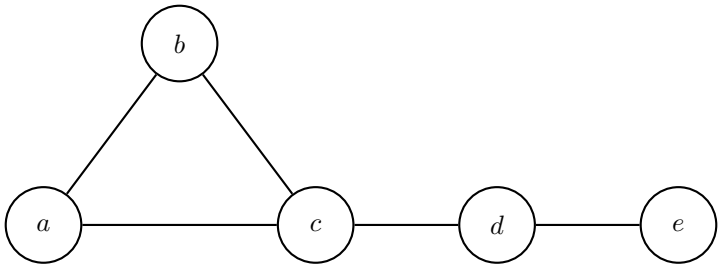
Note: Your TA probably will not cover all the problems. This is totally fine, the discussion worksheets are deliberately made long so they can serve as a resource you can use to practice, reinforce, and build upon concepts discussed in lecture, readings, and the homework.

Section 1: Backpropagation, Linear Programming, and Duality

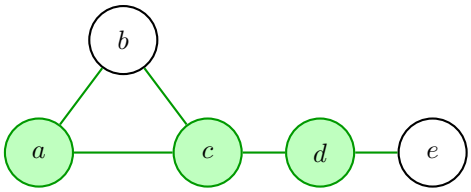
1 Vertex Cover via Linear Programming

A **vertex cover** of a graph $G = (V, E)$ is a subset $S \subseteq V$ such that every edge has at least one endpoint in S. The **minimum vertex cover** problem asks for the smallest such S.

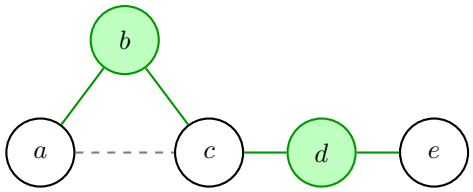
Consider the following graph G:



For example, $S = \{a, c, d\}$ is a valid vertex cover—every edge touches at least one shaded vertex. On the other hand, $\{b, d\}$ is *not* a vertex cover because edge (a, c) has neither endpoint selected:



$S = \{a, c, d\}$: valid cover (✓)



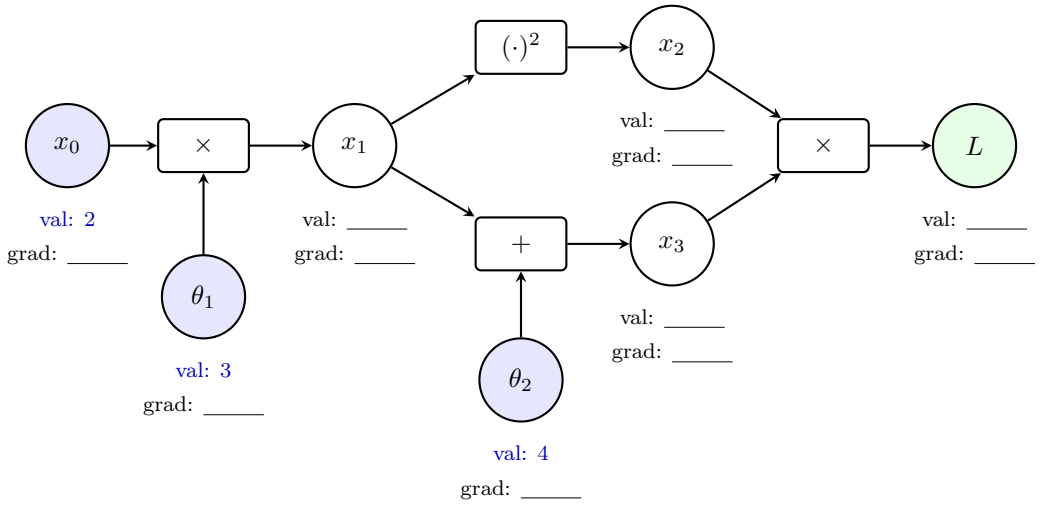
$S = \{b, d\}$: **not** a cover (×)

(a) Formulate the minimum vertex cover problem on a general graph $G = (V, E)$ as an **integer linear program** (ILP). Use a variable $x_v \in \{0, 1\}$ for each vertex, where $x_v = 1$ means "include v" and $x_v = 0$ means "exclude v." Then write out the ILP for the graph above and find its optimal value.

(b) We can relax the ILP into a linear program by replacing $x_v \in \{0, 1\}$ with $x_v \geq 0$. Using the graph G above as a concrete example, derive a **lower bound** on the LP objective by multiplying each edge constraint by a nonnegative coefficient $y_e \geq 0$ and adding. From this, write down the dual LP for a general graph. Interpret the dual LP.

Review: Backpropagation

Consider a computation graph with input $x_0 = 2$ and learnable parameters $\theta_1 = 3$, $\theta_2 = 4$. Note that x_1 feeds into *two* operations—this fork means backprop must **accumulate** gradients from both paths.



Forward Pass. Evaluate nodes in topological order (left to right):

$$x_1 = x_0 \cdot \theta_1 = 2 \cdot 3 = 6, \quad x_2 = x_1^2 = 36, \quad x_3 = x_1 + \theta_2 = 6 + 4 = 10, \quad L = x_2 \cdot x_3 = 36 \cdot 10 = 360.$$

Backward Pass. We want $\frac{\partial L}{\partial \theta_i}$ for every parameter. Seed $\frac{\partial L}{\partial L} = 1$ and propagate right to left. At each operation, apply the chain rule using stored forward values:

Step 1: $L = x_2 \cdot x_3$:

$$\frac{\partial L}{\partial x_2} = \frac{\partial (x_2 \cdot x_3)}{\partial x_2} = x_3 = 10, \qquad \frac{\partial L}{\partial x_3} = \frac{\partial (x_2 \cdot x_3)}{\partial x_3} = x_2 = 36.$$

Step 2: $x_3 = x_1 + \theta_2$ (lower path):

$$\frac{\partial L}{\partial \theta_2} = \frac{\partial L}{\partial x_3} \cdot \frac{\partial x_3}{\partial \theta_2} = 36 \cdot \frac{\partial (x_1 + \theta_2)}{\partial \theta_2} = 36 \cdot 1 = 36.$$

This path also contributes to x_1's gradient:

$$\frac{\partial L}{\partial x_3} \cdot \frac{\partial x_3}{\partial x_1} = 36 \cdot \frac{\partial (x_1 + \theta_2)}{\partial x_1} = 36 \cdot 1 = 36.$$

Step 3: $x_2 = x_1^2$ (upper path). This gives a *second* contribution to x_1's gradient:

$$\frac{\partial L}{\partial x_2} \cdot \frac{\partial x_2}{\partial x_1} = 10 \cdot \frac{\partial (x_1^2)}{\partial x_1} = 10 \cdot 2x_1 = 10 \cdot 12 = 120.$$

Step 4: Accumulate at x_1. Since x_1 feeds into *two* operations, we **sum** the contributions:

$$\frac{\partial L}{\partial x_1} = \underbrace{120}_{\text{from } x_2 \text{ path}} + \underbrace{36}_{\text{from } x_3 \text{ path}} = 156.$$

Step 5: $x_1 = x_0 \cdot \theta_1$:

$$\frac{\partial L}{\partial \theta_1} = \frac{\partial L}{\partial x_1} \cdot \frac{\partial x_1}{\partial \theta_1} = 156 \cdot \frac{\partial (x_0 \cdot \theta_1)}{\partial \theta_1} = 156 \cdot x_0 = 156 \cdot 2 = 312.$$

$$\frac{\partial L}{\partial x_0} = \frac{\partial L}{\partial x_1} \cdot \frac{\partial x_1}{\partial x_0} = 156 \cdot \frac{\partial (x_0 \cdot \theta_1)}{\partial x_0} = 156 \cdot \theta_1 = 156 \cdot 3 = 468.$$

A single backward pass gives us *all* parameter gradients simultaneously.

Generalizing to Arbitrary DAGs. A computation graph is a directed acyclic graph (DAG) with n nodes and m edges. Each node $v = f(u_1, \ldots, u_d)$ computes a primitive operation on its d parent values.

1. **Forward pass**: process nodes in **topological order** (every node is computed after all its parents). At each node, evaluate $v = f(u_1, \ldots, u_d)$ and store the result. Total cost: $O(m)$, since each edge is used once.
2. **Backward pass**: process nodes in **reverse topological order** (every node is processed before its parents). Seed $\frac{\partial L}{\partial L} = 1$. For each node v with children $c_1, \ldots, c_k$ in the graph, accumulate:

$$\frac{\partial L}{\partial v} = \sum_{j=1}^{k} \frac{\partial L}{\partial c_j} \cdot \frac{\partial c_j}{\partial v}.$$

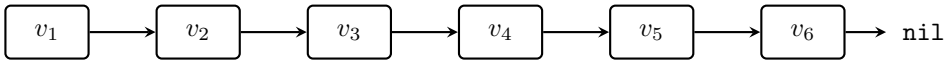
   Each local derivative $\frac{\partial c_j}{\partial v}$ is computed from the stored forward values. Total cost: $O(m)$, since each edge contributes one multiply-add.

Time Complexity. Both passes visit each edge exactly once, so the total work is $O(m)$—the same order as the forward computation itself. One backward pass computes $\frac{\partial L}{\partial v}$ for *every* node v in the graph.

Section 2: FFT & Parallelism

2 Parallel Linked List

You are given a singly linked list of n nodes. Each node v stores only a pointer `next`(v) to its immediate successor (or `nil` for the tail node).



Your goal is to compute, for *every* node v, the distance $d(v)$ from v to the tail of the list (i.e. how many edges lie between v and the end). Design a parallel algorithm achieving:

- **Work**: $T_1 = O(n \log n)$
- **Span**: $T_\infty = O(\log n)$

(a) What are the work and span of the straightforward sequential algorithm?

(b) Design a parallel algorithm that achieves the work and span bounds above.

Parallelism Quick Reference

Algorithm	**Work** (T_1)	**Span** (T_∞)	**Description**
Parallel Prefix (Scan)	$O(n)$	$O(\log n)$	All prefix sums of an n-element array
Carry-Lookahead Addition	$O(n)$	$O(\log n)$	Adds two n-bit binary numbers
Integer Multiplication	$O(n^2)$	$O(\log n)$	Multiplies two n-bit integers

3 All Pairwise Differences

You are given a set A of N integers, where every element satisfies $0 \leq a_i \leq M$. Design an algorithm that runs in $O(M \log M)$ time to output a boolean array `Distances` of size $M + 1$ such that

$$\texttt{Distances}[d] = \texttt{True} \iff \exists\, x, y \in A \text{ with } |x - y| = d.$$

Example. Let $M = 5$ and $A = \{1, 2, 5\}$. The pairwise absolute differences are:

$$|1 - 2| = 1, \quad |1 - 5| = 4, \quad |2 - 5| = 3, \quad \text{and } |a - a| = 0 \text{ for all } a \in A.$$

The set of achievable differences is $\{0, 1, 3, 4\}$, so the output is:

d	0	1	2	3	4	5
`Distances`$[d]$	T	T	F	T	T	F