Wstęp do programowania

Stosy, kolejki, drzewa

Paweł Daniluk

Wydział Fizyki

Jesień 2014

Listy

Lista jest uporządkowanym zbiorem elementów. W Pythonie występuje typ sekwencyjny `list`, który umożliwia wstawianie elementów na dowolną pozycję, usuwanie itd.

Każda lista ma dwa końce.

LIFO

Last In First Out – stos

wstawianie (*push*)

```
stos.append(el)
```

pobieranie z usuwaniem (*pop*)

```
stos.pop()
```

FIFO

First In First Out – kolejka

wstawianie (*enqueue*)

```
kolejka.append(el)
```

pobieranie z usuwaniem (*dequeue*)

```
kolejka.pop(0)
```

Stos – przykład

Odwrotna Notacja Polska (notacja postfiksowa)

Forma zapisu wyrażeń arytmetycznych, w której operator następuje po argumentach (a nie pomiędzy). Wygodna bo nie wymaga nawiasów.

`2 2 2 + *` zamiast `2 * (2 + 2)`.

Stos – przykład c.d.

Sposób obliczania 3 2 1 + *

1. 3
2. 3 2
3. 3 2 1
4. 3 2 1 +
5. 3 3
6. 3 3 *
7. 9

ONP w Pythonie

```
def add(a,b): return a+b
def sub(a,b): return a-b
def mul(a,b): return a*b
def div(a,b): return a/b

def compute_onp(l):
    ops={'+': add, '-': sub, '*': mul, '/':div}

    stack=[]

    for symbol in l:
        if symbol in ops:
            a=stack.pop()
            b=stack.pop()
            stack.append(ops[symbol](a,b))
        else:
            stack.append(symbol)

    if len(stack)!=1:
        print "Error: ", stack

    return stack[0]
```

Typowe zastosowania stosów i kolejek

Zastosowania stosów

- analiza składniowa
- eksploracja ze ścieżką powrotu
- obchodzenie drzew i grafów (DFS)
- strategia dziel i zwyciężaj

Zastosowania kolejek

- kolejki zadań
- buforowanie danych
- obchodzenie drzew i grafów (BFS)

Kolejki priorytetowe

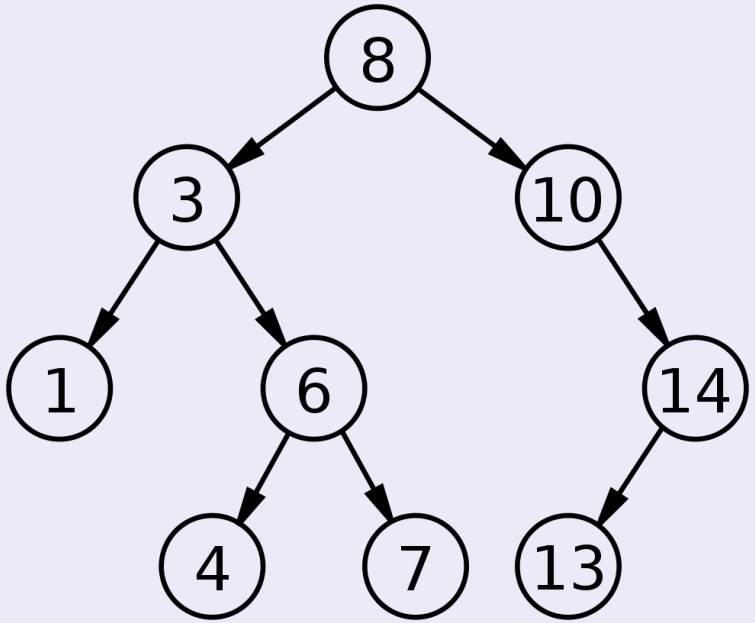
Szczególnym przypadkiem kolejek są kolejki priorytetowe, w których operacja *dequeue* pobiera element o najwyższym priorytecie, który wcale nie musi być najdłużej oczekującym.

Drzewa binarne

Drzewo binarne jest strukturą danych, w której każdy węzeł ma co najwyżej dwa węzły potomne, każdy węzeł z wyjątkiem korzenia ma rodzica oraz żaden węzeł nie może być swoim przodkiem.

Ważne pojęcia

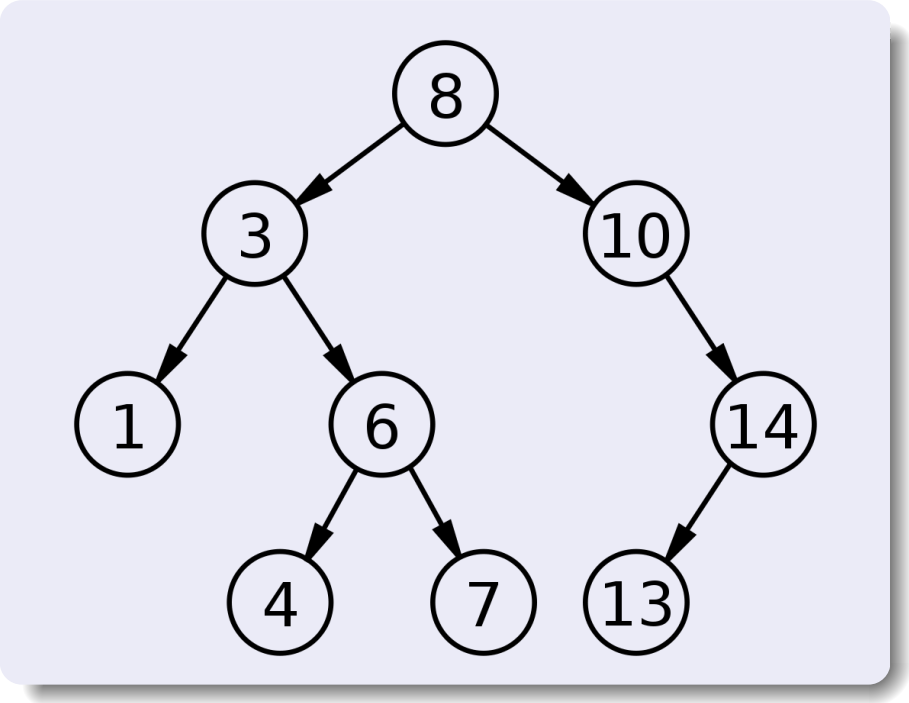
- korzeń
- poddrzewo (lewe, prawe)
- potomek
- rodzic
- przodek



Naiwna implementacja w Pythonie

Na potrzeby tego wykładu posłużymy się implementacją drzew binarnych przy pomocy krotek.

```
(8,
  (3,
    (1, None, None),
    (6,
      (4, None, None),
      (7, None, None)
    )
  ),
  (10,
    None,
    (14,
      (13, None, None),
      None
    )
  )
)
```

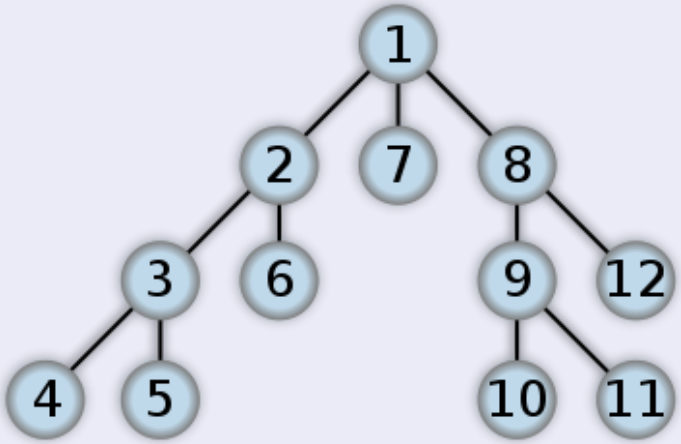


Obchodzenie drzewa

Większość algorytmów operujących na drzewach wymaga sprawdzenia zawartości wierzchołków. Można to robić na wiele sposobów. Wyróżnia się dwa podstawowe: Depth First Search (DFS) i Breadth First Search (BFS).

DFS

Polega na schodzeniu wgłąb. Po odwiedzeniu wierzchołka, odwiedza się wszystkie wierzchołki w lewym poddrzewie, następnie wszystkie wierzchołki w prawym poddrzewie.

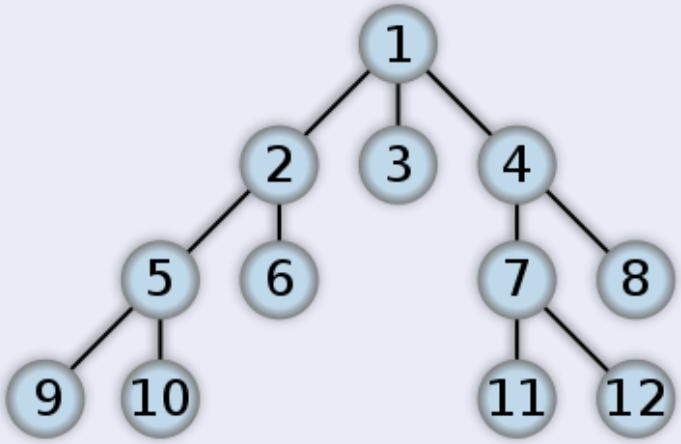


Obchodzenie drzewa

Większość algorytmów operujących na drzewach wymaga sprawdzenia zawartości wierzchołków. Można to robić na wiele sposobów. Wyróżnia się dwa podstawowe: Depth First Search (DFS) i Breadth First Search (BFS).

BFS

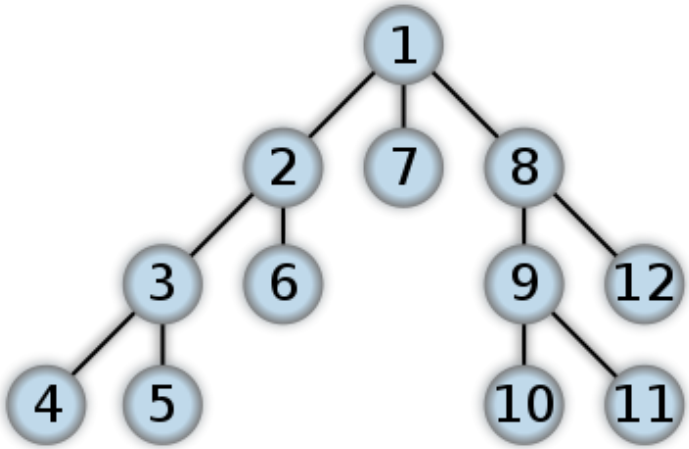
Polega na odwiedzaniu wierzchołków według odległości od korzenia.



Obchodzenie drzewa – DFS

Stos

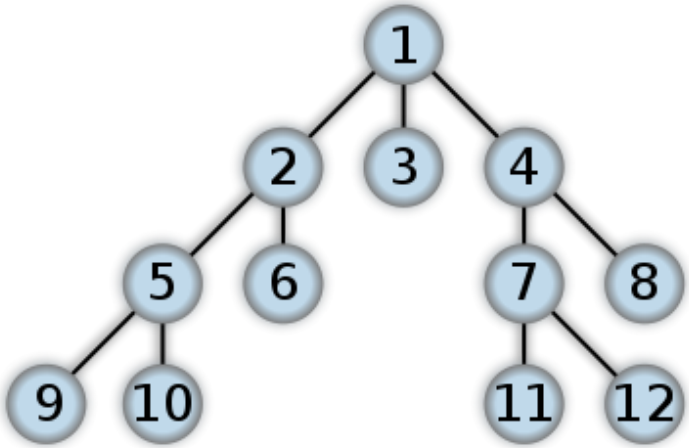
0. 1
1. 8 7 2
2. 8 7 6 3
3. 8 7 6 5 4
4. 8 7 6 5
5. 8 7 6
6. 8 7
7. 8
8. 12 9
9. 12 11 10
10. 12 11
11. 12
12.



Obchodzenie drzewa – BFS

Kolejka

0	1
1	2 3 4
2	3 4 5 6
3	4 5 6
4	5 6 7 8
5	6 7 8 9 10
6	7 8 9 10
7	8 9 10 11 12
8	9 10 11 12
9	10 11 12
10	11 12
11	12
12	



DFS – implementacja w Pythonie

```
def tree_dfs(tree):
    if tree==None:
        return

    stack=[tree]

    while len(stack)>0:
        t=stack.pop()
        print val(t)

        if right(t)!=None: stack.append(right(t))
        if left(t)!=None: stack.append(left(t))
```

DFS – implementacja rekurencyjna

```
def tree_dfs(tree):
    if tree==None:
        return

    print val(t)

    tree_dfs(left(tree))
    tree_dfs(right(tree))
```

Gdzie tutaj jest stos?

DFS – implementacja rekurencyjna

```
def tree_dfs(tree):
    if tree==None:
        return

    print val(t)

    tree_dfs(left(tree))
    tree_dfs(right(tree))
```

Gdzie tutaj jest stos?

Podczas wykonywania programu jest utrzymywany stos (call stack), który utrzymuje aktualnie wykonywane funkcje w kolejności wywołania.

Kopce

Kopce są szczególnym przypadkiem drzew, w których spełniony jest warunek, że element w korzeniu drzewa (i każdego poddrzewa) jest największym elementem w tym drzewie (poddrzewie).

Włożenie nowego elementu do kopca może wymagać jego reorganizacji, ale za to wyjmowanie elementu nawiększego jest łatwe.

Kopce dobrze nadają się do implementowania kolejek priorytetowych.

Sortowanie przez kopcowanie – Heapsort

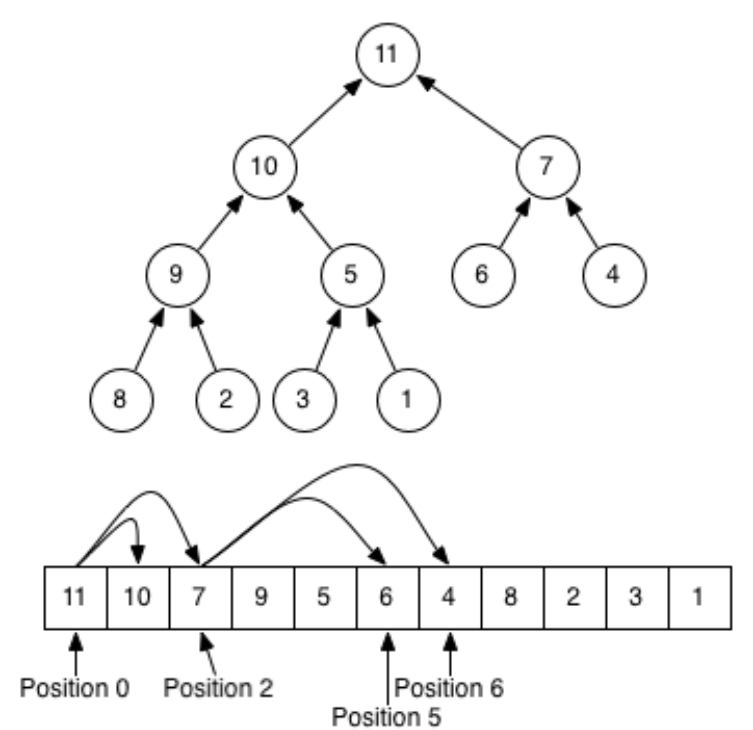
1. Włóż wszystkie elementy do kopca.
2. Wyjmuj elementy ze szczytu kopca za każdym razem odtwarzając kopiec.

http://upload.wikimedia.org/wikipedia/commons/f/fe/Heap_sort_example.gif

Sortowanie przez kopcowanie – Heapsort (aspekty praktyczne)

Dane przechowywane są w tablicy.

left(i)	i*2+1
right(i)	i*2+2
parent(i)	(i-1)/2



Sortowanie przez kopcowanie – Heapsort (algorytm)

1. Zrób kopiec
   1. Dla każdego elementu w tablicy począwszy od pierwszego przenoś w górę kopca na właściwe miejsce
2. Posortuj
   1. Zamień pierwszy element tablicy z ostatnim nieposortowanym
   2. Odtwórz warunek kopca

Kilka słów o modułach

Niewygodnie jest umieszczać wszystkie funkcje i inne elementy dużego programu w jednym pliku. Jeżeli zachodzi potrzeba skorzystania z funkcji, które znajdują się w innym pliku, należy użyć instrukcji `import`.

import

import nazwa Ładuje zawartość pliku `nazwa.py`. Elementy modułu są dostępne poprzez `nazwa.el`.

from nazwa import f Ładuje funkcję `f` z pliku `nazwa.py`. Funkcja `f` jest dostępna bezpośrednio. Ta konstrukcja jest niezalecana.

reload(nazwa) Przeładowuje ponownie moduł `nazwa`.

Kilka słów o modułach

test.py

```
#!/usr/bin/python

def f():
    print "Funkcja␣f."
```

Przykład

```
>>> import test
>>> test.f()
Funkcja f.
>>> from test import f
>>> f()
Funkcja f.
```

Zadanie 1 – wyrażenia nawiasowe

Napisz program, który wczytuje ze standardowego wejścia linię tekstu zawierającą wyrażenie nawiasowe (ciąg nawiasów okrągłych) i rozstrzyga, czy jest ono poprawne.

Przykład wyrażenia poprawnego

```
(()()(()()))
```

Przykład wyrażenia niepoprawnego

```
(()
```

Zadanie 2 – obchodzenie drzew

Zaimplementuj obchodzenie drzewa binarnego (w reprezentacji krotkowej) metodami DFS i BFS.

Zadanie 3 – sortowanie przez kopcowanie

Zaimplementuj algorytm Heapsort.

Zadanie 4 – pasjans

Napisz program, który generuje losową sekwencję kart i sprawdza, czy pasjans o poniższych zasadach daje się ułożyć.

1. Karty rozkładane są na przemian na dwie "drabinki".
2. Jeżeli na którejś z drabinek pojawią się leżące na sobie karty tego samego rodzaju (różniące się kolorem) są one usuwane z gry. Karta z sąsiedniej drabinki jest przenoszona, aby wyrównać wysokość.
3. Pasjans jest udany jeżeli wszystkie karty zostaną usunięte z gry.

Strona wykładu

http://bioexploratorium.pl/wiki/Wstęp_do_programowania_
_2014z