Algorithms: CSE 202 — Homework IV

Problem 1: Hamiltonian path (KT 10.3)

Suppose we are given a directed graph $G = (V, E)$, with $V = \{v_1, v_2, \dots, v_n\}$, 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.

Problem 2: MaxCut for Trees (KT 10.9)

Give a polynomial-time algorithm for the following problem. We are given a binary tree $T = (V, E)$ with an even number of nodes, and a nonnegative weight on each edge. We wish to find a partition of the nodes V into two sets of *equal* size so that the weight of the cut between the two sets is as large as possible (i.e., the total weight of edges with one end in each set is as large as possible). Note that the restriction that the graph is a tree is crucial here, but the assumption that the tree is binary is not. The problem is NP-hard in general graphs.

Problem 3: Heaviest first (KT 11.10)

Suppose you are given an $n \times n$ grid graph G, as in Figure 1 Associated with each node v is a weight $w(v)$,

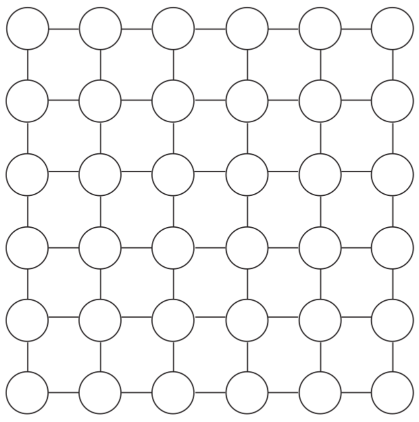
Figure 1: A grid graph

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 $S$
```

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 $w(v) \leq w(v')$ and (v, v') is an edge of G.

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

Problem 4: Bin packing

In the bin packing problem, we are given a collection of n items with weights $w_1, w_2, \ldots, w_n$. We are also given a collection of bins, each of which can hold a total of W units of weight. (We will assume that W is at least as large as each individual w_i.)

You want to pack each item in a bin; a bin can hold multiple items, as long as the total of weight of these items does not exceed W. The goal is to pack all the items using as few bins as possible. Doing this optimally turns out to be **NP**-complete, though you don't have to prove this.

Here's a merging heuristic for solving this problem: We start with each item in a separate bin and then repeatedly "merge" bins if we can do this without exceeding the weight limit. Specifically:

Merging Heuristic:

Algorithm 2: Merging heuristic

```
Start with each item in a different bin
while there exist two bins so that the union of their contents has total weight $W$ do
  Empty the contents of both bins
  Place all these items in a single bin.
end while
Return the current packing of items in bins.
```

Notice that the merging heuristic sometimes has the freedom to choice several possible pairs of bins to merge. Thus, on a given instance, there are multiple possible executions of the heuristic.

Example. Suppose we have four items with weights 1, 2, 3, 4, and W = 7. Then in one possible execution of the merging heuristic, we start with the items in four different bins; then we merge the bins containing the first two items; then we merge the bins containing the latter two items. At this point we have a packing using two bins, which cannot be merged. (Since the total weight after merging would be 10, which exceeds $W = 7$.)

1. Let's declare the size of the input to this problem to be proportional to

$$n + \log W + \sum_{i=1}^{n} \log w_i$$

(In other words, the number of items plus the number of bits in all the weights.) Prove that the merging heuristic always terminates in time polynomial in the size of the input. (In this question, as in **NP**-complete number problems from class, you should account for the time required to perform any arithmetic operations.)

2. Give an example of an instance of the problem, and an execution of the merging heuristic on this instance, where the packing returned by the heuristic does not use the minimum possible number of bins.

3. Prove that in any execution of the merging heuristic, on any instance, the number of bins used in the packing returned by the heuristic is at most twice the minimum possible number of bins.

Problem 5: Maximum coverage

Given a universal set 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 wight of elements covered.

Show that the obvious algorithm, of greedily picking the best set in each iteration until k sets are picked, achieve an approximation factor of $1 - (1 - 1/k)^k > 1 - 1/e$.