ҚАЗАҚСТАН РЕСПУБЛИКАСЫНЫҢ БІЛІМ ЖӘНЕ ҒЫЛЫМ МИНИСТРЛІГІ

СӘТБАЕВ УНИВЕРСИТЕТI

ИАиИТ\_ институты

Программная инженерия кафедрасы



Зертханалық жұмыс

Тақырыбы: реализация рекурсивных функций для классических задач

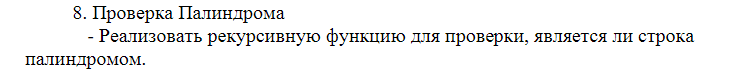
|  |  |  |  |
| --- | --- | --- | --- |
| № | Жұмысты орындау сапасы | Баға диапазоны | Орындалған  % |
| 1 | Орындалған жоқ | 0% |  |
| 2 | Орындалды | 0-50% |  |
| 3 | Материялдық өзіндік жүйелендіру | 0-10% |  |
| 4 | Талап етілген көлемде және көрсетілген мерзімде орындау | 0-5% |  |
| 5 | Қосымша ғылыми әдебиеттерді пайдалану | 0-5% |  |
| 6 | Орындаған тапсырманың ерекшелігі | 0-10% |  |
| 7 | СӨЖ-ді қорғау | 0-20% |  |
|  | Қорытынды: | 0-100% |  |

Оқытушы: Шаяхметов Диас

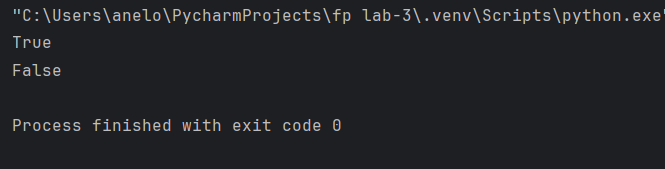
Студент: Оңалбек Әнел

Мамандығы:Computer Science

Алматы 2024



def is\_palindrome(p):  
 if len(p) < 1:  
 return True  
 else:  
 if p[0] == p[-1]:  
 return is\_palindrome(p[1:-1])  
 else:  
 return False  
print(is\_palindrome("қырық"))  
print(is\_palindrome("палиндром"))



Бұл кодта р аргументін қабылдайтын is\_palindrome функциясы құрылған. if len(p) < 1 шартында р жолының ұзындығы 1ден кем болса, яғни жол бос болса онда оны палиндром деп санайды және бұл рекурсияның негізгі жағдайы боп саналады. return True ді қайтарады. егер жол бос болмаса еlse тармағы орындалады. Мұнда біз тағы да шарт береміз: if p[0] == p[-1] бұл шарт р жолының бірінші және соңғы символы бірдей екекнін тексереді. Егер шарт орындалса орнда ол жол палиндром болуы мүмкін және функция тексеруін жалғастырады. return is\_palindrome(p[1:-1]): is\_palindrome функциясының рекурсивті шақыруы p жолының ішкі бөлігін білдіретін аргументпен жүреді. бұл р жолының кесіндісі бір индексінен минус бір индексіне , яғни екінші символдан соңғы символға дейінгі барлық сиволдарды қамтиды. Яғни, бұл кесіндісі бірінші және соңғы сиимволдарсыз р жолының ішкі бөлігін қамтиды. Else егер бірінші және соңғы символдары бірдей болмаса , онда жолымыз палиндром емес болып табылады. Сондықтан return False ты қайтарады. print(is\_palindrome("қырық")) print(is\_palindrome("палиндром")) бұл жолдар is\_palindrome функциясын “қырық” және ”палиндром” жолдарын тексеру үшін шақырады.

1. Что такое рекурсия и как она работает в контексте программирования?

Рекурсия-бұл функция өз денесінде өзін тудыратын процесс. Бағдарламалауда рекурсия бір типтегі қарапайым ішкі есептерге бөлінуі мүмкін мәселелерді шешу үшін қолданылады. Негізгі идея-бұл функция жалпы мәселені шешу үшін өзін қарапайым дәлелдермен шақырады. Рекурсивті функциялар, әдетте, функцияның шексіз шақырылуын болдырмау үшін рекурсияның аяқталу шартын анықтайтын негізгі жағдайға ие.

Санның факториалын есептейтін Python-дағы қарапайым рекурсивті функцияның мысалы:

python

Copy code

def factorial(n):

if N = = 0 : # негізгі жағдай: 0 факториалы 1-ге тең

return 1

else:

return N \* factorial (n - 1) # n-1 санына факториалды есептеуге арналған рекурсивті қоңырау

Factorial функциясы n аргументімен шақырылған кезде, n базалық жағдайдың бар-жоғын тексереді (0-ге тең). Олай болса, функция 1 қайтарады. Әйтпесе, ол N - 1 аргументімен өзін шақырады, нәтижені n-ге көбейтеді.

Алайда, рекурсияны қолданған кезде абай болу керек, өйткені рекурсивті функцияны дұрыс орындамау қоңырау стегінің (Stack overflow) толып кетуіне немесе шексіз циклге әкелуі мүмкін.

1. Какие основные компоненты рекурсивной функции?

Рекурсивті функцияның негізгі компоненттеріне мыналар жатады:

Негізгі жағдай (base Case): Бұл рекурсия аяқталатын және функция өзін-өзі шақыруды тоқтататын шарт. Негізгі жағдай, әдетте, рекурсивті қоңыраулар қажетсіз болатын белгілі бір деңгейге немесе күйге жеткенде функция аяқталатындай етіп анықталатын соңғы шартты білдіреді.

Рекурсивті жағдай (Recursive Case): Бұл функцияның өзі жаңа дәлелдермен шақырылатын функцияның бөлігі. Рекурсивті жағдайда функция қарапайым ішкі мәселені шешу үшін дәлелдердегі кейбір өзгерістермен өзін-өзі шақырады. Бұл қадам негізгі жағдайға жеткенше жалғасады.

Функцияны шақыру (функция қоңырауы): бұл функция денесінің ішіндегі тапсырманың қарапайым нұсқасын шешуге шақыратын орын. Функция шақыруы рекурсивті жағдайда болуы керек.

Айнымалылар (айнымалылар): функцияда қолданылатын айнымалылар функция аргументтері, жергілікті айнымалылар немесе ғаламдық айнымалылар болуы мүмкін. Рекурсивті функция әдетте функцияның негізгі жағдайға қарай жылжуы үшін әрбір рекурсивті қоңыраумен өзгеретін функция дәлелдерін пайдаланады.

Қайтару мәні (қайтару мәні): Бұл функция орындалғаннан кейін қайтаратын мән. Рекурсивті функцияда әдетте нәтижені негізгі жағдайдан қайтару немесе рекурсивті қоңыраулардың нәтижелерін біріктіру үшін қолданылады.

1. Каковы преимущества и недостатки использования рекурсии в программировании?

Рекурсияны қолданудың артықшылықтары мен кемшіліктері бар:

Артықшылықтары:

Қарапайымдылық пен түсініктілік: рекурсивті Алгоритмдер көбінесе мәселенің құрылымын көрнекі түрде көрсетеді, бұл кодты түсінікті және түсінуге оңай етеді.

Әмбебаптық: рекурсивті тәсілдерді ағаштарды айналып өту, факториалды есептеу, сұрыптау және т. б. сияқты әртүрлі мәселелерді шешу үшін қолдануға болады.

Кодтың қысқалығы: кейбір жағдайларда рекурсивті Алгоритмдер итеративті әріптестеріне қарағанда қысқа және талғампаз болуы мүмкін.

Кемшіліктері:

Қоңыраулар стегінің ықтимал толып кетуі (Stack Overflow): рекурсияның тереңдігі тым көп болса, қоңыраулар стегінің толып кетуі мүмкін, әсіресе егер негізгі жағдайға қол жеткізілмесе.

Жадтың жоғары қажеттілігі: рекурсивті қоңыраулар аралық нәтижелер мен қоңырау контекстін стекке сақтау қажеттілігіне байланысты көбірек жадты алуы мүмкін.

Өнімділік: кейбір жағдайларда рекурсивті Алгоритмдер итеративті аналогтарға қарағанда тиімділігі төмен болуы мүмкін, себебі қосымша қоңырау шалу және қоңырау стегін басқару шығындары.

Жөндеудің қиындығы: рекурсивті функциялардағы қателерді олардың рекурсивті сипатына байланысты бақылау және түзету қиынырақ болуы мүмкін.

1. Приведите пример задачи, для которой рекурсия является наилучшим решением.

Рекурсия рекурсивті деректер құрылымдары немесе процестер арқылы табиғи түрде көрінетін тапсырмалар үшін ең қолайлы. Рекурсия ең жақсы шешім болып табылатын классикалық мысалдардың бірі-факториалды есептеу мәселесі.

Мысал: факториалды есептеу

Факторлық сандар

�

n ретінде белгіленеді

�

!

n! және 1 ден бастап барлық оң бүтін сандардың көбейтіндісіне тең

�

N. 0 Факториалы 1 ретінде анықталады.

Факториалды есептеуге рекурсивті тәсіл:

python

Copy code

def factorial(n):

if n == 0:

return 1

else:

return n \* factorial(n - 1)

Бұл мысал рекурсияның факториалдың математикалық анықтамасын табиғи түрде қалай көрсететінін көрсетеді: санның факториалы

�

N көбейтіндіге тең

�

n және алдыңғы санның факториалы

(

�

−

1

)

(n−1). Бұл тапсырмаға рекурсивті көзқарас кодты итеративті әріптесіне қарағанда түсінікті және қысқа етеді.

1. Как вы можете избежать переполнения стека вызовов в рекурсии

Қоңыраулар стегінің толып кетуі рекурсивті қоңыраулар саны тым көп болған кезде және қоңыраулар стегі толған кезде пайда болады. Рекурсиядағы қоңыраулар стегінің толып кетуіне жол бермеу үшін бірнеше стратегияны қолдануға болады:

Итеративті түрлендіру: кейбір рекурсивті алгоритмдерді тиімділікті жоғалтпай итеративті түрлендіруге болады. Мұны рекурсивті қоңыраулардың орнына циклдарды қолдану арқылы жасауға болады.

Құйрық рекурсиясы: егер рекурсивті қоңырау функциядағы соңғы операция болса, онда ол құйрық рекурсиясы деп аталады. Кейбір бағдарламалау тілдері, мысалы, схема, қоңыраулар стегінің толып кетуіне жол бермеу үшін құйрық рекурсиясын оңтайландырады.

Циклдарды пайдалану: рекурсивті қоңыраудың орнына циклды бірдей нәтижеге жету үшін пайдалануға болады. Бұл қоңыраулар стегінің тереңдігін арттырмайды.

Қоңырау стекінің көлемін ұлғайту: кейбір бағдарламалау тілдері мен жұмыс уақыты сізге қоңырау стекінің өлшемін реттеуге мүмкіндік береді. Стек өлшемін ұлғайту қоңырау стегінің толып кетуіне жол бермеуге көмектеседі, бірақ бұл әрқашан ұсынылмайды, себебі бұл жадты пайдаланудың басқа мәселелеріне әкелуі мүмкін.

Сәйкес стратегияны таңдау нақты тапсырмаға және сіз қолданатын бағдарламалау тіліне байланысты.

1. Что такое хвостовая рекурсия и почему она важна?

Каудальды рекурсия-бұл рекурсияның ерекше жағдайы, мұнда рекурсивті қоңырау функцияда орындалатын соңғы операция болып табылады. Каудальды рекурсияның маңыздылығы-оны кейбір бағдарламалау тілдері мен жұмыс уақыты орталары оңтайландыруы мүмкін, сондықтан қоңырау стегінде қосымша орын пайдаланылмайды.

Каудальды рекурсияны оңтайландыру компиляторға немесе аудармашыға рекурсивті қоңырауларды қарапайым циклдарға ауыстыруға мүмкіндік беретін каудальды Рекурсия деп аталады, осылайша қоңырау стегіндегі жадты тұтынуды азайтады.

Құйрық рекурсиясының маңыздылығы-бұл рекурсивті алгоритмдерді қоңырау стегінің толып кету қаупінсіз тиімді пайдалануға мүмкіндік береді. Бұл әсіресе алгоритмдерді ұйымдастыру үшін рекурсияны жиі қолданатын функционалды бағдарламалау тілдері үшін пайдалы.

1. Как реализовать мемоизацию в рекурсивной функции

Мемоизация-бұл функцияның орындалу нәтижелерін бұрыннан белгілі кірістер үшін сақтаудан тұратын оңтайландыру әдісі, сол кірістер арқылы функцияның келесі қоңырауларында қайта есептеулерді болдырмас үшін.

Рекурсивті функцияларда мемоизацияны сөздік көмегімен жүзеге асыруға болады, мұнда функцияның аргументтері кілттер, ал мәндер олардың нәтижелері болып табылады. Python-да рекурсивті функцияда мемуарларды жүзеге асырудың мысалы келтірілген:

python

Copy code

# Нәтижелерді сақтау үшін сөздік жасау

memo = {}

# Мемуармен рекурсивті функция

def fibonacci(n):

# Егер нәтиже бұрыннан белгілі болса, оны сөздіктен қайтарыңыз

if n in memo:

return memo[n]

# Негізгі жағдай

if n <= 1:

return n

# Мемоизацияны қолданатын рекурсивті жағдай

memo[n] = fibonacci(n-1) + fibonacci(n-2)

return memo[n]

# Функцияны қолдану мысалы

басып шығару(fibonacci(10)) # 10 Саны үшін функцияны шақыру

Бұл мысалда fibonacci функциясы есептеу нәтижелерін сақтау үшін memo сөздігін қолданады. Функцияны шақырған сайын алдымен сөздікте берілген Аргументтің нәтижесі бар-жоғы тексеріледі. Егер бар болса, ол қайтарылады, әйтпесе нәтиже Рекурсия арқылы есептеледі, содан кейін нәтиже болашақта пайдалану үшін сөздікте сақталады.

1. Можете ли вы сравнить рекурсивный и итеративный подходы к решению задач?

Әрине, санның факториалын есептеу мысалында есептерді шешудің рекурсивті және итеративті тәсілдерін салыстырайық.

Рекурсивті тәсіл:

python

Copy code

def factorial\_recursive(n):

if n == 0:

return 1

else:

return n \* factorial\_recursive(n - 1)

Итеративті тәсіл:

python

Copy code

def factorial\_iterative(n):

result = 1

for i in range(1, n + 1):

result \*= i

return result

Енді екі тәсілді де салыстырайық:

Кодтың оқылуы:

Рекурсивті тәсіл неғұрлым қысқа және мәнерлі болуы мүмкін, өйткені ол факториалдың математикалық анықтамасын тікелей көрсетеді.

Дегенмен, рекурсивті функцияларды түсіну қиын болуы мүмкін, әсіресе оларда айқын емес негізгі жағдайлар немесе рекурсивті қоңыраулар болса.

Өнімділік:

Итеративті тәсіл әдетте жадты пайдалану және орындау жылдамдығы тұрғысынан тиімдірек, әсіресе үлкен n мәндері үшін.

Рекурсивті тәсіл үлкен n мәндерімен жұмыс істегенде қоңырау стегінің (Stack overflow) толып кетуіне әкелуі мүмкін, өйткені әрбір рекурсивті қоңырау үшін жаңа стек жақтауы бөлінеді.

Икемділік:

Итеративті тәсілді оңтайландыру және әртүрлі жағдайларға бейімдеу оңайырақ.

Рекурсивті тәсіл кейбір тапсырмалар үшін табиғи болуы мүмкін және қысқа кодты қамтамасыз етеді.

Сонымен, рекурсивті және итеративті тәсілдер арасындағы таңдау нақты тапсырмаға, өнімділік талаптарына, кодты оқуға және әзірлеушінің қалауына байланысты.

1. Какие ошибки могут возникать при написании рекурсивных функций и как их избежать?

Рекурсивті функцияларды жазу кезінде әртүрлі қателер болуы мүмкін. Міне, олардың кейбіреулері және оларды болдырмау жолдары:

Шексіз рекурсия:

Себебі: рекурсиядан шығу шартының болмауы немесе ешқашан орындалмайтын дұрыс емес шығу шарты.

Алдын алу: рекурсивті функцияның рекурсияның аяқталуына әкелетін дұрыс негізгі шарты бар екеніне көз жеткізіңіз.

Қоңыраулар стегінің толып кетуі (Stack Overflow):

Себебі: тым көп рекурсивті қоңыраулар, нәтижесінде қоңыраулар стегі таусылады.

Болдырмау: рекурсивті функцияны оңтайландыруға тырысыңыз немесе Python аудармашысы оңтайландыруы мүмкін құйрық рекурсиясын қолданыңыз.

Негізгі жағдайды дұрыс өңдеу:

Себеп: негізгі шарт дұрыс анықталмаған немесе негізгі жағдай дұрыс өңделмеген.

Алдын алу: рекурсивті функцияның негізгі жағдайды дұрыс өңдейтінін тексеріңіз.

Төмен өнімділік:

Себеп: кейбір рекурсивті Алгоритмдер артық есептеулерге байланысты өнімділікте тиімсіз болуы мүмкін.

Болдырмау: мүмкіндігінше алгоритмді оңтайландыруға немесе итеративті тәсілді қолдануға тырысыңыз.

Жад жеткіліксіз:

Себеп: кейбір рекурсивті функциялар стектегі белсенді қоңыраулардың көптігіне байланысты тым көп жадты тұтынуы мүмкін.

Болдырмау: рекурсивті функцияны оңтайландыруға тырысыңыз немесе жадты пайдалануды азайту үшін құйрық рекурсиясын қолданыңыз.

Бұл қателерді болдырмау үшін рекурсивті функцияны мұқият жобалау және сынау, сондай-ақ оны әртүрлі кірістер мен шекаралық жағдайларға тексеру маңызды.

10 Как рекурсия может быть использована для обхода структур данных, таких как деревья и графы?

Рекурсия-ағаштар мен графиктерді айналып өтуге және өңдеуге арналған қуатты құрал. Міне, осы деректер құрылымдарын айналып өту үшін рекурсияны қалай пайдалануға болады:

Ағаштар:

Тікелей айналма жол (Preorder traversal): әр түйінде алдымен ағымдағы түйін өңделеді, содан кейін оның сол жақ ішкі ағашы үшін, содан кейін оң жақ ішкі ағашы үшін функция рекурсивті түрде шақырылады.

Орталықтандырылған айналма жол( Inorder traversal): сол жақ ішкі ағаш үшін функция рекурсивті түрде шақырылады, содан кейін ағымдағы түйін өңделеді, содан кейін оң жақ ішкі ағаш үшін функция рекурсивті түрде шақырылады.

Кері айналма жол (postorder traversal): сол жақ ішкі ағаш үшін, содан кейін оң жақ ішкі ағаш үшін функция рекурсивті түрде шақырылады және ағымдағы түйін өңделеді.

Бағандар:

Тереңдікті айналып өту (Depth-First Search, DFS): ағымдағы шыңның барлық іргелес шыңдарын рекурсивті түрде айналып өтіп, оларды барған деп белгілейді. Бұл процесс графиктің барлық шыңдары өңделгенге дейін жалғасады.

Ені бойынша айналып өту (Breadth-First Search, BFS): әдетте рекурсиясыз қолданылады, бірақ кезек пен график деңгейлері бойынша рекурсивті айналып өту арқылы жүзеге асырылуы мүмкін.

Деректер құрылымын айналып өтуге арналған рекурсивті алгоритмдерді белгілі бір элементті табу, деректер құрылымының биіктігін немесе өлшемін есептеу, жол салу және т.б. сияқты әртүрлі тапсырмаларға оңай бейімдеуге болады. Дегенмен, рекурсияны пайдаланған кезде қоңырау стекінің тереңдігіндегі шектеулерді ескеру және белгілі бір тапсырма мен деректер құрылымына байланысты дұрыс айналып өту әдісін таңдау маңызды.