Computer Physics Communications 31 (1984) 357–362
North-Holland, Amsterdam

MURNAGHAN'S RULE AND THE IRREDUCIBLE CHARACTERS OF THE SYMMETRIC GROUP

Ömer EĞECIOĞLU and Geraldo M. COSTA

Department of Mathematics, University of California, San Diego, La Jolla, CA 92093, USA

Received 10 July 1983

PROGRAM SUMMARY

Title of program: char

Catalogue number: AAMK

Program obtainable from: CPC Program Library, Queen's University of Belfast, N. Ireland (see application form in this issue)

Computer: VAX-11/780

Operating system: Unix

Programming language used: Pascal

High speed storage required: 600 words

Number of bits in word: 32

Peripheral used: console

Number of lines in combined program and test desk: 71

Keywords: symmetric group, irreducible representation, partition, Murnaghan's rule, character value

Nature of physical problem
Calculation of the irreducible characters of the symmetric groups.

Method of solution
Murnaghan's rule for the calculation of the irreducible characters of the symmetric group is implemented in a compact program.

Restrictions on the complexity of the problem
Limited only by available user memory.

Typical running time

$\chi_{(1\,1\,3\,7\,10)}(2\,4\,4\,6\,6)$ 17 ms,

$\chi_{(5\,5\,7\,8\,15)}(3\,4\,4\,4\,7\,9\,9)$ 266 ms.

References

[1] F.D. Murnaghan, The Theory of Group Representations (John Hopkin, Baltimore, 1938).
[2] G. James and A. Kerber, The Representation Theory of the Symmetric Group (Addison-Wesley, Massachusetts, 1981).
[3] D.E. Littlewood, The Theory of Group Characters, 2nd ed. (Oxford Univ. Press, London, 1950).

LONG WRITE-UP

0. Introduction

One of the well known ways of calculating the value of an irreducible character of the symmetric group at a given conjugacy class is the recursive formulation due to Murnaghan [1]. This formula appears in a variety of forms in the literature and we refer the reader to the refs. [2–4] for a detailed explanation of the theory behind it.

Calculation of all the entries in an arbitrary column of the character table of the symmetric group $\mathscr{S}_n$ (i.e., the values of all the irreducible characters at a fixed conjugacy class) can be realized by making use of the Frobenius formula [5]

$$\psi_\mu = \sum_\lambda \chi_\lambda(\mu) S_\lambda, \tag{0.1}$$

in conjunction with Murnaghan's rule where μ and λ are partitions of n, $\chi_\lambda(\mu)$ is the (λ, μ)th entry in the character table of $\mathscr{S}_n$, ψ_μ is the power symmetric function corresponding to the partition μ and S_λ is the Schur function corresponding to λ.

It turns out that the computation of a single entry $\chi_\lambda(\mu)$ of the character table of the symmetric group $\mathscr{S}_n$ can be carried out in a most efficient manner by a careful implementation of Murnaghan's rule.

We present an extremely compact routine (71 lines) which computes the character value of an irreducible character of $\mathscr{S}_n$ at a given conjugacy class. Even though our program has been coded in Pascal, it does not rely on any of the sophisticated data structures offered by Pascal other than the array and its ability to recurse.

As an example of the efficiency of this program, we remark that the calculation of the character value

$$\chi_\lambda(\mu) = -2,$$

where

$$\lambda = (3\ 3\ 4\ 4\ 5\ 6\ 6\ 7),$$
$$\mu = (2\ 7\ 7\ 9\ 13),$$

are partitions of $n = 38$, took 66 ms of cpu time in a VAX-11/780 minicomputer.

Fig. 1. Ferrers' diagram of $\lambda = (2\ 2\ 3)$.

1. Notation and the rule

We denote partitions by ascending order of the parts: $\lambda = (\lambda_1 \le \lambda_2 \le \ldots \le \lambda_k)$. Thus $\lambda = (2\ 2\ 3)$ is a partition of 7 into three parts. It's *shape* or *Ferrer's diagram* is the plane figure whose ith row consists of λ_i squares for $i = 1, 2, \ldots, k$ (fig. 1). Given a partition λ, the partition $\hat{\lambda} = (\hat{\lambda}_1 \le \hat{\lambda}_2 \le \ldots \le \hat{\lambda}_k)$, where $\hat{\lambda}_i = \lambda_i + (i-1)$ for $i = 1, 2, \ldots, k$ will be referred to as the *augmentation* of λ.

A *border strip* B of a partition λ is a consecutive strip of nodes on its rim with the following two properties:

(i) each pair of incident cells share an edge,
(ii) the removal of the nodes of B from the diagram of λ leaves a Ferrer's diagram.

If the number of nodes of a border strip is p then it is called a p-border strip. The *height* $\mathrm{ht}(B)$ of B is defined to be one less than the number of rows it occupies.

The partition $\lambda = (2\ 2\ 3)$ of fig. 1 has two 1-border strips, two 2-border strips, one 3-border strip, one 4-border strip and one 5-border strip with heights 0, 0, 0, 1, 1, 2 and 2, respectively, as shown in fig. 2.

We denote by $\lambda - B$ the partition obtained from λ by removing from its Ferrers' diagram the nodes belonging to the border strip B.

Let $\lambda = (\lambda_1 \le \lambda_2 \le \ldots \le \lambda_k)$ and $\mu = (\mu_1 \le \mu_2 \le \ldots \le \mu_l)$ be two partitions of n. Murnaghan's rule state that the value of the irreducible character of $\mathscr{S}_n$ corresponding to the partition λ at the

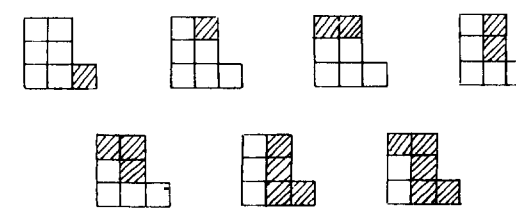

Fig. 2. Border strips of $\lambda = (2\ 2\ 3)$.

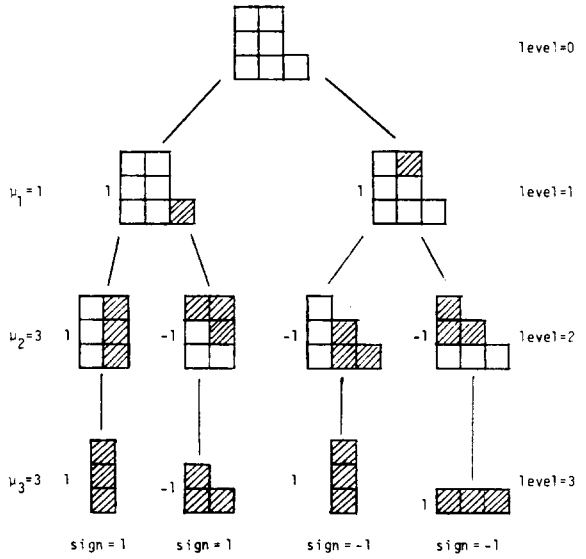

Fig. 3. Calculation of $\chi_{(2\,2\,3)}(1\,3\,3)$.

conjugacy class of permutations with cycle type μ is given by

$$\chi_\lambda(\mu) = \sum_{B_1, B_2, \ldots, B_l} (-1)^{\mathrm{ht}(B_1)+\mathrm{ht}(B_2)+\ldots+\mathrm{ht}(B_l)}, \tag{1.1}$$

where the summation is over all l-tuples $(B_1, B_2, \ldots, B_l)$, where B_i is a μ_i-border strip of $\lambda - (B_1 + B_2 + \ldots + B_{i-1})$ for $i = 1, 2, \ldots, l$ and $B_0 = \emptyset$.

For example, when $\lambda = (2\,2\,3)$ and $\mu = (1\,3\,3)$, the value of $\chi_\lambda(\mu)$ is simply the sum of the signs attached to the terminal nodes at level 3 (= number of parts of μ) of the tree in fig. 3, where the sign of a terminal node is the product of the signs of all the nodes on the path from the root to the terminal node.

Thus by (1.1) the character value is

$$\chi_{(2\,2\,3)}(1\,3\,3) = 1 + 1 - 1 - 1 = 0.$$

2. Computer implementation

2.1. The general method

Even though the generation of the tree in fig. 3 is relatively straightforward, in the general case a substantial amount of overhead comes from detecting the existence and the sign of a border strip of a given size in λ.

It turns out that this can be done with a minimal amount of search by making use of the augmentation of λ as follows:

Suppose we wish to locate the bottommost rows of p-border strips of λ. It is possible to remove a p-border strip that starts at the ith row of λ (ith row from the top of the diagram of λ) if and only if one of the following two conditions are satisfied:

$$\begin{aligned} &\text{(a) } \hat{\lambda}_i = p, \\ &\text{(b) } \hat{\lambda}_i - p \neq \hat{\lambda}_j \quad \text{for any } j,\ 1 \le j \le i. \end{aligned} \tag{2.1}$$

Moreover, if this is the case then the height of the border strip is $i - j$ where j is the largest index such that

$$\hat{\lambda}_i - p < \hat{\lambda}_j.$$

For example if $\lambda = (2\,2\,3)$ and $p = 2$ then $\hat{\lambda} = (2\,3\,5)$ has the rows $\hat{\lambda}_1 = 2$ and $\hat{\lambda}_2 = 3$ which satisfy the conditions (a) and (b) of (2.1), respectively. Thus, λ has two 2-border strips starting at the first and the second rows, in agreement with the list in fig. 2.

2.2. Overview

The program recursively generates a tree where each node is an array that contains the parts of a partition obtained from λ by successively removing border strips of lengths $\mu_1, \mu_2, \ldots$ corresponding to the parts of μ. Each node created carries the sign of the border strip that has been removed from it.

The partitions λ and μ are read from the input file and the augmentation of λ (the array LAMBDA) is generated by the main driver of the program.

The tree of partitions is constructed by the procedure CALCULATE recursively using backtracking. The partitions generated by removing μ_i-border strips, the associated signs and the current level of the tree are passed as arguments. Once a terminal node of the tree is reached, the values of the signs are accumulated in the variable CHARVALUE.

```
function charvalue(lambda, mu: vector; plambda, pmu: integer): integer;

var  i, value: integer;
(*********************************************************************)

  function notexist( parts: vector; size, i: integer): boolean;

  var  half, inf, sup : integer;
  begin
    inf := 1;  sup := size;
    repeat
      half := (inf + sup) div 2;
      if i > parts[ half ] then inf := half + 1
          else sup :=  half - 1
    until ( parts[ half ] = i ) or ( inf > sup );
    if parts[ half ] <> i then  notexist := true
       else  notexist :=  false
  end; { notexist }
(*********************************************************************)

  procedure create( var newpart: vector; var newsize: integer;
                    part: vector; remove, bstrip, partsize: integer);

  var  i, k : integer;
  begin
    newpart[ 0 ] := -1;   part[ 0 ] := 0;
    for k := partsize downto 1 do
      newpart[ k ] := part[ k ];
    while bstrip > 0 do begin
      if part[ remove ] - part[ remove - 1 ] >= bstrip then
          newpart[ remove ] := part[ remove ] - bstrip
        else  newpart[ remove ] :=  part[ remove - 1 ];
      bstrip := bstrip - part[ remove ] + part[ remove - 1 ];
      remove := remove - 1;
      newpart[ 0 ] := - newpart[ 0 ]
    end;
    while (newpart[ k ] <= (k-1)) and (k <= partsize) do  k:=k+1;
    if k > 1 then  for i := k to  partsize do
       newpart[ i - k + 1 ] := newpart[ i ] - k + 1;
    newsize := partsize - k + 1;
  end; { create }
(*********************************************************************)

  procedure calculate(lambda: vector;plambda,level,sign: integer);

  var  newsize, i: integer;
       newlambda : vector;
  begin
    if level = pmu + 1 then  value := value  + sign
       else  for i := plambda downto 1 do
                if lambda[ i ] >= mu[ level ] then
                   if notexist(lambda, i, lambda[i] - mu[level]) then
                      begin
                        create(newlambda, newsize,lambda, i, mu[level],
                                plambda);
                        calculate(newlambda, newsize, level + 1,
                                  newlambda[0]*sign)
                      end
  end; { calculate }
(*********************************************************************)
  begin  { charvalue }
    for i := 1 to plambda do
       lambda[ i ] := lambda[ i ] + (i - 1);
    lambda[ 0 ]  := +1; value := 0;
    calculate(lambda, plambda , 1, lambda[ 0 ]);
    charvalue := value;
  end; { charvalue }
```

Fig. 4. The listing of function charvalue.

2.3. Description of the procedures

The names and brief descriptions of the procedures and functions are as follows:

NOTEXIST

Uses binary search to detect whether or not a border strip of length SIZE exists at row I of the input partition PARTS using the criteria outlined in section 2.1.

CREATE

Removes a border strip of length BSTRIP from the input partition LAMBDA that starts at row REMOVE. Thus this procedure creates the nodes in the next level of the tree (as in fig. 3). The sign of the partition produced is computed as indicated in section 2.1.

CALCULATE

Recursively constructs the partitions at each level of the tree and propagates the signs of the partitions along the different paths generated. The character value is computed by accumulating the signs of the terminal nodes in the variable CHARVALUE. The input parameters to procedure CALCULATE are as follows:

(i) The array LAMBDA that contains the augmentation of the partition λ and its sign in position LAMBDA [0].
(ii) LEVEL: the level of the partition-tree.
(iii) SIGN: the sign that has been propagated so far.
(iv) PLAMBDA: the number of parts of the input partition.

2.4. I/O format

The user is prompted to enter the partitions λ and μ, respectively. The parts are to be entered in ascending order of magnitude. The input file is expected to be a list formatted as

$\lambda_1\lambda_2\ldots\lambda_k\langle\mathrm{cr}\rangle$

for the partition λ corresponding to the irreducible representation and

$\mu_1\mu_2\ldots\mu_l\langle\mathrm{cr}\rangle$

for the partition μ of the conjugacy class where the entries are separated by blanks and $\langle\mathrm{cr}\rangle$ denotes carriage return.

The corresponding character value is written to the output file by the main program.

3. Remarks

The correctness of this implementation has been checked by making use of the character tables of the symmetric groups $\mathcal{S}_n$ for up to $n = 10$, that appear in ref. [2].

Since the calculation of the character values is necessary as a function call in a number of computer applications, the listing of the program char as a Pascal function (named charvalue) is reproduced in fig. 4. The arguments of the function charvalue are the partition of the representation LAMBDA, the partition of the conjugacy class MU, the number of parts PLAMBDA of LAMBDA and the number of parts PMU of MU. The two partitions are of type vector which is

array [O..maxparts] of integer

which must be declared in the main body of the program that uses the function

charvalue.

Acknowledgement

The second author was supported by a fellowship from Conselho Nacional de Pesquisa (CNPq) of Brasil.

References

[1] F.D. Murnaghan, The Theory of Group Representations (John Hopkin, Baltimore, 1938).
[2] G. James and A. Kerber, The Representation Theory of the Symmetric Group (Addison-Wesley, Massachusetts, 1981).
[3] D.E. Littlewood, The Theory of Group Characters, 2nd ed. (Oxford Univ. Press, London, 1950).
[4] Ömer Eğecioğlu, A Combinatorial Derivation of Murnaghan's Rule for the Irreducible Characters of the Symmetric Group (preprint, UCSD).
[5] G. Frobenius, Über die Charaktere der Symmetrischen Gruppe (Sitz. Ber. Preuss. Akad., Berlin, 1900) p. 516.
[6] K. Jensen and N. Wirth, Pascal User Manual and Report (Springer-Verlag, New York, 1975).

TEST RUN OUTPUT

```
Enter the  representation in ascending order :
1 3 3
Enter the conjugacy class in ascending order :
1 1 1 1 1 1
The character value  is          21

( CPU time : 50  msec )



Enter the  representation in ascending order :
1 2 2 3
Enter the conjugacy class in ascending order :
2 3 3
The character value  is          -1

( CPU time : 8  msec )



Enter the  representation in ascending order :
1 2 2 5
Enter the conjugacy class in ascending order :
1 1 3 5
The character value  is           0

( CPU time : 16  msec )



Enter the  representation in ascending order :
1 1 3 7 10
Enter the conjugacy class in ascending order :
2 4 4 6 6
The character value  is          -2

( CPU time : 17  msec )



Enter the  representation in ascending order :
3 3 4 4 5 6 6 7
Enter the conjugacy class in ascending order :
2 7 7 9 13
The character value  is          -2

( CPU time : 66  msec )



Enter the  representation in ascending order :
5 5 7 8 15
Enter the conjugacy class in ascending order :
3 4 4 4 7 9 9
The character value  is         -24

( CPU time : 266  msec )
```