МРНТИ 20.01.45
УДК 004.421.2

DOI: https://doi.org/10.51889/2021-1.1728-7901.20

А.Р. Байдалина[1], С.А. Боранбаев[2]*

[1]С. Сейфуллин атындағы Қазақ агротехникалық университеті, Нұр-Сұлтан қ., Қазақстан
[2]Л.Н. Гумилев атындағы Еуразия ұлттық университеті, Нұр-Сұлтан қ., Қазақстан
**e-mail: boranbaevsa@mail.ru*

PYTHON ТІЛІНДЕ ДЕРЕКТЕР ҚҰРЫЛЫМДАРЫНЫҢ АЛГОРИТМДЕРІН ПРОГРАММАЛАУ

Аңдатпа

Мақалада күрделі деректер құрылымдарына қатысты алгоритмдерді Python тілінде программалау жолдары қарастырылған. Бұл құрылымдар мен оларға тиісті алгоритмдерді білу әртүрлі программалық жабдықтарды құрудың тиімді жолдарын таңдау үшін қажет. «Алгоритмдер және деректер құрылымы» пәнін оқытуда программалаудан бұрын құрылымдарды түсіну маңызды. Себебі, қарастырылып отырылған есепке сәйкес құрылымдарды қолдану барысында осы құрылымдардың мағынасы мен алгоритмдерін жақсы түсіну керек.

Қазіргі уақытта кең қолданыс тауып келе жатқан Python тілінде стандартында құрылым ұғымы жоқ. Мақалада бір жақты байланысқан динамикалық тізбектерге және екілік іздеу ағаштарына қатысты алгоритмдерді Python тілінде программалау мысалдары көрсетілген. Графтарды тереңінен және көлденеңінен қарастыру алгоритмдері Python тіліндегі сөздік арқылы тиімді әрі көрнекі түрде жүзеге асырылды.

Түйін сөздер: Python тілі, бір жақты байланысқан тізбек, екілік іздеу ағашы, **г**рафтарды тереңінен қарастыру, графтарды көлденеңінен қарастыру, Python тіліндегі сөздіктер.

Аннотация

А.Р. Байдалина[1], С.А. Боранбаев[2]
[1]Казахский агротехнический университет имени. С. Сейфуллина, г. Нур-Султан, Казахстан
[2]Евразийский национальный университет имени Л.Н. Гумилева, г. Нур-Султан, Казахстан

ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ СТРУКТУР ДАННЫХ НА ЯЗЫКЕ PYTHON

В статье рассматривается способы программирования алгоритмов сложных структур данных на языке Python. Знание этих структур и соответствующих алгоритмов нужно при выборе оптимальных способов при разработке различного программного обеспечения. При изучении предмета «Алгоритмы и структуры данных» важное значение имеет понимание сути структур данных. Связано это с тем, что манипулирование структурой данных, чтобы она подходила к определённой проблеме, требует понимания сути и алгоритмов этой структуры данных.

Приведены примеры программирования алгоритмов, относящихся динамическим спискам и двоичным деревьям поиска на широко применяемым в настоящее время языке Python. Алгоритмы обхода графа в глубину и в ширину оптимально и наглядно реализованы с использованием словаря языка Python.

Ключевые слова: язык Python, одно связанный список, двоичные деревья поиска, обход графа в глубину, обход графа в ширину, словарь языка Python.

Abstract

PROGRAMMING DATA STRUCTURE ALGORITHMS IN PYTHON

Baidalina A.R.[1], Boranbayev S.A.[2]
[1]S.Seifullin Kazakh Agrotechnical University, Nur-Sultan, Kazakhstan
[2]L.N. Gumilyov Eurasian National University, Nur-Sultan, Kazakhstan

The article discusses ways of programming algorithms for complex data structures in Python. Knowledge of these structures and the corresponding algorithms is necessary when choosing the best methods for developing various software. When studying the subject "Algorithms and Data Structures", it is important to understand the essence of data structures. This is due to the fact that manipulating a data structure to fit a specific problem requires an understanding of the essence and algorithms of this data structure.

Examples of programming algorithms related to dynamic lists and binary search trees in the currently widely used Python language are given. The algorithms for traversing the graph in depth and breadth are optimally and clearly implemented using the Python dictionary.

Keywords: Python, single linked list, binary search trees, Depth first graph traversal, Breadth first graph traversal, Python dictionary.

Кіріспе

Қазіргі уақытта Python тілін қолданылу аясы жылдан жылға кеңею үстінде. Оның негізгі себебі қарапайым қолданбалы есептерді Python тілінде программалау, басқа жоғарғы деңгейдегі алгоритмдік тілдерге қарағанда оңай жүзеге асырылады. C++, Java, Паскаль және т.б. алгоритмдік программалау тілдеріндегі сияқты, программа құрушы адамға есептеу техникасында деректердің өрнектелуі мен орналасу мүмкіндіктерін білуді Python тілі қажет етпейді.

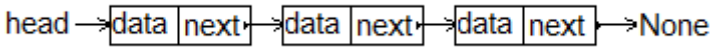
Көптеген оқу орындарында программалау пәндерінде Python тілі жиі қолданыла бастады. Дербес компьютерде программалауды Python тілі арқылы оқыту техникалық мамандықтардың студенттеріне қолайлы болуы мүмкін. IT мамандықтарында программалауды оқытудың келесі сатысы «Алгоритмдер және деректер құрылымы» пәнінде жалғастырылады. Бұл пәнде қарастырылатын барлық алгоритмдер деректердің құрылым деп аталатын арнайы түрі арқылы жүзеге асырылады. Python тілінде құрылым ұғымы жоқ. Құрылым түріндегі деректерді Python тілінде класс арқылы ғана қолдануға болады [1-4]. Ол үшін программалаушы адам объектіге бағытталған программалауды білуі қажет. Деректерді құрылымға қосу, өшіру, ағаш бұтақтарын қарапайым және күрделі бұру сияқты алгоритмдерді класс әдістері арқылы программалау студенттерге қиындық туғызады.

Мақалада жиі қолданылатын деректер құрылымдарының алгоритмдерін Python тілінде күрделі классыз қарапайым программалау қарастырылады.

Бір жақты байланысқан тізбек

Бір жақты байланысқан тізбек динамикалық деректер құрылымдарының негізгі түрлерінің бірі болып есептеледі. Бір бірімен нұсқау арқылы байланысқан, жадыда әр жерде орналасқан, көбейіп немесе азайып отыратын деректер тобы көптеген күрделі алгоритмдерде қолданылады (1-сурет). Осы құрылымды қолдану арқылы стек, кезек және дек құрылымдары құрылады [5-7]. Сонымен қоса бір жақты байланысқан тізбек хеш кестесінде де қолданылады [5].

Байланысқан тізбектерді өрнектеуді Python тілінде екі класты (түйін және тізбек) қолданып жүзеге асыруға да болады [1-4, 8, 9].



Сурет 1. Бір жақты байланысқан тізбек

Бір жақты байланысқан тізбекті Python тілінде бір класты ғана қолданып программалау мысалы.

```
class Node:              #  тізбектің бір элементі (түйіні)
   def __init__(self, val):
      self.data = val
      self.next = None
def add(item):    #   тізбекке элемент қосу функциясы
    global head
    new_node = Node(item)
    new_node.next = head
    new_node.data = item
    head = new_node
def remove( item):     #   тізбектен элементті өшіру функциясы
    current = head
    previous = None
    found = False
    while current is not None and not found:
        if current.data is item:
            found = True
        else:
            previous = current
            current = current.next       # келесі түйінге ауысу
    if found:
        if previous is None:
            self.head = current.next
        else:
```

```
                previous.next = current.next
        else:
            print(item, ' - Value not found.')
    def printList():                    #   тізбекті экранға шығару (жазу) функциясы
        current = head
        while current is not None:
            print( current.data, end = " ")
            current = current.next
        print()
    #  негізгі программа
    head = None
    add(12)
    add(13)
    add(-25)
    add(34)
    printList()
    remove(13)
    printList()
```

Осы мысалдағы үлгі бойынша екі жақты байланысқан және сақиналы тізбектерді де программалауға болады.

Екілік іздеу ағашы

Екілік іздеу ағашы үлкен көлемдегі деректер жиындарында (деректер қорын басқару жүйелерінде, эксперттік жүйелерде) ізделінген деректі жылдам табу үшін қолданылады. Python тілінде екілік іздеу ағашының алгоритмдерін программалау мысалы ретінде AVL ағашын қарастырамыз [6, 7]. AVL ағашының ерекшелігі оның тепе-теңдігінде (балансталған), яғни ағаштың кез келген түйінінің оң және сол жақ бұтақтарының биіктіктерінің айырмасы 1 ден аспауы керек. Ағашқа деректі енгізу немесе ағаштан деректі өшіру кезінде ағаш түйіндерінің тепе теңдігі бұзылады. 1-кестеде AVL ағашын теңестіру жолдары көрсетілген.

Кесте 1. AVL ағашын теңестіру жолдары

Тепе-теңдіктің бұзылу жағдайлары	*Түзету әдісі*
Түйіннің тепе-теңдік коэффициенті -2 және сол жақтағы түйіннің тепе-теңдік коэффициенті -1 болғанда	*Солға қарапайым бұру (L)*
Түйіннің тепе-теңдік коэффициенті 2 және оң жақтағы түйіннің тепе-теңдік коэффициенті 1 болғанда	*Оңға қарапайым бұру (R)*
Түйіннің тепе-теңдік коэффициенті -2 және сол жақтағы түйіннің тепе-теңдік коэффициенті 1 немесе 0 болғанда	*Солға және оңға күрделі бұру (LR)*
Түйіннің тепе-теңдік коэффициенті 2 және оң жақтағы түйіннің тепе-теңдік коэффициенті -1 немесе 0 болғанда	*Оңға және солға күрделі бұру (RL)*

Python тілінде AVL ағашының алгоритмдерін программалау мысалы.

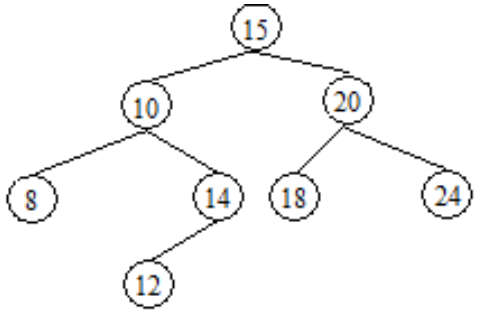
```
class Tree(object):             # ағаш құрылымы
   def __init__(self, data):  # класс конструкторы
      self.data = data          # ағаш  түйініндегі дерек
      self.left = None          # ағаштың сол жақ бұтағына нұсқау
      self.right = None         # ағаштың оң жақ бұтағына нұсқау
      self.height = 1           # ағаш түйінінің биіктігі
def calheight(node):            # ағаш түйінінің биіктігін есептеу функциясы
    if not node.left:
        if not node.right:
            return 1
        else:
            return 1 + node.right.height
```

```
    else:
        if not node.right:
            return 1 + node.left.height
        else:
            return max(node.left.height, node.right.height)+1
def balance(node):              # ағаш түйінінің тепе – теңдік коэффициентін есептеу функциясы
    if not node.left:
        if not node.right:
            return 0
        else:
            return -(node.right.height)
    else:
        if not node.right:
            return node.left.height
        else:
            return (node.left.height-node.right.height)
def rrotate(node):    #  R - бұру
    p=node.left
    node.left=p.right
    p.right=node
    node=p
    calheight(node.right)
    calheight(node)
    return node
def lrotate(node):   #  L - бұру
    p=node.right
    node.right=p.left
    p.left=node
    node=p
    calheight(node.left)
    calheight(node)
    return node
def drrotate(node):   #    LR - бұру
    node.left = lrotate(node.left)
    node = rrotate(node)
    return node
def dlrotate(node):          # RL - бұру
    node.right = rrotate(node.right)
    node = lrotate(node)
    return node
def insert(node, d):      # ағашқа жаңа дерек қосу функциясы, аргументтері
                          # node  - ағаш түбірі, d – енгізілетін дерек
   if node is None:      # егер ағаш бос болса
      return Tree(d)     # онда, дерек ағаш түбіріне орналастырылады
   if (d < node.data):   # деректі екілік іздеу ағашының ережесі бойынша орналастыру
      if not node.left:
         node.left = Tree(d)
      else:
         node.left = insert(node.left, d)
         if(balance(node) == 2):
            if(balance(node.left) == 1):
               node = rrotate(node)
            else:
               node = drrotate(node)
   elif (d > node.data):
      if not node.right:
         node.right = Tree(d)
      else:
```

```
            node.right = insert(node.right, d)
            if(balance(node) == -2):
                if(balance(node.right) == -1):
                    node = lrotate(node)
                else:
                    node = dlrotate(node)
    else:
        print(d, " - already exists")
    node.height = calheight(node)
    return node
def delete(node, d):   # ағаштан деректі өшіру функциясы, аргументтері
                       # node  - ағаш түбірі, d – өшірілетін дерек
    if (d < node.data):
        node.left = delete(node.left, d)
    elif (d > node.data):
        node.right = delete(node.right, d)
    else:
        if not node.left:
            if not node.right:
                temp = node
                node = None
            else:
                temp = node.right
                node = temp
            del temp
        elif not node.right:
            if not node.left:
                temp = node
                node = None
            else:
                temp = node.left
                node = temp
            del temp
        else:
            temp = node.right
            while temp.left:
                temp = temp.left
            node.data = temp.data
            node.right = delete(node.right, temp.data)
        if node:
            node.height= calheight(node)
            if(balance(node) > 1):
                if(balance(node.left) > 0):
                    node = rrotate(node)
                else:
                    node = drrotate(node)
            elif(balance(node) < -1):
                if(balance(node.right) < 0):
                    node = lrotate(node)
                else:
                    node = dlrotate(node)
    return node
def treeprint(node):  # ағашты жоғарыдан төмен қарастыру арқылы экранға жазу
      if node is not None:
            print (node.data, '  ', end = '  ')
            treeprint(node.left)
            treeprint(node.right)
def size(node):        # ағаштағы деректер санын есептеу функциясы
```

```
    if node==None:
       return 0;
    return (size(node.left) + 1 + size(node.right));
def find(node, a):       # ағаштан деректі іздеу функциясы, аргументтері
                         # node  - ағаш түбірі, a – ізделінетін дерек
    if node == None:
        print(a, "- sany jok")
        return False       #  егер ізделінген дерек ағашта жоқ болса
    else:
        if a == node.data:
            print(a, "- sany bar")
            return True   #  егер ізделінген дерек табылса
        else:
            if a < node.data:
                return find(node.left, a)
            else:
                return find(node.right, a)
root = None   # ағаш түбірі
k = [15, 10, 20,  24, 8, 14, 18, 12 ]  # берілген сандар тізімі
root = insert(root, k[0])    # ағашқа бірінші санды орналастыру
for i in range(1, len(k)):   # ағашқа қалған сандарды орналастыру
    insert(root, k[i])
treeprint(root)              # ағаштағы деректерді экранға шығару
print()
print(size(root))            # ағаштағы дерек саны
print(find(root, 10))        # ағаштан 10 санын іздеу
print(find(root, 16))        # ағаштан 16 санын іздеу
delete(root, 20)             # ағаштан 20 санын өшіру
treeprint(root)
```

Программада 2 суреттегі екілік іздеу ағашын қарастырылған.



Сурет 2. Екілік іздеу ағашының мысалы

Осы мысалдағы үлгі бойынша АВЛ, қызыл-қара және үйме ағаштарының алгоритмдерін программалауға болады.

Графтарды Python тілінде анықтау.

Графтарды сыбайластық және түйістілік матрицалары арқылы анықтауды Python тілінде екі өлшемді тізім түрінде жазуға болады [5 -7, 10].

Сонымен қоса графтың сыбайластық тізбегін Python тілінде ***сөздік арқылы*** қарастыру көрнекі, әрі тиімді болады. Граф *төбелері* сөздіктің *кілті* болады, ал оған сыбайлас *төбелер тізімі* осы кілттің *мәні* болады.

Графты сөздік арқылы анықтау мысалы.

```
>>> graph = {'x1': ['x1', 'x3', 'x5'], 'x2': ['x2', 'x5'], 'x3': [], 'x4': ['x3'], 'x5': ['x4'], 'x6': ['x1', 'x5'] }
>>> print( graph.keys())
```

```
dict_keys(['x1', 'x2', 'x3', 'x4', 'x5', 'x6'])
>>> print( graph.values())
dict_values([['x1', 'x3', 'x5'], ['x2', 'x5'], [], ['x3'], ['x4'], ['x1', 'x5']])
>>> graph['x1']
['x1', 'x3', 'x5']
>>> graph['x1'][0]
'x1'
>>> len(graph['x1'])
3
```

Графты тереңдігінен қарастыру.

Граф төбелерін қарастырудың екі негізгі алгоритмі белгілі: графты «тереңдігінен» және «көлденеңінен» қарастыру [3]. Бірінші жағдайда алгоритмде стек құрылымы, ал екінші жағдайда – кезек құрылымы қолданылады.

Python тілінде графты тереңдігінен қарастыру алгоритмін програмалау мысалы.

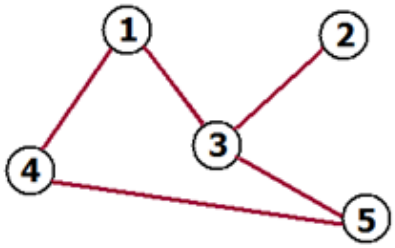
```
def dfs(graph, root):  # функция аргументтері: graph – граф құрылымы, root – бастапқы төбе
    path = []  # қарастыру тізімі, басында бос
    stek = [root]        # стекке алғашқы төбені жазамыз
    while stek:          # стек бос болғанша, қайталау
        node = stek.pop()       # стектен кезекті төбені аламыз
        if node not in path:    # егер ол төбе қарастырылмаған болса
            path.append(node) # онда оны стектегі қарастыру тізіміне қосамыз
            stek.extend([x for x in graph[node] if x not in path]) #  стекке  осы  төбеге  сыбайлас,  бірақ  әлі қарастырылмаған төбелерді қосамыз
            # print(stek)         # стектегі төбелерді көріп отыруға болады
    return path                   # нәтиже, қарастырылу реті бойынша құрылған тізім

#  негізгі программа
graph = { 1: [3, 4], 2: [3], 3: [1, 2, 5], 4: [1, 5], 5: [3, 4] } # графты сөздік арқылы анықтау
print(dfs(graph, 1))
```

Нәтиже.

```
[1, 4, 5, 3, 2]
```

Програмада 3 суреттегі граф қарастырылған.



Сурет 3. Граф мысалы

Графты көлденеңінен қарастыру.

Python тілінде графты көлденеңінен қарастыру алгоритмін програмалау мысалы.

Бұл мысалда да 3 суреттегі граф қарастырылған.

```
def bfs(graph, root):  # функция аргументтері: graph – граф құрылымы, root – бастапқы төбе
    path=[]     # қарастыру тізімі, басында бос
    kezek = [root]       # кезекке алғашқы төбені жазамыз
    while kezek:         # кезек бос болғанша, қайталау
        vertex = kezek.pop(0)    # кезектен келесі төбені аламыз
        if vertex not in path:       # егер ол төбе қарастырылмаған болса,
```

```
            path.append(vertex)    # онда оны қарастыру кезегіне қосу
            kezek.extend(node for node in graph[vertex] if node not in path) # осы төбеге сыбайлас, бірақ әлі қарастырылмаған төбелерді кезекке қосамыз
                # print(kezek)      # кезектегі төбелерді көріп отыруға болады
    return path             # нәтиже, қарастырылу реті бойынша құрылған тізім
# негізгі программа
graph = { 1: [3, 4], 2: [3], 3: [1, 2, 5], 4: [1, 5], 5: [3, 4] }
print(bfs(graph, 1))
```

Нәтиже.

[1, 3, 4, 2, 5]

Қорытынды

Алгоритмдер және деректер құрылымы пәнінде қарастырылатын күрделі алгоритмдерді программалау үшін студенттер программалау тілін жақсы білуі керек. Егер студенттер «Алгоритмдеу және программалау» пәнінде Python тілін меңгерген болса, онда күрделі деректер құрылымдарының алгоритмдерін де осы тілде программалауды үйрену керек. Деректер құрылымдарының алгоритмдерін кең тараған алгоритмдік тілдерде (Си, C++, Java және т.б.) қолдану туралы оқулықтар бар.

Қазіргі уақытта деректер құрылымдарының алгоримдерін Python тілінде қолдану туралы оқулық қазақ және орыс тілдерінде де жоқ. Ашық дерек көздерінде Python тілінде бір күрделі құрылым екі класс арқылы өрнектеу көрсетілген. Дерек түйіні бір класс, ал деректің өзі (бір және екі жақты байланысқан тізбектер, ағаштар) екінші класс арқылы қарастырылған [8, 9].

Мақалада деректер құрылымдарын алгоритмдерін қарапайым құрылым түріндегі бір класс арқылы программалауға болатыны қажетті мысалдармен көрсетілген.

Python тілінде графтарды сөздік арқылы өрнектеу графты қолданудың көрнекі, әрі түсінікті әдісі болады. Басқа алгоритмдік тілдерде (Си, C++, Java және т.б.) сөздік ұғымы жоқ.

Пайдаланылған әдебиеттер тізімі:

1 Lee K., Hubbard S. Data Structures and Algorithms with Python. - Springer, 2015, 363 p.

2 Dr. B. Agarwal. Hands-On Data Structures and Algorithms with Python. - Packt Publishing, 2018, 398 p.

3 Narasimha K. Data Structures and Algorithmic. Thinking with Python. - CareerMonk Publications, 2016, 471p.

4 Лутц М. Программирование на Python. Том 2. – М.: Символ-Плюс, 2018, 992 с.

5 Хопкрофт Дж.Э., Ульман Д.Д., Ахо А.В. Структуры данных и алгоритмы. Классическое издание. – М.: Вильямс, 2018 г., 400 с.

6 Бабичев С.Л. Лекции по алгоритмам и структурам данных. – М.: МАКС Пресс, 2019, – 344с.

7 Павлов Л.А., Первова Н.В. Структуры и алгоритмы обработки данных. Учебник. – М.: Лань, 2020 г., 256 с.

8 The Python Tutorial [Электронный ресурс]. – URL: https://docs.python.org/3/tutorial/datastructures.html #dictionaries (оқылым күні: 15.01.2021)

9 Python - Data Structure [Электронный ресурс]. – URL: https://www.tutorialspoint.com/python_data_structure/ (оқылым күні: 15.01.2021).

10 Тим Р. Совершенный алгоритм. Графовые алгоритмы и структуры данных. – М.: Питер, 2019 г., 256 с.

Refereces

1 Lee K., Hubbard S. Data Structures and Algorithms with Python. - Springer, 2015, 363 p.

2 Dr. B. Agarwal. Hands-On Data Structures and Algorithms with Python. - Packt Publishing, 2018, 398 p.

3 Narasimha K. Data Structures and Algorithmic. Thinking with Python. - CareerMonk Publications, 2016, 471p.

4 Lutc M. (2018) Programmirovanie na Python. Tom 2. []. M.: Simvol-Pljus, - 992. (In Russian)

5 Hopkroft Dzh.Je., Ul'man D.D., Aho A.V. (2018) Struktury dannyh i algoritmy. [Data structures and algorithms]. Klassicheskoe izdanie. – M.: Vil'jams,-, 400. (In Russian)

6 Babichev S.L. (2020) Lekcii po algoritmam i strukturam dannyh. [Lectures on Algorithms and Data Structures].– M.: MAKS Press, – 344. (In Russian)

7 Pavlov L.A., Pervova N.V. Struktury i algoritmy obrabotki dannyh. [Data structures and algorithms]. Uchebnik. – M.: Lan',- 256. (In Russian)

8 The Python Tutorial [Electronic resource]. – URL: https://docs.python.org/3/tutorial/datastructures.html #dictionaries (оқылым күні: 15.01.2021)

9 Python - Data Structure [Electronic resource]. – URL: https://www.tutorialspoint.com/python_data_structure/ (оқылым күні: 15.01.2021).

10 Tim R. Sovershennyj algoritm. (2019) Grafovye algoritmy i struktury dannyh. [Perfect algorithm. Graph algorithms and data structures]. M.: Piter,- 256. (In Russian)