Exercise 8.2 The following problem (and more elaborate extensions) appear in reinforcement learning. Let $G = (V, E)$ be a directed graph and let the edges be annotated by positive edge weights $r : E \to \mathbb{R}_{>0}$. We think of the vertices as states of some device under our control, and the edge weights $r(e)$ as rewards obtained by traversing the edge e as follows. Let $s \in V$ be a fixed starting vertex / state.

Exercise 8.2.1. Given an integer $k \leq n$, the goal is to compute a walk of length at most k maximizing the sum of rewards along that walk. (If you repeat an edge, you get the same reward each time you repeat the edge.) Design and analyze an algorithm for this problem.

Solution. Our approach is to constrain Bellman-Ford to compute minimum walks with no more than k edges.

```
K-Bellman-Ford($G = (V, E), s \in V, k$):
/* Computes the minimum-length $d(s, v)$ of a $k$-walk from $s$ to $v$ for all $v \in V$. */
1. Set $d(s, v) = +\infty$ for all $v \in V$.
2. Set $d(s, s) = 0$.
3. Do the following $k$ times
   A. For all edges $e = (u, v) \in E$
      1. $d(s, v) \leftarrow \min(d(s, v),\ d(s, u) + \ell(u, v))$
```

Given $G = (V, E)$ with positive edge weights, we negate the edge weights to obtain a graph G^- with only negatively weighted edges. Running $\underline{\texttt{K-Bellman-Ford}(G^-, s, k)}$ assigns $d(s, t) := \underline{\texttt{SW}(s, t, k)}$ for all $t \in V$. Hence, the distance of the maximum k-walk in G can be directly computed by negating the minimum value of $\underline{\texttt{SW}(s, t, k)}$ in G^-.

Runtime.

$$T(n) = \sum_{i=1}^{k} \sum_{e \in E} 1 \in O(mk)$$

■

Correctness. The correctness of this algorithm is inherited from the correctness proof of Bellman-Ford. If we want to prove this explicitly, it suffices to show that in k iterations, `K-Bellman-Ford` produces the shortest walks using no more than k edges; in particular, $d(s, t) = \underline{\texttt{SW}(s, t, k)}$ for all $t \in V$.

We approach this proof by induction on k, the number of iterations.

In the base case, $k = 0$, we have $d(s, t) = +\infty = \underline{\texttt{SW}(s, t, 0)}$ for all $t \neq s$, and $d(s, s) = 0 = \underline{\texttt{SW}(s, s, 0)}$, as desired.

Now suppose the claim holds for k. Then on the $(k + 1)$-th iteration, for each $t \in V$, we have one of the following cases:

i. $\underline{\texttt{SW}(s, t, k + 1)} = \underline{\texttt{SW}(s, t, k)}$. Then by the induction hypothesis, the algorithm assigns

$$d(s, t) = \underline{\texttt{SW}(s, t, k)} = \underline{\texttt{SW}(s, t, k + 1)}$$

as desired.

ii. $\underline{\texttt{SW}(s,t,k+1)} \neq \underline{\texttt{SW}(s,t,k)}$. For such t, there must exist some $v \in V$ such that $d(s,v) = \underline{\texttt{SW}(s,v,k)}$ is finite. When $d(s,v) + \ell(v,t)$ is minimal, we get the shortest possible $(k+1)$-walk from s to t through v. Correspondingly, the algorithm assigns

$$\begin{aligned} d(s,t) &= \min_{v \in V:\ (v,t) \in E} d(s,v) + \ell(v,t) \\ &= \min_{v \in V:\ (v,t) \in E} \underline{\texttt{SW}(s,v,k)} + \ell(v,t) \\ &= \underline{\texttt{SW}(s,t,k+1)} \end{aligned}$$

By induction, the claim holds for all k. ■

□

Exercise 8.2.2. Here we also incorporate a *discount rate*. Let $\alpha \in (0,1)$ be given. Given a walk with edges $e_1, \ldots, e_k$, the *discounted total reward* of the walk is given by

$$r(e_1) + \alpha r(e_2) + \cdots + \alpha^{k-1} r(e_k).$$

The idea is that if we think of each edge traversal as also taking a unit of time, then the rewards attained far off in the future are perceived to be worth less than the rewards attained now and in the short term. Given an integer $k \leq n$, the goal is to compute a walk of length at most k maximizing the discounted total reward of the walk. Design and analyze an algorithm for this problem.

Solution. Our approach is to use a modified $\underline{\texttt{SW}(s,t,k)}$ from the text to get the maximum value of a k-walk starting from s.

```
max-discount-walk($s,t,k$):

/* Given a nonnegative integer $k \in \mathbb{Z}_{\geq 0}$ and two vertices $s,t \in V$, returns the maximum total
reward for a path of length $k$ from $s$ to a vertex $t$. */

1. If $k = 0$:
    A. If $s = t$, return 0
    B. Else, return $-\infty$
2. Return $\max \begin{cases} \texttt{max-discount-walk}(s,t,k-1) \\ \max_{v \in V : e := (v,t) \in E} \left\{ \texttt{max-discount-walk}(s,v,k-1) + \alpha^{k-1} \cdot r(e) \right\} \end{cases}$
```

We assume there exists some globally defined directed graph $G = (V,E)$ with positive edge weights $r : E \to \mathbb{R}_{>0}$. To use this algorithm, we run $\underline{\texttt{max-discount-walk}(s,t,k)}$ for all $t \in V$ and $i \leq k$, using DP to cache the solutions to the sub-problems. That is,

$$\max_{\substack{v \in V \\ 0 \leq i \leq k}} \left\{ \underline{\texttt{max-discount-walk}(s,t,i)} \right\} \tag{1}$$

Runtime. Since the meaningful structure of our algorithm is exactly $\underline{\texttt{SW}(s,t,k)}$ from page 145 of the text, by Lemma 8.2 we can compute $\underline{\texttt{max-discount-walk}(s,t,k)}$ for all $t \in V$ and $i \leq k$ in $O(k(m+n))$ time. ■

Correctness. We prove correctness by strong induction. The base case when $k = 0$ is trivial.

Now, assume we have shown that the desired property holds for all values up to k. We wish to show that it must then also hold for k. By assumption, a $k-1$-walk gives a discounted total reward of

$$r(e_1) + \alpha r(e_2) + \cdots + \alpha^{k-2} r(e_{k-1}).$$

Upon running $\underline{\texttt{max-discount-walk}(s,t,k)}$, our algorithm presents two cases: either it adds a new edge, or it does nothing.

Case 1. If $\underline{\texttt{MDW}(s,t,k)} = \underline{\texttt{MDW}(s,t,k-1)}$, then correctness follows directly from the inductive hypothesis.

Case 2. If $\underline{\texttt{MDW}(s,t,k)} \neq \underline{\texttt{MDW}(s,t,k-1)}$, then $\underline{\texttt{MDW}(s,t,k)} = \underline{\texttt{MDW}(s,t,k-1)} + \alpha^{k-1} r(e_k)$ for some kth edge e_k. By the inductive hypothesis, we have that

$$\begin{aligned} \underline{\texttt{MDW}(s,t,k)} &= \underline{\texttt{MDW}(s,t,k-1)} + \alpha^{k-1} r(e_k) \\ &= r(e_1) + \alpha r(e_2) + \cdots + \alpha^{k-2} r(e_{k-1}) + \alpha^{k-1} r(e_k). \end{aligned}$$

Thus this algorithm is correct for all $k \in \mathbb{N}_0$. ■

□

Exercise 9.9. Consider the following special case of SAT, which we will call *k-occurrence-SAT* for a fixed parameter $k \in \mathbb{N}$. The input consists of a SAT formula $f(x_1, \ldots, x_n)$ in CNF such that every variable x_i appears (as is, or negated) in at most k clauses. The problem is to decide whether there is a satisfying assignment. For $k = 3$, either (a) design and analyze a polynomial time algorithm, or (b) show that a polynomial time algorithm for k-occurrence-SAT implies a polynomial time algorithm for (CNF-)SAT.[1]

[1] As a warmup, it might be helpful to first consider the case $k = 5$. If you figure out 5-occurrence SAT, but don't figure out 3-occurrence SAT, we will give partial credit for a solution to 5-occurrence SAT.

Solution. We claim that for $k = 3$, the existance of a polynomial-time algorithm for k-occurrence-SAT implies a polynomial-time algorithm for SAT. To see this, we propose a polynomial-time reduction from any CNF-SAT formula $f(x_1, ..., x_n)$ to an corresponding 3-occurrence-SAT formula.

Reduction. For each variable x_i in f, let $\hat{k}$ be the number of occurrences of x_i in f.

If x_i has $\hat{k} \leq 3$ occurrences, then f already meets the constraints of 3-occurrence-SAT.

In the case that there exists $x_i \in f$ with $\hat{k} > 3$ occurrences, we split x_i into multiple variables. We chose to split x_i into k equivalent variables $x_{i1} = x_{i2} = ... = x_{ik}$ such that j^{th} occurrence of x_i in f can be replaced by a new variable x_{ij}, $1 \leq j \leq \hat{k}$.

We also have to enforce equality between all the new variables x_{ij}. This can be done by simply appending the clauses

$$(\overline{x}_{i1} \vee x_{i2}) \ \wedge \ (\overline{x}_{i2} \vee x_{i3}) \ \wedge \ ... \ \wedge \ (\overline{x}_{i\hat{k}} \vee x_{i1})$$

to the boolean formula. The addition of these $\hat{k}$ clauses adds 2 more occurrences of each x_{ij}, for a total of 3 occurrences.

When we perform the above substitution for all x_i with more than 3 occurrences, notice that the size of the new formula, call it f', is linearly proportional to the size of the original formula f, whence the reduction is polynomial-time. ■

Correctness. Suppose we apply the reduction above to a CNF boolean formula f to get a 3-occurrence boolean formula f'. We will prove that f satisfiable $\iff$ f' satisfiable.

($\Longrightarrow$) If f is satisfiable, then by definition there exists some SAT assignment $A : \{x_i\}_{i=0}^n \to \{0, 1\}$ satisfying $f(x_1, ..., x_n)$. By our construction above, the corresponding 3-occurrence formula f' is also satisfied by assigning each "duplicate" variable x_{ij} to the same value as the original variable x_i.

($\Longleftarrow$) If f' is satisfiable, then by definition there exists some SAT assignment $A : \{x_i\}_{i=0}^n \to \{0, 1\}$ satisfying f'. We also know if we have "duplicate" variables $x_{i1}, ...x_{ik}$ for some x_i, then $A(x_{i1}) = \ ... \ = A(x_{ik})$ is enforced by the equality clauses in f'. Hence, if we set the corresponding variable x_i in f to have the same assignment, f is also satisfied. ■

□

Exercise 10.3. After your glorious app PikPok hit number 1 in the app store, you're preparing for version 2. Obviously, it needs to be great.

You've gathered a list of k features $F_1, \dots, F_k$ that you could potentially add to version 2. However, there are complicated dependencies and requirements among them so you don't necessarily want to add all of them. There are 3 types of specifications defined over pairs of features F_i and F_j:

1. Requirements: Your app must include either F_i or F_j.
2. Conflicts: You cannot include both F_i and F_j.
3. Dependencies: If you include F_i, then you must include F_j.

Collectively, we call requirements, conflicts, and dependencies the *feature specifications.* The feature specifications are given in list form. The high-level task is to decide which of the features to implement, based on the given feature specifications. We have two versions of the problem. For each of the problems [below], 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.

Exercise 10.3.1. In the idealistic feature selection problem, the task is to decide if there is a subset of features that satisfies all the feature specifications.

Solution. We claim that the case of idealistic feature selection reduces to a 2-SAT problem and therefore has a polynomial time solution.

For each feature F_i, define a boolean variable f_i such that f_i true $\iff F_i$ implemented. Then, notice that the constraints for each feature specification can be translated into the language of 2-SAT.

1. Requirement: $(f_i \vee f_j)$
2. Conflict: $(\overline{f_i} \vee \overline{f_j})$
3. Dependency: $(\overline{f_i} \vee f_j)$

Thus, the conjunction of all of the feature specifications as 2-variable clauses gives us a 2-SAT problem. We can therefore determine satisfiability of the feature specifications in polynomial time by using the algorithm for 2-SAT.

Correctness. Suppose 2-SAT returns true. Then there exists some assignment $A : \{f_1, ..., f_k\} \to \{0, 1\}$ satisfying the conjunction of all the feature-spec clauses. Including each feature F_i if and only if $A(f_i) = 1$ gives us a subset of features that satisfies all feature specifications.

On the other hand, if 2-SAT returns false, then there is no assignment on $\{f_1, ..., f_k\}$ that satisfies the conjunction of all the feature-spec clauses. Hence, there is no subset of features that would satisfy all of the feature specifications. ■

□

Exercise 10.3.2. In the realistic feature selection problem, the task is to choose a subset of features that satisfies the maximum number of feature specifications

Solution. A realistic feature selection implies a polynomial time algorithm for SAT; we prove this creating a polynomial-time reduction from any Max 2-SAT problem to a realistic feature selection problem.

Consider a generic Max 2-SAT instance $f(x_1, ..., x_n)$ with m clauses.

If we let $F_1, ..., F_n$ be features, where each x_i is true if and only if its corresponding feature F_i is chosen, then Max 2-SAT on $f(x_1, ..., x_n)$ directly corresponds to a realistic feature selection problem on $F_1, ..., F_n$. In particular, each clause constructed with variables x_i and x_j directly corresponds to a type of feature dependency:

1. $(x_i \vee x_j)$: app must include either F_i or F_j (requirements).
2. $(\overline{x}_i \vee \overline{x}_j)$: cannot include both F_i and F_j (conflicts).
3. $(\overline{x}_i \vee x_j)$: if F_i is included, F_j must be included (dependencies).
4. $(x_i \vee \overline{x}_j)$: if F_j is included, F_i must be included (dependencies).

Hence, we can create a list of feature specifications on $F_1, ..., F_n$ from $f(x_1, ..., x_n)$ in polynomial time.

Therefore, if realistic feature selection on $F_1, ..., F_n$ has a polynomial-time solution, then so does Max 2-SAT on $f(x_1, ..., x_n)$

Correctness. Let $S \subseteq \{F_1, \ldots, F_n\}$ be a subset of features, and define 'F_i chosen' $\iff$ 'x_i is true'. Then, the clauses $\{C_j\}_{j=1}^{k}$ are satisfied $\iff$ S meets the corresponding feature requirements.

Suppose the Max 2-SAT instance $f(x_1, ..., x_n)$ has at most k satisfiable clauses, and label them $C_1, C_2, ..., C_k$. Assume ad absurdum that S satisfies more than k feature specifications. Then, the corresponding 2-SAT instance would satisfy more than k clauses, contradicting our assumption that the number of satisfiable clauses is bounded above by k. Hence, if no assignment satisfies more than k clauses, there can not be some subset of features S satisfying greater than than k feature specifications.

Now, suppose the Max 2-SAT instance $f(x_1, ..., x_n)$ has greater than k satisfiable clauses. Assume ad absurdum that S satisfies less than k feature specifications. Then, f would necessarily satisfy less than k clauses, which contradicts the assumption that f has greater than k satisfiable clauses. Thus, there can not be some subset of features S satisfying less than k feature specifications. ■

□