CSE 202: Design and Analysis of Algorithms **(Due: 12/06/19)**

Homework #4

Instructor: Ramamohan Paturi *Name:* Shihan Ran, *Netid:* A53313589

Problem 1: Hamiltonian path

Problem Description

Suppose we are given a directed graph $G = (V, E)$, with $V = \{v_1, v_2, \ldots, v_n\}$, and we want to decide whether G has a Hamiltonian path from v_1 to v_n. (That is, is there a path in G that goes from v_1 to v_n, passing through every other vertex exactly once?)

Since the Hamiltonian Path Problem is NP-complete, we do not expect that there is a polynomial-time solution for this problem. However, this does not mean that all nonpolynomial-time algorithms are equally "bad." For example, here's the simplest brute-force approach: For each permutation of the vertices, see if it forms a Hamiltonian path from v_1 to v_n. This takes time roughly proportional to $n!$, which is about 3×10^{17} when $n = 20$.

Show that the Hamiltonian Path Problem can in fact be solved in time $O(2^n \cdot p(n))$, where $p(n)$ is a polynomial function of n. This is a much better algorithm for moderate values of n; for example, 2^n is only about a million when $n = 20$.

In addition, show that the Hamiltonian Path problem can be solved in time $O(2^n \cdot p(n))$ and in polynomial space.

Solution

(High-level description)

(Correctness)

(Time complexity)

Problem 2: Remote Sensors

Problem Description

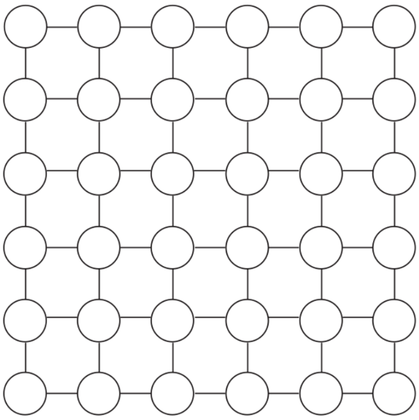

Figure 1: A grid graph

Suppose you are given an $n \times n$ grid graph G, as in Figure 1 Associated with each node v is a weight $w(v)$, which is a nonnegative integer. You may assume that the weights of all nodes are distinct. Your goal is to choose an independent set S of nodes of the grid, so that the sum of the weights of the nodes in S is as large as possible. (The sum of the weights of the nodes in S will be called its total weight.) Consider the following greedy algorithm for this problem.

Algorithm 1: The "heaviest-first" greedy algorithm

Start with S equal to the empty set
while some node remains in G **do**
  Pick a node v_i of maximum weight
  add v_i to S
  Delete v_i and its neighbors from G
end while
return S

(Subproblem 1)

Let S be the independent set returned by the "heaviest-first" greedy algorithm, and let T be any other independent set in G. Show that, for each node $v \in T$, either $v \in S$,or there is a node $v' \in S$ so that $w(v) \leq w(v\prime)$ and $(v, v\prime)$ is an edge of G.

(Solution 1)

(High-level description)

(Correctness)

(Time complexity)

(Subproblem 2)

Show that the "heaviest-first" greedy algorithm returns an independent set of total weight at least $1/4$ times the maximum total weight of any independent set in the grid graph G.

(Solution 2)

(High-level description)

(Correctness)

(Time complexity)

Problem 3: Scheduling

Problem Description

Consider the following scheduling problem. You are given a set of n jobs, each of which has a time requirement t_i. Each job can be done on one of two identical machines. The objective is to minimize the total time to complete all jobs, i.e., the maximum over the two machines of the total time of all jobs scheduled on the machine. A greedy heuristic would be to go through the jobs and schedule each on the machine with the least total work so far.

(Subproblem 1)

Give an example (with the items sorted in decreasing order) where this heuristic is not optimal.

(Solution 1)

(High-level description)

(Correctness)

(Time complexity)

(Subproblem 2)

Assume the jobs are sorted in decreasing order of time required. Show as tight a bound as possible on the approximation ratio for the greedy heuristic. A ratio of 7/6 or better would get full credit. A ratio worse than 7/6 might get partial credit.

(Solution 2)

(High-level description)

(Correctness)

(Time complexity)

Problem 4: Maximum coverage

Problem Description

The maximum coverage problem is the following: Given a universe U of n elements, with nonnegative weights specified, a collection of subsets of $U, S_1, \ldots, S_l$, and an integer k, pick k sets so as to maximize the weight of elements covered. Show that the obvious algorithm, of greedily picking the best set in each iteration until k sets are picked, achieves an approximation factor of $\left(1-(1-1/k)^k\right) > (1-1/e)$.

Solution

(High-level description)

(Correctness)

(Time complexity)