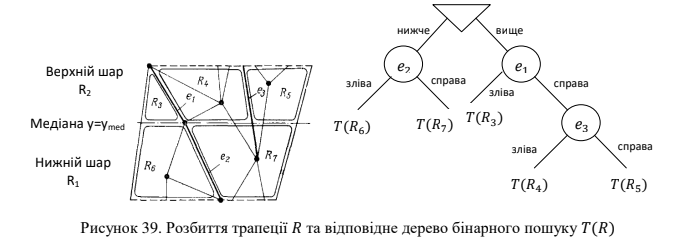
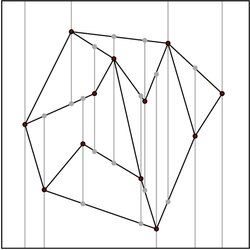
**Задача Локалізації точки**

**У найзагальнішому вигляді, завдання полягає в тому, щоб для заданого розбиття простору на області (вони не перетинаються), визначити область, якій належить точка запиту. Приклад застосування: після кожного клацання мишею на екрані монітору, проблема локалізації точки повинна бути вирішена для того, щоб визначити, яка область екрана комп'ютера знаходиться під вказівником миші.**

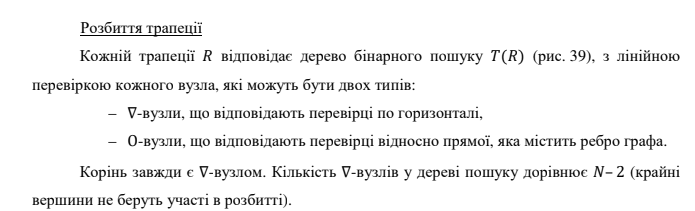
**Метод трапецій**

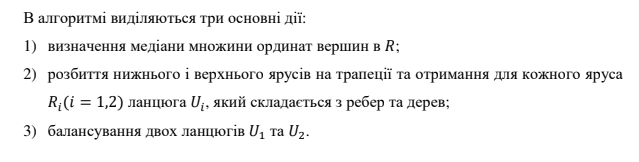
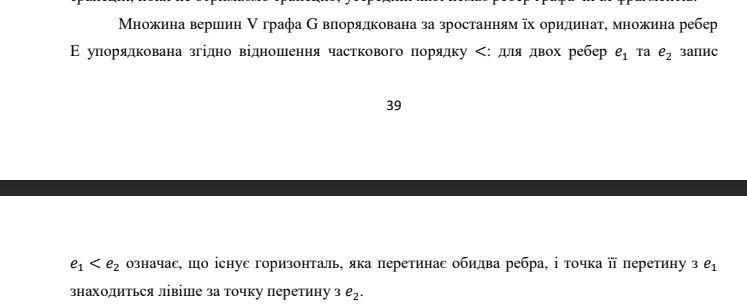


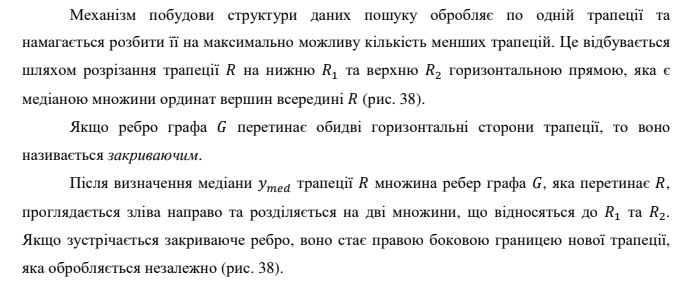
[**Увипадковлений підхід**](https://uk.wikipedia.org/wiki/%D0%A3%D0%B2%D0%B8%D0%BF%D0%B0%D0%B4%D0%BA%D0%BE%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC)**до задачі локалізації точки, і, ймовірно, найбільш практичний, заснований на трапецієподібному розкладанні або трапецієподібній карті. Трапецієподібне розкладання отримують, випускаючи вертикальні промені, що йдуть вгору і вниз з кожної вершини у вихідні підрозділи. Промені треба зупинити, коли вони потрапили в край, і сформувати новий край у підрозділі. Таким чином, ми отримаємо підмножину, яка є розкладанням в плиту, тільки з O(n) ребер і вершин, так як ми тільки додати два ребра і дві вершини для кожної вершини в первинному підрозділі.**

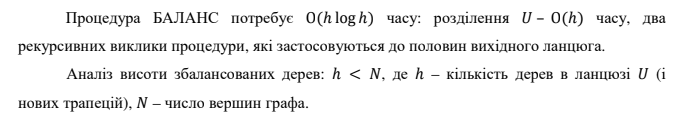
**Ми будуємо [орієнтований ациклічний граф](https://uk.wikipedia.org/wiki/%D0%A1%D0%BF%D1%80%D1%8F%D0%BC%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B9_%D0%B0%D1%86%D0%B8%D0%BA%D0%BB%D1%96%D1%87%D0%BD%D0%B8%D0%B9_%D0%B3%D1%80%D0%B0%D1%84" \o "Спрямований ациклічний граф), в якому вершини є трапеції, які існували в якийсь момент в уточненні, і спрямовані ребра з'єднують трапеції отримані за допомогою підрозділів.**

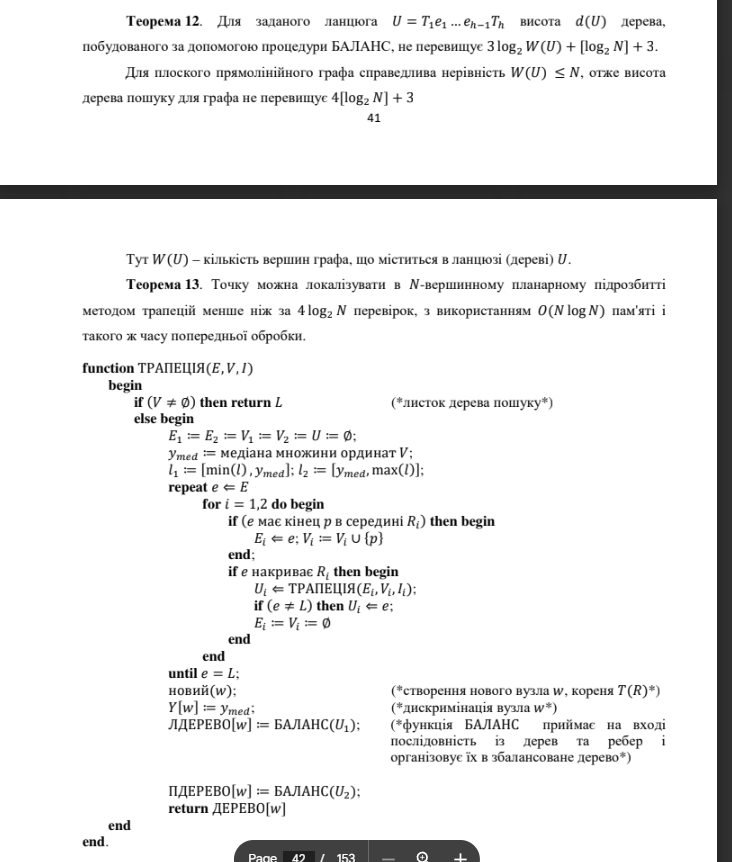
* preprocessing function (**Construction of the trapezoidal map)** is dominated by the sorting steps and is approximately O(n log n + m log m). **O(n log n).**
* point\_localization function is **O(log n)**, where n is the number of nodes in the tree.
* Використовується пам'ять �log⁡� **O(log n)**,

**Очікується, глибина пошуку в цьому орієнтованому графі, починаючи з вершини, яка відповідає** **прямокутнику, є O(log n).**







**Сортування ребер за значеннями y-координат вершин:**

**edges\_list = [tuple(sorted(list(e), key=lambda p: vertices\_coords[p][1])) for e in edges\_list]**

**У цьому рядку кожне ребро представляється у вигляді кортежу, що містить дві вершини. sorted(list(e), key=lambda p: vertices\_coords[p][1]) сортує вершини ребра за їхніми y-координатами, тобто за значеннями, які вони мають в vertices\_coords. Після цього створюється кортеж, що містить відсортовані вершини, що і є відсортованим ребром.**

**Створення списку кортежів (ребро, (вершина1, вершина2)):**

**edges\_list\_t = [(e, (vertices\_coords[e[0]], vertices\_coords[e[1]])) for e in edges\_list]**

**У цьому рядку створюється новий список кортежів edges\_list\_t. Кожен кортеж містить ребро (представлене як кортеж вершин) та координати цих вершин із vertices\_coords.**

**Сортування списку кортежів за допомогою функції порівняння \_edges\_comp:**

**edges\_list\_t = sorted(edges\_list\_t, key=functools.cmp\_to\_key(\_edges\_comp))**

**Використовується функція sorted() для сортування списку кортежів edges\_list\_t за допомогою ключа, який задається функцією порівняння \_edges\_comp. Для використання функції порівняння \_edges\_comp застосовується функція functools.cmp\_to\_key.**

**Оновлення початкового списку ребер:**

**edges\_list = [e\_t[0] for e\_t in edges\_list\_t]**

**У цьому рядку оновлюється початковий список ребер edges\_list, він заповнюється ребрами (першими елементами) з відсортованого списку edges\_list\_t.**

