ЛЕКЦІЯ 3

ГЕОМЕТРИЧНИЙ ПОШУК

Пошук- встановлення зв'язку між зразком і файлом.

3.1. Вступ в геометричний пошук.

Означення. Пошукове повідомлення, у відповідності з яким ведеться перегляд файлу , називають *запитом.*

Означення. Нехай є набір геометричних даних і необхідно дізнатись, чи мають вони певну властивість (скажемо, опуклість). У найпростішому випадку таке питання виникає один раз. Запит такого типу наз. *унікальним.* Запити, обробка, яких повторюється многократно на тому ж самому файлі наз. *масовими.*

Означення. *Передобробкою* будемо називати процес розташування даних у зручній для подальшої обробки структурі даних.

Міри оцінок пошуку :

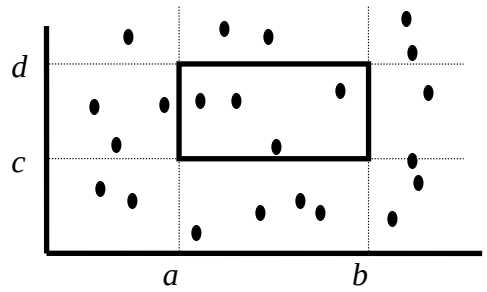
1. Час пошуку. Скільки часу треба як в середньому, так і в гіршому випадку для відповіді на одне запитання.?

2. *Пам'ять.* Скільки необхідно пам'яті для структури даних?

3. *Час попередньої обробки.* Скільки часу необхідно для організації даних перед пошуком?

4. *Час корегування.* Вказано елемент даних. Скільки треба часу на його включення в структуру даних або вилучення із неї?

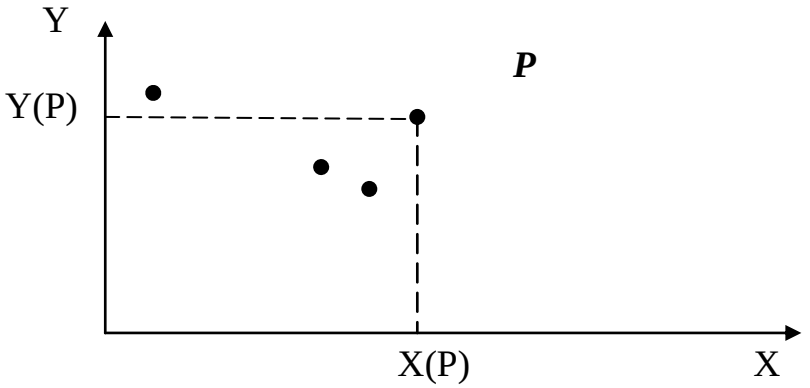
Задача П.1 (РЕГІОНАЛЬНИЙ ПОШУК-ПІДРАХУНОК).
Дано N точок на площині. Скільки із них лежить всередині заданого прямокутника, сторони якого паралельні координатним осям? Тобто, скільки точок (x, y) задовольняють нерівностям $a \le x \le b$, $c \le y \le d$ для заданих a, b, c і d (мал. 2.1)?



Мал. 2.1. Регіональний запит. Скільки точок всередині прямокутника

Метод локусів

Метод локусів в геометричних задах: запиту ставиться у відповідність точка в зручному для для пошуку просторі, а цей простір розбивається на області(локуси), в межах яких відповідь не змінюється. Підзадача, зв'язана з вершиною *p,* полягає у визначені числа точок Q(p) заданої множини, які задовольняють нерівностям $x \le x(p)$ $y \ge y(p)$, тобто кількості точок у лівому нижньому квадранті, яка визначається вершиною *p* (мал. 2.2).

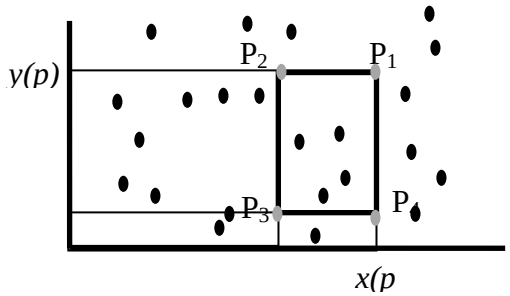


Мал.2.2. Скільки точок " південно -західніше" *p*?

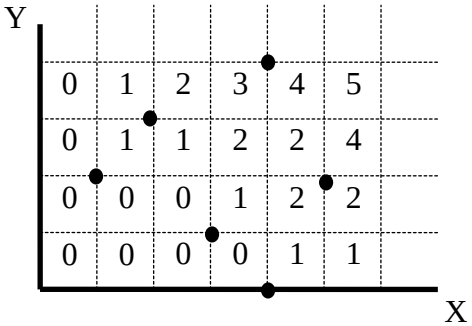
Означення. *Векторне домінування*. Кажуть, що точка(вектор) **v** *домінує* над **w**, якщо $\forall i$ виконується $v_i \ge w_i$. На площині точка **v** домінує над **w** тоді і лише тоді, коли **w** лежить у лівому нижньому квадранті, який визначається **v.** Q(p) - число точок, над якими домінує *p*.

Число точок $N(p_1p_2p_3p_4)$ у прямокутнику $p_1p_2p_3p_4$ визначається таким чином:

$$N(p_1p_2p_3p_4) = Q(p_1) - Q(p_2) - Q(p_4) + Q(p_3) \qquad (2.1)$$



Мал. 2.3. Регіональний пошук у формі чотирьох запитів про домінування.
(Наприклал: $Q(P_1) = 14$, $Q(P_2) = 9$, $Q(P_3) = 1$, $Q(P_4) = 2$. $N(P_1P_2P_3P_4) = 14 - 9 - 2 + 1 = 4$)



Мал..2.4. Прямокутна решітка для домінантного пошуку.

1. Після упорядкування початкових точок по обом координатам залишається виконати два двійкових пошуки (по одному для кожної осі), щоб знайти комірку, яка містить шукану точку. Час запиту буде рівним O(logN).
2. Маємо $O(N^2)$ комірок, тому необхідна квадратична пам'ять.
3. Визначення числа домінування для любої комірки можна зробити за час O(N), що веде до загальної витрати часу $O(N^3)$ на попередню обробку.

Дві головні моделі для задач геометричного пошуку:

Задача ГП1 (ЛОКАЛІЗАЦІЯ ТОЧКИ). Нехай задане N – вершинне розбиття простору G(V,E) (у випадку двовимірного простору – планарний граф) і точка Z. Визначити область розбиття, яка містить точку Z (Локалізувати точку Z на заданому розбитті).

Задача ГП2 (РЕГІОНАЛЬНИЙ ПОШУК). На заданій множині S із N точок задано запитний регіон R. Знайти підмножину точок множини S (або їх кількість), які містяться в регіоні R.

Файл в задачі ГП2 являє собою набір точок простору, а запитом є деяка стандартна геометрична фігура, яка довільно переміщується у цьому просторі.

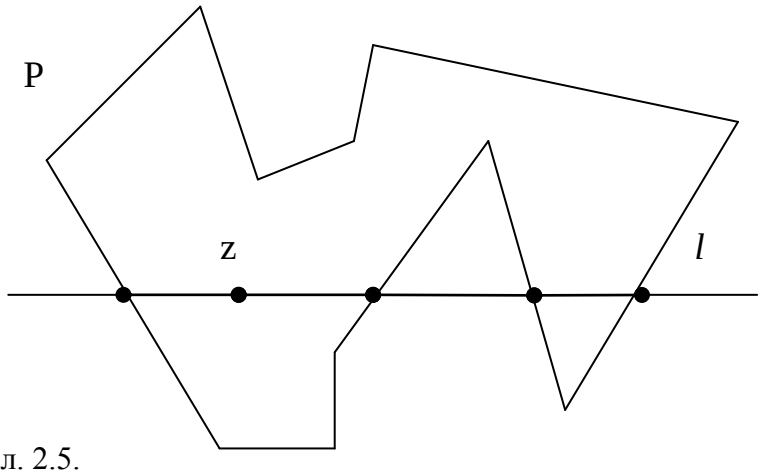
3.2. ЗАДАЧІ ЛОКАЛІЗАЦІЇ ТОЧКИ

Задача П2.(ПРИНАЛЕЖНІСТЬ ПРОСТОМУ МНОГОКУТНИКУ).Дано простий многокутник ***P*** і точка *z*; визначити чи знаходиться точка *z* всередині ***P***.
Задача П3.(ПРИНАЛЕЖНІСТЬ ОПУКЛОМУ МНОГОКУТНИКУ).Дано опуклий многокутник ***P*** і точка *z*; визначити чи знаходиться точка *z* всередині ***P***.

Теорема 3.1 *Приналежність точки z внутрішній області простого N-кутника **P** можна встановити за час O(n) без передобробки.*

Алгоритм

Проведемо через точку горизонталь *l* (мал. 2.5)



Мал. 2.5.

По теоремі Жордана зовнішня і внутрішні області ***P*** добре визначені.
1.Якщо *l* не перетинає ***P*** - то *z* зовнішня точка.
2.Нехай *l* перетинає ***P.***
а) *випадок, коли l не проходить ні через жодну із вершин* ***P*** .Нехай *L*-число точок перетину *l* з границею ***P*** ліворуч від *z*. *z* лежить всередині тоді і лише тоді, коли *L* непарне.
б) *вироджений випадок, коли l проходить через вершини* ***P.***
Нескінчено малий поворот *l* навколо *z* проти часової стрілки ліквідує виродженість:
1) якщо обидві вершини ребра належать *l* , то це ребро відкидається;
2) якщо рівно одна вершина ребра лежить на *l*, то перетин враховується, коли ця вершина з великою ординатою, і ігнорується в противному випадку.
Алгоритм виконуються за час O(N).

begin L:=0;

for *i:*=1 **until** N **do if** ребро (*i*) не горизонтально

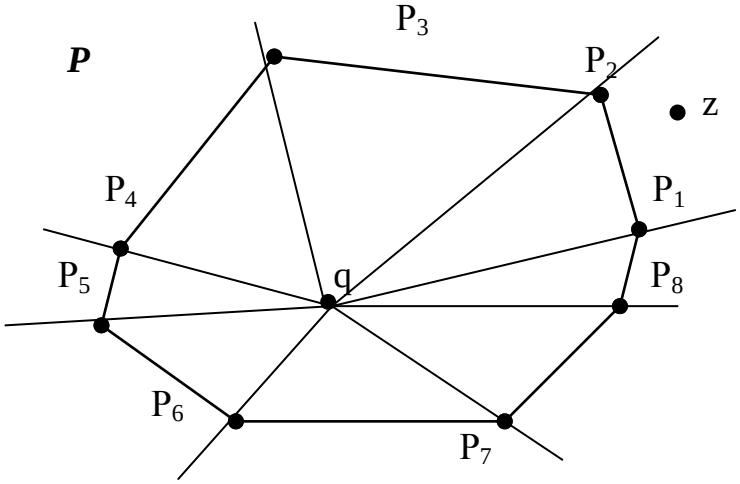
then if (ребро *(i)* перетинає *l*нижнім кінцем ліворуч від *z*)

then L:=L+1;

if (L непарне) **then** *z* всередині **else** *z* зовні

end

Для масових запитів розглянемо випадок, коли *P* - опуклий многокутник.
q - довільна внутрішня точка, z – шукана точка. За точку q можна,наприклад, взяти центр мас (центроїд) трикутника, утвореного любою трійкою вершин ***P:*** $x_q = (x_{p1} + x_{p2} + x_{p3})/3$, $y_q = (y_{p1} + y_{p2} + y_{p3})/3$



Мал. 2.6.

Попередня обробка - O(N) для заданої послідовності $p_1,p_2, \ldots, p_N$.

Процедура ПРИНАЛЕЖНІСТЬ ОПУКЛОМУ МНОГОКУТНИКУ

1. Дана пробна точка z. Визначаємо методом бінарного пошуку клин, в якому вона лежить. Точка z лежить між променями, які визначаються p_i і $p_{i+1} \Leftrightarrow \angle zqp_{i+1} > 0$, $\angle zqp_i < 0$, (час O(*logN*)).

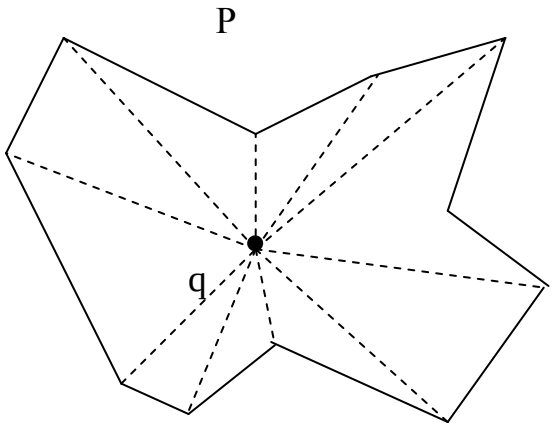
2. Якщо p_i і p_{i+1} знайдені, то z - внутрішня точка тоді і лише тоді, коли кут ($p_ip_{i+1}z$) від'ємний (час O(1)). (*q* та *z* по одну сторону від p_ip_{i+1}).
Якщо покласти $p_i=(x_i,y_i)$, то визначник, рівний

$$2\delta S = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}$$

дає подвоєну *орієнтовану площу* трикутника ($p_1p_2p_3$), де "+" $\Leftrightarrow$, коли обхід ($p_1p_2p_3$) орієнтований проти часової стрілки(лівий поворот) і "-" $\Leftrightarrow$, коли обхід ($p_1p_2p_3$) орієнтований за часовою стрілкою(правий поворот).

Терема 3.2. *Час відповіді на запит про приналежність точки опуклому N-кутнику рівний O(logN) при витраті O(N) пам'яті і O(N) часу на попередню обробку.*

Зірковий многокутник ***P*** містить принаймні одну точку q таку, що [q, p_i] лежить повністю в середині многокутника ***P*** для любої вершини p_i із ***P,i***=1,…,N, (мал. 2.7). Множина шуканих центрів всередині ***P*** є ядром ЗМ.



Мал. 2.7. Зірковий многокутник.

Терема 3.3. *Час відповіді на запит про приналежність точки зірковому N-кутнику рівний O(logN) при витраті O(N) пам'яті і O(N) часу на попередню обробку.*

3.3. ЛОКАЛІЗАЦІЯ ТОЧКИ НА ПЛАНАРНОМУ РОЗБИТТІ

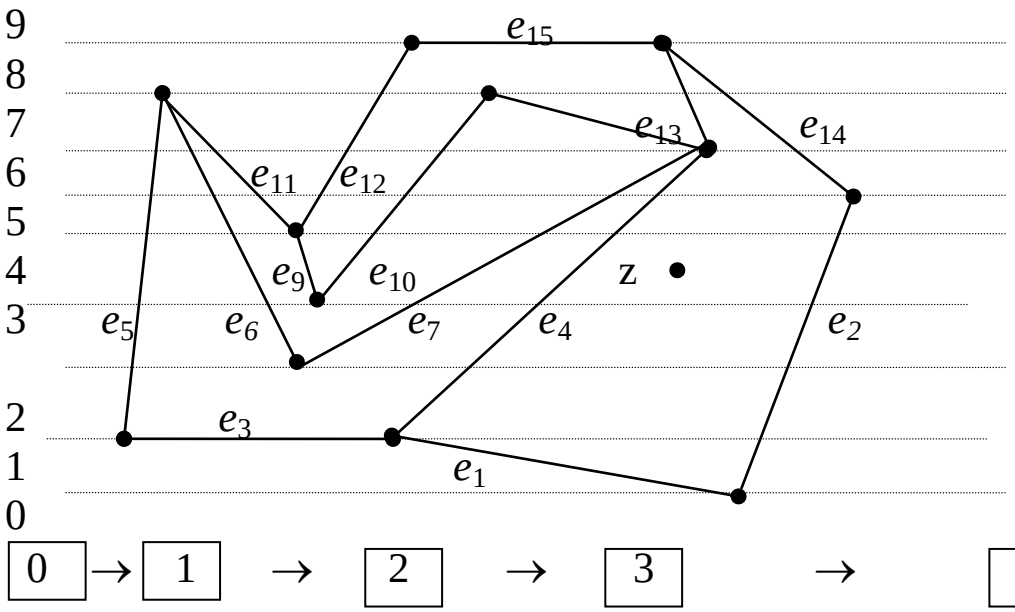
Означення. Планарний граф завжди може бути розміщений на площині тиким чином, що його ребра стануть прямолінійними відрізками. Графи, розміщені таким чином, будуть називатись *плоскими прямолінійними графами*.

Г*оловна ідея полягає* в тому, щоб *створити нові геометричні об'єкти, які дозволяють двійковий пошук.*

3.3.1. Метод смуг

Нехай задано плоский прямолінійний граф G. Проведемо горизонтальні прямі через кожну його вершину. Вони розділяють площину на (N+1) горизонтальні смуги. Якщо провести сортування цих смуг по координаті y, то можна знайти ту смугу, в якій лежить точка z, за час O(log N) (Мал.2.8).

Враховуючи упорядкованість відрізків для визначення тієї трапеції, в яку потрапила точка z, можна скористатись двійковим пошуком з часом O(log N). Звідси отримаємо оцінку часу запиту для найгіршого випадку: O(log N).



0	→	1	→	2	→	3	→	4	. . .
$\{\varnothing\}$		$\{e_1, e_2\}$		$\{e_5, e_3, e_4, e_2\}$		$\{e_5, e_6, e_7, e_4, e_2\}$		$\{e_5, e_6, e_9, e_{10}, e_7, e_4, e_2\}$	
вст.		вст.		вст.		вст.		вст.	
$\{\varnothing\}$		$\{e_1, e_2\}$		$\{e_5, e_3, e_4\}$		$\{e_6, e_7\}$		$\{e_6, e_7\}$	
вил.		вил.		вил.		вил.		вил.	
$\{\varnothing\}$		$\{\varnothing\}$		$\{e_1\}$		$\{e_3\}$		$\{\varnothing\}$	

Попередня обробка . Сортуються усі відрізки всередині кожної смуги: $O(N^2 \log N)$ - для часу і $O(N^2)$ - для пам'яті. Час попередньої обробки можна скоротити до $O(N^2)$. Однак існують такі плоскі прямолінійні графи, які потребують квадратичної пам'яті (Мал.2.10).

Рис. 2.10. Сумарне число відрізків на смугах може досягати $O(N^2)$.

Метод плоского замітання (МПЗ).

Метод плоского замітання характеризується основними структурами даних:
1) *списком точок подій* - послідовністю позицій, які призначені для замітаючої прямої;
2) статусом замітаючої прямої - опис перетину замітаючої прямої з геометричним об'єктом, який замітають.

У методі смуг *список точок подій* - це просто *перелік* знизу вгору смуг, які відповідають *вершинам* плоского прямолінійного графу. *Статус* - це *послідовність відрізків(ребер графа)* всередині смуги, що розташована вище**.**

Витрати ресурсів: а) вставка та вилучення кожного із ребер графа - складність O(log N) на одну операцію; б) запам'ятовування статусу. (На першу операцію піде O(N log N) часу; на другу - $O(N^2)$).

Теорема 3.4. *Локалізацію точки в N-вершинному планарному розбитті можна реалізувати за час O(log N) з використанням $O(N^2)$ пам'яті, якщо $O(N^2)$ часу пішло на обробку.*

3.3.2. Метод ланцюгів

Означення. *Ланцюгом $C = (u_1,..., u_p)$ називається плоский прямолінійний граф з вершинами $\{u_1,..., u_p\}$і ребрами $\{(u_i,..., u_{i+1}): i = 1,... p-1\}$.*

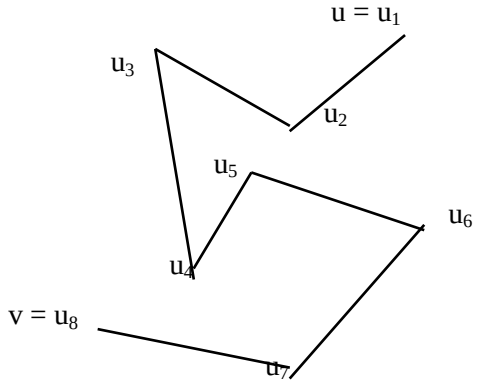


Рис. 2.11(а). Ланцюг загального виду.

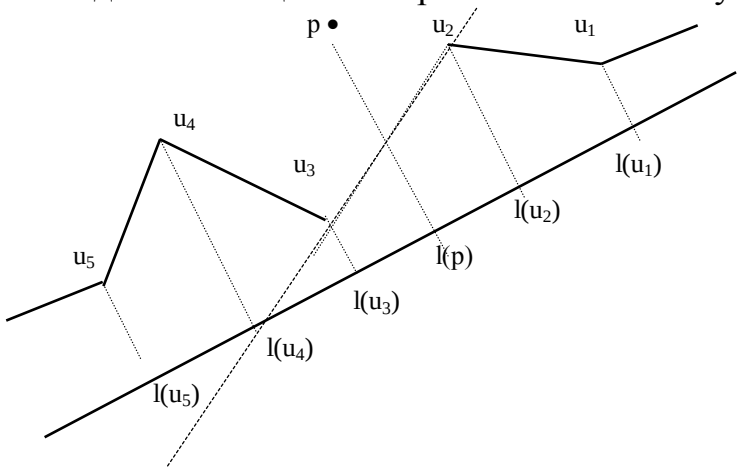
Розглянемо планарне підрозбиття, яке визначається ППЛГ ***G***. Припустимо, що в ***G*** знайдений ланцюг ***C*** (підграф ***G***) одного із типів: 1) ***C*** є циклом ; 2) обидва кінці u_1 і u_p із ***C*** лежать на границі нескінченої області. В останньому випадку додамо ***C*** з обох кінців напівнескінченими паралельними протилежно напрямленими ребрами. Ланцюг кожного типу ділить початкове підрозбиття на дві частини.

Означення. *Дискримінацією точки* ***z*** *відносно ланцюга* ***C*** називається процедура визначення того, по яку сторону від ***C*** лежить пробна точка z.

Означення. *Ланцюг* $C = (u_1,..., u_p)$ *називається монотонним по відношенню до прямої l, якщо будь-яка пряма, ортогональна до l, перетинає* ***C*** *тільки в одній точці.*

Означення. *Простий многокутник називається монотонним, якщо його границю можна розбити на два ланцюга, монотонних відносно однієї прямої.*

Один із подібних ланцюгів зображено на малюнку 2.11(b):



Мал.11(b).

$L(u) = [l(u_1), l(u_2), l(u_3), l(u_4), l(u_5)]$, $l(p)$- проекція на l точки p.

1.O(logN)- локалізація $l(p)$ на $L(u)$.

*2.O(*1*)*- по яку сторону відносно ланцюга розташована точка p.

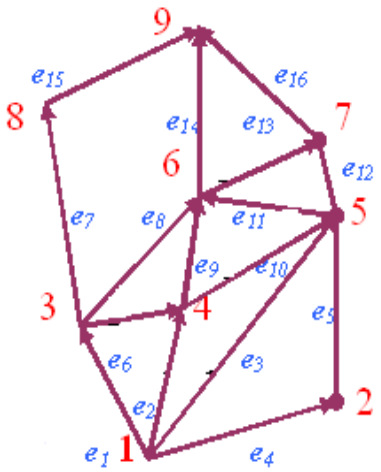
Означення.Припустимо, що існує деяка множина $\boldsymbol{Z}=\{C_1,\ldots,C_r\}$ ланцюгів, монотонних відносно однієї і тієї ж прямої l і яким властиві такі властивості:

Властивість **1.***Об'єднання усіх елементів* ***Z*** *містить заданий ППЛГ* ***G.***
Властивість **2.** *Для будь-якої пари ланцюгів* C_i *і* C_j *із* ***Z*** *ті вузли* C_i, *які не являються вузлами* C_j *, лежать по одну сторону від* C_j.

Тоді множина ***Z*** наз. *повною множиною монотонних ланцюгів графа* ***G.***

Якщо є r ланцюгів в ***Z*** і у найдовшому ланцюгу p вершин , то пошук займе у найгіршому випадку O(*logr .logp*) часу.

Означення. Нехай G - плоский прямолінійний граф з множиною вершин $\{v_1,..., v_N\}$, де вершини індексовані так, що i < j тоді і тільки тоді, коли $y(v_i) < y(v_j)$, або $y(v_i) < y(v_j)$ і $x(v_i) > x(v_j)$. ***Вершина*** $\mathbf{v_j}$ ***називається регулярною,*** **якщо існують такі цілі i < j < k, що (v_i, v_j) і (v_j, v_k) - ребра G. Говорять, що плоский прямолінійний граф G регулярний, якщо кожна вершина регулярна при 1 < j < N (за виключенням двох крайніх вершин v_1 та v_N).**



Мал.2.12 Приклад регулярного плоского прямолінійного графу.

Позначимо через IN(v_j) та OUT(v_j) множини ребер, які входять та виходять з вершини v_j. Нехай ребра в IN(v_j) впорядковані за кутом проти годинникової стрілки, а ребра в OUT(v_j) – за годинниковою стрілкою.

Теорема 2.5. *Для довільної v_j можна побудувати y - монотонний ланцюг від v_1 до v_j.*

Доведення. Методом математичної індукції:1) для j = 2; 2) k < j; 3) v_j регулярна, то ∃ i < j, що (v_i, v_j) ∈ G; 4)∃ ланцюг C від v_1 до v_i, монотонний відносно осі *y* і його з'єднання з ребром (v_i, v_j) дасть також монотонний ланцюг.

Теорема 2.6. *Довільний регулярний граф можна розбити на повну множину ланцюгів, монотонних відносно осі y.*

Доведення. Позначимо через W(*e*) вагу ребра *e* – кількість ланцюгів, яким належить *e*. Введемо позначення:

$$W_{IN}(v) = \sum_{e \in IN(v)} W(e),\ W_{OUT}(v) = \sum_{e \in OUT(v)} W(e),$$

Ваги ребер обираються так, щоб виконувалися умови:
1. кожне ребро має додатну вагу;
2. для довільного v_j (j ≠ 1, N) $W_{IN}(v_j) = W_{OUT}(v_j)$.

Теорема 2.7. *Реалізація умови $W_{IN} = W_{OUT}$ є розв'язком потокової задачі і може бути досягнута за два проходи по графу G.*

Покладемо спочатку W(*e*) = 1 для кожного ребра *e*. При першому проході від v_1 до v_N отримаємо $W_{IN}(v_i) \le W_{OUT}(v_i)$ для всіх не крайніх v_i. При другому проході від v_N до v_1 отримаємо $W_{IN}(v_i) \ge W_{OUT}(v_i)$. Отже після двох проходів матимемо реалізацію умови $W_{IN}(v_i) = W_{OUT}(v_i)$. Позначимо $v_{IN}(v) = |\ IN(v)|$, $v_{OUT}(v) = |\ OUT(v)|$.

Алгоритм балансування по вазі

procedure БАЛАНСУВАННЯ ПОВАЗІ В РЕГУЛЯРНОМУ ППЛГ

begin for кожного ребра e **do** W(e):= 1 (* ініціалізація *)

for i:= 2 **until** N - 1 **do**

begin $W_{IN}(v_i)$:= сума вагів ребер, які входять в v_i;

d_1:= крайнє зліва ребро, яке виходить із v_i;

if ($W_{IN}(v_i) > \gamma_{OUT}(v_i)$) **then** $W(d_1):= W_{IN}(v_i) - \gamma_{OUT}(v_i) + 1$

end (* перший прохід *);

for i:= N - 1 **until** 2 **do**

begin $W_{OUT}(v_i)$:= сума вагів ребер, які виходять із v_i;

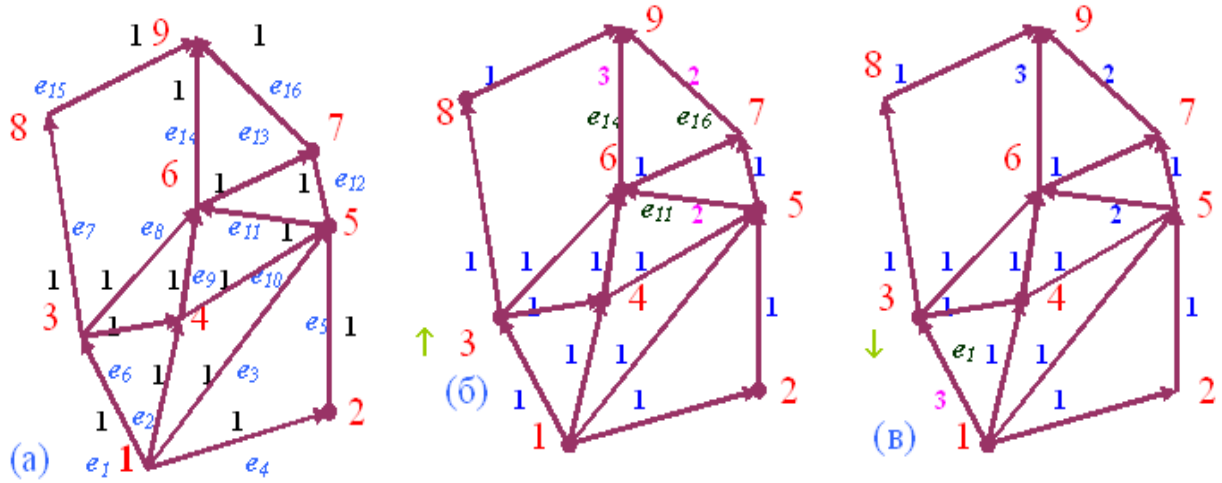
d_2:= крайнє зліва ребро, яке входить в v_i;

if ($W_{OUT}(v_i) > W_{IN}(v_i)$) **then** $W(d_2):= W_{OUT}(v_i)-W_{IN}(v_i)+W(d_2)$

end (* другий прохід *)

end.

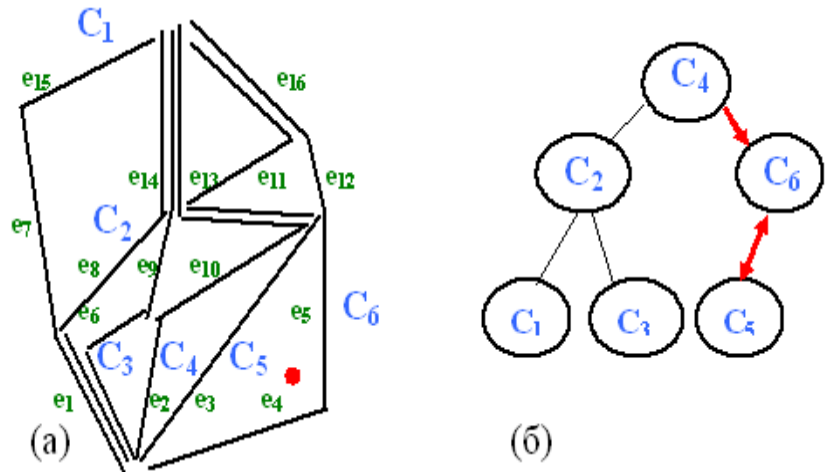
На мал. 2.13(а)-(в) показано конфігурацію ваги ребер після ініціалізації, першого та другого проходів відповідно.



Мал. 2.13. Алгоритм балансування: а) для кожного ребра W(e):= 1 (* ініціалізація *);
б) перевірка умови $W_{IN}(v_i) > W_{OUT}(v_i)$ - **(5)**:($W_{IN}(5) = 3 > \gamma_{OUT}(5) = 2$) $\Rightarrow W(e_{11})$:= 3-2 + 1=2; **(6)**: ($W_{IN}(6) = 4 > \gamma_{OUT}(6) = 2$) $\Rightarrow W(e_{14})$:= 4-2 + 1=3; **(7)**: ($W_{IN}(7) = 2 > \gamma_{OUT}(7) = 1$) $\Rightarrow W(e_{16})$:= 2-1 + 1=2;
в) перевірка умови $W_{IN}(v_i) < W_{OUT}(v_i)$ - **(3)**:($W_{OUT}(3) = 3 > W_{IN}(3) = 1$) $\Rightarrow W(e_{11})$:= 3-1 + 1=3.

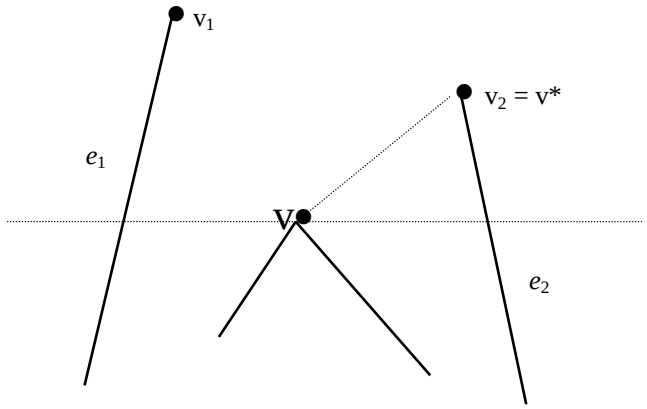
Тоді можна виписати повну множину монотонних ланцюгів Z відносно осі OY:

$C_1 = (e_1, e_7, e_{15})$, $C_2 = (e_1, e_8, e_{14})$, $C_3 = (e_1, e_6, e_9, e_{14})$,
$C_4 = (e_2, e_{10}, e_{11}, e_{14})$, $C_5 = (e_3, e_{11}, e_{13}, e_{16})$, $C_6 = (e_4, e_5, e_{12}, e_{16})$.
$Z = (C_1, C_2, C_3, C_4, C_5, C_6)$ – повна множина монотонних ланцюгів, мал. 2.14 (а), а відповідне дерево пошуку мал.2.14(б).



2.14. Повна множина монотонних ланцюгів та відповідне бінарне дерево.

Регуляризація графа



Мал.2.14 Приклад типової нерегулярної вершини v.

Теорема 2.8. *Довільний планарний граф можна перетворити в регулярний.*

Використовуючи алгоритм плоского замітання, замітаємо граф згори вниз регуляризуючи вершини, які не мають вихідних ребер, а потім знизу вгору для регуляризації вершин які не мають вхідних ребер.

В процесі замітання для кожної вершини v реалізуємо наступні операції:

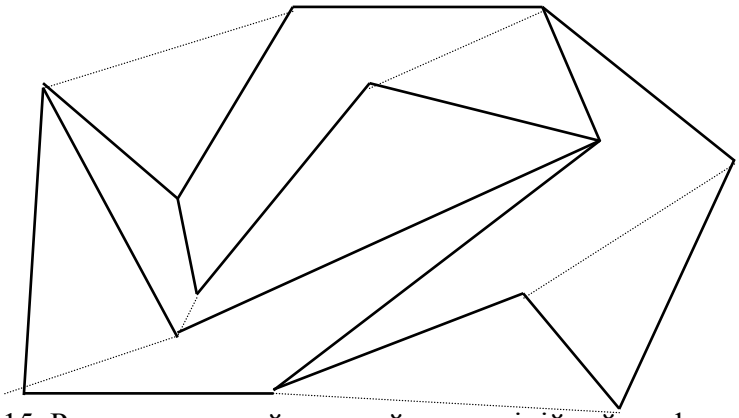
(1) локалізуємо v (по абсцисі) в одному із інтервалів в структурі даних статусу;

(2) корегуємо цю структуру статусу;

(3) якщо v нерегулярна, тоді добавляємо ребро від v до тієї вершини, яка зв'язана з інтервалом, який визначений в операції (1).

Теорема 2.9. *N-вершинний плоский прямолінійний граф можна регуляризувати за час O(N log N) з витратою пам'яті O(N).*

Попередня обробка . Припустимо, що ППЛГ заданий у вигляді структури даних РСПЗ. Побудова множин IN(v) і OUT(v) - O(N). Процедура балансування по вазі - O(N). Регуляризація - O(N log N) часу, отже час попередньої обробки займає O(N log N) .



Мал. 2.15. Регуляризований плоский прямолінійний граф.

Теорема 2.10. *Локалізацію точки в N-вершинному планарному підрозбитті можна реалізувати за час $O(\log^2 N)$ з використанням O(N) пам'яті при витратах O(N log N) часу на попередню обробку.*

3.3.3. Метод деталізації тріангуляції

Тріангуляція на множині вершин V на площині є плоским прямолінійним графом з максимальним числом ребер, що не перевищує 3| V | - 6 (по формулі Ейлера).

Нехай задана N-вершинна тріангуляція G, і вважаємо, що будується послідовність триангуляцій $S_1, S_2, \ldots, S_{h(N)}$, $S_1 = G$, а S_i отримується з S_{i-1} за такими правилами:

Крок (1). Вилучимо деяку множину несуміжних неграничних вершин S_{i-1} і інцендентні їм ребра.

Крок (2). Триангулюємо многокутники, які утворились в результаті вилучення вершин та ребер.

Таким чином, $S_{h(N)}$ не має внутрішніх вершин.

Усі тріангуляції $S_1, S_2, \ldots, S_{h(N)}$ мають одну загальну границю, оскільки на кроці (1) ми вилучали лише внутрішні вершини. Трикутники будемо позначати літерою R з індексами. Трикутник R_j може з'явитися в багатьох тріангуляціях, але R_j належить тріангуляції S_i (позначимо $R_j \in^* S_i$), якщо R_j створений на кроці (2) при побудові S_i.

Структура T, топологію якої представляє направлений ациклічний граф, визначається наступним чином: від трикутника R_k до трикутника R_j проводиться дуга, якщо при побудові S_i після S_{i-1} ми маємо:

1) R_j видаляється з S_{i-1} на кроці (1);

2) R_k створюється в S_i на кроці (2);

3) $R_j \cap R_k \neq \varnothing$.

Критерій вибору вершин. Порядок перегляду та вилучення вершин наступний: почнемо з довільної вершини, помічаємо її сусідів (які не можуть вилучатися на поточному кроці) і продовжуємо доти поки ще є непомічені сусіди.

Вважатимемо, що можна вибрати множину так, щоб виконувались наступні властивості (через N_i позначено число вершин в S_i):

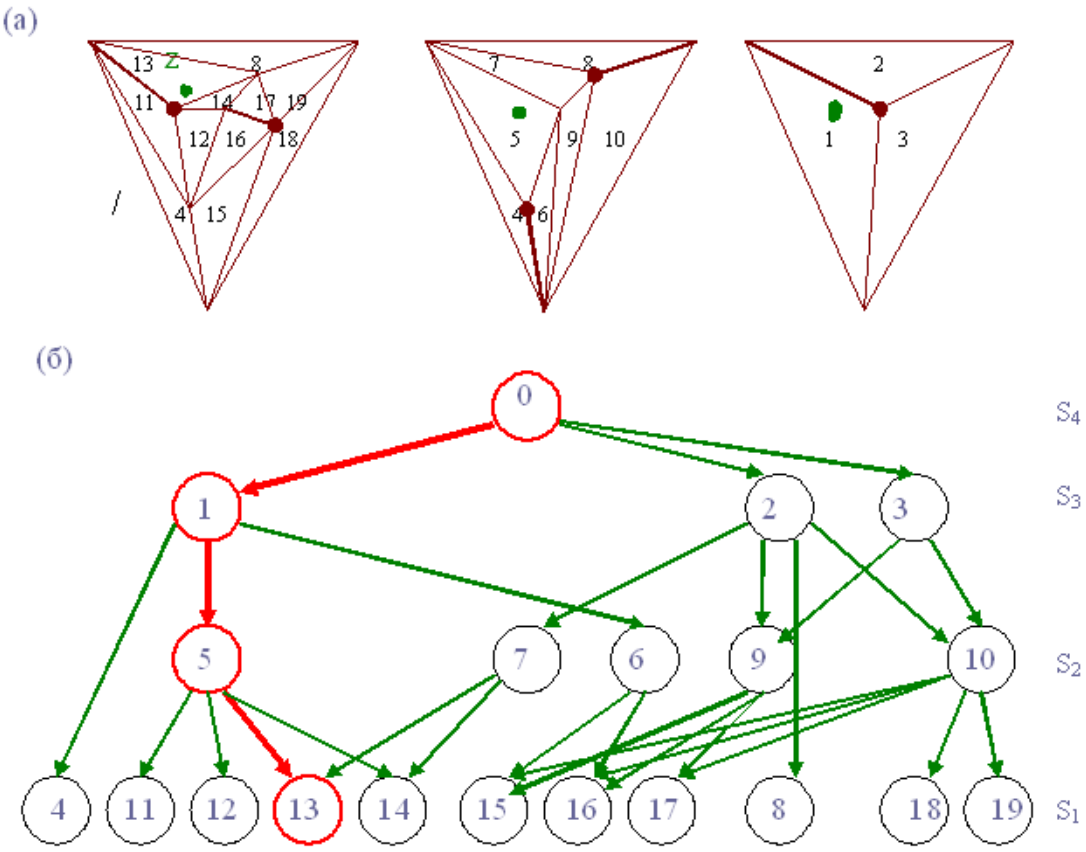
Властивість 1. $N_i = \alpha_i N_{i-1}$, *де* $\alpha_i \le \alpha < 1$ *для* $i = 2, \dots, h(N)$.

Властивість 2. *Кожний трикутник* $R_j \in S_i$ *перетинається не більше чим з* H *трикутниками з* S_{i-1}, *і навпаки.*

З властивості 1 $\Rightarrow$ наслідок, що $h(N) \le \lceil \log_{1/\alpha} N \rceil = O(\log N)$, оскільки при переході від S_{i-1} до S_i вилучається фіксована доля вершин. Із властивостей 1 та 2 випливає, що пам'ять для T рівна O(N). Ця пам'ять використовується для збереження вузлів і вказівників на їх нащадків. З теореми Ейлера про плоскі графи випливає, що S_i містить $F_i < 2N_i$ трикутників. Число вузлів T, які представляють трикутники із S_i не перевищує F_i. Звідси випливає, що загальне число вузлів T меньше, ніж

$$2(N_1 + N_2 + \dots + N_{h(N)}) \le 2N_1(1 + \alpha + \alpha^2 + \dots + \alpha^{h(N)-1}) < 2N / (1 - \alpha)$$

Що до пам'яті для вказівників: по властивості 2 кожен вузол має $\le$ H вказівників $\Rightarrow \le 2NH / (1 - \alpha)$ вказівників з'явиться в T.



Мал. 2.19. Послідовність триангуляцій (а) і відповідний їй направлений ациклічний граф пошуку (б).

Теорема 2.11. *Локалізацію точки в N-вершинному планарному підрозбитті можна зробити за час O(log N) з використанням O(N) пам'яті, якщо O(N log N) часу пішло на попередню обробку.*

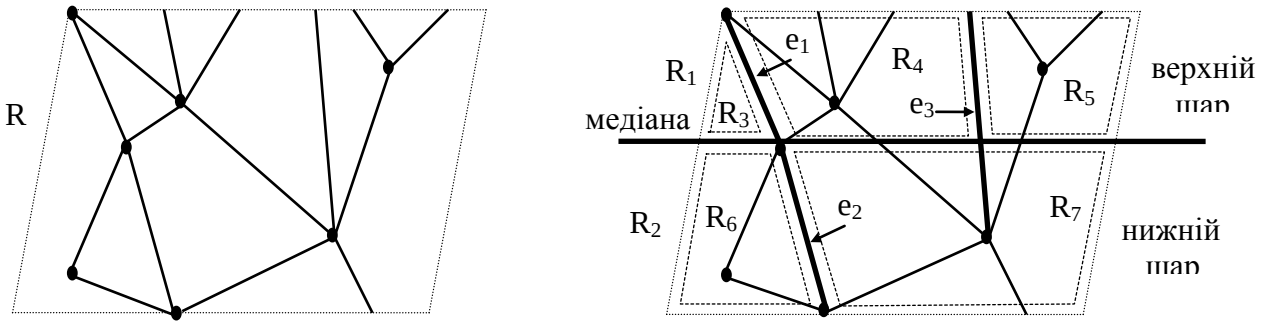
3.3.4. Метод трапецій

Означення. Дано два ребра e_1 та e_2 з E. Запис $e_1 < e_2$ означає що існує горизонталь, яка перетинає обидва ребра, і точка її перетину з e_1 знаходиться лівіше за точку перетину з e_2.

Механізм побудови структури даних пошуку обробляє по одній трапеції та намагається розбити її на максимально можливу кількість менших трапецій. Це відбувається шляхом розрізання трапеції R на нижню R_1 та верхню R_2 горизонтальною прямою, яка є медіаною множини ординат вершин всередині R.

Означення. Якщо ребро графа G перетинає обидві горизонтальні сторони трапеції, то воно називається *накриваючим*.

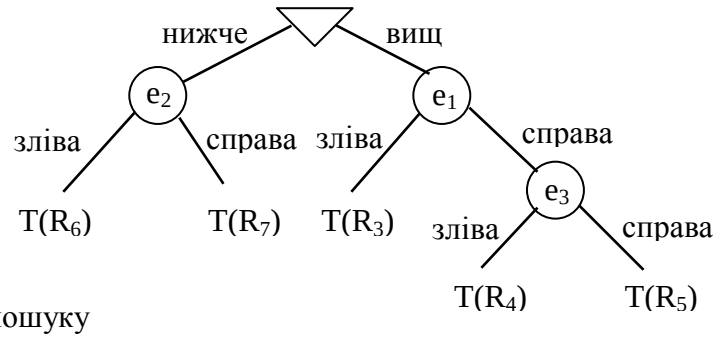
Після визначення медіани y_{med} трапеції R множина ребер графа G, яка перетинає R, проглядається зліва направо та розділяється на дві множини, що відносяться до R_1 та R_2. Якщо зустрічається накриваюче ребро воно стає правою боковою границею нової трапеції, яка обробляється незалежно.



Мал.2.20

Розбиття трапеції

Кожній трапеції R відповідає дерево бінарного пошуку T(R), з лінійною перевіркою кожного вузла, які можуть бути двох типів: ∇ - вузли, що відповідають перевірці по горизонталі, та O - вузли, що відповідають перевірці відносно прямої, яка містить ребро графа. Корінь завжди є ∇ - вузлом. Кількість ∇ - вузлів у дереві пошуку = N - 2.



Мал. 21. Дерево пошуку

Алгоритм

В алгоритмі виділяються три основні дії:

1) визначення медіани множини ординат вершин в R;
2) розбиття нижнього і верхнього ярусів на трапеції і отримання для кожного яруса R_i (i= 1, 2) ланцюга U_i, який складається з ребер та дерев;
3) балансування двох ланцюгів U_1 та U_2.

Процедура БАЛАНС потребує O(h log h) часу: розділення U - O(h) часу, два рекурсивних виклики процедури, які застосовуються до половин вихідного ланцюга.

Аналіз висоти збалансованих дерев. h < N, h - число дерев в ланцюгу U і нових трапецій, N число вершин графа. Якщо многокутник має s вершин, то в ньому ≤ s - 3 діагоналей. Оскільки s ≤ N, то h ≤ s - 2 ≤ N - 2.

Теорема 2.12. *Для заданого ланцюга* $U = T_1 e_1 \ldots e_{h-1} T_h$ *висота* $\delta(U)$ *дерева, побудованого за допомогою процедури БАЛАНС, не перевищує* $3 \log W(U) + \lceil \log_2 N \rceil + 3$.

Для плоского прямолінійного графа справедлива нерівність W(U) ≤ N⇒ висота дерева пошуку для графа не перевищує $4\lceil \log_2 N \rceil + 3$.

Теорема 2.13. *Точку можна локалізувати в N-вершинному планарному підрозбитті менше ніж за* $4 \log_2 N$ *перевірок, з використанням O(N log N) пам'яті і такого ж часу попередньої обробки.*

```
function ТРАПЕЦІЯ(E, V, I)
begin if (V = ∅) then return Λ (*листок дерева пошуку*)
      else begin E1:= E2:= V1:= V2:= U:= ∅;
                 ymed:= медіана множини ординат V;
                 I1:= [min (I), ymed]; I2:= [ymed, max (I)];
                 repeat e ⇐ E;
                        for i = 1,2 do
                                  begin if (e має кінець p всередині Ri)
                                              then
                                               begin Ei ⇐ e;
                                                     Vi:= Vi ∪ {p}
                                              end;
                                        if (e накриває Ri)
                                          then begin Ui ⇐ ТРАПЕЦІЯ(Ei, Vi, Ii);
                                                       if (e ≠ Λ)
                                                         then Ui ⇐ e;
                                                  Ei:= Vi:= ∅
                                                end
                                  end
                 until e = Λновий (w); (*створення нового вузла w, кореня T(R)*)
                       Y[w]:= ymed; (*дискримінація вузла w*)
                       ЛДЕРЕВО[w]:= БАЛАНС(U1);
```

```
        ПДЕРЕВО[w]:= БАЛАНС(U2);
        (*ф-ція БАЛАНС приймає на вході послідовність із
        дерев та ребер і організовує їх в збалансоване дерево*)
        return ДЕРЕВО[w]
    end
end.
```

Вершини трапеції розташовуються в масиві по зростанню ординат. Модифікація процедури ТРАПЕЦІЯ проходить наступним чином. Послідовність ребер проглядається двічі. При першому проході кожна вершина просто помічається іменем тієї трапеції, до якої вона належить, будується масив вершин для кожної породженої трапеції. При другому проході виконується цикл *repeat.*