Wrocław
University
of Science
and Technology

Struktury danych

Wykład 5

Drzewa, kopce

dr inż. Jarosław Rudy

HR EXCELLENCE IN RESEARCH

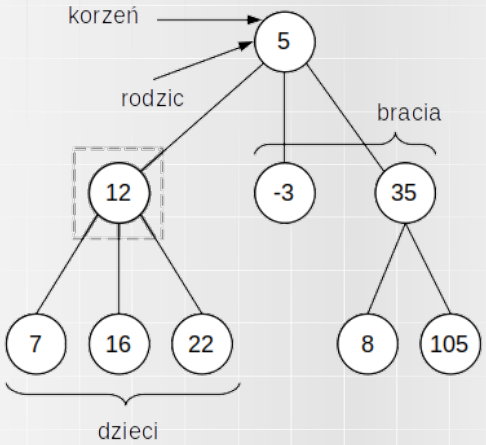
Drzewa (1)

- Matematycznie drzewo jest zbiorem wierzchołków (węzłów) oraz łączących je krawędzi.
- Intuicyjnie, drzewo możemy podzielić na poziomy.
- Na zerowym poziomie jest tylko jeden wierzchołek (zwany korzeniem drzewa) i ma on połączenia jedynie z wierzchołkami na poziomie pierwszym.
- Wierzchołki na pozostałych poziomach:
  - Zawsze mają dokładnie jedno połączenie z jednym z wierzchołków na poziomie poprzednim. Wierzchołek ten nazywany jest rodzicem.
  - Mają dowolną (0 lub więcej) liczbę połączeń z wierzchołkami na poziomie kolejnym. Wierzchołki te nazywa się synami lub dziećmi (rzadziej potomkami).

Drzewa (2)

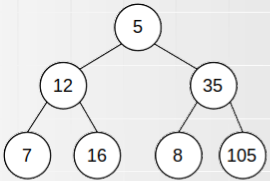
- Analogicznie można określić dla danego węzła dziadka, pradziadka itd. (ogólnie poprzedników) oraz wnuków, prawnuków itd. (ogólniej następników lub potomków).
- Wierzchołki mającego tego samego rodzica nazywamy rodzeństwem lub braćmi itp.
- Wierzchołki bez dzieci nazywają się liścmi.
- Dużo łatwiej i ściślej można zdefiniować drzewo używając pojęcia grafu – drzewo jest grafem, który jest jednocześnie nieskierowany, spójny i acykliczny.
- Tradycyjnie drzewa przedstawiane są z korzeniem na górze.
- Określenie korzenia jest kwestią wyboru – każdy wierzchołek w drzewie może być korzeniem, zaś jego wybór określa relacje pomiędzy wierzchołkami.

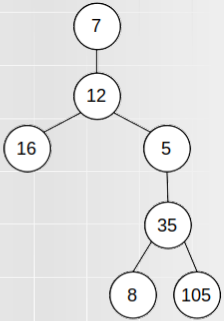
Drzewa (3)

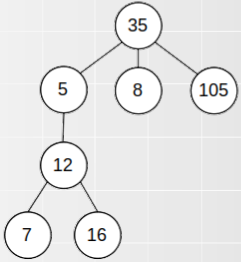


Wierzchołki 7, 16, 22, -3, 8 i 105 są liścmi.

Drzewa (4)







Drzewa (5)

Drzewo jest ADT czy strukturą danych?

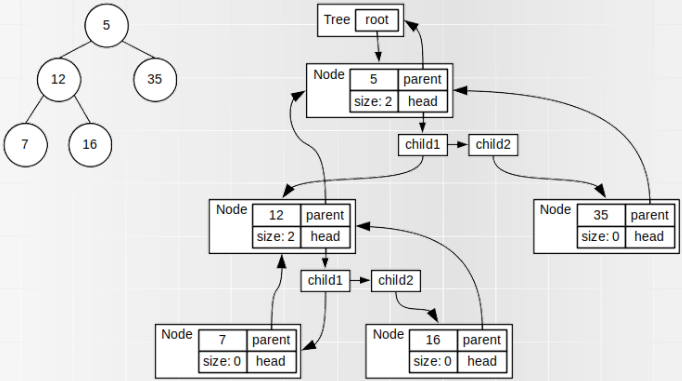
- Powszechnie drzewo uznawane jest za strukturę danych.
  - Dzieje się tak pomimo faktu, że definicja drzewa nie określa sposobu rozmieszczenia danych w pamięci ani złożoności operacji.
- Z matematycznego i informatycznego punktu widzenia na drzewie można określić operacje typu dodanie/usunięcie węzła, wyszukanie węzła, rotacja węzła, przeszukanie drzewa (wszerz, wgłąb itp.).
  - Takie ujęcie może wskazywać na charakter ADT.

Drzewa (6)

- W praktyce istotny jest kontekst.
  - Jeśli charakterystyka drzewa wynika wprost z zagadnienia (np. genealogia, ciąg decyzyjny), to może być bliżej ADT.
  - Jeśli charakterystyka drzewa wynika z wyboru (np. implementacja słownika), to może być bliżej strukturze danych.
- Drzewo często stoi pomiędzy pierwotnym ADT a końcową strukturą danych:
  - Dodaje dodatkowe założenia względem np. słownika.
  - Musi być zaimplementowane w konkretnym sposób.

Drzewa (7)

Drzewo może oznaczać konkretną strukturę np. z użyciem list wiązanych:



Dla wielu drzew (statyczne, zupełne binarne itd.) istnieją dużo lepsze rozwiązania. W szczególności można użyć reprezentacji grafowej.

Drzewa (8)

- Ścieżka (czasem też droga) – ciąg krawędzi łączący dwa wierzchołki (bez powtarzania się krawędzi).
  - W drzewie zawsze istnieje jedna ścieżka od korzenia do danego wierzchołka.
  - Liczba krawędzi na ścieżce nazywana jest jej długością.
- Dla danego wierzchołka A jego poziom to długość ścieżki od korzenia do A.
- Wysokość drzewa to największy z jego poziomów.

Drzewa (10)

Typowe operacje wykonywane na drzewie:

- Dodanie elementu.
- Usunięcie elementu.
- Wyszukanie elementu.
- Usunięcie poddrzewa.
- Rotacja poddrzewa.
- Przeglądnięcie drzewa (odwiedzenie wszystkich wierzchołków w pewnej kolejności).

Drzewa (11)

- Drzewo może być puste – brak wierzchołków, a tym samym brak korzenia.
- W drzewie można rozpatrywać poddrzewa – pewien wierzchołek A wraz ze wszystkimi lub częścią jego potomków. Wtedy A jest korzeniem poddrzewa.
- Zbiór (rozłącznych) drzew nazywany jest lasem.
- Istnieje wiele szczególnych przypadków drzew (drzewa binarne, BST, czerwono-czarne, ósemkowe itd).

Drzewa (12)

Zastosowanie drzew:

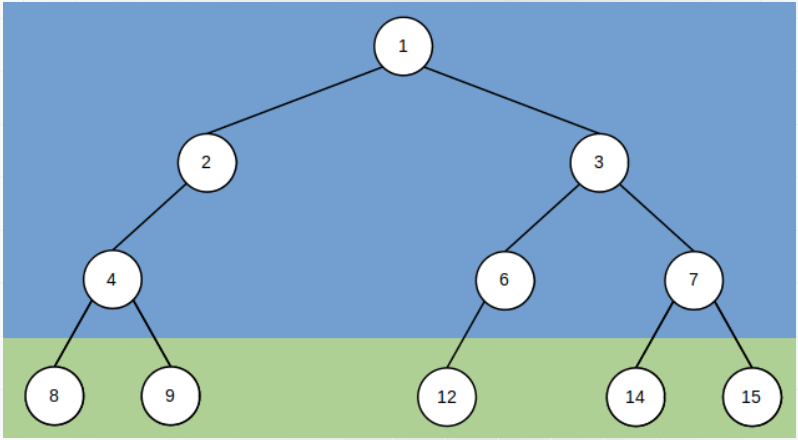
- Modelowanie hierarchii (systemy plików, dokumenty HTML/XML, dziedziczenie klas).
- Procesy decyzyjne (giełda, problem plecakowy, komiwojażera itd.).
- Rozkład gramatyczny zdań (informatyka i lingwistyka).
- Implementacja słowników.
- Implementacja kolejek priorytetowych.

Drzewa binarne (1)

- Drzewo binarne – drzewo, w którym każdy węzeł ma co najwyżej dwoje dzieci.
- Drzewo binarne pełne (full) – drzewo binarne, w którym 1) liście są tylko na ostatnim poziomie, 2) wszystkie nie-liście mają dokładnie 2 dzieci.
- Drzewo binarne zupełne (complete) – drzewo binarne, w którym 1) ostatni poziom wypełniany jest „od lewej", 2) wszystkie pozostałe węzły mają dokładnie 2 dzieci (tzn. poziomy oprócz ostatniego są drzewem pełnym).

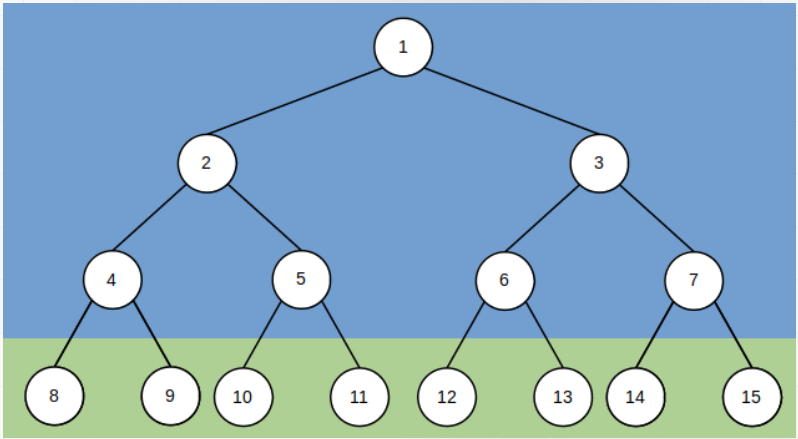
Drzewa binarne (2)

Drzewo binarne



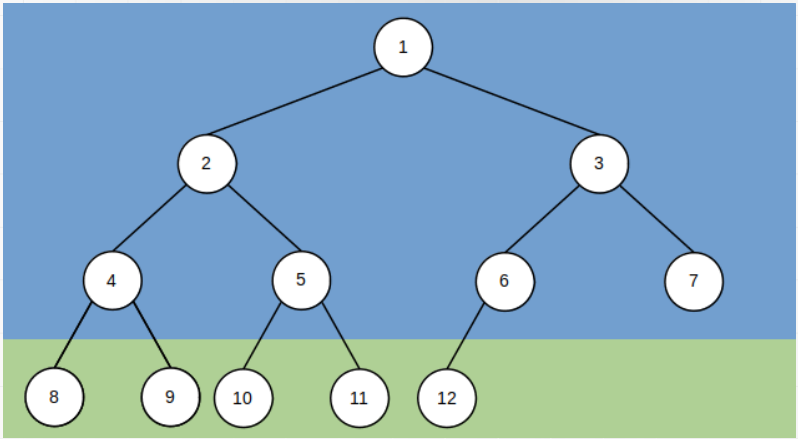
Drzewa binarne (3)

Drzewo binarne pełne



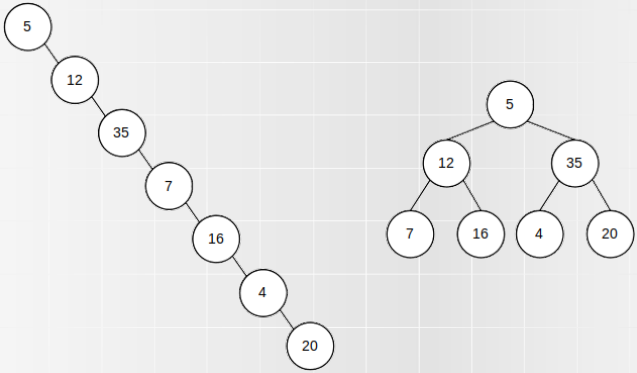
Drzewa binarne (4)

Drzewo binarne zupełne



Drzewa binarne (5)

Wysokość drzewa binarnego o n elementach wynosi $O(n)$, ale dla drzewa binarnego zupełnego już tylko $O(\log n)$!



Przejście przez drzewo (1)

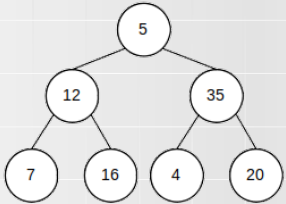
- Przejście przez drzewo oznacza odwiedzenie każdego węzła (zwykle w pewnej kolejności), często wykonując na każdym odwiedzanym węźle pewną czynność (np. wydrukowanie wartości).
- Istnieje kilka ważnych sposobów przejścia przez drzewo:
  - Breadth-first (wszerz) – rodzeństwo węzła jest odwiedzane przed jego dziećmi. Odpowiada użyciu kolejki FIFO.
  - Depth-first (wgłąb) – dzieci węzła są odwiedzane przed jego rodzeństwem. Odpowiada użyciu kolejki LIFO (stosu).
  - Best-fist (najpierw najlepszy) – odwiedzamy węzły wg priorytetu. Odpowiada użyciu kolejki priorytetowej.

Przejście przez drzewo (2)

Dla przejścia w głąb można rozważyć kilka wariantów:

- Pre-order – rodzic jest odwiedzany przed swoimi dziećmi.
- Post-order – rodzic jest odwiedzany po swoich dzieciach.
- In-order – najpierw odwiedzany jest lewy syn, potem rodzic, potem prawy syn (dotyczy drzew binarnych).

Przejście przez drzewo (3)



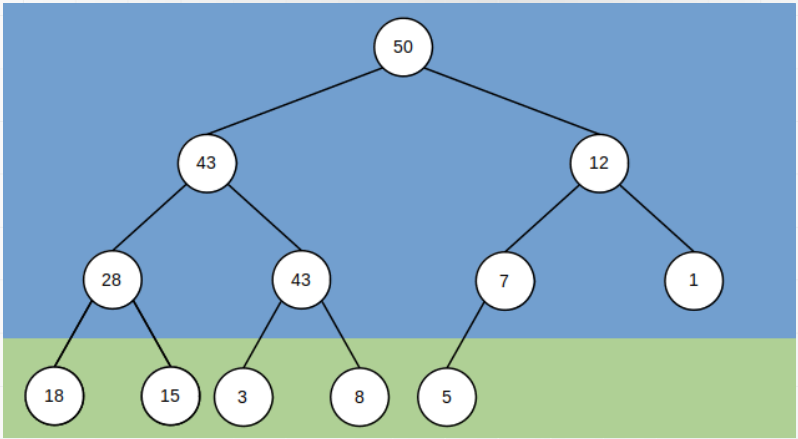
- Breadth-first: 5, 12, 35, 7, 16, 4, 20.
- Depth-first:
  - Pre-order: 5, 12, 7, 16, 35, 4, 20.
  - Post-order: 7, 16, 12, 4, 20, 35, 5.
  - In-order: 7, 12, 16, 5, 4, 35, 20.
- Best-first: 5, 35, 20, 12, 16, 7, 4.

Kopce (1)

- Kopiec (heap) – drzewo w którym zachowana jest zasada kopca tzn. klucz rodzica jest w stałej relacji z kluczami jego dzieci. Najczęściej rozróżniamy:
  - Max heap: rodzic jest nie mniejszy od swoich dzieci.
  - Min heap: rodzic jest nie większy od swoich dzieci.
- W efekcie każda ścieżka od korzenia do liścia jest posortowana (kopiec jest częściowo posortowany).
- Jeśli węzły nie mają klucza, to kluczem staje się sama wartość.
- Zbiór kluczy musi mieć określony częściowy porządek.
- Kopce również mogą być binarne, pełne i zupełne.

Kopce (2)

Kopiec binarny zupełny typu max



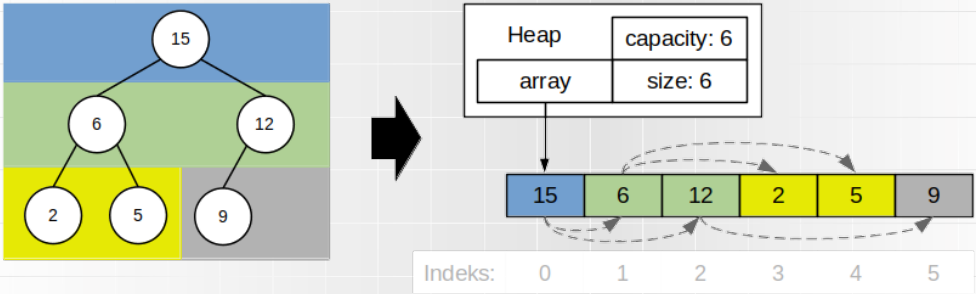
Kopce (3)

Kopiec binarny zupełny można wydajnie i prosto zaimplementować z użyciem tablicy dynamicznej:

- Przyjmując numerację od 0, jeśli rodzic ma indeks k w tablicy, to jego dzieci mają indeksy $2k + 1$ oraz $2k + 2$.
- Jeśli węzeł ma indeks k, to jego rodzic ma indeks $\lfloor \frac{k-1}{2} \rfloor$ (arytmetyka całkowitoliczbowa).
- Ponieważ kopiec jest zupełny (ostatni poziom wypełniany od lewej do prawej), to tablica wykorzystawana jest w pełni bez dziur (nie licząc przestrzeni zaalokowanej na przyszłość).
- Znając indeks, dostęp do węzła jest w czasie $O(1)$.

Kopce (4)

Przykład implementacji (binarnego zupełnego) kopca za pomocą tablicy dynamicznej:



Kopce (5)

Operacje kopcowe (kopiec typu max):

- insert() – dodanie elementu.
- extract-max() – usunięcie i zwrócenie elementu o największym kluczu.
- find-max() – zwrócenie elementu o największym kluczu.
- find() – szukanie elementu o danej wartości/kluczu.
- delete() – usunięcie elementu o danej wartości.
- decrease-key()/increase-key() – zmiana wartości klucza elementu.
- build() – zbudowanie kopca z n elementów.

Kopce (6)

Pomocnicze operacje do przywracania własności kopca.

- heapify-up
  1. Zaczynamy od danego elementu e.
  2. Jeśli e jest korzeniem lub jest w prawidłowej relacji ze swoim rodzicem – koniec algorytmu.
  3. W przeciwnym razie (złamana własność kopca) zamień e miejscami ze swoim rodzicem i wróć do kroku 2.
- Pesymistyczny czas zależny od wysokości drzewa, czyli $O(\log n)$.

Kopce (7)

- heapify-down
  1. Zaczynamy od danego elementu e.
  2. Jeśli e jest liściem lub jest w prawidłowej relacji ze swoimi dziećmi – koniec algorytmu.
  3. W przeciwnym razie (złamana własność kopca) zamień e miejscami ze swoim większym dzieckiem i wróć do kroku 2.
- Dla kopca typu min analogicznie (zamieniamy z mniejszym dzieckiem).
- Pesymistyczny czas $O(\log n)$.

Kopce (8)

- insert()
  - Wstawiamy element na pierwsze wolne miejsce (na ostatnim poziomie lub tworzymy nowy poziom od lewej).
  - Wykonujemy heapify-up() na dodanym węźle.
  - Czas $O(\log n)$ (zamortyzowany dla tablicy dynamicznej).
- extract-max()
  - Wstawiamy ostatni element w miejsce korzenia (zapamiętując stary korzeń).
  - Wykonujemy heapify-down() na nowym korzeniu.
  - Zwracamy zapamiętany stary korzeń.
  - Czas $O(\log n)$.

Kopce (9)

- find-max() – zwracamy korzeń, czas $O(1)$.
- find() – przeszukanie (przejście) drzewa, pesymistycznie $O(n)$.
- delete()
  - Znalezienie węzła e do usunięcia.
  - Zamienienie węzła miejscami z ostatnim.
  - Zmniejszenie rozmiaru tablicy.
  - Wykonanie heapify-up() lub heapify-down() do przywrócenia własności kopca (jeśli potrzebne).
  - Pesymistyczny czas $O(n)$.

Kopce (10)

- increase-key()
  - Znajdź węzeł e do zmiany.
  - Zwiększ klucz.
  - Wykonaj heapify-up() na e (zakładając kopiec typu max).
- decrease-key()
  - Znajdź węzeł e do zmiany.
  - Zmniejsz klucz.
  - Wykonaj heapify-down() na e (zakładając kopiec typu max).

Kopce (11)

- Jedna z operacji dla wielu struktur danych jest wypełnienie pustej struktury (dodanie po kolei wielu elementów).
- Dla n elementów zwykle polega to na n-krotnym wykonaniu operacji typu insert().
- Dla kopca (binarnego zupełnego) nazywamy to budowaniem kopca.
- Naiwne podejście oznacza czas $n \cdot O(\log n) \in O(n \log n)$.
- Istnieje jednak szybszy sposób.

Kopce (12)

- Najpierw dodajemy wszystkie elementy w dowolny sposób, nie dbając o własność kopca.
  - Najprościej jest zaalokować miejsce na n elementów i umieszczać element i-ty w indeksie i.
- Następnie w pętli dla indeksów od $\lfloor \frac{n-1}{2} \rfloor$ do 0 wykonujemy heapify-down (zakładając kopiec typu max).
  - Procedura działa od dołu w górę (bottom-up).
  - Elementy na indeksach dalej niż $\lfloor \frac{n-1}{2} \rfloor$ są liśćmi – kopcowanie ich nie ma sensu.
- Z pomocą pewnego zbieżnego szeregu można wykazać, że taka operacja zajmuje pesymistycznie i średnio czas $O(n)$.

Kopce (13)

Zasosowanie kopców:

- Sortowanie przez kopcowanie.
- Algorytmy selekcji.
- Kolejki priorytetowe.
- Algorytmy grafowe.
  - Algorytm Dijkstry.
  - Algorytm Prima.

Kopce (14)

Kopce umożliwiają wydajną implemenację kolejek priorytetowych:

- insert() – $O(\log n)$.
- extract-max() – $O(\log n)$.
- find-max() – $O(1)$.
- modify-key() – wciąż koszt $O(n)$ (konieczność znalezienia węzła).
  - Możemy dodatkowo przechowywać słownik mapujący indeks na węzeł.
  - Koszt operacji spada do $O(\log n)$, ale stała wzrasta.
  - Słownik oznacza konieczność dodatkowej pamięci.