Hungarian Algorithm (9 pages; 28/8/17)

(1) **Example**: 3 workmen (A, B & C) are to carry out 3 tasks (P, Q & R). They each take certain times (in minutes) for the 3 tasks, as shown in Table 1.

	P	Q	R
A	9	8	6
B	11	8	7
C	10	8	7

Table 1

The aim is to allocate the workers to the tasks, in such a way as to minimise the total time taken. This is an example of an **allocation** (or **assignment**) **problem.**

Assuming that a solution is not obtainable by inspection (which would be more difficult for larger arrays), we can convert the problem to an equivalent simpler one by reducing all the elements of row A by 6 (so that the smallest element is 0); all the elements of row B by 7, and all the elements of row C by 7. Thus it is the relative times that are important.

After **reducing by rows** in this way, the new table is shown in Table 2.

	P	Q	R
A	3	2	0
B	4	1	0
C	3	1	0

Table 2

Our aim is to have enough zeros in the table, so that each workman can be matched to a task with zero time.

For this example, this will be possible if and only if all the zeros in the table can be covered by crossing out a total of 3 rows and/or columns (eg 2 rows and 1 column, or 3 columns), but not by crossing out a smaller number. In general, the desired number of rows/columns to be crossed out is the **size** of the **array** (assumed to be square for the moment); ie the number of rows or columns.

In Table 2, this isn't yet the case, as only 1 column is needed to cover the zeros. However, we can also reduce the table by columns (in the same way), to give Table 3.

	P	Q	R
A	0	1	0
B	1	0	0
C	0	0	0

Table 3

Now 3 rows/columns are needed (3 rows or 3 columns), and the following solutions can be noted (Table 4):

(a)

	P	Q	R
A	**0**	1	0
B	1	**0**	0
C	0	0	**0**

(b)

	P	Q	R
A	**0**	1	0
B	1	0	**0**
C	0	**0**	0

(c)

	P	Q	R
A	0	1	**0**
B	1	**0**	0
C	**0**	0	0

Table 4

The total times associated with these solutions are found by referring to the original table. Thus, for (a) it is $9 + 8 + 7 = 24$, for (b) it is $9 + 7 + 8 = 24$, and for (c) it is $6 + 8 + 10 = 24$

Note

Had all the times been 10 times greater, then we could have divided them all by 10, without changing the nature of the problem - we have just changed its scale. (But note that we cannot divide one row's elements by 10, leaving the other rows untouched. Thus in Table 5b, looked at from the point of view of task P, A's time has been reduced by 81, whilst from the point of view of task Q it has been reduced by only 72; thus workman A is now more likely to be matched with task P, so that the nature of the problem has changed.

	P	Q	R
A	90	80	60
B	110	80	70
C	100	80	70

Table 5a

	P	Q	R
A	9	8	6
B	110	80	70
C	100	80	70

Table 5b

(2) In exam questions, it is normally specified that (say) the rows should be reduced before the columns, as the outcome is generally different. In this example, had we reduced by columns first we would have obtained Table 6.

	P	Q	R
A	0	0	0
B	2	0	1
C	1	0	1

Table 6

and now there is no scope to reduce by rows. However, the array only requires 1 row and 1 column to cover the zeros, and hence no solution can yet be obtained. (But note that there must be a minimum time, and so a solution does exist.)

In order to find the solution, we employ the **Hungarian algorithm.** This involves back-tracking a bit, so that some of the elements of the array are increased, to enable others to be reduced. The net effect will be that there are more reductions than increases, so that we are making progress in simplifying the problem, and eventually creating more zeros.

We already know that, for this example, the solutions can be represented as follows (Table 7):

(a)

	P	Q	R
A	**0**	0	0
B	2	**0**	1
C	1	0	**1**

(b)

	P	Q	R
A	**0**	0	0
B	2	0	**1**
C	1	**0**	1

(c)

	P	Q	R
A	0	0	**0**
B	2	**0**	1
C	**1**	0	1

Table 7

In each case one of the 1s is being used; ie the smallest value apart from the zeros.

We want to manipulate the array, by adding or subtracting a certain amount to/from all the elements in a particular row or column (as this won't change the nature of the problem), in such a way that the 1s can be reduced to zeros.

This is done by crossing out the necessary rows/columns, as before, in order to cover up the zeros, and then adding 1 (in general, the smallest non-zero value in the array) to each element in a crossed out row or column. This means that any element that is an intersection of a crossed out row or column is increased twice. Now there will be no zeros, and we can reduce every element of the array by 1, with the effect that the three 1s in Table 6 now becomes zeros; as do the zeros in Table 6, except for element AQ (which isn't needed for any of the solutions: it would require more than one of the 1s to be used, and so couldn't produce the minimum possible cost).

The 1st stage of the procedure was to cross out the rows/columns needed to cover the zeros. In general, there could be more than one way of doing this, and it is best to choose the way that gives rise to the largest non-zero element amongst the elements that aren't crossed out, as this will reduce the array the most. Table 8 shows the crossed out row and column (when handwritten, a line would be drawn over the 3 elements in the relevant row and column).

	P	Q	R
A	0	0	0
B	2	0	1
C	1	0	1

Table 8

The 2nd stage of the procedure was to add 1 in respect of each crossing out (so that 2 is added to the intersecting element AQ), to give Table 9:

	P	Q	R
A	1	2	1
B	2	1	1
C	1	1	1

Table 9

Then 1 was taken off every element, to give Table 10:

	P	Q	R
A	0	1	0
B	1	0	0
C	0	0	0

Table 10

Now 3 rows/columns are needed to cover the zeros, so we have finished (apart from identifying the matches). Table 10 is identical to Table 3, obtained by first reducing by rows.

Notes

(i) In general, it may be necessary to repeat the process of reducing by rows and/or columns after the Algorithm procedure has been applied.

(ii) When the arrays are large, so that matches may not be found by inspection, the **Maximum Matching** algorithm for bipartite graphs could be used instead (see separate note).

(iii) A shortcut can be applied to the Algorithm procedure, by noting that the overall effect is to reduce the uncovered elements by 1 (for the above example) and increase any intersecting elements by 1.

Summary of the Hungarian algorithm

(i) Reduce the array by rows and columns (having divided all elements by the same number, if possible).

(ii) Cover the zero elements with as many rows and/or columns as are necessary. If the number of covered rows/columns equals the size of the array, then we can find a solution; ie workmen and tasks are matched up so that there is a zero time in each case (in the reduced array). The actual times are then found by referring to the original array.

(iii) If the number of covered rows/columns is less than the size of the array, then we establish the smallest non-covered element: S (say). This value S is then added to each element of a covered row or column (so adding twice for any intersections), and finally S is subtracted from every element of the array. (Alternatively, we can employ the shortcut mentioned above.)

(iv) Repeat the process.

(3) Variations

(i) Problems where the total amount needs to be maximised, rather than minimised.

In this case, we can just multiply each element by -1, and then minimise as before. We can then add the largest value in the original array to every element. This has the effect of removing minus signs and reducing the sizes of values.

(As an alternative to adding the largest value to every element, the minus signs could be eliminated on a row by row (or column by column) basis by adding the necessary amount to each element of the row (column).)

(ii) Non-square arrays.

These are dealt with very easily, by adding in one or more dummy rows or columns, to create a square array. The elements in each dummy row or column are all the same, so that the dummy has no effect on the nature of the problem. The dummy values are often taken to be the largest value in the array, so that they won't hinder the necessary row or column reductions. (A dummy value of zero is sometimes used though. If it is a dummy column (say), then we will not be able to reduce the array by rows.)

(iii) It may be the case that some of the allocations (eg a particular workman to a particular task) are not allowed. This can be dealt with easily by inserting a large number (relative to the values in the array; eg twice the largest value) as the relevant elements. This makes these elements unattractive, and so they won't appear in the solution(s).