Problem 1: Hamiltonian path (KT 10.3)

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

Show that the Hamiltonian Path problem can be solved in time $O(2^n \cdot p(n))$ where $p(n)$ is a polynomial function of n, and in polynomial space.

Algorithm:

Let c_S be the count of paths of length $n-1$ starting from v_1 that do not pass through vertices in the set S, $v_1 \notin S$. We can solve this problem by DP.

DP definition: Let $C(u, k)$ be the count of paths of length k ($0 \leq k \leq n-1$) starting from u and do not pass through vertices in S.

Recurrence formulation: $C(u, 0) = 1$. For $0 < i \leq n-1$, $C(v, i) = \sum_{(v,u) \in E, u \notin S} C(u, i-1)$.

We iterate i from 1 to $n-1$, compute all $C(v, i)$ and finally get $c_S = C(v_1, n-1)$. As we compute the answer for every subset S, we can determine whether there is a Hamiltonian path starting from v_1 by checking if the following running sum is greater than 0: $n_H = \sum_S (-1)^{|S|} c_S$. To check if there is a Hamiltonian path that ends in v_n, we alter G a bit by appending a new node t and a directed edge (v_n, t). Apply the above algorithm to new graph, there is a Hamiltonian path that goes from v_1 to v_n iff n_H of the new graph is greter than 0.

Proof of correctness:

Claim 1. *$\forall (v, i, S)$, the DP algorithm correctly computes $C(v, i)$.*

Proof. Base case: When $i = 0$, the path of length 0 is the node itself, hence $C(v, 0) = 1$.

Inductive step: Assume $\forall u$, $C(u, i-1)$ is correctly computed. Paths of length i starting from v consist of paths that start from the nodes that v points to and have a length of $i-1$. If a node v points to is in S, that node is not considered. By summing up all those paths, we get $C(v, i)$. □

Claim 2. *$n_H > 0$ iff there is a Hamiltonian path starting from v_1.*

Proof. All Hamiltonian paths contribute only to $c_\emptyset$. If a path is not Hamiltonian, it must have some vertices missing. Assume the missing vertices form a set $\hat{S}$. Such a path would contribute 1 to all c_S where S is a subset of $\hat{S}$. When adding c_S up with alternating signs, they cancel out due to binomial theorem. So n_H is the count of remaining paths, which are Hamiltonian paths. □

Claim 3. *$n_H > 0$ in the new graph iff there is a Hamiltonian path that goes from v_1 to v_n in G.*

Proof. A Hamiltonian path starting from v_1 in the new graph can only take the form $(v_1, ..., v_n, t)$. So the existence of such path per Claim 2 indicates a Hamiltonian path of the form $(v_1, ..., v_n)$ in G. On the other hand, if there exists a Hamiltonian path from v_1 to v_n, one can simply extend it to t to get a Hamiltonian path in the new graph. □

Complexity:

DP algorithm takes $O(n^3)$ time since there are n^2 number of $C(v, i)$ to compute and each summation takes at most n operations. There are 2^n subsets, so total time complexity is $O(2^n \cdot n^3)$. Since we are computing a running sum, the space used is just for one call of the DP algorithm, which is $O(n^2)$. ($O(n)$ for caching intermediate results and $O(n^2)$ for storing G.)

Problem 2: Heaviest first (KT 11.10)

Suppose you are given an $n \times n$ grid graph G. Associated with each node v is a weight $w(v)$, which is a non-negative 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: Start with S equal to the empty set. While some node remains in G, we pick a node v_i of maximum weight, add v_i to S and delete v_i and its neighbors from G. Repeat.

2.1: 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 the following claim is true.

Claim 1. *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.*

Proof. We prove by contradiction. Assume that $v \notin S$ and there is no node $v' \in S$ such that (v, v') is an edge of G. Per the greedy algorithm, v is not selected because it is the neighbor of a node in S, which is a contradiction.

Now we prove that $\exists v' \in S$ which is a neighbor of v and $w(v) \leq w(v')$. According to the discussion above we know at least one of v's 4 neighbors is in S. Let its first neighbor that was selected by the algorithm be v', per the greedy algorithm we have $w(v) \leq w(v')$. □

2.2: Show that the following claim is true.

Claim 2. *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.*

Proof. Let the maximum total weight of any independent set in the grid graph G be W_{opt}, and the optimal set of nodes is S_{opt}. Let the total weight of greedy algorithm solution be W. Then:

$$\begin{aligned} W_{opt} &= \sum_{u \in S \cap S_{opt}} w(u) + \sum_{v \in S_{opt} \setminus S} w(v) \\ &= W - \sum_{v' \in S \setminus S_{opt}} w(v') + \sum_{v \in S_{opt} \setminus S} w(v). \end{aligned}$$

Per Claim 1, we can find a node $v' \in S$ for every node v that is not in S but in S_{opt} and have $w(v) \leq w(v')$. The extreme scenario would be that each v' is counted 4 times, because one node can have at most 4 neighbors. We furthur notice that such a neighbor set must not contain nodes

in S_{opt} because of property of independent set. That is, it is a subset of $S \setminus S_{opt}$. So:

$$\begin{aligned} W_{opt} &\leq W - \sum_{v' \in S \setminus S_{opt}} w(v') + 4 \sum_{v' \in S \setminus S_{opt}} w(v') \\ &= W + 3 \sum_{v' \in S \setminus S_{opt}} w(v') \\ &\leq 4W. \end{aligned}$$

Hence $\frac{W}{W_{opt}} \geq \frac{1}{4}$. □

Problem 3: Scheduling

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.

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

Let the jobs be $t = (3, 3, 2, 2, 2)$. Jobs on machine 1 will be $(3, 2, 2)$, and jobs on machine 2 will be $(3, 2)$. So the maximum time is 7. But the optimal assignment is $(3, 3)$ for machine 1 and $(2, 2, 2)$ for machine 2. And the optimal time is 6.

3.2: 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.

Claim 1. *The approximation ratio for the greedy heuristic is* $\frac{7}{6}$.

Proof. Let the optimal total time be OPT, and the total time obtained by greedy heuristic be GR. We know that:

$$OPT \geq \frac{1}{2}\sum_{i=1}^{n} t_i.$$

It's easy to see that if $n \leq 3$, $OPT = GR$. If $n = 4$, we first study the case where each machine has 2 jobs. (The case where one machine has 3 jobs is included in the discussion below.) Say machine 1 has (t_1, t_4) and machine 2 has (t_2, t_3). This happens when $t_1 \leq t_2 + t_3$. So $GR = \max(t_1 + t_4, t_2 + t_3)$. It's optimal since any other combination will bring a greater total time.

If $n \geq 5$, one machine has at least 3 jobs. This is because there are only 2 scenarios for 5 jobs: one get 2, the other get 3; or one get 1, the other get 4. So we have:

$$OPT \geq 3t_n.$$

In the greedy algorithm, the last job t_n will always be assigned to the machine that has a smaller total time so far. Let it be machine 1. There are 2 cases after assignment:

- Machine 2 still has a greater total time. Let the last job added to machine 2 be t_k. We have $\sum_{i=k+1}^{n} t_i \leq t_k$. We can safely exclude all jobs from t_{k+1} to t_n since they have no effect on GR. So GR in this case is just the same GR for the first k jobs. Let's call it GR_k. Once we show that GR_k is bounded by some constant times OPT_k, since we know $OPT_k \leq OPT$, we can conclude that GR is bounded by the same ratio too.

- Machine 1 will have a greater total time, and we have:

$$\begin{aligned} GR &\leq \frac{1}{2}\sum_{i=1}^{n-1} t_i + t_n \\ &= \frac{1}{2}\sum_{i=1}^{n} t_i + \frac{1}{2}t_n \\ &\leq OPT + \frac{1}{6}OPT. \end{aligned}$$

So $\frac{GR}{OPT} \leq \frac{7}{6}$. This is tight as the example shows in **3.1**. □

Problem 4: Maximum coverage

The maximum coverage problem is the following: Given a universe U of n elements, with non-negative weights specified, a collection of subsets of U, $S_1, ..., S_l$, 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 $(1-(1-\frac{1}{k})^k) > (1-\frac{1}{e})$.

Notations: Let OPT denote the value of an optimal solution. Let x_i denote the sum of new element weights covered by the greedy algorithm in the i-th set that it picks. Also, let $y_i = \sum_{j=1}^{i} x_i$, and $z_i = OPT - y_i$. According to our notations, $y_0 = 0$, y_k is the total weight chosen by the greedy algorithm, and $z_0 = OPT$.

Claim 1. $x_{i+1} \geq \frac{z_i}{k}$.

Proof. At each step, the greedy algorithm selects the subset S_j who covers the maximum total weight of uncovered elements. Let's assume the optimal solution uses k sets to cover a total weight of OPT, and at step i, the greedy solution already selects a subset of those optimal sets. Now, regardless of actual choices, the greedy solution could simply select all the remaining sets in the optimal solution and achieve a total weight no less than OPT, that is, the increase of selecting *all* those sets (there are at most k of them) would be at least z_i. Of course, since the greedy algorithm can only select one set, we know that some set must cover at least $\frac{1}{k}$ fraction of z_i. Hence, $x_{i+1} \geq \frac{z_i}{k}$. □

Claim 2. $z_{i+1} \leq (1-\frac{1}{k})^{i+1} \cdot OPT$.

Proof. The claim is true for $i = 0$. We assume inductively that $z_i \leq (1-\frac{1}{k})^i \cdot OPT$. Then

$$\begin{aligned} z_{i+1} &= z_i - x_{i+1} \\ &\leq z_i(1-\frac{1}{k}) \\ &\leq (1-\frac{1}{k})^{i+1} \cdot OPT. \end{aligned}$$

□

Claim 3. *The greedy algorithm achieves an approximation factor of* $(1-(1-\frac{1}{k})^k) > (1-\frac{1}{e})$.

Proof. It follows from Claim 2 that $z_k \leq (1-\frac{1}{k})^k \cdot OPT$. Hence, $y_k = OPT - z_k \geq (1-(1-\frac{1}{k})^k) \cdot OPT$. So $\frac{y_k}{OPT} \geq (1-(1-\frac{1}{k})^k) > (1-\frac{1}{e})$. □