Алгоритми та складність

II семестр

Лекція 6

2-3-4-піраміди

- Різновид пірамід злиття.
- Кожен внутрішній вузол містить 2, 3 або 4 нащадка.
- Все листя розташоване на одній глибині.
- Ключі зберігаються лише в листі, кожен лист містить рівно один ключ *key*[*x*].
- Порядок ключів по листкам при переліченні їх зліва направо не зберігається.
- Кожен внутрішній вузол *x* зберігає у полі *small*[*x*] значення мінімального ключа серед листів піддерева з коренем *x*.
- Корінь *r* має поле *height*[*r*], що зберігає висоту.
- 2-3-4-піраміди зберігаються в оперативній пам’яті.
- Всі операції пірамід злиття потребують логарифмічного часу.

Піраміди злиття (нагадування)

Піраміди, що підтримують операцію злиття.

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

- MAKE_HEAP(): створення нової порожньої піраміди.
- INSERT(H,*x*): вставка готового вузла *x* в піраміду H.
- MINIMUM(H): повертає вказівник на вузол піраміди H з найменшим ключем.
- EXTRACT_MIN(H): видаляє вузол піраміди H з найменшим ключем і повертає вказівник на нього.
- UNION(H_1,H_2): повертає нову піраміду – результат злиття H_1, H_2 (вони не зберігаються).
- DECREASE_KEY(H,*x*,*k*): присвоєння вузлу *x* піраміди H значення ключа *k*, що є меншим за поточне.
- DELETE(H,*x*): видалення вузла *x* з піраміди H.

Піраміди Фібоначчі

- Слабша структура, ніж у біноміальної піраміди.
- Амортизований час операцій, що не використовують видалення, дорівнює O(1).
- Можуть виявитися корисними в алгоритмах, де частка операцій EXTRACT_MIN та DELETE відносно мала (ряд алгоритмів на графах: наприклад, пошук мінімального кістякового дерева, найкоротший шлях з однієї вершини, алгоритми, що використовують DECREASE_KEY для кожного ребра в майже повних графах).
- Складність реалізації, порівняно з бінарними (та *k*-арними) пірамідами, вища.
- Значення констант у формулах часу виконання також високі.

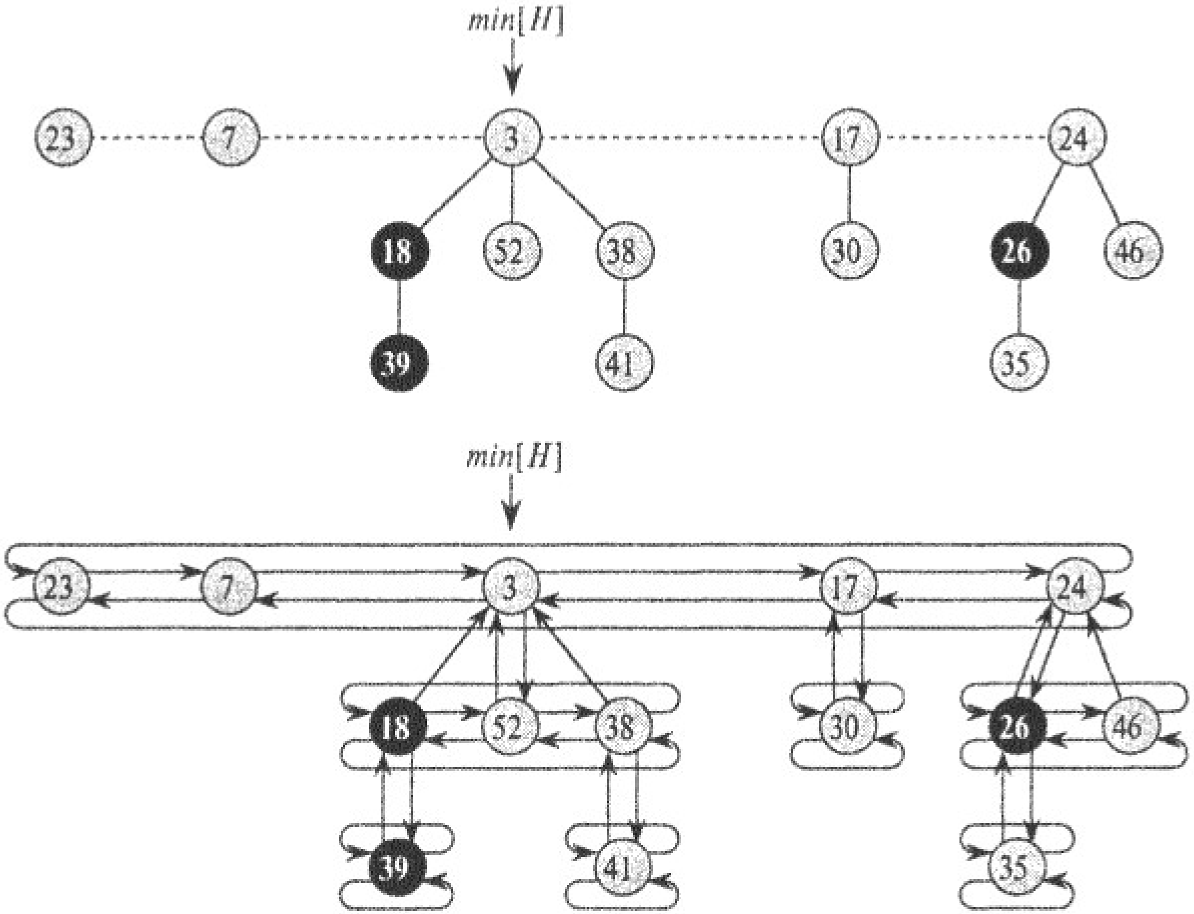
Піраміди Фібоначчі

- Невпорядкований набір дерев з коренем, кожне з яких є незростаючою пірамідою.
- Кожен вузол *x* містить вказівник на батька *p*[*x*] і вказівник на одного з синів *child*[x].
- Дочірні вузли вершини об'єднані в двозв'язний циклічний список (*child list*). (Операції видалення елемента та об'єднання двох таких списків займають константний час.)
- Кожен дочірній вузол *y* має вказівники на лівого та правого своїх братів: *left*[*y*], *right*[*y*]. Порядок братських вузлів довільний.

Піраміди Фібоначчі

- Кожен вузол містить поле кількості синів *degree*[*x*] та логічне поле *mark*[*x*] – ознаку наявності втрат дочірніх вузлів з моменту, коли *x* сам став дочірнім вузлом.
- Значення *mark*[*x*] для новостворених вузлів FALSE; наявна мітка знімається, коли вузол стає дочірнім.
- Всі корені дерев також зв'язані в двозв'язний циклічний список (*root list*). Звернення до піраміди H іде через корінь дерева з мінімальним ключем *min*[H] (мінімальний вузол).
- Кількість вузлів піраміди зберігається в *n*[H].

Піраміди Фібоначчі



Піраміда Фібоначчі з 5 деревами та 14 вузлами

Піраміди Фібоначчі: функція потенціалу

- Для піраміди Фібоначчі H позначимо

  t(H) – кількість дерев у списку коренів H,

  m(H) – кількість вузлів з мітками в піраміді H.

Тоді потенціал піраміди визначається як

$$\Phi(H) = t(H) + 2m(H).$$

- Потенціал множини пірамід – сума потенціалів пірамід-складників.
- Вважаємо, що одиниці потенціалу достатньо для покриття вартості довільної операції часу O(1).
- На початку роботи піраміда порожня, тобто початковий потенціал дорівнює 0. Тоді в подальшому потенціал завжди буде залишатися невід’ємним.
- Верхня границя загальної амортизованої вартості є верхньою границею загальної фактичної вартості послідовності операцій.

Піраміди Фібоначчі: функція потенціалу

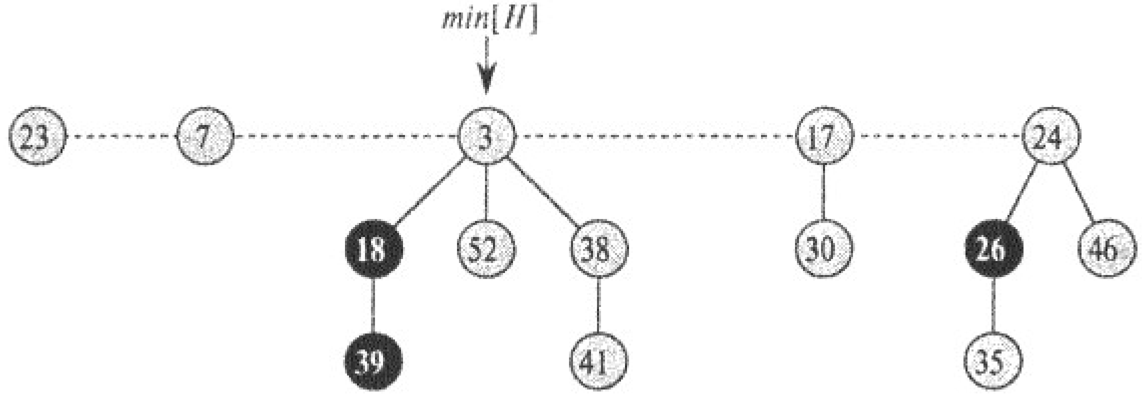
Верхня границя D(*n*) максимальної степені вузла піраміди Фібоначчі з *n* вузлів:

- піраміди без підтримки DECREASE_KEY та DELETE:

$$D(n) \le \lfloor \lg n \rfloor$$

- з підтримкою DECREASE_KEY та DELETE:

$$D(n) = \mathrm{O}(\lg n)$$



Потенціал піраміди: 5 + 2·3 = 11.

Операції над пірамідами злиття

- Якщо підтримуються лише операції MAKE_HEAP, INSERT, MINIMUM, EXTRACT_MIN та UNION, кожна піраміда Фібоначчі буде набором *невпорядкованих біноміальних дерев* (unordered binomial tree):
  - невпорядковане біноміальне дерево U_0 складається з єдиного вузла;
  - невпорядковане біноміальне дерево U_k складається з двох невпорядкованих біноміальних дерев U_{k-1}, причому корінь одного з них є *довільним* дочірнім вузлом іншого.
- Виконується лема про властивості біноміальних дерев з заміною пункту 4 на 4* (буде далі).
- <u>Особливість пірамід Фібоначчі</u>: об’єднання дерев у піраміді буде відбуватися лише під час операції EXTRACT_MIN.

Операції над пірамідами злиття

Лема

(властивості невпорядкованих біноміальних дерев)

Невпорядковане біноміальне дерево U_k

1. має 2^k вузлів;
2. має висоту *k*;
3. має $\binom{k}{i}$ вузлів на глибині *i* = 0,1…*k*;

4* має корінь степеня *k*, а степінь інших вузлів буде менша; при цьому синами кореня є корені піддерев U_0, U_1, …, U_{k-1} в деякому порядку.

- Для піраміди Фібоначчі з *n* вузлами максимальна степінь вузла $D(n) = \lfloor \lg n \rfloor$.

Операції над пірамідами злиття

• Створення піраміди Фібоначчі MAKE_FIB_HEAP

Повертає нову порожню піраміду Фібоначчі H з *n*[H]=0 та *min*[H]=NIL.

Маємо *t*(H)=0 та *m*(H)=0, тому потенціал Ф(H)=0.

Амортизована вартість процедури MAKE_FIB_HEAP дорівнює її фактичній вартості O(1).

• Вставка вузла FIB_HEAP_INSERT

Вузол *x* вже готовий і має заповнене поле *key*[*x*].

```
FIB_HEAP_INSERT(H, x)
  degree[x] ← 0
  p[x] ← NIL
  child[x] ← NIL
  left[x] ← x
  right[x] ← x
  mark[x] ← FALSE
  Присоединение списка корней, содержащего x, к списку корней H
  if min[H] = NIL или key[x] < key[min[H]]
     then min[H] ← x
  n[H] ← n[H] + 1
```

Операції над пірамідами злиття

- Вставка вузла FIB_HEAP_INSERT (далі)

Процедура не намагається об’єднати дерева в піраміді. Послідовне виконання FIB_HEAP_INSERT *k* разів призведе до додавання до списку коренів *k* дерев з одного вузла.

Додавання вузла відбувається за постійний час O(1).

Амортизована вартість процедури.

Нехай отримуємо з піраміди H піраміду H*.

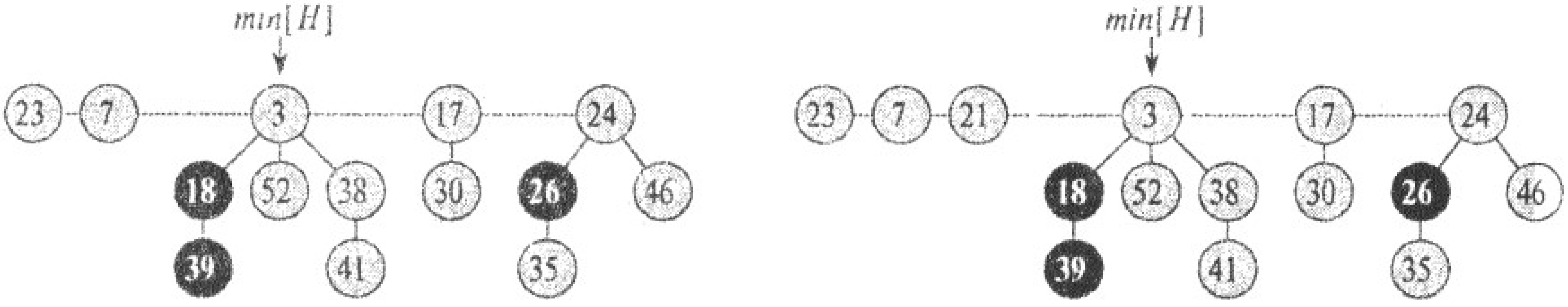
$t(H^*) = t(H) + 1, \quad m(H^*) = m(H).$

Збільшення потенціалу

$((t(H) + 1) + 2m(H)) – (t(H) + 2m(H)) = 1.$

Фактична вартість дорівнює O(1), тому амортизована вартість буде O(1) + 1 = O(1).

Операції над пірамідами злиття



а) б)

Вставка вузла з ключем 21 в піраміду Фібоначчі

Операції над пірамідами злиття

- <u>Пошук мінімального вузла</u>

На нього вказує *min*[H], тому пошук відбувається за час O(1). Потенціал H не змінюється, тому амортизована вартість рівна фактичній O(1).

- <u>Об’єднання двох пірамід FIB_HEAP_UNION</u>

Списки коренів пірамід H_1 та H_2 просто об’єднуються і шукається новий мінімальний вузол.

FIB_HEAP_UNION(H_1, H_2)
$H \leftarrow$ MAKE_FIB_HEAP()
$min[H] \leftarrow min[H_1]$
Добавление списка корней H_2 к списку корней H
if $(min[H_1] = \text{NIL})$ или $(min[H_2] \neq \text{NIL}$ и $key[min[H_2]] < key[min[H_1]])$
    then $min[H] \leftarrow min[H_2]$
$n[H] \leftarrow n[H_1] + n[H_2]$
Освобождение объектов H_1 и H_2
return H

Операції над пірамідами злиття

• <u>Об'єднання двох пірамід FIB_HEAP_UNION (далі)</u>

Знову ж таки, об'єднання дерев відсутнє.

<u>Амортизована вартість процедури.</u>

$$t(\mathrm{H}) = t(\mathrm{H}_1) + t(\mathrm{H}_2),$$

$$m(\mathrm{H}) = m(\mathrm{H}_1) + m(\mathrm{H}_2).$$

Зміна потенціалу дорівнює

$$\Phi(\mathrm{H}) – (\Phi(\mathrm{H}_1) + \Phi(\mathrm{H}_2)) =$$

$$= (t(\mathrm{H}) + 2m(\mathrm{H})) –$$

$$– ((t(\mathrm{H}_1) + 2m(\mathrm{H}_1)) + (t(\mathrm{H}_2) + 2m(\mathrm{H}_2))) =$$

$$= 0.$$

Отже, амортизована вартість рівна фактичній O(1).

Операції над пірамідами злиття

• <u>Видалення мінімального вузла</u>

```
FIB_HEAP_EXTRACT_MIN(H)
z ← min[H]
if z ≠ NIL
   then for (для) каждого дочернего по отношению к z узла x
            do Добавить x в список корней H
               p[x] ← NIL
        Удалить z из списка корней H
        if z = right[z]
           then min[H] ← NIL
           else  min[H] ← right[z]
                 CONSOLIDATE(H)
        n[H] ← n[H] − 1
return z
```

Спочатку всі дочірні вузли мінімального вузла переміщуються в список коренів піраміди, який потім *ущільнюється* процедурою CONSOLIDATE, щоб не було коренів однакової степені.

Операції над пірамідами злиття

- <u>Видалення мінімального вузла (далі)</u>

Ущільнення полягає у повторному виконанні наступних кроків, поки всі корені в списку не матимуть різні значення поля *degree*.

1. Знайти два корені *x* та *y* однакової степені, причому *key*[*x*] ≤ *key*[*y*].
2. Прив’язати (link) *y* до *x*: видалити *y* зі списку коренів та зробити сином *x*. Поле *degree*[*x*] при цьому збільшується, а мітка *mark*[*y*], якщо вона була, знімається.

FIB_HEAP_LINK(H, y, x)

Удалить y из списка корней H

Сделать y дочерним узлом x, увеличить $degree[x]$

$mark[y] \leftarrow$ FALSE

Операції над пірамідами злиття

• Видалення мінімального вузла (далі)

Процедура CONSOLIDATE використовує допоміжний масив A[0..D(*n*(H))]. Якщо A[*i*]=*y*, то *y* в даний момент є коренем степені *degree*[*y*]=*i*.

По завершенні циклу **for** в списку коренів залишиться не більше одного кореня кожної степені, і елементи масиву A вказуватимуть на ці корені.

Якщо перед викликом FIB_HEAP_EXTRACT_MIN всі дерева піраміди були невпорядкованими біноміальними деревами, то після виконання процедури вони такими і залишаться. Шляхи отримання нових дерев:

– дочірні дерева вузла, що видаляється (вони за умовою є невпорядкованими біноміальними деревами);

– дерево, отримане об’єднанням двох дерев однакової степені (якщо вони мають структуру U_k, то їх об’єднання процедурою FIB_HEAP_LINK дає дерево типу U_{k+1}).

Операції над пірамідами злиття

• Видалення мінімального вузла (далі)

```
CONSOLIDATE(H)
for i ← 0 to D(n[H])
    do A[i] ← NIL
for (для) каждого узла w в списке корней H
    do x ← w
       d ← degree[x]
       while A[d] ≠ NIL
           do y ← A[d]          ▷ Узел с той же степенью, что и у x.
              if key[x] > key[y]
                then обменять x ↔ y
              FIB_HEAP_LINK(H, y, x)
              A[d] ← NIL
              d ← d + 1
       A[d] ← x
min[H] ← NIL
for i ← 0 to D(n[H])
    do if A[i] ≠ NIL
         then Добавить A[i] в список корней H
              if min[H] = NIL или key[A[i]] < key[min[H]]
                then min[H] ← A[i]
```

Операції над пірамідами злиття

• <u>Видалення мінімального вузла (далі)</u>

Розглянемо цикл **while** (рядки 6-12).

```
while A[d] ≠ NIL
   do y ← A[d]            ▷ Узел с той же степенью, что и у x.
      if key[x] > key[y]
         then обменять x ↔ y
      FIB_HEAP_LINK(H, y, x)
      A[d] ← NIL
      d ← d + 1
```

Він зв'язує корінь *x* дерева, яке містить вузол *w*, з іншим деревом степені, що співпадає зі степінню *x*. Це робиться, допоки жоден інший корінь не матиме степінь, як у кореня *x*.

Інваріант циклу **while**:

На початку кожної ітерації *d* = *degree*[x].

Скориставшись ним, доведемо коректність алгоритму.

Операції над пірамідами злиття

• <u>Видалення мінімального вузла (далі)</u>

Ініціалізація. Рядок 5 гарантує виконання інваріанту.

Збереження. В кожній ітерації A[*d*] вказує на деякий корінь *y*. Оскільки *d* = *degree*[*x*] = *degree*[*y*], треба зв'язати *x* та *y*. Батьком стає вузол з меншим ключем (за необхідності відбувається обмін значень *x* та *y* в рядках 8-9).

Потім *y* прив'язується до *x* при виклику FIB_HEAP_LINK(H,*y*,*x*). Цей виклик збільшує *degree*[*x*].

З масиву A видаляється посилання на *y*.

В рядку 12 здійснюється відновлення інваріанту *d* = *degree*[*x*].

Завершення. Цикл виконується, поки не отримується A[*d*]=NIL, так що не буде коренів такої ж степені, як в *x*.

Операції над пірамідами злиття

• Видалення мінімального вузла (далі)

Амортизована вартість процедури

Нехай працюємо над пірамідою H з n вузлами.

Підрахуємо фактичну вартість.

Вклад O(D(n)) дає обробка максимум D(n) дочірніх вузлів мінімального вузла в FIB_HEAP_EXTRACT_MIN та рядки 1-2 і 14-19 в CONSOLIDATE.

Розмір списку при виклику CONSOLIDATE не перевищує (t(H)+D(n)–1). При кожному виконанні циклу **while** один з коренів зв'язується з іншим, тому загальний час роботи циклу **for** буде обмеженим згори (t(H)+D(n)).

Отже, загальний фактичний час дорівнює O(t(H)+D(n)).

Операції над пірамідами злиття

- Видалення мінімального вузла (далі)

Потенціал до вилучення мінімального вузла $t(\mathrm{H})+2m(\mathrm{H})$, а після – не перевищує $(\mathrm{D}(n)+1)+2m(\mathrm{H})$, бо залишається не більше $(\mathrm{D}(n)+1)$ коренів і нових міток не утворюється.

Верхня оцінка амортизованої вартості:

$$\mathrm{O}(t(\mathrm{H})+\mathrm{D}(n)) + ((\mathrm{D}(n)+1)+2m(\mathrm{H})) - (t(\mathrm{H})+2m(\mathrm{H})) =$$

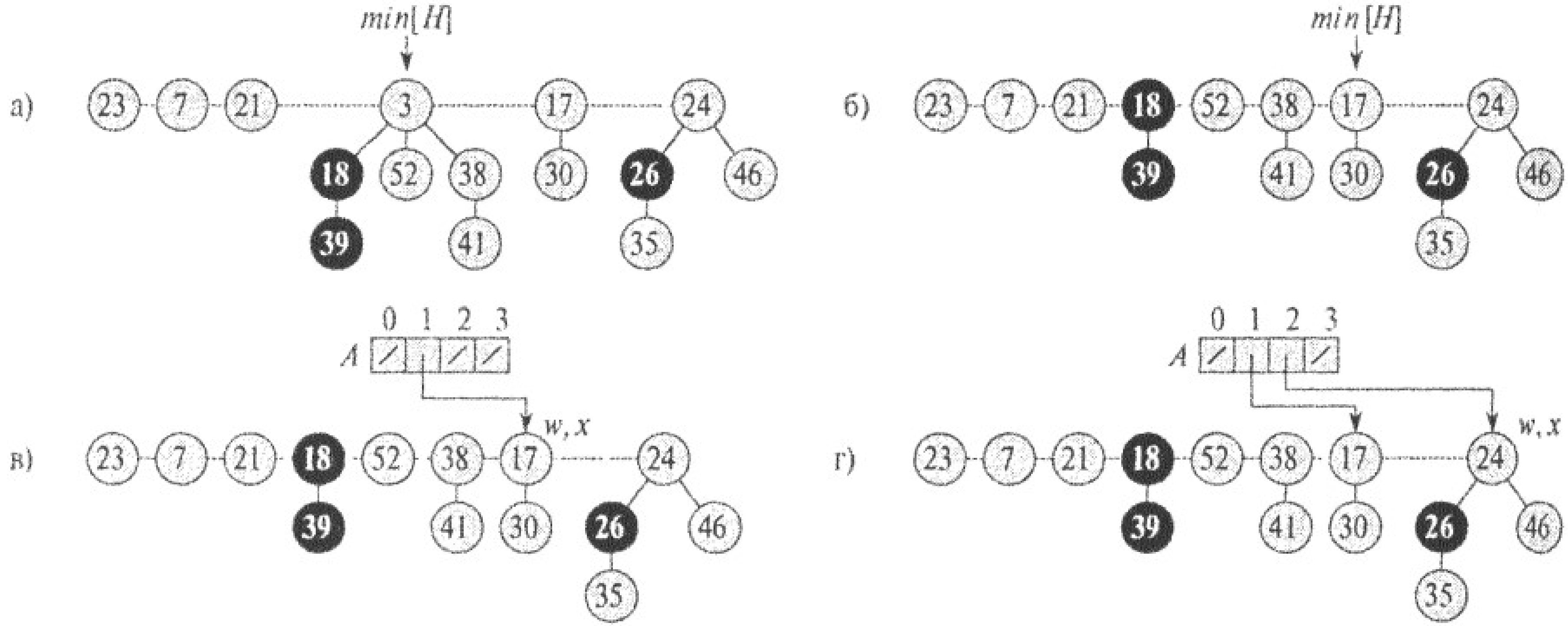
$$= \mathrm{O}(\mathrm{D}(n)) + \mathrm{O}(t(\mathrm{H})) - t(\mathrm{H}) = \mathrm{O}(\mathrm{D}(n)).$$

Остання рівність справедлива, оскільки можна масштабувати одиниці потенціалу так, щоб знехтувати константою, прихованою в $\mathrm{O}(t(\mathrm{H}))$.

Далі покажемо, що $\mathrm{D}(n) = \mathrm{O}(\lg n)$, тому амортизована вартість процедури складе $\mathrm{O}(\lg n)$.

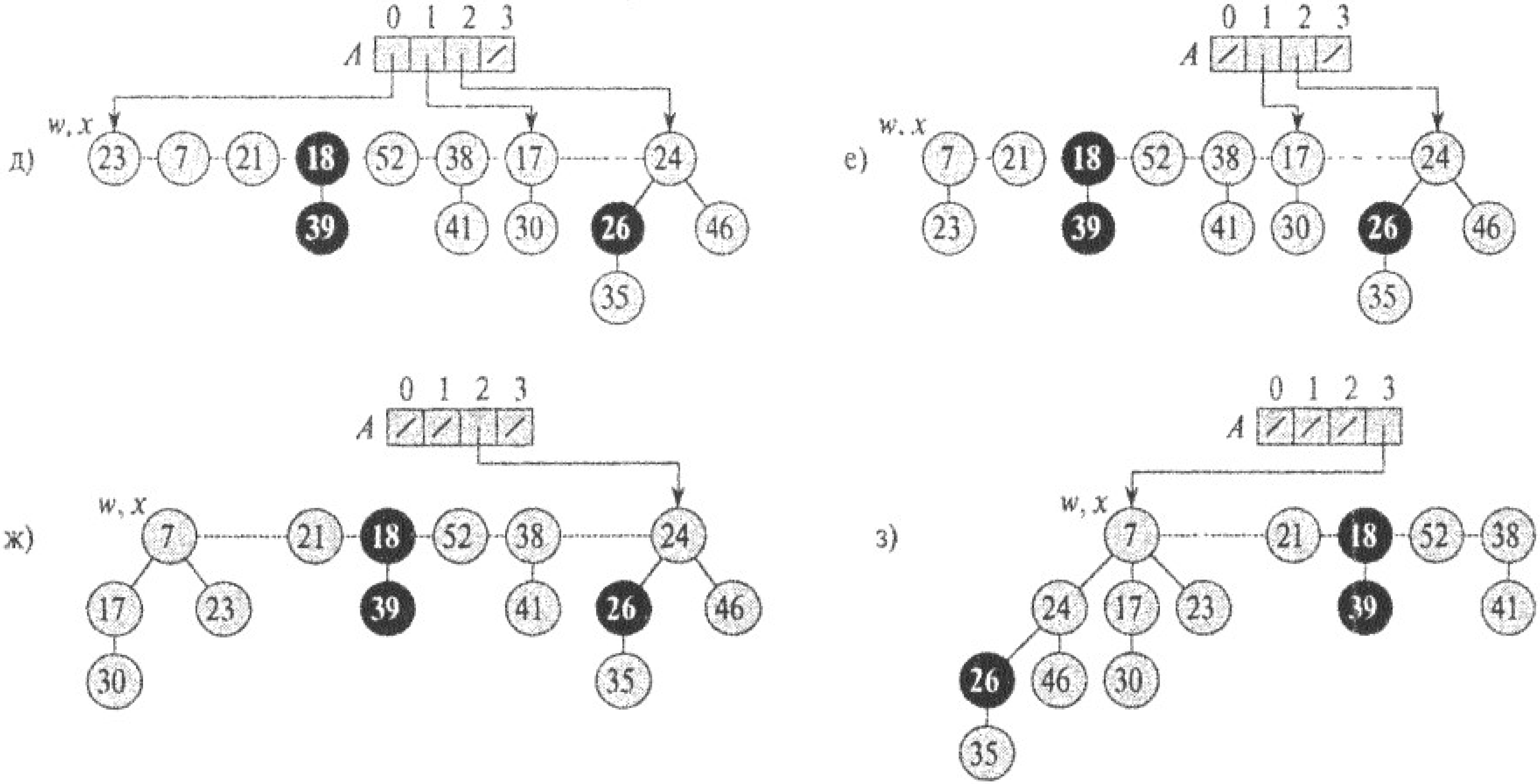
Операції над пірамідами злиття

Приклад видалення мінімального вузла



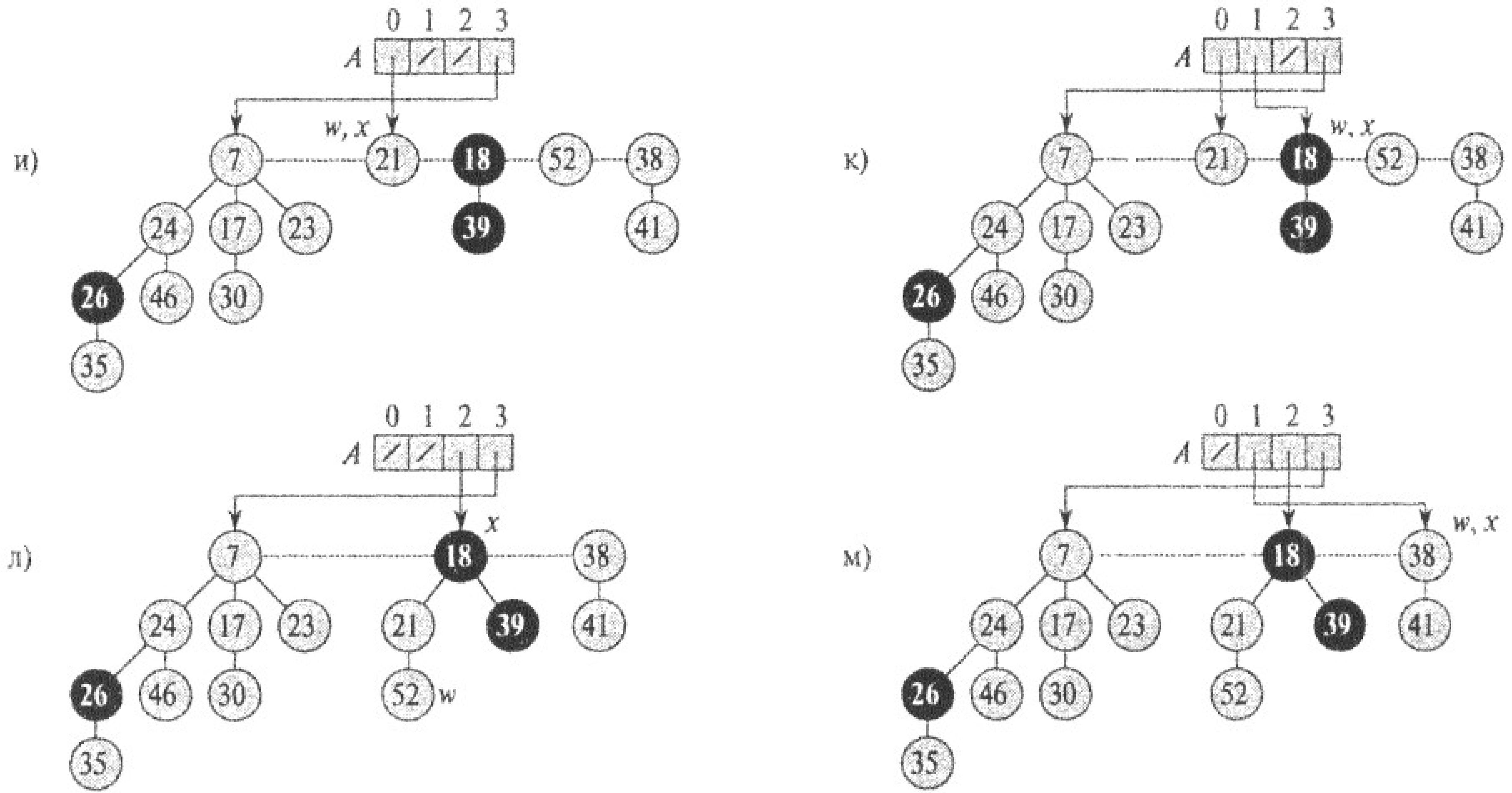
Операції над пірамідами злиття

Приклад видалення мінімального вузла (далі)



Операції над пірамідами злиття

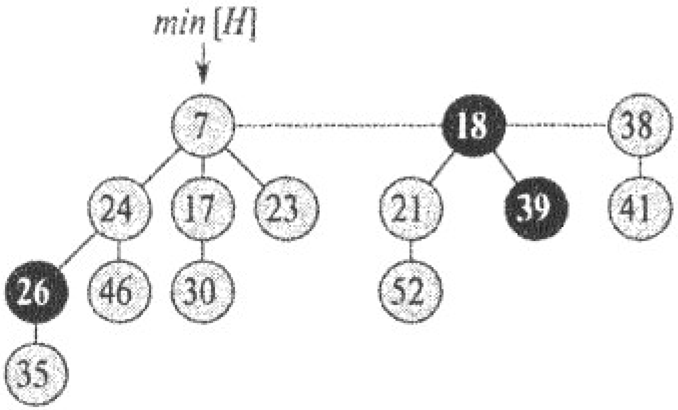
Приклад видалення мінімального вузла (продовження)



Операції над пірамідами злиття

Приклад видалення мінімального вузла
(завершальний етап)

н)



Додаткові операції над пірамідами злиття

• <u>Зменшення ключа</u>

Операція порушує властивість того, що піраміда Фібоначчі складається з невпорядкованих біноміальних дерев (втім, вони близькі до них).

Максимальна степінь D(*n*) матиме порядок O(lg *n*).

```
FIB_HEAP_DECREASE_KEY(H, x, k)
if k > key[x]
    then error "Новый ключ больше текущего"
key[x] ← k
y ← p[x]
if y ≠ NIL и key[x] < key[y]
    then CUT(H, x, y)
         CASCADING-CUT(H, y)
if key[x] < key[min[H]]
    then min[H] ← x
```

Додаткові операції над пірамідами злиття

• Зменшення ключа (далі)

Якщо властивість піраміди в дереві порушена, відбувається операція вирізання і, можливо, каскадного вирізання.

```
CUT(H, x, y)
Удаление x из списка дочерних узлов y, уменьшение degree[y]
Добавление x в список корней H
p[x] ← NIL
mark[x] ← FALSE
```

```
CASCADING_CUT(H, y)
z ← p[y]
if z ≠ NIL
    then if mark[y] = FALSE
            then mark[y] ← TRUE
            else CUT(H, y, z)
                 CASCADING_CUT(H, z)
```

Додаткові операції над пірамідами злиття

• Зменшення ключа (далі)

Вирізання означає, що утворюється нове дерево з коренем *x*.

Каскадне вирізання робить вершину новим коренем, якщо вона була помічена, і рекурсивно піднімається вище, або помічає її і зупиняється. Процес також зупиняється при досягненні кореня.

Тобто при каскадному видаленні вершина вирізатиметься, якщо вона перед цим втратила другого сина.

Додаткові операції над пірамідами злиття

• Зменшення ключа (далі)

Таким чином, поле мітки *mark*[*x*] буде змінюватися в таких випадках:

1. *x* став коренем (скидання мітки);

2. *x* прив'язали до іншого вузла (скидання мітки);

3. вирізали сина *x* (поява мітки).

В кінці процедури FIB_HEAP_DECREASE_KEY може відбутися оновлення мінімального вузла піраміди. Єдиний інший кандидат – перший вирізаний вузол.

Додаткові операції над пірамідами злиття

• Зменшення ключа (далі)

Амортизована вартість процедури

Фактична вартість.

FIB_HEAP_DECREASE_KEY вимагає O(1) часу плюс час на каскадне вирізання. Нехай CASCADING_CUT викликалося *c* разів, кожен з яких без врахування рекурсії вимагає час O(1). Тобто вартість процедури зменшення ключа складе O(*c*).

Додаткові операції над пірамідами злиття

• Зменшення ключа (далі)

Амортизована вартість процедури

Зміна потенціалу.

Кожен рекурсивний виклик CASCADING_CUT, крім останнього, здійснює вирізання та скидає мітку. Після цього отримуємо (t(H)+c) дерев (початкові t(H), (c–1) вирізане та одне з коренем x) та не більше (m(H)–c+2) помічених вузла (каскадом зняли мітку в (c–1) вузла та один можливо помітили в кінці).

Додаткові операції над пірамідами злиття

• <u>Зменшення ключа (далі)</u>

<u>Амортизована вартість процедури</u>

Зміна потенціалу складе

$$((t(\mathrm{H})+c)+2(m(\mathrm{H})-c+2)) - (t(\mathrm{H})+2m(\mathrm{H})) = 4-c.$$

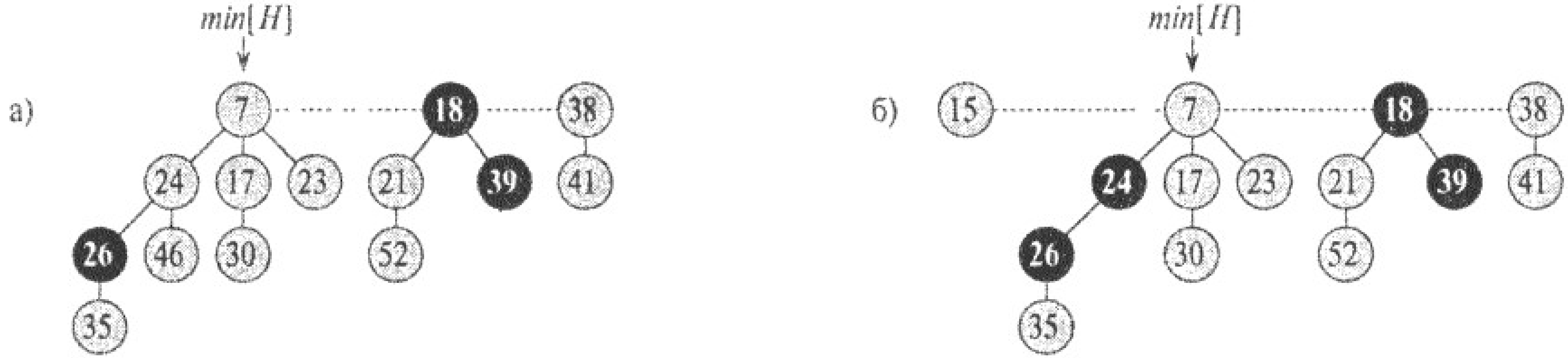
Амортизована вартість не перевищить O(c) + 4 – c = O(1).

(Можна відповідно масштабувати одиниці потенціалу для домінування над константою в O(c).)

Коли мічений вузол видаляється в процесі каскадного вирізання, скидання мітки призводить до зменшення потенціалу на 2: одна одиниця оплачує вирізання і скидання мітки, друга компенсує збільшення потенціалу через появу нового кореня. Тому функція потенціалу містить як член подвоєну кількість помічених вузлів.

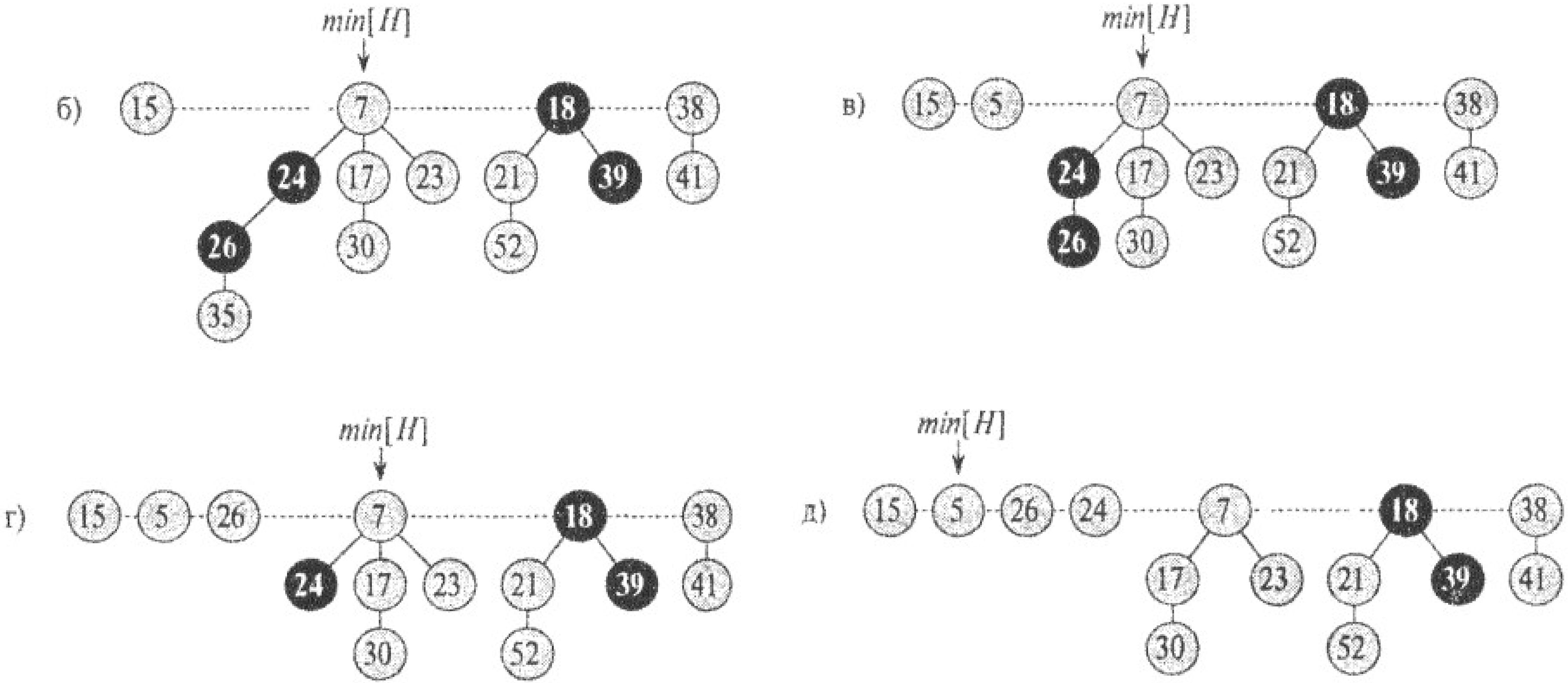
Додаткові операції над пірамідами злиття

Приклад зменшення ключа 46 до 15



Додаткові операції над пірамідами злиття

Приклад зменшення ключа 35 до 5



Додаткові операції над пірамідами злиття

• Видалення вузла

```
FIB_HEAP_DELETE(H, x)
1  FIB_HEAP_DECREASE_KEY(H, x, −∞)
2  FIB_HEAP_EXTRACT_MIN(H)
```

Вважається, що піраміда не містить ключів зі значенням $-\infty$.

Процедура FIB_HEAP_DELETE цілком аналогічна BINOMIAL_HEAP_DELETE: робить *x* мінімальним значенням піраміди і вилучає його.

Амортизований час – сума амортизованого часу процедури зменшення ключа O(1) та амортизованого часу роботи процедури видалення мінімуму O(D(*n*)), тобто він складе O(lg *n*).

Оцінка максимальної степені вузла

Покажемо, що максимальна степінь D(n) матиме порядок O(lg n) за умови вирізання вузла, як тільки він втратив двох синів.

Для кожного вузла x визначимо *size*(x) – кількість вузлів у піддереві з коренем x, включно з ним.

Лема 1. Нехай x – довільний вузол піраміди Фібоначчі та $y_1,\ldots,y_k$ – дочірні вузли x в порядку їх зв'язування з x від ранніх до пізніх. Тоді $degree[y_1] \geq 0$ та $degree[y_i] \geq i - 2$ при $i=2,3,\ldots,k$.

Доведення. Очевидно $degree[y_1] \geq 0$.

Для $i \geq 2$ при зв'язуванні y_i з x всі вузли y_1, $y_2,\ldots$, y_{i-1} вже є дочірніми для x, тому $degree[x] \geq i - 1$. Зв'язування y_i з x можливе лише за умови $degree[x] = degree[y_i]$, тому на момент зв'язування $degree[y_i] \geq i - 1$. Після цього y_i міг втратити не більше одного сина (інакше його вже би вирізали). Тому $degree[y_i] \geq i - 2$.

Оцінка максимальної степені вузла

k-те число Фібоначчі: $F_k = \begin{cases} 0 & \text{при } k = 0, \\ 1 & \text{при } k = 1, \\ F_{k-1} + F_{k-2} & \text{при } k \geq 2. \end{cases}$

Лема 2. Для всіх цілих *k* ≥ 0: $F_{k+2} = 1 + \sum_{i=0}^{k} F_i$.

Доведення. За матіндукцією по *k*.

При *k* = 0:

$$1 + \sum_{i=0}^{0} F_i = 1 + F_0 = 1 + 0 = 1 = F_2.$$

Припустимо $F_{k+1} = 1 + \sum_{i=0}^{k-1} F_i$. Тоді

$$F_{k+2} = F_k + F_{k+1} = F_k + \left(1 + \sum_{i=0}^{k-1} F_i\right) = 1 + \sum_{i=0}^{k} F_i.$$

Оцінка максимальної степені вузла

Лема 3. Нехай x – довільний вузол піраміди Фібоначчі, а $k = degree[x]$ – його степінь.

Тоді $size(x) \geq F_{k+2} \geq \varphi^k$, де $\varphi = (1+\sqrt{5})/2$.

Наслідок. Максимальна степінь D(n) довільного вузла в піраміді Фібоначчі з n вузлами дорівнює O(lg n).

Доведення. Нехай x – довільний вузол в піраміді Фібоначчі з n вузлами і нехай $k = degree$[x]. За лемою 3, $n \geq size(x) \geq \varphi^k$. Прологарифмуємо за основою φ:

$$k \leq \log_\varphi n \, .$$

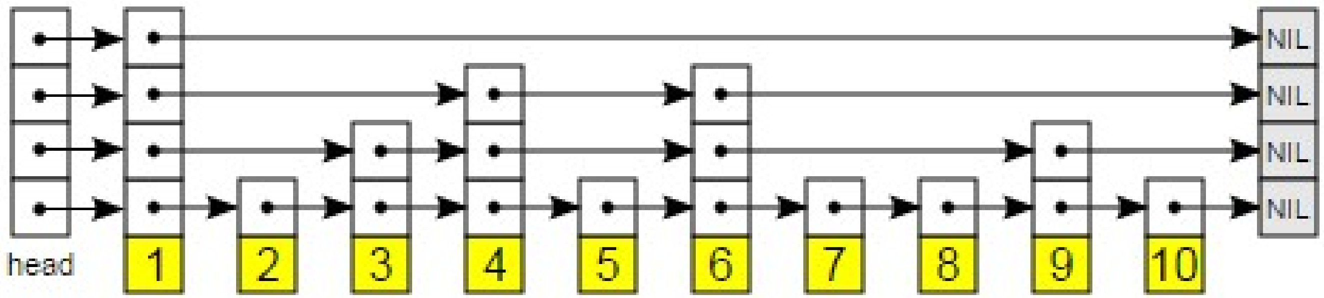
Тому максимальна степінь D(n) довільного вузла дорівнює O(lg n).

Зауваження. В дійсності k є цілим числом, тому справедливою буде оцінка

$$k \leq \left\lfloor \log_\phi n \right\rfloor .$$

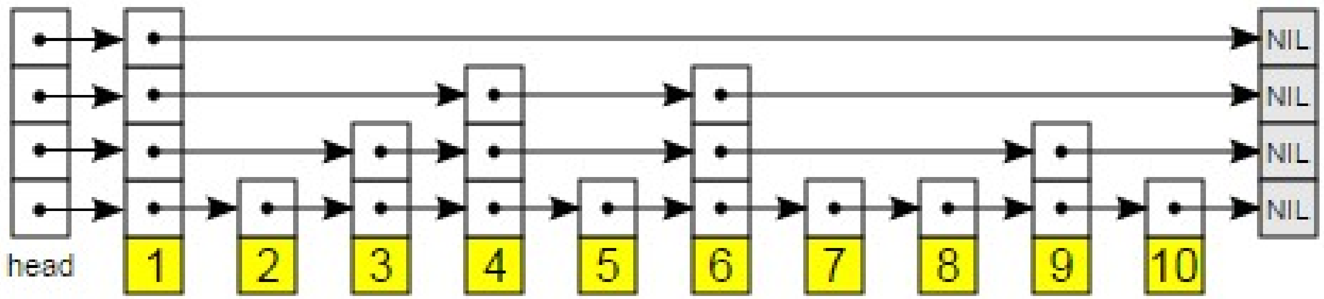
Список з пропусками (Skip List)

- Ймовірнісна структура даних, в основі якої декілька паралельних відсортованих зв'язаних списки (з пропущеними вузлами).
- Основна ідея – реалізація бінарного пошуку для зв’язаних списків.
- Вставка, пошук і видалення виконуються за логарифмічний випадковий час.
- Елементи, що використовуються в списку з пропусками, можуть мати більше одного вказівника і відповідно належати до більш ніж одного списку.



Список з пропусками (Skip List)

- На нижньому рівні – звичайний впорядкований зв'язний список, що містить всі вузли (з імовірністю 1).
- Вузол з кожного рівня *i* присутній в шарі (*i*+1) з фіксованою ймовірністю *p* (найчастіше 1/2 чи 1/4).
- В середньому кожен елемент зустрічається в 1/(1–*p*) списках, а верхній елемент (зазвичай особливий елемент на початку списку з пропусками) – в $\log_{1/p} n$ списках.



Список з пропусками (Skip List)

- *Пошук* починається з головного елемента верхнього списку і продовжується горизонтально, поки поточний елемент $\leq$ цільового.
- Якщо елемент знайшли, пошук завершується; якщо поточний елемент більший ніж цільовий або пошук досяг кінця списку в шарі, процедуру повторюють після повернення до попереднього елемента і спуску на один рівень нижче.
- Очікуване число кроків при пошуку елемента становить $1/p$.
- Вибір різних значень для p дає можливість досягти необхідного балансу між швидкістю пошуку і затратами пам’яті на зберігання списку.

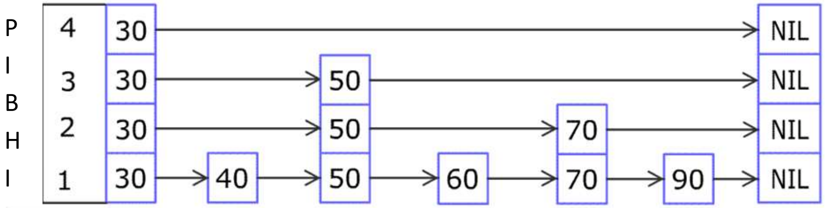
Список з пропусками (Skip List)

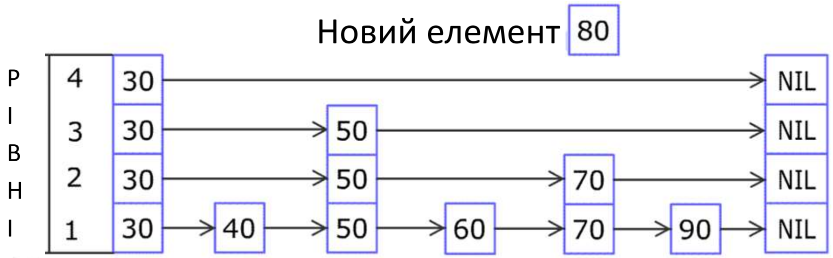
- *Вставка* нового елемента робиться так.

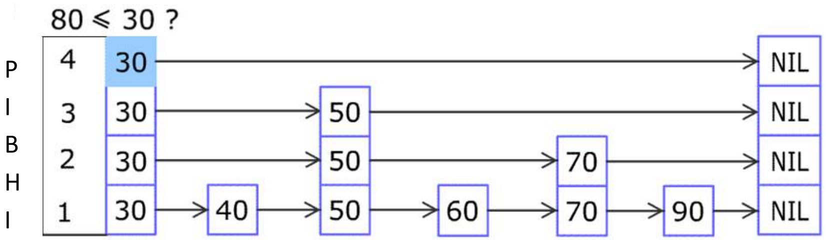
1. За допомогою алгоритму пошуку знаходиться позиція вставки елемента в нижньому списку.
2. Елемент вставляється.
3. «Підкидається монетка» і в залежності від результату елемент проштовхується на рівень вище.
4. Попередній крок повторюється, поки «підкидання монетки» дає позитивний результат.

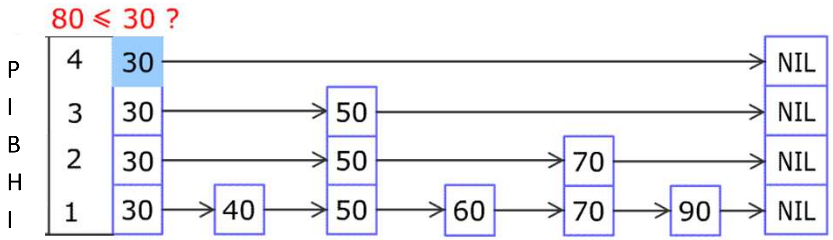
- *Видалення* елемента відбувається елементарно: елемент знаходиться і видаляється з усіх рівнів.

Список з пропусками (Skip List)

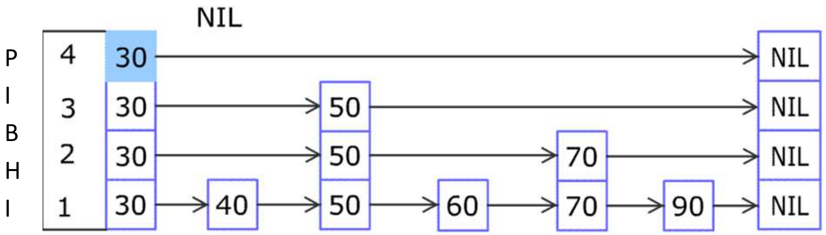


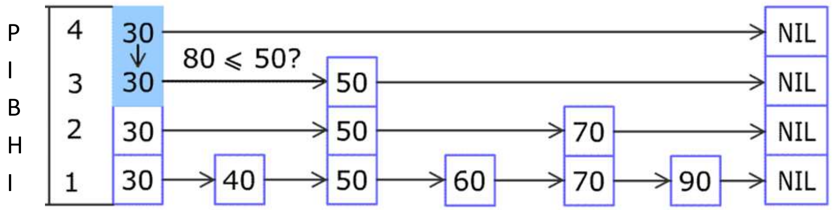


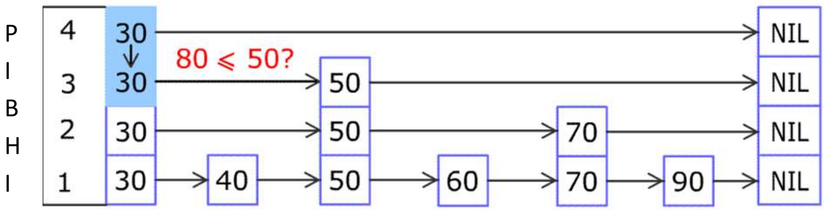


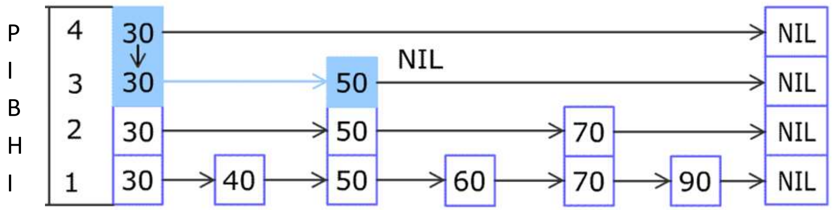


Список з пропусками (Skip List)

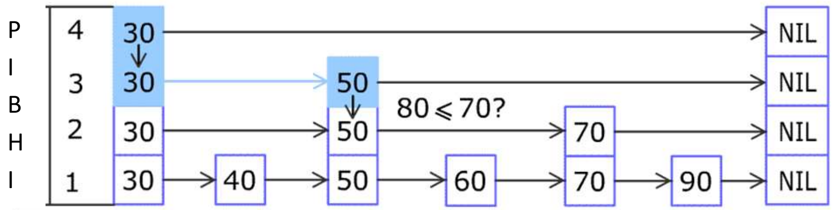


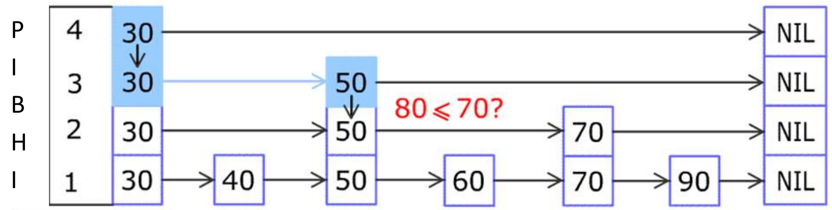


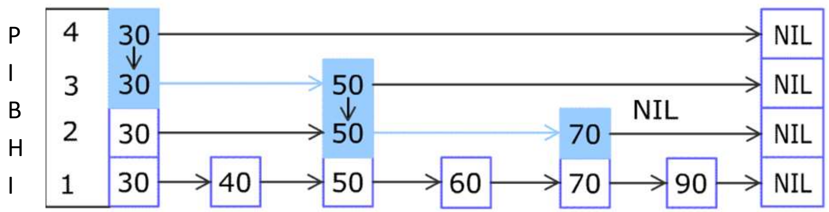


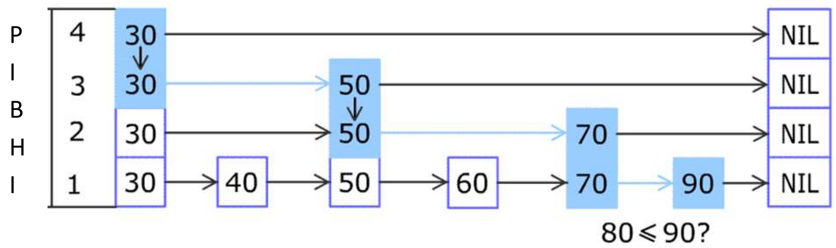


Список з пропусками (Skip List)

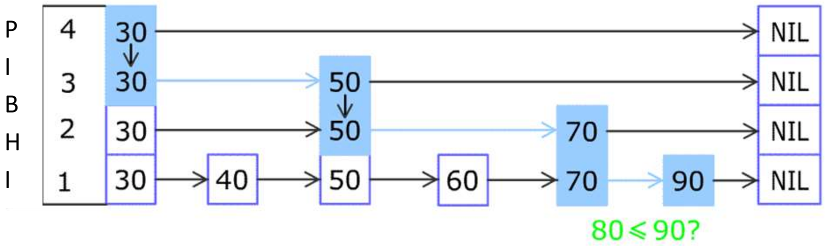


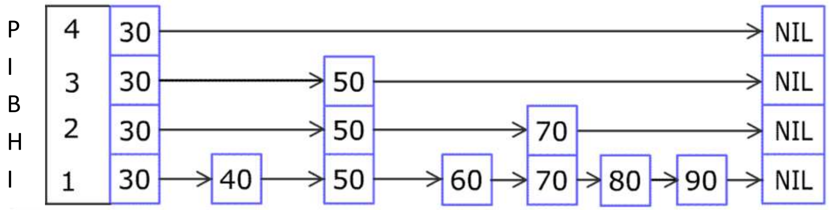




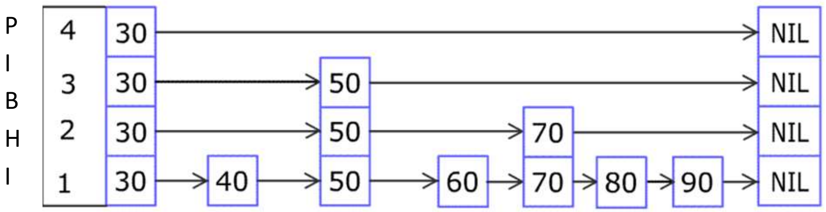


Список з пропусками (Skip List)

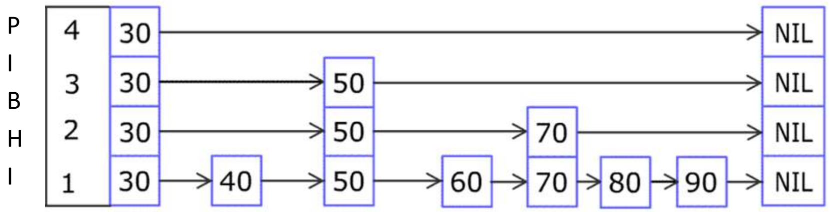




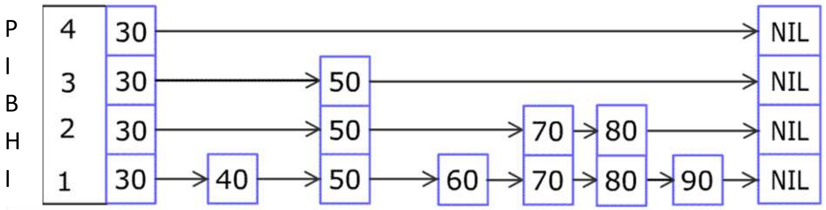
Підкидаємо монетку



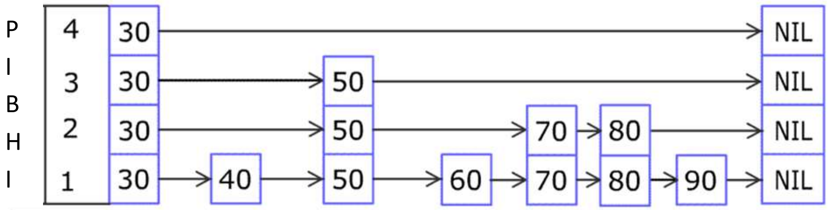
Орел $\Rightarrow$ вставка на другий рівень



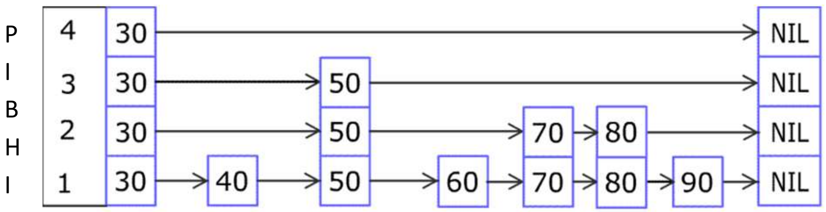
Список з пропусками (Skip List)



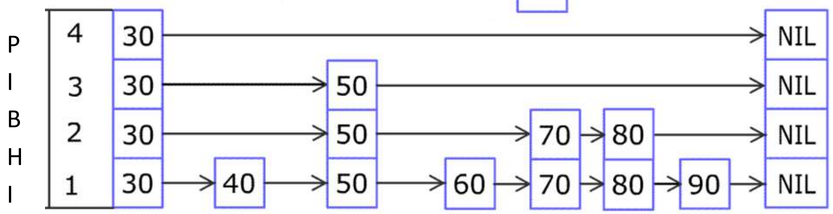
Підкидаємо монетку



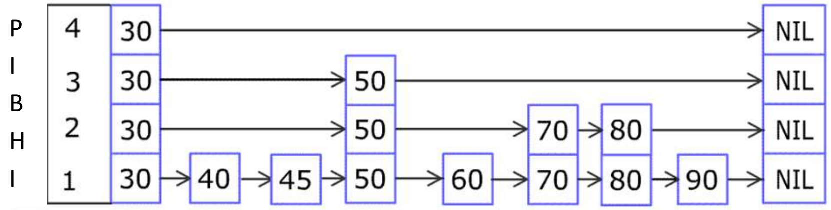
Решка $\Rightarrow$ вставка елемента завершена



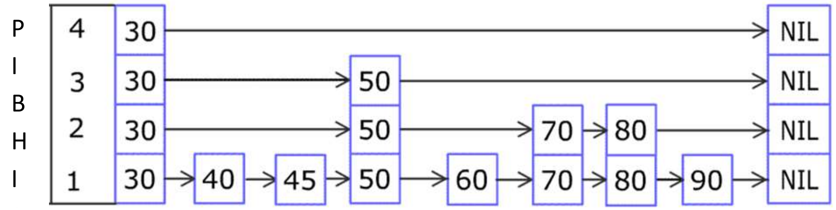
Новий елемент 45



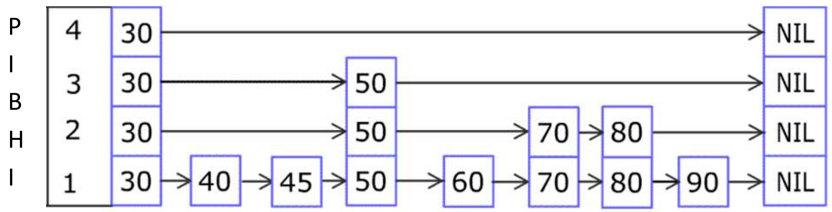
Список з пропусками (Skip List)



Підкидаємо монетку



Решка $\Rightarrow$ вставка елемента завершена



Запитання і завдання

- Доведіть лему про властивості невпорядкованих біноміальних дерев.
- Покажіть, що за підтримки лише операцій MAKE_HEAP, INSERT, MINIMUM, EXTRACT_MIN та UNION максимальна степінь D(*n*) піраміди Фібоначчі з *n* вузлами не перевищує $\lfloor \lg n \rfloor$
- Нехай корінь *x* піраміди Фібоначчі помічений. Як вузол *x* міг стати поміченим коренем? Покажіть, що той факт, що він помічений, не має значення для аналізу, навіть якщо це не корінь, який спочатку був прив'язаний до іншого вузла, а потім втратив одного сина.

Запитання і завдання

- Припустимо, введено узагальнення каскадного вирізання: вузол *x* вирізається з батьківського, як тільки втрачає *k*-го сина, де *k* – деяка константа (в розглянутому нами випадку *k* = 2). Для яких значень *k* буде справедливе співвідношення D(*n*) = O(lg *n*)?
- Введіть нову операцію над пірамідами Фібоначчі FIB_HEAP_CHANGE_KEY(H,*x*,*k*), яка змінює ключ вузла *x*, присвоюючи йому значення *k*. Наведіть її ефективну реалізацію і проаналізуйте амортизований час роботи для випадків, коли *k* більше, менше чи рівне *key*[*x*]. Амортизований час роботи інших операцій не має змінитися.
- Розробіть ефективну реалізацію процедури FIB_HEAP_PRUNE(H,*r*), що видаляє min(*r*, *n*[H]) вузлів з H. Які саме вузли видаляються, не важливо. Проаналізуйте амортизований час її роботи. (При цьому може знадобитися зміна структури даних і функції потенціалу.) Амортизований час роботи інших операцій не має змінитися.