Introduction to Algorithms	**Problem Set 9 (3 problems)**
CS 4820 Fall 2019	**Due 11:59pm Thursday, December 5**

There are three questions on this homework, a coding problem and two written questions.

Your homework submissions for written questions need to be typeset (hand-drawn figures are OK). See the course web page for suggestions on typing formulas. The solution to each written question needs to be uploaded to CMS as a separate pdf file. To help provide anonymity in your grading, do not write your name on the homework (CMS will know it's your submission).

Solutions to the coding problem need to be submitted to the **online autograder** at `https://cs4820.cs.cornell.edu/`.

Collaboration is encouraged while solving the problems, but

1. list the names of those with whom you collaborated with (as a separate file submitted on CMS);
2. you must write up the solutions in your own words;

Remember that when a problem asks you to design an algorithm, you must also prove the algorithm's correctness and analyze its running time.

(1) Knapsack Approximation. (10 points) The KNAPSACK problem discussed in class on Friday, November 22th is given by n items each with a value $v_i \geq 0$ and weight $w_i \geq 0$, and a weight limit W. The problem is to find a subset I of items maximizing the total value $\sum_{i \in I} v_i$ while obeying the weight limit $\sum_{i \in I} w_i \leq W$. You may assume that $w_i \leq W$ for all i, that is, each item individually fits in the knapsack.

This coding problem is asking you to design an efficient algorithm to approximately solve this problem. Your algorithm needs to find a solution that obeys the weight limit with total value of at least 1/2 of the maximum possible. I.e., you will receive full credits if your algorithm achieves an approximation ratio at most 2.

We are also offering you the opportunity to try to get a better solution, if you want to try. Random (or typical) instances of knapsack problems tend to be rather easy to solve to very close to optimal values. Here we are using a library of hard instances to test your program. We will use the following input sizes: $n \leq 10,000$, $v_i, w_i \leq 100,000$, $W \leq 10,000,000$. You should make sure that your program runs in less than 1 second on every test case provided.

We will offer you test where you can see if you can achieve an approximation ratio of 1.2 on the test cases, or if you can achieve an approximation ratio of 1.1 of the optimum. Since this is an NP-complete problem, it is not clear how to do this. You can use any algorithm that you like - just be careful with the time limit. Section 11.8 of the book offers a close approximation, which will not be covered in class; but note that the algorithm discussed there is slow and needs tons of space (as dynamic programs usually do). You may be better off with simple heuristics along the lines suggested by problem (2).

The only libraries you are allowed to `import` are the ones in `java.util.*`

Warning: be aware that the running time of calling a method of a built-in Java class is usually not constant time, and take this into account when you think about the overall running time of your code. For instance, if you used a LinkedList, and use the `indexOf` method, this will take time linear in the number of elements in the list.

We have set up an **online autograder** at `https://cs4820.cs.cornell.edu/`. You can upload solutions as many times as you like before the homework deadline.[1] **You need to have the main method of your code named Main, must not be "public", must not be part of a package**. When you upload a submission, the autograder will automatically compile and run it on a number of

[1]The deadline shown on the auto-grader is the latest time at which you will be allowed to submit your code, including the maximum number of slip days. If you submit your code past the actual deadline, you will automatically be charged slip days accordingly, so please be careful.

public test-cases[2] that we have prepared for you, checking the result for correctness and outputting any problems that may occur. You should use this facility to verify that you have interpreted the assignment specification correctly. After the deadline elapses, your last submission will be tested against a new set of *private* test cases, which your grade for the assignment will be based on.

Input file format: Your algorithm is to read data from `stdin` in the following format:

- The first line has two integers, $n \leq 10,000, W \leq 10,000,000$, the total number of items and the weight limit.
- In the i-th line (let $i = 0, \ldots, n-1$) of the next n lines, there are two numbers $v_i, w_i \leq 100,000$, the value and weight of item i respectively.

Output file format: Your algorithm should output data to `stdout` in the following format:

- The first line contains one number, V, which is the total value obtained by your solution,
- In the next n lines, you should output a binary representation of your solution, where one bit in each line: 1 means you picked item i and 0 means you didn't pick that item (for $i = 0, 1, \ldots, n-1$).

Testing your code: An output file in a public test case will contain an optimal solution (in the above format) to that case. It's totally fine if your output is inconsistent with it. To test your code, you should compute your approximation ratio by comparing the first line of your output with the first line of provided output. We will accept your solution as long as your approximation ratio is no more than 2. Note that your solution should also be valid, which means the total weight of the items you select should be no more than W.

To offer you a chance to test if your algorithm achieves better approximation ratio, there will be three assignments for this problem on the auto-grader. Each of them has a different threshold of approximation ratio (2, 1.2, and 1.1). You can pass tests in one of the assignments if your approximation ratio is no more than than its threshold. We will only grade your code on the assignment with threshold 2.

(2) Knapsack. (10 points) In class on Friday Nov 22nd we have studied the greedy approximation algorithm for the knapsack problem: the problem had n items with weight w_i and value v_i, and weight limit W. The goal is to select a subset $I \subset \{1, \ldots, n\}$ such that $\sum_{i \in I} w_i \leq W$ and maximizing the total value $\sum_{i \in I} v_i$. We saw an $O(n \log n)$ running time 2-approximation algorithm for this problem by taking the better of two different greedy solutions. Here we want to consider a different approximation: ensuring the total value of selected item is greater than the maximum value possible with weight limit W by allowing the total weight of the selected items to go somewhat beyond W.

(a) (5 points) Assume that no item is too big, that is $w_i \leq W/2$ for all i. Using this assumption, give a polynomial-time algorithm that finds a solution with the following guarantee: Suppose the maximum value possible with weight limit W is V^*, then the algorithm needs to output a subset I of items, such that $\sum_{i \in I} w_i \leq \frac{3}{2} W$ and $\sum_{i \in I} v_i \geq V^*$.

(b) (5 points) Give an algorithm with the guarantee it part (a) even if the input has big items.

(3) Vertex Cover on Hyper-graphs. (10 points) The HITTING SET problem is defined as follows: given a set V of nodes with weights w_i for $i \in V$, and a list of subsets $\mathcal{H}$ whose elements are subsets of V, and an integer k. A *hitting set* S is a subset $S \subset V$ such that $S \cap H \neq \emptyset$ for all sets $H \in \mathcal{H}$. the *minimum cost hitting set problem* is to find a hitting set S of minimum total weight $\sum_{i \in S} w_i$.

Note that when all sets $H \in \mathcal{H}$ have exactly 2 elements, then the sets in $\mathcal{H}$ form the edges in a graph G, and the hitting set problem is the vertex cover problem on the graph G. (This also shows that the HITTING SET problem is NP-hard).

(a) (5 points) For the special case of VERTEX COVER we have seen a 2-approximation algorithm in class on Monday, November 25. Show that the bound of 2 we gave comparing the true optimum

[2] They will also be available on CMS.

and the optimum for the value of the linear program used in the approximation algorithm (on page 634) is best possible. Concretely, show that for any constant $\gamma < 2$ there is a vertex cover problem with weights w_i such that the linear programming optimum is more than a factor of γ smaller than the true optimum.

(b) (5 points) Now consider the general case of the HITTING SET, and assume that the maximum size of a set of $\mathcal{H}$ is $\max_{H \in \mathcal{H}} |H| \leq c$. Give a c-approximation algorithm for this case of the min-weight HITTING SET problem.