Frontiers in Computing and Intelligent Systems
ISSN: 2832-6024 | Vol. 9, No. 1, 2024

"Algorithm Analysis and Design" Python Teaching Examples of Recursive and Partitioning Algorithms

Ying Zhang, Min Zhou, Wenxuan Guo, Ruiqi Zhu *, Jieyu Liufu, Jiawei Lin, Zhengtao Li

School of Mathematics and Computer, Guangdong Ocean University, Zhanjiang, Guangdong, 524088, China
* **Corresponding author**: Ruiqi Zhu (Email: 11621133nn1@stu.gdou.edu.cn)

Abstract: Learning algorithm design and analysis is an essential foundation for computational problem-solving and programming. It can cultivate students' analytical and problem-solving skills, logical thinking skills and creativity. This paper shows the core ideas of recursive algorithms and partitioning algorithms and Python examples in conjunction with Python, a popular programming language, to provide adequate references for teachers and students to learn algorithm design and analysis based on the Python language.

Keywords: Algorithm Design and Analysis; Experimental Teaching; Recursive Algorithm; Partitioning Algorithm.

1. Introduction

With the rise of research fields such as big data and artificial intelligence, Python has also become popular recently. Python is a widely used interpreted, high-level, and general-purpose programming language. Python emphasizes the readability of code and concise syntax, making it relatively easy to learn, and with a low learning cost, it is one of the most popular programming languages. However, to date, there are still relatively few courses related to the Python language offered in Chinese universities, especially some core introductory courses in the field of computer science and technology, such as "Data Structures" and "Algorithm Analysis and Design," which are primarily implemented in Java, C++, and C languages, while the tutorial system using Python as the programming language is in urgent need of improvement [1].

This paper focuses on the "Algorithm Analysis and Design" course and designs experimental teaching examples using Python. The paper provides the core ideas of recursive algorithms and divide-and-conquer algorithms, as well as Python examples. These examples are practical and typical, and the reference programs provided are complete, providing a solid reference for teachers and students to learn this course.

2. Recursive Algorithm

Recursive algorithms involve a function or process calling itself, which can be either direct or indirect recursion. It is one of the core technical ideas in computer science, reflecting a way of thinking that simplifies complex problems [2]. The characteristic of recursive algorithms is that they can decompose complex problems into simpler sub-problems. These sub-problems have the same nature and form as the original problem but are smaller in scale and, thus, more accessible to solve [3]. Recursive algorithms have two fundamental components:

(1) There must be a base case that stops the recursion, i.e., the exit condition of recursion;

(2) There must be a recursive structure consistent with the original problem's structure but a smaller input size than the original problem.

Example problem: The Fibonacci sequence, derived from a rabbit breeding problem in the famous work "Liber Abaci" by the Italian mathematician Fibonacci, is a sequence: 1, 1, 2, 3, 5, 8, 13, 21, 34... That is, except for the first two numbers, which are both 1, from the third number onwards, each number is the sum of the two preceding numbers. Find the value of the Fibonacci sequence when n=34.

Problem analysis:

Firstly, we know the general formula for the Fibonacci sequence is:

$$\begin{cases} F(n) = 0 & n = 0 \\ F(n) = 1 & n = 1 \\ F(n) = F(n-1) + F(n-2) & n \geq 2 \quad n \in N \end{cases}$$

For n less than or equal to 1, we directly return the F(n) value.

For n greater than or equal to 2, we recursively calculate the values of F(n-1) and F(n-2), and after computing the values of F(n-1) and F(n-2), we return the value of F(n).

The program and the running result are shown in Figure 1.

Considerations for using recursive algorithms:

Recursion is defining a problem or its solution based on a simplified version of the problem [4]. Therefore, when defining and solving sub-problems, the method of solving the sub-problems must be consistent with the original problem.

Using recursion requires memory space from the stack. If the recursion is too deep, it will consume too much stack space, which has high memory requirements and thus reduces the efficiency of the program's operation [5]. At the same time, there is a risk of Stack Overflow.

Problems solved through recursion must be within the scope of legitimate solutions. Taking the Fibonacci sequence as an example, when n=-1, the program will return -1, but in reality, the problem does not hold and should prompt an input error. Therefore, before recursion, the program needs to validate the data.

```
def fib_rec(n):
    if n <= 1:
        f=n
    else:
        f=fib_rec(n-1)+fib_rec(n-2)
    return f

if __name__ == '__main__':
    num = int(input('请输入数字：'))
    print('{0:5}==>{1:6d}'.format('fib('+str(num)+')', fib_rec(num)))
```

```
请输入数字：34
fib(34)==>5702887
```

Figure 1. Fibonacci Sequence

3. Partitioning Algorithm

Divide and conquer is a strategy that involves breaking down a complex problem into equivalent smaller sub-problems, solving each of them individually, and then combining the solutions to form the solution to the original problem[6]. The fundamental idea is to directly transform a complex problem into several more minor problems that are less in size but identical to the original problem and solve them separately[7]. A divide and conquer algorithm generally includes the following three main steps:

(1) Divide the original problem into several smaller, independent sub-problems that are the same or similar in form to the original problem[8];

(2) Use recursion to find the solutions to each sub-problem;

(3) Merge the sub-problems' solutions to obtain the original problem's solution.

Example problem: Like-minded book friends. Recommendation websites like Douban will find readers in its database with very similar ratings to yours for a series of books and recommend them to you, helping you find like-minded book friends. Suppose your ratings for five books from low to high are [1,2,3,4,5], and reader A's ratings for these five books are [2,4,1,3,5], and reader B's ratings are [3,4,1,5,2], then Douban might recommend reader A to you. The recommendation criterion is to measure similarity by calculating the inverse order quantity. For example, the inverse order count for [1,2,3,4,5] is 0, for [2,4,1,3,5] it is 3, and for [3,4,1,5,2] it is 5, which means compared to reader B, reader A's taste is closer to yours. Solve the inverse order problem programmatically. For example, the input sequence is [5,9,3,8,6,7,10].

Problem analysis:

For an array A, let LA be the left half, and RA be the right half divided by the midpoint. Let N(A) be the inverse order count of A, and f(X) be the number of RA elements greater than the number X in LA. Then, we derive the formula:

$$N(A) = N(LA) + N(RA) + \sum_{i=1}^{length(LA)} f(LA[i])$$

According to the divide and conquer strategy, we continue to break down, and when the number of elements in A is 1, its N(A) is 0. Therefore, what we need to solve additionally is $\sum_{i=1}^{length(LA)} f(LA[i])$

First, we think of a naive algorithm to traverse all elements in LA and RA one by one, but the time complexity of this is O(n^2), which is no different from direct brute force enumeration, so we try to optimize the algorithm;

Since the order of elements inside LA and NA has no effect on $\sum_{i=1}^{length(LA)} f(LA[i])$, we can sort RA in non-decreasing order when calculating N(RA), and use binary search to find *f(LA[i])*, at this time, the time complexity is $O(n(\log n)^2)$;

When calculating LA, also sort it in non-decreasing order and enumerate the Xth element in LA.

$$f(LA[X] \geq f(LA[X-1]))$$

According to the above conclusion, we only need to define a variable temp to record the number of elements in LA that are greater than the elements in RA, set ans to record the current sum of f(LA[i]), and each time a new element of LA is enumerated, only the following judgment is needed:

$$\begin{cases} ans += temp & temp == length(RA) \\ ans += temp & LA(X) \leq RA[temp] \\ temp += 1 & while(LA[X] > RA[temp]) \end{cases}$$

The time complexity of the algorithm is $O(n\log n)$.

The program and the running result are shown in Figure 2.

Considerations for using divide and conquer algorithms:

(1) Divide and conquer algorithms typically rely on recursive algorithms, so when using divide and conquer algorithms, the problem must also meet the requirements for using recursive algorithms.

(2) Reasonably dividing the original problem into several sub-problems and being able to solve them correctly is the key and difficulty of divide and conquer algorithms. Therefore, we must focus on how to divide and "conquer" after the division [9].

4. Conclusion

Recursion is a powerful tool for designing and describing algorithms and is often used to describe complex algorithms. Although recursion is used in the implementation of many complex algorithms and is one of the foundations of learning algorithm design, recursive algorithms are abstract and complex for beginners to understand, making it one of the focal points in the curriculum of algorithm design and analysis. Divide and conquer algorithms have a wide range of practical applications. Teaching students about divide and conquer algorithms is also necessary to cultivate their ability to transform real-world problems into abstract models. This article uses Python as the programming language to explain the above two types of algorithms, with classic examples and specific analysis, which has a particular practical value.

```
def count_inversions_dc(A):
    lenA = len(A)
    if lenA <= 1:
        return 0, A
    middle = lenA // 2
    leftA = A[:middle]
    rightA = A[middle:]
    countLA, leftA = count_inversions_dc(leftA)
    countRA, rightA = count_inversions_dc(rightA)
    countLRA, mergedA = merge_and_count(leftA, rightA)
    return countLA+countRA+countLRA, mergedA

def merge_and_count(A, B):
    i, j, inv_count =0, 0, 0
    alist = []
    lenA = len(A);lenB = len(B)
    while i<lenA and j<lenB:
        if A[i]<B[j]:
            alist.append(A[i])
            i+=1
        else:
            inv_count += lenA-i
            alist.append(B[j])
            j+=1
    while i<lenA:
        alist.append(A[i])
        i+=1
    while j<lenB:
        alist.append(B[j])
        j+=1
    return inv_count, alist

if __name__ == '__main__':
    alist =[5,9,3,8,6,7,10]
    print('逆序对数及原序列按从大到小的顺序为: ')
    print(count_inversions_dc(alist))
```

```
逆序对数及原序列按从大到小的顺序为:
(7, [3, 5, 6, 7, 8, 9, 10])
```

Figure 2. Inversion Pairs

Acknowledgments

This work was supported by the Guangdong Ocean University Research Startup Foundation (060302102304) and the Guangdong Ocean University Innovation and Entrepreneurship Training Program Project Grant (CXXL 2024132 and CXXL2024141).

References

[1] Na Jiaofen, Yang Wenya, Li Hongchan, Zhu Haodong. Reflections on the Teaching Process of "Algorithm Design and Analysis" [J]. Modern Education, 2019, 6(35): 189-190.

[2] Xu Anxi. A Brief Discussion on Teaching Strategies of Recursive Algorithms [J]. China New Telecommunications, 2022, 24(03): 170-171.

[3] Chen Xin. Discussion on Teaching Methods of Recursive Algorithms [J]. Fujian Computer, 2022, 38(09): 67-70.

[4] Ali Grami, Chapter 14 - Recursion, Discrete Mathematics, 2023: 249-269.

[5] Li Wei. A Brief Analysis of Recursive Algorithms in C Language [J]. Computer Knowledge and Technology, 2012, 8(30): 7229-7235.

[6] Long Tenfang, Gao Jinwen. Application of "Divide and Conquer" Method in Algorithm Design [J]. Journal of Bohai University (Natural Science Edition), 2004, (01): 22-24.

[7] Wang Haiyuan. Two Approaches and Forms of Divide and Conquer Algorithms [J]. Journal of Shanghai Normal University (Natural Science Edition), 2003, (01): 39-43.

[8] Research and Application of Scope Planning Methods for IT Projects [D]. Yang Xiaolei. Shanghai Jiao Tong University, 2008.

[9] Xi Yuxin. Research on Divide and Conquer Algorithms and Dynamic Programming Algorithms [J]. Yangtze River Information and Communication, 2021, 34(06): 44-46.

[10] Liu Zhuoya. Research on Recursive Algorithms Based on C Language in Digital Technology and Application, 2018, 36(3): 132.