1 Organization

The second programming assignment (taken from [http://www.spoj.com/problems/KNAPSACK/](http://www.spoj.com/problems/KNAPSACK/)) is concerned with the 0-1 knapsack problem and will count for 15\% of your final grade. It consists of two parts. The first part can give you a grade up to 7. You can optionally also do the second part, which can bring your grade up to 10.

For the first part you will be (mechanically) graded based on the number of successfully passed test cases. A test case is considered successfully passed if your program gives the correct answer and it halts before the time limit.

For the second part you will be graded based on the quality of your implementation and your report.

You are allowed to work in a group of up to two students. Of course close collaboration between different groups is not allowed. Groups that hand in similarly structured programs will be questioned.

The deadline for handing in your assignment is Friday **December 15 at 5PM.** This deadline is *hard as nails.* (You are only allowed to hand in your assignment once, in principle revisions of programs will not be considered.) Email your assignment to p.vukmirovic@vu.nl.
2 First part: Dynamic programming

You are packing for a vacation on the sea side and you are going to carry only one bag with capacity \( S \) (\( 1 \leq S \leq 2000 \)). You also have \( N \) (\( 1 \leq N \leq 2000 \)) items that you might want to take with you. Unfortunately you cannot fit all of them in the knapsack, so you will have to choose. For each item you are given its size and its value. You want to maximize the total value of all the items you are going to bring. What is this maximum total value?

2.1 Input

The program will get the data from standard input\(^1\). On the first line you are given \( S \) and \( N \), separated by spaces. \( N \) lines follow with two integers on each line describing one of your items. The first number is the size of the item and the next is the value of the item.

Reading input from any other source will cause your program to fail the test case.

2.2 Output

You should output a single integer - the total maximum value from the best choice of items for your trip.

Any other output (such as a prompt for user to enter data or a debug printing) will cause your program to fail the test case.

2.3 Time limit

Your program will be terminated after five seconds. Beware that this is a strict time limit and that you will have to make sure your program is efficient. Note that brute-force solution will inevitably lead to a timeout. Thus, you are asked to implement the dynamic programming solution, which is explained on the slides (of lecture 3).

2.4 Test cases

A test case is given to make sure you understand the problem. On the left the input to the algorithm is given, and on the right the output. You are

\(^1\)http://www.linfo.org/standard_input.html
strongly advised to test the program with test cases of your own!

Test case 1

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>13</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

More test cases

A full-length test case will be available on the website. The purposes of this test case are to test the runtime of your program, and to guarantee that your program accepts input in the required form. Note that you can test the execution times of your program using the `time` command.

3 Second part: Branch and bound

For the second part of the assignment you are asked to implement a solution to the 0-1 knapsack problem (as described above) that uses the branch-and-bound technique (which is explained in lecture 13). Alongside your program, you are asked to provide a small report in which you reflect on implementation issues you faced as well as an evaluation of performance of your implementation on a benchmark of test cases. Specifically, you should compare run times of the branch-and-bound version to the dynamic programming version of your solution. Why do you think one is faster than the other? Are there any specific constraints that you have put on the input problem in one implementation, that are lifted in the other? – here I want to make them see that in DP solution they are basically constraining the capacity to be of reasonable size. How do worst-case time complexities compare and what do you observe in the empirical analysis?

3.1 I/O

There are no constraints on the I/O for this part of the assignment since it will not be mechanically tested. But it is strongly recommended to read the
data in this second part of the assignment the same way you did in the first part, since you can reuse code in that way and you will be given test cases in that format.

### 3.2 Test cases

A few test cases of reasonable sizes will be available on the website.

*Good luck!*