問題解法における再帰関数利用の研究

－POJ Judge Online の問題解法を通して－

瓜生　隆弘

A Study of a Solution to a Problem Using Recursive functions
- A Problem from POJ Judge Online –

Takahiro Uryu

Abstract

Sometimes it is useful to use recursive functions on a computer programming. In this paper, I take the problem of the online judge system provided by Peking University as an example.

I created a simple solution for the same problem and a solution that uses a recursive function, and measured both calculation times. It was found that the solution method using the recursive function has an easy-to-understand structure, but it takes more time to calculate than the simple solution method. It seems that there is room for improvement, but I would like to leave that consideration to subsequent papers.

The solution method of this paper is described using Python.

Keywords: Python, Recursive function, Computer Programming, Algorithm, POJ

１．はじめに

ＰＯＪとは、北京大学が運営するオンラインジャッジシステムである。2004 年以来、様々なイベントが開催され、大学生やプログラミング愛好家が参加してきた。2014 年頃から活動が下火になってはいるが、2021 年現在でもさまざまな問題セットが提供されている。ただし C++98（GCC4.4.0）が採用されており、C++11 以後の機能は利用できない。また Python 言語には対応していない。

本論文では、その中からひとつの問題を例題として取り上げた。コンピュータプログラミングにおいて時に再帰関数を利用することが有効な場合がある。取り上げた問題に対して、素朴な解法と再帰関数を利用する解法を作成し、処理時間について比較、考察を行った。また、本論文で示した解法等は Python 言語を利用して記述した。

２．再帰関数とは

コンピュータプログラミングにおける再帰関数とは、自分自身を呼び出す関数と説明できる。再帰関数を設計する際の条件を示す。

条件１：最終的にたどり着く終着点を記述すること。

条件２：問題をより小さくしていくように設計すること。

上記２つの条件により、問題は最終的に終着点にたどり着き、無限に繰り返すことはなく、必ず答えを得ることができる。たとえば、与えられた正の整数までの総和を求めることを考えよう。つまり整数

ｎが与えられ、ｎ以下の正の整数の総和（1+2+3+…+n）を求めることを考える。総和を求めるには、以下に示すような再帰関数を考えればよい。

```
# 再帰関数の定義
def f(n):
        if n==0:
        return 0;
else:
        return n+f(n-1)
# 再帰関数の定義おわり

# 以下、メインとなるプログラム
n=int(input())
print(n,”までの総和は”,f(n))
# メインプログラムおわり
```

例えば、n=6 の時の動作を確認すると、f(6)に対して、その値は 6+f(5)となり、f(5)が呼び出される。f(5)=5+f(4)なので f(4)が呼び出され、f(4)は 4+f(3)、f(3)は 3+f(2)、f(2)は 2+f(1)、f(1)は 1+f(0)と次々に呼び出されることとになる。ここで f(0)は 0 なので、f(1)=1+0、f(2)=2+1、f(3)=3+3、f(4)=4+6、f(5)=5+10、f(6)=6+15 と遡り、最終的に 21 が返されるという仕組みである。

３．取り組んだ問題

ＰＯＪで提供されている問題より、POJ3617 を取り上げた。この問題は辞書順最小の問題に分類できる。簡単に説明すると、与えられた文字列の先頭と最後尾の文字を辞書順に比較し、辞書順で前にある文字を出力する問題である。出力した文字は文字列から削除する。文字列の長さが０になるまでこの操作を繰り返すことで解答を得る。(参考資料を参照)

４．素朴な解法

与えられた文字列の先頭の文字と最後尾の文字を辞書順に比較するプログラムを作成した。比較したいふたつの文字が同じ場合、それぞれ、ひと文字内側の文字を比較する。その文字も同じ場合はさらに内側を比較する。素朴な解法を【コード１】に示した。

５．再帰関数を利用する解法

素朴な解法で見たように、文字列の先頭と最後尾の文字を比較する際、ふたつの文字が同じ場合、それぞれ、ひと文字内側の文字を比較する。その文字も同じ場合、さらに内側を比較する処理が必要である。文字列の内側を順々に調べていく部分を再帰的に考えることができる。

先頭の位置を head とすれば、調べる文字の位置は head+i（i=1,2,3…）と変化していく。最後尾の位置を tail とすれば、調べる文字の位置は tail-i（i=1,2,3…）と記述できる。したがって、条件１として示しておいた、最終的にたどり着く終着点は head >= tail となる。ここまで調べても順番が決まらないなら、どちらを採用しても結果は同じになるから、本プログラムでは最後尾の文字を採用することとした。

処理が進むと文字列の長さは短くなっていくので条件２もクリアできる。

以上の発想により作成した再帰関数を利用する解法を【コード２】に示した。

６．まとめ

乱数を使い、2,000 文字からなる文字列を発生させ、【コード１】と【コード２】の処理速度を測定した。筆者の環境で、それぞれを 360 回ずつ実行した結果を以下に示す。

素朴な解法の平均処理時間は 0.00432 秒（標本数 360、標準偏差 0.000586)、再帰関数を利用する解法の平均処理時間は 0.00515 秒（標本数 360、標準偏差 0.000650)。それぞれの結果より 2 つの平均値の差の検定（大標本）を行った結果、検定統計量 t=18.04985 となった。$t_{0.05}$=1.645（有意水準 0.05、片側）であるから、処理時間に有意な差があり、採用する解法が処理時間に影響することがわかった。

この結果より再帰関数を利用する解法において、内側の文字を調べた結果が以降の処理で全く利用されていないことが有意差がある原因なのではないかと考えた。内側の文字を調べた結果をメモ化するなどの手法を導入することで、処理時間を短縮・改善できるのではないかと考察した。

７．参考文献

秋葉拓哉、岩田陽一、北川宣稔「プログラミングコンテストチャレンジブック第 2 版」マイナビ出版、2018 年

POJ　北京大学オンラインジャッジシステム(最終閲覧日 2021 年 9 月 20 日) http://poj.org/problem?id=3617

村上正康、安田正實「統計学演習」培風館、2005 年

８．参考資料

筆者の環境　ハードウエア

ノートパソコン　DELL ALIENWARE 17R5 (2019 年製)

プロセッサ i9-8950HK CPU @2.90GHz

メモリ　32.0GB

64 ビットオペレーションシステム x64 ベース

ソフトウエア

Microsoft Visual Studio Community 2019 Version 16.6.3

Python3.8.2

例題として取り上げた問題

北京大学オンラインジャッジシステム問題ＰＯＪ3617 より

Best Cow Line

Time Limit: 1000MS　　Memory Limit: 65536K

Total Submissions: 56990　　Accepted: 14310

Description

FJ is about to take his N (1 ≤ N ≤ 2,000) cows to the annual"Farmer of the Year" competition. In this contest every farmer arranges his cows in a line and herds them past the judges. The contest organizers adopted a new registration scheme this year: simply register the initial letter of every cow in the order they will appear (i.e., If FJ takes Bessie, Sylvia, and Dora in that order he just registers BSD). After the registration phase ends, every group is judged in increasing lexicographic order according to the string of the initials of the cows' names.

FJ is very busy this year and has to hurry back to his farm, so he wants to be judged as early as possible. He decides to rearrange his cows, who have already lined up, before registering them.
FJ marks a location for a new line of the competing cows. He then proceeds to marshal the cows from the old line to the new one by repeatedly sending either the first or last cow in the (remainder of the) original line to the end of the new line. When he's finished, FJ takes his cows for registration in this new order.
Given the initial order of his cows, determine the least lexicographic string of initials he can make this way.

Input
* Line 1: A single integer: N
* Lines 2..N+1: Line i+1 contains a single initial ('A'..'Z') of the cow in the ith position in the original line

Output
The least lexicographic string he can make. Every line (except perhaps the last one) contains the initials of 80 cows ('A'..'Z') in the new line.

Sample Input

6
A
C
D
B
C
B
Sample Output

ABCBCD
Source

USACO 2007 November Silver
(出典：北京大学 Peking University http://poj.org/problem?id=3617)

【コード１】# 素朴な解法

```
S=input()
ans=''
```

```
    head=0
    while len(S)>0:
        tail=len(S)-1
        if S[head]<S[tail]: # 先頭が小さいなら先頭を採用し、先頭を削除する
            ans+=S[head]
            S=S[1:]
        elif S[head]>S[tail]: # 最後尾が小さいなら最後尾を採用し、最後尾を削除する
            ans+=S[tail]
            S=S[:tail]
        else: # 先頭と最後尾が同じ文字の場合、１文字内側を比較する
            judge=False
            i=1
            while head+i<tail-i:
                if S[head+i]<S[tail-i]:
               # 先頭から見たほうが小さい場合、先頭を採用し、ジャッジを True にする
                    ans+=S[head]
                    S=S[1:]
                    judge=True
                    break
                elif S[head+i]>S[tail-i]:
               # 最後尾から見たほうが小さい場合、最後尾を採用し、ジャッジを True にする
                    ans+=S[tail]
                    S=S[:tail]
                    judge=True
                    break
                i+=1
            if judge==False:
           # 判定が付かなかった場合、どちらでも良いが、最後尾を採用しておく
                ans+=S[tail]
                S=S[:tail]
    print(ans)
```

【コード２】# 再帰関数を利用する解法

```
# 再帰関数
def f(h,t):  # h,t は比較する文字の位置
    if h>=t: # 比較している文字の位置が同じか、前後が入れ替わったら False を返して終了する
        return False
    if S2[h+1]<S2[t-1]: # 先頭から見たほうが小さい時、先頭を採用したいので True を返す
        return True
    elif S2[h+1]>S2[t-1]: # 最後尾から見たほうが小さい時、最後尾を採用したいので False を返す
```

```
        return False
    else: # 比較している文字が同じなら、さらに１文字内側を比較する　＜再帰＞
        h+=1
        t-=1
        return f(h,t)
# 再帰関数ここまで
    S2=input()
    ans2=''
    head=0
    while len(S2)>0:
        tail=len(S2)-1
        if S2[head]<S2[tail]: # 先頭が小さいなら先頭を採用し、先頭を削除する
            ans2+=S2[head]
            S2=S2[1:]
        elif S2[head]>S2[tail]: # 最後尾が小さいなら最後尾を採用し、最後尾を削除する
            ans2+=S2[tail]
            S2=S2[:tail]
        else: # 先頭と最後尾が同じなら、再帰関数を使って、どちらを採用するか調べる
            if f(head,tail)==True:
                ans2+=S2[head]
                S2=S2[1:]
            else:
                ans2+=S2[tail]
                S2=S2[:tail]
    print(ans)
```

おわり