

Algorithms Refresher

CS 4364 & 5364

Some string terminology

Examples

An *alphabet* Σ is a set of characters

- typically we say $|\Sigma| = \sigma$

A *string* $S \in \Sigma^*$ is a finite set of characters from our alphabet

- note the set notation above: Σ^* -- set of all strings of any length
- Σ^3 -- set of 3 character strings

We can also write a string as an array $S[1...n]$ and address a single char.

We also sometimes write a string as an ordered list of characters

abstractly, $S = s_1 s_2 \dots s_n$

A *substring* can be written as $S_{i,j}$ or $S[i...j]$

$$\Sigma = \{'a', 'e', 'i', 'o', 'u'\}$$

$$\sigma = 5$$

$$S = \text{"aaoiie"}$$

$$\Sigma^* = \{ "", "a", "e", "i", "o", "u", "aa", "ae", \dots \}$$

$$\Sigma^3 = \{ "aaa", "aae", "aai", \dots \}$$

$$S[3] = 'o'$$

$$s_6 = 'e'$$

$$S_{2,4} = S[2...4] = \text{"aoi"}$$

Complexity Analysis

Given two lists of n and m elements respectively, how many comparisons are needed to compare all elements in one with each element of the other?

Given a list of n elements, how many comparisons are needed to compare each pair of elements? each triple?

I have an algorithm that for each string in a pool of m items performs an $O(\log n)$ search for elements in another list of items. It stops when it finds match. Whats the *worst case* running time? Whats the *best case* running time?

- What if I told you the search was $\Omega(1)$ in addition to being $O(\log n)$?

Amortized Analysis

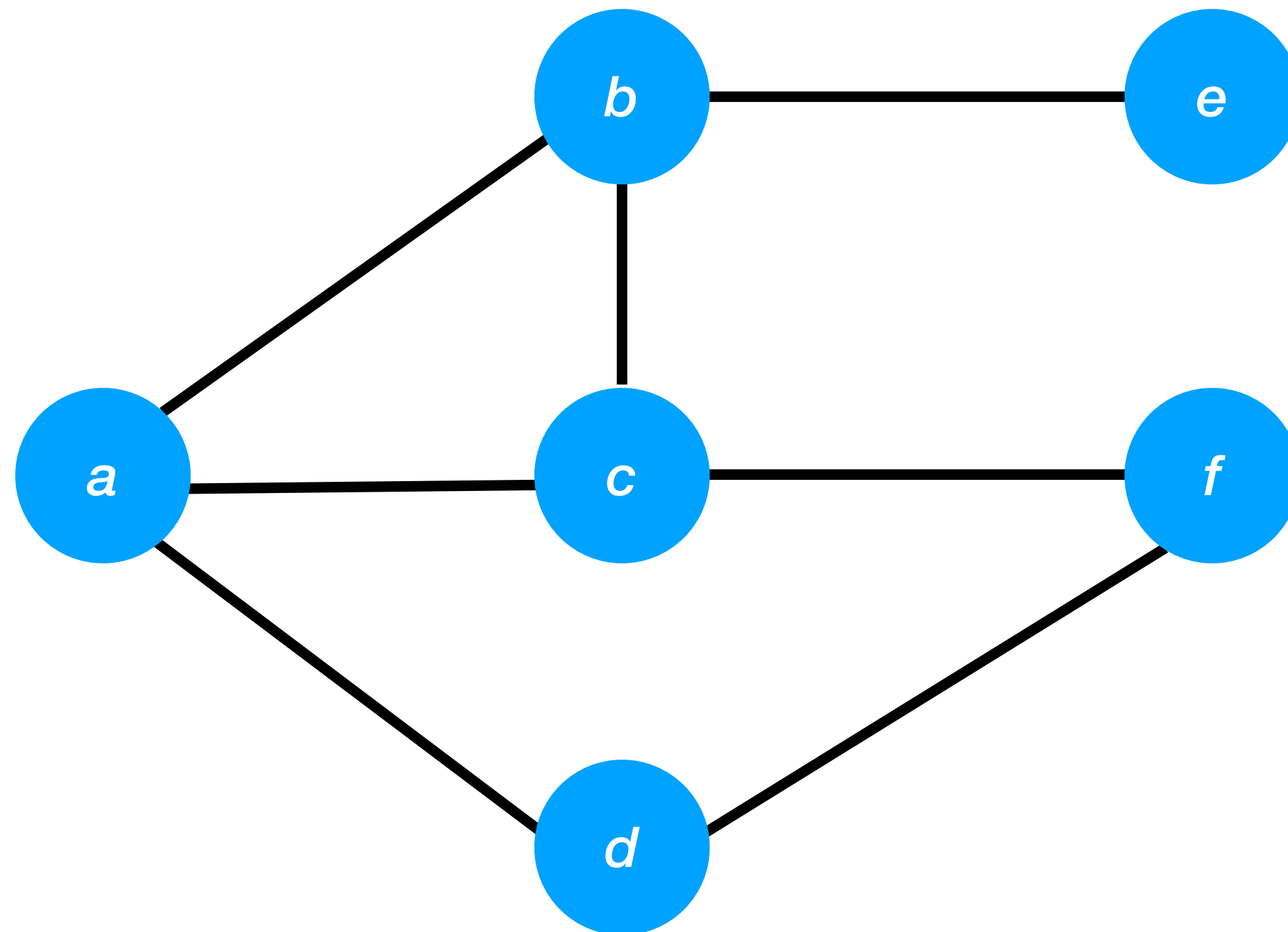
Sometimes we can't always calculate the expected running time just by analyzing the worst case running time of a single iteration

Think of tree re-balancing:

- a normal insert into a balanced tree takes a small amount of time
- if the tree is highly un-balanced it takes longer
- rebalancing can be performed to decrease running time later and the "cost" of this operation is averaged over many iterations (its distributed)

Graphs

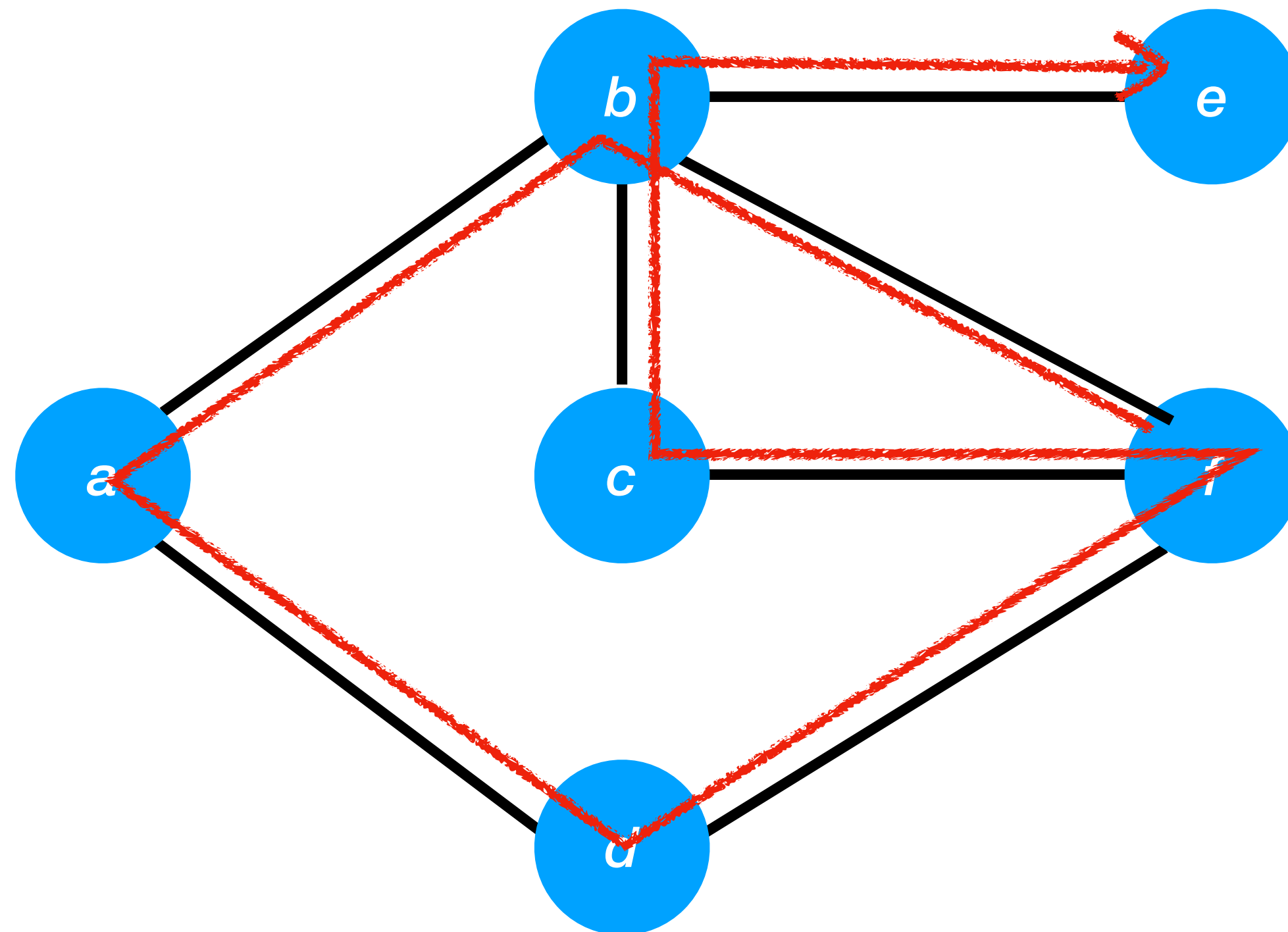
- $G = (V, E)$
- V -- vertex set $\{a, b, c, d, e, f\}$, $|V| = n$
- E -- edge set $\{(a, b), (a, c), (a, d), (b, e), (c, f), (d, f), (c, b)\}$, $|E| = m$



Graphs

An *Eulerian Path* visits each edge exactly once

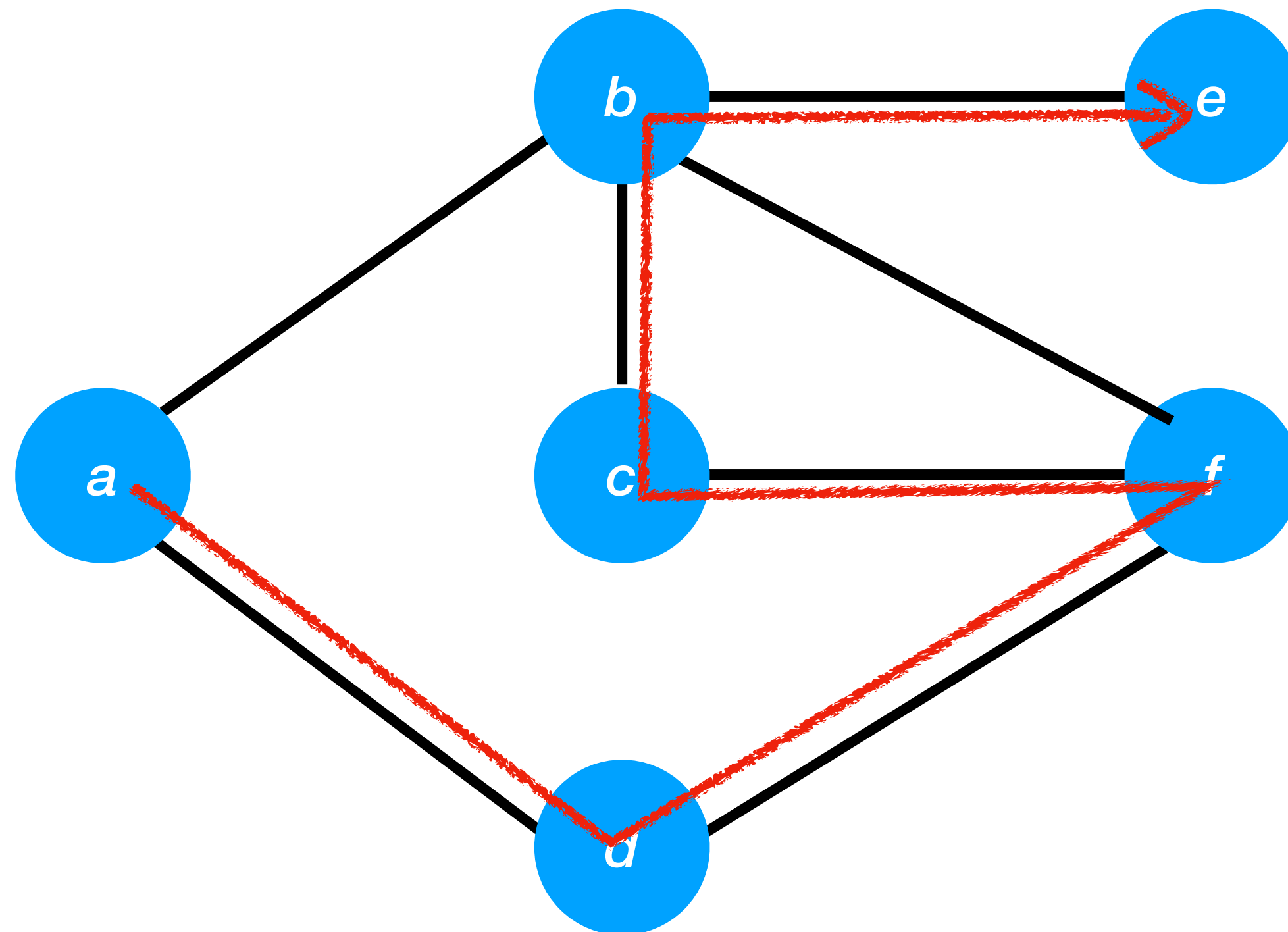
- an *Eulerian Cycle* starts and ends at the same vertex



Graphs

An *Hamiltonian Path* visits each *vertex* exactly once

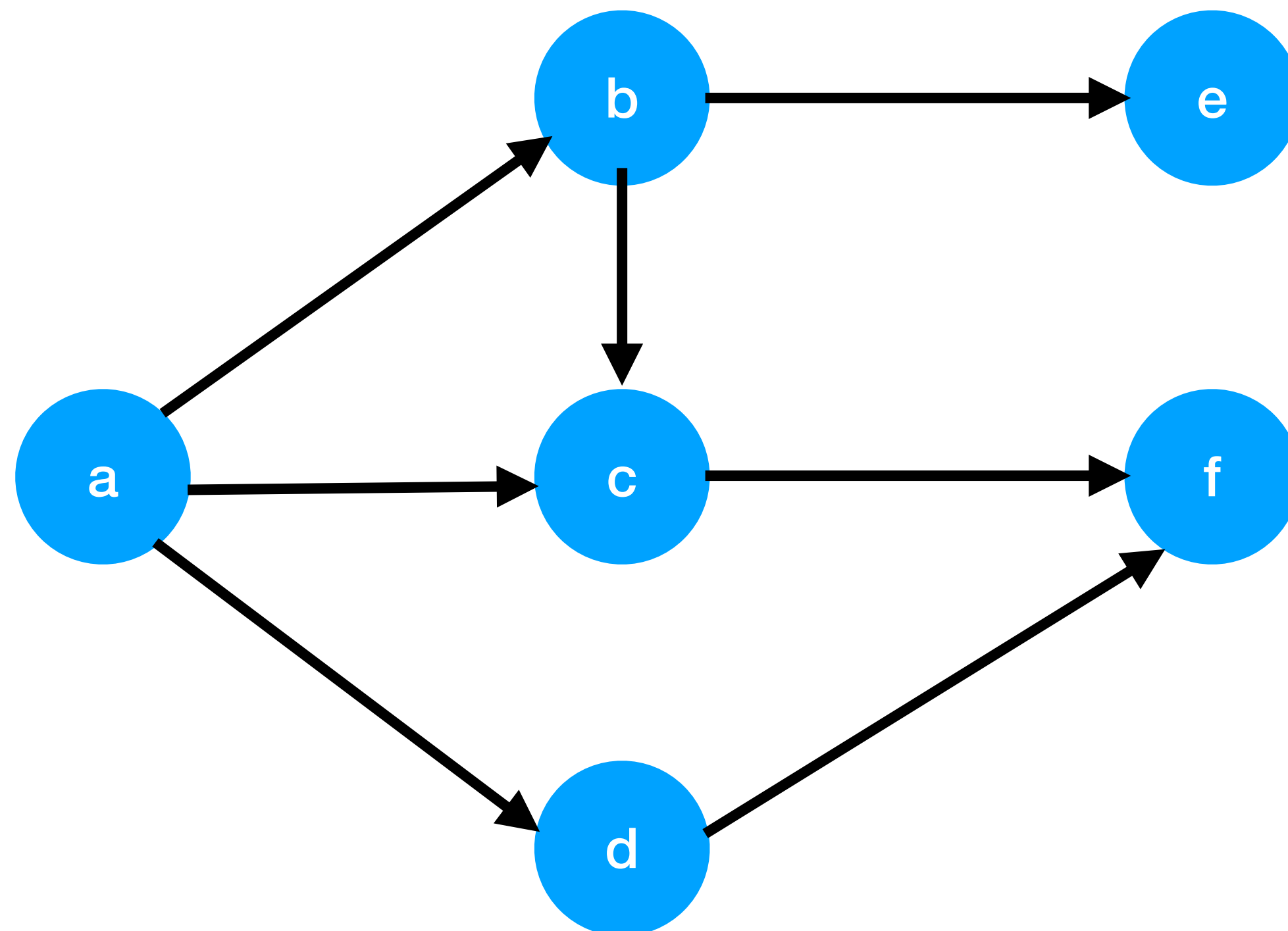
- an *Hamiltonian Cycle* starts and ends at the same vertex



Graphs

Directed Acyclic Graphs

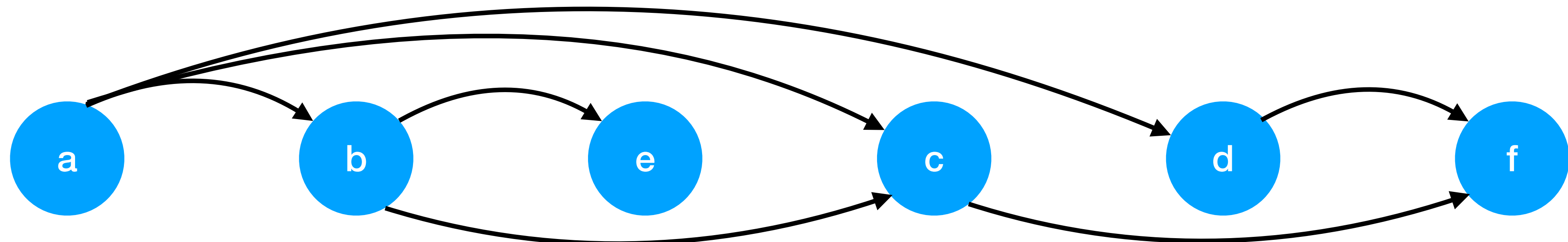
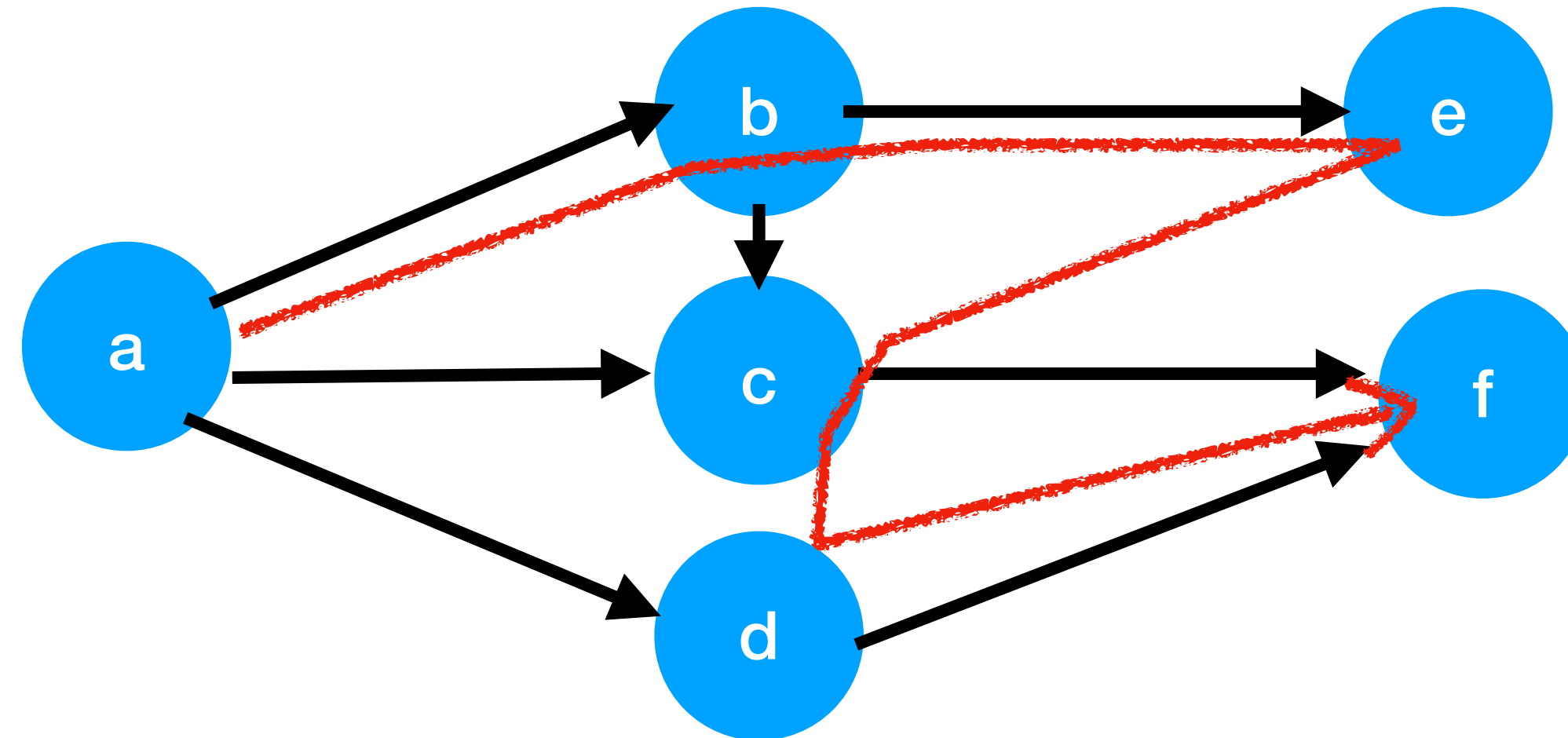
- most common graph type in computational biology problems



Graphs

Directed Acyclic Graphs

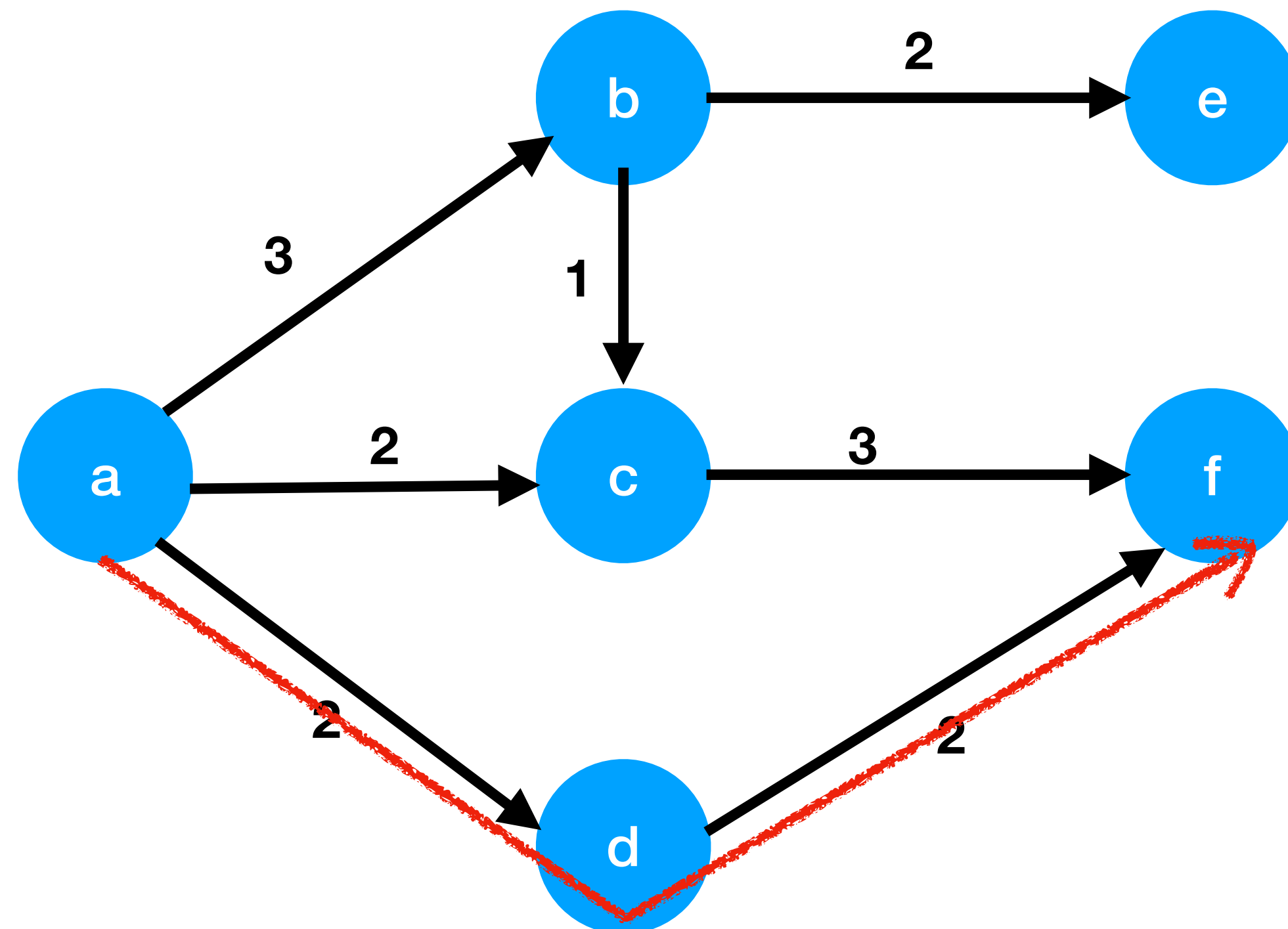
- nodes can be *ordered* such that a vertex occurs before any of its children



Graphs

Directed Acyclic Graphs

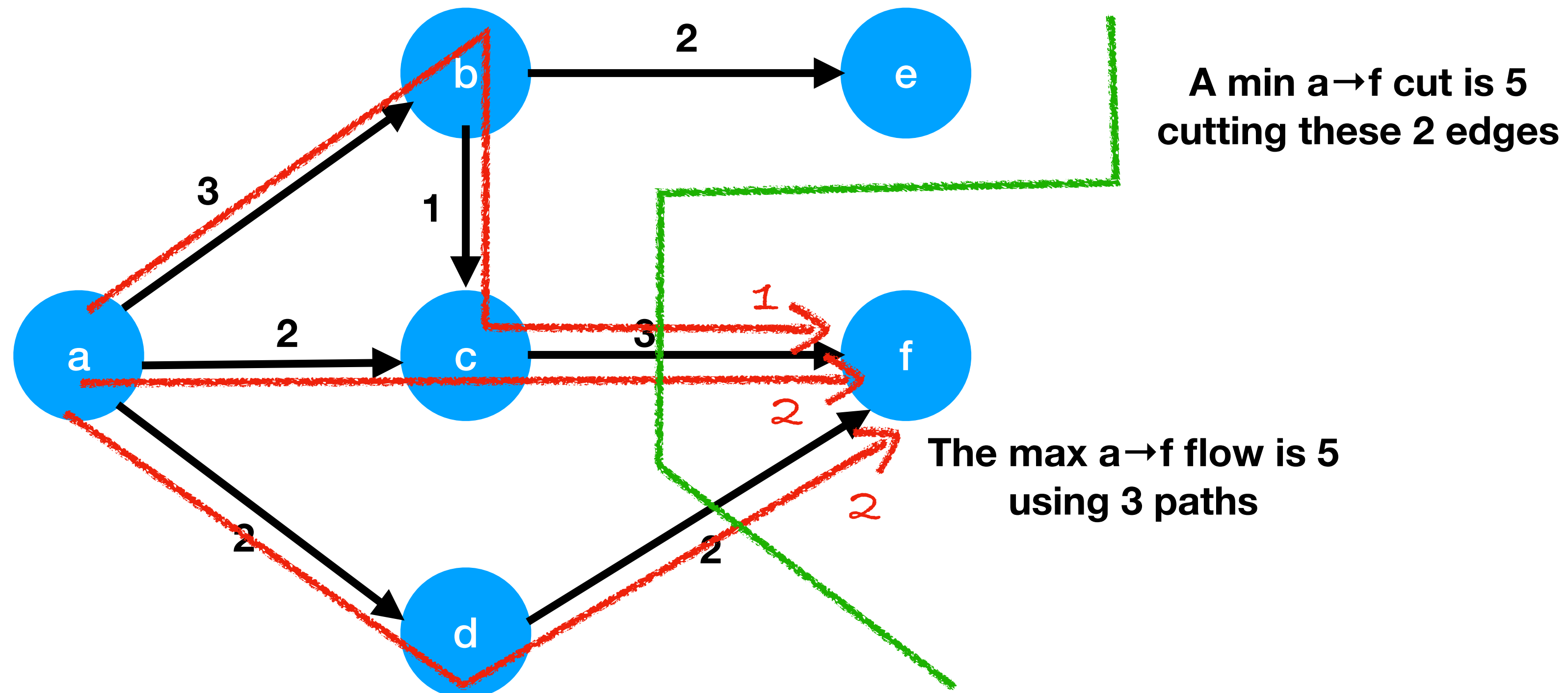
- when weighted shortest (min weight) path can be solved in $O(m+n)$ time



Graphs

Directed Acyclic Graphs

- sometimes used to model flow networks
- one common algorithm used on these networks is min-cut/max-flow



Dynamic Programming

Memoization is the simplest form of dynamic programming:

- assume you have some recursive method `RecFun(a)`
- memoization is basically the idea of adding the following to the top of the method definition:

```
def RecFun(a) {  
    if (defined(SavedResults[a])) return SavedResults[a];  
    ...  
}
```

- think about Fibonacci, if we want the 5th fibonacci number we need to compute the 4th and 3rd, but to compute the 4th we also need the 3rd so if we saved it we wouldn't have to compute it again.

Dynamic Programming

Flipping that on its head, really we need to compute all of the values we will ever need in `SavedResults[]`

Sometimes we can do that in an order that makes it easier.

- again think back to Fibonacci, we know the 1st and 2nd by definition
- we can then easily calculate the 3rd
- since we now have the second and 3rd we can easily ($O(1)$) calculate the 4th and so on

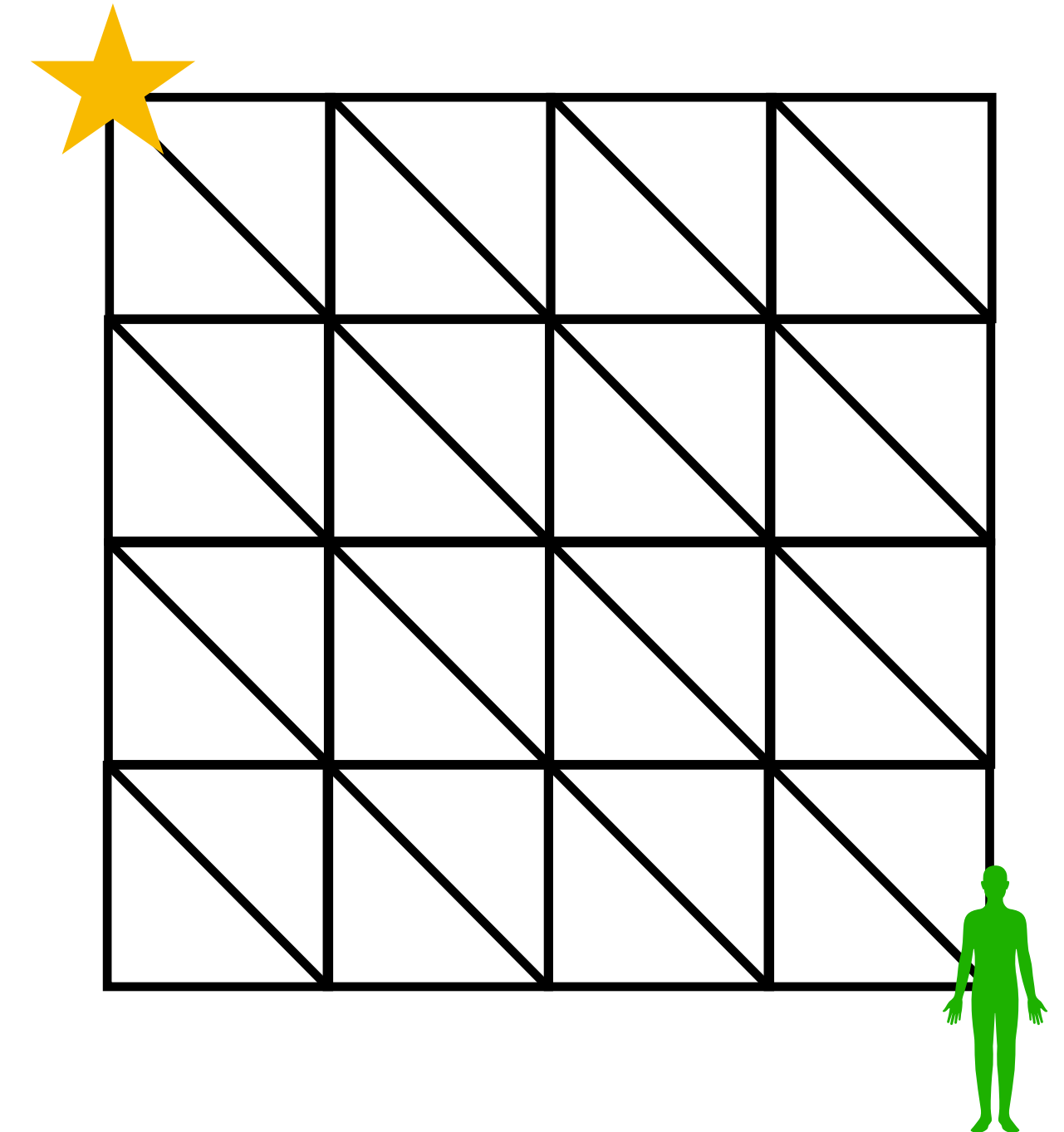
This doesn't follow the same *order* as the recursive calls but contains the same **recurrence relation**

- that that is $\text{Fibonacci}(x) = \text{Fibonacci}(x-1) + \text{Fibonacci}(x-2)$ so if we fill in a table in *increasing* value we know the values needed will be there

Dynamic Programming

Imagine you are in a city that is on a grid, but also has all of the diagonal roads going North-West to South-East

- You want to get from the far SE corner to the NW corner
- The problem is the city is very hilly and you've been walking around like a tourist all day
- Thankfully the map you have has the incline of each road segment
- How do you plan your route? (assuming you don't want to backtrack by going south or east at all)



Bit Operations

Rank & Select are operations on binary numbers which are defined as:

$$\text{rank}_c(B, i) = \left| \{j \mid 0 \leq j \leq i, B[j] = c\} \right|$$

$$\text{select}_c(B, j) = \min \{i \mid \text{rank}_c(B, i) = j\}$$

for some binary string $B[0...(n-1)]$ and $c \in \{0, 1\}$

i	0	1	2	3	4	5	6	7	8	9
B	0	1	1	0	1	0	0	1	0	1
rank ₁	0	1	2	2	3	3	3	4	4	5
rank ₀	1	1	1	2	2	3	4	4	5	5

note that $\text{rank}_0(B, i) = i - \text{rank}_1(B, i) + 1$

j	select ₁	select ₀
1	1	0
2	2	3
3	4	5
4	7	6
5	8	9

Where do we apply this?

Graphs

- Genome Assembly
- Hashing
- Phylogeny (trees)

Network Flow

- Transcript Assembly/Quantification

Dynamic Programming

- Alignment
- RNA Folding