ЛЕКЦІЯ 5

ЗАДАЧІ ПОБУДОВИ ОПУКЛОЇ ОБОЛОНКИ

4.1. Постановка задач опуклої оболонки.

У відповідності з визначенням ОО- це найменша опукла множина, яка містить S.

Означення 5.1 Нехай у просторі E^d задано k різних точок p_1, p_2, ... , p_k. Множина точок

$$p = \alpha_1 p_1 + \alpha_2 p_2 + ... + \alpha_k p_k$$

$$(\alpha_j \in R, \alpha_j \geq 0, \alpha_1 + \alpha_2 + ... + \alpha_k = 1)$$

називається *опуклою множиною,* яка породжена точками p_1, p_2, ... , p_k, а p називається *опуклою комбінацією.*

Означення 5.2. *Опуклою оболонкою* ***conv(L)*** підмножини L називається найменша опукла множина, яка містить L.

Щоб охарактеризувати структуру conv(L) скінченої множини точок L, необхідно узагальнити поняття опуклого многокутника та опуклого многогранника.

Означення 5.3. *Поліедральною множиною* називається перетин скінченої множини замкнутих півпросторів.

Зауваження. Поліедральна множина є опукла, так як півпростір являється опуклою множиною й перетин опуклих множин теж є опуклою множиною. В загальному випадку скінчену d- мірну поліедральну множину називають *опуклим d- політопом* (або просто політопом).

Теорема. 5.1. *Опукла оболонка скінченої множини точок в E^d є опуклим політопом; і навпаки кожен опуклий політоп є опуклою оболонкою деякої скінченної множини точок.*

Опуклий політоп задається описом своїх границь, які складаються з *граней*. Кожна грань опуклого політопа є опуклою множиною; *k- грань* означає k- вимірну грань. Якщо політоп P має розмірність d, то його (d - 1) грані називаються *гіпергранями*, (d - 2)- грані називаються *підгранями,* 1 - грані - *ребрами*, а 0-грані- *вершинами.*

Число F(d, N) гіперграней d-політопа з N вершинами може набувати таких значень:

$$F(d, N) = \begin{cases} 2N \binom{N - d/2 - 1}{d/2 - 1} & \text{для парних d} \\ 2 \binom{N - \lfloor d/2 \rfloor - 1}{\lfloor d/2 \rfloor} & \text{для не парних d} \end{cases}$$

Оцінка складності - $O(N^{\lfloor d/2 \rfloor})$.

Позначимо границю опуклої множини через CH(S).

Задача ОО1.(ОПУКЛА ОБОЛОНКА). В E^d задана множина S, яка містить N точок, необхідно побудувати їх опуклу оболонку (т.д. повний опис границі CH(S)) .

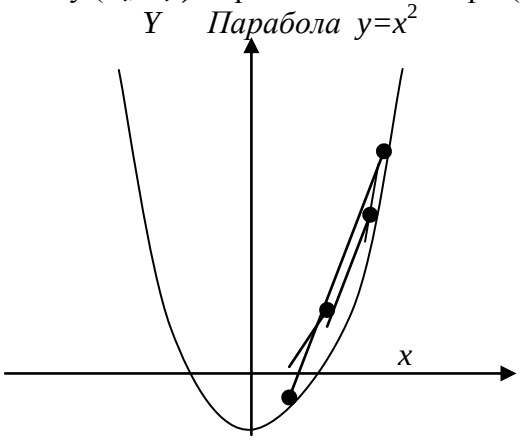
Задача ОО2.(КРАЙНІ ТОЧКИ). В E^d задана множина S, яка містить N точок. Необхідно вибрати ті з них, які є вершинами опуклої оболонки conv(S).

Задача ОО1 асимптотично є такою ж складною, як і задача ОО2, тобто КРАЙНІ ТОЧКИ $\propto_N$ ОПУКЛА ОБОЛОНКА.

Нижня оцінка задачі пошуку опуклої оболонки: для знаходження опуклої оболонки множини N точок у просторі $d \geq 2$ необхідно O(NlogN) операцій. Задачу сортування можна за N кроків звести до задачі пошуку опуклої оболонки.

Теорема.5.2. *Задача сортування зводиться за лінійний час до задачі побудови опуклої оболонки, і для знаходження впорядкованої опуклої оболонки N точок на площині потрібно часу* $\Omega(N \log N)$.

Доведення. Нехай задані N додатних дійсних чисел $x_1, x_2, \ldots, x_N$. Поставимо у відповідність числу x_i точку (x_i, x_i^2) і присвоїмо їй номер i (мал.3.1).



Мал.3.1. Ілюстрація доведення теореми 3.2.

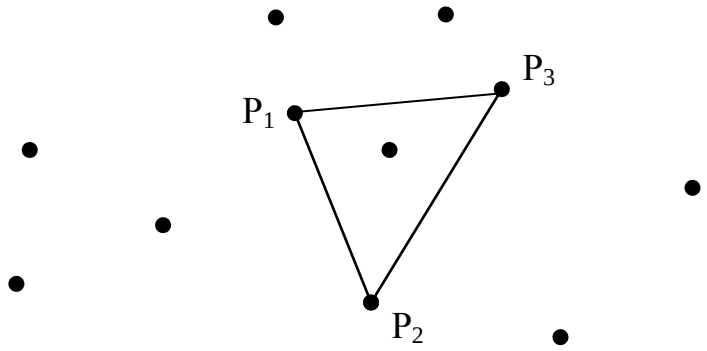
Усі ці точки лежать на параболі $y=x^2$. Опукла оболонка цієї множини точок, представлена у стандартному вигляді, буде складатись зі списку точок множини, упорядкованої по значенню абсциси. Один перегляд списку дозволяє прочитати в певному порядку значення x_i.

Опукла оболонка буде описана послідовністю точок (вершин), x-координати яких, будуть упорядковані, тому, маючи опуклу оболонку, можемо отримати впорядкований масив точок $x_1\ x_2\ \ldots x_n$. Нижня оцінка задачі сортування дорівнює нижній оцінці задачі пошуку опуклої оболонки і дорівнює O(N log N).

Означення 5.5. Точка p опуклої множини S називається *крайньою*, якщо не існує точок a, в $\in$ S, таких, що p $\in$ (a, в). Звідси випливає, що для, того щоб знайти опуклу оболонку, потрібно виконати такі два кроки:

1. Визначити крайні точки.
2. Упорядкувати точки так, щоб вони утворили опуклий многокутник.

Теорема 5.3. *Точка p не являється крайньою точкою, тоді і тільки тоді, коли p належить деякому трикутнику* $p_1p_2p_3$, *де* $p_1p_2p_3 \in S$ *та* $p_1 \neq p$, $p_2 \neq p$, $p_3 \neq p$.$\in O(N^3)$ трикутників, які визначаються N точками з множини S.

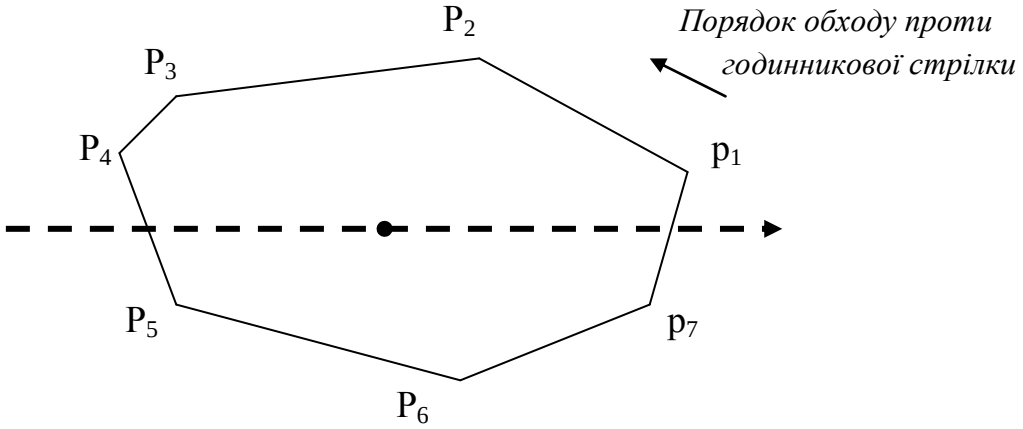


Мал.3.3. Точка *p* не є крайньою, так як вона знаходиться внутрі трикутника ($p_1p_2p_3$).

За час $O(N^3)$ можна визначити, чи являється точка крайньою. Побудова цієї процедури для всіх N точок множини S потребує часу $O(N^4)$.

Теорема.5.4. *Промінь, який виходить із внутрішньої точки опуклої обмеженої множини S, перетинає її границю в одній точці.*

Теорема.5.5. *Послідовні вершини опуклого многокутника впорядковані за полярним кутом відносно довільної внутрішньої точки.*



Мал.3.4.Вершини многокутника *P* упорядковані відносно точки q.

Якщо відомі крайні точки деякої множини, то опуклу оболонку *P* можна знайти, вибравши точку q , як внутрішню точку оболонки, і, упорядкувавши потім крайні точки у відповідності з полярним кутом відносно q.

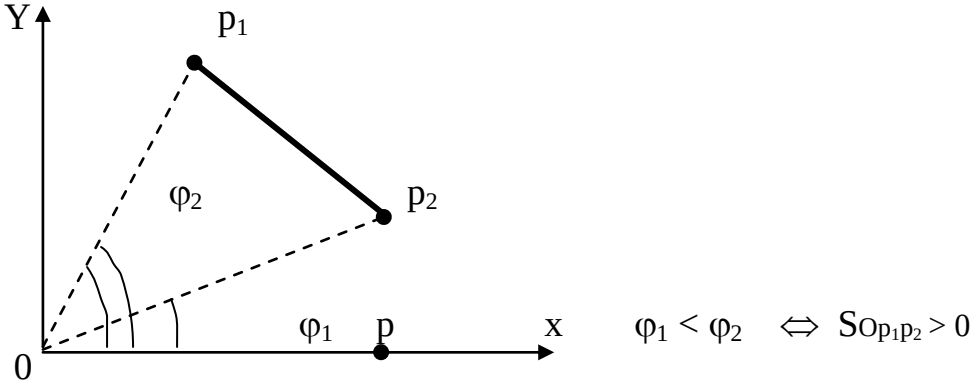
Способи знаходження точки q.

1)За точку q можна взяти центроїд множини крайніх точок p_1, p_2, ... , p_k: $p=(x_p, y_p)$, $x_p= (\Sigma x_i)/k$, $y_p= (\Sigma y_i)/k$

так, як відомо, що центроїд множини точок є внутрішньою точкою опуклої оболонки. Центроїд множини із N точок у k-мірному просторі може бути тривіально визначений за O(Nk) арифметичних операцій.

2)Грехем запропонував метод знаходження внутрішньої точки, згідно якого достатньо взяти центроїд довільних трьох не колінеарних точок, починаючи з двох довільних точок і, по черзі, досліджуючи N-2 точки, що лишились, шукаючи серед них одну, яка не лежить на прямій, що визначається першими двома точками. В гіршому випадку для цього треба час O(N).

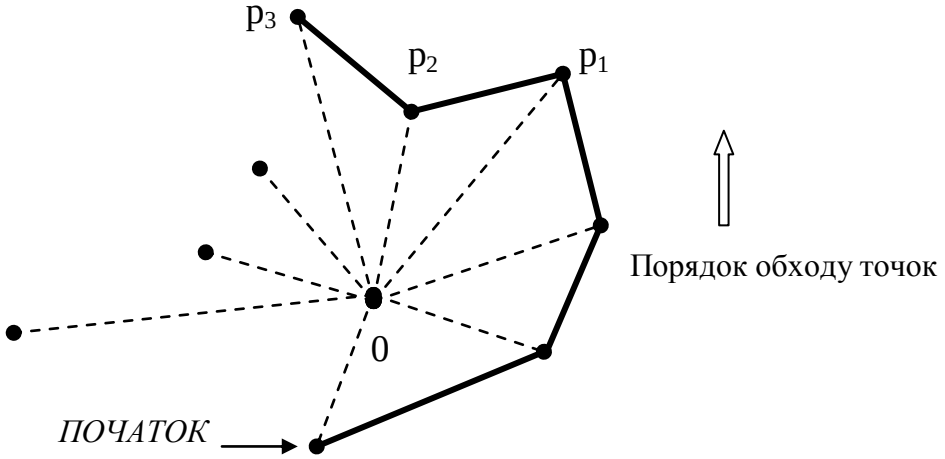
Знайшовши крайні точки множини S, можна за час O(N) знайти точку q , яка є внутрішньою точкою оболонки і відсортувати крайні точки відносно q як початку координат за полярним кутом. Це можна зробити , перейшовши до полярної системи координат за час O(N) , а потім відсортувавши їх за час O(NlogN). Нехай задані дві точки p_1 і p_2. p_2 утворює з віссю OX строго менший полярний кут, ніж p_1, тоді і лише тоді , коли трикутник (O, p_2, p_1) має строго додатну площу (мал. 3.5).



Мал.3.5. Порівняння полярних кутів через визначення орієнтованої площі $\Delta(0, p_2, p_1)$.

4.2. МЕТОД ГРЕХЕМА ПОБУДОВИ ОПУКЛОЇ ОБОЛОНКИ

Припустимо, що внутрішня точка уже знайдена, і являється початком координат. Упорядкуємо лексографічно N точок у відповідності із значеннями полярного кута й відстані від початку координат. При цьому не потрібно обчислювати дійсну відстань між двома точками, а лише порівняти дві величини. Порівняння відстаней необхідно виконувати лише у випадку, якщо дві точки мають один і той же полярний кут. Але тоді вони лежать на одній прямій, і порівняння буде тривіальним. Якщо точка не є вершиною опуклої оболонки, то вона є внутрішньою точкою для деякого трикутника (Opq), де *p* і *q*- послідовні вершини опуклої оболонки. **Суть алгоритму Грехема** полягає **в однократному перегляді упорядкованої послідовності точок, у процесі якого вилучаються внутрішні точки.** Точки, що лишилися являються упорядкованими вершинами оболонки, (мал.3.6).



Мал.3.6. Початок обходу точок у методі Грехема. Вершина p_2 вилучається, якщо кут $p_1p_2p_3$ виявляється вгнутим.

Перегляд починається з точки, яка позначена як ПОЧАТОК, за яку можна взяти саму праву з найменшою ординатою точку з даної множини, і яка напевне є вершиною опуклої оболонки. Трійки послідовних точок многократно перевіряються у порядку обходу проти часової стрілки з метою визначення, чи утворюють вони кут, більший або рівний π. Якщо внутрішній
$\angle p_1p_2p_3 \geq \pi$, $p_1p_2p_3$ утворюють "правий поворот" ($\Delta > 0$)
$\angle p_1p_2p_3 < \pi$, $p_1p_2p_3$ утворюють "лівий поворот" ($\Delta < 0$)

Δ визначається по формулі (3.4)

$$\Delta = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \qquad (3.4)$$

В залежності від результату перевірки кута, який утворюється трійкою вершин, можливі два варіанти продовження перегляду:

1. $p_1p_2p_3$ утворюють правий поворот. Вилучити вершину p_2 і перевірити трійку $p_0p_1p_3$.
2. $p_1p_2p_3$ утворюють лівий поворот. Продовжити перегляд, перейшовши до трійки $p_2p_3p_4$.

Перегляд завершається тоді, коли, обійшовши усі вершини, знову приходимо у вершину ПОЧАТОК.

Алгоритм обходу Грехема

Procedura ОБОЛОНКА-ГРЕХЕМА(S)

1. Знайти внутрішню точку q.
2. Використовуючи q як початок координат, упорядкувати точки множини S Лексографічно у відповідності з полярним кутом і відстанню від q. Організувати точки множини у вигляді кільцевого двічі зв’язаного списку з посиланням НАСТУП і ПОПЕР і вказівником ПОЧАТОК на першу вершину. Значення *true* логічної змінної *f* вказує на те, що вершина ПОЧАТОК виявилась досягнутою при прямому просуванні по оболонці, а не в результаті повернення.
3. (Обхід)

```
begin v:= ПОЧАТОК; w:= ПОПЕР[v]; f:=false;
   while (НАСТУП[v]≠ПОЧАТОК or f=false) do
      begin if НАСТУП[v]=w then f:=false;
                if (три точки v, НАСТУП[v]НАСТУП[НАСТУП[v]]
              утворюють лівий поворот) then v:=НАСТУП[v]
                else begin ВИЛУЧИТЬ НАСТУП[v];
                           v:=ПОПЕР[v]
                     end
       end
end
```

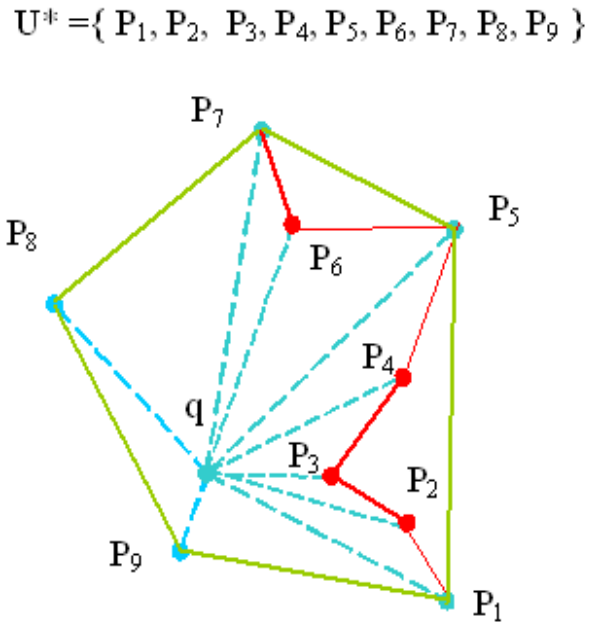
Після завершення алгоритму список містить упорядковані вершини оболонки.

Терема 5.6. *Опукла оболонка N точок на площині може бути знайдена за час O(NlogN) при пам'яті O(N) із використанням лише арифметичних операцій і порівнянь.*

Доведення. З попереднього обговорення АГ зрозуміло, що в ньому використовуються лише арифметичні операції й порівняння. Кроки 1 і 3 вимагають лінійного часу, тоді як крок 2, який є визначальним по часу роботи, виконується за час O(NlogN). Для представлення зв'язного списку точок достатньо O(N) пам'яті.

Приклад.

Трійка	кут	перехід	СН
$p_1p_2p_3$	$< \pi$	$\Rightarrow p_2p_3p_4$	p_1p_2
$p_2p_3p_4$	$> \pi$	$\Leftarrow p_1p_2p_4$	$-p_3$
$p_1p_2p_4$	$> \pi$	$\Leftarrow p_1p_4p_5$	$-p_2$
$p_1p_4p_5$	$> \pi$	$\Leftarrow p_1p_5p_6$	$-p_4$
$p_1p_5p_6$	$< \pi$	$\Rightarrow p_5p_6p_7$	p_5
$p_5p_6p_7$	$> \pi$	$\Leftarrow p_1p_5p_7$	$-p_6$
$p_1p_6p_7$	$< \pi$	$\Rightarrow p_6p_7p_8$	
$p_5p_7p_8$	$< \pi$	$\Rightarrow p_7p_8p_9$	p_7
$p_7p_8p_9$	$< \pi$	$\Rightarrow p_8p_9p_1$	p_8
$p_8p_9p_1$	$< \pi$	END	p_9



Мал.3.7

4.3. МЕТОД ДЖАРВІСА

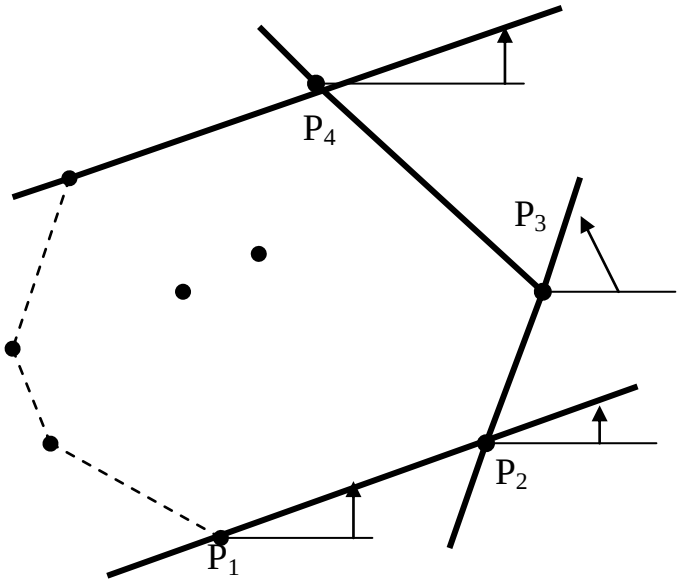
Враховуючи той факт, що многокутник можна задати як вершинами так і його ребрами, цікавим було б визначати ребра ОО, замість його вершин.

Теорема. 5.7. *Відрізок l, який визначається двома точками, є ребром ОО тоді і лише тоді , коли усі інші точки заданої множини лежать на l або по одну сторону від нього.*

Теорема 5.8. *Якщо знайдений відрізок* $l = pq$, $p, q \in S$, $pq \in CH(S) \Leftrightarrow$ *для будь-якого* $r \in S$; pq *- є ребром опуклої оболонки;* $S_{p,r,q} \leq 0$; $S_{p,r,q}$ *- орієнтована площа трикутника.*

Припустимо, що знайдена найменша в лексографічному порядку точка p_1 заданої множини точок. Ця точка за відомо є вершиною оболонки, тепер необхідно знайти наступну за нею вершину p_2 опуклої оболонки. Точка p_2 - це точка , яка має найменший додатній полярний кут відносно точки p_1 як

початку координат. Аналогічно наступна точка p_3 має найменший полярний кут відносно точки p_2 як початку координат, і кожна наступна точка опуклої оболонки може бути знайдена за лінійний час. **Алгоритм Джарвіса** обходить кругом опуклу оболонку, породжуючи в необхідному порядку послідовність крайніх точок, по одній точці на кожному кроці (мал.3.9). Таким чином будується частина ОО (ломана лінія) від найменшої у лексографічному порядку точки (p_1 на мал. 3.9) до найбільшої в лексографічному порядку точки (p_4 на тому ж малюнку 3.9). Побудова ОО завершується знаходженням іншої ламаної, яка виходить із найбільшої у лексографічному порядку точки в найменшу. Виходячи із симетричності цих двох етапів необхідно змінити на протилежні напрямки осей координат і мати справу з полярними кутами, найменшими відносно від'ємного напрямку осі *x*.



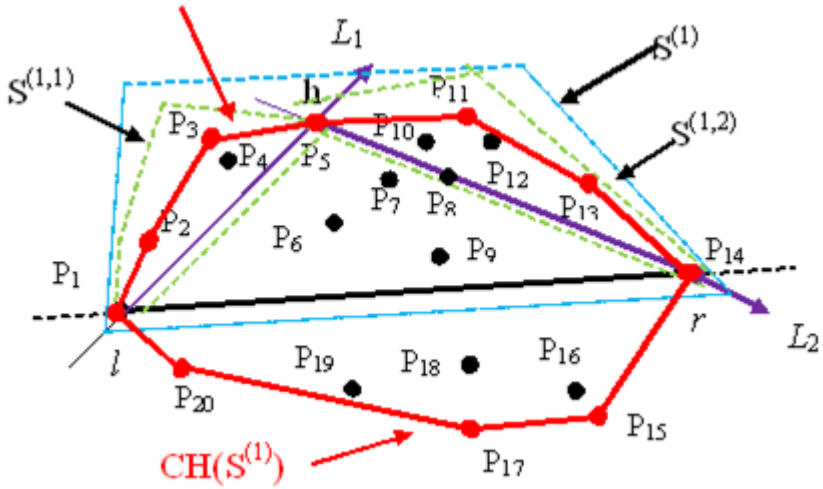
Мал.3.9. Побудова опуклої оболонки Методом Джарвіса.

Так як усі N точок множини можуть лежати на її ОО (бути її вершинами), АД витрачає на знаходження кожної точки оболонки лінійний час, то час виконання алгоритму в гіршому випадку рівний $O(N^2)$, що гірше ніж в АГ. Якщо в дійсності число вершин ОО рівне *h*, то час виконання алгоритму Джарвіса буде $O(hN)$, і дуже ефективний , коли апріорі відомо, що *h* мале. Наприклад, якщо оболонка є многокутником із довільним постійним числом сторін, то її можна знайти за лінійний, відносно числа точок, час.

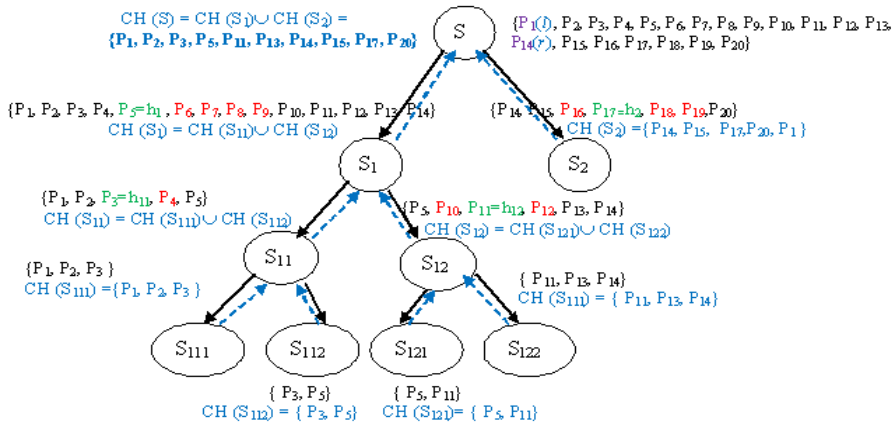
4.4.ШВИДКІ МЕТОДИ ПОБУДОВИ ОПУКЛОЇ ОБОЛОНКИ.

Із-за близькості з алгоритмом ШВИДКОСОРТ алгоритмів, які розглядатимуться, дамо їм назву - швидкі методи побудови опуклої оболонки, або ШВИДКОБОЛ. Відповідний ШВИДКОБОЛ-метод розбиває множину S із N точок на дві підмножини, кожна х яких буде містити одну із двох ламаних, з'єднання яких дає многокутник опуклої оболонки.

1. Початкове розбиття множини визначається прямою, яка проходить через дві точки l і r , які мають, відповідно, найменшу й найбільшу абсциси (мал. 3.10) . Позначимо через $S^{(1)}$ підмножину точок, які розташовані вище або на прямій, яка проходить через l і r , а через $S^{(2)}$ симетрично визначену підмножину точок, розташованих нижче або на тій же прямій.



Мал.3.10 а. Точки l , r і h визначають розбиття множини $S^{(1)}$.



Мал.3.10 б. Граф алгоритму.

2. Визначим точку h, для якої трикутник (hlr) має максимальну площу серед усіх трикутників $\{(plr)$: $p \in S^{(1)}\}$, а якщо таких точок більше ніж одна, то вибираємо ту з них, у якої кут $\angle$ (hlr) більший.

4. Потім будуються дві прямі: одна L_1, спрямована із l в h , інша L_2- із h в r.

Зазначимо, що точка *h* гарантовано належить опуклій оболонці. Для кожної точки множини $S^{(1)}$ визначається її положення відносно цих прямих. Ні жодна з точок не знаходиться одночасно ліворуч як від L_1, так і від L_2, окрім того, усі точки, розташовані праворуч від обох прямих, являються внутрішніми точками трикутника (*lrh*) і тому можуть бути вилучені з подальшої обробки. Точки, розташовані ліворуч від L_1 або на ній (і розташовані праворуч від L_2), утворюють множину $S^{(1,1)}$; аналогічно утворюється множина $S^{(1,2)}$. Утворені множини $S^{(1,1)}$ і $S^{(1,2)}$ передаються на наступний рівень рекурсивної обробки.

Виберемо відповідний набір початкових значень $\{l_0, r_0\}$ точок *l* і *r:* як *l* точку (x_0, y_0), яка має найменшу абсцису, а за r_0 візьмемо точку $(x_0, y_0-\varepsilon)$, де ε- довільне мале додатне число. Звідси випливає, що за початковою прямою , яка розбиває множину на частини, обирається вертикальна пряма, яка проходить через точку l_0. Після завершення алгоритму точка r_0 вилучається.

Припустимо, що множина S містить принаймні дві точки , а функція САМАДАЛЬНЯТОЧКА (S;l,r) обчислює точку $h \in S$ описаним вище способом; окрім того, ШВИДКОБОЛ повертає як результат упорядкований список точок, а " *" позначає операцію зчеплення.(конкатенацію) списків.

function ШВИДКОБОЛ(S;l,r)

1. **begin if** (*S={l,r}*) **then return***(l, r)* (* ОО складається з єдиного орієнтованого ребра *)
2. **else begin** *h:*= САМАДАЛЬНЯТОЧКА(S;l,r);
3. $S^{(1)}$:= точки множини S , розташовані ліворуч від [*l*,*h*] або на ній;
4. $S^{(2)}$:= точки множини S , розташовані ліворуч від [*h*,*r*] або на ній;

return ШВИДКОБОЛ($S^{(1)}$;l,r) *
(ШВИДКОБОЛ($S^{(2)}$;h,r)-h)

end

end

Таким чином, якщо уже маємо функцію ШВИДКОБОЛ, то поставлену задачу можна розв'язати за допомогою такої простої програми:

begin $l_0=(x_0, y_0)$:= точка множини S із найменшою абсцисою;
$r_0=(x_0, y_0 - \varepsilon)$;
ШВИДКОБОЛ(S; l_0, r_0);
вилучити точку r_0 (* це еквівалентно тому, що покласти ε = 0*)
end

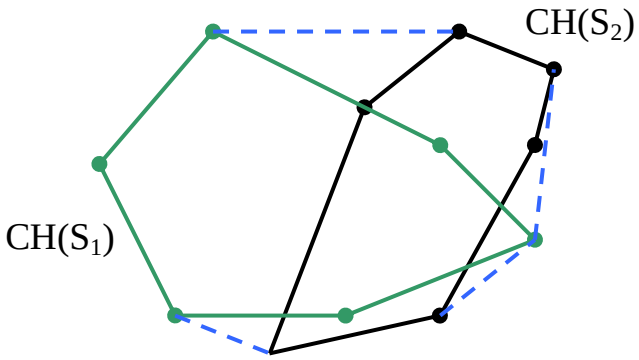
Потім слідує рекурсивне звернення до функції для обробки $S^{(1)}$ і $S^{(2)}$. Якщо потужність кожної з цих множин не перевищує потужності множини S, помноженої на деяку константу меншу 1, і ця умова має місце на кожному рівні рекурсії, то час виконання алгоритму рівний O(NlogN). Однак, в гіршому випадку ШВИДКОБОЛ, не дивлячись на його простоту, має той же недолік, що і у ШВИДКОСОРТ, який дає час виконання $O(N^2)$.

4.5. АЛГОРИТМ ТИПУ ’’РОЗПОДІЛЯЙ ТА ВОЛОДАРЮЙ’’

Припустимо, при розв’язку задачі побудови опуклої оболонки, початкова множина точок була розбита на дві частини S_1 і S_2 , кожна з яких містить половину точок початкової множини. Якщо тепер рекурсивно знайдено окремо $CH(S_1)$ і $CH(S_2)$, то які додаткові витрати необхідні для побудови $CH(S_1 \cup S_2)$, т.т. ОО початкової множини? Для відповіді на це питання можна скористатись таким співвідношенням:

$$CH(S_1 \cup S_2)= CH(CH(S_1) \cup CH(S_2)) \qquad (3.7)$$

Головна думка полягає в тому, що $CH(S_1)$ і $CH(S_2)$- це не просто неупорядковані множини точок, а опуклі многокутники (мал.3.11).



Мал.3.11. Побудова опуклої оболонки методом "розподіляй та володарюй".

Задача ОО4. (ОПУКЛА ОБОЛОНКА ОБ’ЄДНАННЯ ОПУКЛИХ МНОГОКУТНИКІВ).
Задано два опуклих многокутники P_1 і P_2. Знайти опуклу оболонку їх, об’єднання.

procedura СЛИБОЛ(S)
1. $|S| \le k_0$ (k_0- деяке невелике ціле число), то побудувати опуклу оболонку одним із прямих методів і зупинитись; інакше перейти до кроку 2.
2. Розбити початкову множину S довільним чином на дві приблизно рівні підмножини S_1 і S_2.
3. Рекурсивно знайти опуклі оболонки S_1 і S_2.
4. Злити (об’єднати) дві отримані оболонки разом, утворюючи CH(S).

Позначимо символом U(N)час, необхідний для знаходження ОО об'єднання опуклих многокутників, кожен із яких має N/2 вершин. Якщо T(N)- час, необхідний для знаходження ОО множини з N точок, то, використовуючи співвідношення (3.7), отримаємо:

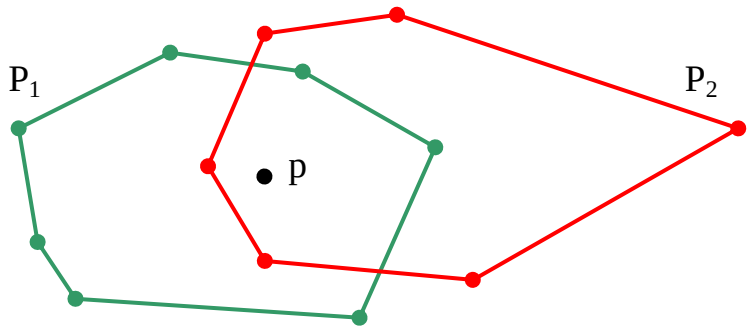
$$T(N) \le 2T(N/2) + U(N). \qquad (3.8)$$

Алгоритм "злиття" запропонований М. Шеймосом .

proceduraОБОЛОНКА-ОБ'ЄДНАННЯ-ОПУКЛИХ-МНОГОКУТНИКІВ(P_1, P_2).

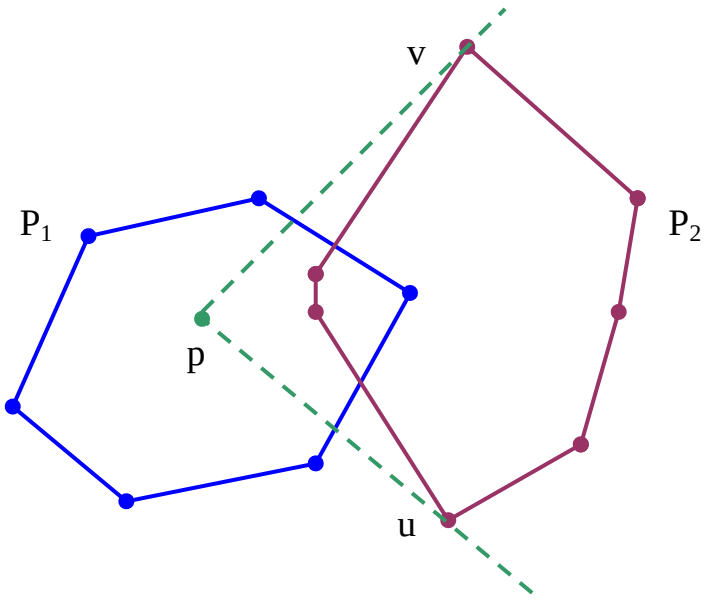
1. Знайти деяку внутрішню точку p многокутника P_1. (Наприклад, центроїд трьох довільних вершин P_1. Така точка p буде внутрішньою точкою $CH(P_1 \cup P_2)$).
2. Визначити, чи є p внутрішньою точкою P_2. Це може бути зроблено за час O(N). Якщо p не є внутрішньою точкою P_2, перейти до кроку 4.
3. p є внутрішньою точкою P_2 (мал.1.12). По теоремі 3.6 вершини як P_1, так і P_2 виявляються упорядкованими у відповідності із значенням полярного кута відносно точки p. За час O(N) можна отримати упорядкований список вершин як P_1, так і P_2 шляхом злиття списків вершин цих многокутників.

   Перейти до кроку 4.



Мал.3.12. Точка p знаходиться внутрі многокутника P_2.

4. p не є внутрішньою точкою P_2 (мал.3.13). Якщо дивитись із точки p, то многокутник P_2 лежить у клині з кутом розвороту меншим або рівним π. Цей клин визначається двома вершинами u і v многокутника P_2, які можуть бути знайдені за лінійний час за один обхід вершин многокутника P_2. Ці вершини розбивають границю P_2 на два ланцюги вершин, які є монотонними відносно зміни полярного кута з початком у p. Один із цих двох ланцюгів, яки є опуклим по напрямку до точки p, може бути відразу вилученим, так як його вершини будуть внутрішніми точками $CH(S_1 \cup S_2)$. Інший ланцюг P_2 і границя P_1 являють собою два упорядкованих списки, які вміщують у сумі не більше N вершин. За час O(N) їх можна злити в один список вершин $P_1 \cup P_2$, упорядкованих по куту відносно точки p.
5. Тепер до отриманого списку можна застосувати метод обходу Грехема, який вимагає лише лінійний час. Тепер одержана опукла оболонка $P_1 \cup P_2$.



Мал 3.13. Точка *p* знаходиться зовні многокутника P_2.

Якщо многокутник P_1 має m вершин, а P_2 має n вершин , то час виконання цього алгоритму рівний O(m+n), що є оптимально. Так що тепер відомо, що U(N)=O(N), і розв'язок рекурентного співвідношення (3.8) у цьому випадку є T(N) = O(Nlog N). Таким чином має місце теорема:

Теорема 5.9. *Опукла оболонка об'єднання двох опуклих многокутників може бути знайдена за час, пропорційний сумарному числу їх вершин.*

Означення 5.6. *Опорною прямою* до опуклого многокутника P називається пряма l, яка проходить через деяку вершину многокутника P таким чином, що внутрішня частина P повністю знаходиться по одну сторону від l (у деякому розумінні поняття опорної прямої подібне поняттю дотичної).

Очевидно, що два опуклих многокутники P_1 і P_2 з n і m вершинами відповідно, таких, що один не лежить у нутрі іншого, мають загальні опорні прямі (принаймні не більше 2min(n,m)). Якщо уже побудована ОО об'єднання P_1 і P_2 , то опорні прямі обчислюються в результаті перегляду списку вершин CH($P_1 \cup P_2$). Кожна пара послідовних вершин CH($P_1 \cup P_2$), одна з яких належить P_1 , а інша P_2 , визначає опорну пряму.

4.6.ДИНАМІЧНІ АЛГОРИТМИ ПОБУДОВИ ОПУКЛОЇ ОБОЛОНКИ.

Означення 5.7. Алгоритм, який обробляє дані в міру надходження, наз. *відкритим* , а алгоритм, який обробляє усю сукупність у цілому, називається *закритим*.

Означення 5.8. Будемо називати часовий інтервал між уводом двох останніх елементів даних *затримкою надходження даних.*

Означення 5.9. Вважатимемо, що надходження даних відбувається (розподілено) *рівномірно по часу*. В цьому випадку корекція повинна виконуватись за час, який не перевищує постійну затримку надходження даних. Алгоритми, що працюють у такому режимі, і називаються відповідно *алгоритмами реального часу*.

Задача ВО6 (ВІДКРИТИЙ АЛГОРИТМ ДЛЯ ОПУКЛОЇ ОБОЛОНКИ)

На площині задана послідовність із N точок $p_1,\ldots,p_N$. Необхідно знайти їх опуклу оболонку, організувавши обробку таким чином, щоб після обробки точки p_i мали CH($\{p_1, \ldots,p_i\}$).

Задача ВО6 (ОПУКЛА ОБОЛОНКА В РЕАЛЬНОМУ ЧАСІ)

На площині задана послідовність із N точок $p_1,\ldots,p_N$. Необхідно знайти їх опуклу оболонку при умові, що час затримки надходження точок постійний.

Якщо не брати до уваги час обробки, можна розробити відкритий алгоритм побудови ОО, застосувавши підхід оснований на алгоритмі Грехема, який складається з двох кроків (сортування й обходу), а саме:

1. Вводити точки до тих пір, доки не будуть знайдені трьох не колінеарні точки. Їх центроїд буде внутрішня точка кінцевої ОО і, таким чином, підходить для початку координат, відносно якого визначаються полярні кути точок при сортуванні в алгоритмі ОБОЛОНКА ГРЕХЕМА.
2. Підтримувати зв'язаний список упорядкованих крайніх точок. При надходженні точки p_i тимчасово вставити її у цей список у відповідності з її полярним кутом, витративши часу на це O(i).
3. Виконати перегляд зв'язаного списку крайніх точок методом Грехема. Так як перегляд методом Грехема лінійний, то необхідно лише O(i) часу. Можливі три варіанти завершення цього кроку:

а) точка p_i є крайньою обробленої на поточний момент множини, і її включення в число крайніх точок викликає вилучення декількох інших точок;

б) точка p_i є крайньою, але ніякі інші точки не вилучаються;

г) точка p_i є внутрішньою для опуклої оболонки , яка створюється, і тому вона

вилучається.

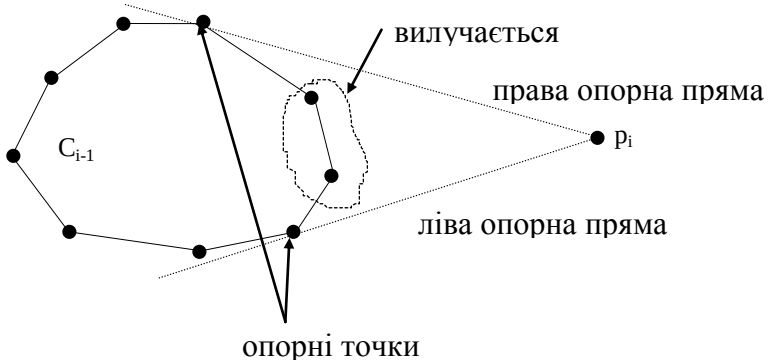
Загальний час виконання цього алгоритму в гіршому випадку рівний $O(N^2)$, що має місце, якщо кожна точка являється крайньою.

Теорема 5.10. *Будь-який відкритий алгоритм побудови опуклої оболонки в гіршому випадку витрачає на обробку між надходженнями послідовних елементів даних час* $\Omega(\log N)$.

Алгоритм Преппарата.

Якщо точки обробляються в порядку p_1, p_2,.... і p_i поточна точка, то позначимо через C_{i-1} опуклу оболонку множини { p_1, p_2, ..., p_{i-1}}. Необхідно побудувати з точки p_i опорні прямі до C_{i-1}, якщо вони існують. В першому випадку повинна вилучитись відповідна низка вершин, яка міститься між двома опорними точками, а замість них уставляється точка p_i ,мал..3.14. Для

зручності опорні прямі будемо називати *лівою* або *правою,* у відповідності з тим, по яку сторону вона знаходиться, якщо дивитись з точки p_i на C_{i-1} .



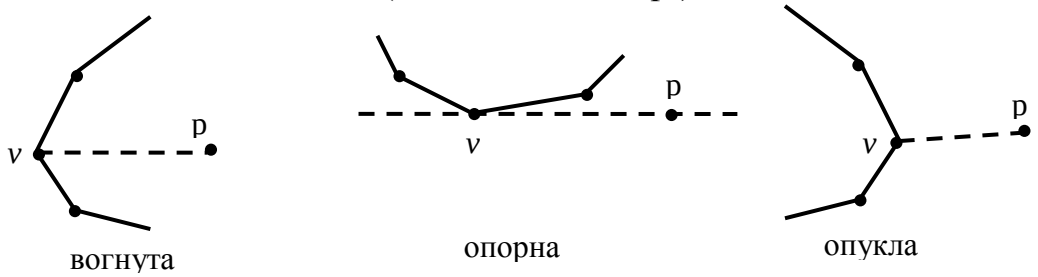
Мал..3.14. Опорні прямі із точки p_i до опуклого многокутника C_{i-1}.

Класифікація вершини по відношенню до відрізка[p,v].

Вершина v називається *ввігнутою*, якщо відрізок [p,v] перетинає внутрішню частину многокутника *C*. Інакше, якщо дві суміжні з v вершини лежать по одну сторону від прямої, яка проходить через точки p і v, вершина v називається *опорною*. У випадку, що лишився v наз. *опуклою* вершиною.

Якщо v опорна вершина, то на цьому розв'язання задачі завершується. Припустимо. що шукається ліва опорна пряма . Якщо точка v не є опорною, то необхідно рухатись по вершинам многокутника *C* проти або за часовою стрілкою в залежності від того, чи є v ввігнутою або опуклою вершиною (мал.3.15). Таким способом можна визначити дві опорні точки (якщо вони існують). Якщо це зроблено, необхідно мати можливість вилучити із циклічної послідовності вершин многокутника *C* низку вершин (можливо, порожню) і вставити в утворений розрив точку *p. виконуються* такі операції:

I. ПОШУК в упорядкованій послідовності елементів (кільцевого списку вершин оболонки) для визначення опорних прямих із точки p_i;
II. РОЗЧЕПЛЕННЯ послідовності на дві послідовності й ЗЧЕПЛЕННЯ двох послідовностей;
III. ВСТАВКА одного елемента (поточної точки p_i).

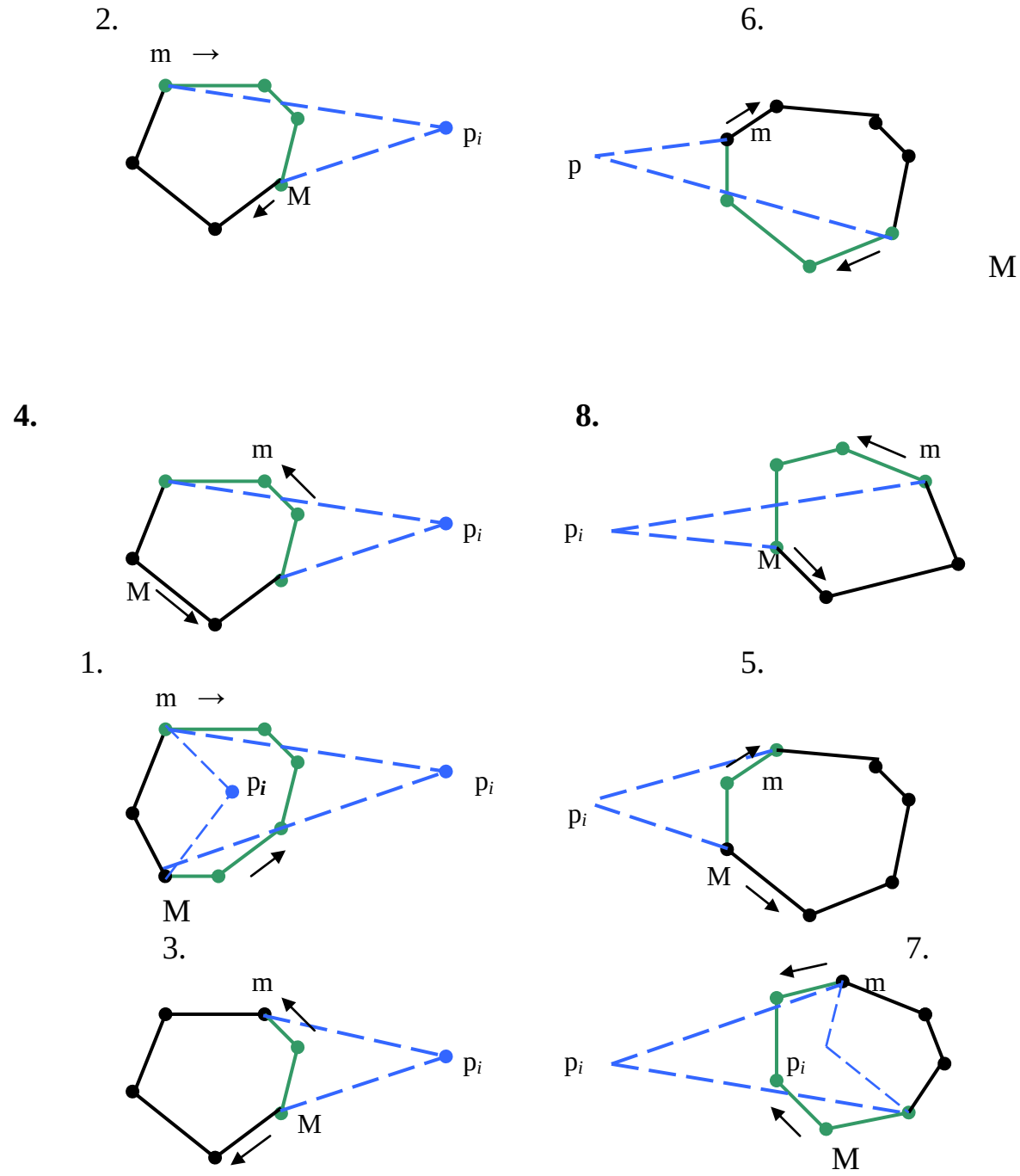


Мал..3.15. Класифікація вершин v многокутника *C* відносно відрізка [p,v].

Структура даних, в повній мірі задовольняюча перерахованим вимогам, називається *зчепленою чергою*. Вона реалізується з допомогою збалансованого по висоті дерева пошуку, і при цьому кожна з указаних вище операцій виконується за час O(log*i*) у гіршому випадку, де *i* – число вузлів дерева. Природно, кільцева послідовність вершин зображається в цій

деревовидній структурі даних ланцюгом, і при цьому перший і останній елементи вважаються суміжними. В структурі *T* виділяються дві вершини : *m*- самий лівий член ланцюгу і *M*- кореневий член ланцюгу. Крім того, використовується кут $\angle(mp_iM)$, який позначимо α. Цей кут називається *опуклим*, якщо він менший або дорівнює π, і *ввігнутим* у противному.

В залежності від класифікації вершин *m* і *M* (увігнута, опорна, опукла) і кута α можливі всього вісімнадцять випадків. Однак всі ці випадки можна звести до восьми(які покривають усі можливості), зведених у таблиці 1 і показаним на малюнку 3.16.



Мал..3.16. Вісім можливих випадків в залежності від класифікації вершин m, M і кута α.

Таблиця 1.

	α	m	M
1	Опуклий	Увігнута	Увігнута
2	Опуклий	Увігнута	Неувігнута
3	Опуклий	Неувігнута	Опукла
4	Опуклий	Неувігнута	Неопукла
5	Увігнутий	Опукла	Опукла
6	Увігнутий	Опукла	Не опукла
7	Увігнутий	Неопукла	Увігнута
8	Увігнутий	Неопукла	Не увігнута

Наступна процедура здійснює пошук вершини *l:*

function ЛІВИЙ_ПОШУК(Т)
Input:дерево Т, яке описує послідовність вершин
Output: вершина *l*

1. **begin** с:=КОРІНЬ(Т);
2. **if** (*pc*-опорна пряма) **then** *l:=c*
3. **else begin if** (с- опукла вершина) **then**
   Т:=ЛДЕРЕВО(с) **else** Т:=ПДЕРЕВО(с)
4. *l*:=ЛІВИЙ_ПОШУК(Т)
   end;
5. **return** *l*

end

Враховуючи, що дерево Т збалансоване і містить не більше $I<N$ вершин, а на обробку в кожному вузлі потрібно обмежений час, отримуємо для вершини пошуку оцінку O(log*i*).

В залежності від того, передує вершина *l* вершині *r* у дереві Т або ні, треба виконати дещо різні дії (мал.3.17). В першому випадку необхідно двічі виконати операцію розчеплення й один раз зчеплення; у другому випадку тільки двічі виконується операція розчеплення.

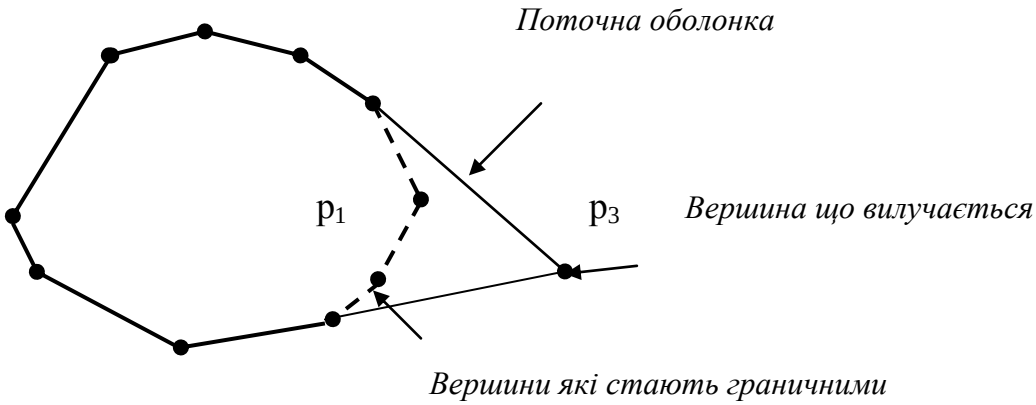
Як уже згадувалось раніше, обидві операції РОЗЧЕПИТИ й ЗЧЕПИТИ виконуються за час O(log*i*). В результаті, враховуючи, що корекція опуклої оболонки може бути виконана за час O(log*i*), маємо теорему.

Теорема 5.11.*Опукла оболонка множини із N точок на площині може бути знайдена за допомогою відкритого алгоритму за час θ(NlogN) із часом корекції θ(logN), т.т. може бути побудована в реальному часі.*

4.7.УЗАГАЛЬНЕННЯ:ПІДТРИМКА ДИНАМІЧНОЇ ОПУКЛОЇ ОБОЛОНКИ.

Наведемо формальну постановку цієї задачі (мал..3.18):

Задача ОО7.(ПІДТРИМКА БОЛОНКИ). Задана порожня множина S і послідовність із N точок (p_1, p_2, ..., p_N), кожна з яких або додається до множини S, або вилучається з нього. Необхідно підтримувати опуклу оболонку множини S.



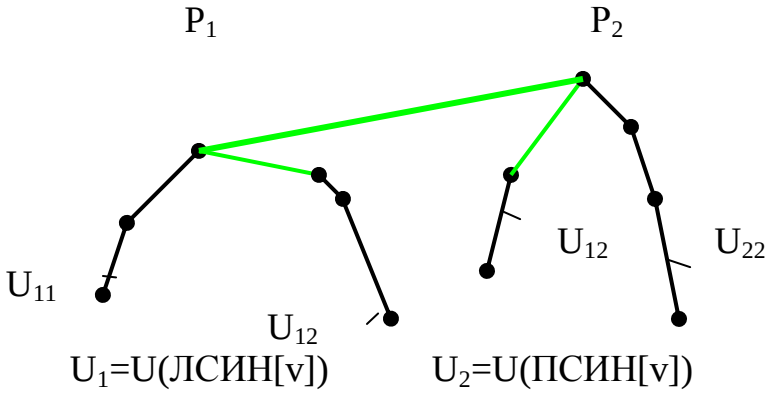
Мал..3.18. При вилучені точки p_3 точки p_1 і p_2 стають граничними точками опуклої оболонки.

Використовується той факт, що границя опуклої оболонки являється об'єднанням двох (опуклих) монотонних ламаних ліній (ланцюгів), які обмежують оболонку зверху і знизу і у відповідності з цим називаються *B-оболонкою* й *H-оболонкою* множини точок. Роглянемо побудову B-оболонки.

Упорядковуємо S по *x* – координаті. Нехай v- вузол дерева *T*. Позначимо через ЛСИН[v] і ПСИН[v] його лівого й правого нащадків, відповідно. Побудуємо B-оболонку точок, які зберігаються в листках піддерева з коренем у вузлі v. Позначимо через U(v)- B-оболонку множини точок , які зберігаються в листках пріддерева з коренем v. Виходячи із принципу індукції, припустимо, що уже існують U(ЛСИН[v] і ПСИН[v]). Далі (мал..3.19) визначаються дві опорні точки p_1 і p_2 єдиного спільного відрізку для двох оболонок. Тут використовується функція З'ЄДНАТИ(U_1, U_2), яка дозволяє знайти опорний відрізок для двох B-оболонок U_1, U_2. Функція З'ЄДНАТИ дозволяє ефективно розчіпляти U_1, які складають упорядковану пару (U_{11}, U_{12}), і аналогічно U_2- на пару ланцюгів (U_{21}, U_{22}). При цьому виконується така умова: опорна точка $p_1 \in U_1$ входить у склад U_{11}, а точка $p_2 \in U_2$ входить склад U_{22} (тобто в обох випадках опорна точка належить "зовнішньому" підланцюгу). На цьому етапі, зчепивши U_{11} і U_{22}, одержуємо шукану B-оболонку $U_1 \cup U_2$. Природно, щоб кожен вузол v дерева *T* указував на зчеплену чергу, яка представляє ту частину U(v), яка не належить U(БАТЬКО[v]).

Обернена операція: маючи U(v), одержати U(ЛСИН[v]) і U(ПСИН[v]). Знаючи ребро [p_1, p_2], яке з'єднує опорні точки (одне ціле число J[v], яке вказує положення точки p_1 у ланцюзі вершин U(v)), можна розчепити U(v) на ланцюги U_{11} і U_{22}, які у свою чергу можуть бути зчеплені з ланцюгами, які зберігаються в ЛСИН[v] і ПСИН[v], відповідно. Структура даних *T* доповнюється такими атрибутами, які зв'язуються з кожним вузлом v дерева:

1) вказівник на зчеплену чергу Q[v], яка містить частину U(v), яка не входить у U(БАТЬКО[v]) (якщо v є коренем, то Q[v]=U(v);
2) цілим числом J[v], яке вказує положення (індекс) лівої опорної точки в U(v).



Мал.3.19. Для побудови В-оболонки об'єднання оболонок U_1 і U_2 будується спільний опорний відрізок (міст) $[p_1,p_2]$.

Функція З"ЄДНАТИ(U_1, U_2):

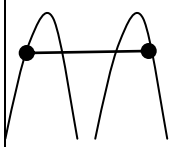
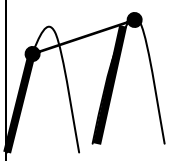
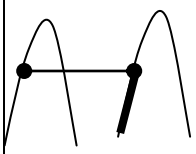
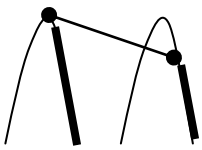
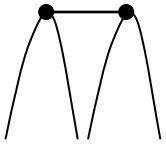
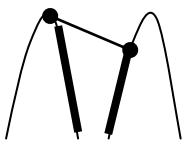
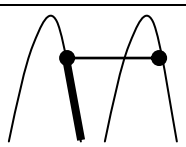
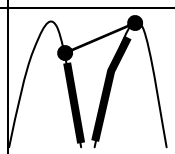
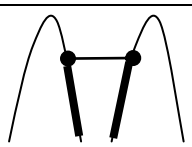
1. Знайти p_1 і p_2 .
2. Розчепити U_1 на U_{11} та U_{12}.
3. Розчепити U_2 на U_{21} та U_{22}.
4. Зчепити U_{11} з U_{22} $\Leftrightarrow$ CH($S_1 \cup S_2$): U_{11}*U_{22}

Структура даних використовує пам'ять об'ємом O(N).

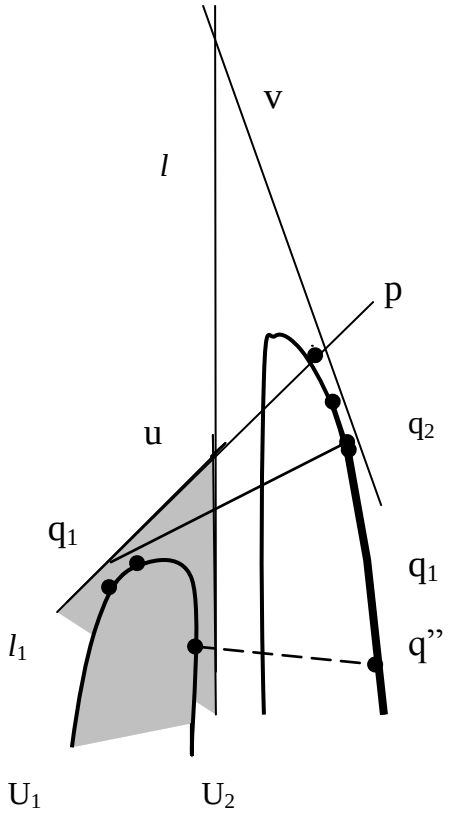
Лема 3.1. *З'єднання двох розділених опуклих ланцюгів, які містять у сумі N точок, може бути виконане за O(logN) точок.*

Доведення. Нехай дано дві В-оболонки U_1 і U_2 і дві вершини $q_1 \in U_1$ і $q_2 \in U_2$. Кожна із цих двох вершин може бути класифікована відносно відрізка $[q_1,q_2]$ як опукла, опорна або ввігнута . В залежності від класифікації можливі 9 випадків, схематично показаних на малюнку 3.20(а). Усі випадки зрозумілі, за винятком випадку (q_1,q_2) = (увігнута, ввігнута), який більш детально представлений на малюнку 3.20 (б).

(а)

$q_2 \backslash q_1$	увігнута	опорна	опукла
ввігнута			
опорна			
опукла			

б)



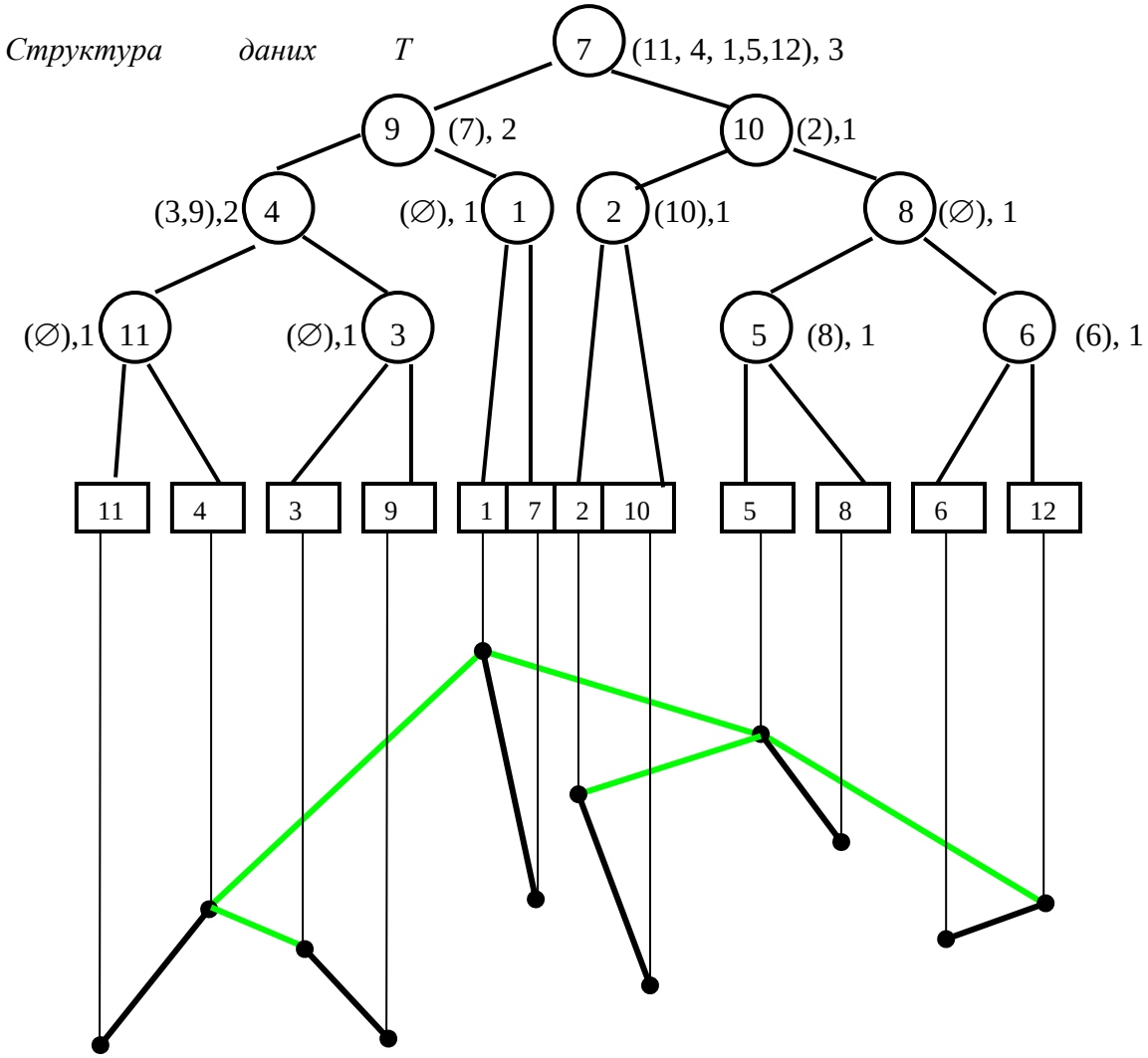
Мал..3.21 (а) Усі можливі випадки, які виникають при виборі деякої вершини в U_1 і деякої вершини в U_2; (б) ілюстрація випадку (ввігнута, ввігнута).

Нехай пряма l_1 проходить через q_1 і її правого сусіда на U_2. Позначимо через p точку перетину прямих l_1 і l_2. U_1 і U_2 роздільні вертикаллю l.
1)Припустимо, що точка p- праворуч прямої l. p_1 може належати лише заштрихованій області і $u(y)<v(y); \Rightarrow$ довільна вершина $q'' \in U_{2прав.}$, відносно q_2, є ввігнутою відносно відрізка [q',q"], де q'- довільна вершина U_1 і $U_{2прав.}$, відносно q_2 не розглядається. Що стосується ланцюга ліворуч від вершини q_1, то для нього не можна це стверджувати.2)Якщо точка перетину p знаходиться ліворуч від прямої l, то ланцюг ліворуч від вершини q_1 можна не розглядати.

	$T(U_1)$	$T(U_2)$
0	$v(q_1)$	$v(q_2)$
1	$v(q_1)$ (або ПС[$v(q_1)$])	ЛС[$v(q_2)$] (або $v(q_2)$)
2	ПС[$v(q_1)$]	ПС [$v(q_2)$]
3	$v(q_1)$	ПС [$v(q_2)$]
4	ЛС [$v(q_1)$]	ЛС [$v(q_2)$]
5	Результат	Результат
6	ЛС [$v(q_1)$]	ПС [$v(q_2)$]
7	ЛС [$v(q_1)$]	$v(q_2)$
8	ЛС [$v(q_1)$]	ПС [$v(q_2)$]
9	ЛС [$v(q_1)$]	ПС [$v(q_2)$]

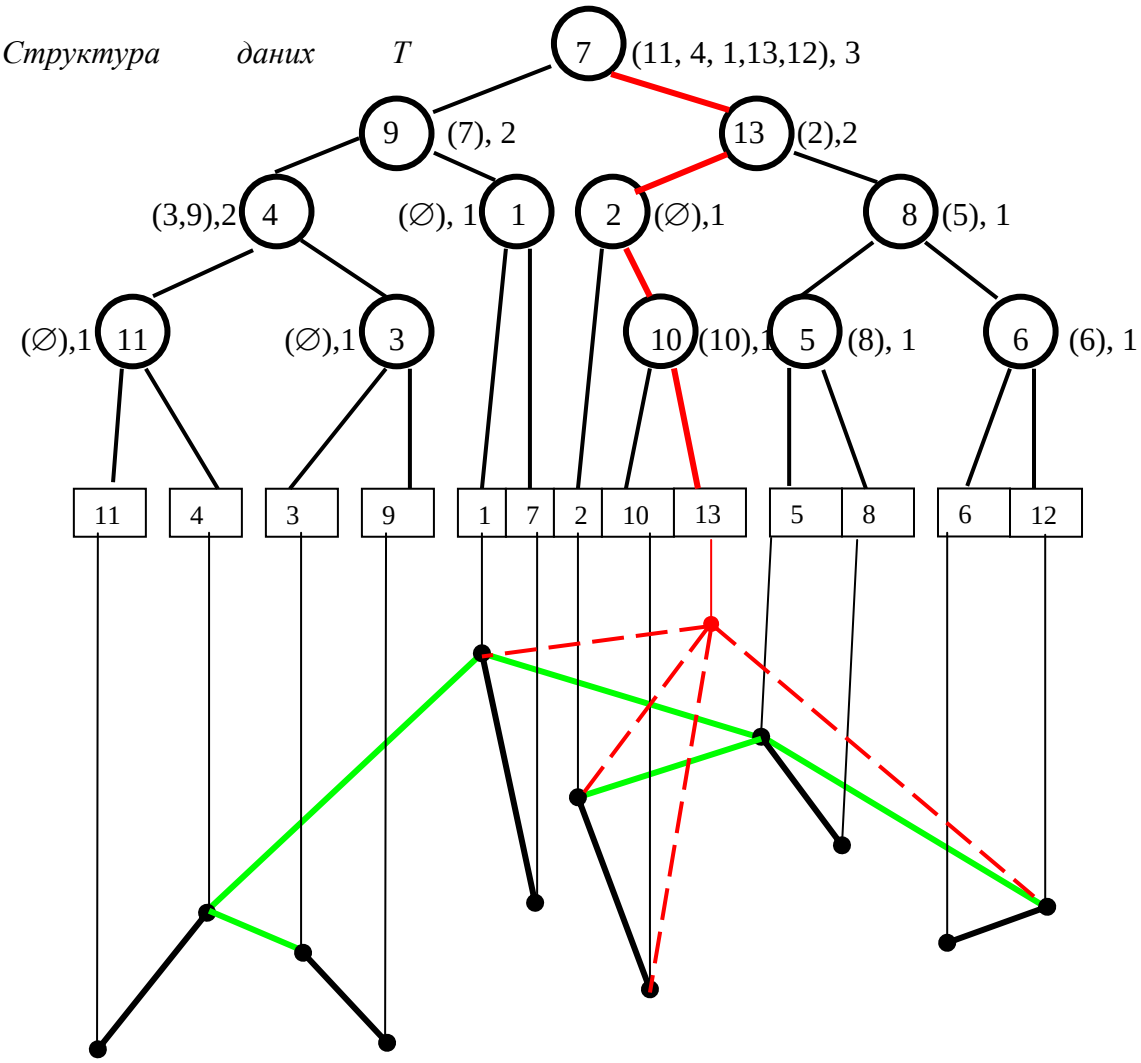
Якщо цей процес починається з кореневих вершин обох дерев, які представляють U_1 і U_2 відповідно, то зі збалансованості цих дерев час виконання функції З'ЄДНАТИ(U_1, U_2) складає O(logN), де N – це, зазвичай, сумарне число вершин у двох оболонках.

Маючи функцію З'ЄДНАТИ, можна аналізувати процес динамічної підтримки опуклої оболонки на площині, мал. 3.22. Номери точок відповідають порядку, в якому вони додавались до множини. В структурі даних *T* кожний лист відповідає точці, кожний не листовий вузол відповідає мосту (тобто опорному відрізку, який з'єднує дві оболонки). Для кожного такого вузла вказана пара Q[v],J[v]. Структура даних описує деяке *дерево оболонки*.



Мал..3.22. Множина точок на площині і відповідна їй верхня опукла оболонка та структура даних *T*.

Розглянемо вставку нової точки p. Будемо вважати, що точка p_{13} додається до множини точок, представленої на малюнку 3.22, як це показано на малюнку 3.23.



Мал..3.23. Вставка точки p_{13}. При вставці зачеплені вузли: {(11,4,1,13,12), 3}, {(2),2}, {(∅),1}, {(5), 1}, {(10),1}, {13}

Більш формально, спуск по дереву з метою вставки точки p здійснюється шляхом виклику рекурсивної процедури СПУСК(v,p):

```
   procedura СПУСК(v,p)

begin if (v≠лист) then
        begin (QL,QR):= РОЗЧЕПИТИ(U[v], J[v])
              U[ЛСИН[v]]:= ЗЧЕПИТИ(QL,Q[ЛСИН[v]]);
              U[ПСИН[v]]:= ЗЧЕПИТИ(Q[ПСИН[v]],QR);
              if (x[p] ≤ x[v]) then v:= ЛСИН[v] else v:= ПСИН[v];
              СПУСК(v,p)
         end
end.
```

Підйом по дереву здійснюється за допомогою наступної процедури:

```
   procedura ПІДЙОМ(v)

begin if (v ≠ корінь) then
        begin (Q1, Q2, Q3, Q4, J):= З'ЄДНАТИ(U[v], U[БРАТ[v]]);
              Q[ЛСИН[БАТЬКО[v]]]:= Q2;
              Q[ПСИН[БАТЬКО[v]]]:= Q3;
              U[БАТЬКО[v]]:= ЗЧЕПИТИ(Q1,Q4);
               J[БАТЬКО[v]]:= J;
               ПІДЙОМ(БАТЬКО[v])
         end
    else  Q[v]:= U[v]
end.
```

Так як k ≤ N, то час обробки одного вузла у процедурі СПУСК рівний O(logN). А так як глибина дерева *T* також рівна O(logN), то час виконання процедури СПУСК дорівнює $O(\log^2 N)$ у гіршому випадку. Процедура ПІДЙОМ, дає час O(logN).

Теорема 5.12. *Динамічна підтримка B-оболонки і H-оболонки із N точок на площині може бути виконана з часовими витратами на операції ставки і вилучення, рівними $O(\log^2 N)$ у гіршому випадку.*