ALGORITHMS, FALL 2018, HOMEWORK 10

Due Thursday, November 15 at 11:59pm.
Worth 2% of the final grade.
Submit problems 1 and 2 on separate pages. Also please submit 1(c) starting on a separate page from 1(b).

1. You are given k bricks, arranged in k stacks of height 1. In general, a "move" involves choosing a stack and redistributing all of its bricks into other stacks, but when doing this each of the stacks can receive at most one brick. During the move, you are allowed to make new stacks (in fact this is unavoidable if you don't have enough existing stacks in which to place all the bricks), but again only one brick can go in each new stack.
   The *reward* of one move is precisely the number of bricks in the stack that you choose to redistribute. Suppose that you get to make some huge number of moves (say, n, far greater than k).

   (a) Develop a strategy to maximize your average reward per move (equivalent to maximizing the total reward over n moves). Express this as a function of k, using Θ-notation. In other words your maximization doesn't have to be entirely precise; you may assume that k is any convenient number that will make the math easier for your strategy, but you cannot assume that $k = O(1)$. Notice that any strategy provides a lower bound on reward optimality. The better the strategy, the better (higher) the lower bound.

   (b) Use amortization (specifically, the potential method) to obtain an upper bound on the best possible average reward. This upper bound should match what you get in part (a), using Θ-notation. If it helps to think in terms of cost instead, pretend that your friend is playing this game and it will cost you because you must pay their reward. In this context, you want to find an upper bound on the total cost of your friend's n moves, without having the slightest idea of what their strategy is.
   For amortization, as usual, you want to take advantage of the fact that "expensive" moves do not happen that often. Your potential function should create a potential difference that offsets expensive moves, making their amortized cost relatively smaller. So, first think about defining what an expensive move is, and what a non-expensive move is. It will help to have a good strategy in part (a) to determine what the cutoff should be. Then think about something that changes a lot in the configuration of stacks whenever you have an expensive move, and use that as your $\Delta\Phi$, then reverse-engineer Φ. Always remember that when you evaluate Φ_i at any given time, you are taking a snapshot of the current state of the world. Φ cannot be a function of "what changed" between two iterations. That is the job of $\Delta\Phi$.

   (c) Instead of the potential method, use the accounting method. Explain why your accounting works.

2. Let G be network of V cities, and E roads, each of which directly links some pair of cities. Some cities are wonderful vacation spots. Other cities are awful places to live, and their residents are eager to go on vacation. Finally, there are cities that are neither awful, nor wonderful, they're just boring.

   For each awful city the residents want to use the road network to arrive at any wonderful vacation city. They don't care which one it is or how long it takes to get there. They don't mind driving through boring cities, but they refuse to pass through any other awful city along the way.

   Assume that there is universal agreement on which cities are wonderful, boring or awful, and that the network has been properly represented by an Algorithms expert.

   How can you list all awful cities for which the residents will be able to go on vacation? Argue correctness and provide a time complexity for your algorithm.