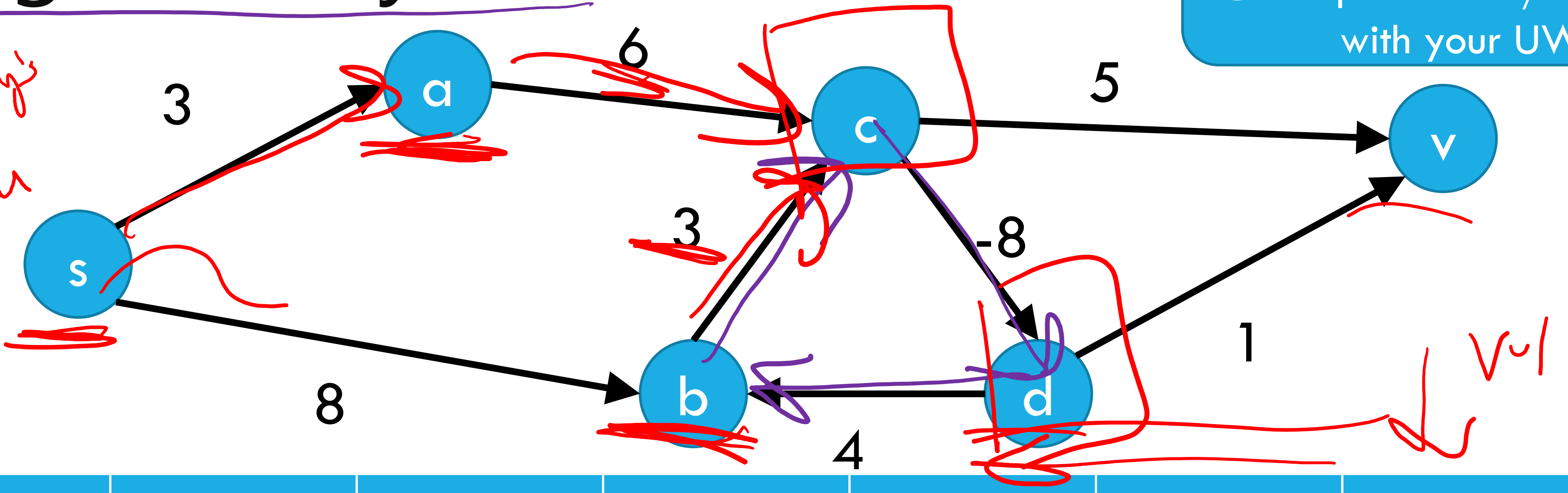
Linear Programming

CSE 417 Winter 21
Lecture 16

Negative Cycle

Fill out the poll everywhere for Activity Credit!
Go to pollev.com/cse417 and login with your UW identity

Do Monday's activity now if you haven't yet.



Vertex\\i	0	1	2	3	4	5	6
S	0	0	0	0	0 $\boxed{0+8}$	0	0
A	∞	3	3	3	3 $\boxed{3+6}$	3	3
B	∞	8	8	8	5 $\boxed{5+3}$	$\boxed{5}$	5
C	∞	∞	9	9	9	$\boxed{8}$	8
D	∞	∞	∞	1	1 $\boxed{1+4}$	1	0
V	∞	∞	∞	14	2	2	2

Negative Cycles

If you have a negative length edge: Dijkstra's might or might not give you the right answer.

And it can't even tell you if there's a negative cycle (i.e. whether some of the answers are supposed to be negative infinity)

For Bellman-Ford:

Run one extra iteration of the main loop– if any value changes, you have a negative length cycle. Some of the values you calculated are wrong.

Run a BFS from the vertex that just changed. Anything you can find should have $-\infty$ as the distance. (anything else has the correct [finite] value).

If the extra iteration doesn't change values, no negative length cycle.

Laundry List of shortest pairs (so far)

Algorithm	Running Time	Special Case only	Negative edges?
BFS	$O(m+n)$	ONLY unweighted graphs	X
Simple DP	$O(m+n)$	ONLY for DAGs	Yes!
Dijkstra's	$O(m+n\log n)$		X
Bellman-Ford	$O(mn)$		Yes!

All Pairs Shortest Paths

All Pairs

For Dijkstra's or Bellman-Ford we got the distances from the source to every vertex.

What if we want the distances from every vertex to every other vertex?

Why? Most commonly pre-computation.

Imagine you're google maps – you could run Dijkstra's every time anyone anywhere asks for directions…

Or store how to get between transit hubs and only use Dijkstra's locally.

Another Recurrence

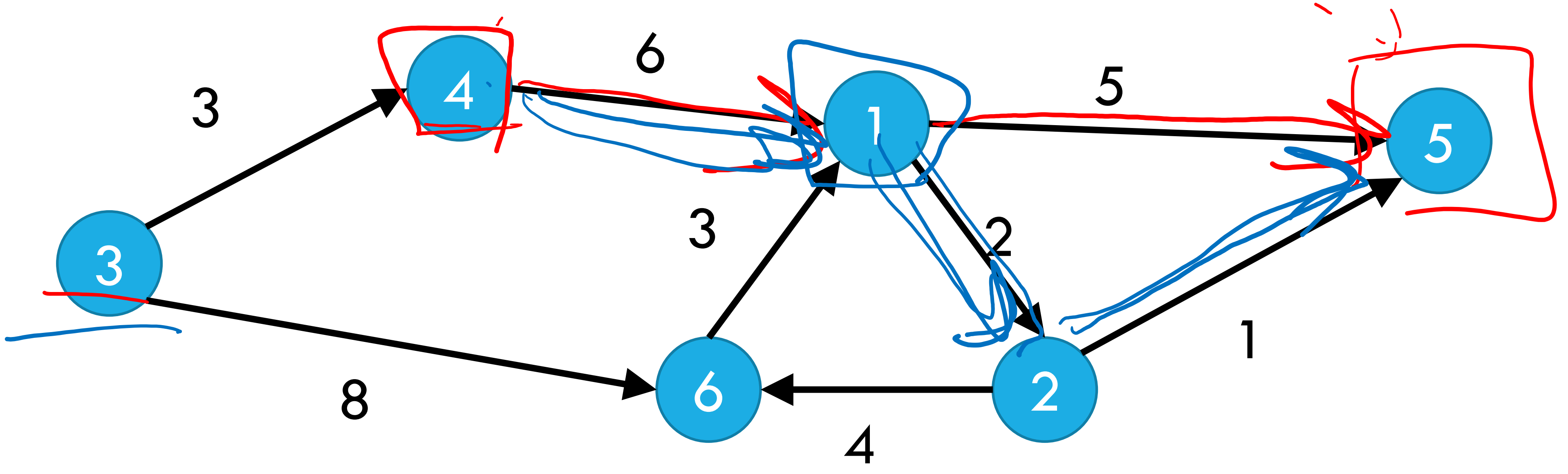
$$dist(v) = \begin{cases} 0 & \text{if } v \text{ is the source} \\ \min_{u:(u,v)\in E} \{dist(u) + weight(u,v)\} & \text{otherwise} \end{cases}$$

Another clever way to order paths.

Put the vertices in some (arbitrary) order $1,2,\ldots,n$

Let $dist(u,v,i)$ be the distance from u to v where the only **intermediate** nodes are $1,2,\ldots,i$

$dist(u, v, i)$



$dist(4,5,0) = \infty$

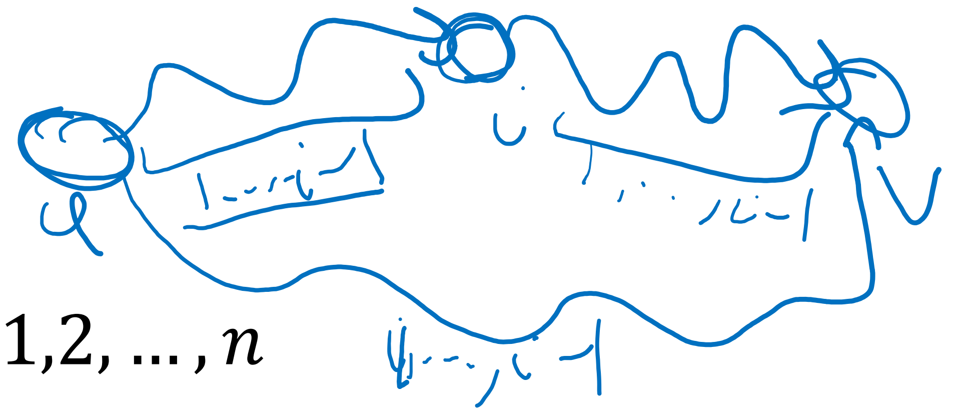
$dist(4,5,1) = 11$

$dist(4,5,2) = 9$

$dist(4,5,6) = 9$

$dist(4,5,3) = 9$

Another Recurrence



Put the vertices in some (arbitrary) order $1,2,\dots,n$

Let $dist(u,v,i)$ be the distance from u to v where the only **intermediate** nodes are $\underline{1,2,\dots,i}$

$$\text{dist}(u,v,i)=\begin{cases} weight(u,v) & \text{if } i=0, (u,v) \text{ exists} \\ 0 & \text{if } i=0, u=v \\ \infty & \text{if } i=0,\ \text{ no edge } (u,v) \\ \min\{\text{dist}(u,i,i-1)+\text{dist}(i,v,i-1),\text{dist}(u,v,i-1)\} & \text{otherwise}\end{cases}$$

Pseudocode

```
dist[][] = new int[n-1][n-1]
for(int i=0; i<n; i++)
    for(int j=0; j<n; j++)
        dist[i][j] = edge(i,j) ? weight(i,j) : ∞
for(int i=0; i<n; i++)
    dist[i][i] = 0
for every vertex r          i 0…n
    for every vertex u
        for every vertex v
            if(dist[u][r] + dist[r][v] < dist[u][v])
                dist[u][v]=dist[u][r] + dist[r][v]
```

"standard" form of the "Floyd-Warshall" algorithm. Similar to Bellman-Ford, you can get rid of the last entry of the recurrence (only need 2D array, not 3D array).

Running Time

$\underline{O(n^3)}$

How does that compare to Dijkstra's?

Running Time

$O(n^3)$

If you really want all-pairs…

Could run Dijkstra's n times…

$O(mn + n^2 \log n)$

If $m \approx n^2$ then Floyd-Warshall matches!

Floyd-Warshall also handles negative weight edges.

If $dist(u, u) < 0$ then you've found a negative weight cycle.

Takeaways

Some clever dynamic programming on graphs.

Which library to use?

Need just one source?

Dijkstra's if no negative edge weights.

Bellman-Ford if negative edges.

Need all sources?

Flord-Warshall if negative edges or $m \approx n^2$

Repeated Dijkstra's otherwise

Linear Programming

Linear Programming

Used WIDELY in business and operations research.

Excel has a linear program solver.

A very **expressive** language for problem-solving

Can represent a wide-variety of problems, including some we've already seen.

Deep, beautiful theory…that we do not have time to cover.

Outline of rest of the week

What is a linear program?

A simple example LP

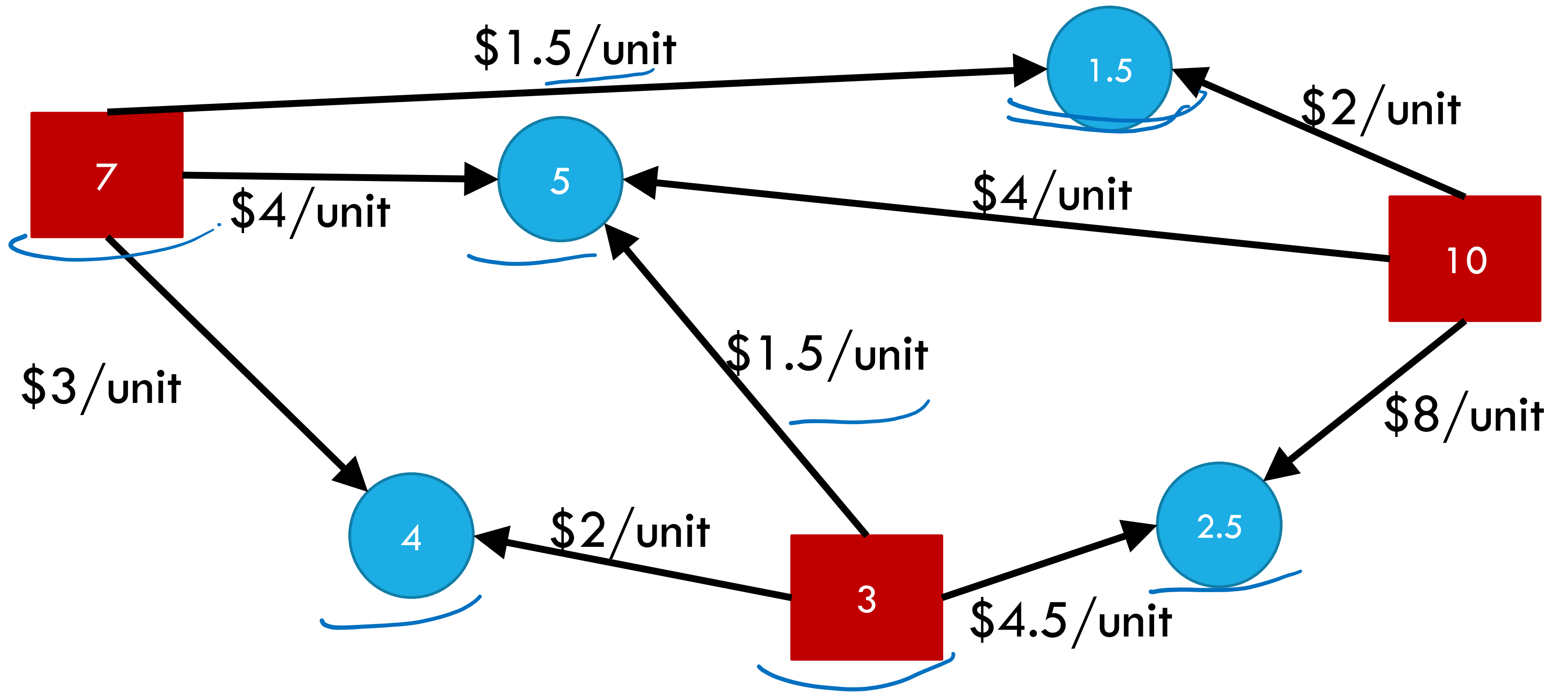
Computational Issues

An application – Vertex Cover on trees (again)

In a few weeks, we'll return to LPs as a method of approximating NP-hard problems.

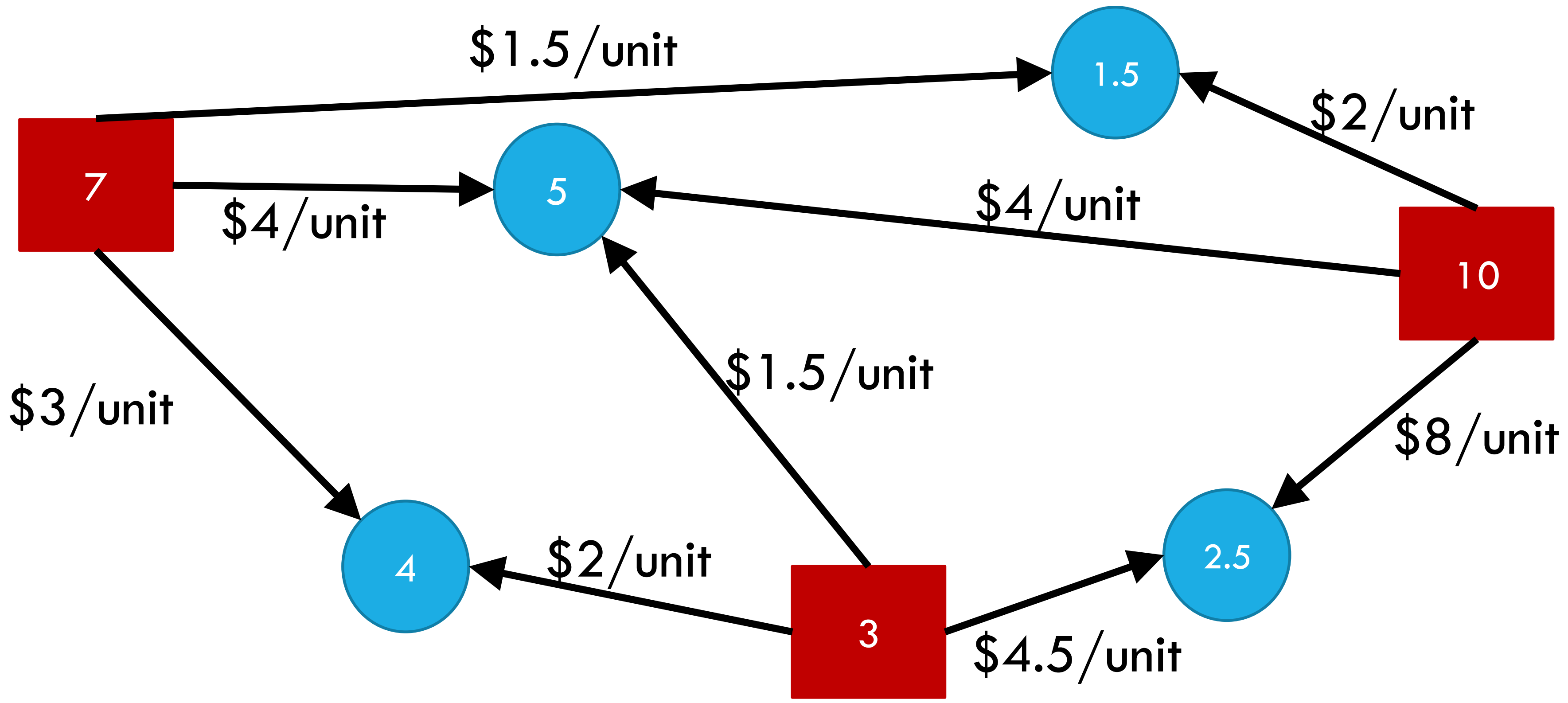
Example Problem

You're laying down soil for a bunch of new gardens. You got a few big piles of soil delivered (more than enough to cover the gardens)



Example Problem

What variables
should we use?

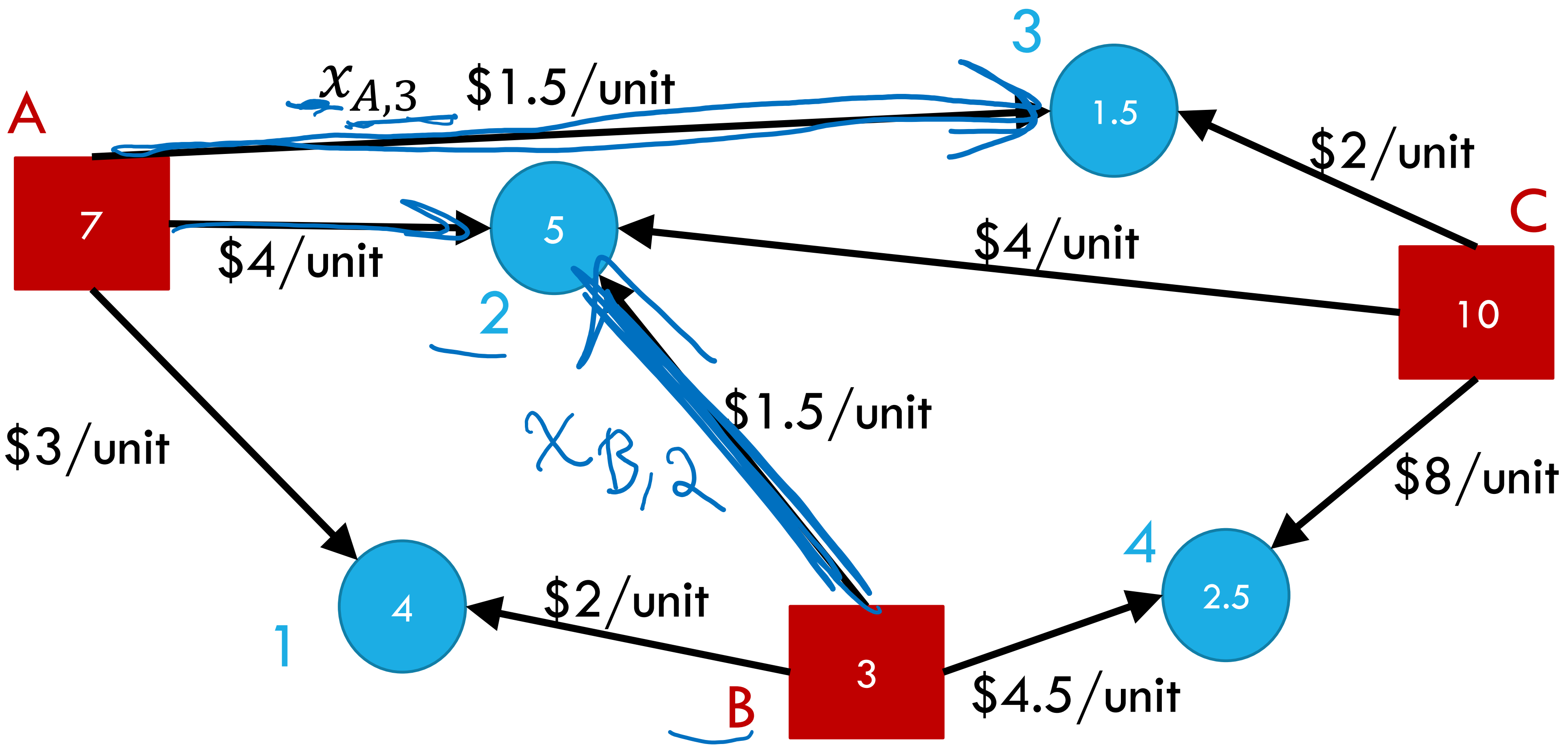


Example Problem

What variables should we use?

One for each edge (how much to move from a pile to a garden)

E.g. $x_{A,3}$ is how many units moved from A to 3.

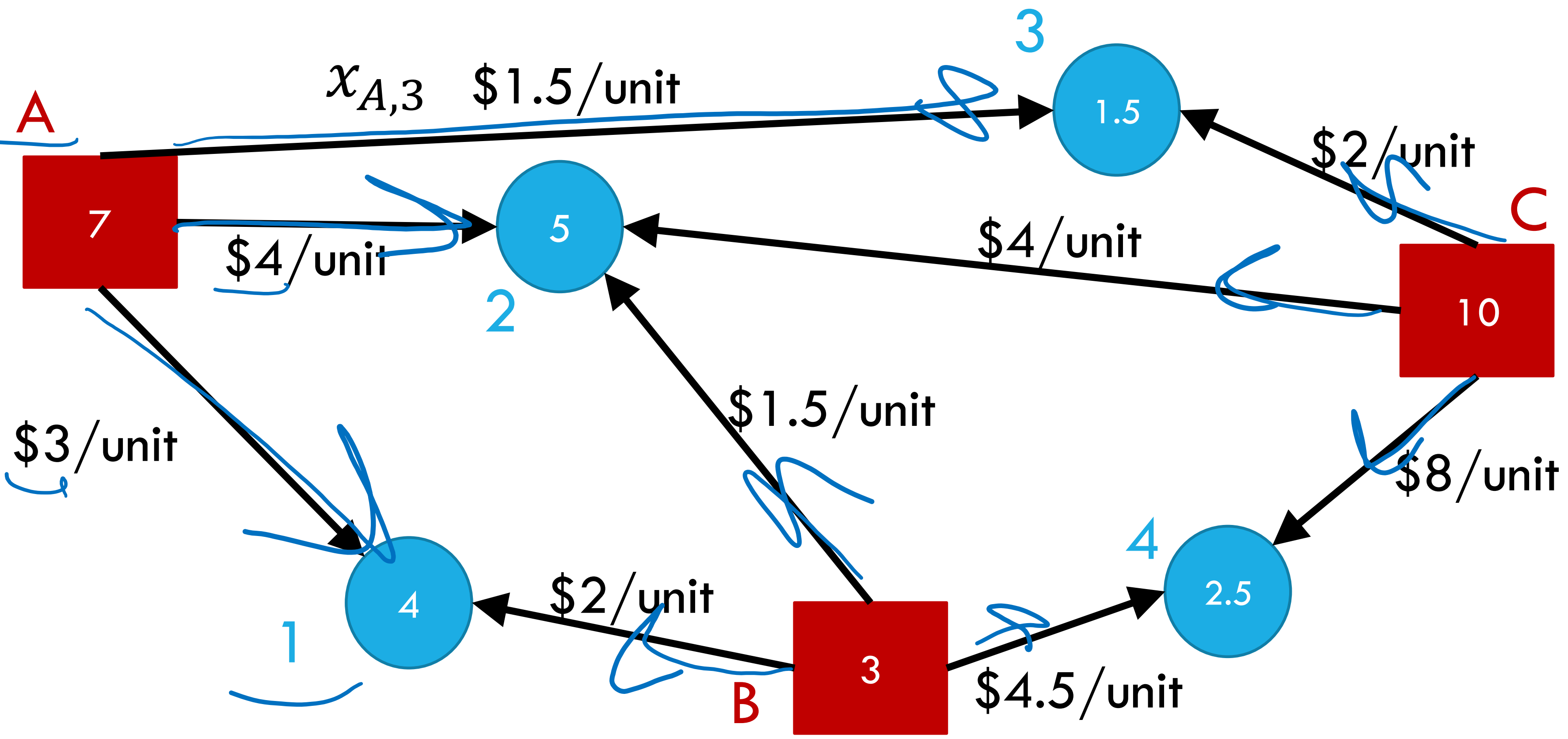


Example Problem

What's the cost (in terms of the variables)?

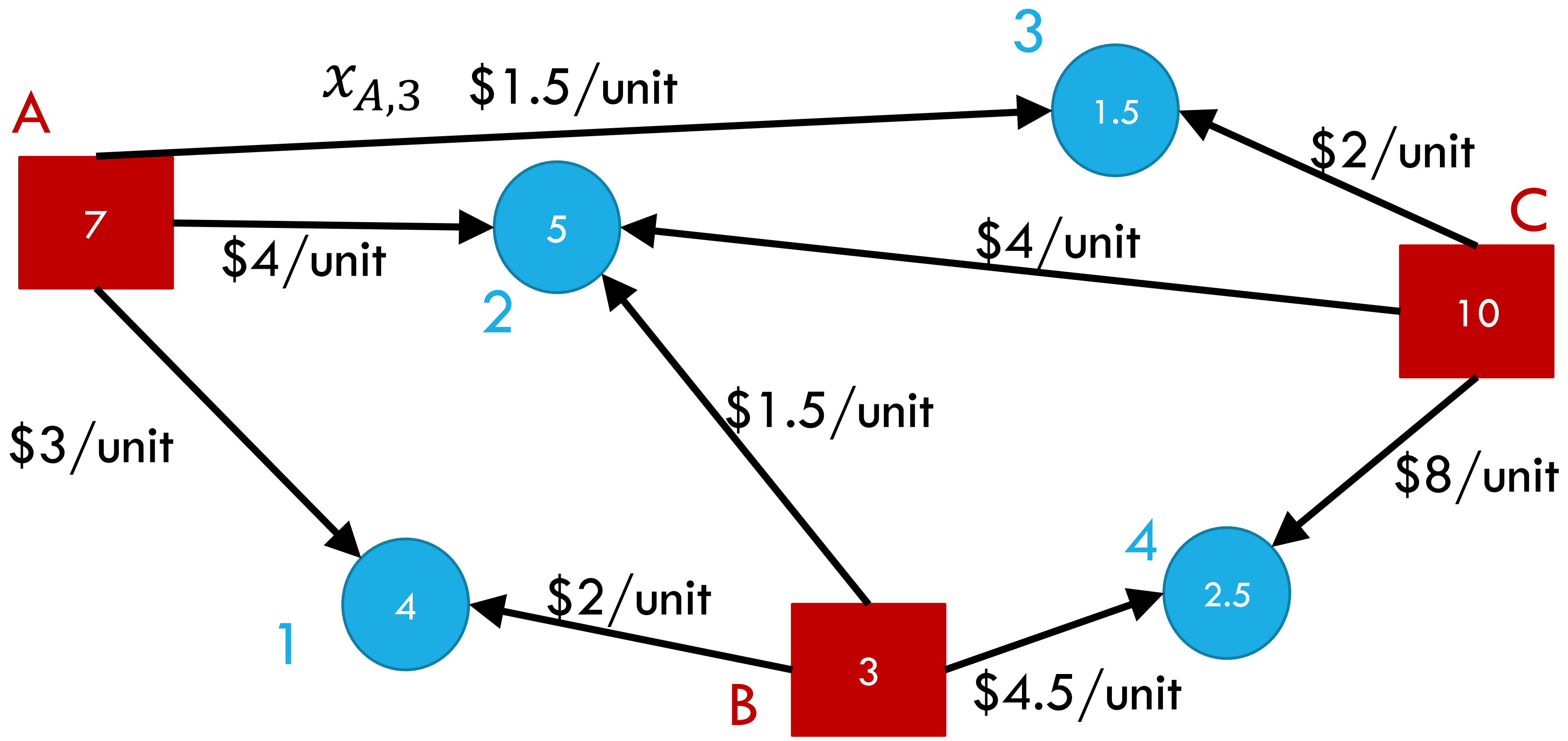
$$(x_{A,1} \cdot 3 + x_{A,2} \cdot 4 + x_{A,3} \cdot 1.5) + (x_{B,1} \cdot 2 + x_{B,2} \cdot 1.5 + x_{B,4} \cdot 4.5) + (x_{C,2} \cdot 4 + x_{C,3} \cdot 2 + x_{C,4} \cdot 8)$$

Sum cost*var for all the variables



Example Problem

What constraints are there on the variables?



Example Problem

What constraints are there on the variables?

Gardens each get enough soil:

$x_{A,1} + x_{B,1} \geq 4$

$x_{A,2} + x_{B,2} + x_{C,2} \geq 5$

$x_{A,3} + x_{C,3} \geq 1.5$

$x_{B,4} + x_{C,4} \geq 2.5$

Can't overuse a pile:

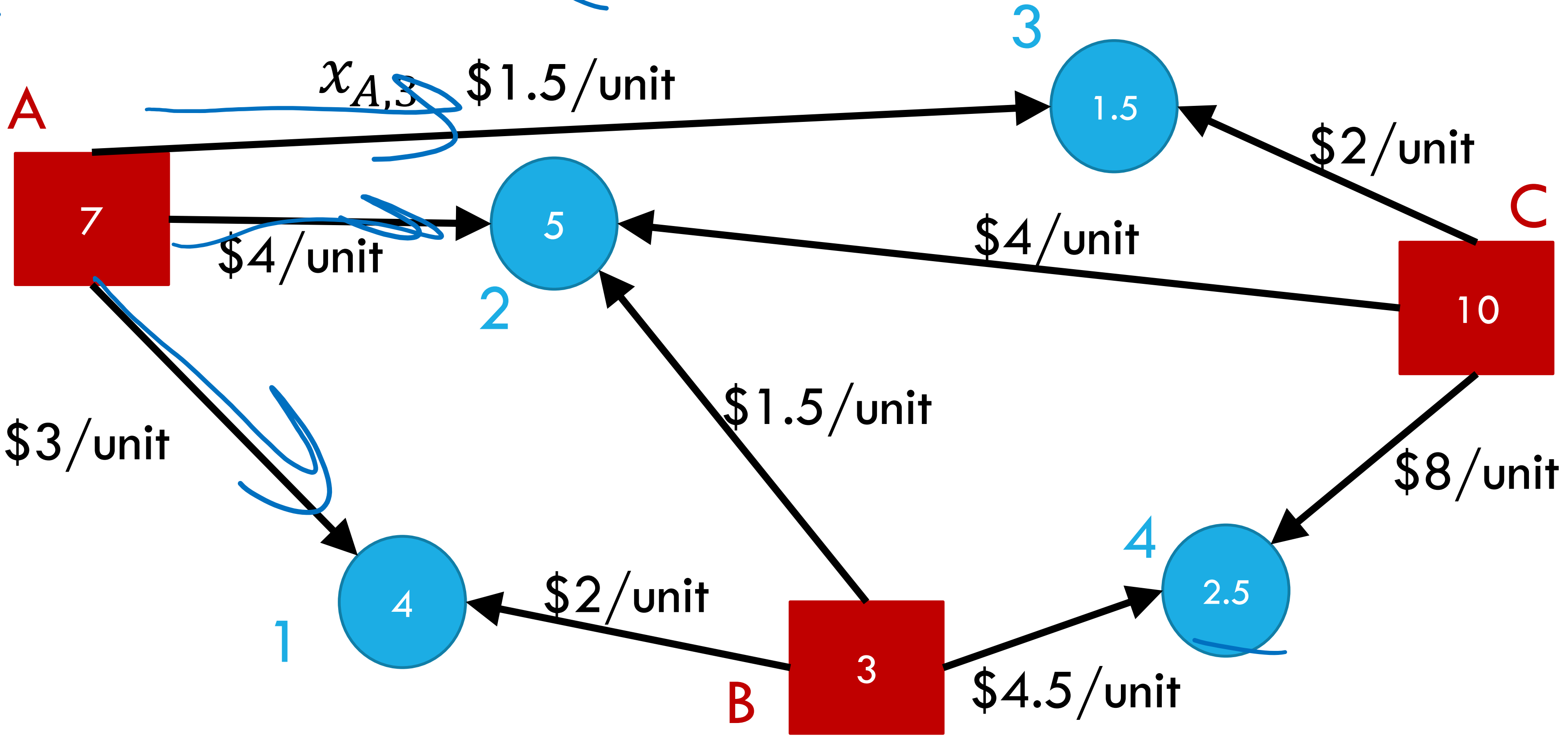
$x_{A,1} + x_{A,2} + x_{A,3} \leq 7$

$x_{B,1} + x_{B,2} + x_{B,4} \leq 3$

$x_{C,2} + x_{C,3} + x_{C,4} \leq 10$

No anti-soil:

$x_{i,j} \geq 0$ for all i, j



Full Definition

$\boxed{\text{Minimize:}}$ $\underbrace{(x_{A,1} \cdot 3 + x_{A,2} \cdot 4 + x_{A,3} \cdot 1.5) + (x_{B,1} \cdot 2 + x_{B,2} \cdot 1.5 + x_{B,4} \cdot 4.5) + (x_{C,2} \cdot 4 + x_{C,3} \cdot 2 + x_{C,4} \cdot 8)}$

Subject To:

$$\left\{\begin{array}{l}
x_{A,1} + x_{B,1} \geq 4 \\
x_{A,2} + x_{B,2} + x_{C,2} \geq 5 \\
x_{A,3} + x_{C,3} \geq 1.5 \\
x_{B,4} + x_{C,4} \geq 2.5 \\
x_{A,1} + x_{A,2} + x_{A,3} \leq 7 \\
x_{B,1} + x_{B,2} + x_{B,4} \leq 3 \\
x_{C,2} + x_{C,3} + x_{C,4} \leq 10 \\
x_{i,j} \geq 0 \text{ for all } i, j
\end{array}\right.$$

$$\underline{1} \cdot \underline{x_{A,1}} + \underline{1} \cdot x_{B,1} \underline{\geq} 4$$

A Linear Program

A linear program is defined by:

Real-valued **variables**

Subject to a list of **linear constraints**

A linear constraint is a statement of the form: $\sum a_i x_i \le c_i$
where a_i are constants, the x_i are variables and c_i is a constant.

Maximizing or minimizing a linear objective function

A linear objective function is a function of the form: $\sum b_i x_i$
where b_i are constants and the x_i are variables.

Linear constraints

Fill out the poll everywhere for Activity Credit!
Go to pollev.com/cse417 and login with your UW identity

Can you write each of these requirements as linear constraint(s)?

Some of these are tricks…

x_i times x_j is at least 5

$5x_i$ is equal to 1

$x_i \leq 5$ OR $x_i \geq 7$

x_i is non-negative.

x_i is an integer.

Linear constraints

$x_i \leq \frac{5}{x_j}$

Fill out the poll everywhere for Activity Credit!
Go to pollev.com/cse417 and login with your UW identity

Can you write each of these requirements as linear constraint(s)?

$x_i x_j \leq 5$

Some of these are tricks…

x_i times x_j is at least 5 $\quad x_i x_j \geq 5$ ✗

$5x_i$ is equal to 1 ✓ $\quad 5x_i \leq 1$ and $5x_i \geq 1 \rightarrow -5x_i \leq -1$

$x_i \leq 5$ OR $x_i \geq 7$ ✗

x_i is non-negative. $\quad x_i \geq 0 \rightsquigarrow -x_i \leq 0$

x_i is an integer. ✗

What are we looking for?

A solution (or point) is a setting of all the variables

A **feasible point** is a point that satisfies all the constraints.

An **optimal point** is a point that is feasible and has at least as good of an objective value as every other feasible point.

Example Problem

Gardens each get enough soil:

$x_{A,1} + x_{B,1} \geq 4$

$x_{A,2} + x_{B,2} + x_{C,2} \geq 5$

$x_{A,3} + x_{C,3} \geq 1.5$

$x_{B,4} + x_{C,4} \geq 2.5$

Can't overuse a pile:

$x_{A,1} + x_{A,2} + x_{A,3} \leq 7$

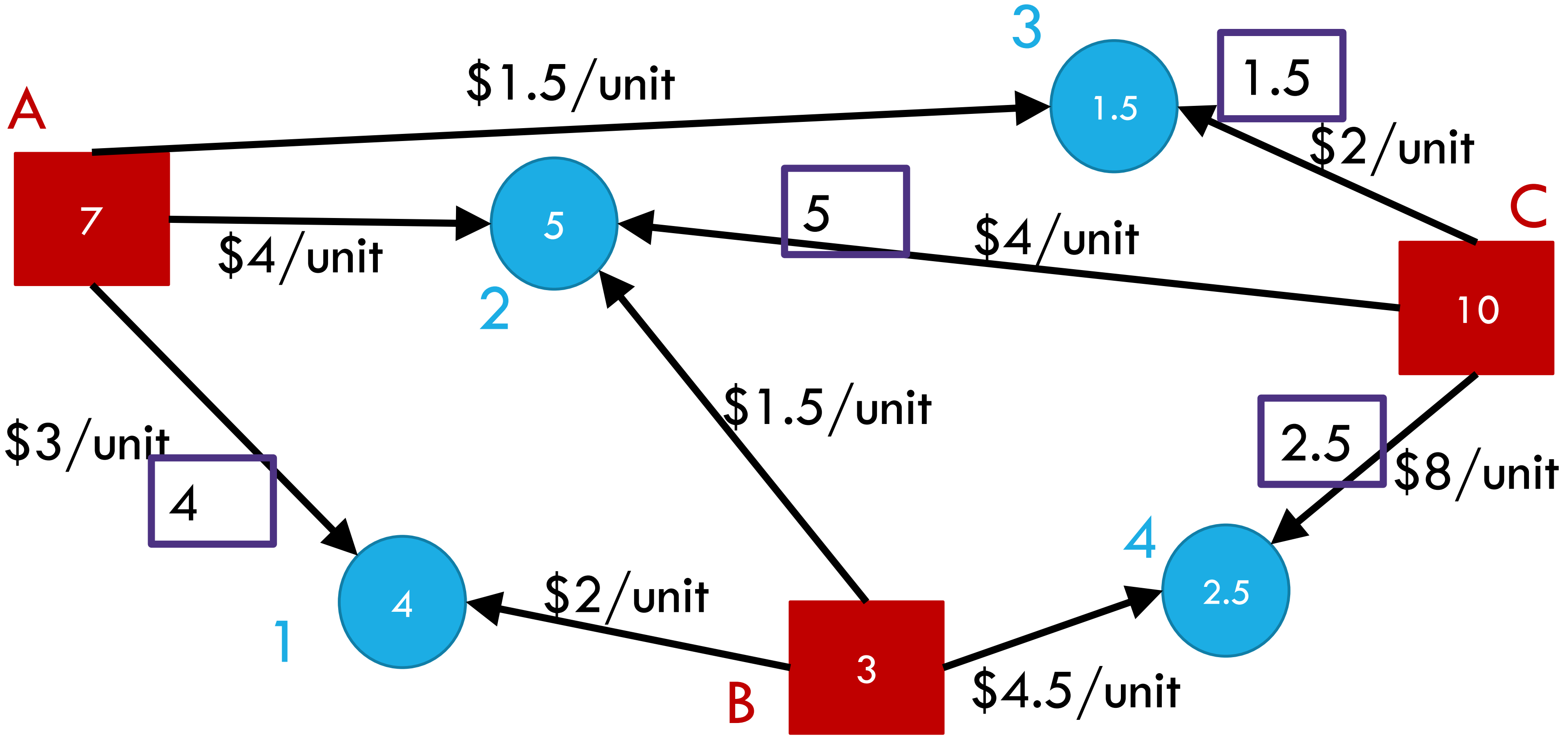
$x_{B,1} + x_{B,2} + x_{B,4} \leq 3$

$x_{C,2} + x_{C,3} + x_{C,4} \leq 10$

No anti-soil:

$x_{i,j} \geq 0$ for all i, j

A feasible point.
Objective: 55



Example Problem

Gardens each get enough soil:

$x_{A,1} + x_{B,1} \geq 4$

$x_{A,2} + x_{B,2} + x_{C,2} \geq 5$

$x_{A,3} + x_{C,3} \geq 1.5$

$x_{B,4} + x_{C,4} \geq 2.5$

Can't overuse a pile:

$x_{A,1} + x_{A,2} + x_{A,3} \leq 7$

$x_{B,1} + x_{B,2} + x_{B,4} \leq 3$

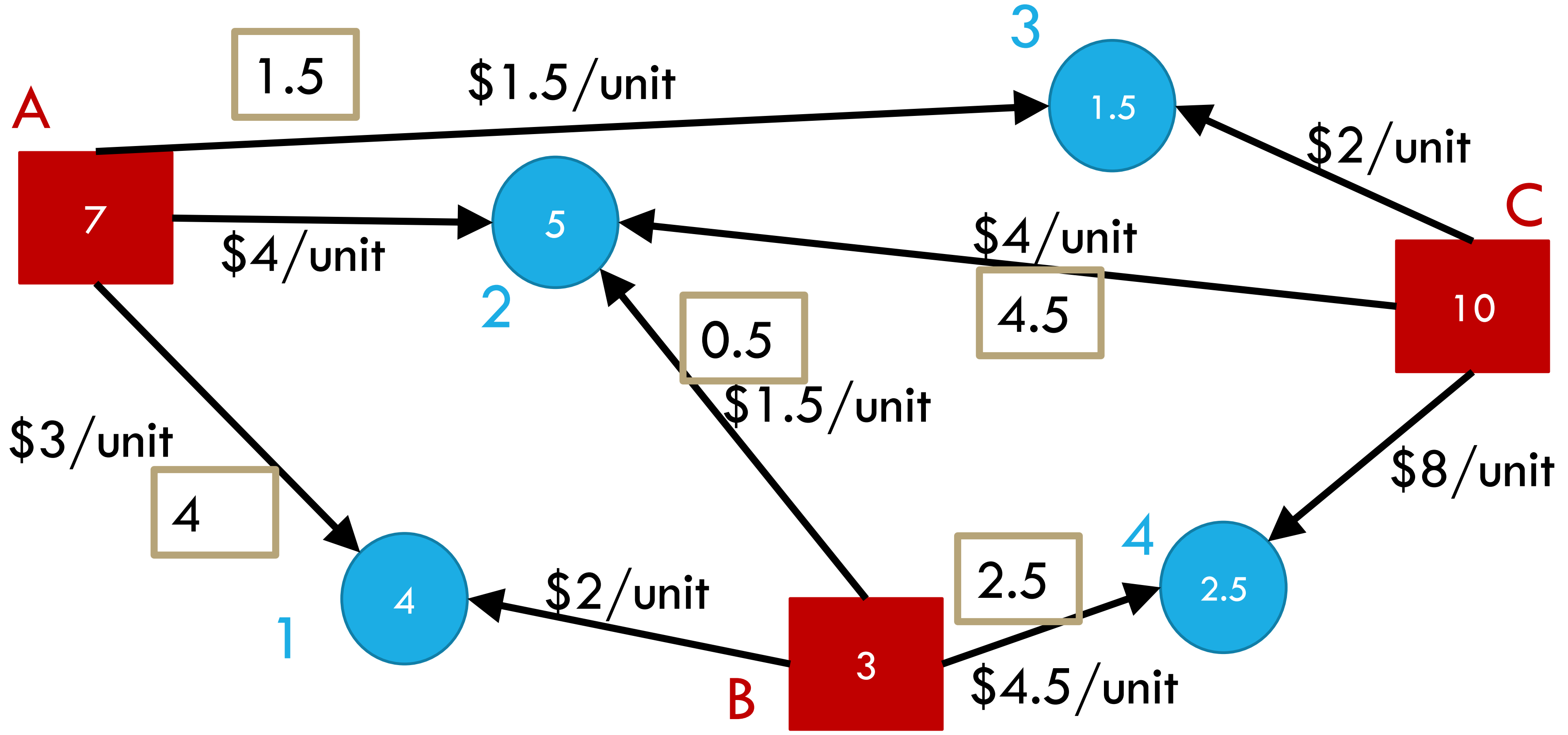
$x_{C,2} + x_{C,3} + x_{C,4} \leq 10$

No anti-soil:

$x_{i,j} \geq 0$ for all i, j

A feasible point.
Objective: 44.25

This is an optimal point. There are others!



Another Question

Change the problem

Instead of infinitely divisible dirt…

What if instead we’re moving whole unit things (the dirt is in bags we can’t open or we’re moving bikes or plants or anything else that can’t be split)

Well, the constraints will change (your “demand” and “supplies” will be integers)

Non-Integrality

Some linear programs have optimal solutions that have some (or all) variables as non-integers (even with only integers in the objective function and constraints) .

For dirt or water or anything arbitrarily divisible, no big deal!

For cell phones or bicycles…possibly a big deal! (also possibly not!)

What do you do if you need integers?

Integer Programs are linear programs where you can mark some variables as needing to be integers.

In practice – often still solvable (Excel also has a solver for these problems). But no longer guaranteed to be efficient.

And "just rounding" an LP answer often gets you really close.

In theory – lots of theory has been done for when the best answer will be an integer

But sometimes there's just not a lot to be done…

Solving LPs

For this class, we're only going to think about library functions to solve linear programs (i.e. we won't teach you how any of the algorithms work)

The most famous is the **Simplex Method** – can be quite slow (exponential time) in the worst case. But rarely hits worst-case behavior.

Very fast in practice. Idea: jump from extreme point to extreme point.

The **Ellipsoid Method** was the first theoretically polynomial time algorithm $O(n^6)$ where n is the number of bit needed to describe the LP (usually $\approx$ the number of constraints)

Interior Point Methods are faster theoretically, and starting to catch up practically. $O(n^{2.373})$ theoretically

Extra Practice

You have 20 pounds of gold and 40 pounds of silver.

You can turn 2 pounds of silver and 3 pound of gold into a (really heavy) necklace that can be sold for $\$10$.

You can also turn 9 pounds of silver and 1 pound of gold into a (really fancy) shield that can be sold for $\$15$.

How many of each should you make to maximize your profit? (fractional values are ok for this problem

Extra Practice

You have 20 pounds of gold and 40 pounds of silver.

You can turn 2 pounds of silver and 3 pound of gold into a (really heavy) necklace that can be sold for $\$10$.

You can also turn 9 pounds of silver and 1 pound of gold into a (really fancy) shield that can be sold for $\$15$.

How many of each should you make to maximize your profit?

Max $10N + 15S$

Subject to

$2N + 9S \leq 40$

$3N + S \leq 20$

Plugging into an LP solver would give $N = 5.6$ and $S = 3.2$
(we'll give resources for solvers next lecture)