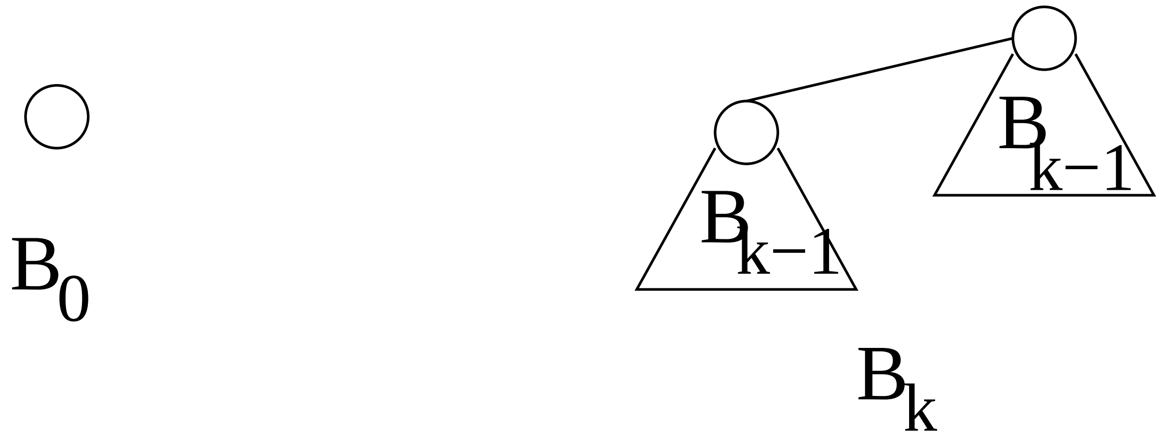
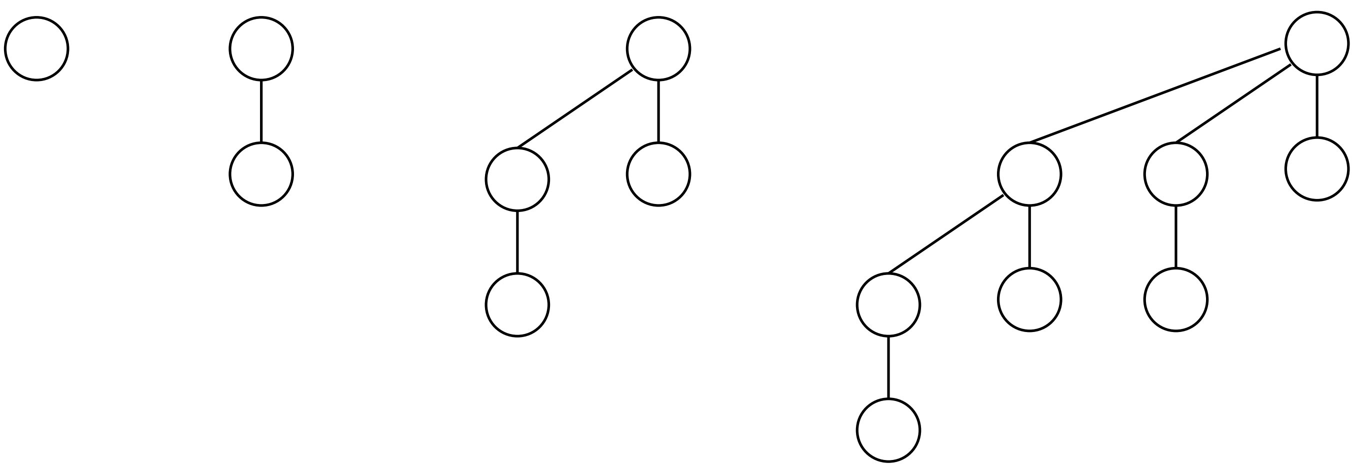
Kopce złączalne

Operacje:

- `Make-Heap()` tworzy nowy pusty kopiec
- `Insert(`H`,`x`)` wstawia x o kluczu *key*$[x]$ do kopca H
- `Minimum(`H`)` zwraca wsk. do węzła w H o min. kluczu
- `Extract-Min(`H`)` zwraca węzeł o minimalnym kluczu i usuwa go z H
- `Union(`H_1`,`H_2`)` tworzy nowy kopiec z wszystkich węzłów H_1 i H_2 (H_1 i H_2 – niszczone)
- `Decrease-Key(`H`,`x`,`k`)` nadaje nową (niewiększą) wartość kluczowi *key*$[x]$
- `Delete(`H`,`x`)` usuwa węzeł x z H

Drzewa dwumianowe





Własności drzew dwumianowych

Lemat. W drzewie dwumianowym B_k:

1. jest 2^k węzłów
2. wysokość wynosi k
3. jest dokładnie $\binom{k}{i}$ węzłów na głębokości i, dla $i = 0, \ldots, k$.
4. Stopień korzenia $= k$ (największy), a i-ty syn korzenia od lewej strony jest korzeniem drzewa B_{k-i}, dla $i = 1, \ldots, k$.

Wniosek. Maksymalny stopień w n-węzłowym drzewie dwumianowym wynosi $\lg n$.

Dowód Lematu

D-d. Indukcja po k. Dla $k = 0$ – oczywiste.
Niech $k > 0$.

1. B_k dwa połączone drzewa B_{k-1}: $2^{k-1} + 2^{k-1} = 2^k$
2. Maksymalna głębokość węzła: $1+$(maks. gł. węzła w niżej podłączonym B_{k-1}) $= (k-1) + 1 = k$
3. Niech $D(k, i)$ – liczba węzłów na gł. i w B_k.
   $D(k, i) = D(k-1, i-1) + D(k-1, i) = \binom{k-1}{i} + \binom{k-1}{i-1} = \binom{k}{i}$
4. W B_k korzeń to jedyny węzeł o większym stopniu niż węzły w B_{k-1}. Stopień korzenia w B_k wynosi $1+$ stopień korzenia w B_{k-1}.
   Poddrzewo B_{k-1} jest dołączane jako nowy skrajny lewy syn korzenia, a pozostali synowie (z zał. ind.) to $B_{(k-1)-1=k-2}, \ldots, B_0$

□

Kopiec dwumianowy

Kopiec dwumianowy H – zbiór drzew dwumianowych, taki że:

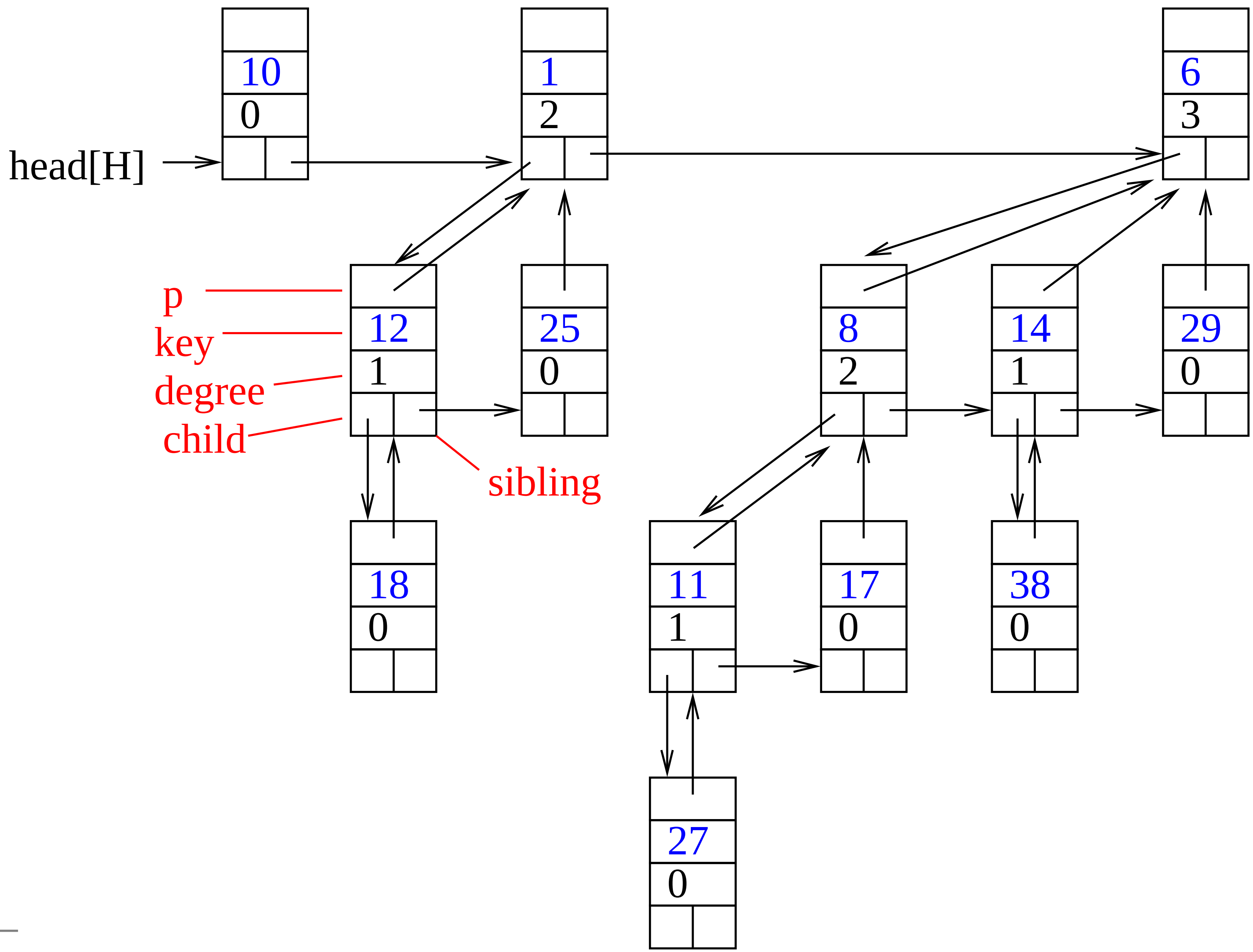
1. Każde drzewo – *uporządkowane kopcowo* (t.j. klucz w węźle niemniejszy niż klucz ojca)
2. Dla każdego $d \geq 0$ istnieje w H co najwyżej jedno drzewo o korzeniu stopnia d (B_d)

Kopiec dwumianowy

Reprezentacja: *head*$[H]$ wskaźnik na listę (korzeni) drzew (uporządkowaną ściśle rosnąco wg. stopnia).
Pola węzła x:

- *key*$[x]$ – klucz
- *p*$[x]$ – ojciec x (jeśli x – korzeń, to *p*$[x] = NIL$)
- *child*$[x]$ – skrajnie lewy syn x (jeśli x – liść, to *child*$[x] = NIL$)
- *degree*$[x]$ – liczba synów x (jeśli x – liść, to *degree*$[x] = 0$)
- *sibling*$[x]$ – brat x (na prawo od x)
  (jeśli x – skrajnie prawy brat, to *sibling*$[x] = NIL$)
  (jeśli x – korzeń, to *sibling*$[x]$ – następnik na liście korzeni)

Kopiec dwumianowy



operacje

`Make-Binomial-Heap` tworzy obiekt H, taki że *head*$[H] = NIL$. Czas: $\Theta(1)$.

`Binomial-Heap-Minimum(`H`)`

```
y ← NIL
x ← head[H]
min ← ∞
while x ≠ NIL do
    if key[x] < min then
        min ← key[x]
        y ← x
    x ← sibling[x]
return y
```

Czas: $O(\lg n)$. (Długość listy korzeni $\leq \lfloor \lg n \rfloor + 1$, bo maksymalny stopień korzenia $\leq \lfloor \lg n \rfloor$)

Binomial-Link

Podłączenie drzewa B_{k-1} o korzeniu y do drzewa B_{k-1} o korzeniu z. (Drzewo o korzeniu z staje się drzewem B_k, a y staje się nowym skrajnie lewym synem węzła z.)

```
Binomial-Heap-Link(y, z)
p[y] ← z
sibling[y] ← child[z]
child[z] ← y
degree[z] ← degree[z] + 1
```

Czas: $\Theta(1)$

Binomial-Heap-Union

`Binomial-Heap-Merge(`H_1`,`H_2`)` – scala listy korzeni H_1 i H_2 w listę posortowaną niemalejąco wg stopnia w czasie $O(m)$, gdzie m liczba korzeni na obu listach. (Implementacja – ćwiczenie.)

```
Binomial-Heap-Union(H1,H2)
H ← Make-Binomial-Heap()
head[H] ← Binomial-Heap-Merge(H1,H2)
zwolnij obiekty H1 i H2 (ale nie węzły)
if head[H] = NIL then
    return H
prev-x ← NIL
x ← head[H]
next-x ← sibling[x]
while next-x ≠ NIL do ▷ początek while
...
```

Binomial-Heap-Union

```
    if (degree[x] ≠ degree[next-x]) or
       (sibling[next-x] ≠ NIL and
        degree[sibling[next-x]] = degree[x]) then
      prev-x ← x  ▷ 1 i 2
      x ← next-x  ▷ 1 i 2
    else if key[x] ≤ key[next-x] then
        sibling[x] ← sibling[next-x]  ▷ 3
        Binomial-Link(next-x, x)  ▷ 3
      else if prev-x = NIL
          then head[H] ← next-x  ▷ 4
          else sibling[prev-x] ← next-x  ▷ 4
        Binomial-Link(x, next-x)  ▷ 4
        x ← next-x  ▷ 4
    next-x ← sibling[x]  ▷ koniec while
return H
```

Binomial-Heap-Union – analiza

Przypadki:

1. *degree*$[x] \neq$*degree*[*next-x*]
2. Stopień x i dwóch kolejnych – jednakowe. (Może wystąpić gdy scaliliśmy dwa drzewa B_{k-1} w jedno B_k a dwa następne są typu B_k.)
3. Stopień x i *next-x* – jednakowe, a następnego po *next-x* nie ma lub ma większy stopień, oraz *key*$[x] \leq$*key*[*next-x*]. Podczepiamy *next-x* pod x.
4. Stopień x i *next-x* – jednakowe, a następnego po *next-x* nie ma lub ma większy stopień, oraz *key*$[x] >$*key*[*next-x*]. Podczepiamy x pod *next-x*.

Po każdym przypadku x jest bliżej końca listy *head*$[H]$.
Czas: $O(\lg n)$, gdzie n – łączna liczba węzłów w H_1 iH_2.

Binomial-Heap-Insert

`Binomial-Heap-Insert(`H`,`x`)`

$H' \leftarrow$ `Make-Binomial-Heap()`

p$[x] \leftarrow NIL$

child$[x] \leftarrow NIL$

sibling$[x] \leftarrow NIL$

degree$[x] \leftarrow 0$

head$[H'] \leftarrow x$

$H \leftarrow$ `Binomial-Heap-Union(`H`,`H'`)`

Czas: $O(\lg n)$.

Binomial-Heap-Extract-Min

`Binomial-Heap-Extract-Min(`H`)`

1 *znajdź w H korzeń x z minimalnym kluczem i usuń x z listy korzeni H*

2 $H' \leftarrow$ `Make-Binomial-Heap()`

3 *odwróć kolejność na liście synów x – child$[x]$ i podłącz ją do head$[H']$*

4 $H \leftarrow$ `Binomial-Heap-Union(`H`,`H'`)`

5 `return` x

Czas: $O(\lg n)$

Binomial-Heap-Decrease-Key

```
Binomial-Heap-Decrease-Key(H, x, k)
if k > key[x] then
    error “nowy klucz większy od bieżącego”
key[x] ← k
y ← x
z ← p[y]
while z ≠ NIL and key[y] < key[z] do
    zamień key[y] ↔ key[z]
    ▷ jeśli są inne pola w y i z
    ▷ to również zamień ich wartości
    y ← z
    z ← p[y]
```

Czas: $O(\lg n)$. (Głębokość węzła $x \leq \lfloor \lg n \rfloor$.)

Binomial-Heap-Delete

```
Binomial-Heap-Delete(H,x)
1 Binomial-Heap-Decrease(H,x,−∞)
2 Binomial-Heap-Extract-Min(H)
```

Czas: $O(\lg n)$.

Kopce Fibonacciego

Operacje nie związane z usuwaniem: w zamort. cz. $O(1)$.
Atrybuty węzła x:

- *key*$[x]$ – klucz
- *p*$[x]$ – ojciec x
- *child*$[x]$ – jedno z dzieci x
- *left*$[x]$, *right*$[x]$ – lewy i prawy brat (wskaźniki *left* i *right* łączą rodzeństwo w listę cykliczną.)
- *degree*$[x]$ – liczba synów x
- *mark*$[x]$ – czy węzeł stracił syna od ostatniego razu kiedy sam został synem innego węzła

Atrybut kopca H: *min*$[H]$ – wsk. do korzenia o min. kluczu, *n*$[H]$ – liczba węzłów H.

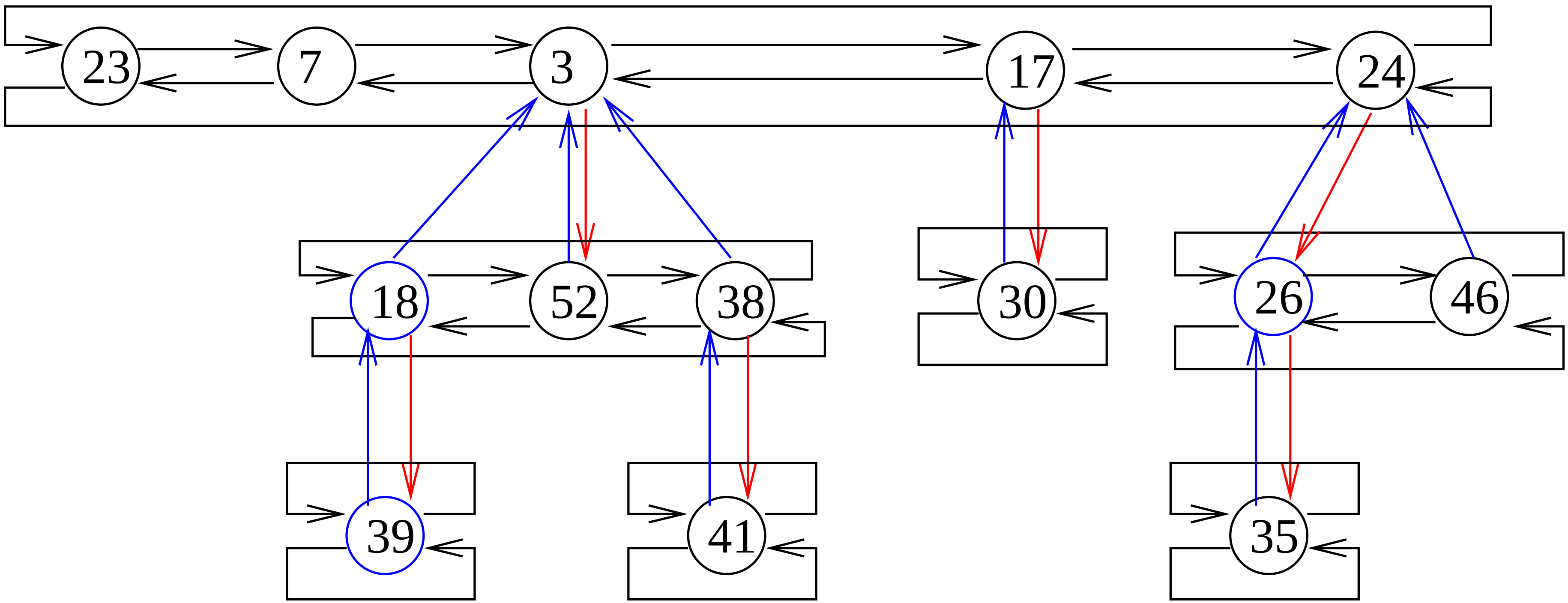
funkcja potencjału

$t(H)$ – liczba korzeni w kopcu H
$m(H)$ – liczba *zaznaczonych* węzłów w H (t.j. takich, w których *marked*$= TRUE$)
Potencjał kopca H:

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

Dla pustego kopca H: $\Phi(H) = 0$ i oczywiście dla dowolnego H: $\Phi(H) \geq 0$

Kopiec Fibonacciego



niebieskie węzły – *marked*$= TRUE$

Potencjał: $\Phi(H) = t(H) + 2m(H) = 5 + 2 \cdot 3 = 11$.

Własnośći

nieuporządkowane drzewo dwumianowe U_k:

- U_0 – jeden węzeł
- U_{k+1} – U_k po podłączeniu korzenia innego U_k jako dowolnego nowego syna swojego korzenia.

Jeśli stosujemy tylko (dalej zdefiniowane) `Make-Heap`, `Insert`, `Minimum` i `Extract-Min` to *min*$[H]$ wskazuje na listę korzeni nieuporządkowanych drzew dwumianowych. Stąd $D(n)$ (maksymalny stopień węzła w kopcu n-elementowym) jest $\lg n$.

operacje

`Make-Fib-Heap()` – tworzy obiekt H z atrybutami *min*$[H] = NIL$ oraz *n*$[H] = 0$. Czas: $O(1)$.

`Fib-Heap-Insert(`H`,`x`)`

degree$[x] \leftarrow 0$
p$[x] \leftarrow NIL$
child$[x] \leftarrow NIL$
left$[x] \leftarrow x$
right$[x] \leftarrow x$
mark$[x] \leftarrow FALSE$
połącz x *z listą korzeni* H
`if` *min*$[H] = NIL$ `or` *key*$[x] <$*key*`[`*min*$[H]$`]`
   `then` *min*$[H] \leftarrow x$
n$[H] \leftarrow$*n*$[H] + 1$

Przyrost potencjału: $t(H') + 2m(H') - t(H) - 2m(H) = 1$.
Czas (koszt faktyczny): $O(1)$. Koszt zamort.: $O(1) + 1$

operacje

`Fib-Heap-Minimum(`H`)` – zwraca $\mathit{min}[H]$. Czas: $O(1)$.

`Fib-Heap-Union(`H_1`,`H_2`)`

$H \leftarrow$ `Make-Fib-Heap()`
$\mathit{min}[H] \leftarrow \mathit{min}[H_1]$
sklej listę korzeni H_2 *z listą korzeni* H_1
if $(\mathit{min}[H_1] = NIL)$ **or**
    $(\mathit{min}[H_2] \neq NIL$ **and** $\mathit{key}[\mathit{min}[H_2]] < \mathit{key}[\mathit{min}[H_1]])$
  then $\mathit{min}[H] \leftarrow \mathit{min}[H_2]$
$n[H] \leftarrow n[H_1] + n[H_2]$
zwolnij obiekty H_1 *i* H_2
return H

Przyrost Φ: $\Phi(H) - (\Phi(H_1) + \Phi(H_2)) = 0$.

Czas: $O(1)$. Koszt zamort.: $O(1) + 0$.

Fib-Extract-Min

Fib-Extract-Min(H)

$z \leftarrow$ *min*$[H]$

if $z \neq NIL$ then

  for *każdy syn* x *węzła* z do

    dodaj x *do listy korzeni* H

    p$[x] \leftarrow NIL$

  usuń z *z listy korzeni* H *(nie zmieniając right*$[z]$*)*

  if $z =$ *right*$[z]$ then

    ▷ z *był jedynym korzeniem (i nie miał synów)*

    min$[H] \leftarrow NIL$

  else

    min$[H] \leftarrow$ *right*$[z]$

    Cosolidate(H) ▷ *następny slajd*

  n$[H] \leftarrow$ *n*$[H] - 1$

return z

Consolidate

Consolidate(H)

for $i \leftarrow 0$ to $D(n[H])$ do

    $A[i] \leftarrow NIL$ ▷ $A[0..D]$ *gdzie* D *- maks. stopień*

for *każdy* w *na liście korzeni* H do

    $x \leftarrow w$

    $d \leftarrow$ *degree*$[x]$

    while $A[d] \neq NIL$ do

        $y \leftarrow A[d]$

        if *key*$[x] >$ *key*$[y]$ then

            zamień $x \leftrightarrow y$

        Fib-Heap-Link(H,y,x) ▷ *następny slajd*

        $A[d] \leftarrow NIL$

        $d \leftarrow d + 1$

    $A[d] \leftarrow x$

...

Consolidate, Fib-Heap-Link

min$[H] \leftarrow NIL$
`for` $i \leftarrow 0$ `to` $D($*n*$[H])$ `do`
  `if` $A[i] \neq NIL$ `then`
    dodaj $A[i]$ *do listy korzeni* H
    `if` *min*$[H] = NIL$ `or` *key*`[`$A[i]$`]`$<$*key*`[`*min*$[H]$`]`
      `then` *min*$[H] \leftarrow A[i]$

`Fib-Heap-Link(`H`,`y`,`x`)`
usuń y *z listy korzeni* H
uczyń y *synem* x *i zwiększ degree*$[x]$ *o* 1
mark$[y] \leftarrow FALSE$

Analiza

$t(H') \leq D(n) + 1$ i $m(H') \leq m(H)$. Przyrost Φ:
$\leq ((D(n) + 1) + 2m(H)) - (t(H) + 2m(H)) \leq D(n) + 1 - t(H)$
Faktyczny koszt `Fib-Heap-Extract-Min`:

- $O(D(n))$ – linie `3-5` oraz w `Consolidate`: linie `1-2` i `14-19`
- `Consolidate` linie `3-13`:
  - Na początku dł. listy korzeni: $\leq D(n) + t(H) - 1$.
  - Pętla `for` wykonywana $\leq D(n) + t(H) - 1$ razy.
  - Każde wykonanie `while` eliminuje 1 korzeń, więc łącznie jest ich też $\leq D(n) + t(H) - 1$.

Stąd, faktyczny koszt: $O(D(n) + t(H))$.
Koszt zamortyzowany: $O(D(n) + t(H)) + D(n) + 1 - t(H) = O(D(n)) + O(t(H)) - t(H) = O(D(n))$

dalsze operacje

Poniższe operacje mogą spowodować, że kopce Fibonacciego nie będą zbiorem nieuporządkowanych drzew dwumianowych, ale dalej będzie: $D(n) = O(\lg n)$.

Fib-Heap-Decrease-Key(H,x,k)

if $k >$ *key*$[x]$ then
    error “nowa wartość klucza większa”
key$[x] \leftarrow k$
$y \leftarrow$ *p*$[x]$
if $y \neq NIL$ and *key*$[x] <$ *key*$[y]$ then
    Cut(H,x,y) $\triangleright$ *następny slajd*
    Cascading-Cut(H,y) $\triangleright$ *następny slajd*
if *key*$[x] <$ *key*[*min*$[H]$] then
    min$[H] \leftarrow x$

Cut, Cascading-Cut

`Cut(`H`,`x`,`y`)`

usuń x *z listy synów* y *i zmniejsz degree*$[y]$ *o 1*

dodaj x *do listy korzeni* H

p$[x] \leftarrow NIL$

mark$[x] \leftarrow FALSE$

`Cascading-Cut(`H`,`y`)`

$z \leftarrow$ *p*$[y]$

if $z \neq NIL$ **then**

    if *mark*$[y] = FALSE$ **then**

        mark$[y] \leftarrow TRUE$

    else

        `Cut(`H`,`y`,`z`)`

        `Cascading-Cut(`H`,`z`)`

analiza

`Fib-Heap-Decrease-Key(`H`,`x`,`k`)` – jeśli zmiana *key*$[x]$ narusza porządek kopca, to przerzuca x na liste korzeni i zapewnia przestrzeganie następującej reguły:

Reguła (dla każdego w): Po następującej sekwencji zdarzeń:

1. w zostaje korzeniem
2. w podłączony pod inny węzeł
3. w traci dwóch synów

w zostaje odcięty od swego ojca i zostaje korzeniem. (`Cascading-Cut` – rekurencyjnie wymusza tę zasadę również dla przodków w.)

Pole *mark* – "lampka ostrzegawcza" po utracie pierwszego syna.

`Fib-Heap-Decrease-Key(`H`,`x`,`k`)` – w razie potrzeby uaktualnia *min*$[H]$

analiza

Koszt faktyczny `Fib-Heap-Decrease-Key`:

- wiersze: `1-5` i `8-9` czas: $O(1)$
- wiersze: `6-7` czas: $O(c)$, gdzie c liczba rekurencyjnych wywołań `Cascading-Cut`
- łączny koszt: $O(c)$

Przyrost Φ ($c - 1$ pól *marked* – skasowane i ≤ 1 ustawione):
$((t(H) + c) + 2(m(H) - c + 2)) - (t(H) + 2m(H)) = 4 - c$
Koszt zamortyzowany: $O(c) + 4 - c = O(1)$

Fib-Heap-Delete

```
Fib-Heap-Delete(H,x)
1 Fib-Heap-Decrease-Key(H,x,−∞)
2 Fib-Heap-Extract-Min(H)
```

Koszt zamortyzowany: $O(1) + O(D(n)) = O(D(n))$.

Pozostaje oszacować $D(n)$ – maksymalny stopień.

Oszacowanie maksymalnego stopnia

Niech *size*(x) – liczba węzłów w poddrzewie o korzeniu x.

Lemat 1. Dla x – węzła, niech *degree*$[x] = k$, oraz $y_1 \ldots y_k$ – synowie x w kolejności podłączania pod x. Wtedy *degree*$[y_1] \geq 0$ oraz *degree*$[y_i] \geq i - 2$ dla $i = 2, \ldots, k$.

D-d. *degree*$[y_1] \geq 0$ – oczywiste.
Dla $i \geq 2$ – kiedy y_i był przyłączany do x, $y_1 \ldots y_{i-1}$ były już synami x więc było *degree*$[x] \geq i - 1$. y_i zostaje przyłączony do x tylko gdy *degree*$[x]$ =*degree*$[y_i]$. Stąd w chwili przyłączenia *degree*$[y_i] \geq i - 1$. Od tej pory y_i stracił ≤ 1 syna (bo nadal jest synem x). Stąd *degree*$[y_i] \geq i - 2$. $\square$

liczby Fibonacciego

$$F_k = \begin{cases} 0, & \text{jeśli } k = 0 \\ 1, & \text{jeśli } k = 1 \\ F_{k-1} + F_{k-2} & \text{jeśli } k \geq 2 \end{cases}$$

Lemat 2. Dla wszystkich $k \geq 0$: $F_{k+2} = 1 + \sum_{i=0}^{k} F_i$

D-d. Dla $k = 0$: $1 + \sum_{i=0}^{0} F_i = 1 + F_0 = 1 + 0 = 1 = F_2$.

Zakładając: $F_{k+1} = 1 + \sum_{i=0}^{k-1} F_i$ mamy:

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

$\square$

Ćwiczenie: Wykazać, że $F_{k+2} \geq \phi^k$, gdzie $\phi = (1 + \sqrt{5})/2$ ($= 1.61803\ldots$).

Oszacowanie $D(n)$ (c.d.)

Lemat 3. Niech x – węzeł i $k =$*degree*$[x]$. Wtedy *size*$(x) \geq F_{k+2} \geq \phi^k$.

D-d Niech s_k – najmniejsza możliwa wartość *size*(z) dla z takiego, że *degree*$[z] = k$. Widać, że: $s_0 = 1$, $s_1 = 2$ oraz $s_2 = 3$ (utrata wnuka).

Niech $y_1 \ldots y_k$ – synowie x w kolejności podłączania. Z lematu 1 mamy: *size*$(x) \geq s_k \geq 2 + \sum_{i=2}^{k} s_{i-2}$. ($x$ i y_1 wnoszą $\geq 1 + s_0 = 2$ a pozostałe – jak w sumie od $i = 2$ do k.)

Wykażemy przez indukcję, że $s_k \geq F_{k+2}$.

Przypadki $k = 0$ i $k = 1$ – oczywiste.

Załóżmy, że $k \geq 2$ i $s_i \geq F_{i+2}$ dla $i \leq k$. Zachodzi:

$s_k \geq 2 + \sum_{i=2}^{k} s_{i-2} \geq 2 + \sum_{i=2}^{k} F_i = 1 + \sum_{i=0}^{k} F_i = F_{k+2}$.

$\square$

Oszacowanie $D(n)$ (c.d.)

Wniosek. Maksymalny stopień $D(n)$ węzła w n-węzłowym kopcu Fibonacciego wynosi $O(\lg n)$.

D-d. Niech x – dowolny węzeł i $k =$*degree*$[x]$. Z lematu 3 $n \geq$*size*$(x) \geq \phi^k$. Biorąc logarytm przy podstawie ϕ otrzymujemy: $k \leq \log_\phi n$. Stąd największy stopień $D(n)$ jest $O(\lg n)$.
$\square$

Porównanie

kopiec: koszt:	binarny (pesym.)	dwumianowy (pesym.)	Fibonacciego (zamort.)
`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`	$\Theta(n)$	$O(\lg n)$	$\Theta(1)$
`Decrease-Key`	$\Theta(\lg n)$	$\Theta(\lg n)$	$\Theta(1)$
`Delete`	$\Theta(\lg n)$	$\Theta(\lg n)$	$O(\lg n)$