作业反馈与讨论

2013-11-15

Problem 2.10

- 设计一个算法检查一个向量是否是一个排列。

Problem 2.10

The following algorithm checks whether the vector P of length N represents any permutation of A_N. It uses a vector A of length N that contains Boolean values (true or false) to keep track of the integers already encountered in P. The result is set into the variable E, which is true upon termination of the algorithm precisely if P indeed represents a permutation.

```
for I going from 1 to N do the following:
    A[I] ← false;
I ← 1;
E ← true;
while E is true and I ≤ N do the following:
    J ← P[I];
    if 1 ≤ J ≤ N and A[J] is false then do the following:
        A[J] ← true;
        I ← I + 1;
    otherwise
        E ← false.
```

- 设计一个算法产生所有使用1~N的排列。

Problem 2.11

Here is an algorithm which, given N, prints all the permutations of A_N. It uses two vectors A and P of length N each. The vector A contains Boolean values and represents those integers already considered in the current permutation being generated in the vector P.

```
for I going from 1 to N do the following:
    A[I] ← true;
call perms-from 1.
```

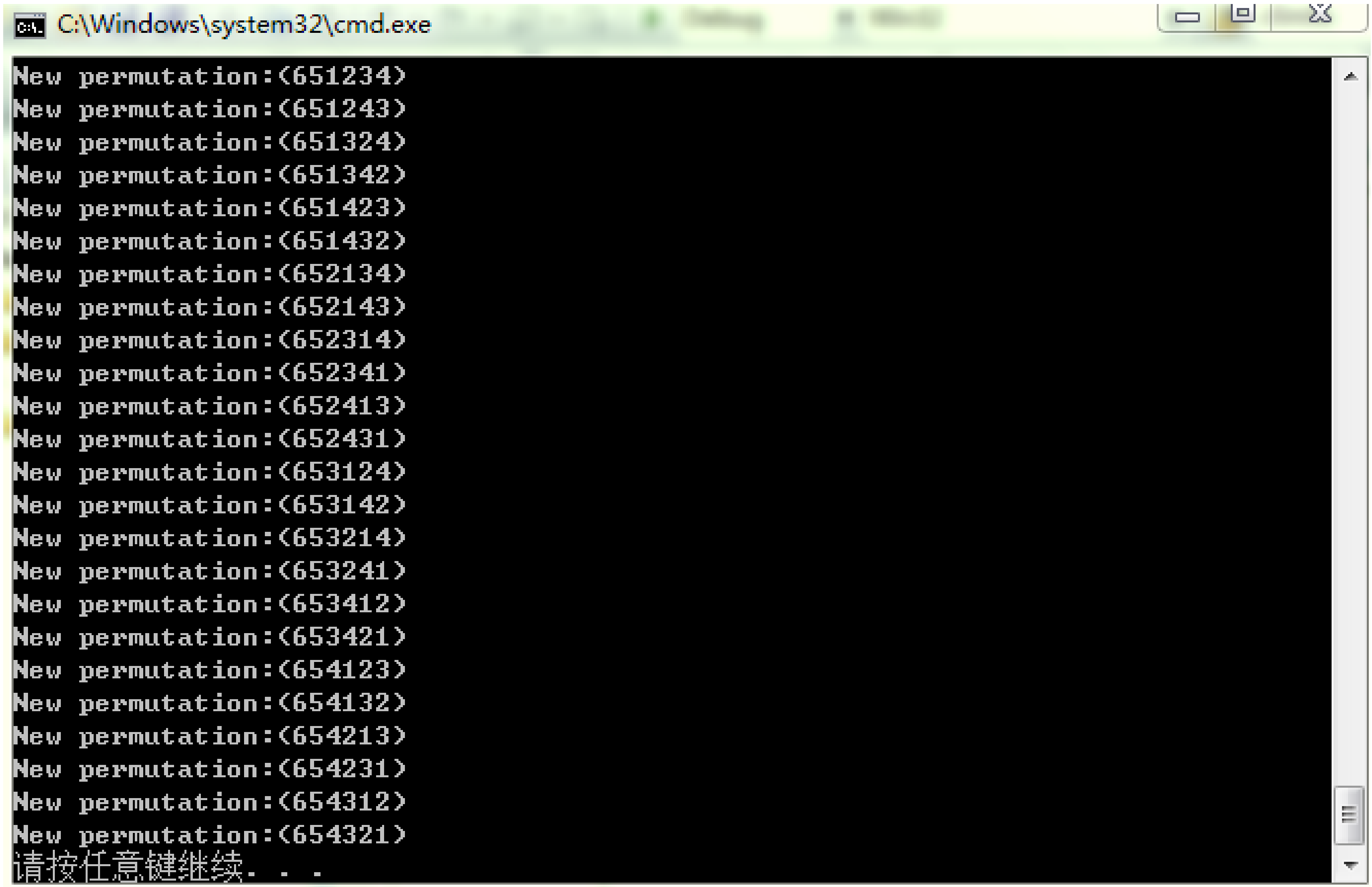
where the subroutine **perms**, with local variable J, is defined by

```
subroutine perms-from K:
    if K > N then do the following:
        print("New permutation: (");
        for J going from 1 to N do print(P[J]);
        print(")");
    otherwise (i.e., K ≤ N) do the following:
        for J going from 1 to N do the following:
            if A[J] is true then do the following:
                P[K] ← J;
                A[J] ← false;
                call perms-from K + 1;
                A[J] ← true;
    return.
```

```
#include <iostream>
#define N 7 //产生~6构成的permutation.
using namespace std;
void permsfrom(int);
bool check[N];
int p[N];
int main()
{
        for(int l = 1;l<N;l++)
                {
                        check[l]=true;
                }

        permsfrom(1);
}

void permsfrom(int k)
{
        if (k>N-1){
                cout<<"New permutation:(";
                for(int j = 1;j<=N-1;j++)
                        cout<<p[j];
                cout<<")"<<endl;
        }
        else
        {
                for(int j=1;j<=N-1;j++)
                {
                        if (check[j] ==true)
                        {
                                p[k]=j;
                                check[j]=false;
                                permsfrom(k+1);
                                check[j]=true;
                        }
                }
        }
}
```


C:\Windows\system32\cmd.exe
New permutation:(651234)
New permutation:(651243)
New permutation:(651324)
New permutation:(651342)
New permutation:(651423)
New permutation:(651432)
New permutation:(652134)
New permutation:(652143)
New permutation:(652314)
New permutation:(652341)
New permutation:(652413)
New permutation:(652431)
New permutation:(653124)
New permutation:(653142)
New permutation:(653214)
New permutation:(653241)
New permutation:(653412)
New permutation:(653421)
New permutation:(654123)
New permutation:(654132)
New permutation:(654213)
New permutation:(654231)
New permutation:(654312)
New permutation:(654321)
请按任意键继续. . .

Problem 2.12

- 使用单堆栈、队列或者双堆栈获得给定的数字序列
- (a) ……

(b) We prove that the following permutations cannot be obtained by a stack:

   i. The permutation (3, 1, 2). In order to print 3 first, the input integers 1 and 2 have to be previously pushed on to the stack. But this can only happen in the order 1, 2, so that 2 will necessarily be on the top. Now, 2 has to be popped and immediately printed, otherwise it is lost.

   ii. The permutation (4, 5, 3, 7, 2, 1, 6). In order to print 4 first, the integers 1, 2, and 3 must be pushed (in this order) on to the stack. After printing 5, the integer 3 has to be popped and printed. Now, in order to print 7, the input 6 has to be first pushed on to the stack. Therefore, the integer at the top of the stack is now 6, and 2 cannot be printed before it.

(c) It is easy to check all $4! = 24$ permutations of A_4 and find that precisely 10 of them cannot be obtained by a stack. Alternatively, the number of permutations of A_N that can be obtained by a stack is given by the formula

$$\frac{(2 \times N)!}{N! \times (N+1)!}$$

which we will not prove here. Therefore, A_4 has

$$\frac{(2 \times 4)!}{4! \times (4+1)!} = \frac{8!}{4! \times 5!} = 14$$

permutations obtained by a stack, so that $24 - 14 = 10$ permutations are not.

Catalan Numbers

Problem 2.14

- 证明2.12(b)中的序列可以使用一个队列或者两个栈得到。
- 证明任何一个排列都可以通过使用一个队列得到；
- 证明任何一个排列都可以通过使用两个栈得到。

Problem 2.15

The following algorithm prints the series of operations on one or two stacks for obtaining a given input permutation. The variable R is true at the end precisely if the permutation can be obtained by one stack. The algorithm uses two stacks, S and S', with the **push**, **pop**, and **is-empty** operations. The result is produced in the variable E, which is true upon termination precisely when the input permutation can be obtained by a single stack.

```
E ← true;
I ← 1;
while input is not empty do the following:
    read(Y);
    while Y > I do the following:
        push(I, S);
        print("read(X)");
        print("push(X, S)");
        I ← I + 1;
    if Y = I then do the following:
        print("read(X)");
        print("print(X)");
        I ← I + 1;
    otherwise (i.e., Y < I) do the following:
        pop(Z, S);
        print("pop(X, S)");
        while Z ≠ Y do the following:
            E ← false;
            push(Z, S′);
            print("push(X, S′)");
            pop(Z, S);
            print("pop(X, S)");
        print("print(X)");
        while is-empty(S′) is false do the following:
            pop(Z, S′);
            print("pop(X, S′)");
            push(Z, S);
            print("push(X, S)").
```

Problem 2.16

2.16. Consider the treesort algorithm described in the text.

(a) Construct an algorithm that transforms a given list of integers into a binary search tree.

(b) What would the output of treesort look like if we were to reverse the order in which the subroutine **second-visit-traversal** calls itself recursively? In other words, we consistently visit the right offspring of a node before we visit the left one.

BNF

- **BNF**是“Backus Naur Form”的缩写。John Backus和Peter Naur首次引入一种形式化符号来描述给定语言的语法（最早用于描述ALGOL 60 编程语言，参见[Naur60]）。确切地说，早在**UNESCO**（联合国教科文组织）关于ALGOL 58的会议上提出的一篇报告中，Backus就引入了大部分BNF符号。虽然没有什么人读过这篇报告，但是在Peter Naur读这篇报告时，他发现Backus对ALGOL 58的解释方式和他的解释方式有一些不同之处，这使他感到很惊奇。首次设计ALGOL的所有参与者都开始发现了他的解释方式的一些弱点，所以他决定对于以后版本的ALGOL应该以一种类似的形式进行描述，以让所有参与者明白他们在对什么达成一致意见。他做了少量修改，使其几乎可以通用，在设计ALGOL 60的会议上他为ALGOL 60草拟了自己的BNF。看你如何看待是谁发明了BNF了，或者认为是Backus在1959年发明的，或者认为是Naur在1960年中发明。（关于那个时期编程语言历史的更多细节，参见1978年8月，《Communications of the ACM（美国计算机学会通讯）》，第21卷，第8期中介绍Backus获图灵奖的文章。这个注释是由来自Los Alamos Natl.实验室的William B. Clodius建议的）。
- 自从那以后，几乎每一个新编程语言书的作者都使用BNF来描述语言的语法规则。

- BNF的元符号：
- ::= 表示“定义为 ”
- | 表示“或者”
- < > 尖括号用于括起类别名字。
- 尖括号将语法规则名字（也称为非终结符）同终结符区分开来，终结符想表达什么意思就怎么书写。

- 可选项被括在元符号“[”和“]”中
- 重复项（零个或者多个）被括在元符号“{”和“}”中
- 仅一个字符的终结符用引号（"）引起来，以和元符号区别开来

PASCAL in BNF

- http://bernhard.userweb.mwn.de/Pascal-EBNF.html

C in BNF

- The C Programming Language

ALGOL 60 in BNF

- ALGOL 60

- 递归下降分析（用于语法分析）

Example 4.29: Consider the grammar

$$\begin{array}{lcl} S & \rightarrow & c\ A\ d \\ A & \rightarrow & a\ b \mid a \end{array}$$

To construct a parse tree top-down for the input string $w = cad$.

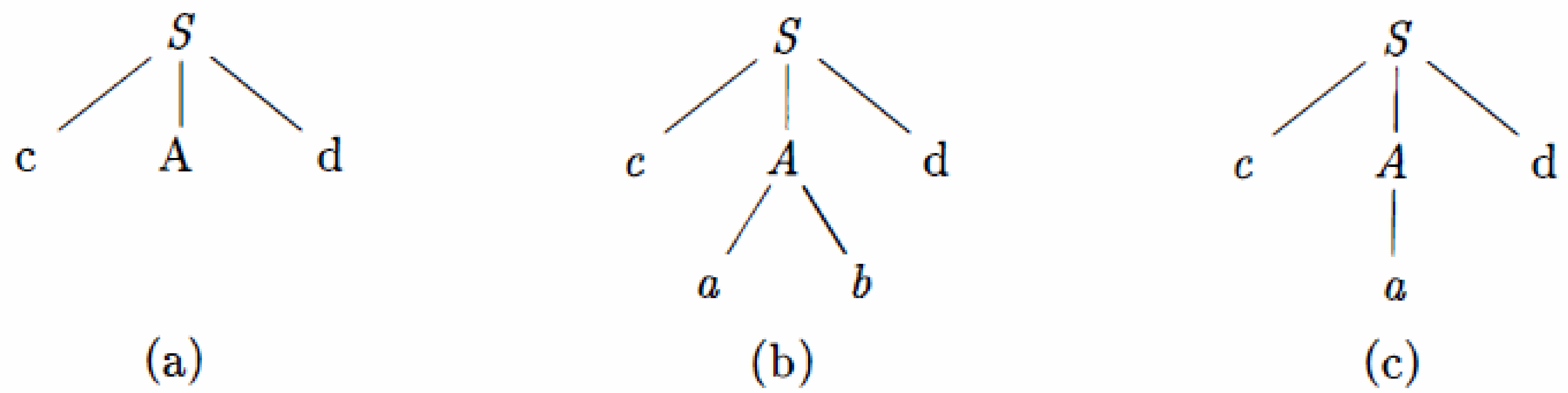


Figure 4.14: Steps in a top-down parse

Compilers: Principles, Techniques, & Tools, Aho, Lam, Sethi, Ullman.

4.4.1 Recursive-Descent Parsing

```
void A() {
        Choose an A-production, A → X1 X2 ··· Xk;
        for ( i = 1 to k ) {
                if ( Xi is a nonterminal )
                        call procedure Xi();
                else if ( Xi equals the current input symbol a )
                        advance the input to the next symbol;
                else /* an error has occurred */;
        }
}
```