

# Алгоритми та складність

ІІ семестр

Лекція 5

# Піраміди злиття (mergeable heaps)

Піраміди, що підтримують операцію злиття.

За замовчуванням вважаємо їх неспадаючими (вузол з мінімальним ключем у вершині).

- $\text{MAKE\_HEAP}()$ : створення нової порожньої піраміди.
- $\text{INSERT}(H, x)$ : вставка готового вузла  $x$  в піраміду  $H$ .
- $\text{MINIMUM}(H)$ : повертає вказівник на вузол піраміди  $H$  з найменшим ключем.
- $\text{EXTRACT\_MIN}(H)$ : видаляє вузол піраміди  $H$  з найменшим ключем і повертає вказівник на нього.
- $\text{UNION}(H_1, H_2)$ : повертає нову піраміду – результат злиття  $H_1, H_2$  (вони не зберігаються).
- $\text{DECREASE\_KEY}(H, x, k)$ : присвоєння вузлу  $x$  піраміди  $H$  значення ключа  $k$ , що є меншим за поточне.
- $\text{DELETE}(H, x)$ : видалення вузла  $x$  з піраміди  $H$ .

# Піраміди злиття (mergeable heaps)

Час виконання операцій при різних реалізаціях пірамід злиття

Процедура	Бинарная пирамида (наихудший случай)	Биномиальная пирамида (наихудший случай)	Пирамида Фibonacci (амортизированное время)
MAKE_HEAP	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
INSERT	$\Theta(\lg n)$	$O(\lg n)$	$\Theta(1)$
MINIMUM	$\Theta(1)$	$O(\lg n)$	$\Theta(1)$
EXTRACT_MIN	$\Theta(\lg n)$	$\Theta(\lg n)$	$O(\lg n)$
UNION	$\Theta(n)$	$\Omega(\lg n)$	$\Theta(1)$
DECREASE_KEY	$\Theta(\lg n)$	$\Theta(\lg n)$	$\Theta(1)$
DELETE	$\Theta(\lg n)$	$\Theta(\lg n)$	$O(\lg n)$

- При необхідності злиття бінарна піраміда виявляється не найкращим варіантом.
- Жодна з реалізацій не надає ефективної реалізації пошуку за ключем.

# Біноміальні дерева (binomial trees)

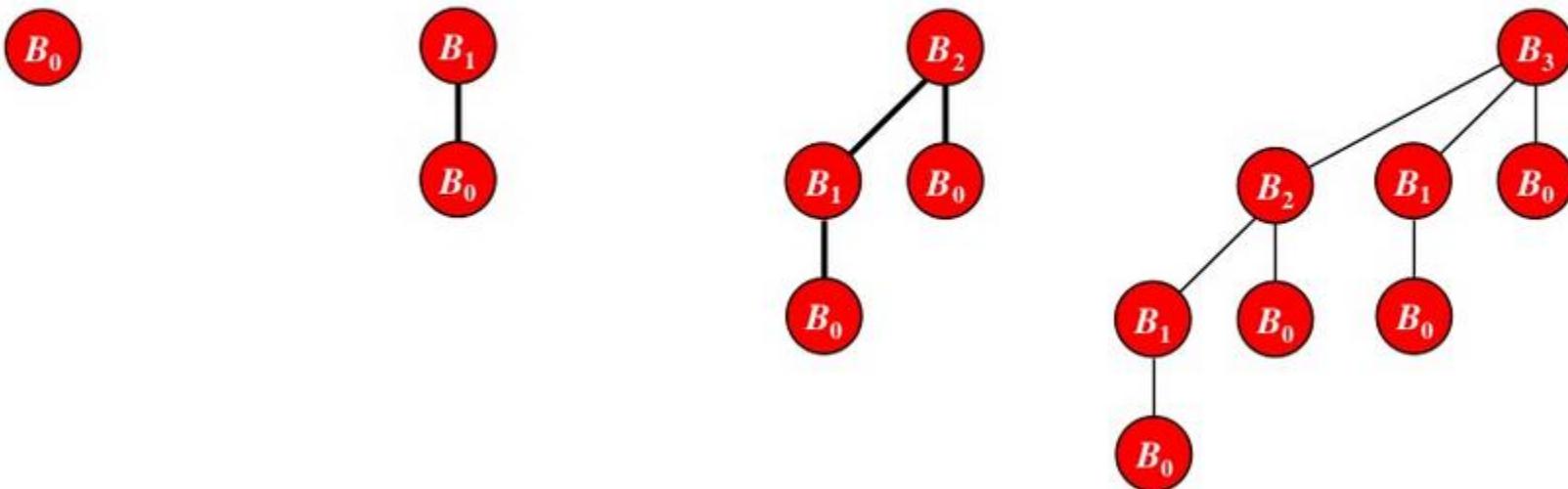
- Впорядковане дерево, що визначається рекурсивно.
- Біноміальне дерево  $B_0$  порядку 0 складається з єдиної вершини.
- Біноміальне дерево  $B_k$  порядку  $k$  складається з двох зв'язаних біноміальних дерев  $B_{k-1}$ : корінь одного є крайнім лівим сином іншого.

$$(n = 2^0 = 1)$$

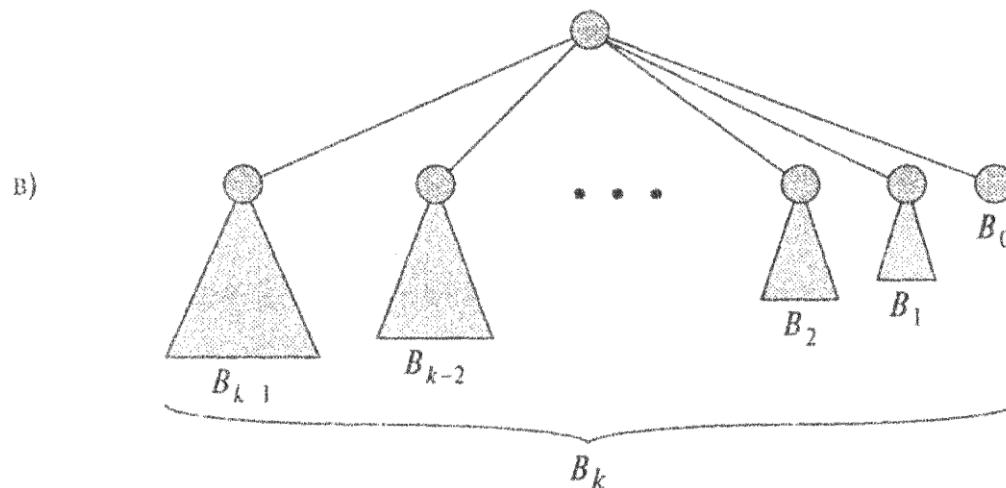
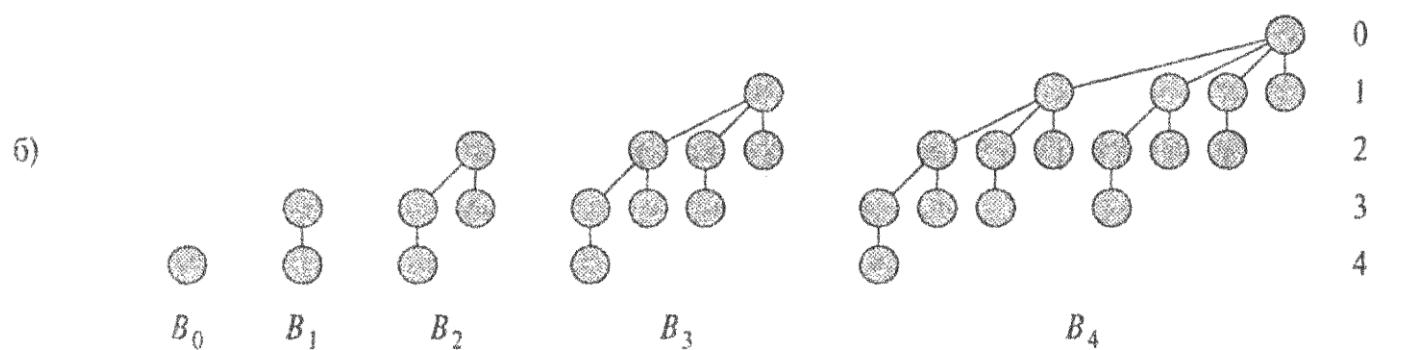
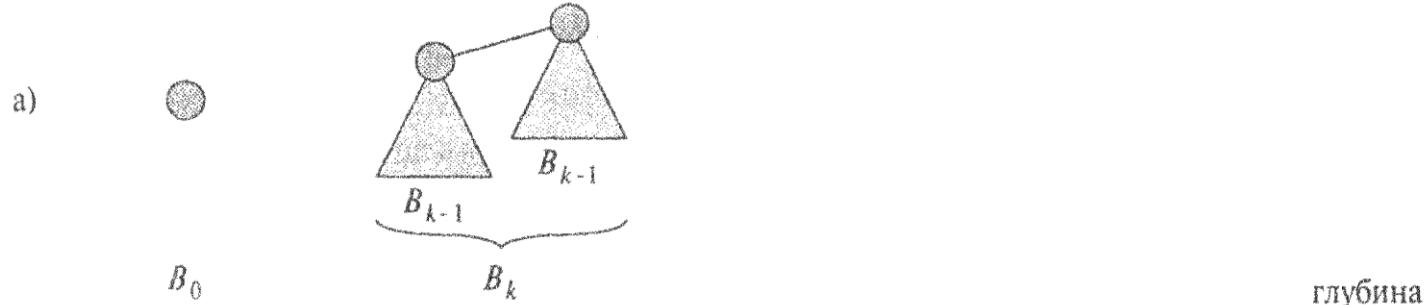
$$(n = 2^1 = 2)$$

$$(n = 2^2 = 4)$$

$$(n = 2^3 = 8)$$



# Біноміальні дерева (binomial trees)



# Біноміальні дерева (binomial trees)

## Лема (властивості біноміальних дерев)

Біноміальне дерево  $B_k$

1. має  $2^k$  вузлів;
2. має висоту  $k$ ;
3. має рівно  $\binom{k}{i}$  вузлів на глибині  $i = 0, 1, \dots, k$ ;
4. має корінь степені  $k$ , а степінь синів менша степеня кореня; при цьому якщо синів пронумерувати зліва направо числами  $(k-1), (k-2), \dots, 0$ , то  $i$ -й син є коренем біноміального дерева  $B_i$ .

# Біноміальні дерева (binomial trees)

Доведення. Індукція по  $k$ .

Для бази індукції  $B_0$  перевірка тривіальна.

Нехай всі властивості виконуються для  $B_{k-1}$ .

1. Біноміальне дерево  $B_k$  складається з двох копій  $B_{k-1}$ , тому містить  $2^{k-1} + 2^{k-1} = 2^k$  вузлів.
2. Виходячи зі способу зв'язування двох копій  $B_{k-1}$  в дерево  $B_k$ , глибина останнього на одиницю перевищує глибину  $B_{k-1}$ . Враховуючи припущення індукції, висота  $B_k$  дорівнює  $(k-1)+1=k$ .

# Біноміальні дерева (binomial trees)

## Доведення (далі)

3. Нехай  $D(k, i)$  – кількість вузлів на глибині  $i$  біноміального дерева  $B_k$ . Виходячи зі способу зв'язування двох копій  $B_{k-1}$  в дерево  $B_k$ , кількість вузлів біноміального дерева  $B_k$  на глибині  $i$  дорівнює кількості вузлів біноміального дерева  $B_{k-1}$  на глибині  $i$  плюс кількість вузлів у  $B_{k-1}$  на глибині  $(i-1)$ :

$$D(k, i) = D(k - 1, i) + D(k - 1, i - 1) =$$

$$= \binom{k-1}{i} + \binom{k-1}{i-1} = \binom{k}{i}.$$

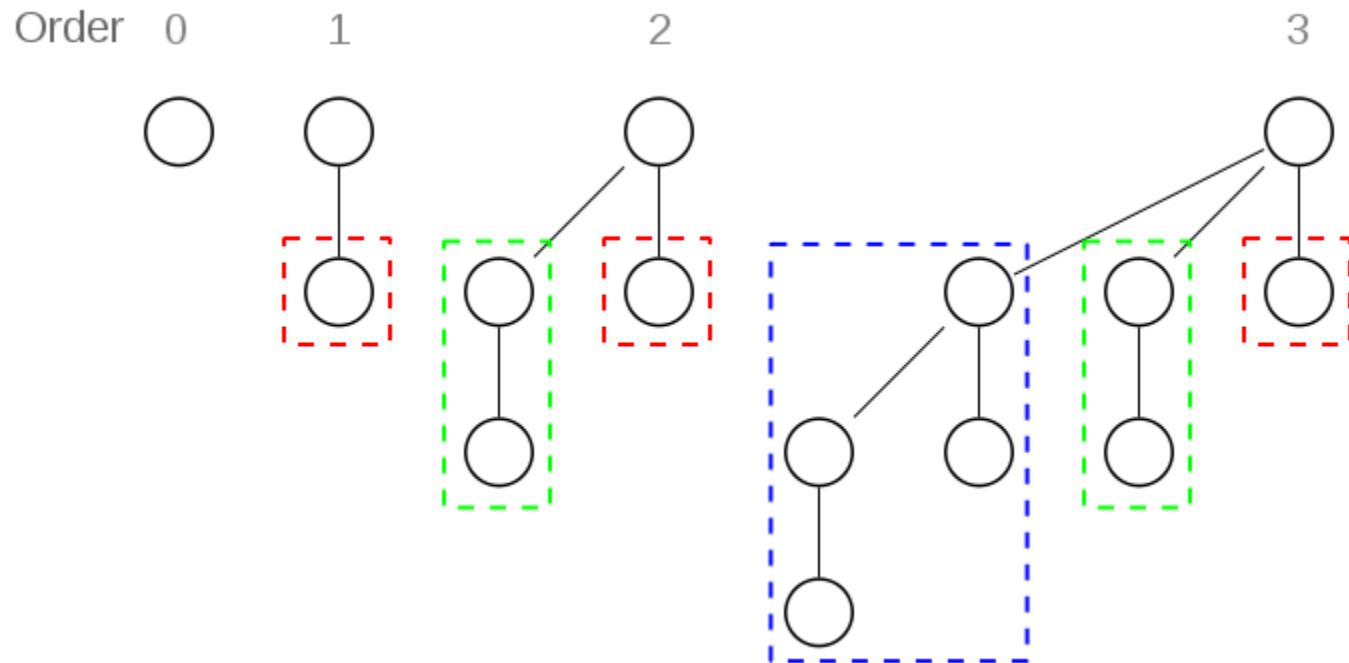
# Біноміальні дерева (binomial trees)

## Доведення (далі)

4. Корінь біноміального дерева  $B_k$  є єдиним вузлом, що перевищує степінь  $B_{k-1}$ : він має на одного сина більше. Оскільки (припущення індукції) корінь  $B_{k-1}$  має степінь  $(k-1)$ , то степінь кореня дерева  $B_k$  буде  $k$ . Якщо за припущенням синами біноміального дерева  $B_{k-1}$  будуть відповідно зліва направо біноміальні дерева  $B_{k-2}, B_{k-3}, \dots, B_0$ , то після прив'язки зліва ще одного дерева  $B_{k-1}$  синами новоствореного  $B_k$  стають  $B_{k-1}, B_{k-2}, \dots, B_0$ .

Наслідок. Максимальна степінь вузла біноміального дерева з  $n$  вершинами складає  $\lg n$ .

# Біноміальні дерева (binomial trees)



## Біноміальна піраміда (binomial heap)

Біноміальна піраміда (біноміальна купа)  $H$  – множина біноміальних дерев, що задовольняють властивостям біноміальних пірамід:

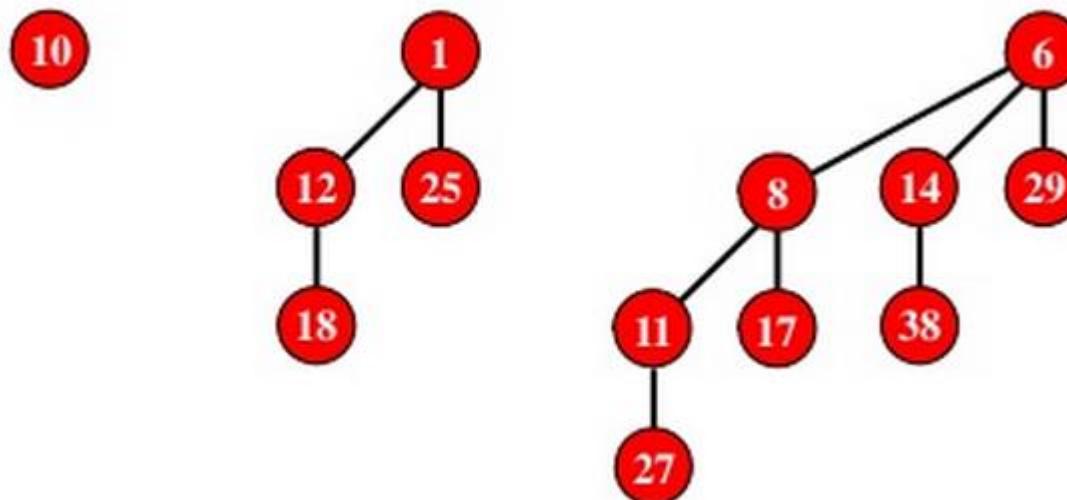
1. кожне біноміальне дерево в  $H$  є неспадаючою пірамідою (мінімальний елемент на вершині);
2. для довільного невід'ємного  $k$  в  $H$  існує не більше одного біноміального дерева відповідного порядку.

Біноміальна піраміда з  $n$  вузлів складається не більше ніж з  $(\lfloor \lg n \rfloor + 1)$  біноміальних дерев.

# Біноміальна піраміда (binomial heap)

- Нехай біноміальна купа має  $n$  вузлів.
- Одніці в двійковому записі  $n$  відповідають порядкам біноміальних дерев, що входять до цієї купи.

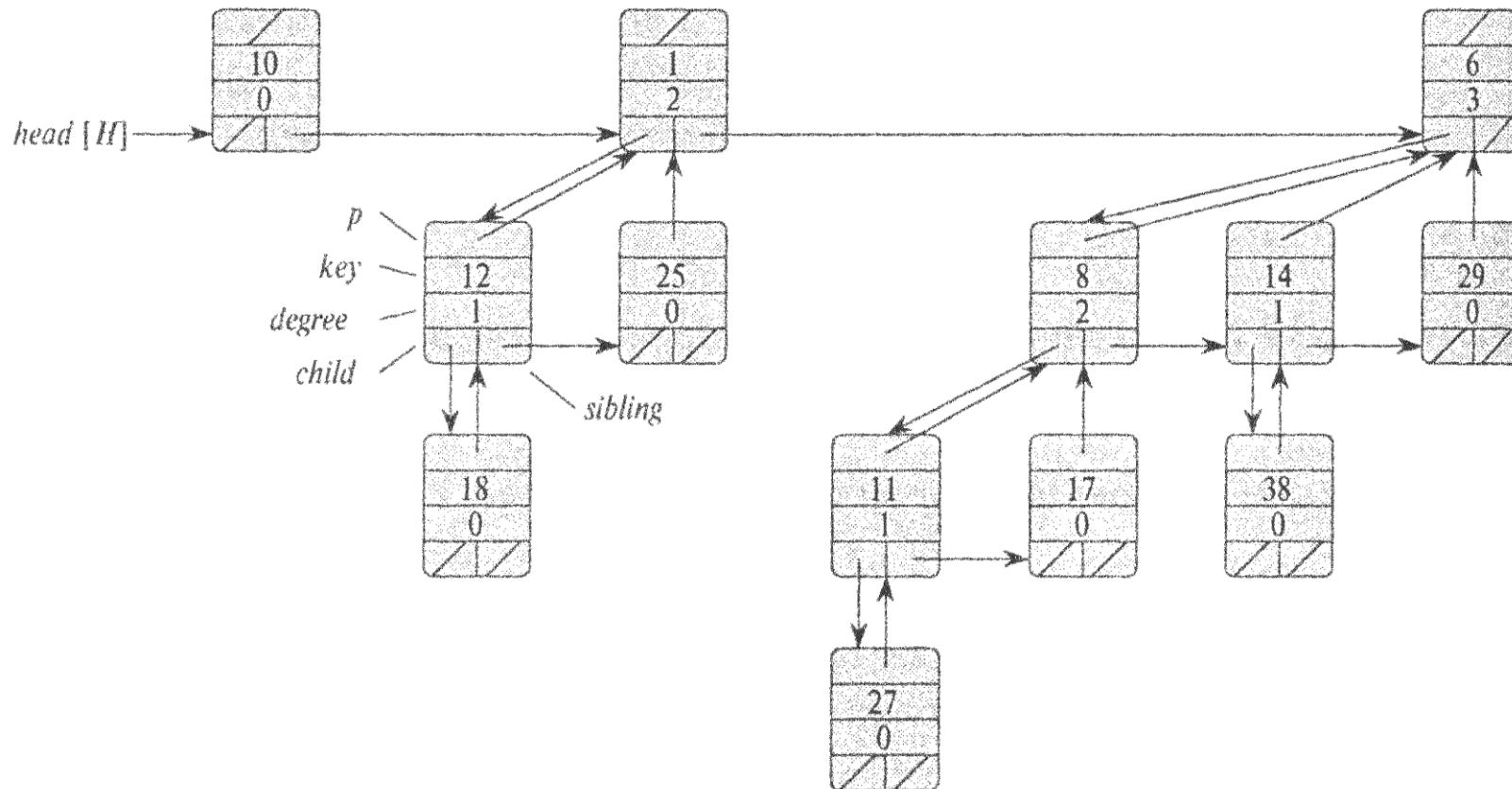
Наприклад, для 13 вузлів:  $13 = 1101_2$ . Біноміальна піраміда складатиметься з біноміальних дерев  $B_0$ ,  $B_2$  та  $B_3$  з 1, 4 і 8 вузлів відповідно.



# Представлення біноміальних пірамід

- Кожне біноміальне дерево зберігається у представленні з лівим дочірнім та правим сестринським вузлами.
- Ключ  $key[x]$ , вказівник на батька  $p[x]$ , вказівник на найлівішого сина  $child[x]$ , вказівник на правого брата  $sibling[x]$ , кількість дочірніх вузлів  $degree[x]$ .
- Біноміальна піраміда представлена списком коренів її біноміальних дерев впорядкованим за зростанням степенів дерев.
- Вказівник на перший корінь біноміальної піраміди  $H$ :  $head[H]$ .
- Якщо  $x$  – корінь, то  $sibling[x]$  вказує на наступний корінь у списку.

# Представлення біноміальних пірамід

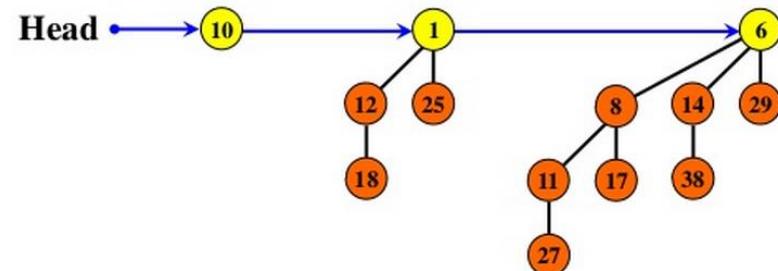


# Операції над біноміальними пірамідами

- Створення порожньої біноміальної піраміди. Час роботи  $\Theta(1)$ .
- Пошук мінімального ключа (вважаємо, що відсутні ключі зі значенням  $\infty$ ):

**BINOMIAL\_HEAP\_MINIMUM( $H$ )**

```
1   $y \leftarrow \text{NIL}$ 
2   $x \leftarrow \text{head}[H]$ 
3   $min \leftarrow \infty$ 
4  while  $x \neq \text{NIL}$ 
5      do if  $\text{key}[x] < min$ 
6          then  $min \leftarrow \text{key}[x]$ 
7           $y \leftarrow x$ 
8           $x \leftarrow \text{ sibling}[x]$ 
9  return  $y$ 
```



Час роботи  $O(\lg n)$ , бо перевіряємо не більше  $(\lfloor \lg n \rfloor + 1)$  коренів.

# Операції над біноміальними пірамідами

- Злиття двох біноміальних пірамід.
  1. Злити списки коренів  $H_1$  та  $H_2$  в упорядкований список (BINOMIAL\_HEAP\_MERGE).
  2. Відновити властивості біноміальної піраміди  $H$ .

Процедура BINOMIAL\_HEAP\_MERGE діє аналогічно етапу злиття в сортуванні злиттям, на кожному кроці переміщаючи в результатуючий список дерево меншого порядку. Час її роботи  $O(\lg n)$ , де  $n$  – сумарна кількість вершин в двох пірамідах.

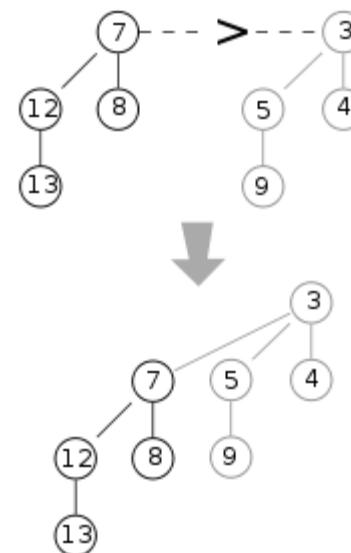
Після злиття списків коренів відомо, що купа  $H$  містить не більше двох коренів однакової степені, і вони стоять підряд. Тому будемо зв'язувати корені однієї степені поки всі корені не отримають різні степені.

# Операції над біноміальними пірамідами

Допоміжна операція зв'язування двох біноміальних дерев одного порядку  $B_{k-1}$  в біноміальне дерево  $B_k$  (дерево з коренем у "підчеплюється" до дерева  $z$ ):

**BINOMIAL\_LINK( $y, z$ )**

- 1  $p[y] \leftarrow z$
- 2  $sibling[y] \leftarrow child[z]$
- 3  $child[z] \leftarrow y$
- 4  $degree[z] \leftarrow degree[z] + 1$



Час виконання процедури  $O(1)$ .

# Операції над біноміальними пірамідами

**BINOMIAL\_HEAP\_UNION( $H_1, H_2$ )**

- 1  $H \leftarrow \text{MAKE\_BINOMIAL\_HEAP}()$
- 2  $head[H] \leftarrow \text{BINOMIAL\_HEAP\_MERGE}(H_1, H_2)$
- 3 Освобождение объектов  $H_1$  и  $H_2$ , но не  
списков, на которые они указывают
- 4 **if**  $head[H] = \text{NIL}$
- 5   **then return**  $H$
- 6  $prev-x \leftarrow \text{NIL}$
- 7  $x \leftarrow head[H]$
- 8  $next-x \leftarrow sibling[x]$
- 9 **while**  $next-x \neq \text{NIL}$
- 10     **do if** ( $degree[x] \neq degree[next-x]$ ) или  
          ( $sibling[next-x] \neq \text{NIL}$  и  $degree[sibling[next-x]] = degree[x]$ )

# Операції над біноміальними пірамідами

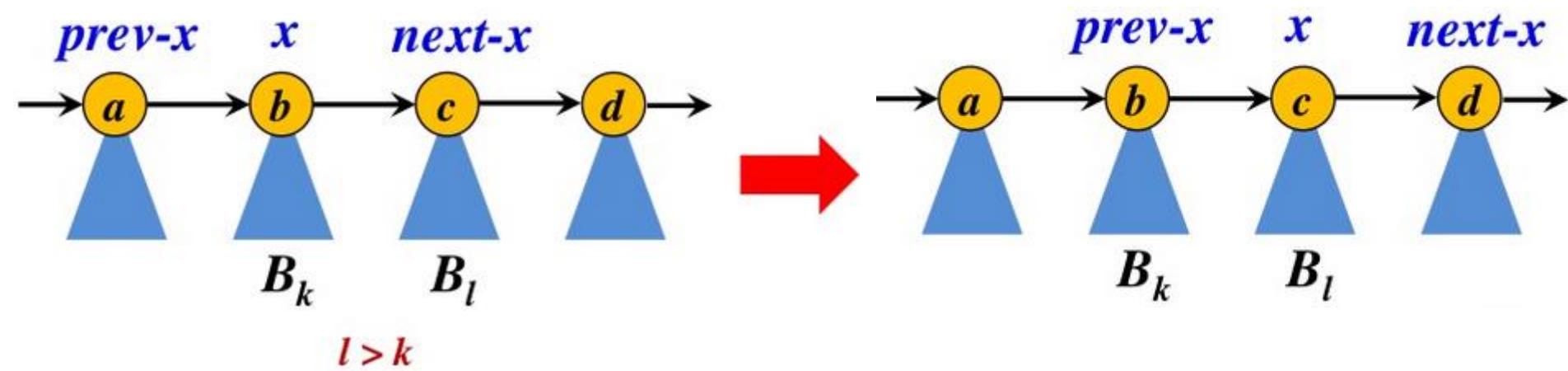
```
11      then prev-x  $\leftarrow x$                                 ▷ Случай 1 и 2
12          x  $\leftarrow next-x$                                 ▷ Случай 1 и 2
13      else if key[x]  $\leq key[next-x]$ 
14          then sibling[x]  $\leftarrow sibling[next-x]$           ▷ Случай 3
15              BINOMIAL_LINK(next-x, x)                  ▷ Случай 3
16          else if prev-x = NIL                         ▷ Случай 4
17              then head[H]  $\leftarrow next-x$                   ▷ Случай 4
18          else sibling[prev-x]  $\leftarrow next-x$           ▷ Случай 4
19              BINOMIAL_LINK(x, next-x)                  ▷ Случай 4
20          x  $\leftarrow next-x$                                 ▷ Случай 4
21      next-x  $\leftarrow sibling[x]$ 
22 return H
```

# Операції над біноміальними пірамідами

Для відновлення властивостей біноміальної піраміди використаємо три допоміжні вказівники:  $x$  – поточний корінь,  $prev\text{-}x$  – попередник  $x$ ,  $next\text{-}x$  – наступник  $x$ . Далі рухаємось списком коренів. Можливі ситуації:

## Випадок 1.

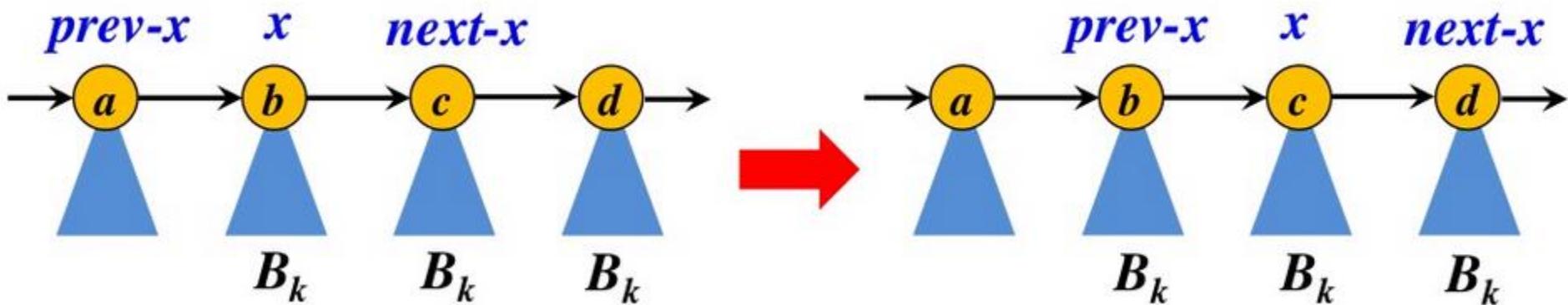
$$degree[x] \neq degree[next\text{-}x]$$



# Операції над біноміальними пірамідами

## Випадок 2.

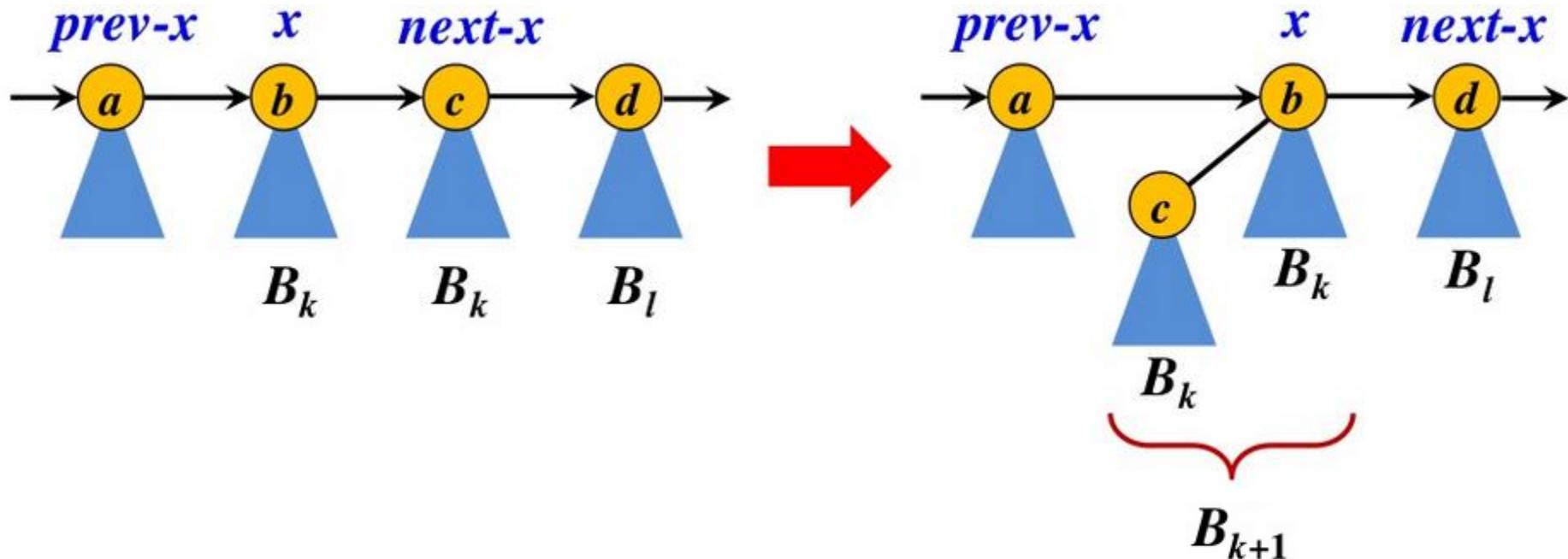
$$\text{degree}[x] = \text{degree}[\text{next-}x] = \text{degree}[\text{ sibling}[\text{next-}x]]$$



# Операції над біноміальними пірамідами

## Випадок 3.

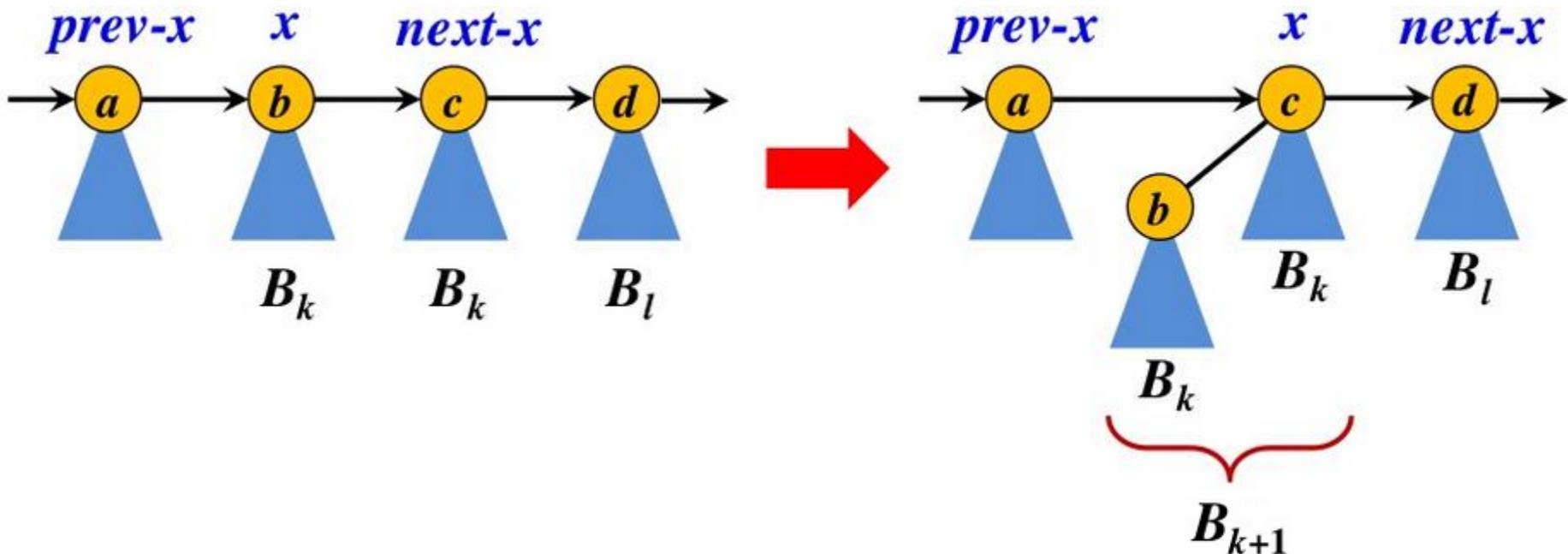
$degree[x] = degree[next-x] \neq degree [sibling[next-x]]$ ,  
 $key[x] \leq key[next-x]$



# Операції над біноміальними пірамідами

## Випадок 4.

$degree[x] = degree[next-x] \neq degree [sibling[next-x]]$ ,  
 $key[x] > key[next-x]$



## Операції над біноміальними пірамідами

Порахуємо час роботи BINOMIAL\_HEAP\_UNION.

Нехай біноміальна купа  $H_1$  містить  $n_1$  вузлів, а  $H_2$  –  $n_2$  вузлів та  $n_1 + n_2 = n$ .

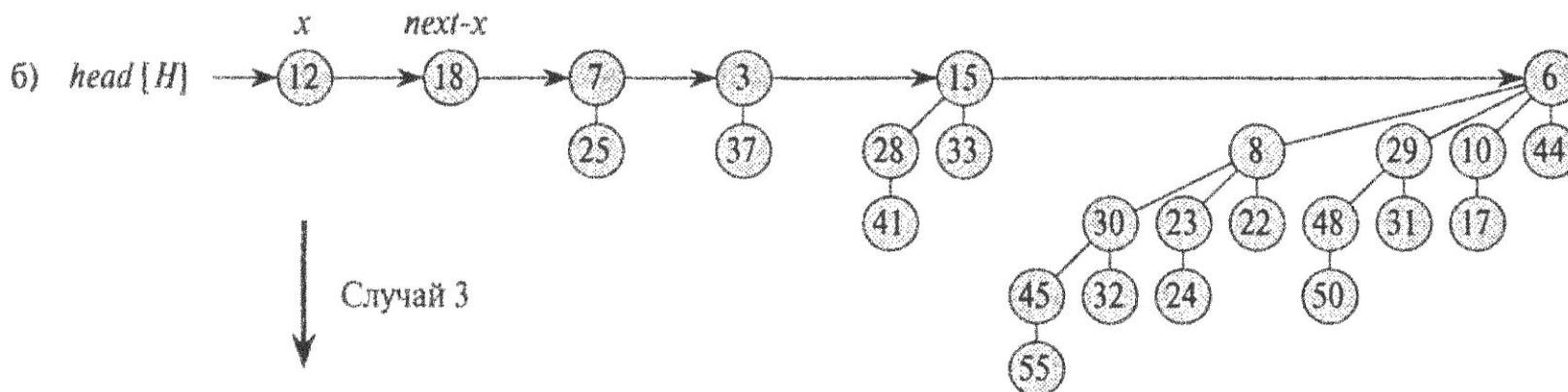
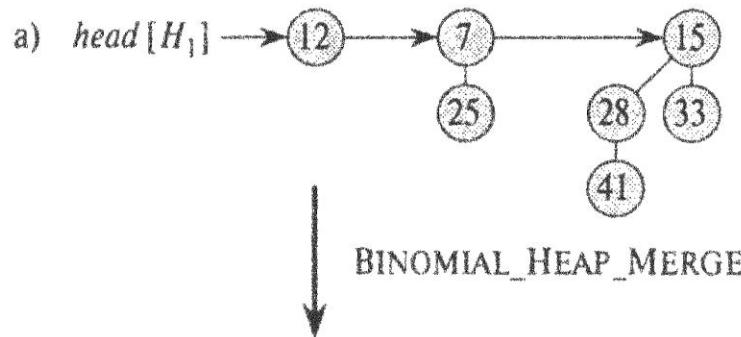
Тоді  $H_1$  має максимум  $\lfloor \log n_1 \rfloor + 1$  корінь, а  $H_2$  –  $\lfloor \log n_2 \rfloor + 1$  корінь, тому в  $H$  буде не більше  $\lfloor \log n_1 \rfloor + \lfloor \log n_2 \rfloor + 2 \leq 2\lfloor \log n \rfloor + 2 = O(\log n)$  коренів – час роботи BINOMIAL\_HEAP\_MERGE.

Кожна ітерація циклу виконується за константний час. Оскільки ми проходимо весь список, то ітерацій буде не більше  $\lfloor \log n_1 \rfloor + \lfloor \log n_2 \rfloor + 2$ .

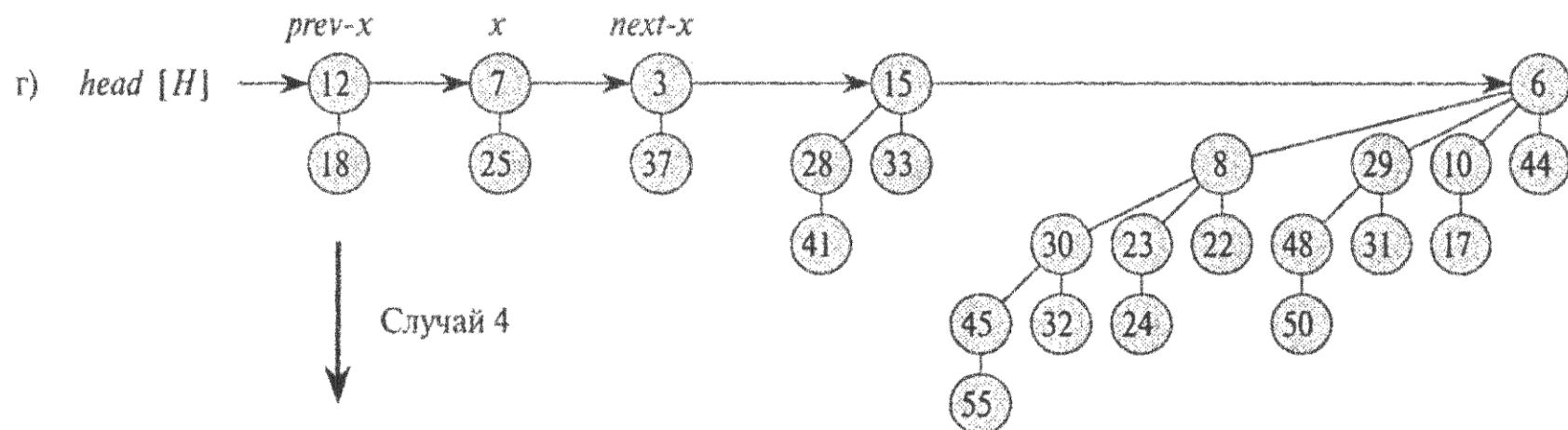
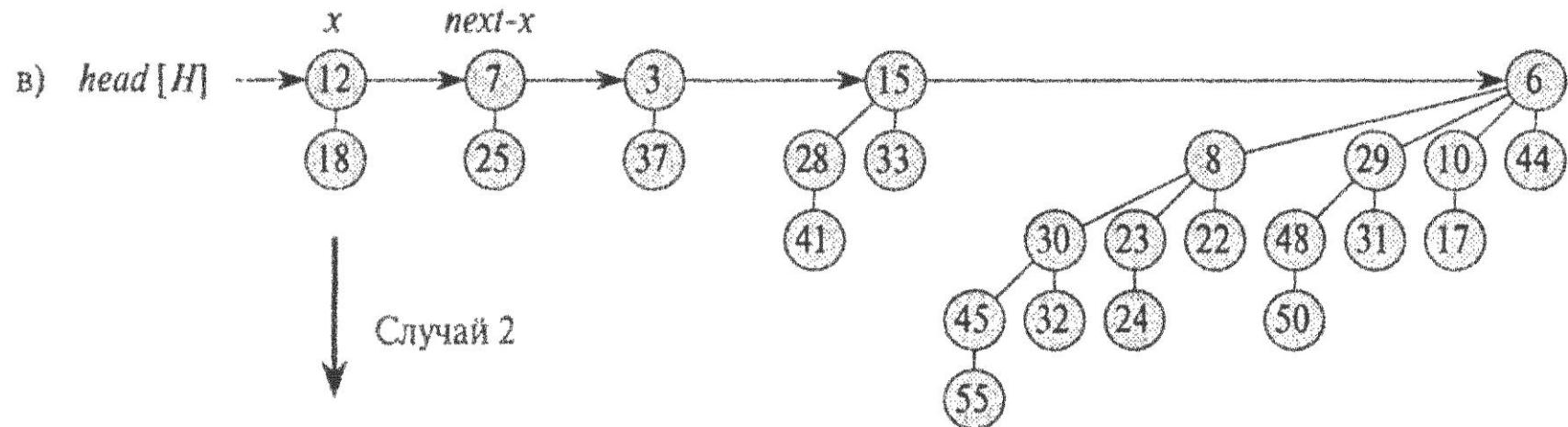
Отже, загальний час виконання BINOMIAL\_HEAP\_UNION складе  $O(\log n)$ .

# Операції над біноміальними пірамідами

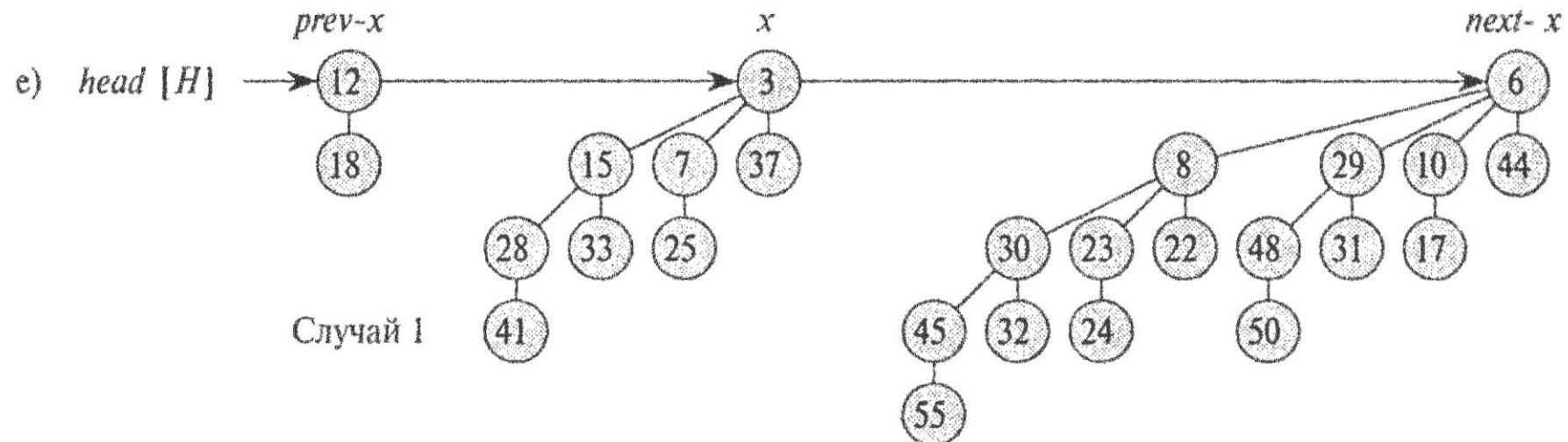
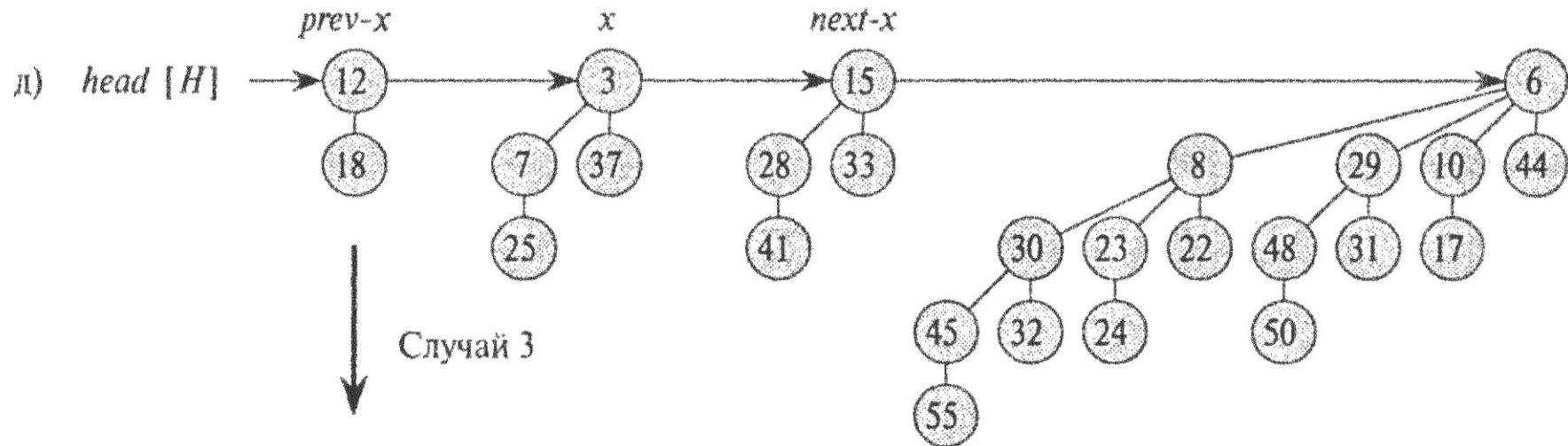
Приклад злиття двох біноміальних куп.



# Операції над біноміальними пірамідами



# Операції над біноміальними пірамідами



# Операції над біноміальними пірамідами

- Вставка вузла.

Створюється біноміальна піраміда з одним вузлом (час  $O(1)$ ) та зливається з початковою пірамідою (загальний час  $O(\lg n)$ ).

**BINOMIAL\_HEAP\_INSERT( $H, x$ )**

- 1  $H' \leftarrow \text{MAKE\_BINOMIAL\_HEAP}()$
- 2  $p[x] \leftarrow \text{NIL}$
- 3  $child[x] \leftarrow \text{NIL}$
- 4  $sibling[x] \leftarrow \text{NIL}$
- 5  $degree[x] \leftarrow 0$
- 6  $head[H'] \leftarrow x$
- 7  $H \leftarrow \text{BINOMIAL\_HEAP\_UNION}(H, H')$

# Операції над біноміальними пірамідами

- Вилучення мінімального вузла.

`BINOMIAL_HEAP_EXTRACT_MIN( $H$ )`

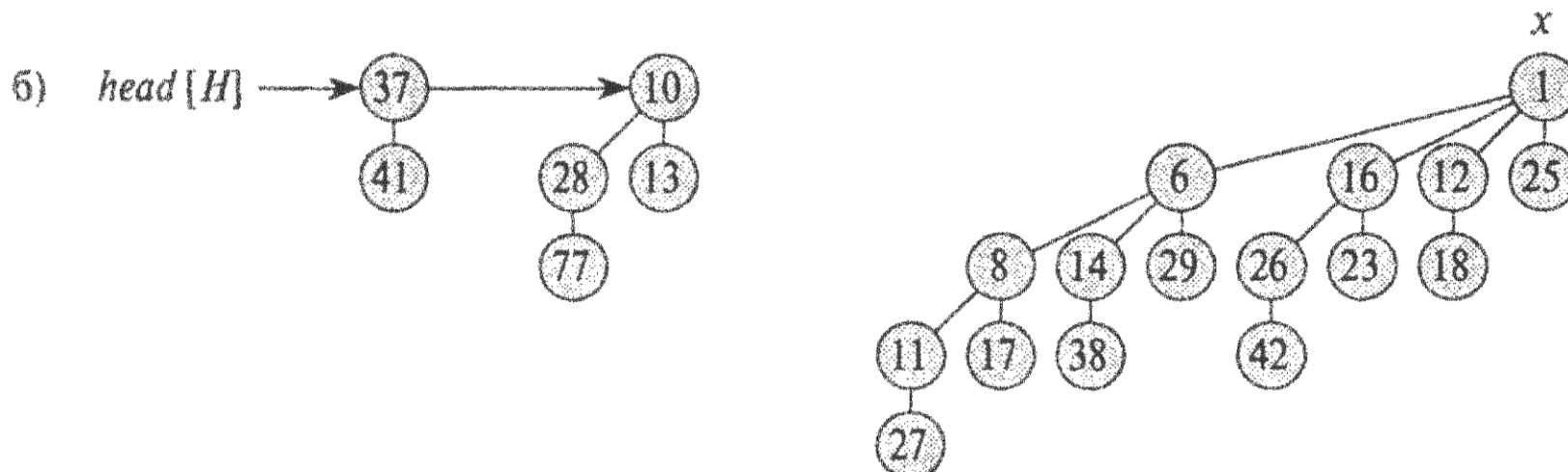
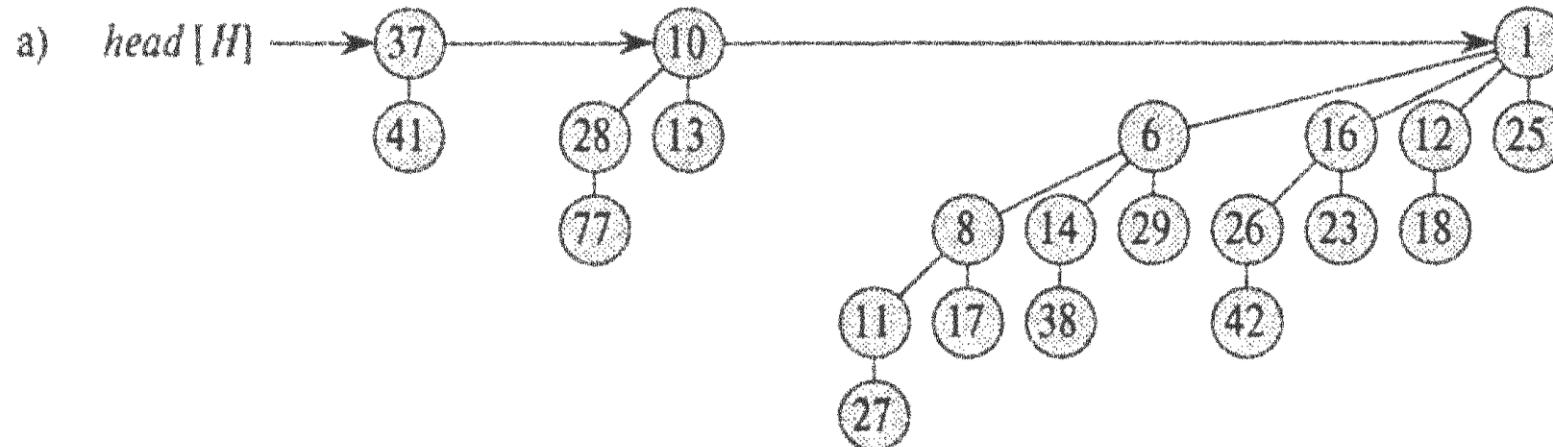
- 1 Поиск корня  $x$  с минимальным значением ключа в списке корней  $H$ , и удаление  $x$  из списка корней  $H$
- 2  $H' \leftarrow \text{MAKE\_BINOMIAL\_HEAP}()$
- 3 Обращение порядка связанного списка дочерних узлов  $x$ , установка поля  $r$  каждого дочернего узла равным NIL и присвоение указателю  $head[H']$  адреса заголовка получающегося списка
- 4  $H \leftarrow \text{BINOMIAL\_HEAP\_UNION}(H, H')$
- 5 **return**  $x$

Видаляємо мінімальний корінь. Утворюємо з його синів нову біноміальну піраміду (перестановкою у зворотному порядку). Зливаємо новоутворену піраміду з вихідною.

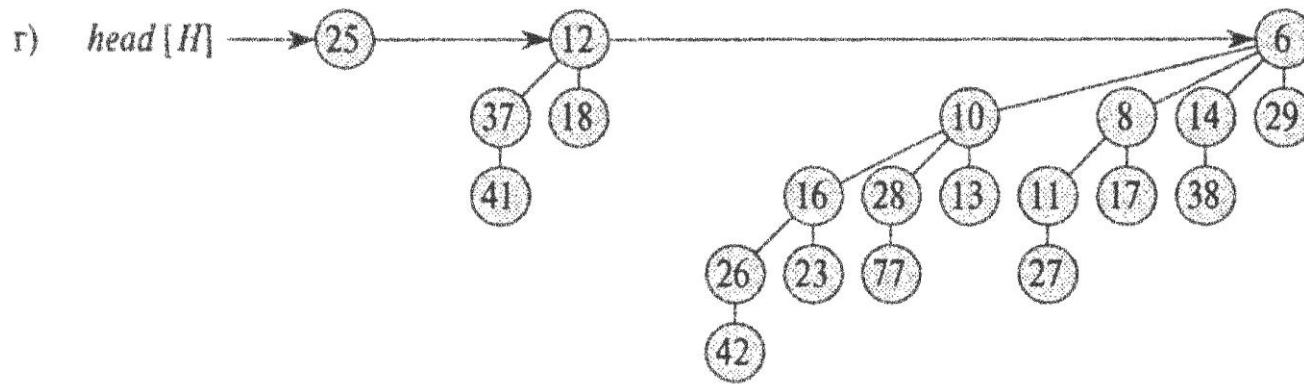
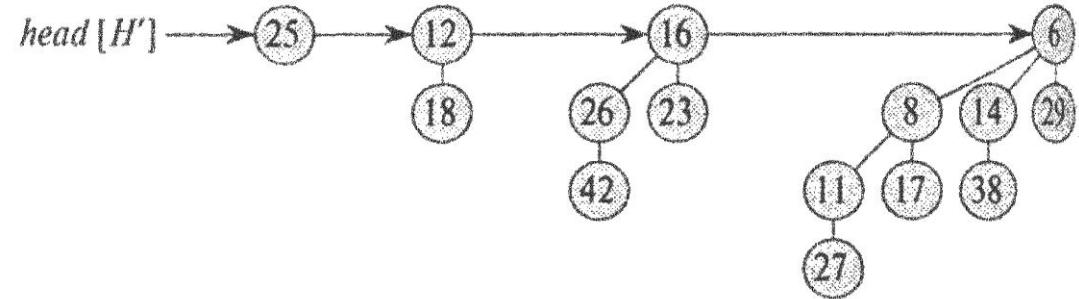
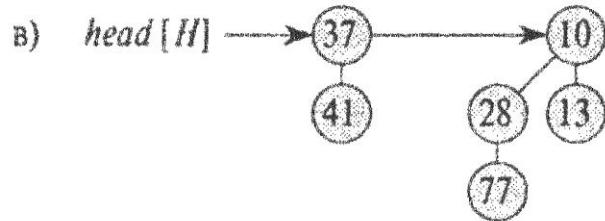
Час роботи процедури  $O(\lg n)$ .

# Операції над біноміальними пірамідами

Приклад вилучення мінімального вузла



# Операції над біноміальними пірамідами



# Операції над біноміальними пірамідами

- Зменшення ключа.

Значення ключа замінюється на менше. Після цього рухаємось у напрямку кореня, обмінюючи значення, якщо порушенна умова неспадаючої піраміди.

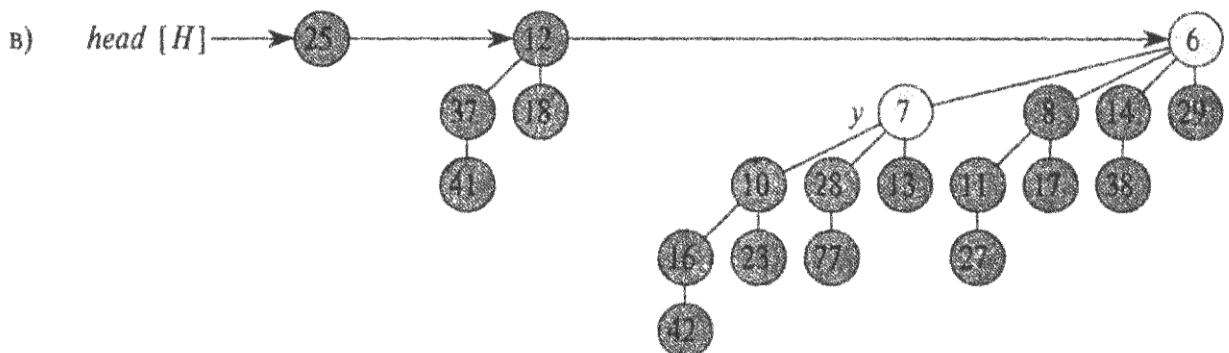
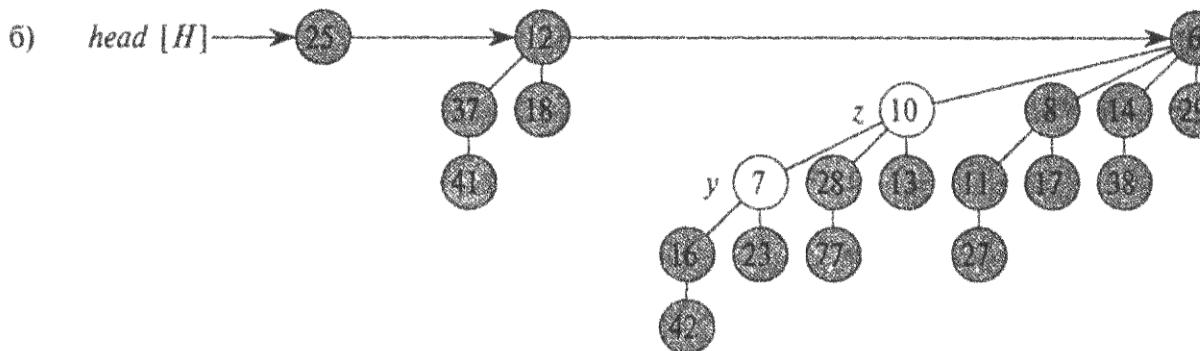
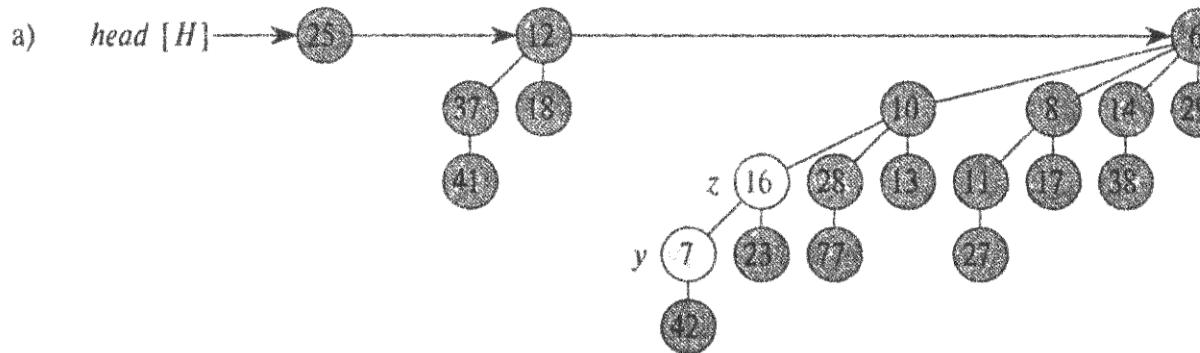
**BINOMIAL\_HEAP\_DECREASE\_KEY( $H, x, k$ )**

- 1 **if**  $k > key[x]$
- 2     **then error** “Новый ключ больше текущего”
- 3      $key[x] \leftarrow k$
- 4      $y \leftarrow x$
- 5      $z \leftarrow p[y]$
- 6     **while**  $z \neq \text{NIL}$  и  $key[y] < key[z]$
- 7         **do** Обменять  $key[y] \leftrightarrow key[z]$ 
  - 8             ▷ Если  $y$  и  $z$  содержат сопутствующую
  - 9             ▷ информацию, обменять также и ее
- 10          $y \leftarrow z$
- 11          $z \leftarrow p[y]$

Процедура виконується максимум за час  $O(\lg n)$ .

# Операції над біноміальними пірамідами

## Приклад зменшення ключа (зменшення до 7 в позиції у)



# Операції над біноміальними пірамідами

- Видалення ключа.

`BINOMIAL_HEAP_DELETE( $H, x$ )`

- 1 `BINOMIAL_HEAP_DECREASE_KEY( $H, x, -\infty$ )`
- 2 `BINOMIAL_HEAP_EXTRACT_MIN( $H$ )`

Вважаємо, що жоден ключ не може містити ключ  $-\infty$ .

Ключ у вузлі для видалення робиться  $-\infty$ . При цьому розглянута вершина стає одним з коренів, мінімальним, і може бути вилучена за допомогою процедури `BINOMIAL_HEAP_EXTRACT_MIN`.

Сумарний час виконання складе  $O(\lg n)$ .

# Амортизаційний аналіз (amortized analysis)

- Амортизаційний аналіз – метод підрахунку часу, необхідного для виконання послідовності операцій над структурою даних.
- Час усереднюється по всіх виконуваних операціях, і аналізується середня продуктивність операцій в гіршому випадку.
- Використовується, щоб показати, що навіть якщо деякі з операцій послідовності є дорогими, то при усередненні по всіх операціях середня їх вартість буде невеликою за рахунок того, що ці операції нечасто зустрічаються.
- Отримана оцінка не є ймовірнісною: це оцінка середнього часу виконання операцій для найгіршого випадку.

# Амортизаційний аналіз

Основні методи амортизаційного аналізу:

- метод усереднення (метод групового аналізу, *aggregate analysis*);
- метод передплати (метод бухгалтерського обліку, *accounting method*);
- метод потенціалів (*potential method*).

# Амортизаційний аналіз

## Груповий аналіз

- Якщо в найгіршому випадку загальний час виконання послідовності всіх  $n$  операцій сумарно дорівнює  $T(n)$ , то в найгіршому випадку *середня* (або *амортизована*) ціна однієї операції визначається співвідношенням  $T(n)/n$ .
- Знайдена амортизована вартість може бути застосована до всіх операцій послідовності, навіть якщо вони різnotипні.

## Амортизаційний аналіз

Розглянемо стек. Основні його операції PUSH та POP виконуються за час  $O(1)$ , приймемо цей час за 1.

Повна вартість послідовності з  $n$  операцій PUSH та POP дорівнюватиме  $n$ , їх фактичний час виконання  $\Theta(n)$ .

Додамо до стеку операцію MULTIPOP( $S, k$ ), що видалятиме зі стеку  $k$  елементів (або всі, що залишилися, якщо їх менше за  $k$ ):

**MULTIPOP( $S, k$ )**

```
1 while STACK_EMPTY( $S$ ) = FALSE и  $k > 0$ 
2     do POP( $S$ )
3          $k \leftarrow k - 1$ 
```

Час роботи операції прямо залежить від кількості виконань операції POP .

# Амортизаційний аналіз

$\text{MULTIPOP}(S, k)$

```
1 while  $\text{STACK\_EMPTY}(S) = \text{FALSE}$  и  $k > 0$ 
2     do  $\text{POP}(S)$ 
3          $k \leftarrow k - 1$ 
```

Нехай стек реально містить  $s$  елементів.

Кількість ітерацій циклу **while** дорівнюватиме числу  $\min(s, k)$  елементів, які вийдуть зі стека.

При кожній ітерації виконується один виклик  $\text{POP}$ , тому повна вартість процедури  $\text{MULTIPOP}$  дорівнюватиме  $\min(s, k)$ , а фактичний час буде лінійною функцією від цієї величини.

## Амортизаційний аналіз

Якою буде вартість послідовності з  $n$  операцій PUSH, POP та MULTIPOP за умови порожнього спочатку стека?

В найгіршому випадку вартість операції MULTIPOP  $O(n)$ , бо в стеку не більше  $n$  об'єктів. Це є верхньою границею часу роботи довільної стекової операції.

Тоді в найгіршому випадку вартість послідовності з  $n$  операцій складе  $O(n^2)$ .

Знайдена таким чином оцінка занадто груба. Розглядалася найгірша вартість кожної операції окремо.

Проаналізуємо границю часу виконання *послідовності* з  $n$  операцій в найгіршому випадку.

## Амортизаційний аналіз

Якою буде вартість послідовності з  $n$  операцій PUSH, POP та MULTIPOP за умови порожнього спочатку стека?

Перед тим, як щось дістати зі стека, його треба туди помістити.

Сумарна кількість операцій POP (включаючи виклики з MULTIPOP) для непорожнього стека не може перевищити кількість операцій PUSH, яких буде не більше  $n$ .

Для будь-якого  $n$  послідовність з  $n$  операцій PUSH, POP та MULTIPOP займе час  $O(n)$ .

Тому середня вартість кожної з операцій буде  $O(n) / n = O(1)$ .

# Амортизаційний аналіз

## Метод бухгалтерського обліку

- Кожна операція має свою нараховану амортизовану вартість та фактичну вартість.
- *Кредит* – додатна різниця між амортизованою та фактичною вартістю операції.
- Наявний кредит можна використати на покриття від'ємної різниці між амортизованою та фактичною вартістю інших операцій.
- Для всіх послідовностей операцій їх повна амортизована вартість має бути верхньою границею їх повної фактичної вартості, а повний кредит завжди невід'ємним.

# Амортизаційний аналіз

Фактичні вартості операцій для розглянутого стеку:

PUSH: 1

POP: 1

MULTIPOP:  $\min(k, s)$ , де  $k$  – аргумент процедури,  $s$  – поточна кількість елементів у стеку.

Візьмемо наступні амортизовані вартості:

PUSH: 2

POP: 0

MULTIPOP: 0.

В нашому випадку всі амортизовані вартості дорівнюють  $O(1)$ .

При цьому фактична вартість MULTIPOP – величина змінна.

## Амортизаційний аналіз

Покажемо, що довільну послідовність стекових операцій можна оплатити через нарахування амортизованих вартостей.

Уявімо стек як стос тарілок у кафе. При додаванні тарілки 1 грошова одиниця витрачається на оплату самої операції PUSH, а ще 1 грошова одиниця (з нарахованих 2-х) залишається в запасі, ніби на тарілці.

В довільний момент часу кожній тарілці стека відповідає 1 грошова одиниця кредиту.

Запасна одиниця кредиту на тарілці піде на оплату її діставання зі стеку. На операцію POP плата не нараховується, а її фактична вартість оплачується кредитом. Завдяки невеликій переоцінці операції PUSH зникає необхідність нарахувань на операцію POP.

## Амортизаційний аналіз

Так само не потрібно нараховувати плату на операцію MULTIPROP.

При діставанніожної тарілки з неї береться 1 грошова одиниця кредиту і витрачається на оплату фактичної вартості POP.

Стек завжди містить невід'ємну кількість тарілок, тому попередньо нарахованої суми завжди достатньо для оплати операції MULTIPROP і сума кредиту завжди буде невід'ємною.

Отже для довільних послідовностей з  $n$  операцій PUSH, POP та MULTIPROP їх повна амортизована вартість  $O(n)$  є верхньою границею їх повної фактичної вартості.

# Амортизаційний аналіз

## Метод потенціалів

- Введемо для кожного стану структури даних  $D_i$  величину  $\Phi(D_i)$  – *потенціал*. Спочатку потенціал дорівнює  $\Phi(D_0)$ , а після  $i$ -ї операції  $\Phi(D_i)$ .
- Амортизована вартість  $\hat{c}_i$ ожної операції  $c_i$  – її фактична вартість плюс приріст потенціалу в результаті її виконання.
- Тоді повна амортизована вартість  $n$  операцій

$$\sum_{i=1}^n \hat{c}_i = \sum_{i=1}^n (c_i + \Phi(D_i) - \Phi(D_{i-1})) = \sum_{i=1}^n c_i + \Phi(D_n) - \Phi(D_0).$$

- Якщо  $\Phi$  можна визначити так, що  $\Phi(D_n) \geq \Phi(D_0)$ , то повна амортизована вартість буде верхньою границею повної фактичної вартості. Якщо  $n$  невідоме: умова  $\Phi(D_i) \geq \Phi(D_0)$  для всіх  $i$  (напр.  $\Phi(D_0) = 0$ )

# Амортизаційний аналіз

Для розглянутого стеку визначимо функцію потенціалу  $\Phi$  як кількість елементів в цьому стеку.

Для порожнього стека  $\Phi(D_0) = 0$ . Оскільки кількість об'єктів у стеку невід'ємна, то  $\Phi(D_i) \geq 0 = \Phi(D_0)$  для всіх  $i$ .

Нехай на  $i$ -му кроці стек містить  $s$  елементів.

PUSH: різниця потенціалів  $\Phi(D_i) - \Phi(D_{i-1}) = (s+1) - s = 1$ ;  
амортизована вартість  $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 1+1 = 2$ .

POP: різниця потенціалів  $\Phi(D_i) - \Phi(D_{i-1}) = -1$ ;  
амортизована вартість  $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 1-1 = 0$ .

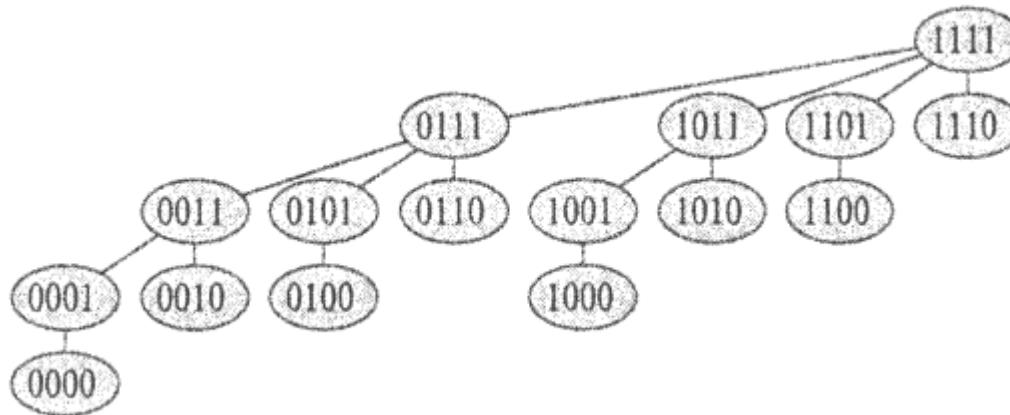
MULTIPOP: видалили  $r = \min(k, s)$  елементів;  
різниця потенціалів  $\Phi(D_i) - \Phi(D_{i-1}) = -r$ ;  
амортизована вартість  $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = r-r = 0$ .

Для всіх операцій амортизовані вартості дорівнюють  $O(1)$ .

Тому повна амортизована вартість для  $n$  операцій буде  $O(n)$ .

## Запитання і завдання

- Припустимо, вузли біноміального дерева  $B_k$  помічені двійковими числами при обході в оберненому порядку:



Нехай вузол  $x$  знаходиться на глибині  $i$ , відмічений числом  $l$ , та нехай  $j=k-i$ . Покажіть, що двійкове представлення  $x$  має  $j$  одиниць. Скільки  $k$ -рядків містить рівно  $j$  одиниць? Доведіть, що степінь  $x$  дорівнює кількості одиниць справа від найправішого нуля в двійковому представленні  $l$ .

## Запитання і завдання

- Доведіть коректність BINOMIAL\_HEAP\_UNION через наступний інваріант циклу:

На початку кожної ітерації **while** в рядках 9-21  $x$  вказує на корінь, який

- або єдиний корінь такого степеня,
- або перший з двох існуючих коренів такого степеня,
- або перший чи другий з трьох існуючих коренів такого степеня.

І крім того, всі корені – попередники попередника  $x$  в списку коренів – єдині свого степеня в списку коренів, і якщо попередник  $x$  має степінь, відмінний від степеня  $x$ , то такий степінь єдиний в списку коренів. Також, степені вузлів монотонно зростають в процесі проходу по списку коренів.

## Запитання і завдання

- Чому процедура `BINOMIAL_HEAP_MINIMUM` може некоректно працювати при існуванні ключа зі значенням  $\infty$ ? Переробіть її, щоб алгоритм працював коректно і за цих умов.
- Припустимо жодним чином не можливо представити ключ зі значенням  $-\infty$ . Перепишіть процедуру `BINOMIAL_HEAP_DELETE`, щоб вона коректно працювала за таких умов, час роботи має бути  $O(\lg n)$ .
- Який зв'язок між вставкою в біноміальну піраміду та збільшенням двійкового числа, що її представляє? Між злиттям двох біноміальних пірамід та сумуванням двох двійкових чисел? Відповівши на ці запитання, переробіть процедуру `BINOMIAL_HEAP_INSERT` так, щоб не відбувався виклик `BINOMIAL_HEAP_UNION`.