Linear Programming

CSE 417 Winter 24
Lecture 16

Some Logistics

What should you do if you get feedback on a problem you don't understand?

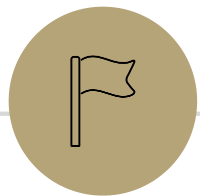
Option 1: Submit a regrade request.

That will go to the TA who graded that problem; they'll be able to say the most exactly what they noticed.

Can submit even if you think the grade is probably right, but you don't understand what was wrong with your submission. (but don't submit for "hint on how to fix it")

Option 2: Come talk to us at Office Hours.

The TA you talk to might not have graded that problem! So they might not be sure, but they'll do their best (and they can talk to other staff members offline and get back to you).



DP on graphs

First Half of Today

Dynamic Programming on Graphs

We’re building up to “Bellman-Ford”

A very clever algorithms – we won’t ask you to be as clever.

But a standard library function, so it’s good to know.

And deriving them together is good for practicing DP skills.

Want to understand: why is DP on graphs with cycles harder than DP on trees?

Shortest Paths

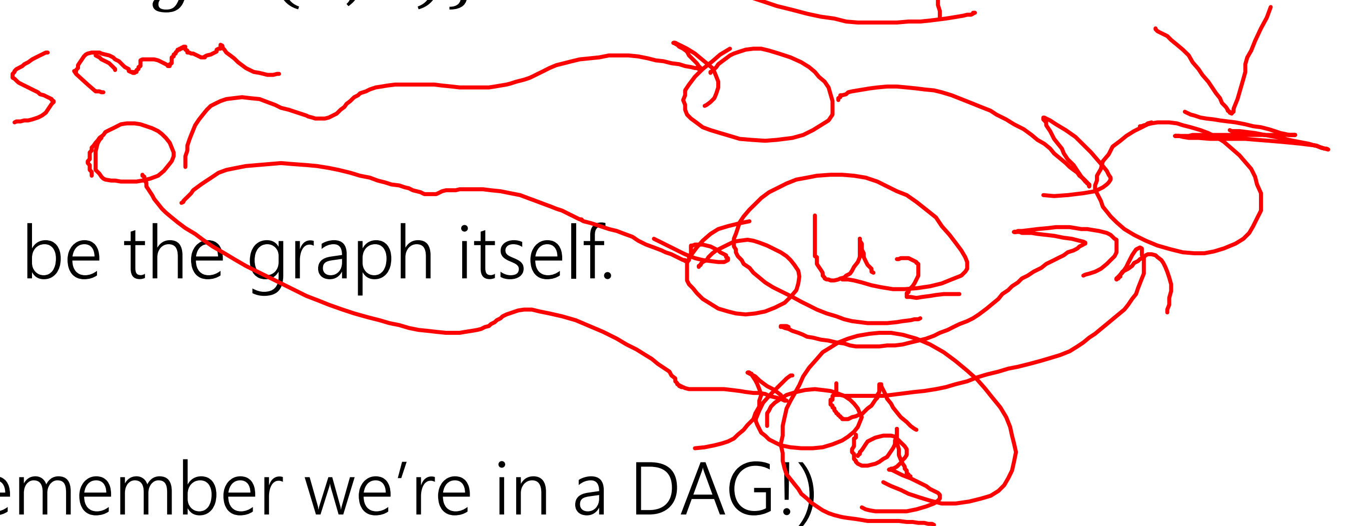
Shortest Path Problem
Given: A directed graph and a vertex s Find: The length of the shortest path from s to t.

The length of a path is the sum of the edge weights.

Baseline: Dijkstra's Algorithm

A recurrence

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



Our memoization structure can be the graph itself.

What's an evaluation order? (Remember we're in a DAG!)

A recurrence

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

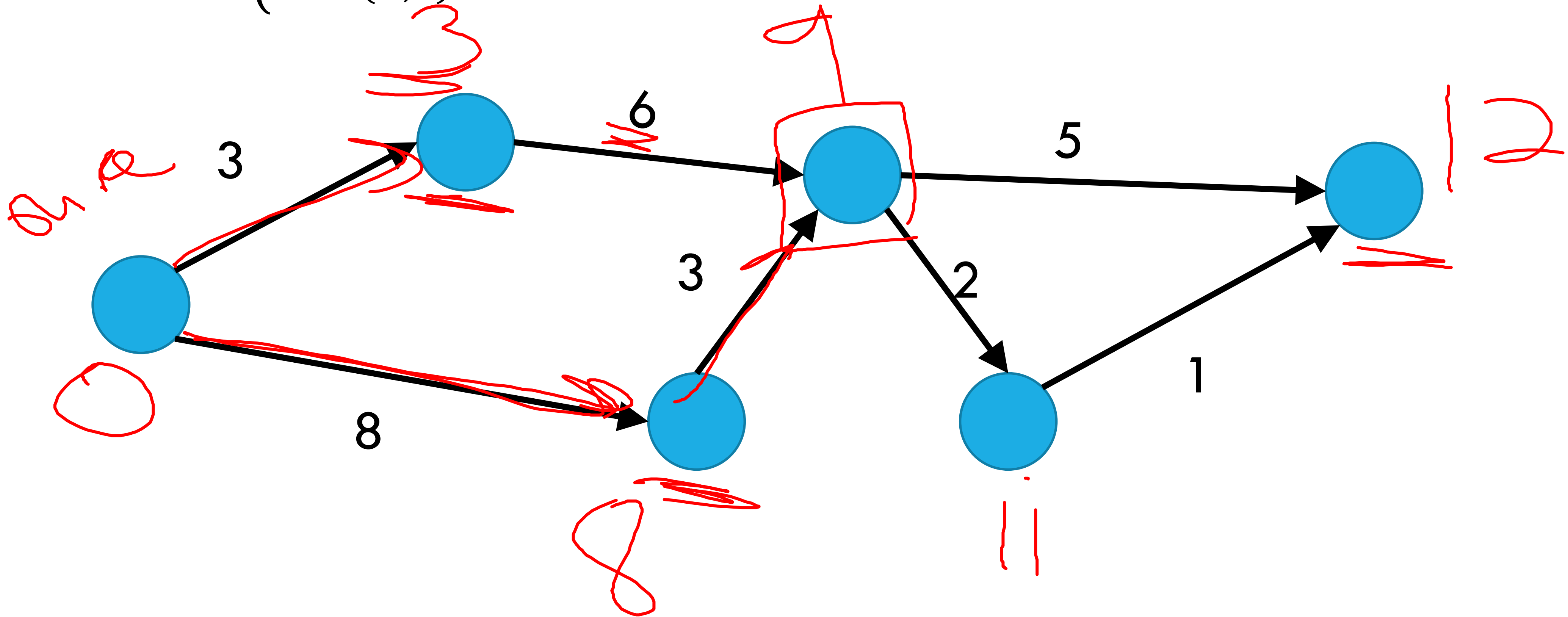
Our memoization structure can be the graph itself.

What's an evaluation order? (Remember we're in a DAG!)

A topological sort! – we need to have distances for all incoming edges calculated.

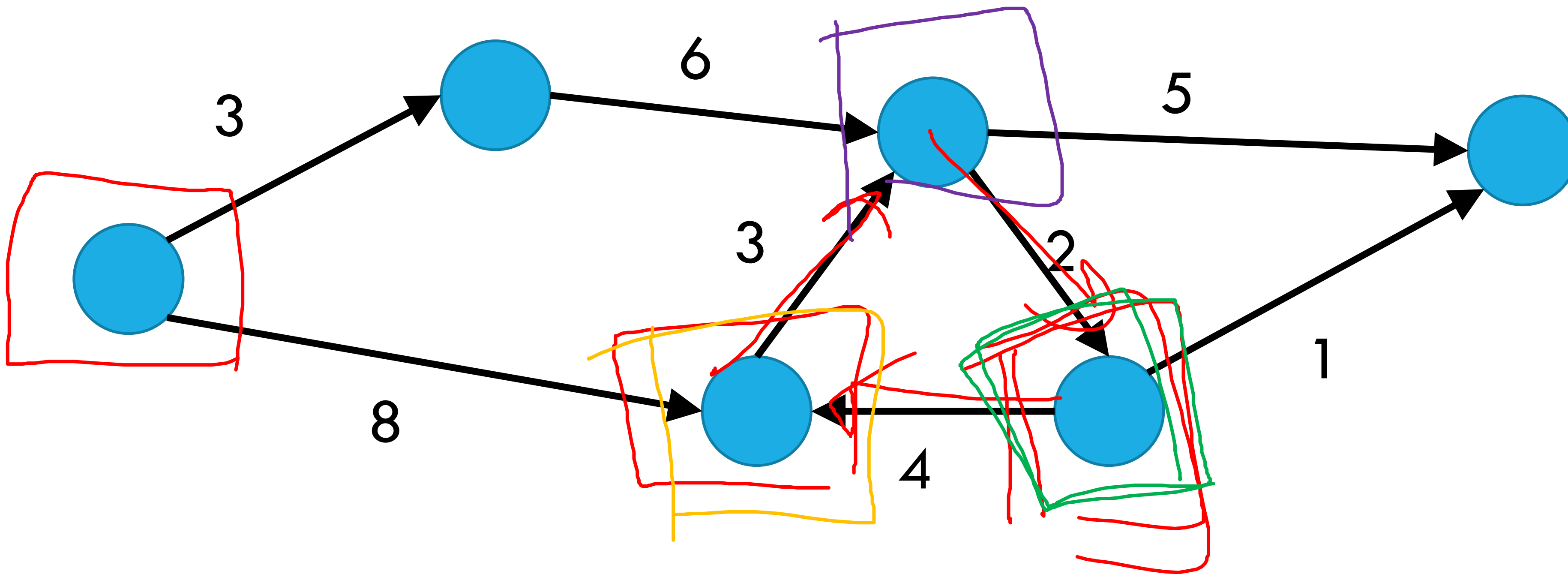
In a DAG

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



What about cycles?

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



Cycles

We need some way to “order” the paths.

I.e. we need to be sure we always have **something** to look up.

It doesn’t have to be the perfect distance necessarily…

As long as we’ll realize it and update later

And as long as we can fix it to the true distance eventually.

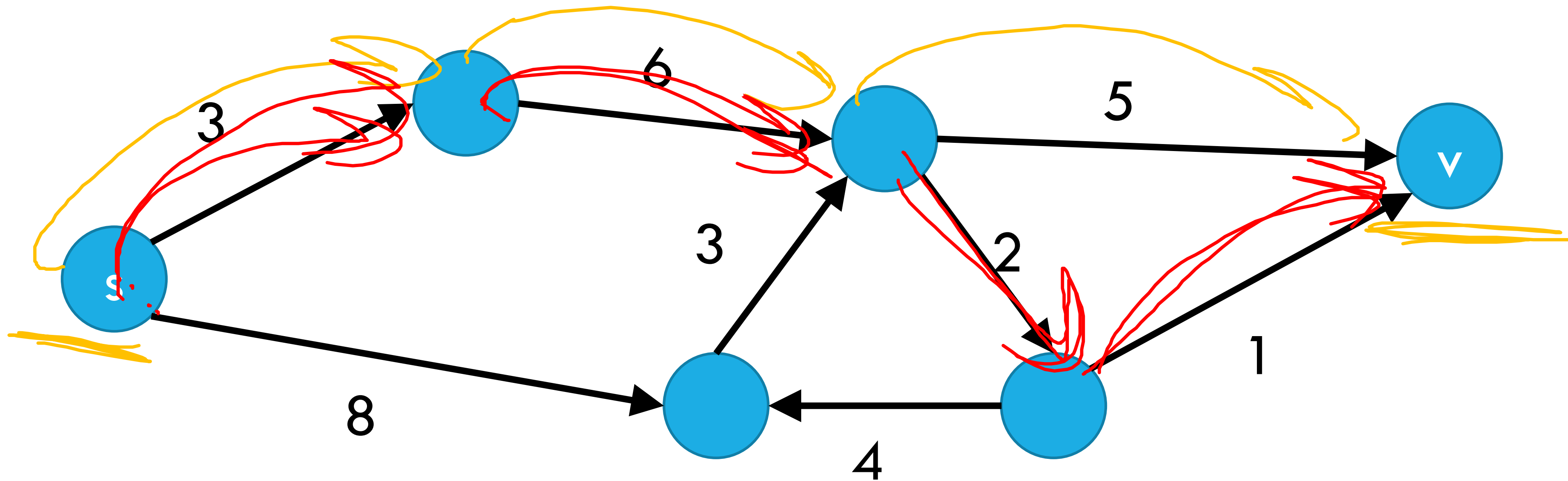
Ordering

Instead of $dist(v)$, (the true distance) right from the start, we'll let $dist(v,i)$ to be the length of the shortest path from the source to v that uses at most i edges.

That breaks ties – counting the number of edges required!

$$dist(v,i) =$$

Distances



$dist(v, 2) = \infty$ (can't get there in 2 hops)

$dist(v, 3) = 14$

$dist(v, 4) = 12$

Ordering

Instead of $dist(v)$, (the true distance) right from the start, we'll let

$dist(v, i)$ to be the length of the shortest path from the source to v that uses at most i edges.

That breaks ties – counting the number of edges required!

$dist(v, i) =$

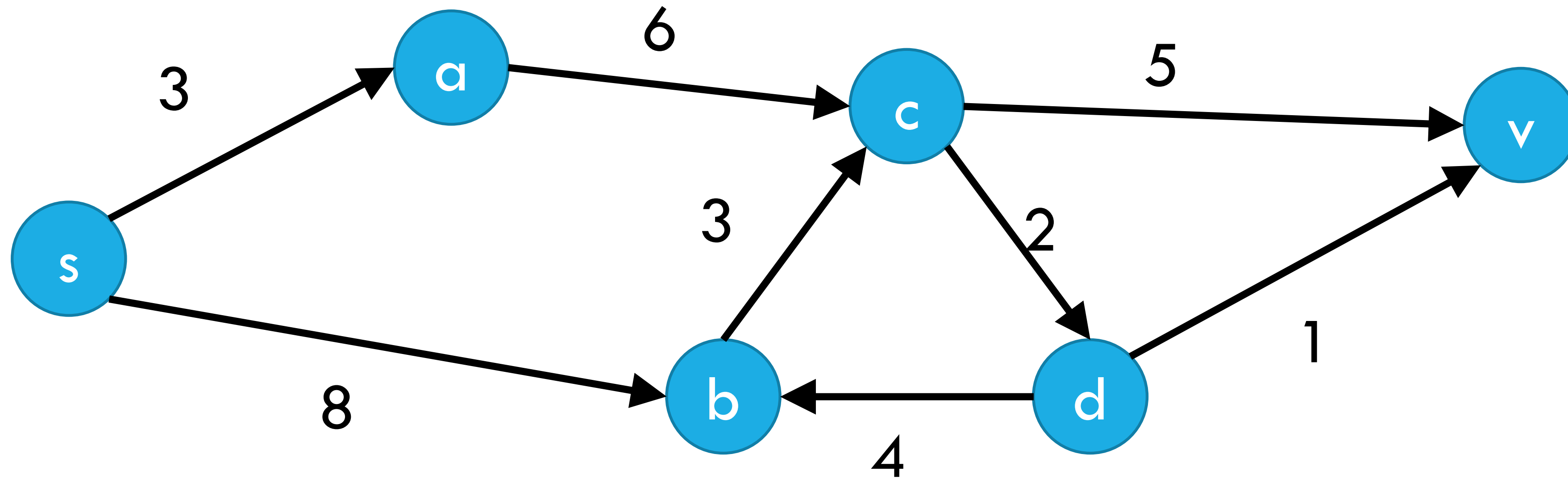
Ordering

Instead of $dist(v)$, we want the

$dist(v, i)$ to be the length of the shortest path from the source to u that uses at most i edges.

$$dist(v,i) = \begin{cases} 0 & \text{if } i = 0 \text{ and } v \text{ is the source} \\ \infty & \text{if } i = 0 \text{ and } v \text{ is not the source} \\ \min\left\{ \min_{u:(u,v)\in E} \{dist(u, i-1)\} + w(u,v), dist(v, i-1) \right\} & \text{o/w} \end{cases}$$

Sample calculation



Vertex\i	0	1	2	3	4	5
S	0	0	0	0	0	0
A	∞	3	3	3	3	3
B	∞	8	8	8	8	8
C	∞	∞	9	9	9	9
D	∞	∞	∞	11	11	11
V	∞	∞	∞	14	12	12

Pseudocode

```
Initialize source.dist[0]=0, u.dist[0]=∞ for others
for(i from 1 to ??)
    for(every vertex v) //what order?
        v.dist[i] = v.dist[i-1]
        for(each incoming edge (u,v))//hmmm
            if(u.dist[i-1]+weight(u,v)<v.dist[i])
                v.dist[i]=u.dist[i-1]+weight(u,v)
            endIf
        endFor
    endFor
endFor
```

$$dist(v,i) = \begin{cases} 0 & \text{if } i = 0 \text{ and } v \text{ is the source} \\ \infty & \text{if } i = 0 \text{ and } v \text{ is not the source} \\ \min\left\{ \min_{u:(u,v)\in E} \{dist(u,i-1)\} + w(u,v), dist(v,i-1) \right\} & \end{cases}$$

Pseudocode

```
Initialize source.dist[0]=0, u.dist[0]=∞ for others
for(i from 1 to n-1)
    for(every vertex
        v.dist[i] = v.dist[i-1]
        for(each incoming edge (u,v))//hmmm
            if(u.dist[i-1]+weight(u,v)<v.dist[i])
                v.dist[i]=u.dist[i-1]+weight(u,v)
            endIf
        endFor
    endFor
endFor
```

The shortest path will never need more than $n-1$ edges
(more than that and you've got a cycle)

Pseudocode

```
Initialize sour
for(i from 1 to
    for(every vertex v) //what order?
        v.dist[i] = v.dist[i-1]
        for(each incoming edge (u,v))//hmmm
            if(u.dist[i-1]+weight(u,v)<v.dist[i])
                v.dist[i]=u.dist[i-1]+weight(u,v)
            endIf
        endFor
    endFor
endFor
```

Only ever need values from the last iteration
Order doesn't matter!!

Pseudocode

```
Initialize source.dist[0]=0, u.dist[0]=∞ for others
for(i from 1 to n-1)
   for(every vertex v) //any order
      v.dist[i] = v.dist[i-1]
      for(each incoming edge (u,v))//hmmm
         if(u.dist[i-1]+weight(u,v)<v.dist[i])
            v.dist[i]=u.dist[i-1]+weight(u,v)
         endIf
      endFor
   endFor
endFor
```

Graphs don't usually have easy access to their incoming edges (just the outgoing ones)

Pseudocode

```
Initialize source.dist[0]=0, u.dist[0]=∞ for others
for(i from 1 to n-1)
    for(every vertex v) //any order
        v.dist[i] = v.dist[i-1]
        for(each incoming edge (u,v))//hmmm
            if(u.dist[i-1]+weight(u,v)<v.dist[i])
                v.dist[i]=u.dist[i-1]+weight(u,v)
            endIf
        endFor
    endFor
endFor
```

But the order doesn't matter – as long as we check every edge, the processing order is irrelevant.
So if we only have access to outgoing edges…

Pseudocode

```
Initialize source.dist[0]=0, u.dist[0]=∞ for others
for(i from 1 to n-1)
    set u.dist[i] to u.dist[i-1] for every u
    for(every vertex u) //any order
        for(each outgoing edge (u,v))//better!
            if(u.dist[i-1]+weight(u,v)<v.dist[i])
                v.dist[i]=u.dist[i-1]+weight(u,v)
            endIf
        endFor
    endFor
endFor
```

Pseudocode

```
Initialize source.dist[0]=0, u.dist[0]=∞ for others
for(i from 1 to n-1)
    set u.dist[i] to u.dist[i-1] for every u
    for(every vertex u) //any order
        for(each outgoing edge (u,v))//better!
            if(u.dist[i-1]+weight(u,v)<v.dist[i])
                v.dist[i]=u.dist[i-1]+weight(u,v)
            endIf
        endFor
    endFor
endFor
```

We don't really need all the different values…
Just the most recent value.

Pseudocode

```
Initialize source.dist=0, u.dist=∞ for others
for(i from 1 to n-1)
    set u.dist[i] to u.dist[i-1] for every u
    for(every vertex u) //any order
        for(each outgoing edge (u,v))//better!
            if(u.dist+weight(u,v)<v.dist)
                v.dist=u.dist+weight(u,v)
            endIf
        endFor
    endFor
endFor
```

We don't really need all the different values…
Just the most recent value.

Pseudocode

```
Initialize source.dist=0, u.dist=∞ for others
for(i from 1 to n-1)
    for(every vertex u) //any order
        for(each outgoing edge (u,v))//better!
            if(u.dist+weight(u,v)<v.dist)
                v.dist=u.dist+weight(u,v)
            endIf
        endFor
    endFor
endFor
```

We don’t really need all the different values…
Just the most recent value.

A Caution

We did change the code when we got rid of the indexing

You might have a mix of `dist[i]`,`dist[i+1]`,`dist[i+2]`,… at the same time.

That's ok!

You'll only "override" a value with a better one.

And you'll eventually get to $dist(u, n-1)$

After iteration i, u stores $dist(u, k)$ for some $k \geq i$.

Exit early

If you made it through an entire iteration of the outermost loop and don't update any $dist()$

Then you won't do any more updates in the next iteration either. You can exit early.

More ideas to save constant factors on Wikipedia (or the textbook)

Laundry List of shortest pairs (so far)

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

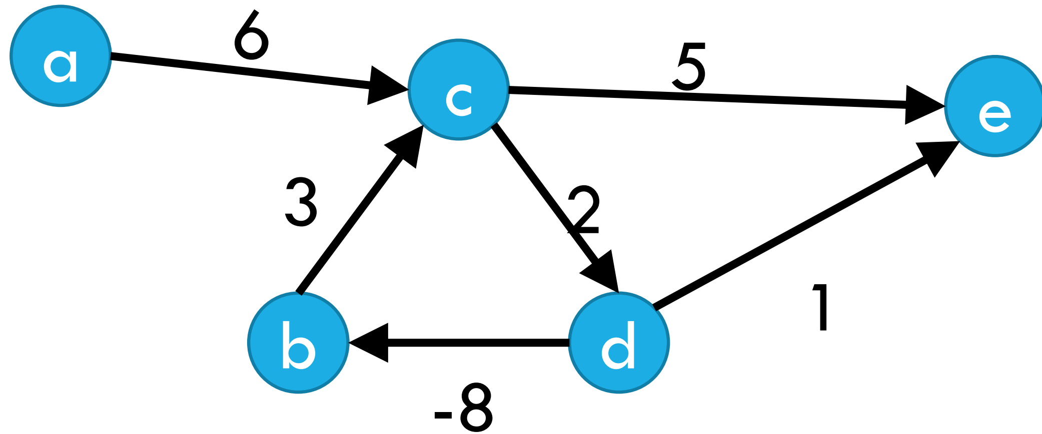
Pseudocode

```
Initialize source.dist=0, u.dist=∞ for others
for(i from 1 to n-1)
   for(every vertex u) //any order
      for(each outgoing edge (u,v))//better!
         if(u.dist+weight(u,v)<v.dist)
            v.dist=u.dist+weight(u,v)
         endIf
      endFor
   endFor
endFor
```

What happens if there's a negative cycle?

Negative Edges

Negative Cycles

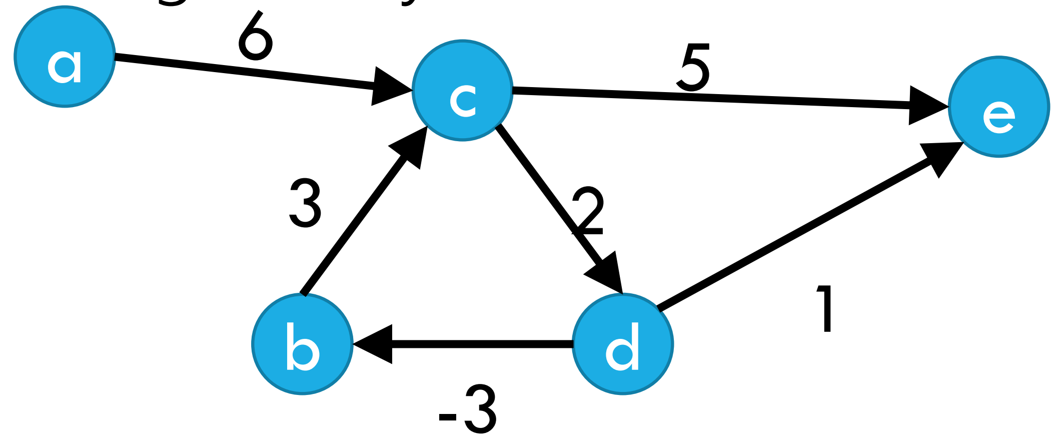


The fastest way from a to e

(i.e. least-weight walk) isn't defined!

No valid answer ($-\infty$)

Negative edges, but only non-negative cycles



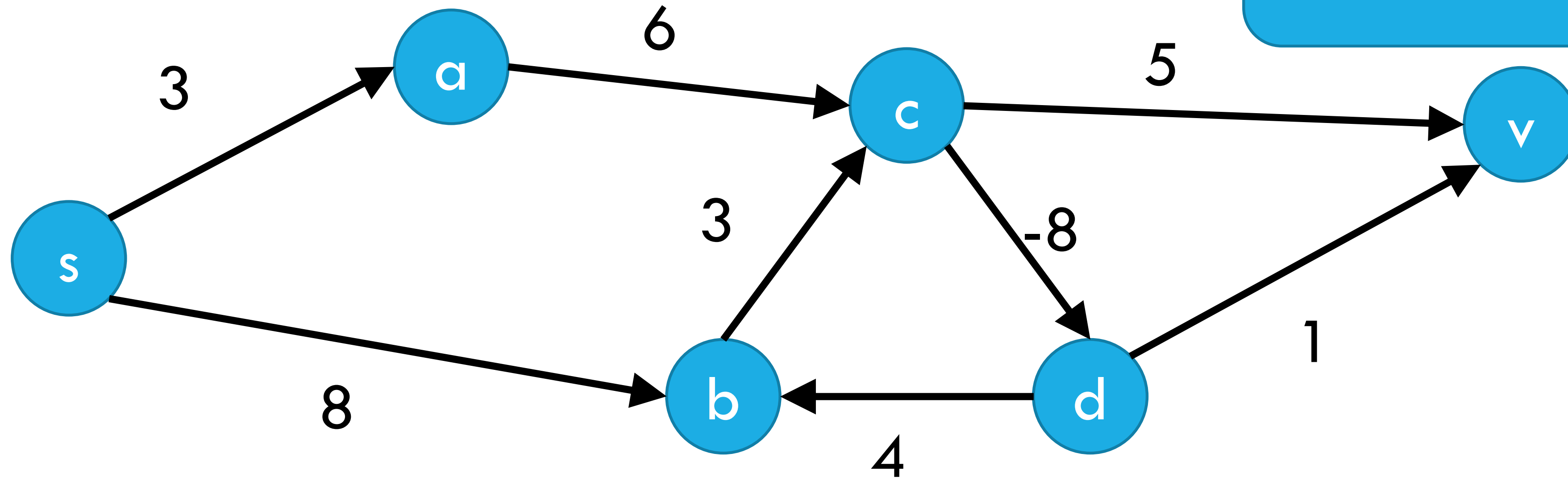
Dijkstra's might fail

But the shortest path IS defined.

There is an answer

Negative Cycle

Pollev.com/Robbie



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

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	X
Dijkstra's	$O(m+n\log n)$		X
Bellman-Ford	$O(mn)$		Yes!

Why Bellman-Ford?

Bellman-Ford; a DP-inspired algorithm for shortest-paths.

Ended up with pseudocode very similar to Dijkstra's…

…but slower.

Why? Handle negative edge weights.

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:

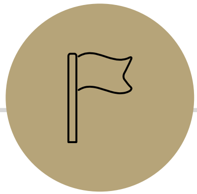
Run one extra iteration of the main loop– if any value changes, you have a negative length cycle. Some of the values you calculated are wrong.

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	X
Dijkstra's	$O(m+n\log n)$		X
Bellman-Ford	$O(mn)$		Yes!



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 LPs

What is a linear program?

A simple example LP

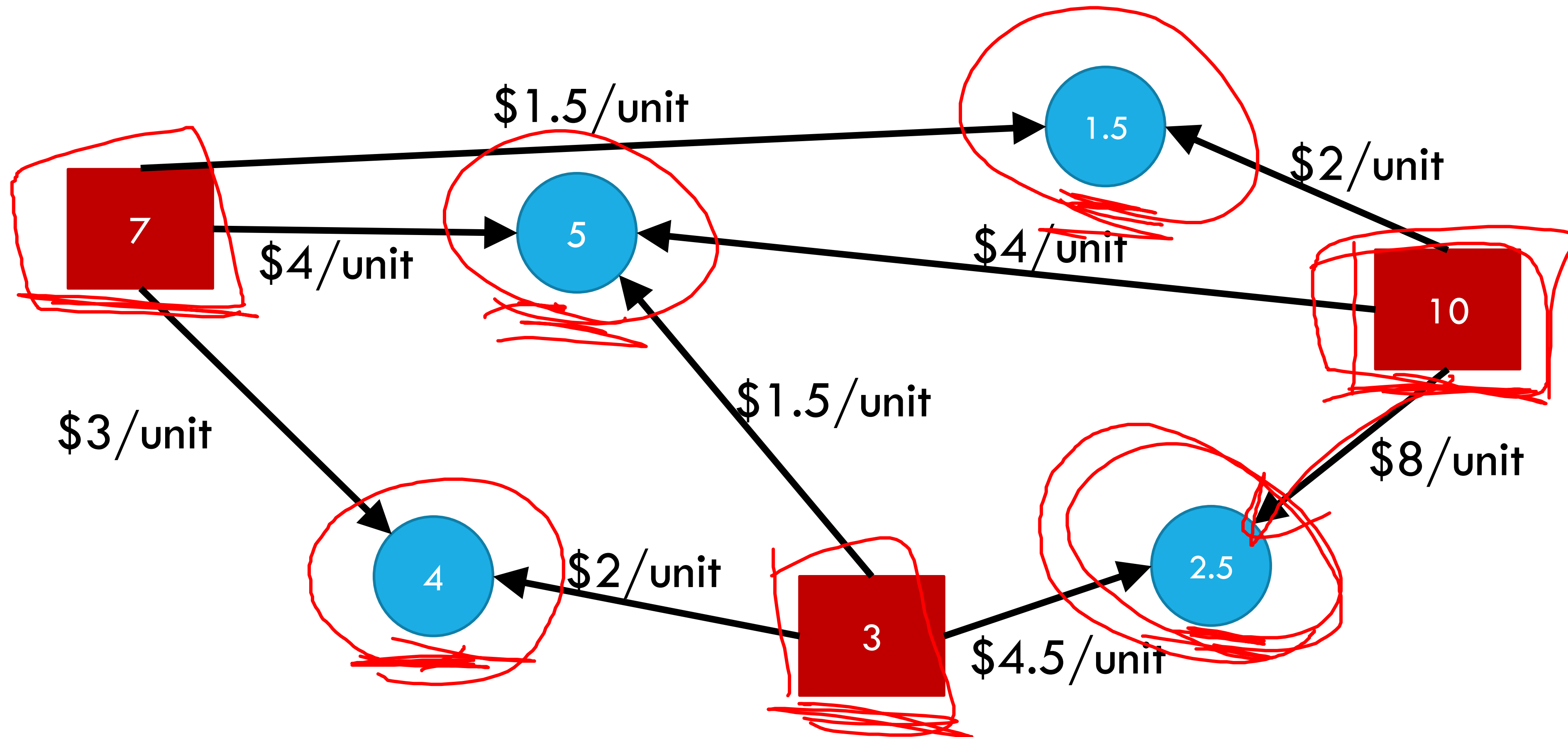
Computational Issues

An application – Vertex Cover on trees (again)

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

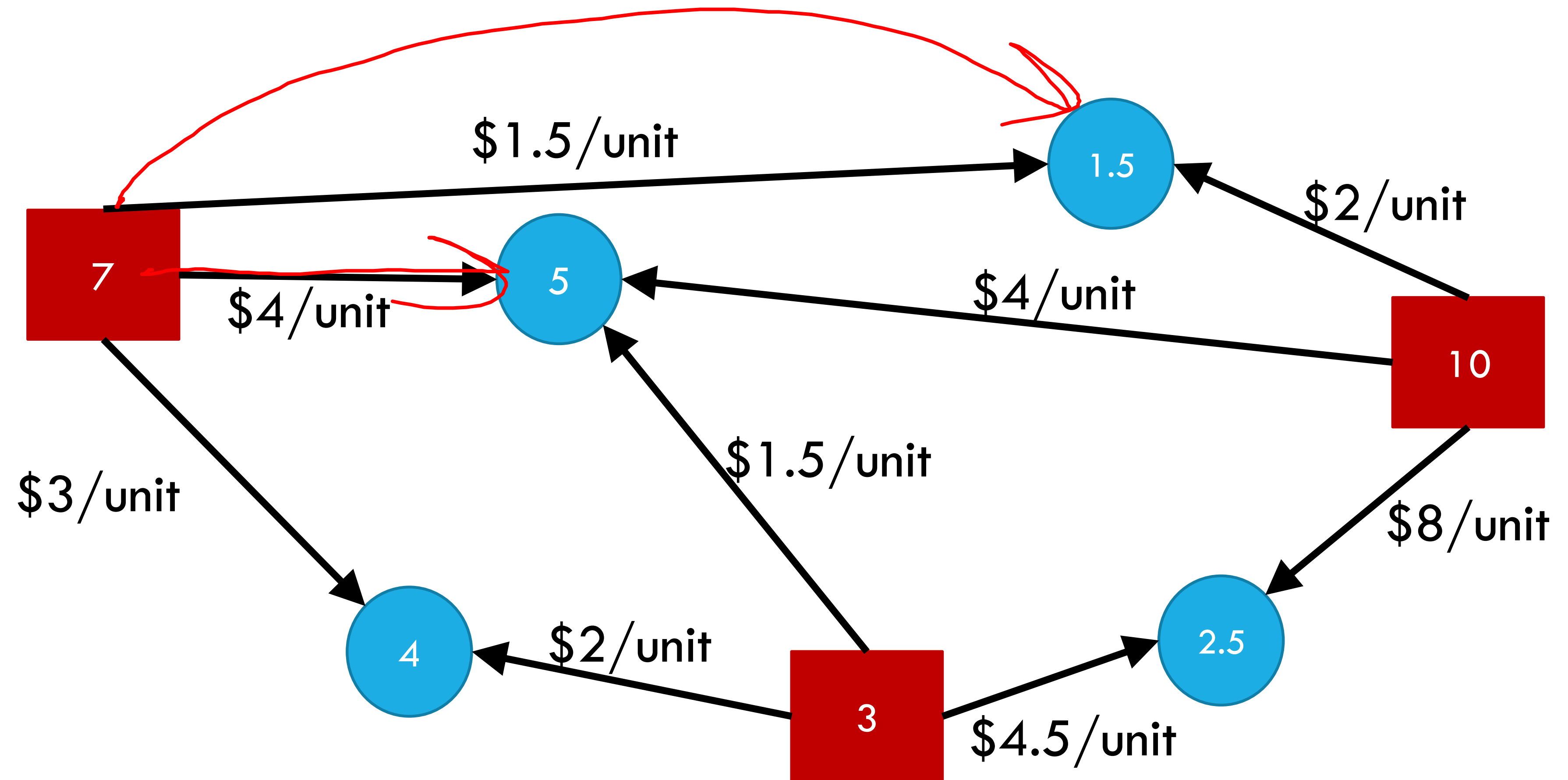
Example Problem

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?

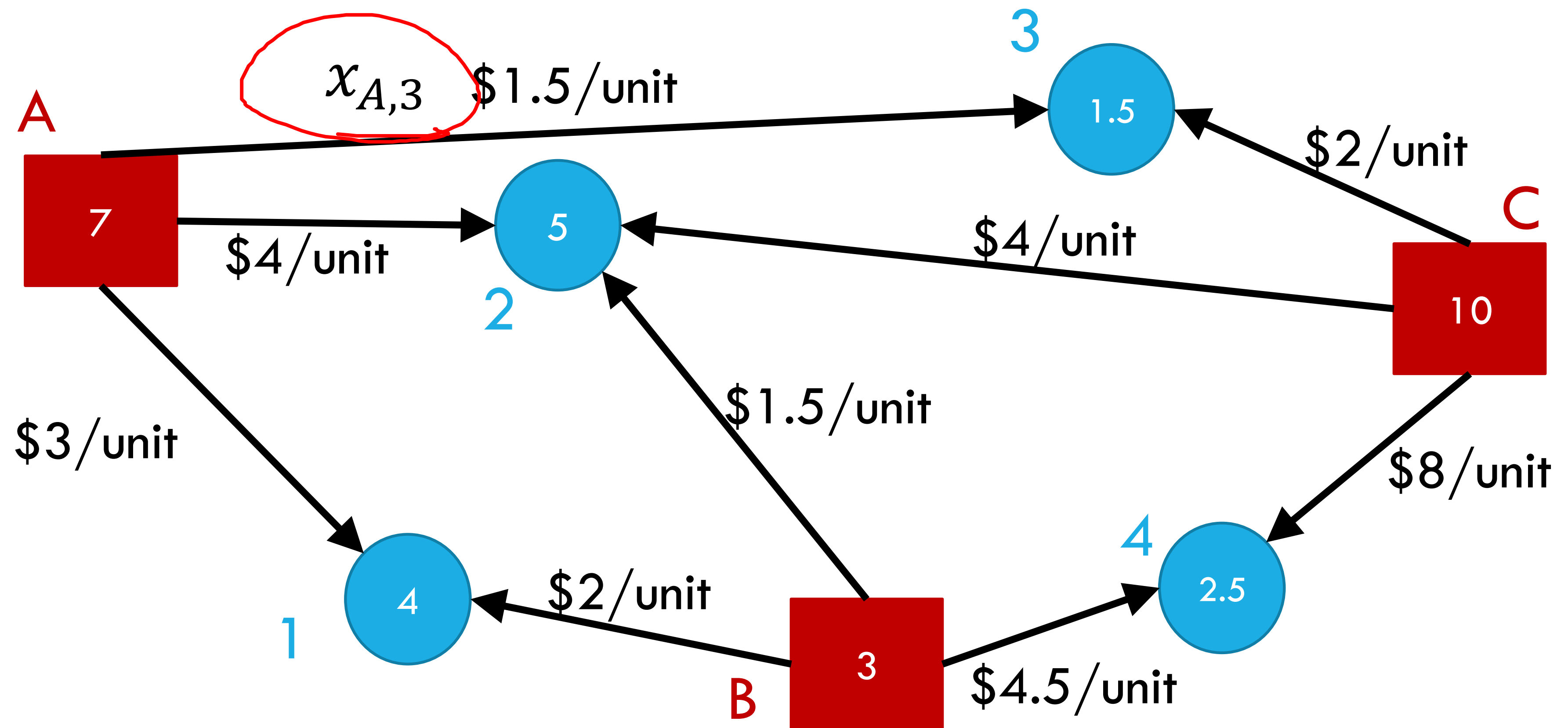


Example Problem

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.

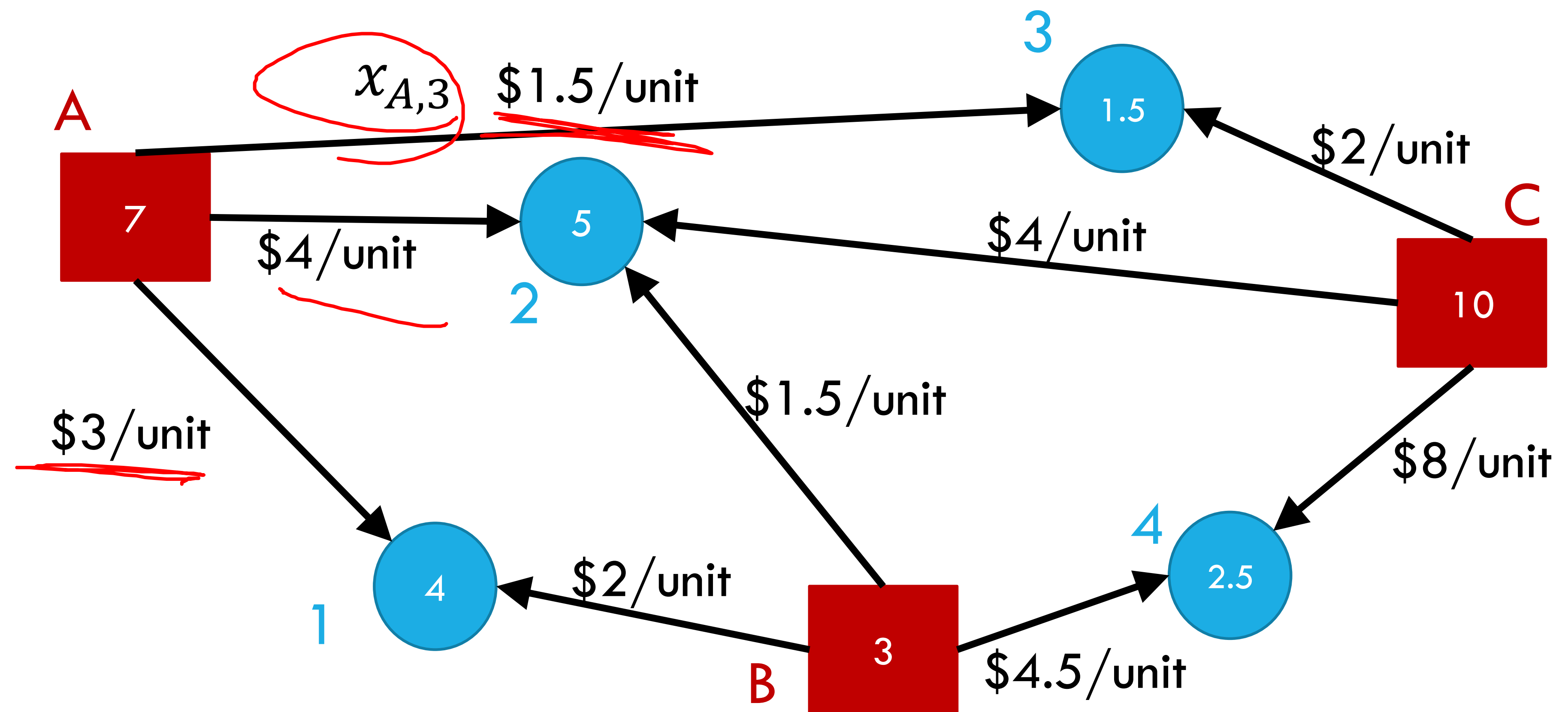


Example Problem

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

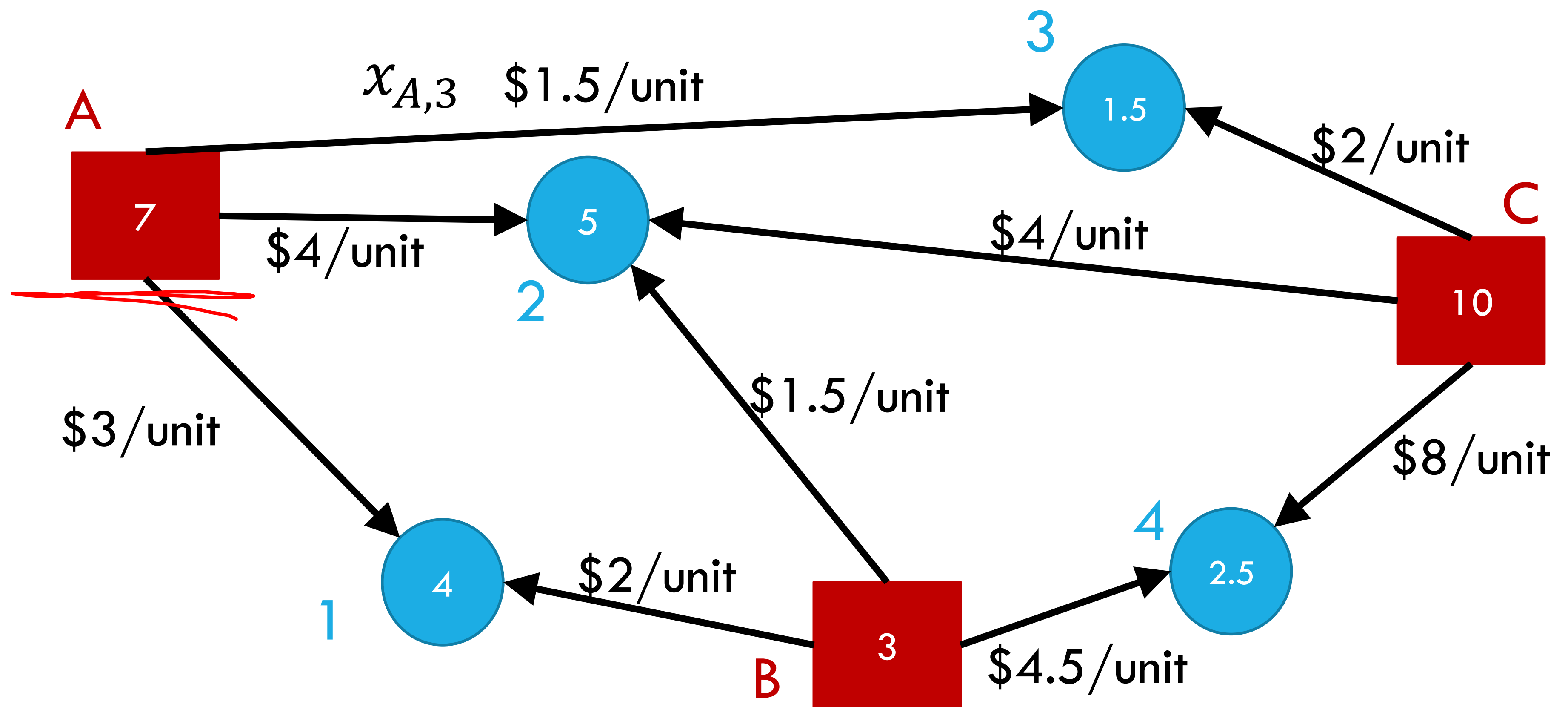
$$(x_{A,1} \cdot 3 + x_{A,2} \cdot 4 + x_{A,3} \cdot 1.5) +$$
$$(x_{B,1} \cdot 2 + x_{B,2} \cdot 1.5 + x_{B,4} \cdot 4.5) +$$
$$(x_{C,2} \cdot 4 + x_{C,3} \cdot 2 + x_{C,4} \cdot 8)$$

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:

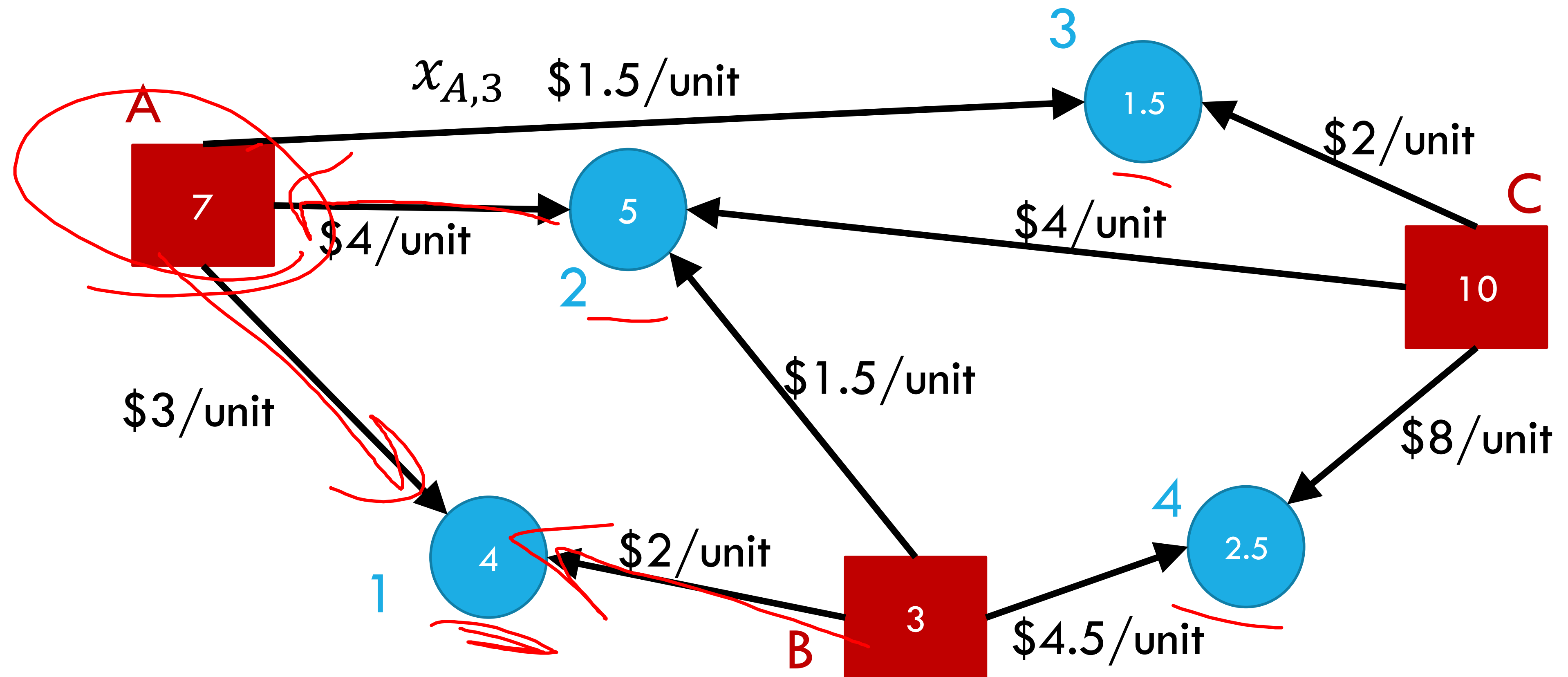
$x_{A,1} + x_{A,2} + x_{A,3} \leq 7$

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

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

No anti-soil:

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



Full Definition

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

Subject To:

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

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

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

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

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

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

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

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

A Linear Program

A linear program is defined by:

Real-valued **variables**

Subject to satisfying **everything** in a list of **linear constraints**

A linear constraint is a statement of the form: $\sum a_i x_i \leq c_i$
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.

$\sum$ const · var

Linear constraints

Pollev.com/robbie

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

Pollev.com/robbie

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.

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:

$x_{A,1} + x_{B,1} \geq 4$
$x_{A,2} + x_{B,2} + x_{C,2} \geq 5$
$x_{A,3} + x_{C,3} \geq 1.5$
$x_{B,4} + x_{C,4} \geq 2.5$

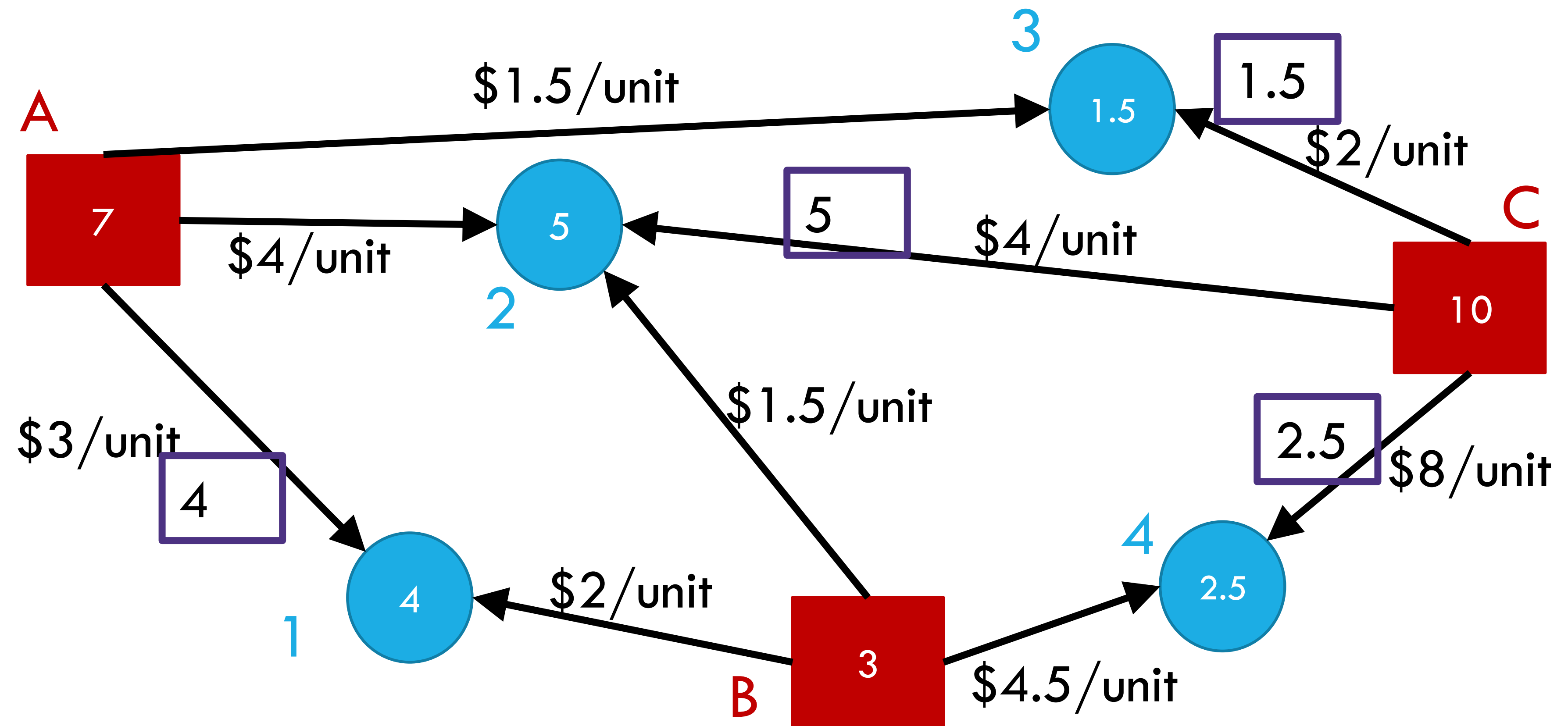
Can't overuse a pile:

$x_{A,1} + x_{A,2} + x_{A,3} \leq 7$
$x_{B,1} + x_{B,2} + x_{B,4} \leq 3$
$x_{C,2} + x_{C,3} + x_{C,4} \leq 10$

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:

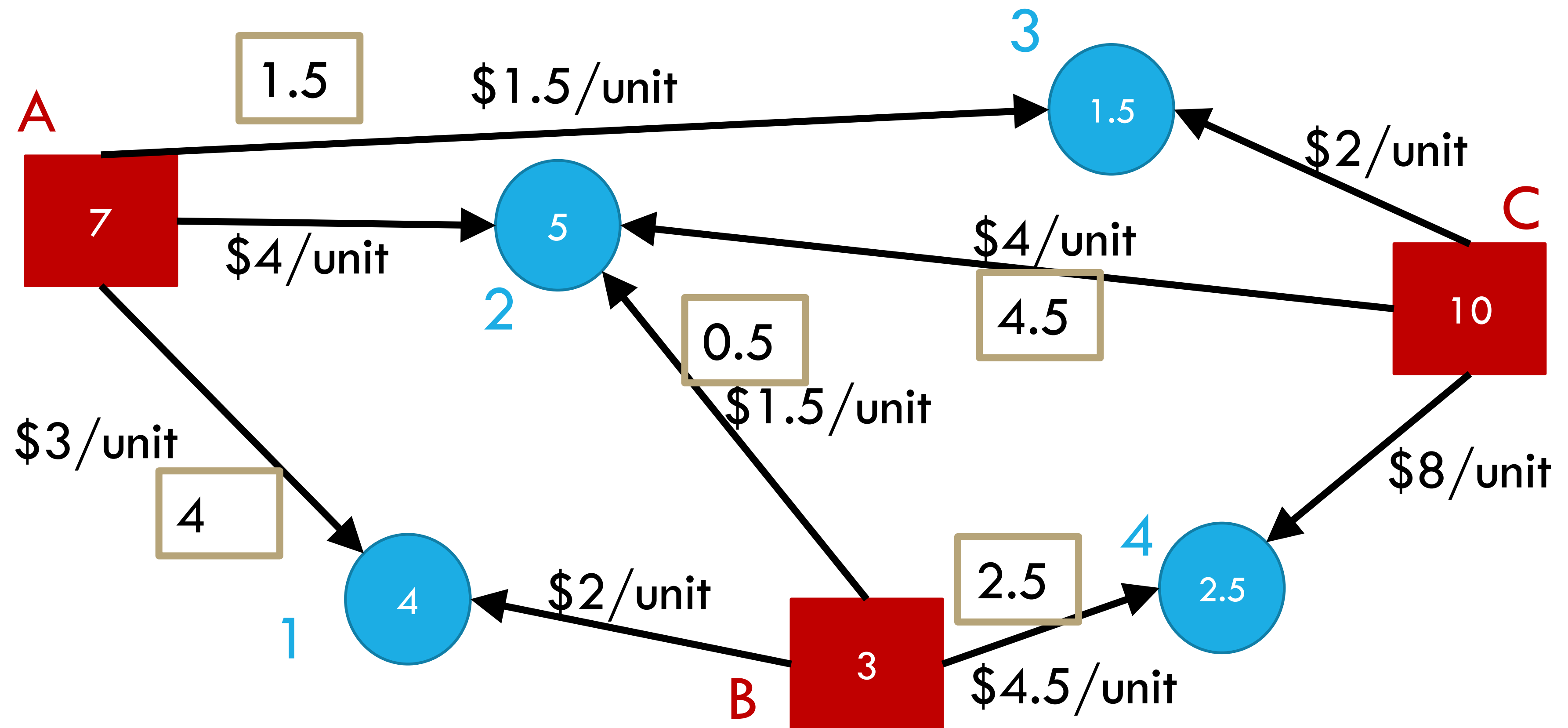
$x_{A,1} + x_{A,2} + x_{A,3} \leq 7$
$x_{B,1} + x_{B,2} + x_{B,4} \leq 3$
$x_{C,2} + x_{C,3} + x_{C,4} \leq 10$

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)