Introduction to Algorithms **Problem Set 9**
CS 4820 Spring 2016 **Due 11:59pm Sunday, May 1st**

Your homework submissions need to be typeset (hand-drawn figures are OK). See the course web page for suggestions on typing formulas. Solution to each question needs to be uploaded to CMS as a separate pdf file. (Questions with multiple parts need to be uploaded as a single file.)

1. *(10 points) with option for bonus points* The KNAPSACK problem discussed in class on Wednesday, April 20th (see also Section 11.8 of the book, which will be covered Monday, April 24th) 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. For full credit, your algorithm needs to find a solution with total value of at least 1/2 of the maximum possible. You can get bonus points (which will count towards your homework total) proportional to the quality of the solution you find, if your solution is better than 1/2 the maximum value. Specifically, for each test case, let $\beta = \frac{\text{optimal solution}}{\text{your solution}}$, you will get 13 points if $\beta <= 1.2$, will get 18 points if $\beta <= 1.1$, and will get 20 points if $\beta <= 1.01$. 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.

Here are the size of the input. $n \leq 10,000$, $v_i, w_i \leq 100,000$, $W \leq 10,000,000$. You will have time limit of 1 second for each test case. We provide two biggest test cases for you, and you need to make sure your program can solve them in 1 second on your computer. We will use a powerful desktop computer for testing your program. It is unlikely that your program runs in less than 1 second in your computer but gets time limit exceeded error in our computer. If that happens, you can always bring your computer to us, and we will use your computer to test the program again. So don't worry about the difference between your computer and test computer.

Since this is a NP-complete problem, we don't have specific algorithm requirements for you. You can use any algorithms that you like, just be careful with the time limit.

For the code, you must use the framework code (*Framework.java*) we provide on CMS. We will put the code you have submitted in the part of the framework that we have specified by the commented line
//YOUR CODE GOES HERE
and run the code.

We require you to submit both *Fragment.txt* and *Framework.java.* We will first test your program using *Fragment.txt*, and if it fails to compile, we will use your *Framework.java* file (which is a whole java program that **includes** your *Fragment.txt*, and works well on your own computer) to test instead. However, you will get at most 5 out of 10 points if your *Fragment.txt* file fails to compile, even if your *Framework.java* file passes all the test data.

Please read the provided framework carefully before starting to write your code. You can test your code with the test cases provided on CMS. *Framework.java* will take two command line arguments. The first one is the name of the input file, and the second is the name of the output file. The input file should be in the same folder in which your compiled java code is. After you compile and run your code, the output file will also be in the same folder. In order to test your code with the provided test cases, copy the test cases in the folder in which you have compiled your code, and set the name of the input file to be the name of one of the sample inputs (*SampleTest_i.txt* in which $0 \leq i \leq 5$). You can compare your output with the provided sample outputs (*SampleOutput_i.txt* in which $0 \leq i \leq 5$).

The format of the input file is the following:

- The first line has two numbers, n, W, which are the number of items, and the weight limit respectively.

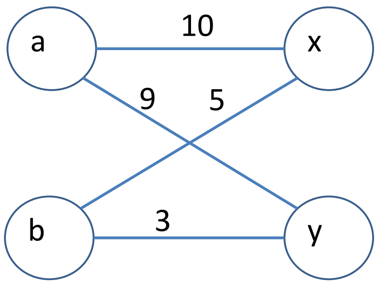
- In the i-th line (let $i = 0, \cdots, n-1$) of the next n lines, there are two numbers v_i, w_i, denoting value and weight of item i.

The format of the output file is the following:

- The first line has one number, V, which is your total value.
- In the i-th line (let $i = 0, \cdots, n-1$) of the next n lines, you either output 1 or 0, where 1 means you picked item i and 0 means you didn't pick that item.

Make sure your total value matches with your item selection. The output file will be considered wrong if total value and item selection don't match.

2. *(10 points)* We have studied the matching problem, given a bipartite graph $G = (V, E)$, we have seen how to find in G a matching $M \subseteq E$ of maximum size, using flows. On Wednesday, April 20th we have seen that a simple online algorithm can get a matching of size at least 1/2 of the maximum size. In this problem, we consider a greedy algorithm for a weighted version of this problem: suppose each edge e has a value $v_e \geq 0$, and we are looking for a matching M of largest total value $\sum_{e \in M} v_e$. For example, the nodes of G are students and possible internships, then v_e for an edge (u, v) connecting a student u with an internship v can represent the value that the student can produce if he/she takes this internship. The goal now becomes to find a matching of largest total value. There are in fact polynomial time algorithms that can do this (see section 7.13 of the book if you are interested). Here, we will consider the following greedy algorithm: start with the empty matching $M = \emptyset$. Consider the edges in decreasing order of value v_e. When edge $e = (u, v)$ comes up in the order, add e to the matching if at this point neither u nor v is matched.



For example, in the graph of the figure, the maximum value matching uses the two edges (a, y) and (b, x) which has total value 9+5=14. The greedy algorithm considers edges in the order $(a, x), (a, y), (b, x), (b, y)$. Takes edges (a, x) and then can only take (b, y), resulting in a total value of only $10 + 3$.

Show that the greedy algorithm has approximation ratio 2 but not better by doing the following two things:

(a) Give an example when the greedy algorithm finds a matching that has a factor of 2 smaller value than the maximum possible. (To achieve this, you may order equal value edges in any order you want.)

(b) Show that the greedy algorithm is guaranteed to return a solution that has value at least 1/2 of the maximum possible value.

3. *(10 points)* A natural application of center selection (of Section 11.2) is when we have n locations (say houses in a large rural settlement). We select k centers to have a form of rescue center, say a fire station, with the goal to minimize the maximum distance between a house and the closest rescue center.

However, it turns out that only certain locations are viable for rescue centers. In the new version of the problem we are given n locations S, and a set of m possible center locations A, and are given distances between all pairs of locations. We will assume that distances between the locations $S \cup A$ satisfy the properties on page 606 (of being a symmetric distance function).

The problem is to select a subset $C \subset A$ of the possible center locations of $|C| = k$ to minimize the maximum distance of a location in S from the closest selected center: $\max_{s \in S} \min_{c \in C} d(s, c)$.

Given a distance limit r, it turns out to be NP-complete to decide if there is a set $C \subset A$ of size k with $\max_{s \in S} \min_{c \in C} d(s, c) \leq r$. Give a polynomial time algorithm that either concludes that no such set C exists, or finds a subset $C \subset A$ of k points with $\max_{s \in S} \min_{c \in C} d(s, c) \leq 3r$.