ALGORYTMY I STRUKTURY DANYCH
WYKŁAD V (materiały pomocnicze)

Struktury danych, kolejka priorytetowa, struktura Find-Union

Polsko Japońska Wyższa Szkoła Technik Komputerowych

Warszawa, 7 grudnia 2008

Plan wykładu:

- kolejka priorytetowa:
  - kopiec binarny - drzewo,
  - kopiec binarny - tablica,
  - kopiec binarny - efektywna budowa,
  - kopiec binarny - algorytm sortowania,
  - kopiec lewicowy,
- struktura Find-Union:
  - listy z balansowaniem,
  - drzewa n-arne z balansowaniem i kompresją ścieżek.

Kolejka priorytetowa

Kolejka priorytetowa

Idea (model standardowy kolejki priorytetowej):

- $\langle E \cup PQ \cup \{true, false\}, empty, member, min, insert, delmin \rangle$, gdzie PQ jest uniwersum multizbiorów,
- $empty\,(pq) \equiv_{df} (pq = \emptyset)$,
- $member\,(pq, e) \equiv_{df} (e \in pq)$,
- $min\,(pq) =_{df} (min\,(\{e : e \in pq\}))$,
- $insert\,(pq, e) =_{df} (pq \cup \{e\})$,
- $delmin\,(pq, e) =_{df} (pq \setminus min\,(\{e : e \in pq\}))$.

Specyfikacja kolejki priorytetowej:

- sygnatura:
  - $\langle E \cup PQ, empty, member, min, insert, delmin \rangle$,
  - $empty : PQ \to \{true, false\}$,
  - $member : PQ \times E \to \{true, false\}$,

Kolejka priorytetowa

Specyfikacja kolejki(c.d.):

- sygnatura:
  - $min : PQ \rightarrow E$,
  - $insert : PQ \times E \rightarrow PQ$,
  - $delmin : PQ \rightarrow PQ$,
- aksjomaty:
  - $\langle E, \leq \rangle$ jest zbiorem liniowo uporządkowanym,
  - $member\,(pq, e) \equiv P\,(pq, e)$, gdzie P jest następującym programem

```
while (!empty(pq)) {
   if (min(pq)==e) return true; else pq=delmin(pq);
}
return false;
```

Kolejka priorytetowa

Specyfikacja kolejki(c.d.):

- aksjomaty:
  - $\neg empty\,(pq) \Rightarrow (\forall e \in E\,(member\,(pq, e) \Rightarrow min\,(pq) \leq e))$,
  - $member\,(insert\,(pq, e)\,, e)$,
    $e \neq e' \Rightarrow member\,(pq, e) \equiv member\,(insert\,(pq, e')\,, e)\,,$
  - $member\,(min\,(pq)\,, pq)$,
    $e \neq min\,(pq) \Rightarrow member\,(pq, e) \equiv member\,(delmin\,(pq)\,, e)\,,$
  - program `while (!empty(pq)) pq=delmin(pq);` ma własność stopu.

Twierdzenie. *Dowolna struktura, która spełnia aksjomaty specyfikacji kolejki priorytetowej jest izomorficzna z pewną standardową strukturą kolejek priorytetowych.*

Pytanie. Jaka jest złożoność średnia i pesymistyczna operacji kolejki priorytetowej min, $insert$ oraz $delmin$ w przypadku implementacji struktury odpowiednio w drzewie BST i AVL (zakładamy, że elementy kolejki priorytetowej nie powtarzają się)?

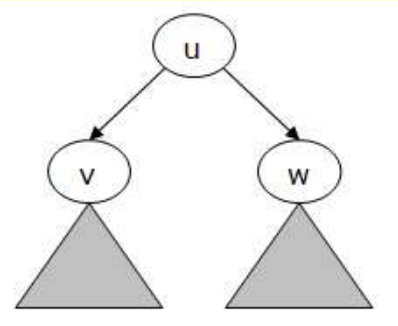
Kolejka priorytetowa

(kopiec binarny – drzewo)

Kolejka priorytetowa – kopiec binarny - drzewo

Definicja. *Kopcem binarnym (typu min)* nazywamy drzewo binarne $G = (V_G, E_G, et)$, gdzie:

- $et : V_G \to E$ jest funkcją etykietowania wierzchołków i E jest pewnym niepustym, liniowo uporządkowanym zbiorem etykiet $\langle E, \leq \rangle$,
- dla każdej trójki wierzchołków u, v, w, jeżeli:
  - v jest lewym następnikiem wierzchołka u, to $et\,(u) \leq et\,(v)$,
  - w jest prawym następnikiem wierzchołka u, to $et\,(u) \leq et\,(w)$,

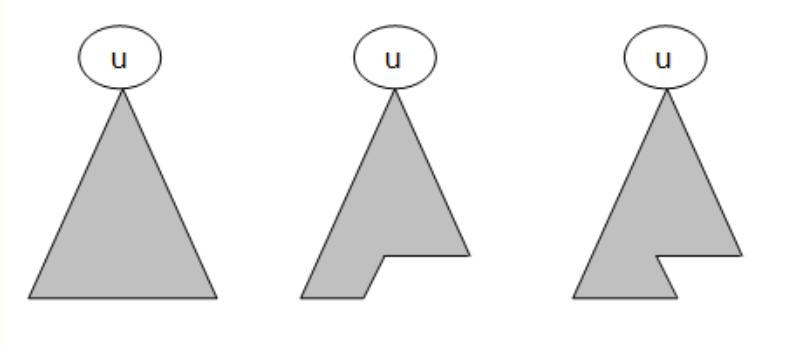


Uwaga! W dalszej części wykładu kopiec binarny będziemy nazywali kopcem.

Kolejka priorytetowa – kopiec binarny - drzewo

Definicja (c.d.).

- drzewo jest drzewem doskonałym, z ewentualnym wyjątkiem ostatniego poziomu, na którym wszystkie liście są zgrupowane skrajnie na lewo (tzw. lewostronne wypełnienie)



Uwaga! Analogiczną definicję można wprowadzić dla kopców typu max. W dalszej części tego wykładu kopcem będziemy domyślnie nazywali kopiec typu min.

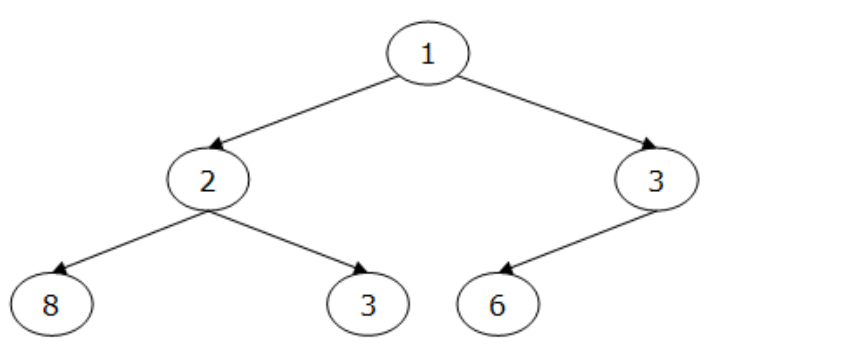
Pytanie. Czy kopiec jest drzewem zrównoważonym w sensie zrównoważenia struktury AVL?

Pytanie. Z ilu co najmniej i co najwyżej wierzchołków składa się kopiec wysokości 6?

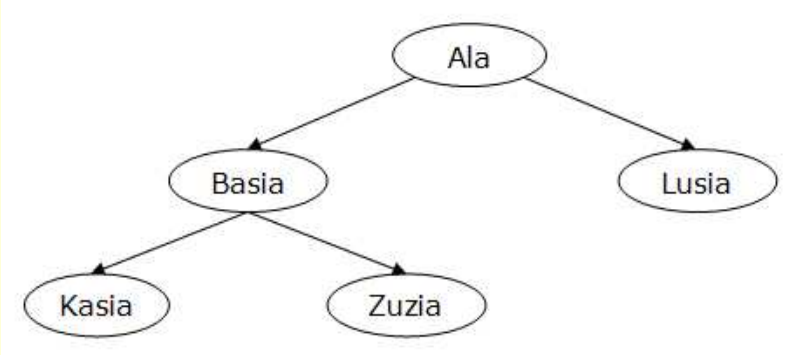
Kolejka priorytetowa – kopiec binarny - drzewo

Przykłady:

- zbiór etykiet $\langle \mathbb{N}, \leq \rangle$:

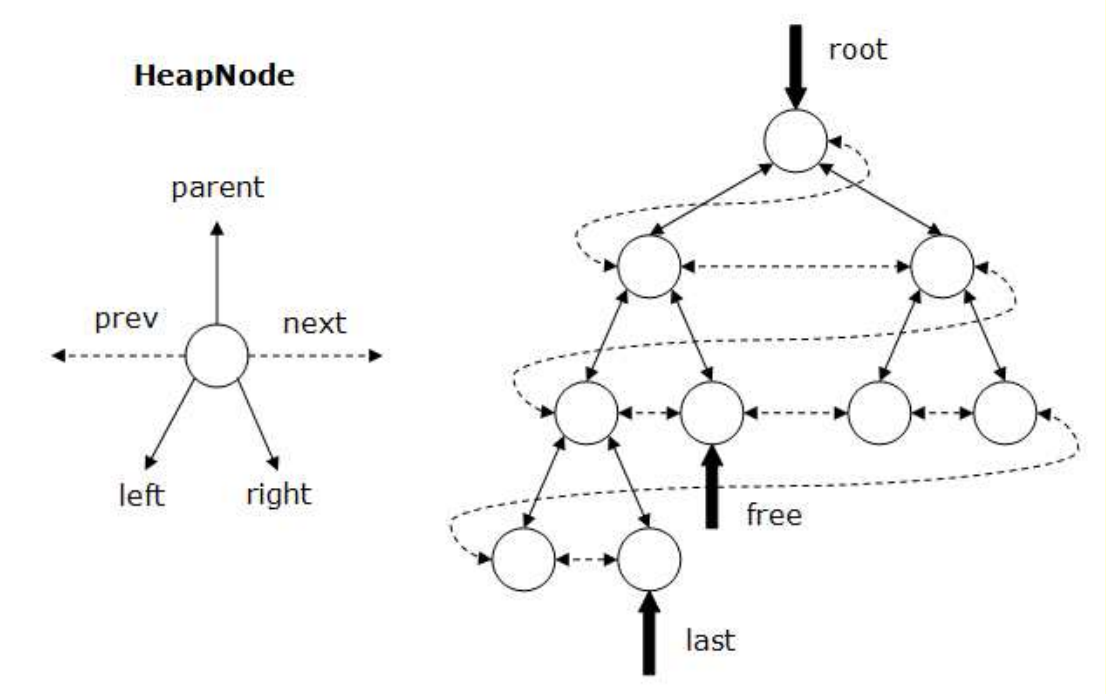


- zbiór etykiet $\langle \Pi, \leq_{leks} \rangle$, gdzie Π jest zbiorem słów języka polskiego:



Kolejka priorytetowa – kopiec binarny - drzewo

Szczegóły implementacji.

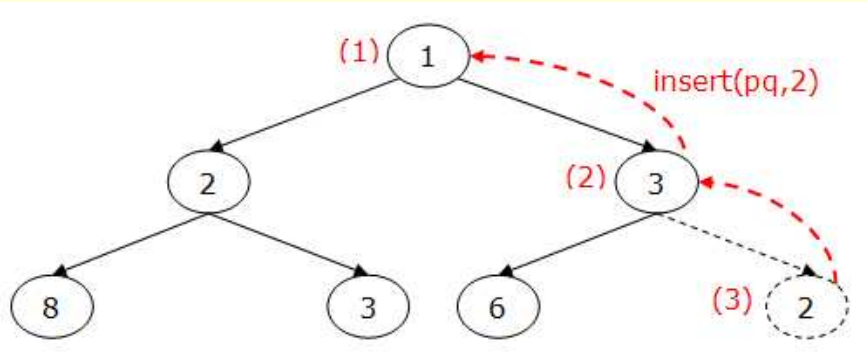


Kolejka priorytetowa – kopiec binarny - drzewo

Operacji $insert\,(pq, e)$ – **idea.** Niech H będzie kopcem-drzewem, będącym implementacją kolejki priorytetowej pq dla uniwersum elementów E, i niech e będzie elementem uniwersum E, wtedy:

- utwórz nowy wierzchołek z etykietą e na ostatnim poziomie drzewa i na „pierwszej wolnej" skrajnie lewej pozycji (odpowiednio dowiązania $free.left$ albo $free.right$),
- rozpoczynając od nowo utworzonego wierzchołka v:
  - jeżeli $et\,(v) < et\,(v.parent)$, zamień etykiety wierzchołków v oraz $v.parent$, przejdź do wierzchołka $v.parent$ i powtórz postępowanie, w p.p. zakończ działanie algorytmu.

Przykład. Wstawiamy do kolejki priorytetowej $pq = \{1, 2, 3, 3, 6, 8\}$ element 2.



Kolejka priorytetowa – kopiec binarny - drzewo

Operacja $delmin\,(pq)$ – **idea.** Niech H będzie kopcem-drzewem, będącym implementacją kolejki priorytetowej pq dla uniwersum elementów E, wtedy:

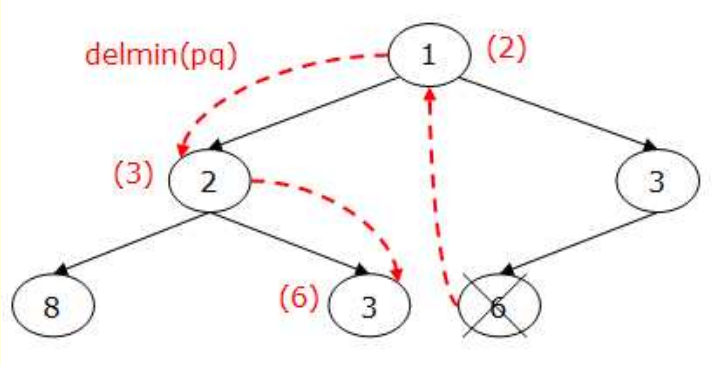
- zamień etykiety wierzchołka korzenia oraz wierzchołka v znajdującego się na ostatnim poziomie drzewa i na „ostatniej zajętej" skrajnie prawej pozycji (dowiązanie $last$),
- usuń wierzchołek v,
- rozpoczynając od korzenia kopca (jeżeli $empty\,(pq) = false$):
  - niech v będzie aktualnie rozważanym wierzchołkiem, wtedy jeżeli

$$et\,(v) > min\,(\{et\,(v.left)\,, et\,(v.right)\})\,,$$

    to zamień etykietę wierzchołka v z mniejszą z etykiet wierzchołków następników wierzchołka v, niech będzie to $et\,(u)$, przejdź do wierzchołka u i powtórz powyższe postępowanie, w p.p. zakończ działanie algorytmu.

Kolejka priorytetowa – kopiec binarny - drzewo

Przykład. Usuwamy wierzchołek minimalny z kolejki priorytetowej $pq = \{1, 2, 3, 3, 6, 8\}$.



Pytanie. Jaka jest złożoność średnia i pesymistyczna operacji kolejki priorytetowej min, $insert$ oraz $delmin$ w przypadku implementacji struktury w kopcu-drzewie?

Pytanie. Jaka jest złożoność średnia i pesymistyczna operacji $member$ w przypadku implementacji struktury w kopcu-drzewie?

Kolejka priorytetowa

(kopiec binarny – tablica)

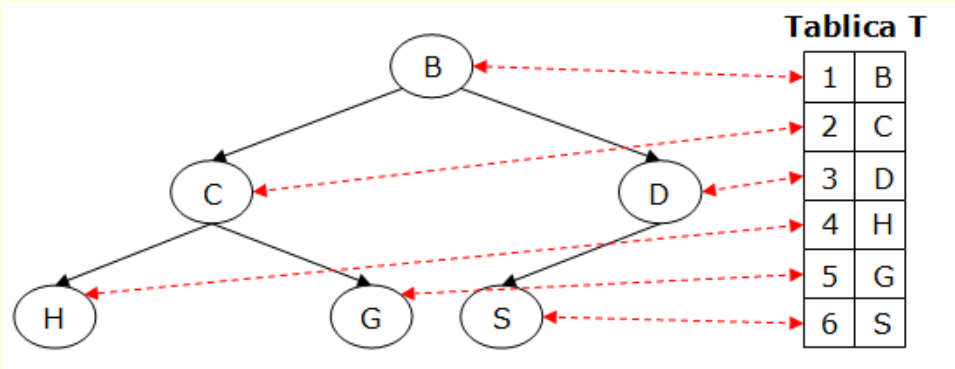
Kolejka priorytetowa – kopiec binarny - tablica

Pytanie. Czy kopiec binarny można efektywnie zaimplementować w tablicy statycznej?

Odpowiedź. Tak, przy założeniu, że „z góry" znamy maksymalną liczbę n elementów przechowywanych w strukturze kopca. Wtedy dla każdego wierzchołka kopca-drzewa v, jeżeli indeks elementu v w tablicy statycznej T równy jest i, to (dla ułatwienia przyjmujemy, że tablica T indeksowana jest począwszy od wartości 1 do n):

- następnik lewy oraz prawy wierzchołka v, o ile istnieją, to elementy tablicy $T\left[2 \cdot i\right]$ oraz $T\left[2 \cdot i + 1\right]$,
- poprzednik wierzchołka v, o ile istnieje, to element $T\left[\left\lfloor \frac{i}{2} \right\rfloor\right]$.

Przykład. Kolejka priorytetowa $pq = \{B, C, D, H, G, S\}$ i jej równoważne implementacje, kopiec-drzewo i kopiec-tablica.

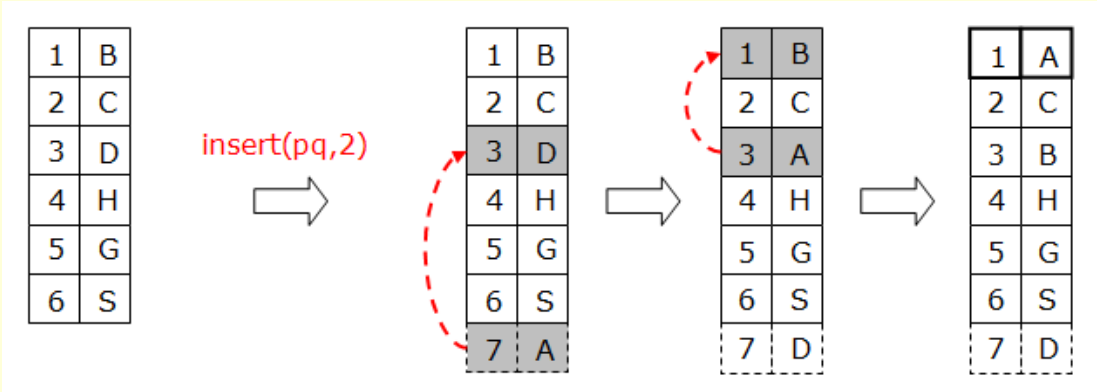


Kolejka priorytetowa – kopiec binarny - tablica

Operacja $insert\,(pq, e)$ – **idea.** Niech T będzie kopcem-tablicą, będącą implementacją kolejki priorytetowej pq dla uniwersum elementów E, i niech e będzie elementem uniwersum E, wtedy:

- wstaw element e na „pierwszą wolną" pozycję w tablicy T, niech będzie to pozycja i-ta,
- rozpoczynając rozpoczynając od elementu $T\,[i]$:
  - jeżeli $T\,[i] < T\left[\frac{i}{2}\right]$, zamień elementy $T\,[i]$ oraz $T\left[\frac{i}{2}\right]$, podstaw $i = \left\lfloor \frac{i}{2} \right\rfloor$ i powtórz postępowanie, w p.p. zakończ działanie algorytmu.

Przykład. Wstawiamy do kolejki priorytetowej $pq = \{B, C, D, H, G, S\}$ element A.



Pytanie. Jaka jest złożoność średnia i pesymistyczna operacji kolejki priorytetowej $insert$ w przypadku implementacji struktury w kopcu-tablicy?

Kolejka priorytetowa – kopiec binarny - tablica

Operacja $delmin\,(pq)$ – **idea.** Niech H będzie kopcem-tablicą, będącą implementacją kolejki priorytetowej pq dla uniwersum elementów E, wtedy:

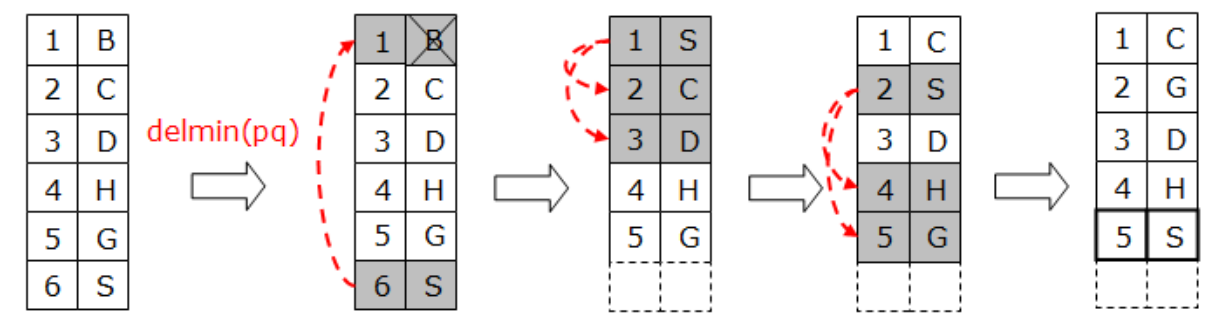
- podstaw $T\,[1] = T\,[i]$, gdzie i jest indeksem „ostatniej zajętej" pozycji w tablicy T,
- usuń element i-ty tablicy T,
- rozpoczynając od elementu $T\,[1]$ (jeżeli $empty\,(pq) = false$):
  - niech j będzie indeksem aktualnie rozważanego elementu w tablicy T, wtedy jeżeli

$$T\,[j] > min\,(\{T\,[2 \cdot j]\,, T\,[2 \cdot j + 1]\})\,,$$

    to zamień $T\,[j]$ z mniejszym z elementów $T\,[2 \cdot j]\,, T\,[2 \cdot j + 1]$, niech będzie to $T\,[k]$, podstaw $j = k$ i powtórz powyższe postępowanie, w p.p. zakończ działanie algorytmu.

Kolejka priorytetowa – kopiec binarny - tablica

Przykład. Usuwamy wierzchołek minimalny z kolejki priorytetowej $pq = \{B, C, D, H, G, S\}$.



Pytanie. Jaka jest złożoność średnia i pesymistyczna operacji kolejki priorytetowej $delmin$ w przypadku implementacji struktury w kopcu-tablicy?

Pytanie. Jaka jest złożoność średnia i pesymistyczna operacji min oraz $member$ w przypadku implementacji struktury w kopcu-tablicy?

Pytanie. Jak efektywnie wyznaczyć pierwszą wolną/ostatnią zajętą pozycję w kopcu-tablicy T? Czy brak dodatkowej pamięci zmieni rząd złożoności operacji $insert$ albo $delete$?

Kolejka priorytetowa

(kopiec binarny – efektywna budowa)

Kolejka priorytetowa – kopiec binarny - efektywna budowa

Idea algorytmu HeapConstruct. Niech $e_1, e_2, \ldots, e_n$ będzie ciągiem n elementów pewnego zbioru E z wyróżnioną relacją porządku liniowego $\leq$:

- zapisz elementy ciągu w tablicy T (dla ułatwienia przyjmujemy, że tablica T indeksowana jest począwszy od wartości 1 do n),
- dla $i = \left\lfloor \frac{n}{2} \right\rfloor, \left\lfloor \frac{n}{2} \right\rfloor - 1, \ldots, 1$ wykonaj (*):
  - niech j będzie indeksem aktualnie rozważanego elementu w tablicy T, wtedy jeżeli

$$T[j] > min\left(\{T[2 \cdot j], T[2 \cdot j + 1]\}\right),$$

to zamień $T[j]$ z mniejszym z elementów $T[2 \cdot j], T[2 \cdot j + 1]$, niech będzie to $T[k]$, podstaw $j = k$ i powtórz powyższe postępowanie, w p.p. przerwij działanie i powróć do (*).

Kolejka priorytetowa – kopiec binarny - efektywna budowa

Zadanie. Przedstaw „krok po kroku" działanie algorytmu HeapConstruct dla ciągu liczb 4,5,2,8,9,4,1,7,6.

Fakt. Pesymistyczną złożoność czasową algorytmu HeapConstruct można ograniczyć przez

$$W(n) \leq \sum_{h=0}^{\lfloor \lg n \rfloor} \left(\left\lceil \frac{n}{2^{h+1}} \right\rceil \cdot O(h) \right),$$

gdzie $\left\lceil \frac{n}{2^{h+1}} \right\rceil$ jest górnym ograniczeniem liczby węzłów będących korzeniami poddrzew wysokości h w kopcu n-elementowym, stąd

$$W(n) \leq n \sum_{h=0}^{\lfloor \lg n \rfloor} \left\lceil \frac{O(h)}{2^{h+1}} \right\rceil = O\left(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h} \right)$$

i ponieważ $\sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h} \leq \sum_{h=0}^{\infty} \frac{h}{2^h} = 2$, to $W(n) = O(n)$.

Pytanie. Czy złożoność czasową algorytmu HeapConstruct w wariancie implementacji strukturze dowiązaniowej (drzewie binarnym) jest także rzędu $O(n)$?

Pytanie. Jaka jest złożoność pamięciowa algorytmu HeapConstruct?

Kolejka priorytetowa

(kopiec binarny – sortowanie)

Kolejka priorytetowa – kopiec binarny - algorytm sortowania

Idea algorytmu HeapSort. Niech $e_1, e_2, \ldots, e_n$ będzie ciągiem n elementów pewnego zbioru E z wyróżnioną relacją porządku liniowego $\leq$, wtedy:

- zbuduj kopiec prze kolejne wstawienie elementów rozważanego ciągu do początkowo pustej struktury albo stosując algorytm HeapConstruct,
- wykonaj n razy operację min oraz $delmin$.

Rezultatem działania algorytmu jest uporządkowana niemalejąco permutacja elementów ciągu $e_1, e_2, \ldots, e_n$.

Zadanie. Przedstaw „krok po kroku" działanie algorytmu HeapSort dla ciągu liczb 4,5,2,8,9,4,1,7,6.

Pytanie. Jaka jest średnia i pesymistyczna złożoność czasowa algorytmu HeapSort?

Pytanie. Jaka jest złożoność pamięciowa algorytmu HeapSort?

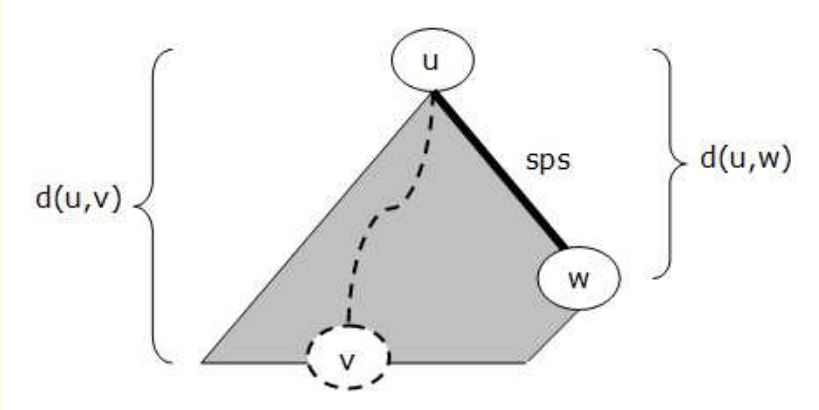
Kolejka priorytetowa

(kopiec lewicowy)

Kolejka priorytetowa – kopiec lewicowy

Definicja. *Kopcem lewicowym* nazywamy drzewo binarne $G = (V_G, E_G, et)$, gdzie:

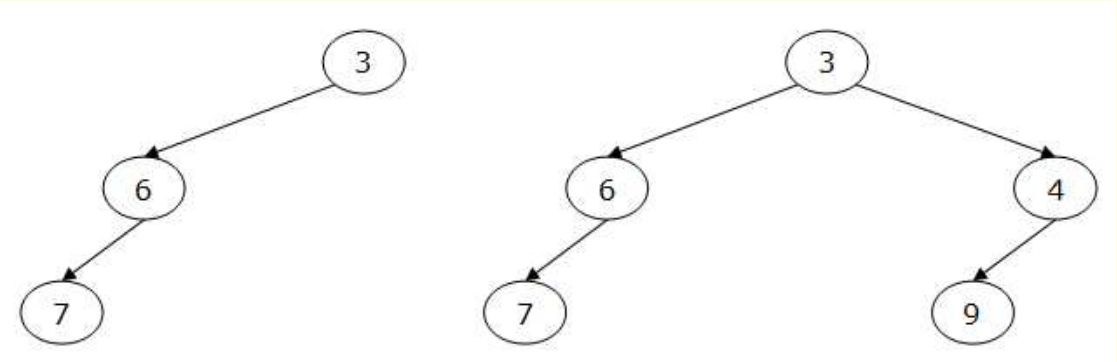
- $et : V_G \to E$ jest funkcją etykietowania wierzchołków i E jest pewnym niepustym, liniowo uporządkowanym zbiorem etykiet $\langle E, \leq \rangle$,
- etykiety wierzchołków ułożone są zgodnie z porządkiem kopcowym (typu min albo max),
- dla każdej trójki wierzchołków u, v, w, gdzie wierzchołki u, w są ustalone oraz wierzchołek w nie posiadają prawego następnika a wierzchołek v nie posiada lewego lub prawego następnika, różnica długości ścieżek z wierzchołka u do wierzchołka v oraz z wierzchołka u do wierzchołka w (tzw. *skrajnie prawa ścieżka*, inaczej $sps\,(u) = d\,(u, w)$), jest liczbą nieujemną (tj. $sps\,(u) \leq d\,(u, v)$).



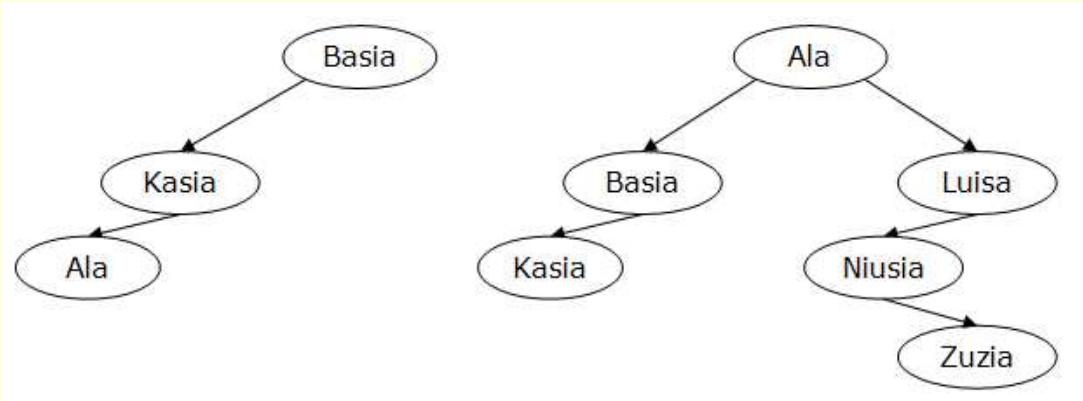
Kolejka priorytetowa – kopiec lewicowy

Przykłady:

- drzewa binarne będące kopcami lewicowymi, zbiór etykiet $\langle \mathbb{N}, \leq \rangle$:



- drzewa binarne nie będące kopcami lewicowymi, zbiór etykiet $\langle \Pi, \leq_{leks} \rangle$:



Kolejka priorytetowa – kopiec lewicowy

Lemat. *Niech H będzie kopcem lewicowym składającym się z n wierzchołków. Długość skrajnie prawej ścieżki w kopcu H jest nie większa niż $\lfloor \lg n \rfloor$.*

Dowód. Załóżmy, że H jest n-elementowym kopcem lewicowym, w którym skrajnie prawa ścieżka jest długości d większej niż $\lfloor \lg n \rfloor$. Ponieważ kopiec H jest drzewem lewicowym, to do poziomu d włącznie jest także drzewem doskonałym (w p.p. istniałaby ścieżka korzeń-wierzchołek nie posiadający lewego lub prawego następnika o długości mniejszej niż d, czyli kopiec H nie byłby drzewem lewicowym). Stąd kopiec H zbudowany jest z co najmniej $2^{d+1} - 1$ wierzchołków. Ponieważ z założenia $d > \lfloor \lg n \rfloor$, to $2^{d+1} - 1 > 2^{\lfloor \lg n \rfloor + 1} - 1 \geq 2^{\lg n}$, czyli liczba wierzchołków w kopcu H jest większa niż $2^{\lg n} = n$ – sprzeczność.

Ostatecznie w każdym n-elementowym kopcu lewicowym długość skrajnie prawej ścieżki jest mniejsza albo równa $\lfloor \lg n \rfloor$.

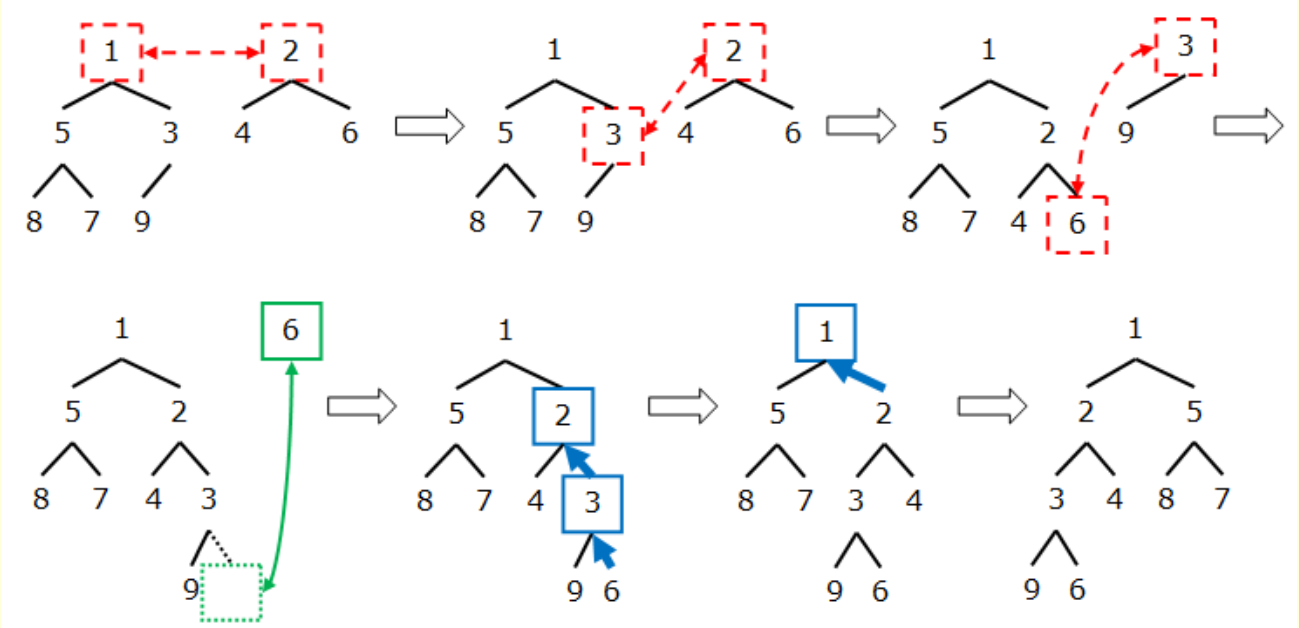
Kolejka priorytetowa – kopiec lewicowy

Operacja $merge\,(v_1, v_2)$ – **idea.** Niech v_1 oraz v_2 będą dowiązaniami do korzeni kopców lewicowych odpowiednio H_1 oraz H_2. Wykonaj kolejno:

- jeżeli drzewo H_1 albo H_2 jest drzewem pustym, to:
  - jeżeli drzewo H_1 jest drzewem pustym, to rezultatem scalania jest drzewo H_2, w.p.p. rezultatem scalania jest drzewo H_1,
- w p.p.:
  - jeżeli $et\,(v_1) > et\,(v_2)$, to zamień miejscami poddrzewa o korzeniach v_1 oraz v_2,
  - wykonaj rekurencyjnie scalanie $v_1.right = merge(v_1.right, v_2)$,
  - jeżeli $d\,(sps\,(v_1.left)) < d\,(sps\,(v_1.right))$, to zamień miejscami poddrzewa o korzeniach $v_1.left$ oraz $v_1.right$,
  - rezultatem scalania jest drzewo o korzeniu w wierzchołku v_1.

Kolejka priorytetowa – kopiec lewicowy

Przykład. Scalanie dwóch kopców lewicowych z użyciem metody $merge$. Kolorem czerwonym zaznaczono wierzchołki będące argumentami porównania $et(v_1) > et(v_2)$, kolorem zielonym właściwe scalanie a kolorem niebieskim wierzchołek korzeń będący argumentem porównania $d(sps(v_1.left)) < d(sps(v_1.right))$.



Kolejka priorytetowa – kopiec lewicowy

Wniosek. Jeżeli v_1 oraz v_2 są dowiązaniami do korzeni kopców lewicowych, to rezultat operacji $merge(v_1.right, v_2)$ jest także kopcem lewicowym.

Wniosek. Niech v_1 oraz v_2 będą dowiązaniami do korzeni kopców lewicowych odpowiednio n i m wierzchołkowego, wtedy

$$W\left(merge(v_1.right, v_2), n, m\right) = O\left(\lg\left(\max\left(n, m\right)\right)\right).$$

.**Prytanie.** Jaka jest złożoność pamięciowa operacji $merge$?

Kolejka priorytetowa – kopiec lewicowy

Operacja $insert\,(pq, e)$ – **idea.** Niech H_1 będzie kopcem lewicowym z wierzchołkiem korzeniem v_1, będącym implementacją kolejki priorytetowej pq dla uniwersum elementów E, i niech e będzie elementem uniwersum E, wtedy:

- utwórz nowy 1-wierzchołkowy kopiec lewicowy H_2 o korzeniu v_2 z etykietą e,
- wykonaj $v_1 = merge\,(v_1, v_2)$.

Operacja $delmin\,(pq)$ – **idea.** Niech H będzie kopcem lewicowym z wierzchołkiem korzeniem v, będącym implementacją kolejki priorytetowej pq dla uniwersum elementów E, wtedy:

- usuń wierzchołek v,
- wykonaj $v = merge\,(v.left, v.right)$.

Pytanie. Jaka jest złożoność czasowa i pamięciowa operacji kolejki priorytetowej min, $insert$, $delmin$ w przypadku implementacji struktury w kopcu lewicowym?

Struktura Find-Union

Struktura Find-Union

Przypomnienie. Niech $E = \{e_1, e_2, \ldots, e_n\}$ będzie zbiorem, *podziałem zbioru* E nazywamy rodzinę zbiorów $U = \{S_1, S_2, \ldots, S_k\}$ taką, że:

- $S_i \neq \emptyset$, dla każdego $1 \leq i \leq k$,
- $S_i \cap S_j = \emptyset$, dla każdego $1 \leq i < j \leq k$,
- $\bigcup_{i=1}^{k} S_i = E$.

Idea struktury Find-Union: niech $U = \{S_1, S_2, \ldots, S_k\}$ będzie podziałem zbioru $E = \{e_1, e_2, \ldots, e_n\}$ a $\mathcal{U}$ zbiorem wszystkich możliwych podziałów zboru E, wtedy

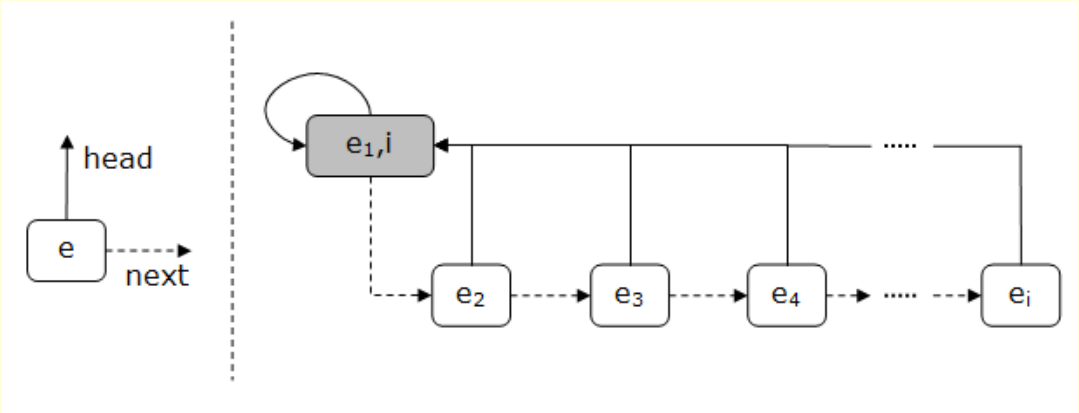
- $\langle E \cup \mathcal{U}, init, find, union \rangle$,
- $init\,(E) =_{df} (U)$ taki, że dla każdego $1 \leq i \leq k$ zachodzi $|S_i| = 1$,
- $find\,(U, e) =_{df} (S_i \in U)$ taki, że $e \in S_i$ i $1 \leq i \leq k$,
- $union\,(U, S_i, S_j) =_{df} (U')$ taki, że $U' = (U \setminus \{S_i, S_j\}) \cup \{S_i \cup S_j\}$ i $1 \leq i < j \leq k$.

Struktura Find-Union

(listy z balansowaniem)

Struktura Find-Union – listy z balansowaniem

Reprezentacja zbioru S_i:



gdzie:

- e_1 – reprezentant zbioru S_i,
- i – licznik elementów zbioru S_i, tj. $i = |S_i|$.

Struktura Find-Union – listy z balansowaniem

Szkic realizacji operacji:

- $init\,(E)$ – utworzenie n jednoelementowych list,
- $find\,(U, e)$ – odczytanie etykiety elementu $e.head$,
- $union\,(U, S_i, S_j)$ – przyłączenie listy krótszej na koniec dłuższej (tzw. *balansowanie*), dla każdego elementu listy krótszej zmiana dowiązań do reprezentanta zbioru.

Zadanie. Przedstaw krok po kroku stan struktury Find-Union U dla zbioru $E = \{1, 2, 3, 4, 5, 6\}$ i ciągu operacji:

$$init\,(E)\,; union\,(U, find\,(U, 1)\,, find\,(U, 2))\,; union\,(U, find\,(U, 5)\,, find\,(U, 6))\,;$$

$$union\,(U, find\,(U, 2)\,, find\,(U, 6))\,; union\,(U, find\,(U, 3)\,, find\,(U, 6))\,.$$

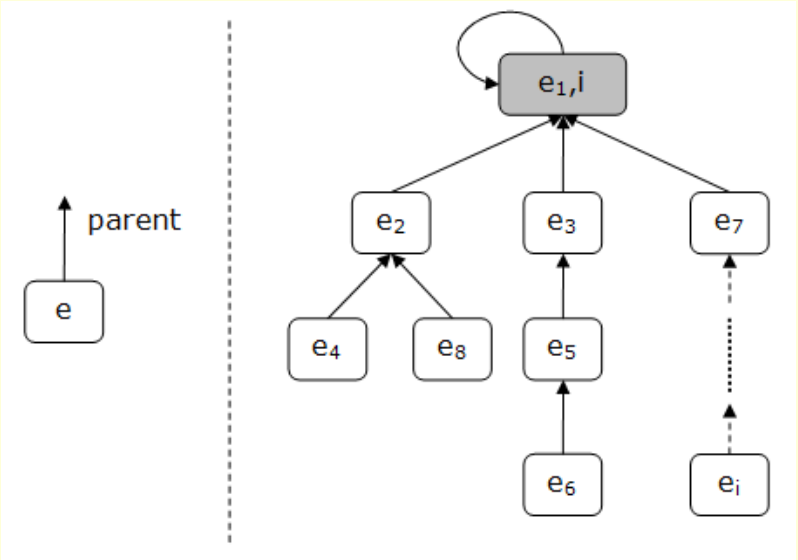
Twierdzenie. *Koszt ciągu operacji $init$, oraz przemieszanych m operacji $find$ oraz n operacji $union$ dla implementacji struktury Find-Union w postaci zbioru list z balansowaniem jest rzędu $O\,(m + n \lg n)$.*

Struktura Find-Union

(drzewa n-arne z balansowaniem i kompresją ścieżek)

Struktura Find-Union – drzewa n-arne z balansowaniem i kompresją ścieżek

Reprezentacja zbioru S_i:



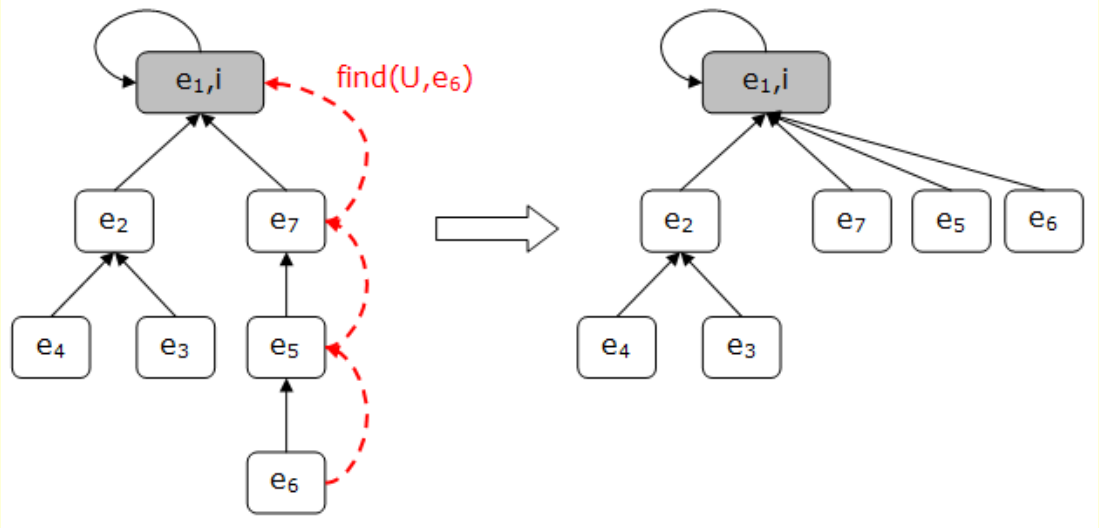
gdzie:

- e_1 – reprezentant zbioru S_i,
- i – licznik elementów zbioru S_i, tj. $i = |S_i|$.

Struktura Find-Union – drzewa n-arne z balansowaniem i kompresją ścieżek

Szkic realizacji operacji:

- $init\,(E)$ – utworzenie n jednoelementowych drzew,
- $find\,(U, e)$ – przejście ścieżki z wierzchołka o etykiecie e do korzenia drzewa i odczytanie etykiety w korzeniu drzewa, w trakcie przechodzenia „dowiązanie" atrybutu $parent$, wszystkich odwiedzonych wierzchołków, bezpośrednio do korzenia drzewa (tzw. *kompresja ścieżek*), np.



Struktura Find-Union – drzewa n-arne z balansowaniem i kompresją ścieżek

Szkic realizacji operacji (c.d.):

- $union\,(U, S_i, S_j)$ – przyłączenie mniejszego drzewa bezpośrednio do korzenia drzewa większego (tzw. *balansowanie*).

Zadanie. Przedstaw krok po kroku stan struktury Find-Union U dla zbioru $E = \{1, 2, 3, 4, 5, 6\}$ i ciągu operacji:

$$init\,(E)\,; union\,(U, find\,(U, 1)\,, find\,(U, 2))\,; union\,(U, find\,(U, 5)\,, find\,(U, 6))\,;$$

$$union\,(U, find\,(U, 2)\,, find\,(U, 6))\,; union\,(U, find\,(U, 3)\,, find\,(U, 6))\,.$$

Twierdzenie. *Koszt ciągu operacji $init$, oraz przemieszanych m operacji $find$ oraz n operacji $union$ dla implementacji struktury Find-Union w postaci zbioru drzew z balansowaniem i kompresją ścieżek jest rzędu $O\left((m+n)\lg^* n\right)$, gdzie*

$$\lg^* n = \min\left(i \in \mathbb{N} : \left\lfloor \underbrace{\lg \lg \ldots \lg}_{i}(n) \right\rfloor = 1\right).$$