Exercise 12.15. It's-a me, Mario! This problem is inspired by Super Mario World, where Mario must make it from some starting point to the flag in front of a castle, collecting as many coins as possible along the way.

Let $G = (V, E)$ be a directed graph where each vertex $v \in V$ has $C_v \geq 0$ coins. For a walk in G, we say that the *number of coins collected by the walk* is the total sum of C_v over all *distinct* vertices v in the walk. (If we visit a vertex v more than once, we still only get C_v coins total.) Given $s, t \in V$, the goal is to compute the maximum number of coins collected by any (s, t)-walk.

Exercise 12.15.1. Let G be a DAG. For this problem, either (a) design and analyze a polynomial time algorithm (the faster the better), or (b) prove that a polynomial time algorithm would imply a polynomial time algorithm for SAT.

Solution. We claim that the following algorithm suffices.

`mcw-dag`(v, t)`:`

/ given $v, t \in V$ in a DAG, computes the maximum number of coins collected by any (v,t)-walk and returns $-\infty$ if no such walk exists. */*

1. If $v = t$ then return C_v.
2. Let $m \leftarrow -\infty$.
3. For $(v, w) \in \delta^+(v)$ do
   A. $m \leftarrow \max(m,\ C_v +$ `mcw-dag`$(w, t))$
4. Return m

We find the maximum number of coins collected along any (s, t)-walk by calling `mcw-dag`(s, t).

Runtime. Given a goal vertex $t \in V$, if we cache the return value of `mcw-dag`(v, t) for all $v \in V$, then our algorithm has the following runtime complexity:

$$O\left(\sum_{v \in V}(1 + d^+(v))\right) = O(m + n)$$

■

Correctness. The correctness of this algorithm follows from performing induction, in reverse topological order, according to the recursive specification. We claim that `mcw-dag`(v, t) returns the maximum number of coins collected by any (v, t)-walk for all $v \in V$.

In the base case, v is the last vertex in topological order—a sink—so the maximum number of collectible coins is C_v if $v = t$ and $-\infty$ if $v \neq t$, as returned in the algorithm.

Now we assume that for some $v \in V$, the claim holds for all vertices that follow v in topological order. In the case where v is a sink, there are no vertices reachable from v and consequently, the algorithm correctly returns C_v if $v = t$ and $-\infty$ if $v \neq t$. Otherwise, since the claim holds, by assumption, for all w such that $(v', w) \in \delta^+(v')$, we can conclude that the maximum number of collectible coins on a (v, t)-walk is

$$C_v + \max_{w:\ (v,w) \in \delta^+(v)} \texttt{mcw-dag}(w, t)$$

as computed in the algorithm.

Thus, by induction, the claim that `mcw-dag`(v, t) returns the maximum collectible coins along a (v, t)-walk holds for all $v \in V$. ■

□

Exercise 12.15.2. Let G be a general directed graph. For this problem, either (a) design and analyze a polynomial time algorithm (the faster the better), or (b) prove that a polynomial time algorithm would imply a polynomial time algorithm for SAT.

Solution. When G is a general directed graph, we compress G to its condensation graph G', then apply the algorithm given in 12.15.1 to G'.

`mcw(`s, t`):`

/ given $s, t \in V$ in a general digraph, computes the maximum number of coins collected by any (s,t)-walk and returns $-\infty$ if no such walk exists. */*

/ see 12.15.1 for* `mcw-dag()` *pseudocode. */*

1. Let $\{S_1, S_2, ..., S_k\}$ be the strongly-connected components in G.
2. Contract each S_i into a single vertex s_i in $G' = (V', E')$ with $C_{S_i} = \sum_{v \in S_i} C_v$.
3. For $i \neq j$, $(s_i, s_j) \in E'$ iff there exist $u \in S_i$, $v \in S_j$ with $(u, v) \in E$.
4. Let S_s and S_t be the sccs containing s and t, respectively.
5. Return `mcw-dag(`s_s, s_t`)`.

Calling `mcw(`s, t`)` gives us the maximum number of coins collected by any (s,t)-walk.

Runtime. The runtime of SCC-compression on G is in $O(m + n)$, as is the runtime of `mcw-dag()`. Hence, the total runtime complexity of our algorithm is in $O(m + n)$. ■

Correctness. Since the correctness of `mcw-dag()` was proven in 12.15.1, it suffices for us to show that our SCC-compression preserves the maximal number of coins collected along any (s, t)-walk.

First, we claim that a maximal coin-collection walk in G necessarily collects all of the coins in every SCC it visits; otherwise, we would be able to make "detours" in at least one SCC to pick up coins we've missed, contradicting the maximality of the walk. We call such a walk an SCC-walk.

Since we now know the maximum number of coins collected from an (s, t)-walk in G can be given by an SCC-walk on G; so, we can simply maximize coin collection on only the SCC-walks in G, rather than on all walks.

We have a clear correspondence between coins collected on SCC-walks in G and all walks in G':

(i) First, for every (s, t)-SCC-walk W on G, there exists a walk of the same coin-numerage from s_s to s_t in G' by taking all the SCCs W passes through.

(ii) Conversely, for every walk from s_s to s_t in G', there exists an (s, t)-SCC-walk in G with the same total weight.

By this bijection, we can conclude that the maximum coin collection over all (s, t)-walks in G is equal to the maximum coin collection over all (s_s, s_t)-walks in G'. ■

□

Exercise 13.4. Let $G = (V, E)$ be a directed graph. We say that a set of vertices is *almost independent* if each $v \in S$ has at most one neighbor in S.[5] Consider the problem of computing the maximum cardinality of any almost independent set of vertices. For this problem, either (a) design and analyze a polynomial time algorithm (the faster the better), or (b) prove that a polynomial time algorithm would imply a polynomial time algorithm for SAT.

[5] Two vertices u and v are neighbors if they are connected by an edge.

Solution. □

Exercise 13.6. Recall the dominating set problem[†] from section 13.3. Here we will consider the weighted version where the vertices are given positive weights, and the goal is to compute the minimum weight dominating set. For each of the following problems, either (a) design and analyze a polynomial time algorithm (the faster the better), or (b) prove that a polynomial time algorithm would imply a polynomial time algorithm for SAT.

[†]A set of vertices $S \subseteq V$ is a *dominating set* if every vertex $v \in V$ is either in S or the neighbor of a vertex in S. The minimum dominating set problem is to compute the minimum cardinality dominating set.

Exercise 13.6.1. The minimum weight dominating set problem for intervals, *with the additional assumption that* no two intervals are nested. To state it more precisely: the input consists of n weighted intervals $\mathcal{I}$. The non-nested assumptions means that for any two intervals $I, J \in \mathcal{I}$, we never have I contained in J or J contained in I.

The goal is to compute the minimum weight subset $S \subseteq \mathcal{I}$ of intervals such that every interval in $\mathcal{I}$ is either in S or overlaps some interval in S.

- (For 1 pt. extra credit) Extend your algorithm to general intervals.[8]

[8]Of course, anyone who has already solved the general case automatically solves the special case where no two intervals are nested.

Solution. □

Exercise 13.6.2. The minimum weight dominating set problem in trees.

Solution. □