координати цієї вершини (поле $xval$), а також вказівників $left$ і $right$, що вказують на *e*- або *g*-вузол, що знаходяться відповідно ліворуч та праворуч від вершини. *x*-вузол відповідає на питання: "Шукана точка знаходиться ліворуч чи праворуч від вершини?".

e-вузол відповідає ребру ланцюга і складається з поля *edge* з представленням цього ребра, а також вказівників up та $down$, які вказують на вузли $L_{l(k)}$ та $L_{r(k)}$, що можуть бути будь-якими з трьох типів. *e*-вузол відповідає на питання: "Шукана точка знаходиться на, над чи під ребром?". *g*-вузол відповідає прогалині ланцюга і складається з поля $chain$, що вказує номер цього ланцюга, а також вказівників up та $down$, аналогічно *e*-вузлу. На рис. 1 зображені три типи вузлів, на рис. 2 зображений L_k список.

Визначимо значення вказівників $down(t)$ та $up(t)$ для деякого вузла *t*. Вище вказана властивість інтервалів списку L_k забезпечує покриття *x*-інтервалом списку L_k, який відповідає *e*- або *g*-вузлу, одного *x*-інтервалу I списку $L_{l(k)}$, або двох таких інтервалів I_1, I_2, розділених деякою вершиною x_k списку $L_{l(k)}$.

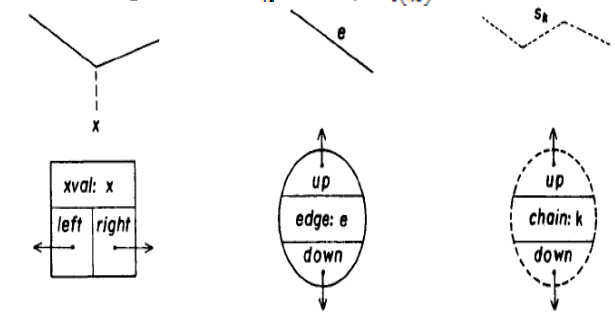

Рис. 1 Три типи вузлів.

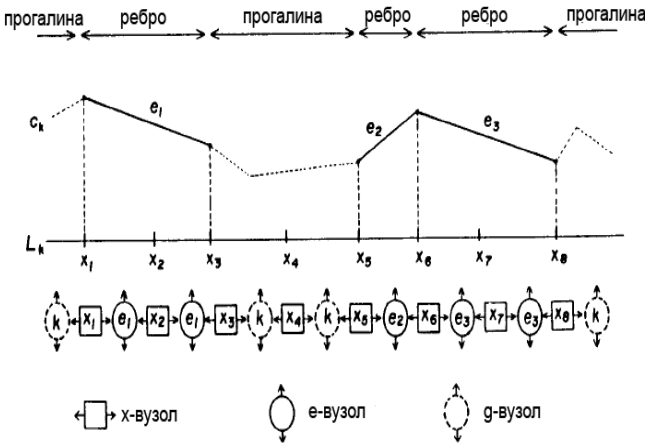

Рис. 2 L_k список.

У першому випадку, *down*(*t*) вказує на *e*- або g-вузол списку $L_{l(k)}$, що відповідає інтервалу I; а в другому випадку, $down(t)$ вказує на *x*-вузол, що відповідає абсцисі x_k. Посилання $up(t)$ вказує на вузол списку $L_{l(k)}$, визначений аналогічним чином. У окремому випадку, коли $r(k)$ та $l(k)$ є листками структури *T*, $down(t) = up(t) = nil$. Таким чином матимемо наступний алгоритм.

Модифікований алгоритм пошуку

{Вхідні дані: вказівник на корінь орієнтованого ациклічного графу і кількість регіонів *n*. *Вихідні дані:* зберігаються в змінній *loc.}*

1. $i := 0, j := n-1, t \leftarrow root$.
2. Поки $i < j$:

{ Відомо, що точка p лежить над s_i та під s_{j+1} ланцюгами. Це означає, що p лежить в одному з регіонів $R_i, R_{i+1}, \ldots, R_j$. Змінна t представляє один з вузлів графа певного типу, і допомагає встановити регіон розташування точки p }

3. Якщо t є *e*-вузол, тоді $e \leftarrow edge(t)$:

{ Відомо, що $p_x \in Пе$ (проекція ребра e на вісь x) }

4. Якщо p лежить на e, $loc \leftarrow e$ та завершити роботу алгоритму.

5. Якщо p лежить над e тоді $t \leftarrow up(t)$ та $t := index(above(e))$, інакше $t \leftarrow down(t)$ та j := $index(below(e))$.

6. Інакше, якщо t є x-вузол, тоді:
{Цей x-вузол переміщує до відповідного ребра наступного ланцюга, відносно якого необхідно порівнювати точку p }
7. Якщо $p_x \leq xval(t)$ тоді $t \leftarrow left(t)$, інакше $t \leftarrow right(t)$.

8. Інакше, якщо t є g-вузол, тоді:

{Визначено розташування точки p відносно відповідного ребра ланцюга прогалини. }
9. Якщо $j < chain(t)$ тоді $t \leftarrow down(t)$, інакше $t \leftarrow up(t)$.
10. $loc \leftarrow R_i$ та закінчити пошук.

3. Побудова структури даних

Побудова структури даних відбувається "знизу вгору" (від листків до кореня), одночасно з уточненням ланцюгів c_k. Вже було описано, як отримуються списки L_k. Тобто, ми вже маємо в своєму розпорядженні три відсортованих списки значень x: ті, які відповідають $L_{l(k)}$, $L_{r(k)}$, а також список кінців ребер, що зберігається з ланцюгом c_k. x-значення у списку L_k є об'єднанням x-значень ланцюга c_k і кожної іншої точки, що присутня як в $L_{l(k)}$ так і в $L_{r(k)}$. Причому, якщо k-ий вузол є листком в дереві ланцюгів T (тобто відповідає певному регіону), або $k \geq n$, тоді список L_k порожній. Просування x-значення від сина до батьківського вузла задовольняє двом корисним властивостям:

Лема 1. Інтервал між двома сусідніми x-значеннями в списку L_k покриває не більше двох інтервалів в $L_{l(k)}$, та в $L_{r(k)}$ аналогічно.

Лема 2. Загальна кількість x-значень в списках L_k, просумованих для кожного k, не збільшується.

Доведення. Нехай a_k це кількість ребер в ланцюгу c_k , тоді $\sum_{k \epsilon T} a_k = m$, оскільки кожне ребро планарного розбиття зустрічається в ланцюгах тільки один раз. Нехай b_k це кількість x-значень в L_k, і A_k(відповідно B_k) це сума a_i(відповідно b_i) по всім вузлам i, що знаходяться в піддереві з коренем в вершині k. Для доведення леми достатньо показати $B_r \leq 4A_r = 4m$ для кореневої вершини $r = lca(0, n-1)$. Будемо доводити за індукцією, припускаючи гіпотезу $B_i + b_i \leq 4A_i$. Гіпотеза тривіально справджується для i-их вузлів, які є листками в дереві ланцюгів T. З того, що $B_k = B_{l(k)} + B_{r(k)} + b_k$, $A_k = A_{l(k)} + A_{r(k)} + a_k$, та $b_k \leq 2a_k + (b_{l(k)} + b_{r(k)})/2$, оскільки кожне ребро в ланцюгу c_k вносить не більше двох значень до списку L_k, випливає $B_i + b_i \leq B_{l(i)} + B_{r(i)} + 4a_i + b_{l(i)} + b_{r(i)} \leq 4A_i$ ■

Інтуїтивно зрозуміло, що половина x-значень ланцюга c_k поширюється батьківському ланцюгу, чверть діду, восьма частина прадіду, і так далі. Зазначений факт ілюструє лінійність загальної довжини L списків.

Алгоритм побудови структури даних

1. i:=1. Поки i < n:
{ Побудова ще одного рівня графу. Номер першого вузла =i, різниця між послідовними вузлами (що лежать в одному піддереві) = 2i. K-й вузол на цьому рівні є найменшим спільним нащадком листків від k — i до k + i — 1; його нащадками (якщо i > 1) є вузли k — i/2 та k + i/2 }
2. $k \leftarrow i$. Поки $i < n$:
{ Створюємо список L_k із ланцюгу c_k. $L' = L_{l(k)}$ та $L'' = L_{r(k)}$ }
3. Якщо $i = 1$, тоді $L' \leftarrow \emptyset$, інакше $L' \leftarrow L_{k-i/2}$.
4. Якщо $i = 1$ або $k + i/2 \geq n$, тоді $L'' \leftarrow \emptyset$, інакше $L'' \leftarrow L_{k+i/2}$
5. Якщо $k \geq n$, тоді нехай c_k є звичайною прогалиною від $x = -\infty$ до $x = +\infty$. Розбиваємо ребра і прогалини ланцюга c_k в кожній точці іншого x-значення списків L' та L".
6. $L_k \leftarrow \emptyset$. Для кожного ребра чи прогалини e в ланцюгу c_k:

а) Додати в L_k e-вузол чи g-вузол t, що представляє e, $edge(t) \leftarrow e$ чи $chain(t) \leftarrow k$.

б) Якщо $L' = \emptyset$, тоді $down(t) \leftarrow nil$, інакше, якщо Πe покриває тільки один x-інтервал списку L', тоді вказівник $down(t)$ посилаємо на відповідне ребро або прогалину списку L'. Інакше, якщо Πe покриває два x-інтервали L', тоді створити x-вузол t', що вибирає між двома відповідними ребрами чи прогалинами списку L', та $down(t) \leftarrow t'$.

в) Таким же чином, встановлюємо вказівник $up(t)$ в nil, або в e-вузол або в g-вузол списку L'', або в новий x-вузол, що вибирає між двома вузлами списку L''.

7. $k \leftarrow k + 2i$.
8. $i \leftarrow 2i$.

9. Нехай $r \leftarrow lca(0, n-1)$. Побудуємо збалансоване бінарне дерево *x*-вузлів списку L_r, тоді *root* вказує на корінь цього дерева.

Різні вузли L_k можуть вказувати на один і той же вузол (рис.3). Зауважимо, що структура, побудована Кіркпатріком [2] також відповідає орієнтованому ациклічному графу. Таке спільне використання піддерев здається є однією з основних властивостей алгоритмів локалізації точки, які одночасно досягають оптимальної оцінки часу і пам'яті. Щоб зменшити кількість вказівників, можна зберігати вузли кожного списку L_k в послідовних комірках пам'яті (в масивах), у їхньому природньому порядку зліва-направо. Тоді початкове знаходження x_p в L_r можна знайти за допомогою звичайного бінарного пошуку. Окрім того, при побудові L_k можна переглядати списки $L_{l(k)}$ та $L_{r(k)}$ без зайвих вказівників. Також, таке послідовне зберігання вузлів усунуло б необхідність зберігання $left$ та $right$ вказівників *x*-вузлів, що приводить до зменшення використаного об'єму пам'яті на одну третину.

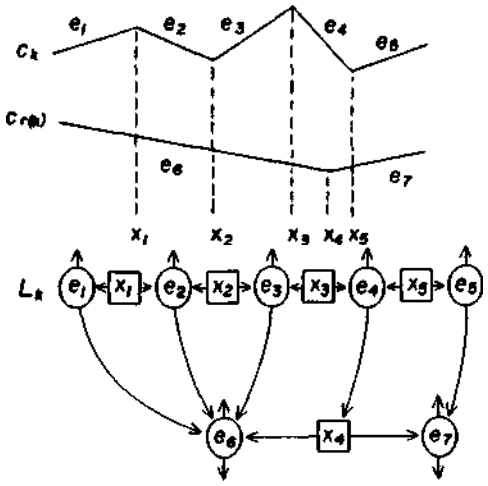

Рис. 3 Сходження декількох вказівників в один вузол.

У будь-якому випадку, початкове знаходження x_p в кореневому списку L_r може бути встановлено за $O(\log m)$. Після цього алгоритм виконує точно $\lceil log\, n \rceil$ порівнянь в *e*- або *g*-вузлах (одне на кожному рівні T), і не більше цієї ж кількості порівнянь в *x*-вузлах. Таким чином, загальний час запиту $O(\log m)$. Список L_k з t *x*-значеннями представлений в структурі не більше ніж t *x*-вузлами та $t+1$ *e*- або *g*-вузлами і, як було показано, може бути побудований за $O(t)$ часу. Використовуючи лему 2, робимо висновок, що орієнтований ациклічний граф не збільшує кількість *x*-вузлів та *e*- або *g*-вузлів, і може бути побудований за $O(m+n)$ з дерева ланцюгів *T*. Таким чином, має місце наступний висновок.

Теорема 1. Маючи дерево ланцюгів для планарного розбиття з *n* ребрами, можна побудувати структуру даних за час $O(n)$, що використовує $O(n)$ пам'яті, і дозволяє локалізувати точку за $O(\log n)$ часу.

4. Висновки

В роботі представлена модифікація методу ланцюгів, що дозволяє побудувати лінійну за розміром структуру даних, яка вирішує задачу локалізації точки для планарних розбиттів на площині з оптимальною оцінкою складності часу пошуку $O(\log n)$. Основна ідея полягає в тому, щоб зберігати рівно одне ребро серед усіх ланцюгів, які були вперше представлені Лі та Препаратою, а також з'єднувати їх вказівниками. Структура даних може бути побудована із звичайного дерева ланцюгів за лінійний час, як це було описано в підрозділі 3. Запропонована модифікація може бути узагальнена і на випадок криволінійного розбиття.

Список використаних джерел

1. *Bilardi G. and Preparata F. P.* Probabilistic analysis of a new geometric searching technique. Manuscript, Dept. of EECS, Univ. of Illinois. -1982.
2. *Kirkpatrick, D. G.* Optimal search in planar subdivisions // SIAM Journal on Computing. –1983. –N12. –P. 28-35.
3. *Dobkin D. P., Lipton R. J.* Multidimensional searching problems//SIAM Journal on Computing. –1976. – Vol. 5. –P. 181-186.
4. *Preparata F. P.* A new approach to planar point location// SIAM Journal on Computing. –1981. –Vol. 10. –P. 473-482.
5. *Lee D. T., Preparata F. P.* Location of a point in a planar subdivision and its applications // SIAM Journal on Computing. -1977. – V 6.- P. 594-606.
6. *Lipton R. J.,* Tarjan R. E. Applications of a planar separator theorem // Proc. 18th IEEE Symp. on Foundations of Computer Science. – 1977. – P. 162-170.
7. *Shamos M. I.* Geometric complexity // Proc. 7th ACM Symp. on Theory of Computing. – 1975.-P. 224-233.

Надійшла до редакції 28.03.13