Exercise 11.5 Let $x_1, \ldots, x_n \in \mathbb{N}$. 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.[2]

[2] You can use the solution of one subproblem to solve another, as long as there's no circular dependencies overall.

Exercise 11.5.1. The *partition problem* asks if one can partition $x_1, \ldots, x_n$ into two parts such that the sums of each part are equal.

Solution. We claim that a polynomial time solution for the partition problem would imply a polynomial time solution for SAT. To see this, we present a polynomial time reduction from subset sum, a problem known to be hard, to the partition problem.

Notation. Given some set $S = \{s_1, \ldots, s_n\}$, we denote the sum $\sum_{s \in S} s$ with ΣS.

Consider an arbitrary instance of subset sum. That is, suppose we have a set of positive integers

$$A := \{\alpha_1, \ldots, \alpha_n\} \subseteq \mathbb{N}$$

and a positive integer target value $T \in \mathbb{N}$.

Now, let $x := 2T - \Sigma A$ and define a new set $\overline{A} := A \cup \{x\}$.

Note that if $T < \Sigma A/2$, the value of x becomes less than 0. However, if there exists some set $B = \{\beta_1, \ldots, \beta_k\} \subseteq A$ such that $\Sigma B = T$, then $A \setminus B$ sums to $\Sigma A - T \leq \Sigma A/2$. Thus, WLOG we can simply rephrase the problem to use $\Sigma A - T$ as the target value instead.

Then, notice that

$$\begin{aligned} \Sigma \overline{A} &= \Sigma A + 2T - \Sigma A \\ &= 2T. \end{aligned}$$

Correctness. (SS $\Longrightarrow$ PP) Suppose there exists some $B \subseteq A$ such that $\Sigma B = T$. Consider the partition of $\overline{A}$ defined as $\overline{B} = B \cup \{x\}$. Then,

$$\Sigma \overline{B} = T + 2T - \Sigma A.$$

The remaining partition is then $C := \overline{A} \setminus \overline{B}$, and

$$\begin{aligned} \Sigma C &= 2T - T + 2T - \Sigma A \\ &= T + 2T - \Sigma A, \end{aligned}$$

and we can see that these two sums are equal. Hence, the partition problem is solved.

(SS $\Longleftarrow$ PP) Suppose the set $\overline{A}$ has a valid partition such that each of the two subsets sum to T. Recall that $\overline{A}$ is defined as the union of A and the singleton set $\{x\}$. By the pigeonhole principle, we know that one of these subsets of $\overline{A}$ is a subset of A, whence the subset sum problem is solved. ■

Since each step in the reduction process takes only $O(1)$ or $O(n)$ time, the entire reduction can be done in polynomial time relative to the size of A. Thus, a polynomial time solution for the partition problem implies a polynomial time solution for SAT. □

Exercise 11.5.2. The *3-partition problem* asks if one can partition $x_1, \ldots, x_n$ into 3 parts such that the sums of each part are all equal.

Solution. We claim that a polynomial time solution for the 3-partition problem would imply a polynomial time solution for SAT. To see this, we present a polynomial time reduction from the partition problem, which we showed to be hard in 11.5.1.

Notation. Given sets A and B, we denote the *disjoint union* of A and B by $A \sqcup B$.

Consider an arbitrary instance of the partition problem. That is, consider a set of positive integers

$$A := \{\alpha_1, \ldots, \alpha_m\} \subseteq \mathbb{N}.$$

We wish to find two disjoint subsets $B, C \subseteq A$ such that $B \sqcup C = A$ and $\Sigma B = \Sigma C$.

Note that if ΣA is odd or the cardinality of A is less than 2, then the problem becomes impossible. Thus, WLOG we may assume that $\Sigma A = 2n$ for some $n \in \mathbb{N}$ and that A contains at least 2 elements.

Now, let $x := n$ and define a new set $\overline{A} := A \cup \{x\} \implies \Sigma\overline{A} = 3n$.

Correctness. (PP $\implies$ 3P) Assume that $\exists B \subseteq A$ such that $\Sigma B = n$. Let $C := A \setminus B$ and notice that $\Sigma C = 2n - n = n$. By construction,

$$\overline{A} = C \sqcup B \sqcup \{x\} \text{ and } \Sigma C = \Sigma B = \Sigma\{x\}.$$

Hence, the 3-partition problem is solved.

(PP $\impliedby$ 3P) Assume that there exists a valid 3-partition of $\overline{A}$. That is, assume that there exist $A_1, A_2, A_3 \subseteq \overline{A}$ such that

$$\Sigma A_1 = \Sigma A_2 = \Sigma A_3 = n \text{ and } A_1 \sqcup A_2 \sqcup A_3 = \overline{A}.$$

We already know $\{x\} \subseteq \overline{A}$ and $x = n$, so WLOG we can set $A_1 := \{x\}$. Then, we have that $A_2 \sqcup A_3 = \overline{A} \setminus A_1 = A$, and we know $\Sigma A_2 = \Sigma A_3 = n$, whence the partition problem is solved. ■

This reduction can obviously be done in polynomial time relative to the size of A. Thus a polynomial time solution for the 3-partition problem would imply a polynomial time solution for the partition problem, which we have already shown would imply a polynomial time solution for SAT. □

Exercise 11.5.3. The *any-k-partition problem* asks if one can partition $x_1, \dots, x_n$ into k parts, for any integer $k \geq 2$, such that the sums of each part are all equal.

Solution. We claim that a polynomial time solution for the k-partition problem would imply a polynomial time solution for SAT. To see this, we present an inductive proof of a polynomial time reduction from the 3-partition problem to the k-partition problem.

As stated, our base case will be the 3-partition problem, which we showed to be hard in 11.5.2. Assume that we have shown that the ℓ-partition problem is hard for all $3 \leq \ell < k$.

Consider an arbitrary instance of the $(k-1)$-partition problem. That is, consider a set of positive integers

$$A := \{\alpha_1, \dots, \alpha_m\} \subseteq \mathbb{N}.$$

We wish to find $k-1$ pairwise disjoint subsets $A_1, A_2, \dots, A_{k-1} \subseteq A$ such that

$$\bigsqcup_{1 \leq i \leq k-1} A_i = A \quad \text{and} \quad \Sigma A_1 = \Sigma A_2 = \cdots = \Sigma A_{k-1}.$$

Note that if ΣA is not divisible by $k-1$ or if the cardinality of A is less than $k-1$, then the problem is rendered impossible. Thus, WLOG we may assume that $\Sigma A = (k-1)n$ for some $n \in \mathbb{N}$ and that A contains at least $k-1$ elements.

By our inductive hypothesis, we have that the existence of a polynomial time solution for the $(k-1)$-partition problem implies the existence of a polynomial time solution for SAT.

Now, let $x := n$ and define a new set $\overline{A} := A \cup \{x\} \implies \Sigma\overline{A} = kn$.

Correctness. ($(k-1)\mathrm{P} \implies k\mathrm{P}$) Assume that there exists a valid $(k-1)$-partition for $\overline{A}$. That is, assume that there exist $k-1$ pairwise disjoint subsets $A_1, \dots, A_{k-1} \subseteq A$ such that

$$\bigsqcup_{1 \leq i \leq k-1} A_i = A \quad \text{and} \quad \Sigma A_1 = \Sigma A_2 = \cdots = \Sigma A_{k-1}.$$

By construction, we have that

$$\overline{A} = \left[\bigsqcup_{1 \leq i \leq k-1} A_i\right] \sqcup \{x\} \quad \text{and} \quad \Sigma A_1 = \Sigma A_2 = \cdots = \Sigma A_{k-1} = \Sigma\{x\} = n.$$

Hence, the k-partition problem is solved.

($(k-1)\mathrm{P} \impliedby k\mathrm{P}$) Assume that there exists a valid k-partition of $\overline{A}$. That is, assume there exist k pairwise disjoint subsets $A_1, A_2, \dots, A_k \subseteq \overline{A}$ such that

$$\overline{A} = \bigsqcup_{1 \leq i \leq k} A_i \quad \text{and} \quad \Sigma A_1 = \Sigma A_2 = \cdots = \Sigma A_k = n.$$

We already know $\{x\} \subseteq \overline{A}$ and $x = n$, so WLOG we can set $A_1 := \{x\}$. Then, we have that

$$\bigsqcup_{2 \leq i \leq k} A_i = \overline{A} \setminus A_1 = A \quad \text{and} \quad \Sigma A_2 = \Sigma A_3 = \cdots = \Sigma A_k = n,$$

whence the $(k-1)$-partition problem is solved. ■

This reduction can obviously be done in polynomial time relative to the size of A. Thus, a polynomial time solution for the k-partition problem implies a polynomial time solution for the $(k-1)$-partition problem, and by induction does so for the 3-partition problem (and equivalently for SAT). □

Exercise 11.5.4. The *almost-partition problem* asks if one can partition $x_1, \ldots, x_n$ into two parts such that the two sums of each part differ by at most 1.

Solution. We claim that a polynomial time solution for the almost-partition problem would imply a polynomial time solution for SAT. To see this, we present a polynomial time reduction from the partition problem, which we proved in 11.5.1 to be hard.

Suppose we want to solve the partition problem on a set of positive integers $A := \{a_1, ..., a_n\} \subseteq \mathbb{N}$, given a solution to the almost-partition problem as a blackbox. We transform A into the the set

$$A' := \{2a_i : a_i \in A\} = \{2a_1, ..., 2a_n\}$$

and then apply the almost-partition solution to A'. Since all elements in A' are even, it is impossible for partitions to differ by exactly 1. Hence, we claim A has a partition if and only if A' has an almost-partition.

Correctness. To prove correctness, let us first assume that A' has an almost-partition; that is, there exists some $B' \subseteq A'$ for which

$$\sum B' = \sum (A' \setminus B') \quad \text{or} \quad \sum B' = \sum (A' \setminus B') \pm 1$$

B' and $A' \setminus B'$ are both subsets of A', so we have $2 \mid \sum B'$ and $2 \mid \sum (A' \setminus B')$.

Since $\sum B' = \sum (A' \setminus B') \pm 1$ cannot be true, we must have $\sum B' = \sum (A' \setminus B')$, which can be rewritten $2 \sum B' = \sum A'$.

Let $B := \{a_i : 2a_i \in B'\} \subseteq A$. Then $2 \sum B = \sum B' = \frac{1}{2} \sum A' = \sum A$; hence, A has an exact partition.

Conversely, we now assume that A has an exact partition given by $\sum B = \sum (A \setminus B)$ for some $B \subseteq A$. If we define $B' := \{2a_i : a_i \in B\} \subseteq A$, then A' also has an exact partition given by $\sum B' = \sum (A' \setminus B')$ which is, by definition, an almost-partition of A'. ■

This reduction can clearly be performed in polynomial time relative to the input size of A and expression size of the integers in A. Since we proved above that almost-partition can be used to solve the exact partition problem, which is known to be hard, we can conclude that a polynomial time solution for the almost-partition problem would also imply a polynomial-time solution for SAT. □

Exercise 11.5.5. [3]Let n be even. The *perfect partition problem* asks if one can partition $x_1, \ldots, x_n$ into two parts such that

(a) Each part has the same sum.

(b) Each part contains the same number of x_i's.

[3]IMO, this one is the trickiest.

Solution. We claim that a polynomial time solution for the partition problem would imply a polynomial time solution for SAT. To see this, we present a polynomial time reduction from the partition problem, a problem known we showed to be hard in 11.5.1, to the perfect partition problem.

Consider an arbitrary instance of the partition problem. That is, consider a set of positive integers

$$A := \{\alpha_1, \ldots, \alpha_m\} \subseteq \mathbb{N}.$$

We wish to find two disjoint subsets $B, C \subseteq A$ such that $B \sqcup C = A$ and $\Sigma B = \Sigma C = \Sigma A/2$.

Note that if ΣA is odd or the cardinality of A is less than 2, then the problem becomes impossible. Thus, WLOG we may assume that $\Sigma A = 2n$ for some $n \in \mathbb{N}$ and that A contains at least 2 elements.

Correctness. (PP $\implies$ PPP) Assume the partition problem is solved. That is, assume that $\exists B, C \subseteq A$ such that $B \sqcup C = A$ and $\Sigma B = \Sigma C = n$. Then, we can say

$$B := \{\beta_1, \ldots, \beta_i\} \quad \text{and} \quad C := \{\gamma_1, \ldots, \gamma_j\},$$

such that $\{\beta_1, \ldots, \beta_i, \gamma_1, \ldots, \gamma_j\} = A$.

WLOG we can say $i \geq j$, that is, $|B| \geq |C|$, so define $\varphi = i - j$. Now, we have 2 cases depending on the value of φ.

Case 1 ($\varphi = 0$). This case is trivial; if $i - j = 0$ then B and C have equal cardinalities and by our hypothesis we have that $B \sqcup C = A$ and $\Sigma B = \Sigma C$. Thus the perfect partition problem is solved. ■

Case 2 ($\varphi \geq 1$). Let $S := \{s_1, \ldots, s_{\varphi+1}\}$ such that $s = 1$ for each $s \in S$. Next, let $\psi := \varphi + 1$ and $\Psi = \{\psi\}$. Note that $\Sigma S = \Sigma \Psi = \varphi + 1$. Then by hypothesis

$$B \sqcup C = A \quad \text{and} \quad \Sigma(B \cup \Psi) = \Sigma(C \cup S).$$

Now, notice that

$$\begin{aligned} |C \cup S| &= |C| + x + 1 \\ &= |C| + |B| - |C| + 1 \\ &= |B| + 1 \\ &= |B \cup \Psi| \end{aligned}$$

Hence the perfect partition problem is solved. ■

(PP $\impliedby$ PPP) Assume the perfect partition problem is solved. That is, assume that there exist disjoint $B, C \subseteq A$ such that

$$B \sqcup C = A, \qquad \Sigma B = \Sigma C = n, \qquad \text{and} \qquad |B| = |C|.$$

Obviously, the partition problem is solved. ■

Each step of the reduction process can be done in polynomial time relative to the size of A, so it follows that the a polynomial time solution to the perfect partition problem implies a polynomial time solution to the partition problem, whence a polynomial time solution to SAT. □

Exercise 11.8 (Approximating subset sum.) Let $\epsilon \in (0, 1)$ be fixed. Here we treat ϵ as a fixed constant (like $\epsilon = .1$, for 10% error); in particular, running times of the form $O(n^{O(1/\epsilon)})$ count as a polynomial.

A $(1 \pm \epsilon)$-approximation algorithm for subset sum is one that (correctly) either:

1. Returns a subset whose sum lies in the range $[(1 - \epsilon)T, (1 + \epsilon)T]$.
2. Declares that there is no subset that sums to (exactly) T.

Note that such an algorithm does not solve the (exact) subset sum problem.

Exercise 11.8.6. Suppose every input number x_i was "small", in the sense that $x_i \leq \epsilon T$. Give a polynomial time $(1 \pm \epsilon)$-approximation algorithm for this setting.

Solution. □

Exercise 11.8.7. Suppose every input number x_i was "big", in the sense that $x_i > \epsilon T$. Give a polynomial time $(1 \pm \epsilon)$-approximation algorithm for this setting.

Solution. □

Exercise 11.8.8. Now give a polynomial time $(1 \pm \epsilon)$-approximation algorithm for subset sum in the general setting (with both big and small inputs).

Solution. □

Exercise 12.10 Let $G = (V, E)$ be a directed graph with m edges and n vertices, where each vertex $v \in V$ is given an integer label $\ell(v) \in \mathbb{N}$. The goal is to find the length of the longest path[3] in G where the labels of the vertices are (strictly) increasing.

[3]Recall that a path is a walk that does not repeat vertices.

Exercise 12.10.9. Suppose G is 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 the following polynomial-time algorithm suffices.

lip-dag(s):

/ given $s \in V$ in a DAG, computes the length of the longest strictly-increasing path starting at vertex s, measured by the number of edges. */*

1. Let $m \leftarrow 0$.
2. For $(s, v) \in \delta^+(s)$ such that $\ell(v) > \ell(s)$ do
   A. m $\leftarrow$ max $(m, 1 + \texttt{lip}(v))$
3. return m

We can solve the original problem by computing max $_{\forall v \in V}$ lip-dag(v).

Runtime. If we cache the return value of lip-dag(v) for all $v \in V$ (as well as the maximum of all these return values), our algorithm has the following runtime complexity:

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

■

Correctness. The correctness of this algorithm follows from performing induction, in reverse topological order, according to the recursive specification. We claim that for all vertices $s \in V$, the algorithm correctly computes the length of the longest strictly increasing path starting with vertex s, measured in the number of edges.

In the base case, s is the last vertex in topological order, so s is a sink and the longest increasing path starting from s has length 0, as returned in the algorithm.

Now assume that the claim holds for some vertex s as well as all vertices that follow s in topological order. Take $s' \in V$ to be a vertex that immediately preceeds s in topological order, and notice that the claim holds for all v such that $(s', v) \in \delta^+(s')$. If s' has no out-neighbors with larger weight, then the algorithm correctly returns a maximum increasing path length of 0. Otherwise, the longest path starting at s' has length

$$\max_{v \,:\, (s',v) \,\in\, \delta^+(s'), \ell(v) > \ell(s')} \texttt{lip-dag}(v)$$

as computed in the algorithm.

Hence, by induction, the claim—algorithmic correctness—holds for all $s \in V$.

■

□

Exercise 12.10.10. Consider now the problem for general graphs. 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. The intuition here is that, because it is not possible to have a strictly-increasing cycles, we can treat G just as we would a DAG.

We claim the same polynomial-time algorithm as in 12.10.1 solves this problem.

lip(s):

/* *given $s \in V$, computes the length of the longest strictly-increasing path starting at vertex s, measured by the number of edges.* */

1. Let $m \leftarrow 0$.
2. For $(s, v) \in \delta^+(s)$ such that $\ell(v) > \ell(s)$ do
   A. m $\leftarrow$ max $(m, 1 + \texttt{lip}(v))$
3. return m

Again, we can solve the original problem by computing max $_{\forall v \in V}$ lip(v).

Runtime. If we cache the return value of lip(v) for all $v \in V$ (as well as the maximum of all these return values), our algorithm has the following runtime complexity:

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

■

Correctness. To clarify the DAG-like nature of this traversal (i.e. why there are no cyclic dependencies in the recursive calls), consider calling lip on some vertex $s \in V$. We make recursive calls to adjacent vertices $v \in V$, $(s, v) \in \delta^+(s)$ only when $\ell(v) > \ell(s)$, which creates a sort of topological ordering-by-weight on G.

Explicitly: we claim that for all vertices $s \in V$, the algorithm correctly computes the length of the longest strictly increasing path starting with vertex s, measured in the number of edges. To prove this, we perform induction based on the weight $\ell(v)$ of all $v \in V$.

In the base case, consider $s \in V$ with maximal weight; that is, $\ell(s) \geq \ell(v)$ for all $v \in V$. Then since $\ell(v) > \ell(s)$ is always false, lip(s) makes no recursive calls, and the algorithm returns 0, as desired.

Now assume that the claim holds for some vertex s as well as all vertices $v \in V$ such that $\ell(v) \geq \ell(s)$.

Take $s' \in V$ to be a vertex that immediately preceeds s in weight order (that is, $\ell(s') \leq \ell(s)$ and there exists no $v \in V$ for which $\ell(s') < \ell(v) < \ell(s)$), and notice that the claim holds for all v such that $(s', v) \in \delta^+(s')$ and $\ell(v) > \ell(s')$ by assumption.

If s' has no out-neighbors with larger weight, then the algorithm correctly returns a maximum increasing path length of 0. Otherwise, the longest path starting at s' has length

$$\max_{v \,:\, (s',v) \,\in\, \delta^+(s'),\ \ell(v) > \ell(s')} \texttt{lip}(v)$$

as computed in the algorithm.

By induction, we can conclude that lip(s) returns the length of the longest increasing path for all $s \in V$. ■

□

Exercise 12.10.11. Suppose instead we ask for the length of the longest path in G where G is a general graph and the labels of the vertices are weakly increasing.[4] 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.

[4] A sequence $x_1, \dots, x_k$ is weakly increasing if $x_1 \le x_2 \le \cdots \le x_k$.

Solution. We claim that a polynomial time solution for the partition problem would imply a polynomial time solution for SAT. To see this, we present a polynomial time reduction from the longest path problem, which is known to be hard, to the longest weakly increasing path problem.

The approach is simple: given a directed graph G, we label every vertex with the same weight 1 to get a vertex-weighted graph G'. This is clearly a polynomial-time reduction.

Correctness. We claim that `longest-weakly-increasing-path`(G') computes `longest-path`(G).

If we take any path P in G, then the exact same path in G' is weakly increasing, since $1 \le 1 \le 1 \le \ldots$. Conversely, take any weakly-increasing path P' in G'. Since removing vertex weights does not alter reachability, P' is also a path in G.

This correspondence preserves path lengths; hence, the maximum path length `longest-path`(G) is exactly the maximum weakly-increasing path `longest-weakly-increasing-path`(G'). ■

□