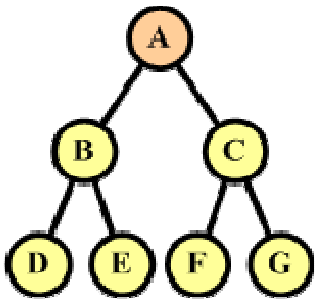
Sortowanie stogowe

Drzewo binarne
Binary Tree

Dotychczas operowaliśmy na prostych strukturach danych, takich jak **tablice**. W tablicy elementy ułożone są zgodnie z ich numeracją, czyli **indeksami**. Jeśli za punkt odniesienia weźmiemy element $d[i]$ ($i = 2,3,..,n\text{-}1$; n - liczba elementów w tablicy), to elementem poprzedzającym go będzie element o mniejszym o 1 indeksie, czyli $d[i - 1]$. Elementem następnym będzie element o indeksie o 1 większym, czyli $d[i + 1]$. Jest to tzw. **hierarchia liniowa** - elementy następują jeden za drugim. Graficznie możemy przedstawić to tak:

Pierwszy element $d[1]$ nie posiada **poprzednika** (ang. predecessor - elementu poprzedzającego w ciągu). Ostatni element $d[n]$ nie posiada **następnika** (ang. successor - elementu następnego w ciągu). Wszystkie pozostałe elementy posiadają poprzedniki i następniki.

Drzewo binarne jest hierarchiczną strukturą danych, którego elementy będziemy nazywali **węzłami** (ang. node) lub **wierzchołkami**. W hierarchii liniowej każdy element może posiadać co najwyżej jeden **następnik**. W drzewie binarnym każdy węzeł może posiadać dwa następniki (stąd pochodzi nazwa drzewa - binarny = dwójkowy, zawierający dwa elementy), które nazwiemy **potomkami**. **dziećmi** lub **węzłami potomnymi** danego węzła (ang. child node).



Węzły są połączone **krawędziami** symbolizującymi następstwo kolejnych elementów w strukturze drzewa binarnego. Według rysunku po prawej stronie węzeł **A** posiada dwa węzły potomne: **B** i **C**. Węzeł **B** nosi nazwę **lewego potomka** (ang. left child node), a węzeł **C** nosi nazwę **prawego potomka** (ang. right child node).

Z kolei węzeł **B** posiada węzły potomne **D** i **E**, a węzeł **C** ma węzły potomne **F** i **G**. Jeśli dany węzeł nie posiada dalszych węzłów potomnych, to jest w strukturze drzewa binarnego **węzłem terminalnym**. Taki węzeł nosi nazwę **liścia** (ang. leaf node). Na naszym rysunku liściami są węzły terminalne **D**, **E**, **F** i **G**.

Rodzicem, **przodkiem** (ang. parent node) lub **węzłem nadrzędnym** będziemy nazywać węzeł leżący na wyższym poziomie hierarchii drzewa binarnego. Dla węzłów **B** I **C** węzłem nadrzędnym jest węzeł **A**. Podobnie dla węzłów **D** i **E** węzłem nadrzędnym będzie węzeł **B**, a dla **F** i **G** będzie to węzeł **C**.

Węzeł nie posiadający rodzica nazywamy **korzeniem** drzewa binarnego (ang. root node). W naszym przykładzie korzeniem jest węzeł **A**. Każde drzewo binarne, które zawiera węzły posiada dokładnie jeden korzeń.

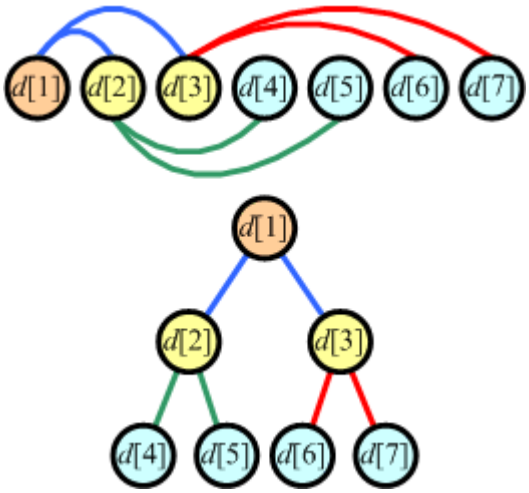
Odwzorowanie drzewa binarnego

Jeśli chcemy przetwarzać za pomocą komputera struktury drzew binarnych, to musimy zastanowić się nad sposobem reprezentacji takich struktur w pamięci. Najprostszym rozwiązaniem jest zastosowanie zwykłej tablicy n elementowej. Każdy element tej tablicy będzie reprezentował jeden węzeł drzewa binarnego. Pozostaje nam jedynie określenie związku pomiędzy indeksami elementów w tablicy a położeniem tych elementów w strukturze drzewa binarnego.

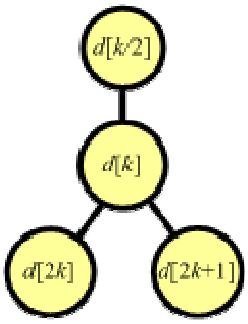
Zastosujmy następujące odwzorowanie:

- Element $d[1]$ będzie zawsze korzeniem drzewa.
- i-ty poziom drzewa binarnego wymaga 2^{i-1} węzłów. Będziemy je kolejno pobierać z tablicy.

Otrzymamy w ten sposób następujące odwzorowanie elementów tablicy w drzewo binarne:



Dla węzła k-tego wyprowadzamy następujące wzory:



węzły potomne mają indeksy równe:

$2k$ - lewy potomek

$2k+1$ - prawy potomek

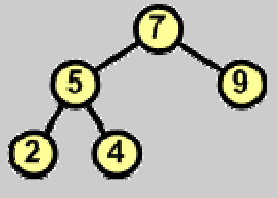
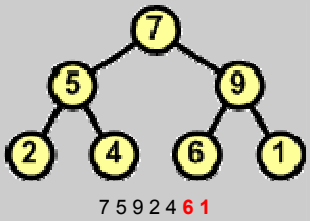
węzeł nadrzędny ma indeks równy $[k/2]$ (dzielenie całkowitoliczbowe)

Sprawdź, iż podane wzory są również spełnione w drzewach binarnych o większych rozmiarach niż prezentuje nasz przykład (pomocna może być kartka papieru).

Przykład:

Skonstruować drzewo binarne z elementów zbioru $\{7\ 5\ 9\ 2\ 4\ 6\ 1\}$

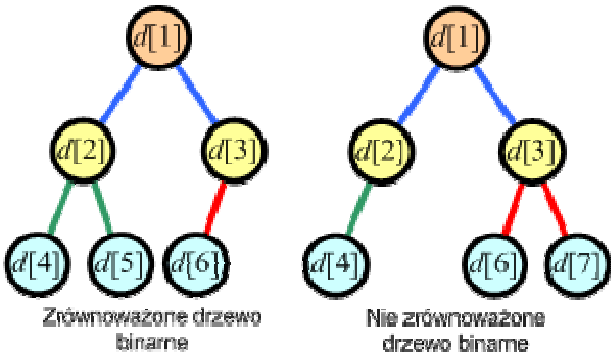
Operacja	Opis
 7 5 9 2 4 6 1	Konstrukcję drzewa binarnego rozpoczynamy od korzenia, który jest pierwszym elementem zbioru, czyli liczbą 7.
 7 5 9 2 4 6 1	Do korzenia dołączamy dwa węzły potomne, które leżą obok w zbiorze. Są to dwa kolejne elementy, 5 i 9.

	Do lewego węzła potomnego (5) dołączamy jego węzły potomne. Są to kolejne liczby w zbiorze, czyli 2 i 4.
	Pozostaje nam dołączyć do prawego węzła ostatnie dwa elementy zbioru, czyli liczby 6 i 1. Drzewo jest kompletne.

Zrównoważone drzewa binarne
Balanced Binary Trees

Umówmy się na potrzeby tego artykułu, iż binarne drzewo jest **zrównoważone i uporządkowane**, jeśli na wszystkich poziomach za wyjątkiem ostatniego posiada maksymalną liczbę węzłów, a na poziomie ostatnim węzły są ułożone kolejno od lewej strony. Innymi słowy, jeśli ostatni węzeł drzewa binarnego posiada numer i-ty, to drzewo zawiera wszystkie węzły od numeru 1 do i.

Warunek ten gwarantuje nam, iż każdy element tablicy będzie reprezentował pewien węzeł w drzewie binarnym - czyli w tej strukturze nie wystąpią dziury.



Drzewo po lewej stronie nie posiada węzła $d[7]$. Ale posiada wszystkie węzły od $d[1]$ do $d[6]$, jest zatem zrównoważone i uporządkowane. Można je bez problemu przedstawić za pomocą tablicy elementów od $d[1]$ do $d[6]$.

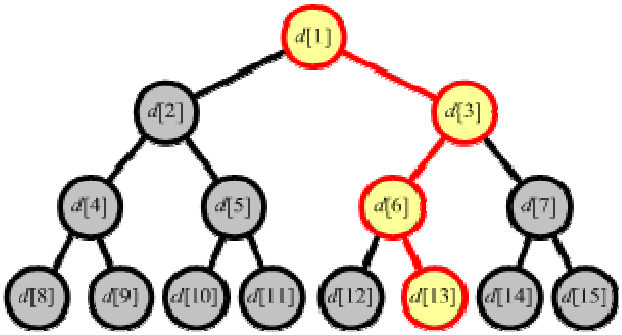
Drzewo po prawej stronie nie posiada węzła $d[5]$. Takiego drzewa nie przedstawimy poprawnie za pomocą tablicy elementów od $d[1]$ do $d[7]$, ponieważ nie mamy możliwości zaznaczenia

(bez dodatkowych zabiegów), iż element $d[5]$ nie należy do struktury drzewa. Zatem nie będzie to uporządkowane i zrównoważone drzewo binarne.

W uporządkowanych i zrównoważonych drzewach binarnych bardzo prosto można sprawdzić, czy k-ty węzeł jest liściem. Będzie tak, jeśli węzeł ten nie posiada węzłów potomnych. Zatem, jeśli drzewo binarne składa się z n węzłów, to wystarczy sprawdzić, czy $2k > n$. Jeśli tak, węzeł jest liściem. Jeśli nie, węzeł posiada potomka o indeksie 2k, zatem nie może być liściem.

Ścieżki na drzewach binarnych

Ścieżką nazwiemy ciąg węzłów drzewa binarnego spełniających warunek, iż każdy węzeł poprzedni jest rodzicem węzła następnego. Jeśli ścieżka składa się z k węzłów, to długością ścieżki jest liczba k - 1.



Na powyższym rysunku zaznaczona została ścieżka biegnąca poprzez węzły $\{d[1], d[3], d[6], d[13]\}$. Ścieżka ta zawiera cztery węzły, ma zatem długość równą 3.

Wysokością drzewa binarnego nazwiemy długość najdłuższej ścieżki od korzenia do liścia. W powyższym przykładzie najdłuższa taka ścieżka ma długość 3, zatem zaprezentowane drzewo binarne ma wysokość równą 3.

Dla n węzłów zrównoważone drzewo binarne ma wysokość równą:

$$h = [\log_2 n]$$

Podsumowanie nowej terminologii

węzeł	(ang. node)	- element drzewa binarnego
rodzic, **węzeł nadrzędny**, **przodek**	(ang, parent node)	- węzeł leżący o 1 poziom wyżej w hierarchii
dziecko, **potomek**, **węzeł potomny**	(ang. child node)	- węzeł leżący o 1 poziom niżej w hierarchii, dla którego bieżący węzeł jest rodzicem.
korzeń drzewa	(ang. root node)	- pierwszy węzeł na najwyższym poziomie hierarchii, który nie posiada rodzica
liść, **węzeł terminalny**	(ang. leaf node)	- węzeł nie posiadający węzłów potomnych
ścieżka	(ang. path)	- droga na drzewie binarnym wiodąca poprzez poszczególne wierzchołki

Zadania dla ambitnych

1. Zaprojektuj algorytm, który wyznacza ścieżkę od korzenia drzewa binarnego do wskazanego węzła. Na podstawie algorytmu napisz odpowiedni program w wybranym języku programowania.

2. Wykorzystując zaprojektowany w zadaniu 1 algorytm napisz program, który dla zadanego drzewa binarnego wyznacza wszystkie ścieżki od korzenia do poszczególnych liści.

3. Zaprojektuj algorytm, który sprawdza, czy istnieje ścieżka pomiędzy dwoma węzłami drzewa binarnego. Na podstawie tego algorytmu napisz program w wybranym języku programowania.

Kopiec

Heap

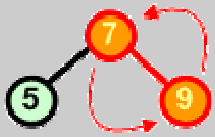
Kopiec jest drzewem binarnym, w którym wszystkie węzły spełniają następujący warunek (zwany warunkiem kopca):

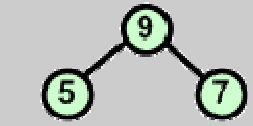
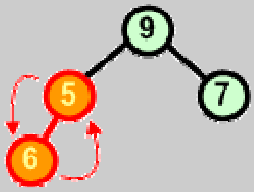
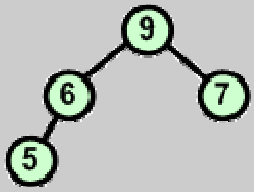
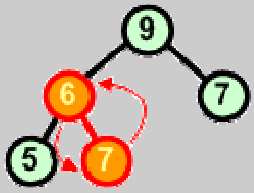
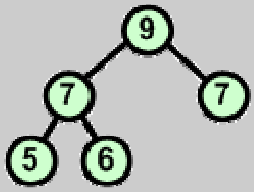
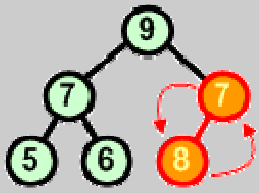
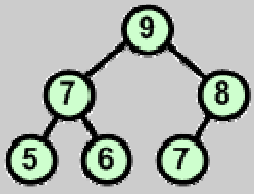
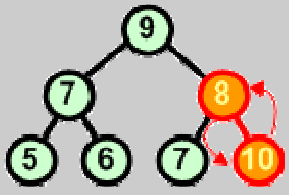
Węzeł nadrzędny jest większy lub równy **węzłom potomnym** (w porządku malejącym relacja jest odwrotna - mniejszy lub równy).

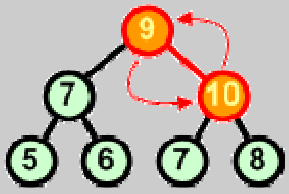
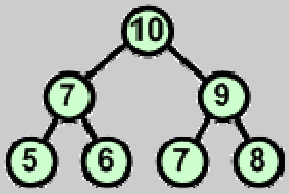
Konstrukcja kopca jest nieco bardziej skomplikowana od konstrukcji drzewa binarnego, ponieważ musimy dodatkowo troszczyć się o zachowanie warunku kopca. Zatem po każdym dołączeniu do kopca nowego węzła, sprawdzamy odpowiednie warunki i ewentualnie dokonujemy wymian węzłów na ścieżce wiodącej od dodanego węzła do korzenia.

Przykład:

Skonstruować kopiec z elementów zbioru $\{7\ 5\ 9\ 6\ 7\ 8\ 10\}$

Operacja	Opis
	Budowę kopca rozpoczynamy od pierwszego elementu zbioru, który staje się korzeniem.
	Do korzenia dołączamy następny element. Warunek kopca jest zachowany.
	Dodajemy kolejny element ze zbioru.
	Po dodaniu elementu 9 warunek kopca przestaje być spełniony. Musimy go przywrócić. W tym celu za nowy węzeł nadrzędny wybieramy nowo dodany węzeł. Poprzedni węzeł nadrzędny wędruje w miejsce węzła dodanego - zamieniamy węzły 7 i 9 miejscami.

	Po wymianie węzłów 7 i 9 warunek kopca jest spełniony.
 7 5 9 **6** 7 8 10	Dołączamy kolejny element 6. Znów warunek kopca przestaje być spełniony - zamieniamy miejscami węzły 5 i 6.
	Po wymianie węzłów 5 i 6 warunek kopca jest spełniony.
 7 5 9 6 **7** 8 10	Dołączamy kolejny element 7. Warunek kopca przestaje obowiązywać. Zamieniamy miejscami węzły 6 i 7.
	Po wymianie węzłów 6 i 7 warunek kopca obowiązuje.
 7 5 9 6 7 **8** 10	Dołączamy kolejny węzeł. Powoduje on naruszenie warunku kopca, zatem wymieniamy ze sobą węzły 7 i 8.
	Po wymianie węzłów 7 i 8 warunek kopca znów obowiązuje.
 7 5 9 6 7 8 **10**	Dołączenie ostatniego elementu znów narusza warunek kopca. Zamieniamy miejscami węzeł 8 z węzłem 10.

	Po wymianie węzłów 8 i 10 warunek kopca został przywrócony na tym poziomie. Jednakże węzeł 10 stał się dzieckiem węzła 9. Na wyższym poziomie drzewa warunek kopca jest naruszony. Aby go przywrócić znów wymieniamy miejscami węzły, tym razem węzeł 9 z węzłem 10.
	Po wymianie tych węzłów warunek kopca obowiązuje w całym drzewie - zadanie jest wykonane.

Charakterystyczną cechą kopca jest to, iż korzeń zawsze jest największym (w porządku malejącym najmniejszym) elementem z całego drzewa binarnego.

Algorytm

Poniżej przedstawiamy algorytm konstrukcji kopca z elementów zbioru.

Specyfikacja problemu

Dane wejściowe

d[ ] - Zbiór zawierający elementy do wstawienia do kopca. Numeracja elementów rozpoczyna się od 1.

n - Ilość elementów w zbiorze, $n \in \mathrm{N}$

Dane wyjściowe

d[ ] - Zbiór zawierający kopiec

Zmienne pomocnicze

i - zmienna licznikowa pętli umieszczającej kolejne elementy zbioru w kopcu, $i \in \mathrm{N}$, $i \in \{2,3,...,n\}$

j,k - indeksy elementów leżących na ścieżce od wstawianego elementu do korzenia, $j,k \in \mathrm{C}$

x - zmienna pomocnicza przechowująca tymczasowo element wstawiany do kopca

Lista kroków

K01: **Dla** *i* = 2, ..., *n*: **wykonuj** K02...K05

K02: *j* ← *i*; *k* ← *j* **div** 2

K03: *x* ← d[*i*]

K04: **Dopóki** (*k* > 0) □ (d[*k*] < *x*): **wykonuj**

d[*j*] ← d[*k*]

j ← *k*

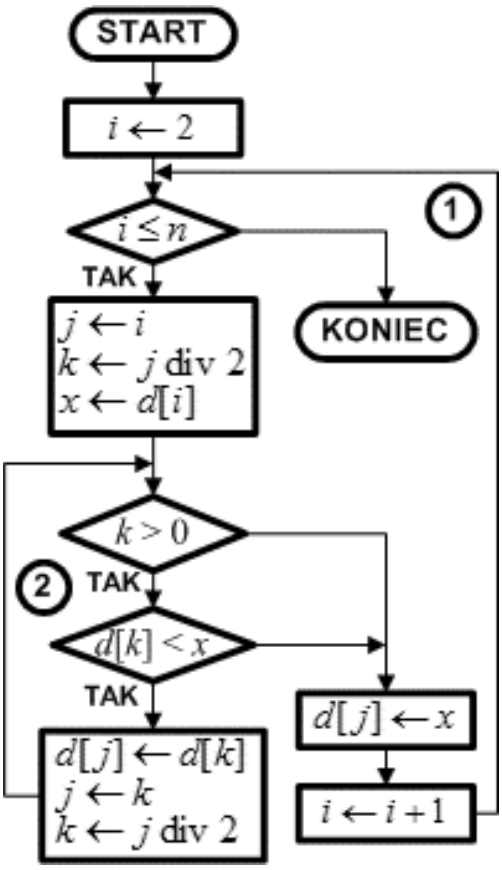
k ← *j* **div** 2

K05: d[*j*] ← *x*

K06: **Zakończ**

Uwaga: przy porządku malejącym należy zmienić drugi warunek w kroku K04 z $d[k] < x$ na $d[k] > x$.

Schemat blokowy



Algorytm tworzy kopiec w tym samym zbiorze wejściowym $d[\ ]$. Nie wymaga zatem dodatkowych struktur danych i ma złożoność pamięciową klasy $O(n)$.

Pętla nr 1 wyznacza kolejne elementy wstawiane do kopca. Pierwszy element pomijamy, ponieważ zostałby i tak na swoim miejscu. Dlatego pętla rozpoczyna wstawianie od elementu nr 2.

Wewnątrz pętli nr 1 inicjujemy kilka zmiennych:

j - pozycja wstawianego elementu (liścia)

k - pozycja elementu nadrzędnego (przodka)

x - zapamiętuje wstawiany element

Następnie rozpoczynamy pętlę warunkową nr 2, której zadaniem jest znalezienie w kopcu miejsca do wstawienia zapamiętanego elementu w zmiennej x. Pętla ta wykonuje się do momentu osiągnięcia korzenia kopca ($k = 0$) lub znalezienia przodka większego od zapamiętanego elementu. Wewnątrz pętli przesuwamy przodka na miejsce potomka, aby zachować warunek kopca, a następnie przesuwamy pozycję j na pozycję zajmowaną wcześniej przez przodka. Pozycja k staje się pozycją nowego przodka i pętla się kontynuuje. Po jej zakończeniu w zmiennej j znajduje się numer pozycji w zbiorze $d[\ ]$, na której należy umieścić element w x.

Po zakończeniu pętli nr 1 w zbiorze zostaje utworzona struktura kopca.

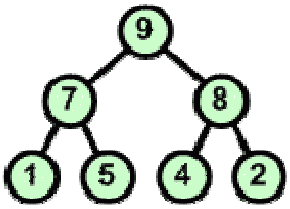
Algorytm

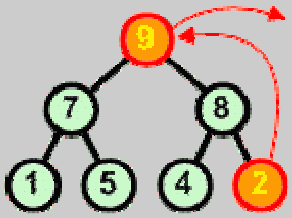
W tym rozdziale nauczymy się rozbierać kopiec. Zasada jest następująca:

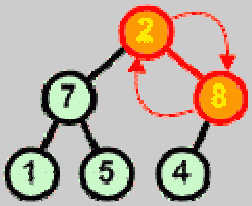
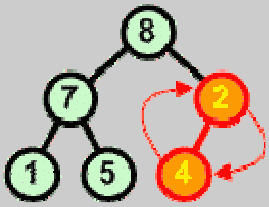
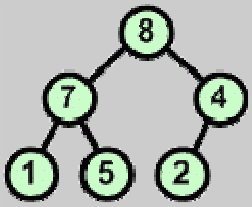
1. Zamień miejscami korzeń z ostatnim liściem, który wyłącz ze struktury kopca. Elementem pobieranym z kopca jest zawsze jego korzeń, czyli element największy.
2. Jeśli jest to konieczne, przywróć warunek kopca idąc od korzenia w dół.
3. Kontynuuj od kroku 1, aż kopiec będzie pusty.

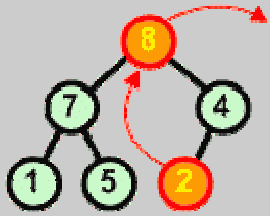
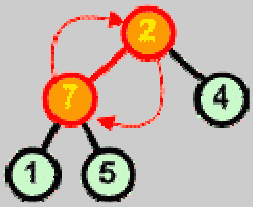
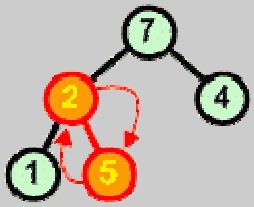
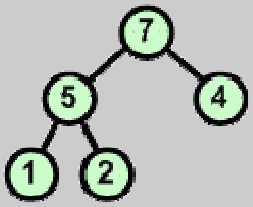
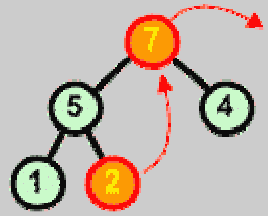
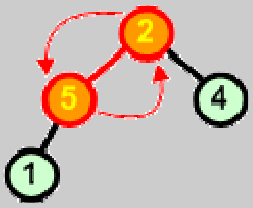
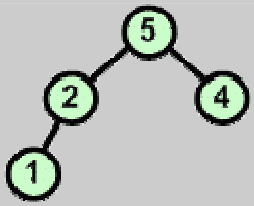
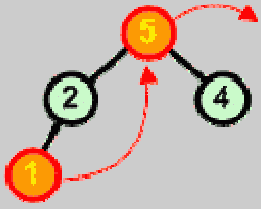
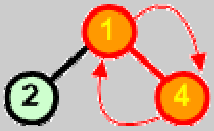
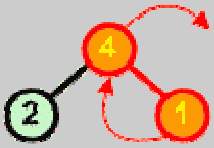
Przykład:

Dokonaj rozbioru następującego kopca:



<table>
<tr><th>Operacja</th><th colspan="3">Opis</th></tr>
<tr><td> 

9</td><td colspan="3">Rozbiór kopca rozpoczynamy od korzenia, który usuwamy ze struktury kopca. W miejsce korzenia wstawiamy ostatni liść.</td></tr>
<tr><td rowspan="2"> 
</td><td> 
</td><td> 
</td><td rowspan="2">Poprzednia operacja zaburzyła strukturę kopca. Idziemy zatem od korzenia w dół struktury przywracając warunek kopca - przodek większy lub równy od swoich potomków. Praktycznie polega to na zamianie przodka z największym potomkiem. Operację kontynuujemy dotąd, aż natrafimy na węzły spełniające warunek kopca.</td></tr>
</table>

 8 9	Usuwamy z kopca kolejny korzeń zastępując go ostatnim liściem		
			W nowym kopcu przywracamy warunek kopca.
 7 8 9	Usuwamy z kopca kolejny korzeń zastępując go ostatnim liściem		
		W nowym kopcu przywracamy warunek kopca. W tym przypadku już po pierwszej wymianie węzłów warunek koca jest zachowany w całej strukturze.	
 5 7 8 9	Usuwamy z kopca kolejny korzeń zastępując go ostatnim liściem		
		Przywracamy warunek kopca w strukturze.	
 4 5 7 8 9	Usuwamy z kopca kolejny korzeń zastępując go ostatnim liściem		
		Przywracamy warunek kopca w strukturze.	

	Usuwamy z kopca kolejny korzeń zastępując go ostatnim liściem.
	Po wykonaniu poprzedniej operacji usunięcia w kopcu pozostał tylko jeden element - usuwamy go. Zwróć uwagę, iż usunięte z kopca elementy tworzą ciąg uporządkowany.

Operację rozbioru kopca można przeprowadzić w miejscu. Jeśli mamy zbiór $d[\ ]$ ze strukturą kopca, to idąc od końca zbioru (ostatnie liście drzewa) w kierunku początku zamieniamy elementy z pierwszym elementem zbioru (korzeń drzewa), a następnie poruszając się od korzenia w dół przywracamy warunek kopca w kolejno napotykanych węzłach.

Specyfikacja problemu

Dane wejściowe

d[ ] - Zbiór zawierający poprawną strukturę kopca. Elementy są numerowane począwszy od 1.

n - Ilość elementów w zbiorze, $n \in \mathrm{N}$

Dane wyjściowe

d[ ] - Zbiór zawierający elementy pobrane z kopca ułożone w porządku rosnącym

Zmienne pomocnicze

i - zmienna licznikowa pętli pobierającej kolejne elementy z kopca, $i \in \mathrm{N}$, $i \in \{n, n-1, ..., 2\}$

j,*k* - indeksy elementów leżących na ścieżce w dół od korzenia, $j, k \in \mathrm{N}$

m - indeks większego z dwóch elementów potomnych, $m \in \mathrm{N}$

Lista kroków

K01: **Dla** $i = n, n - 1, ..., 2$: **wykonuj** K02...K08

K02: d[1] ↔ d[*i*]

K03: $j \leftarrow 1$; $k \leftarrow 2$

K04: **Dopóki** ($k < i$) **wykonuj** K05...K08

K05: **Jeżeli** ($k + 1 < i$) □ (d[$k + 1$] > d[k]), **to** $m \leftarrow k + 1$ **inaczej** $m \leftarrow k$

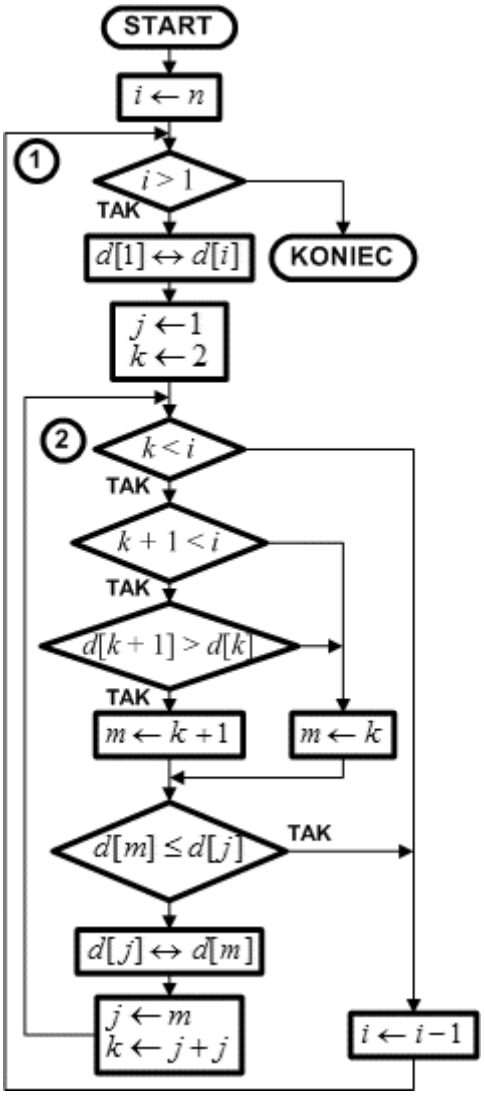
K06: **Jeżeli** d[m] ≤ d[j], **to wyjdź** z pętli K04 i **kontynuuj następny obieg** K01

K07: d[j] ↔ d[m]

K08: $j \leftarrow m$; $k \leftarrow j + j$

K09: **Zakończ**

Schemat blokowy



Rozbiór kopca wykonywany jest w dwóch zagnieżdżonych pętlach. Pętla nr 1 zamienia miejscami kolejne liście ze spodu drzewa z korzeniem. Zadaniem pętli nr 2 jest przywrócenie w strukturze warunku kopca.

Pętla nr 1 przebiega wstecz indeksy elementów zbioru $d[\ ]$ poczynając od indeksu n a kończąc na indeksie 2. Element i-ty jest wymieniany z korzeniem kopca. Po tej wymianie rozpoczynamy w pętli nr 2 przeglądanie drzewa od korzenia w dół. Zmienna j przechowuje indeks przodka, zmienna k przechowuje indeks lewego potomka. Pętla nr 2 kończy się, gdy węzeł j-ty nie posiada potomków. Wewnątrz pętli wyznaczamy w zmiennej m indeks większego z dwóch węzłów potomnych. Następnie sprawdzamy, czy w węźle nadrzędnym j-tym jest zachowany warunek kopca. Jeśli tak, to następuje wyjście z pętli nr 2. W przeciwnym razie zamieniamy miejscami węzeł nadrzędny j-ty z

jego największym potomkiem m-tym i za nowy węzeł nadrzędny przyjmujemy węzeł m-ty. W zmiennej k wyznaczamy indeks jego lewego potomka i kontynuujemy pętlę nr 2.

Po wyjściu z pętli nr 2 zmniejszamy zmienną i o 1 - przenosimy się do kolejnego, ostatniego liścia drzewa i kontynuujemy pętlę nr 1.

W celu wyznaczenia klasy złożoności obliczeniowej algorytmu rozbioru kopca zauważamy, iż pętla zewnętrzna nr 1 wykona się $n - 1$ razy, a w każdym obiegu tej pętli pętla wewnętrzna wykona się maksymalnie $\log_2 n$ razy. Daje to zatem górne oszacowanie $O(n \log n)$ w przypadku pesymistycznym oraz $O(n)$ w przypadku optymistycznym, który wystąpi tylko wtedy, gdy zbiór $d[\ ]$ będzie zbudowany z ciągu tych samych elementów.

Algorytm

Jeśli przeczytałeś uważnie poprzednie dwa rozdziały, to zasadę sortowania przez kopcowanie zrozumiesz od razu - w zbiorze tworzymy kopiec, a następnie dokonujemy jego rozbioru. W wyniku wykonania tych dwóch operacji zbiór zostanie posortowany.

Specyfikacja problemu

Dane wejściowe

d[ ] - Zbiór zawierający elementy do posortowania, które są numerowane począwszy od 1.

n - Ilość elementów w zbiorze, $n \in \mathrm{N}$

Dane wyjściowe

d[ ] - Zbiór zawierający elementy posortowane rosnąco

Zmienne pomocnicze i funkcje

Twórz_kopiec - procedura budująca kopiec z elementów zbioru d[ ]. Kopiec powstaje w zbiorze d[ ].

Rozbierz_kopiec - procedura dokonująca rozbioru kopca utworzonego w zbiorze d[ ]. Wynik rozbioru trafia z powrotem do zbioru d[ ].

Lista kroków

K01: **Twórz_kopiec**

K02: **Rozbierz_kopiec**

K03: Zakończ algorytm

Ponieważ sortowanie przez kopcowanie składa się z dwóch następujących bezpośrednio po sobie operacji o klasie czasowej złożoności obliczeniowej $O(n \log n)$, to dla całego algorytmu klasa złożoności również będzie wynosić $O(n \log n)$.