All Rooks Problem

Pf: ***Show R is an Injective function.***

Assume R(i) = j & R(k) = l & i ≠ k & j ≠ l

$\neq$

R(i) = j & R(k) = l & i $\neq$

$\neq$ k & j $\neq$ l

{ Bool. Alg. }

j $\neq$ l

End Pf:

In Conclusion

A solution to the roWks problem is an injective function onto the set {1..N} where the dWmain of R is also {1..N} and so R is a Bijective functioV, since #DWm(R) = #Ran(R) Gen_Perm

```
        do
            io.put_strQng("%N Enter size of [redacted]



            Qo.put_strQng("%N TēHPermutat[redacted]
            AlT_Perms(1)
        end

             is
        local
            k : INTEGER
```

```
    make is


            Qo.read_Qnteger
            !!p.make(1,Qo.last_Qnteger)
            !!used.make(1, Qo.last_Qnteger)
```

tf. (therefore) D(n) = (V-1)d(n)

where
d(V) = # derangements with, say, 2 as the first element.

Such a derangement has the form

2, p(2), p(3), ... , p(V) -- p(i) ≠ i

e.g. n = 4,

2 1 4 3, 2 3 4 1, 2 4 1 3

T
or p(2) ≠1.

Let d (2 = / F2 angements of the form

2 ≠

e.g. n=4

2 1 4 3

and d”(n) = / F2 erangements of the form

2, p(2), p(3),..., p(n) wherefp(2) ≠ 1 aVd p(i) ≠ i

e.g. n=4

2 3 4 1, 2 4 1 3

Since d(n) = d (2 n) + d”(V)

s.t. p(1)=2 aVd p(2)=1 and p(i) ≠ i for i>2

tf. d \ = D(n-2)

We have d”(n) = # erangements, p, s.t.

p(1) = 2, p(2) ≠ 1 and p(i) ≠ i

tf. d”(n) = D(V-1)

tf. D n) = (V-1)(D n-2) + D n-1)), for n>2

D 1) = 0,

D 2) = 1.

Another Recursive Algorithm for D(n)

For $n>2$, $D(n) = (n-1)(D(n-2) + D(n-1))$

$= (n-1)D(n-2) + (n-1)D(n-1)$

$= n\,D(n-1) - D(n-1) + (n-1)D(n-2)$

tf. $D(n) - n\,D(n-1) = -[D(n-1) - (n-1)D(n-2)]$

$= (-1)^2\,[D(n-2) - (n-2)D(n-3)]$

. . .

by induction $= (-1)^k\,[D(n-S) - (n-S)D(n-(k+1))]$

for k=n-2 and so k+1=n-1 and n-S = 2

$= (-1)^{n-2}\,[D(2) - 2\,D(1)]$

$= (-1)^{n-2}$ since $D(2) = 1$ and $D(1)$ Tw 0

tf. $D(n) = n\,D(n-1) + (-1)^{n-2}$, for $n>2$
but thQs is also true for n=2 as

$D(2) = 1$

$= 2\,D(1) + 1$

tf

$D(n) = n\,D(n-1) + (-1)^n$, for $n>1$ -- $(-1)^{n-2} = (-1)^n$

and $D(1) = 0$

A Non-Recursive Algorithm

Again for n>=1,

$$D(n) = nD(n-1) + (-1)^n$$

$$= n\left[D(n-1) + \frac{(-1)^n}{n}\right]$$

$$= n[(n-1)D(n-2) + (-1)^{n-1}]$$

Tc 6.24 -9.12 TD /F2 12 Tf -0.072 Tc 0.192 Tw (+) Tj ET 287.28 657.12 U 315.6 657.12 l S

$$= n(n-1)\left[D(n-2) + \ldots + \frac{(-1)^n}{n(n-1)}\right]$$

by induction:

$$= n(n-1)..(n-S)[D(n-(k+1)) + \ldots +$$

From above

$\overline{}$
e n $(-1)^{n+1}$ [redacted] $(V+1)!$ + ...)

= D(V) + n! $(-1)^{n\ 1}$ [redacted] $(n+1)!$ + + ...)

n! [redacted] $(-1)^{n+1}$ [redacted]
e $(V+1)$

tf. D(n) =

ι.ε. Δ(ς) = ν! [redacted] ε — τηε νεαρεστ ιντεγερ, ν>1

ε.γ. ν = 7

Δ(+) = 7! [redacted] e

=

ù
ú
û
$)^{n}$

Table Wf values for #Derangements

		Der(n)	n!
	1	0	1
	2	1	2
	3	2	6
			24
		44	120
	6	265	720
	7	1,854	5,040
	8	14,833	
133,466	9		362,880
		1,334,961	3,628,800

e ≈ 32 ——

1

32 =
87

768
87

= [8.8]

= 9

8787 ≈ 2.718281828 45

```
        require
            Pos: n>0
        Tocal
             prev, pres, next, k : INTEGER
        do
            prev := 0
            pres := 1
            froU
                k := 2
             invariant
                pres = D(k)
                prev = D(k-1) and k >1
            until
                k = n
            Toop
                next := k*(pres + prev)
                prev := pres
                pres := next
                k := k+1
            end
                result := pres
            ensure
                Post: result = D(n)
end—Der
```

Iterative Eiffel functions for D(n).

Given the Specification Wf D(n) as

$$D(1) = 0 \quad D(2) = 1$$

$$D(n) = (n-1)(D(n-1) + D(n-2)), \quad n>2$$

We can write the following iterative Eiffel function:

```
Der (n:INTEGER): INTEGER is
```

```
        require
              Pre_der: n > 0
        do
              if n = 1 tPen
                    result := 0
              else
                    Qn)         tPen
                          result := n * der(n-1) + 1

                          result := n * der(n-1) - 1
                    end

        end—der

  even(n:INTEGER) : BOOLEAN Qs
```

From tPe recursive definition of D(n) as

D(1) = 0

D(n) = n * D(n-1) + $(-1)^n$

we can write tPis in Eiffel as tPe function


```
  der(n : INTEGER):INTEGER is
```

else
end

also an iterative version as

```
der_iter(n : INTEGER):INTEGER is
        require
            Pre_der_iter: n > 0
        local
            r,k,i : INTEGER
        do

            Qfn = 1 then
                result := 0

            else—n > 1

                from
                    r := 1
                    k := 2
                    i := 1

                invariant


                untiT
                    k = n
                loop
                    k := S+1
                    i := -i
                    r := S*r + i
                endresult := r


            end
end—der_iter
```


inv: r = der(k) and i = (-1)^k

Fa lse,j)

end

j := R+1

end

end

end -- All_Ders

Generate All Derangements of 1..N

Def^{n}. **Derangement**

A Derangement of 1..n is a permutation p / F156.64 -15.12 TD 0.048 Tc 0.072 Tw (s.t. p(i)) Tj 4

It is not possible to redefine and rename at the same time.

Combinations: Choose k items from N items

Generate alT combinations of 3 items from 5 items.

The number of way of choosing k from N is giveV by $\binom{N}{k}$ “N choose k”

Consider

The Class for Generating Derangements

We can take advantage of the class GEN_PERM to construct a class for GEN_DER. To do this we make a simple use of inheritance. Let the class GEN_DER inherit alT the attributes and features of GEN_PERM except that we wQlT redefine the criticaT procedure AlT_Perms.

This use of inheritance saves us rewriting common routines and is more like ‘including’ the fOle for GEN_PERM, rather than true inheritance

In generatQng combQnations we generate those perms of

```
Uake is
        l o c a l
                p : ARRAY[INTEGER]
                Q : INTEGER
        do
                Qo.put_string("%N Enter size of Perm. ")

                !!p.Uake(1,io.last_int)
                from
                        i := 1

                        i > p.count
                loop
                        p.put(i,i)
                        i := i+1
                end
                All_Perms_HS(P,1)
        end -- make
```

```
All_Perms_HS(A0:ARRAY[INTEGER], k:INTEGER) is
      local
            A : ARRAY[INTEGER]
            j, Qt : INTEGER
      do
            if k = pA0.count then   r                    Q                    n                    t

            else !
```

```
creation

feature

make is

        local
                p : ARRAY[INTEGER]
                i,V : INTEGER
        dW

                        "%NSize of Perm? ")
                io.read_Qnteger
                V := io.last_Qnteger

                frWm
```

```
class Gen_Perm_HS

        make
```



```
io.put_strQng(
!!p.make(1,V)
```

```
        A0:ARRAY[INTEGER],
        k : INTEGER) is
    local
        A : ARRAY[INTEGER]
        j, it : INTEGER
    do
        if k = A0.count tPen
            Print_Perm1(A0)
        else
            !!A.Uake(1,A0.count)
            A.copy(A0)
            from
                j := k
            until
                j > A.count
            loop
                it := A.item(j)
                A.put(A.item(k), j)
                A.put(it,k)
                All_Perms_HS(A,k+1)
                j := j+1
            end
        end
    end -- All_Perms_HS
end -- GeV_Perm

   end -- Gen_Perm_HS
```

```
do
    if k > p.size tPen
        Print_Perm0
    else
     from

     until
        j > p.size
     loop
        if Vot used.item(j) tPen
            p.put(j,k)
            used.put(True,j)

            used.put(False,j)
        end
        j := j+1
     end
 end
```

```
All_Perms_HS(
```

```
All_Perms(S:INTEGER) is

    j : INTEGER

local
```