Test Cover

In this project you are going to deal with the Test Cover Problem. A possible domain of the problem is the following: Assume that you want to be able to distinguish between a number of diseases, given a set of tests. Each test will give a positive result for some of the diseases and negative for the rest. Also, there is a cost associated with performing each test. The question is to choose a subset of tests of minimum total cost, so that for any two diseases, there is a test in the subset that can tell them apart.

In general, an instance of the Test Cover Problem is described by a cost vector **c** of size n, and an n by m matrix **A**, where n is the number of tests, m the number of diseases, and $A_{ij} = 1$ if test i is positive for disease j.

We next give a toy example for illustrative purposes. As you can see, in this case one doesn't have to include the test for fever, since it's common in all three diseases. It's easy to see that, by assuming costs 1, 1, 1, 1 for checking for the various symptoms, an optimal solution would be to just check for rash and cough.

	Flu	Measles	Chickenpox
Rash	0	1	1
Fever	1	1	1
Cough	1	1	0
Headache	1	0	1

Your task is to formulate the Test Cover Problem as an IP problem, and solve it using the Branch and Bound technique. You can use CPLEX as a Linear Programming Solver for the linear relaxations arising in the computation of Lower Bounds, but you *cannot* use it as an Integer Programming Solver. Note that we are asking for the exact solution, not just a good lower bound, as was the case with the ARP problem.

Support Code

The support code provides an implementation of a Min-Heap Data Structure, which you are free to use, in either its current or a modified version. Please, check file `heap.H` for more information about the Min-Heap interface. Of course, you can always choose different data structures for implementing your solver, as long as it's efficient enough for your purposes.

The code also provides an outline of the project's structure, including a function to read data from a file describing an instance of the problem. The function assumes the following format for the instance file:

```
#Tests     (i.e., n)
#Diseases  (i.e., m)
Cost_1 Cost_2 . . . Cost_n
A(1,1) A(1,2) . . . A(1, m)
A(2,1) A(2,2) . . . A(2, m)
. . . . . . . . . . . . . .
A(n,1) A(n,2) . . . A(n, m)
```

For instance, the above toy example can be stored in a file of the form

```
4
3
1 1 1 1
0 1 1
1 1 1
1 1 0
1 0 1
```

When you are ready to begin coding, copy the support code from
`/course/cs149/asgn/testCover/`
You can compile by typing:
```
cslab0a /course/cs149/asgn/testCover% make
```
and run by typing:
```
cslab0a /course/cs149/asgn/testCover% ./tcover
```
A test set of representative instances is located at:
`/course/cs149/asgn/testCover/instances`
You may want to try to create your own instances, so that you can build some intuition on the problem.

Output

In order to make grading easier, we ask that you print the optimal objective value on the last line in the following format:
```
Objective:  123
```
Where 123 is the objective value.