Ідеї розв'язку задач

25 березня 2009 р.

1 "Брехуни" (Андрій Коротков).

Відповідаючи на питання учасників, одні члени журі завжди кажуть правду, інші – завжди брешуть.

Хто може потенційно казати правду Припустимо, що серед учасників ток-шоу рівно k людей, що завжди говорять правду. Тоді всі вони дадуть у відповідь число k, а решта, як брехуни, дадуть відповідь, відмінну від k. Отже, буде одержано рівно k відповідей k. Отже, якщо розбити учасників на групи за даною ними відповіддю, то кожна така група або вся каже правду, або вся бреше, причому необхідною умовою для їх правдивості є рівність кількості людей у групі даній цією групою відповіддю.

Якщо остання умова виконується, то група потенційно може бути правдивою. Підрахуємо кількість таких груп. Якщо така група одна, то, зрозуміло, вона і є правдивою. Якщо таких груп не менше за дві, то потенційно кожна із них може бути правдивою, причому за результатами першого опитування виявити, яка саме з них каже правду, неможливо, бо за припущення правдивості будь-якої з них не виникає суперечностей із умовою (всі відповіді на перше запитання будуть коректними, яка з груп не була б правдивою).

Вибір учасників для другого опитування Для другого опитування достатньо вибрати по одному учаснику з кожної групи, *для якої дана ними відповідь рівна кількості людей, що дали таку відповідь*. Тоді серед цих учасників буде рівно один, що завжди каже правду (правдива лише одна із груп), і він при другому опитуванні дасть відповідь 1, а ніхто інший відповідь 1 не дасть, що і дозволить виявити його, а за ним і всю його групу. *Відповіддю буде кількість вище вказаних груп, якщо таких груп не менше за 2, та 0, якщо така група одна.*

Доведення необхідності такого вибору Якщо з деякої потенційно правдивої групи не взяти учасників на друге опитування взагалі, то якщо ця група виявиться правдивою, то на друге опитування потраплять самі брехуни, і вони можуть дати будь-які ненульові відповіді. Якщо ж на друге опитування потрапить хоч один, що говорить правду, то ті, що говорять правду, дадуть ненульову відповідь, і, якщо брехуни також дадуть ненульові відповіді, то одержимо два різних несуперечливих сценарії розвитку другого опитування, і для довільного обраного критерію визначення тих, що кажуть правду, в одному випадку він спрацює, а в іншому ні. Тому в такому разі таке друге опитування не дозволить однозначно визначити тих, що говорять правду. Отже, вибір хоча б одного учасника з кожної потенційно правдивої групи є необхідною умовою визначення тих, що говорять правду, за другим голосуванням. Як було зазначено вище, вибір рівно одного учасника з кожної з таких груп є достатнім для визначення тих, що говорять правду, за другим голосуванням. Отже, вибір рівно одного учасника з кожної такої групи і є оптимальним розв'язком поставленої задачі. Ще раз зазначимо, що при наявності лише одної потенційно правдивої групи проводити друге опитування не потрібно.

2 "Гірський туризм" (Даніїл Нейтер).

2.1 Формалізація задачі і загальна ідея розв'язку.

Нехай N – кількість одинометрових відрізків, на які розбито хребет. T – максимальна допустима довжина шуканого маршруту. H_i– висота i-го відрізка (починаючи з 1). Нехай p_i– привабливість i-го відрізка. Також нехай L_i – індекс останнього відрізка такого, що $L_i < i$ і $H_{L_i} \geq H_i$ (якщо такого відрізка не існує, то $L_i = 0$). Аналогічно R_i – індекс першого відрізка такого, що $R_i > i$ і $H_{R_i} \geq H_i$ (якщо такого відрізка не існує, то $R_i = N + 1$). Тоді $p_i = R_i - L_i - 2$ і $S_i = p_i + p_{i+1} + \cdots + p_{i+T-1}$ задача полягає у знаходженні $max\{S_i\}$ (очевидно, оскільки все p_i невід'ємні, то завжди оптимально буде вибирати маршрут довжини саме T).

Розв'язок задачі складається з двох етапів:

1. Знайти значення L_i і R_i.
2. Знайти $max\{p_i + p_{i+1} + \cdots + p_{i+T-1}\}$.

2.2 Можливі варіанти реалізації першого етапу.

Очевидна реалізація першого етапу алгоритму має часову складність за $O(N^2)$, що є недостатньо ефективним для розв'язку задачі. Ефективні реалізації мають часову складність $O(N)$.

Крім того, існують реалізації зі складність $O(N \log N)$, але вони є менш ефективними і потребують складних структур даних (дерева відрізків, бінарні дерева, або RMQ[1] + бінарний пошук).

2.2.1 Очевидний алгоритм.

Для кожного i від 1 до N знаходимо L_i за наступною схемою:

1. Покладемо $k = i - 1$.
2. Доки є вірним, що $k > 0$ і $H_k < H_i$, присвоюємо $k = k - 1$.
3. $L_i = k$.

R_i знаходяться аналогічно.
Часова складність такого алгоритму $O(N^2)$, просторова– $O(N)$.

2.2.2 Ефективний алгоритм 1

Ключовим для роботи даного алгоритму є той факт, що якщо $H_k < H_i$, то $H_j \leq H_k < H_i$ для всіх j від $L_k + 1$ до k (За означенням L_k). Тому у реалізації очевидного алгоритму від k замість $k-1$, можемо перейти на L_k. Назвемо це перестрибуванням з k на L_k. Алгоритм прийме наступний вигляд:

Для кожного i від 1 до N знаходимо L_i за наступною схемою:

1. Покладемо $k = i - 1$.
2. Доки є вірним, що $k > 0$ і $H_k < H_i$, присвоюємо $k = L_k$.
3. $L_i = k$.

R_i знаходяться аналогічно.
Коректність алгоритму випливає з наведених вище міркувань.

Доведемо, що часова складність алгоритму $O(N)$. Назвемо пропуском k ситуацію, коли на другому кроці відбувається перехід від k до L_k. Доведемо наступну лему:

Лема 2.1. *Кожне k протягом роботи алгоритму може бути пропущено не більше одного разу. Після пропуску позиція k більше на другому кроці розглядатися не буде.*

[1]Range Minimum Query - структура, яка дозволяє відповідати на запити максимумів на довільних відрізках за час $O(1)$ за умови виконання попередньої підготовки за час $O(N \log N)$ і потребує $O(N \log N)$ пам'яті. Існує також реалізація RMQ, яка потребує $O(N)$ пам'яті і часу на підготовку, але вона є занадто складною для написання під час туру.

Доведення. Припустимо, що один раз вже стався пропуск k під час пошуку L_{i_0}. Тоді $H_k < H_{i_o}$. І припустимо, що k повторно розглядається при пошуку L_i, $i > i_0$. Тоді перед розгляданням k ми повинні перетнути i_o. Є два варіанти:

- i_o було пропущено. Але $H_k < H_{i_o}$, з чого випливає $L_{i_o} \neq k$. Протиріччя.
- i_o було перестрибнуто під час пропуску i_1 ($i0 < i1 < i$). Тоді $H_k < H_{i_0} < H_{i_1}$. Але $H_{i1} \leq H_{L_{i_1}} \leq H_{L_{L_{i_1}}} \leq \cdots \leq H_k$. Протиріччя.

В обох випадках ми прийшли до протиріччя. Отже початкове припущення було невірне, і після пропуску позиція k більше розглядатись не буде. □

Тепер проаналізуємо час виконання алгоритму. Кроки 1 і 3 виконуються один раз для кожного i від 1 до N. Крок 2 за доведеною вище лемою виконується для кожного k від 1 до N не більше одного разу. Крім того, одноразове виконання кожного часу потребує $O(1)$ часу. Таким чином сумарна часова складність алгоритму $O(N)$. Просторова складність, очевидно, теж складає $O(N)$.

2.2.3 Ефективний алгоритм 2

Існує альтернативний алгоритм, що дозволяє визначити L_i і R_i за $O(N)$. В процесі роботи цей алгоритм використовує стек S, у якому зберігаються індекси k елементів, для яких у поточний момент вже відомо L_k, але ще не відомо R_k. Позначимо вершину стеку $Top(S)$. Алгоритм виглядає так:

1. На початку роботі стек S порожній.
2. Для кожного i від 1 до N виконуємо наступні дії:
   (a) Доки S не порожній і $H_{Top(S)} < H_i$, виштовхуємо $Top(S)$ зі стеку. При цьому присвоюємо $R_{Top(S)} = i$.
   (b) Якщо стек порожній, $L_i = 0$, інакше $L_i = Top(S)$.
   (c) Якщо стек непорожній і $H_{Top(S)} = H_i$, виштовхуємо $Top(S)$ зі стеку. При цьому присвоюємо $R_{Top(S)} = i$.
3. Для всіх k, що залишились у стеку після завершення другого кроку, присвоюємо $R_k = N + 1$.

Кожне число протягом роботи алгоритму попадає у стек рівно один раз, тому в сумі алгоритм виконується за $O(N)$ операцій.

Висоти елементів, які знаходяться в стеку, являють собою строго спадну послідовність. Число буде виштовхнуто зі стеку тільки тоді, коли буде знайдено елемент з більшою або рівною висотою. З цього випливає коректність алгоритму.

2.3 Можливі варіанти реалізації другого етапу.

2.3.1 Очевидний алгоритм.

Для кожного i від 1 до $N - T + 1$, підраховуємо S_i прямим сумуванням доданків і вибираємо максимум отриманих сум.

Часова складність $O(NT)$.

2.3.2 Ефективний алгоритм.

Використаємо метод ковзання. Спочатку знайдемо S_1 за означенням. Далі для знаходження S_{i+1} використаємо тотожність $S_{i+1} = S_i + p_{i+T} - p_i$.

Очевидно, часова складність такого алгоритму буде $O(N)$.

2.4 Особливості реалізації

При реалізації, треба враховувати, що S_i можуть не вміщуватися у 32-бітні типи даних, тому для підрахунку сум треба використовувати 64-бітні типи.

Крім того, роботу програми можна прискорити, якщо одразу весь зміст вхідного файлу у тимчасовий буфер та виділити з нього числа вручну. Але це не є обов'язковим для ефективного розв'язку.

3 "Танглеграм" (Тарас Галковський).

Вхідними даними задачі є перестановка B чисел від 1 до 2^N, що задає відповідності між листками дерев вигляду: $1 - B_1, 2 - B_2 \dots 2^N - B_{2^N}$. Тобто початковий порядок листків першого дерева задається ідентичною перестановкою $A = (1, 2, ...2^N)$. Операції обміну сусідніх піддерев першого дерева призведуть до змін перестановки A. Отже стан дерев у будь-який час, а також початковий і результуючий стан системи можна описати парою перестановок (A, B), що задають відповідності між листками $A_1 - B_1, A_2 - B_2, \dots, A_{2^N} - B_{2^N}$.

Перші 2^{N-1} елементів перестановки A є листками лівого піддерева, відповідно, останні 2^{N-1} елементів перестановки A є листками правого піддерева. Будемо їх позначати A_l та A_r, аналогічно визначивши перестановки B_l та B_r. Підзадачами (A, B) будемо називати задачі (A_l, B_l) та (A_r, B_r).

Кількість перетинів відрізків-відповідностей задачі (A, B) будемо позначати $f(A, B)$.

Назвемо A' оптимальною перестановкою A, якщо $\forall C \ : \ f(A', B) \leq f(C, B)$, де C перестановка A. Тобто A' є шуканою перестановкою, що забезпечує мінімальну кількість перетинів відрізків.

Лема 3.1 (Про локальну оптимальність)**.** *Нехай A' оптимальна перестановка A. Тоді A'_l та A'_r є оптимальними перестановками для підзадач (A_l, B_l) та (A_r, B_r).*

Доведення. Очевидно від супротивного. □

Лема 3.2 (Про розділення підзадач)**.** *$f(A, B) = f(A_l, B_l)+f(A_r, B_r)+g(A_l, B_l, A_r, B_r)$, де $g(A_l, B_l, A_r, B_r)$- – кількість перетинів лише між відповідностями, що починаються у різних піддеревах.*

Доведення. $f(A, B)$ підраховує перетини між відрізками-відповідностями, (а) обидва відрізки виходять з листів лівого піддерева, (б) обидва відрізки виходять з листів правого піддерева, (в) один з відрізків виходить з листа лівого піддерева, а інший– з листа правого піддерева.

Оскільки перераховані класи перетинів, очевидно, не перетинаються, і крім того покривають всі можливі варіанти, лема доведена. □

Отже, з леми 1 та 2 отримуємо алгоритм знаходження оптимальної перестановки: для кореня дерева порахувати $g(A_l, B_l, A_r, B_r)$ та $g(A_r, B_r, A_l, B_l)$. Вибравши порядок, що забезпечує мінімальну кількість перетинів, обміняємо, якщо необхідно піддерева, і для кожного піддерева повторимо алгоритм.

Обчислення $g(A_l, B_l, A_r, B_r)$ можна провести кількома методами. Найпростіший: для кожної пари відрізків, де один відрізок виходить з лівого піддерева, а інший виходить з правого, перевіряємо, чи перетинаються. Очевидно, складність такої процедури $O(m^2)$, де m– кількість листів у лівому та правому піддереві.

Швидше можна реалізувати обчислення g таким чином: відсортуємо кінці відрізків, що починаються у лівому піддереві (i, j), де $i \in A_l$, $j \in B$. Результат– список L, що складається з j таких відрізків. Оберемо деякий відрізок, що починається у правому піддереві (i, j), де $i \in A_r$, $j \in B$. Помітимо, що кількість перетинів цього відрізку із відрізками, що починаються у лівому піддереві, буде кількість чисел у L більших за j. Складність пошуку елементу у відсортованому списку розміру m є $O(\log m)$, а складність сортування $O(m \log m)$.

Неважко також помітити, що обчислення g можна проводити відразу для всіх піддерев глибини i (так легше оцінити загальну складність алгоритму). Отже для кожного з N рівнів піддерев виконуються алгоритми складності $O(m^2)$ чи $O(m \log m)$. Відповідно загальна складність розв'язків є $O(m^2 \log m)$ чи $O(m \log^2 m)$, де m– кількість листів дерева, тобто $m = 2^N$, де N– глибина дерева.

4 "Еволюція" (Даніїл Нейтер).

4.1 Опис алгоритму

Як зрозуміло з малюнку, прямимми потомками i-го виду, є види $2i$ і $2i+1$. Відповідно, безпосереднім предком вида i є вид з номером $[i/2]$.

Звідси випливає основна ідея роз'вязку. Нехай треба знайти останнього спільного предка видів a і b. Доки $a \neq b$, ділимо більше з пари чисел на 2 з остачею. Отримане в результаті число є номером виду останнього спільного предка.

Пояснимо, чому такий розв'язок є корректним.

Твердження 4.1. Номер виду, який знаходиться на k-му кроці еволюції, записується за допомогою k-бітного числа, старший біт якого 1.

Доведення. Доводиться очевидним чином за принципом індукції по k. □

Очевидним наслідком наведеного вище твердження є факт, що якщо a є предком b, то $a < b$.

Нехай c – останній спільний предок a і b. Тоді у кожний момент виконання алгоритму є 3 варіанти:

- $a = b$. Тоді очевидно, алгоритм зупиниться, вивівши у якості відповіді a. Очевидно, що $c = a = b$.
- a і b знаходяться на різних кроках еволюції. Нехай для, визначеності, b знаходиться на пізнішому кроці. Тоді з доведеного вище твердження випливає, що $a < b$. Маємо $c \leq a < b$, тому $c <= [b/2]$ і ми можемо перейти до розгляду пари вершин a і $[b/2]$.
- a і b знаходяться на одному крокі еволюції, але $a \neq b$. Тоді зрозуміло, що $c < a$ і $c < b$, звідки маємо $c <= [b/2]$ і ми можемо перейти до розгляду пари вершин a і $[b/2]$.

Таким чином алгоритм є коректним.

4.2 Особливості реалізації

Основна особливість реалізації полягає в тому, що номери видів можуть досягати $2^N - 1$. А при $N = 100$ це більше, ніж здатні представити 32-бітні і 64-бітні типи. Тому для проходження всіх тестів, необхідно реалізовати дії з довгими числами: а саме порівняння і цілочисельне ділення на 2. Обидві ці операції легко реалізуються з часовою складністю $O(N)$. Оскільки на кожному кроці алгоритму одне з пари чисел a і b зменшується вдвічі, алгоритм гарантовано завершиться за $2N - 2$ кроків. Таким чином загальна часова складність алгоритму складає $O(N^2)$.

5 “Торговець” (Андрій Коротков).

> В задаче “Торговец” не указаны ограничения для количества алмазов, яблок и шелка, а также их стоимостей. Может, стоит указать?
>
> з питань учасників

Може скластися враження, що завжди вигідно брати всі товари, але це не так.

Твердження 5.1. Який би торговець не вибрав шлях, кожен з видів товару потрібно брати або повністю, або не брати взагалі (для отримання максимального прибутку).

Доведення. Нехай обрано шлях, що проходить вершинами $1 = v_1, v_2, \ldots, v_{k-1}, v_k = N$. Позначимо через C сумарну вартість проїду по ребрах від v_1 до v_2, від v_2 до v_3,..., від v_{k-1} до v_k; через c_i – кількість одиниць ваги i-го виду товару у торговця; через p_i – вартість однієї одиниці ваги i-го товару у столиці; через z_i – суму по всіх містах $v_2, \ldots, v_{k-1}$ податків за i-й вид товару, виражену у відсотках. Тоді торговець одержить прибуток $[((100 - z_1)/100)c_1p_1 + ((100 - z_2)/100)c_2p_2 + ((100 - z_3)/100)c_3p_3 - C]$. Звідси видно, що якщо $z_i < 100$, то прибуток за i-й вид товару буде додатнім, якщо $z_i > 100$, то від'ємним, а якщо $z_i = 100$, то нульовим. Якщо прибуток по товару від'ємний, то брати його з собою невигідно незалежно від кількості, якщо нульовий, то значення не має (будемо вважати, що товар не беремо), якщо додатній, то чим більше взяти товару, тим більше буде добуток і прибуток за товар. Отже, твердження доведено. □

5.1 Загальна ідея розв'язку

Переберемо всі варіанти беремо/не беремо по кожному з видів товару – всього $2^3 = 8$ варіантів. Коли обрано, які товари буде везти торговець, то: а) однозначно визначено, скільки коштів він одержить за продаж товару у столиці; б) однозначно визначено, скільки податків у золотих заплатить торговець при відвідуванні кожного міста.

Отже, задача зводиться до мінімізації витрат на подорож (оскільки загальний прибуток = гроші за продаж товарів у столиці – витрати на подорож; і перший доданок фіксований). У термінах графу необхідно знайти мінімальний маршрут із вершини 1 у вершину N, якщо відомі вартості ребер та вартості вершин. Розглянемо можливі реалізації цієї підзадачі:

5.2 Алгоритм Дейкстри

Розглянемо ціну ребра як суму його безпосередньої вартості та вартості вершини, в яку воно входить: якщо буде відбуватися прохід по ребру $(a; b)$, то кандидатом на мінімальну вартість маршруту до вершини b буде $f[a] + c(a; b) + p[b]$, де через $f[x]$ позначено мінімальну вартість маршруту до вершини x, $c(x; y)$ – вартість ребра із вершин x до вершини y; $p[x]$ – вартість вершини x. У кожному маршруті кожне ребро і кожна вершина будуть враховані у вартість рівно один раз. Вибір на кожному кроці сірої вершини з найменшою вартістю буде коректний: так само, як у чистому алгоритмі Дейкстри, легко показати від супротивного, що для такої вершини знайдено маршрут найменшої вартості. Оцінка складності підзадачі $O(N^2)$. *Сумарна оцінка для задачі* $O(8N^2)$.

5.3 Топологічне сортування із динамікою

Орієнтуємо ребра у протилежному напрямку, та зробимо топологічне сортування. Вершина N буде найстаршою. Позначимо через $t[i]$ i-ту в топологічному порядку вершину ($t[N] = N$); через $f[i]$ максимальний прибуток, який можна одержати, мандруючи із вершини 1 до вершини i, та вважаючи, що товари продаються у i-й вершині.

Початкові умови:

$f[1]$ = вартості взятих із собою товарів у столиці.

$f[i] = 0$ для $i = 2..N$

Обчислення $f[i]$*:*

Пройдемо циклом по i від 1 до N, на i-му кроці обчислюючи $f[t[i]]$. $f[t[i]] = max\{f[adj[t[i], j]] - c[t[i], adj[t[i], j]] - p[t[i]] \mid j = 1..num[i]\}$, якщо $t[i] \neq 1$. Тут через $adj[x, y]$ позначено суміжну вершину до x, що стоїть у списку суміжних вершин під номером y; через $c[x, y]$ – вартість ребра із вершини x у вершину y; $p[i]$ – вартість вершини i; $num[i]$ – кількість суміжних з i-ю вершин. Формат формули близький до її запису у коді програми.

Оскільки вершина топологічно старша за всіх своїх потомків, то при обчисленні функції для вершини функції для потомків вже будуть обчислені, так що динаміка коректна.

Оцінка складності $O(N + M)$, що у найгіршому випадку дає $O(N^2)$. *Сумарна оцінка для задачі* $O(8(N + M))$.

5.4 Можливі евристики та неефективні розв'язки

1. Повний перебір усіх шляхів із визначенням відсотків податку по кожному з видів товару та вартості проїзду по дорогах. *Оцінка складності* $O(N!)$, хоча при розріджених графах та відтинах завідомо неоптимальних шляхів швидкість буде набагато більшою порівняно із вказаною.

2. Брати всі товари (*евристика*).

3. Для пошуку маршруту найменшої вартості використовувати алгоритм Флойда-Уоршала. Оцінка його складності $O(N^3)$. *Сумарна оцінка на задачу* $O(8N^3)$.

4. Проводиться динаміка без топологічного сортування (*евристика*).

6 “Кола” (Тарас Галковський).

Под понятием окружности подразумевается кольцо или круг? В условии написано про кольцо.

з питань учасників

6.1 Загальна ідея розв’язку.

Задача легко розв’язується перебором всіх пар кіл, і перевіркою, чи перетинаються кола кожної пари. Нескладно переконатися, що кількість таких пар n^2, де n – кількість кіл. Очевидно, що перебирати всі пари є неоптимально. Наприклад, кола, що розташовані по різні кінці площини, і між ними знаходиться велика кількість інших кіл. Як розрізнити такі пари кіл: такі, що знаходяться "далеко"одне від одного та такі, що знаходяться поруч, і між ними немає інших кіл?

Проведемо деяку пряму, паралельну Ox. Вона перетне деякі кола. Помітимо точки перетину прямої та кіл мітками. Очевидно, що підозрілими є лише такі пари кіл, що мітки яких є сусідами на такій прямій: між ними немає іншої мітки.

Припустимо, що кола не перетинаються, а пряма "замітає"площину, паралельно до Ox. Тоді мітки, що ми розставляємо на прямій будуть "рухатися", "плавати"по прямій. Але можна довести (Лема 1), що порядок міток на цій прямій не буде змінюватися: можливими змінами є лише поява нових міток (по 2 на кожне коло) та зникнення старих. Моментами таких подій будуть точки дотику замітаючої прямої до кола.

Отже, нові пари сусідніх кіл (по відношенню до замітаючої прямої) будуть з’являтися лише в моменти дотику прямої до кіл. На кожну подію дотику певного кола вперше утвориться дві нових пари сусідніх кіл, а на подію дотику до певного кола вдруге, утвориться лише одна пара сусідніх кіл.

Теоретичне обгрунтування, чому ми можемо зупинятися лише в точках дотику замітаючої прямої до кіл, та чому необхідно перевіряти лише пари сусідів, які утворюються при доданні/виключенні наведено у наступній частині.

Зауважимо, що перевірка перетину кіл може виконуватися з використанням цілочисельної арифметики, а дробові числа використовуються лише для сортування міток на замітаючій прямій. Точність обчислення в цьому випадку, з огляду на наведені обмеження задачі має бути не менше за 10^{-5}, що забезпечується стандартними типами запропонованих компіляторів.

6.2 Обгрунтування

Означення 6.1. *Структурою перетину* P в момент часу t назвемо впорядковану множину точок, таку що: для кожного кола C_i, яке перетинається замітючою прямою в момент часу t, в множині P знаходиться рівно дві, можливо, співпадаючі точки $(l_i(t),\, r_i(t))$, що є точками перетину замітаючої прямої та кола. Нехай на замітаючий прямій визначений певний напрям. На структурі перетину визначимо природній порядок: $\forall a, b \in P\ a \prec_P b$, якщо a слідує раніше за b на замітаючій прямій.

Лема 6.1 (Про кола, що перетинаються)**.** *Нехай для довільних моментів часу* t_1, t_2, *кола* C_i, C_j *перетинаються замітаючою прямою в точках* $l_i(t)$, $r_i(t)$ *та* $l_j(t)$, $r_j(t)$ *відповідно. Якщо* $r_j(t_1) \prec_{P_{t_1}} l_i(t_1) \ \wedge\ l_i(t_2) \prec_{P_{t_2}} r_j(t_2)$ *тоді кола* C_i *та* C_j *перетинаються.*

Доведення. За теоремою про середні значення функцій отримуємо, що $\exists t \in [t_1, t_2] :\ r_j(t) = l_i(t)$. Отже кола мають спільну точку. Що і треба було довести. □

Важливим наслідком леми 6.1 є факт, що при відсутності кіл, що перетинаються, порядок точок перетину у структурі перетинів не буде змінюватися. Цей факт необхідний для підтримання структури перетинів, при переміщенні замітаючої прямої. Також наслідком з леми 6.1 є властивість, що для двох кіл, що перетинаються існує такий момент часу t, коли ці кола є сусідами в структурі перетину. Отже, для того, щоб впевнитися, що жодна пара кіл не перетинається нам необхідно перевіряти всі сусідні пари в структурі перетину.

6.3 Схема алгоритму

Опишемо схему алгоритму для визначення наявності кіл, що перетинаються серед заданих. Нехай замітаюча пряма рухається площиною. Протягом свого руху пряма перетинає деякі кола. Для деякого кола C_i моментом часу, коли пряма вперше дотикається до нього є s_i, і відповідно останнім дотиком замітаючої прямої є момент часу f_i. З визначення структури перетину, ця структура в точці s_i буде поповнюватися двома точками $(l_i(t), r_i(t))$. При внесенні нових точок до структури перетину, перевіряємо сусідні з ними точки інших кіл на перетин між цими парами кіл. Аналогічно, в момент часу f_i, зі структури зникнуть дві точки $(l_i(t), r_i(t))$. У структурі перетину утвориться нова пара сусідів, які треба також перевірити на перетин.

Оцінимо складність алгоритму. Першим кроком алгоритму сортується всі s_i, f_i – моменти зупинки замітаючої прямої, тобто $O(n \log n)$, де n – кількість кіл. Кожна зупинка може збільшити кількість елементів у структурі перетину на 2 елементи, але структура сусідства не буде більшою за $2n$ елементів. Якщо операції пошуку та вставки в структурі перетину реалізовані за $O(\log m)$, де m – кількість елементів в структурі, то загальна складність алгоритму $O(n \log n)$.

Реалізувати структуру перетину можна збалансованими деревами, які забезпечать необхідну складність операцій вставки та пошуку.

6.4 Предикат перетину двох кіл

Коло – геометричне місце точок, рівновіддалених від центру. Два кола перетинаються, якщо мають хоча б одну спільну точку. Нагадаємо, що два кола можуть знаходитися в трьох принципово різних положеннях: перетинатися, бути вкладеними одне в одне та не перетинатися. Перевірити взаєморозташування двох кіл можливо за допомогою (а) розв'язати систему рівнянь, що складається з рівнянь обох кіл; (б) використовуючі значення радіусів кіл, та відстані між їх центрами.