Introduction to Algorithms	Problem Set 3
CS 4820, Spring 2018	Due 11:59pm Thursday, February 15

Hand in your solutions electronically using CMS. Each solution should be submitted as a separate file. Collaboration is encouraged while solving the problems, but:

1. **list the names of those with whom you collaborated;**
2. **you must write up the solutions in your own words;**
3. **you must write your own code.**

Remember that when a problem asks you to design an algorithm, you must also prove the algorithm's correctness and analyze its running time. The running time must be bounded by a polynomial function of the input size *unless otherwise specified.*

(1) *(10 points)*
Suppose we are given a graph $G = (V, E)$ with costs $(c_e)_{e \in E}$ on the edges and costs $(c_v)_{v \in V}$ on the vertices. We wish to designate a subset of the vertices as *active* and a subset of the edges as *eligible*, such that every inactive vertex can be joined to at least one active vertex by a path made up of eligible edges. Design an algorithm to choose the set of active vertices, A, and the set of eligible edges, F, so as to minimize their combined cost, $\sum_{v \in A} c_v + \sum_{e \in F} c_e$.

Remark: You may assume that G is a connected graph. In particular, if G has n vertices and m edges, you may assume $m \geq n - 1$.

(2) In class we've been talking about applications of dynamic programming to optimization. There are also many applications of dynamic programming to counting, and to calculating probabilities. This exercise explores one such application. Recall that an instance of the *interval scheduling* problem consists of n intervals $I_1, I_2, \ldots, I_n$, where each interval I_k (for $k = 1, \ldots, n$) is a closed interval $[s_k, f_k]$ with start time s_k and finish time $f_k > s_k$. A set of intervals is *non-conflicting* if no two of its elments overlap.

In this exercise we assume we are given an instance of the interval scheduling problem such that the numbers $s_1, s_2, \ldots, s_n, f_1, f_2, \ldots, f_n$ are all distinct, and such that the intervals are ordered by increasing finish time: $f_1 < f_2 < \cdots < f_n$.

(2a) *(7 points)*
Design an algorithm to count how many subsets of $\{I_1, I_2, \ldots, I_n\}$ are non-conflicting. Remember that the empty set and one-element sets are always non-conflicting.

(2b) *(3 points)*
Let Ω denote the collection of all non-conflicting subsets of $\{I_1, \ldots, I_n\}$. Given the list of intervals $I_1, \ldots, I_n$, and an index k in the range $1 \leq k \leq n$, design an algorithm to compute the probability that a uniformly random element of Ω contains interval I_k.

In your solution to (2b), you may omit the running time analysis. The algorithm you design must still have running time bounded by a polynomial function of n, but you don't need to include the analysis of running time in your write-up. You are also free to use the algorithm from part (2a) as a subroutine in part (2b), even if you didn't succeed in solving (2a).

(3) *(10 points)*
In the TROMINO TILING problem one is given a subset of a square grid, and one must decide whether the given subset can be tiled (i.e., covered without overlaps) by L-shaped trominoes.

The input to the problem can be given in the form of a $k \times n$ array of 0's and 1's. Denoting this array by A, the entry $A[i, j]$ equals 1 if the grid cell in location (i, j) belongs to the subset to be tiled, and if not then $A[i, j] = 0$.

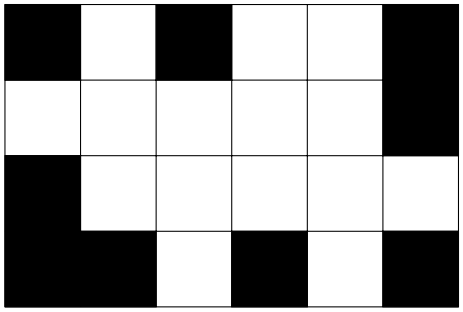
Design an algorithm to solve TROMINO TILING in time $O(n) \cdot 2^{O(k)}$. Your algorithm only needs to output a simple "yes" or "no" answer indicating whether or not the given subset of the grid can be tiled by L-shaped trominoes. In describing your algorithm, you may assume you have access to a subroutine called WIDTHTWO that solves the case when n (the width of the grid) is equal to 2, and k (the grid's height) is an arbitrary positive integer. The running time of the WIDTHTWO subroutine is $O(k)$. You are *not* responsible for designing or analyzing the WIDTHTWO subroutine[1]; you are welcome to just assume it exists and treat it as a "black box."

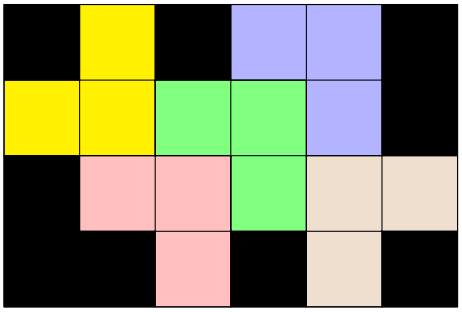
Note that the running time of the algorithm you are being asked to design is **not polynomial in the input size**. It is linear in n but exponential in k. Partial credit will be awarded for algorithms whose running time depends super-exponentially on k, or has super-linear but still polynomial dependence on n. No credit will be awarded for algorithms whose running time is exponential in n.

Example. In the following example $k = 4$ and $n = 6$, and the matrix A is given by

$$A = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}.$$

The region to be tiled is the set of white squares in the left figure below. The right figure illustrates a tromino tiling of the region.





[1] If you're curious, though, you might find it to be an instructive exercise to try designing an algorithm for the WIDTHTWO subroutine that runs in $O(k)$.