Homework 2: BFS, DFS, and Greedy

Handed out Wednesday, September 13. Due at the start of class Monday, September 25.

Problem 1. In class we presented a greedy algorithm for scheduling a set of n tasks, in which each task is given a duration t_i and deadline d_i. We showed that scheduling the tasks in increasing order of deadline minimizes the maximum lateness. (Recall that if task i is scheduled at time $s(i)$, then its lateness is $\ell_i = max(0, s(i) + t_i - d_i)$.) Define the *average lateness* to be $(1/n)\sum_{i=1}^{n} \ell_i$.

(a) Provide a counterexample to show that scheduling tasks in increasing order of deadline does not minimize average lateness. Briefly explain your example.

(b) Provide a counterexample to show that scheduling tasks in increasing order of duration does not minimize average lateness. Briefly explain your example.

(c) Suppose that we redefine lateness to be $\ell_i = s(i) + t_i - d_i$ (ignoring the "max"). Thus, if the task finishes before the deadline, its lateness is negative. (Intuitively, this can be thought of as a bonus, which can be applied to reduce the positive lateness of some other task.) Prove that if tasks are scheduled in increasing order of duration, then average lateness (under this modified definition) will be minimized. (Hint: Use the same sort of exchange argument we used in class to prove that earliest deadline first minimizes maximum lateness.)

Remark: When constructing a counterexample, try to make the counterexample as simple as possible. For example, a counterexample with three tasks is better than one with five tasks, because it is easier to understand. Also, avoid ambiguous situations. For example, if your algorithm schedules the earliest deadline first, you should not have two identical deadlines and then impose assumptions about which one the algorithm will choose first.

Problem 2. This problem involves the question of computing change for a given coin system. A coin system is defined to be a sequence of coin values $v_1 < v_2 < ... < v_n$, such that $v_1 = 1$. For example, in the U.S. coin system we have six coins with values $\langle 1, 5, 10, 25, 50, 100\rangle$. The question is what is the best way to make change for a given integer amount A.

(a) Let $c \geq 2$ be an integer constant. Suppose that you have a coin system where there are n types of coins of integer values $v_1 < v_2 < ... < v_n$, such that $v_1 = 1$ and, for $1 < i \leq n, v_i = c \cdot v_{i-1}$. (For example, for $c = 3$ and $n = 4$, an example would be $\langle 1, 3, 9, 27\rangle$.) Describe an algorithm which given n, c, and an initial amount A, outputs an n-element vector that indicates the minimum number of coins in this system that sums up to this amount. (Hint: Use a greedy approach.)

(b) Given an initial amount $A \geq 0$, let $\langle m_1, ..., m_n\rangle$ be the number of coins output by your algorithm. Prove that the algorithm is correct. In particular, prove the following:

(i) For $1 \leq i \leq n, m_i \geq 0$
(ii) $\sum_{i=1}^{n} m_i \cdot v_i = A$
(iii) The number of coins used is as small as possible

Prove that your algorithm is optimal (in the sense that it generates the minimum number of coins) for any such currency system.

(c) Give an example of a coin system (either occurring in history, or one of your own invention) for which the greedy algorithm may fail to produce the minimum number of coins for some amount. Your coin system must have a 1-cent coin.

Problem 3. You are given an integer n and two sequences of nonnegative integers $R = \langle r_1, ..., r_n\rangle$ and $C = \langle c_1, ..., c_n\rangle$, such that $0 \leq r_i, c_j \leq n$, and $\sum_i r_i = \sum_j c_j$. Given these sequences, you are asked to determine whether it is possible to place pawns on an n x n chess board such for $1 \leq i, j \leq n$, row i has exactly r_i pawns and column j has exactly c_j pawns (see Fig. 1). If so, specify which squares of the board contain pawns. (There may be many valid solutions, and your algorithm can generate any one of them.)

Show that there exists an algorithm that solves this problem in $O(n^2)$ time. Prove that your algorithm is correct. (Hint: Greedy.)

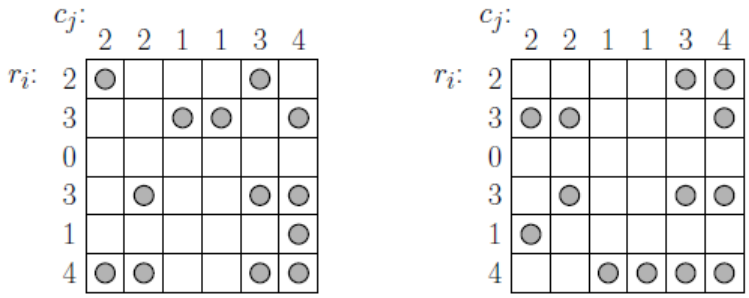

Figure 1: Two possible solutions to the inputs $R = \langle 2, 3, 0, 3, 1, 4 \rangle$ and $C = \langle 2, 2, 1, 1, 3, 4 \rangle$.

Problem 4. Programming Depth-First and Breadth-First Search

Write a Python program that reads in an adjacency matrix representing a graph, either from the command line or as input and outputs the following information for each input matrix:

a. Using each vertex of the graph as the starting point, write the order in which the vertices of the graph are visited using depth-first search.

b. Using each vertex of the graph as the starting point, write the order in which the vertices of the graph are visited using breadth-first search.

Convert each adjacency matrix into an equivalent adjacency list. Vertices are numbered consecutively from 1 to n. Store adjacent vertices in increasing order in the adjacency list.
(You will need to parse the string input back into a 2-d list before you can convert the adjacency matrix to an adjacency list. Refer to the code file PythonReviewGraphs.py for help with this.)
Example input:
G1 = [[0,0,1,0,1], [0,0,1,0,0], [1,1,0,0,1], [0,0,0,0,0], [1,0,1,0,0]]
G2 = [[0,1,0,1,0], [1,0,1,0,1], [0,1,0,1,1], [1,0,1,0,1], [0,1,1,1,0]]
G3 = [[0,1,0,1,1,0,0,0,0,0,0,0], [1,0,1,0,1,0,0,0,0,0,0,0], [0,1,0,0,1,1,0,0,0,0,0,0], [1,0,0,0,1,0,1,0,0,0,0,0], [1,1,1,1,0,1,1,1,1,0,0,0], [0,0,1,0,1,0,0,0,1,0,0,0], [0,0,0,1,1,0,0,1,0,1,0,0], [0,0,0,0,1,0,1,0,1,1,1,1], [0,0,0,0,1,1,0,1,0,0,0,1], [0,0,0,0,0,0,1,1,0,0,1,0], [0,0,0,0,0,0,0,1,0,1,0,1], [0,0,0,0,0,0,0,1,1,0,1,0]]

Example Output:
Depth-First Search G1:
1 3 2 5
2 3 1 5
3 1 5 2
4
5 1 3 2

Breadth-First Search G1:
1 3 5 2
2 3 1 5
3 1 2 5
4
5 1 3 2

Depth-First Search G2:
1 2 3 4 5
2 1 4 3 5
3 2 1 4 5
4 1 2 3 5
5 2 1 4 3

Breadth-First Search G2:
1 2 4 3 5
2 1 3 5 4
3 2 4 5 1
4 1 3 5 2
5 2 3 4 1

Depth-First Search G3:
1 2 3 5 4 7 8 9 6 12 11 10
2 1 4 5 3 6 9 8 7 10 11 12
3 2 1 4 5 6 9 8 7 10 11 12
4 1 2 3 5 6 9 8 7 10 11 12
5 1 2 3 6 9 8 7 4 10 11 12
6 3 2 1 4 5 7 8 9 12 11 10
7 4 1 2 3 5 6 9 8 10 11 12
8 5 1 2 3 6 9 12 11 10 7 4
9 5 1 2 3 6 4 7 8 10 11 12
10 7 4 1 2 3 5 6 9 8 11 12
11 8 5 1 2 3 6 9 12 4 7 10
12 8 5 1 2 3 6 9 4 7 10 11

Breadth-First Search G3:
1 2 4 5 3 7 6 8 9 10 11 12
2 1 3 5 4 6 7 8 9 10 11 12
3 2 5 6 1 4 7 8 9 10 11 12
4 1 5 7 2 3 6 8 9 10 11 12
5 1 2 3 4 6 7 8 9 10 11 12
6 3 5 9 2 1 4 7 8 12 10 11
7 4 5 8 10 1 2 3 6 9 11 12
8 5 7 9 10 11 12 1 2 3 4 6
9 5 6 8 12 1 2 3 4 7 10 11
10 7 8 11 4 5 9 12 1 2 3 6
11 8 10 12 5 7 9 1 2 3 4 6
12 8 9 11 5 7 10 6 1 2 3 4

Challenge Problem. Challenge problems count for extra credit points.

This problem involves the question of how the number of back edges in a graph affects the number of simple cycles. Recall that a cycle is *simple* if no vertex appears twice in the cycle. For the purposes of counting, two cycles containing the same set of vertices and the same set of edges are the same cycle (whether you use a different starting node or traverse the cycle backwards).

(a) Suppose that G is an undirected graph, and after running DFS you find that there is exactly one back edge. What is the maximum number of simple cycles that G can have? Explain.

(b) Answer (a) again, but now suppose that there are two back edges in the DFS. Explain.

(c) Answer (a) again, but now suppose that there are k back edges. Express your answer as a function of k. Explain. (Hint: I don't know the exact number. Try to derive as large a value as you can. You can express your answer asymptotically (e.g., $O(k)$, $O(k^2)$, $2^{O(k)}$).

(d) Suppose now that G is a directed graph, and after running DFS you find that there is exactly one back edge, but there are potentially an arbitrary number of forward and cross edges. As a function of the number of vertices n, what is the maximum number of simple cycles that G can have? (If you prefer, you can express your answer in terms of some other parameter, such as the total number of edges or the numbers of forward/cross edges.)