Звіт

Геворгян Артем,
ІПС-32
Лабораторна робота з комп’ютерної графіки
Пошук точки у планарному графі за допомогою методу ланцюгів

Опис структури пошуку

Метою роботи алгоритма є побудова деревовидної структури даних, яка містить ланцюги, які визначають положення перевіюваної точки у планарному графі. Пошук здійснюється у готовій структурі, за наступною схемою. Ланцюги представлені вузлами у бінарному дереві, яке будується за допомогою нумерації як при обході за лівою стороною.

Приклад

```
    4
  2   6
1 3  5 7
```

Для обходу дерева у процесі пошуку необхідно знаходити батьківський вузол, а також дочірні вузли для заданого вузла x. Також треба знати, як рушити вліво та вправо від поточного вузла.

Опис побудови структури пошуку

Вхід алгоритму.

Набір точок у масиві, які представляють вершини графу, та ребра графу. Граф планарний та не містить точок, до яких не проведено сполучення із деякої точки, яка знаходиться вище за неї.

Крок 1. Відсортувати координати у порядку спадання за y-складовою, та перетворити граф ребер.

Крок 2. Побудувати row та column-списки. Для кожної вершини, перший список містить вказівники на всі ребра, які виходять із вершини, другий - на всі, які закінчуються у даній вершині. Також відсортувати ребра у кожному зі списків, для кожної вершини, за x-координатою. Оцінка та сама.

Крок 3. Побудувати відображення вершина -> вага, яке містить кількість ребер, які виходять із верхньої вершини даного ребра.

Крок 4. Побудова структури даних для пошуку точки у графі. Маркування регіонів у графі. Це етап побудови безпосередньої структури пошуку. Алгоритм працює із точками за порядком спадання їх y-координат. Точки мають бути посортовані за x як за другим ключем за зростанням.

Підтримуємо індекс ланцюга, який є мінімальним таким, що містить у собі точку, яка зараз розглядається.

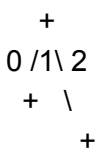
Перерахунок його відбувається наступним чином. На початку роботи 0, оскільки мінімальне ребро (у сенсі його положення у дереві пошуку), яке містить верхню точку, має індекс 0. А максимальне - індекс найвіддаленішого правого вузла. Але ми підтримуємо тільки мінімальний індекс.

Насправді нам потрібно розуміти мінімальний та максимальний індекс кожного ребра, яке виходить із даної точки, перше значення буде співпадати зі значенням, яке ми підтримуємо (для найлівішого ребра, що виходить із поточної вершини, для наступних ребер ми його перерахуємо), а максимальний ланцюг знайдемо так: додамо до нього значення, яке рівне кількості ребер, які покриваються усіма ланцюгами (для цього потрібна різниця суми ваг всіх вихідних ребер та всіх вхідних ребер для даної вершини), які містять поточне ребро.

Використовуючи функцію пошуку найближчого батьківського вузла у дереві для двох індексів, про які йде мова, знаходимо індекс найпершого ланцюга, який містить поточне ребро. Позначаємо цей зв’язок у структурі пошуку.

При обробці кожного ребра також маркуємо область зліва та справа від нього за допомогою цілих чисел.

Приклад



При роботі із ребрами будемо збільшувати значення поточного лівого регіону по відношенню до даного ребра, та виставляти значення правішого як на одиницю більше за значення лівого.

Після закінчення обробки усіх вершин (коли встановимо позначку 2 у даному прикладі), перейдемо до наступної точки (вершини). Оскільки немає таких ребер, до яких не веде ребро із такої, що знаходиться вище, то ми гарантовано записали для наступної точки, яку будемо обробляти, мінімальне значення та максимальне значення ланцюгів, які містять її (або, більш важливо, усі вихідні ребра тієї вершини), а також позначили області зліва та справа від кожного ребра. Ці значення ми використаємо на наступному кроці.

Пошук

Для пошуку візьмемо корінь дерева і будемо визначати, куди піти нижче по ньому, за допомогою звичайної процедури пошуку у бінарному дереві. Шукаємо проміжок між ланцюгами, у який потрібно спуститися. Зупинимося у листовому вузлі гарантовано або раніше. Для пошуку потрібно використовувати функції навігації по дереву,які описані раніше.

Складність: O (к-ть вершин * O(log(к-ть вершин))), (викор. т. Ейлера, к-ть ребер є O(к-ь вершин), сортування O(log (к-ть вершин))).