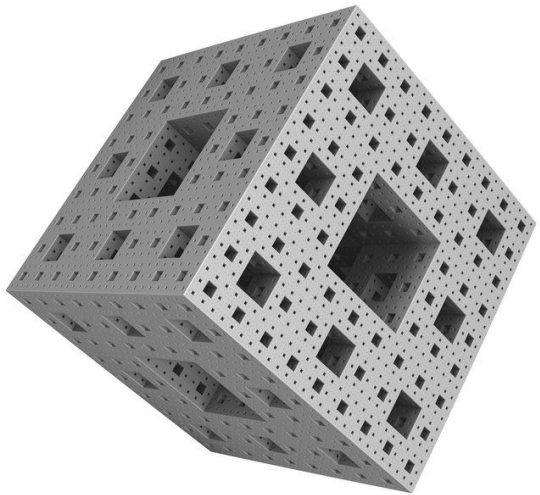
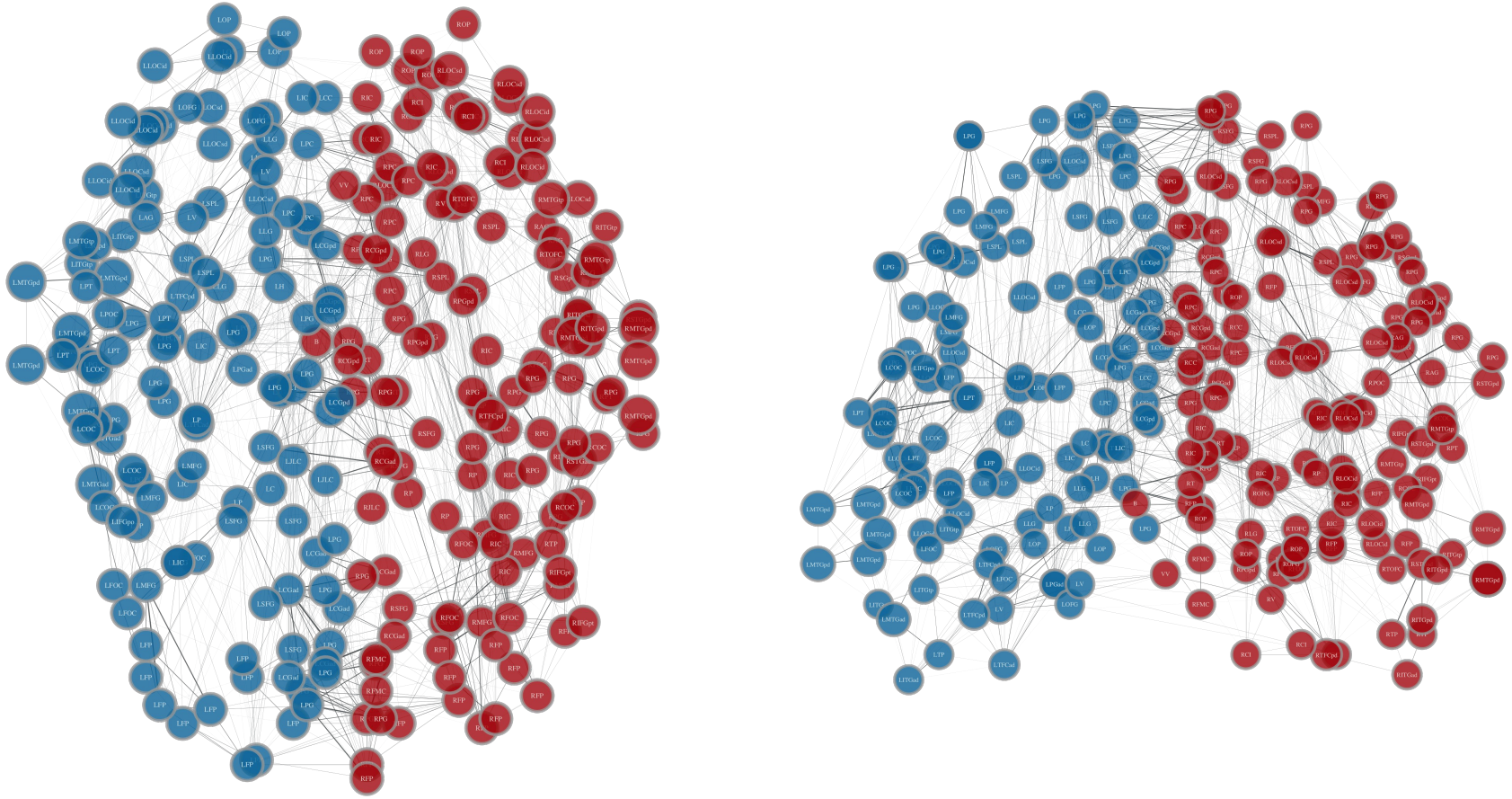
# YEAH - Recursion to the Rescue!

Anton Apostolatos



*Menger Sponge*



# A4: Recursion to the Rescue!

Doctors Without Orders


Disaster Preparations

DNA Detective

Winning the Presidency

# Recursive Backtracking

```
if (problem is sufficiently simple) {
    return whether or not the problem is solvable
} else {
    for (each choice) {
        try out that choice
        if (that choice leads to success) {
            return success;
        }
    }
    return failure;
}
```

# Outline before you write!

# Doctors Without Orders

## *Doctors*

Dr. M. Howard
**8 hours free**

Dr. L. Fine
**6 hours free**

Dr. C. Howard
**5 hours free**

## *Patients*

Needs 3 hours

Needs 2 hours

Needs 4 hours

Needs 5 hours

Needs 2 hours

Needs 3 hours

Dr. M. Howard

**8 hours free**

Needs 3 hours

Needs 2 hours

Needs 3 hours

Dr. L. Fine

**6 hours free**

Needs 4 hours

Needs 2 hours

Dr. C. Howard

**5 hours free**

Needs 5 hours

```
struct Doctor {
    string name;
    int hoursFree;
};
```

```
struct Patient {
    string name;
    int hoursNeeded;
};
```

*If so, populate this with schedule*

```
bool canAllPatientsBeSeen(Vector<Doctor> doctors,
                          Vector<Patient> patients,
                          Map<string, Set<string>>& schedule)
```

*Return whether it's possible for all patients to be attended*

# Tips and Tricks

- Think about what decisions you have at every step (what you're exploring) and what the base case could be

- Before writing any code, go through simple toy examples by hand to make sure your proposed solution's logic is sound

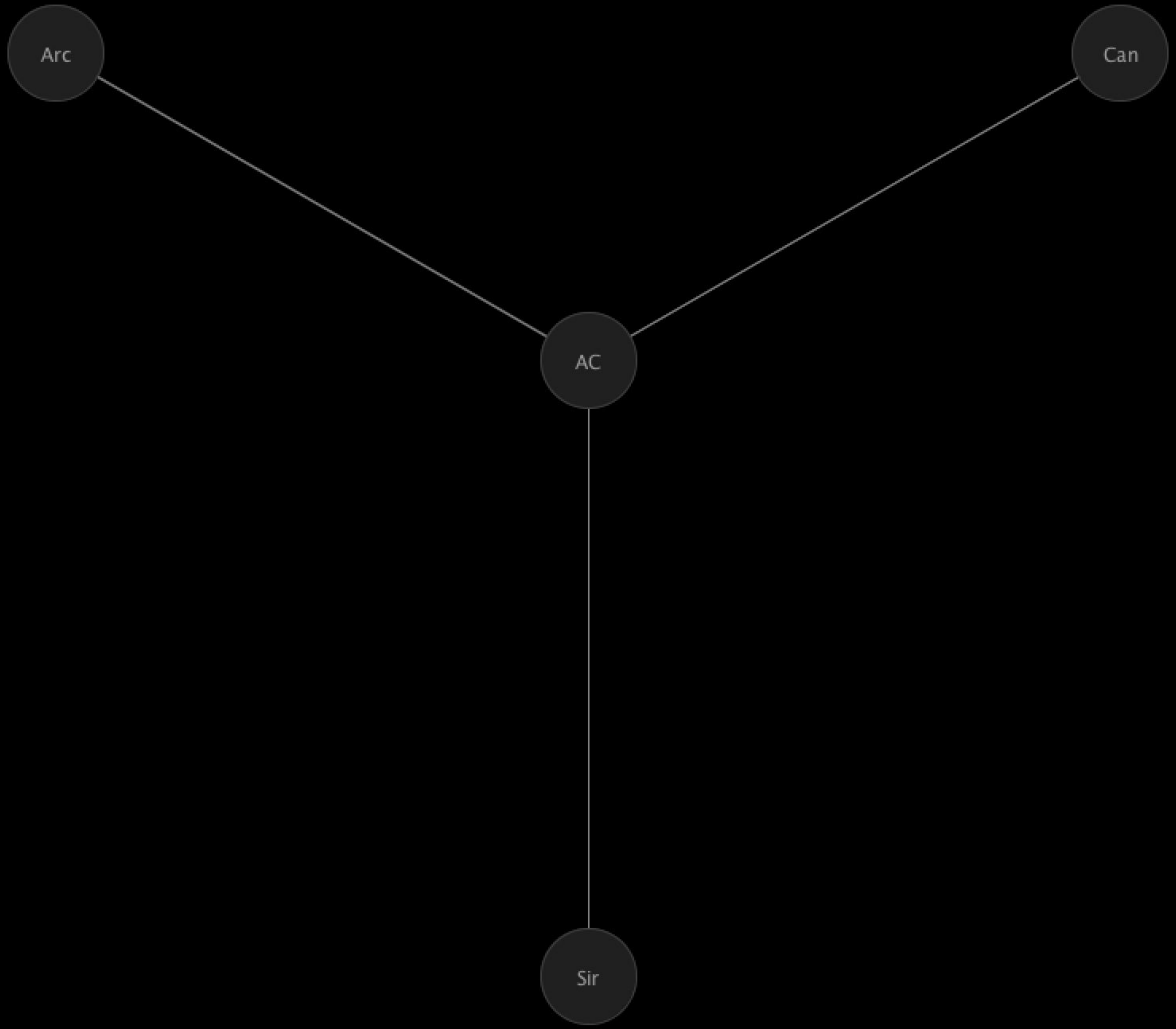
- If your function returns **`false`**, the final contents of **`schedule`** don't matter

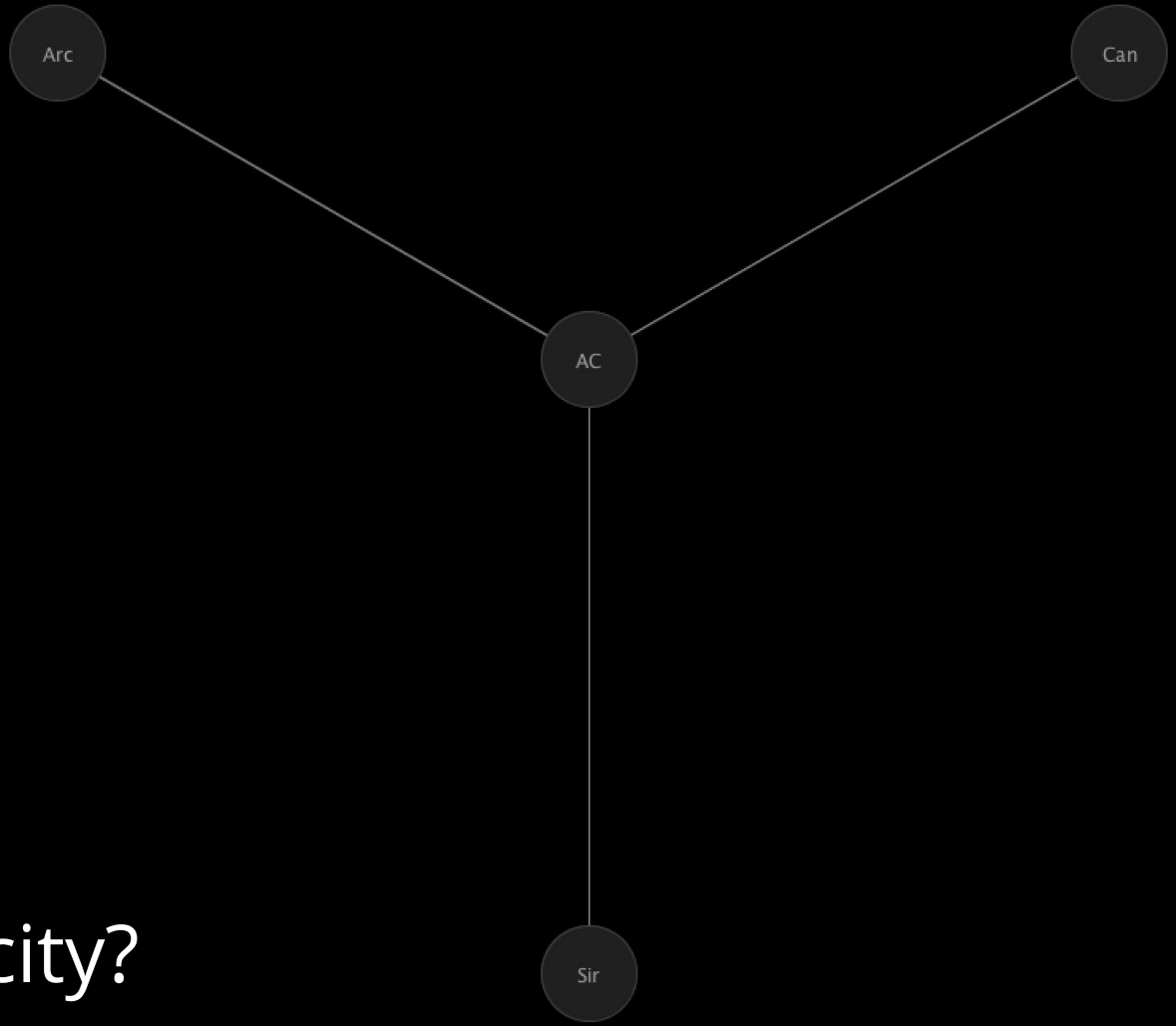
- You can assume no two doctors or patients have the same name

- Start by only worrying about getting the return value right; then work on populating **`schedule`**

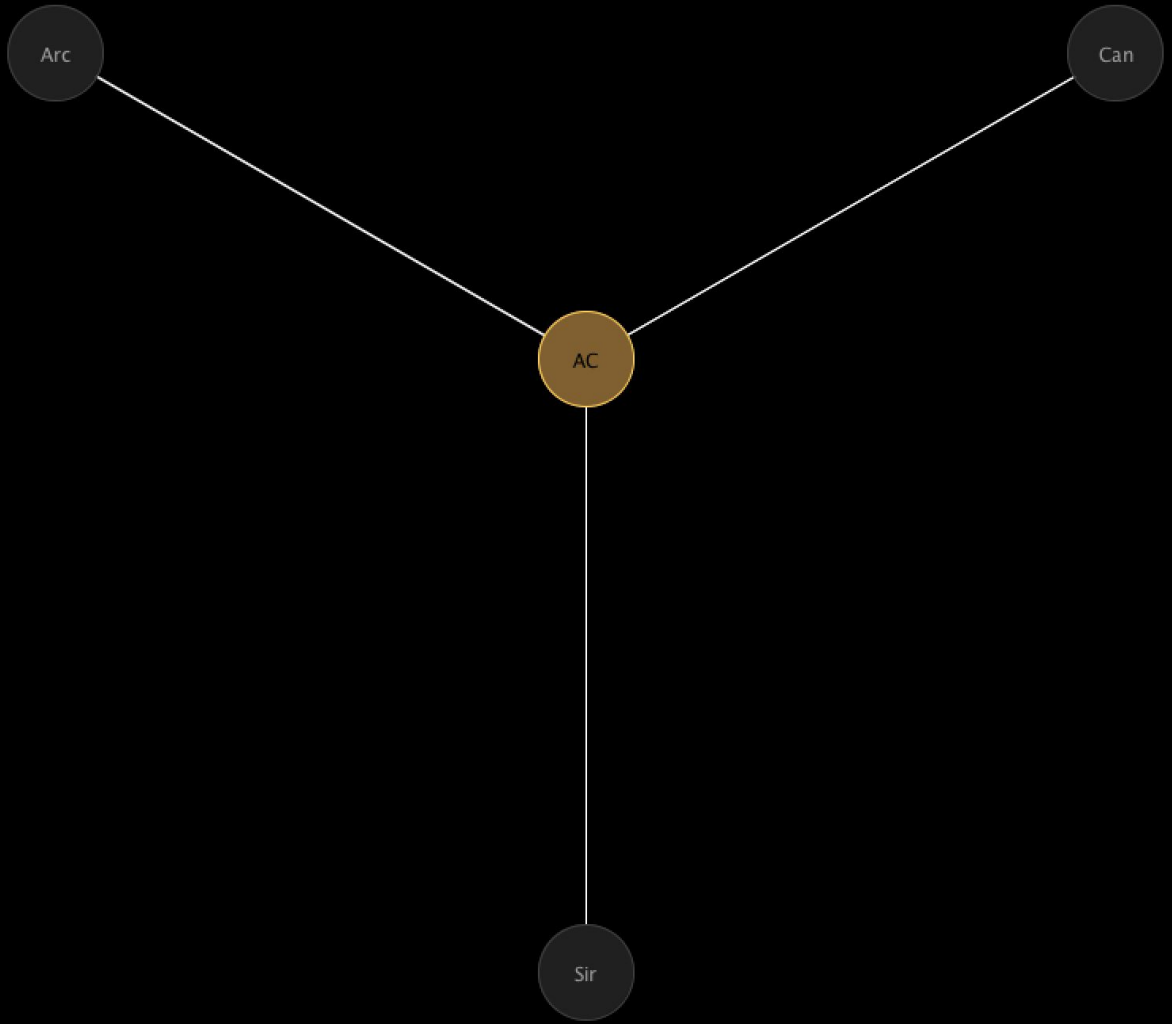
# Questions?

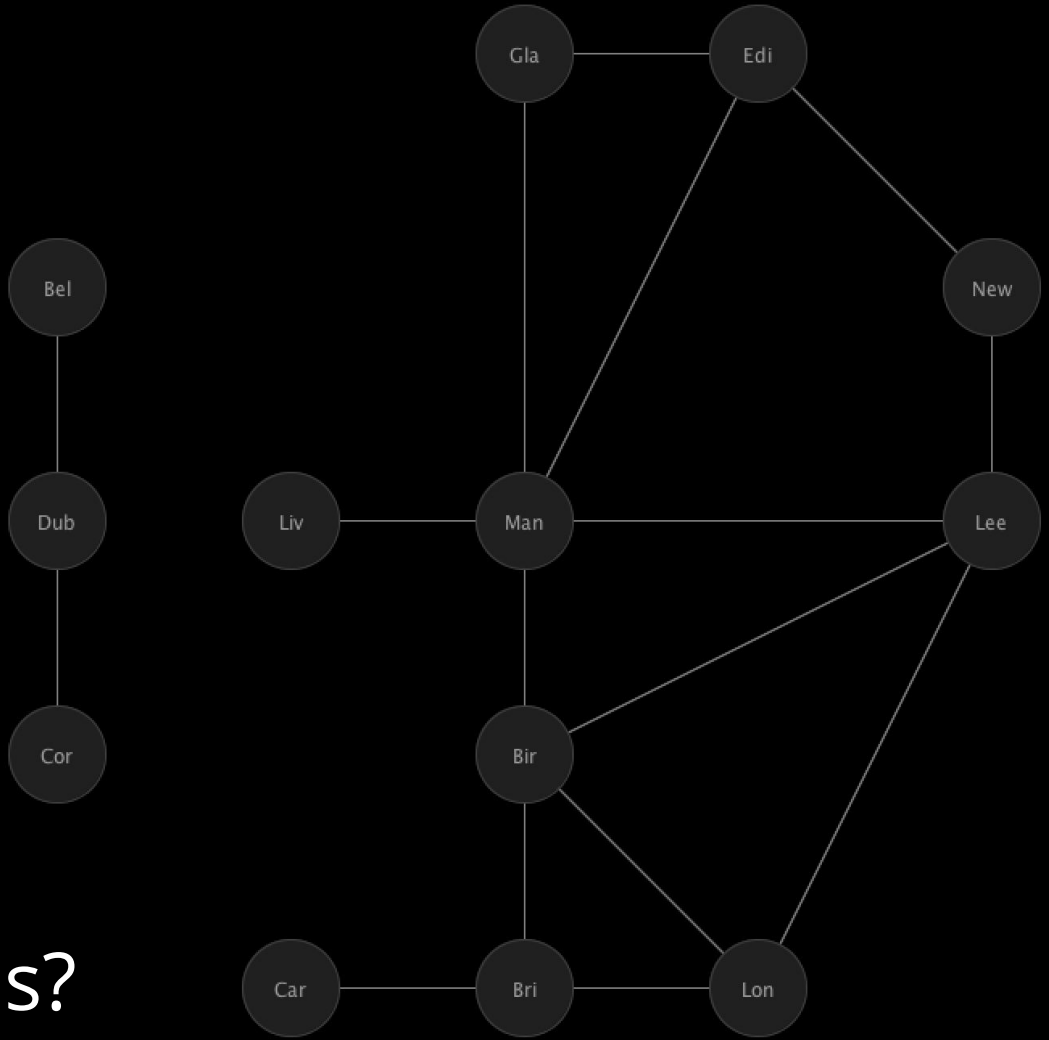
# Disaster Preparations


Arc
Can
AC
Sir

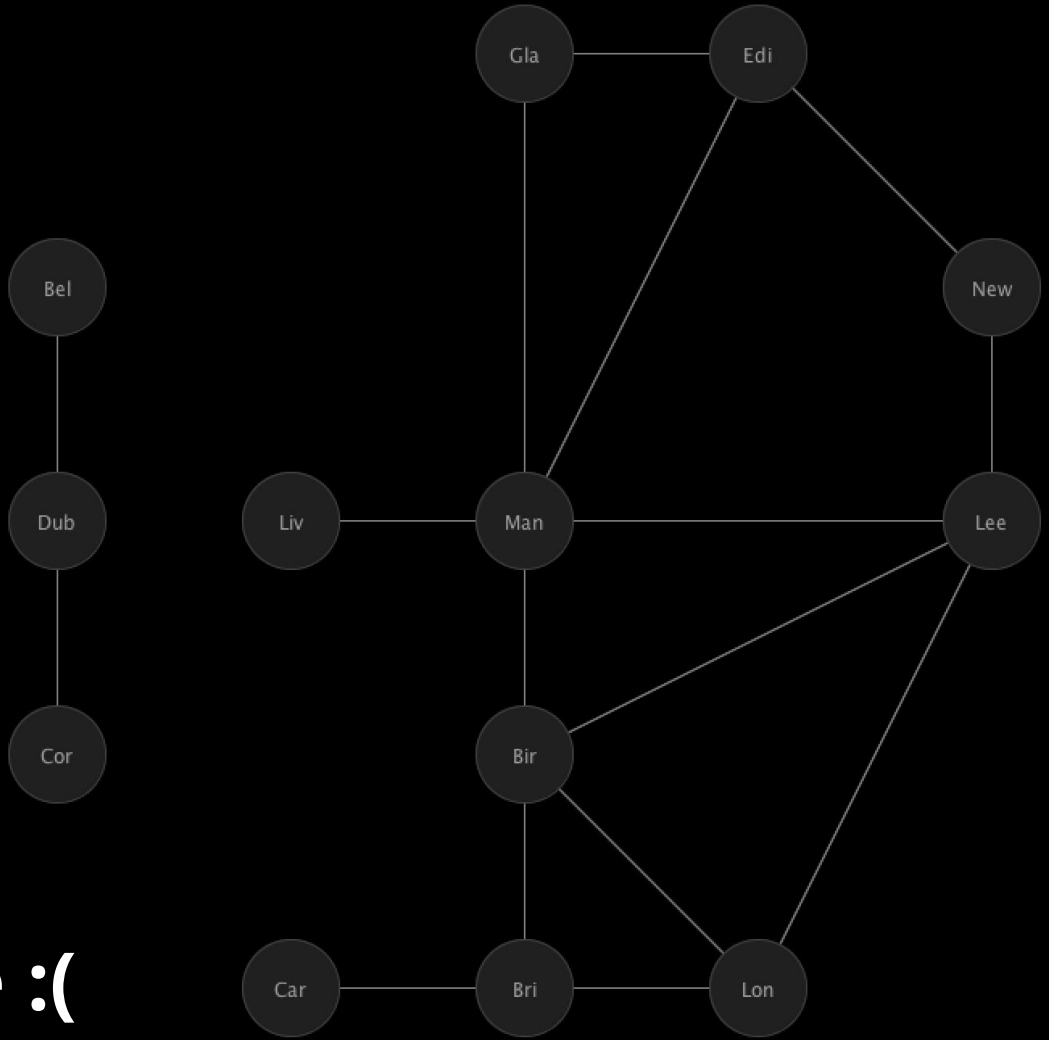



1 city?

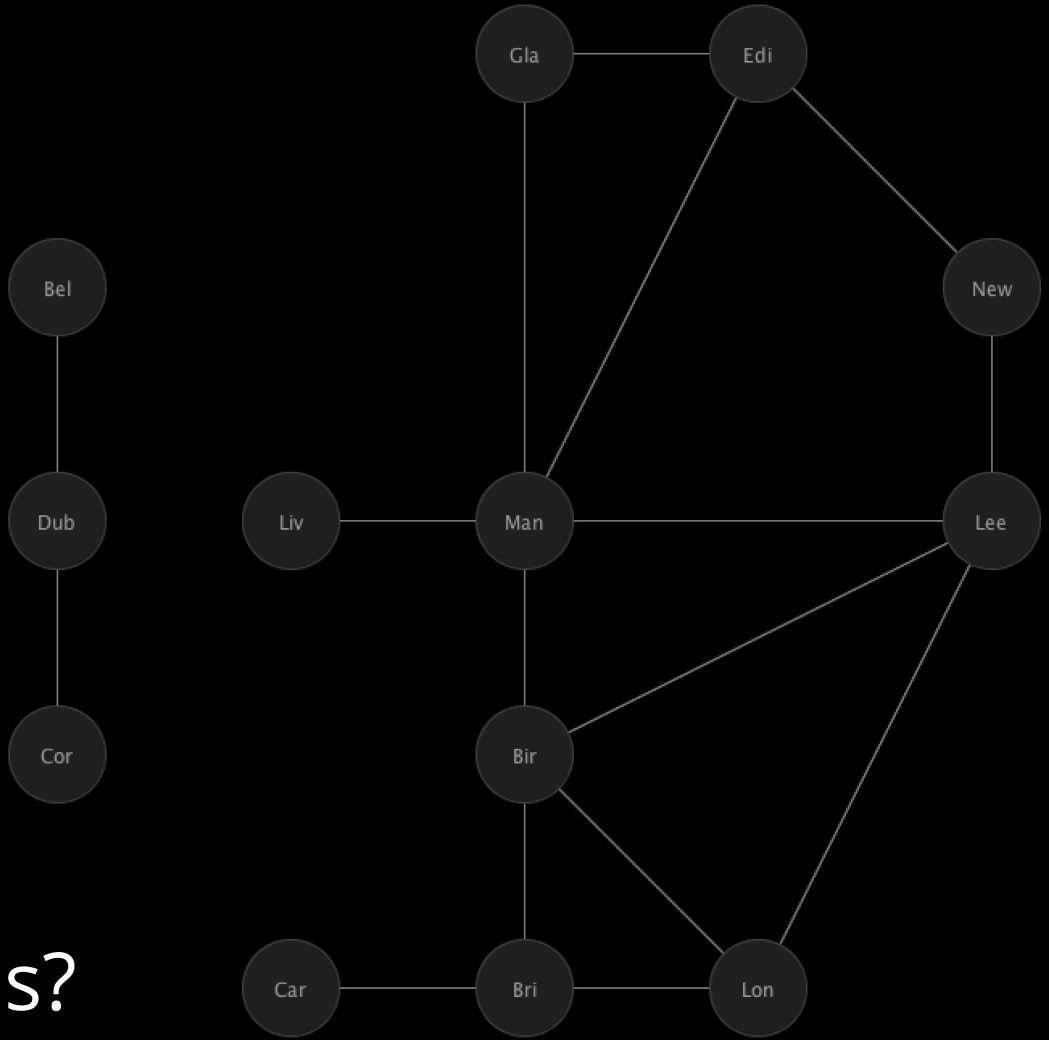

Arc
Can
AC
Sir



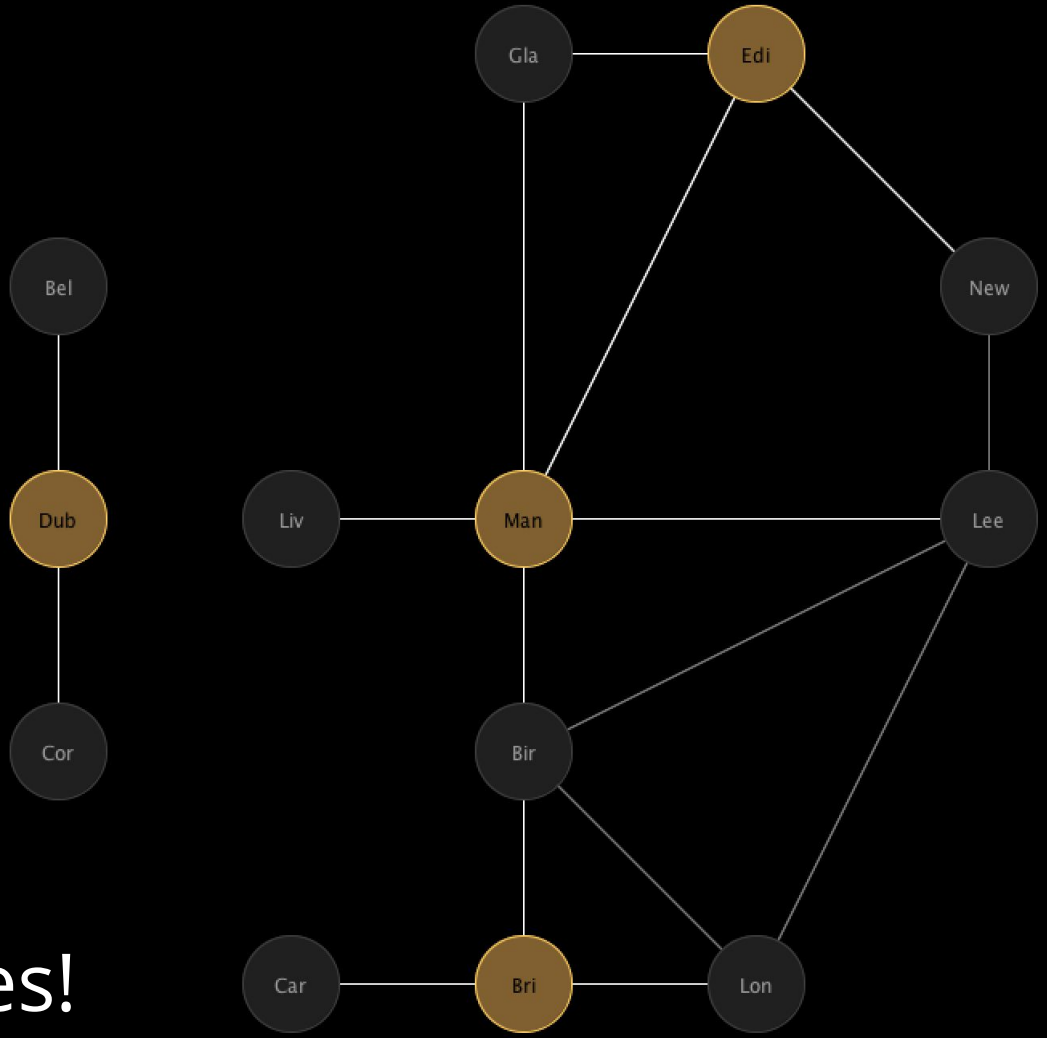

3 cities?




Nope :(




4 cities?




Yes!

```
"Sacramento":    {"San Francisco", "Portland", "Salt Lake City", "Los Angeles"}
"San Francisco": {"Sacramento"}
"Portland":      {"Seattle", "Sacramento", "Salt Lake City"}
```

```
bool canBeMadeDisasterReady(Map<string, Set<string>> roadNetwork,
                            int numCities,
                            Set<string>& locations)
```

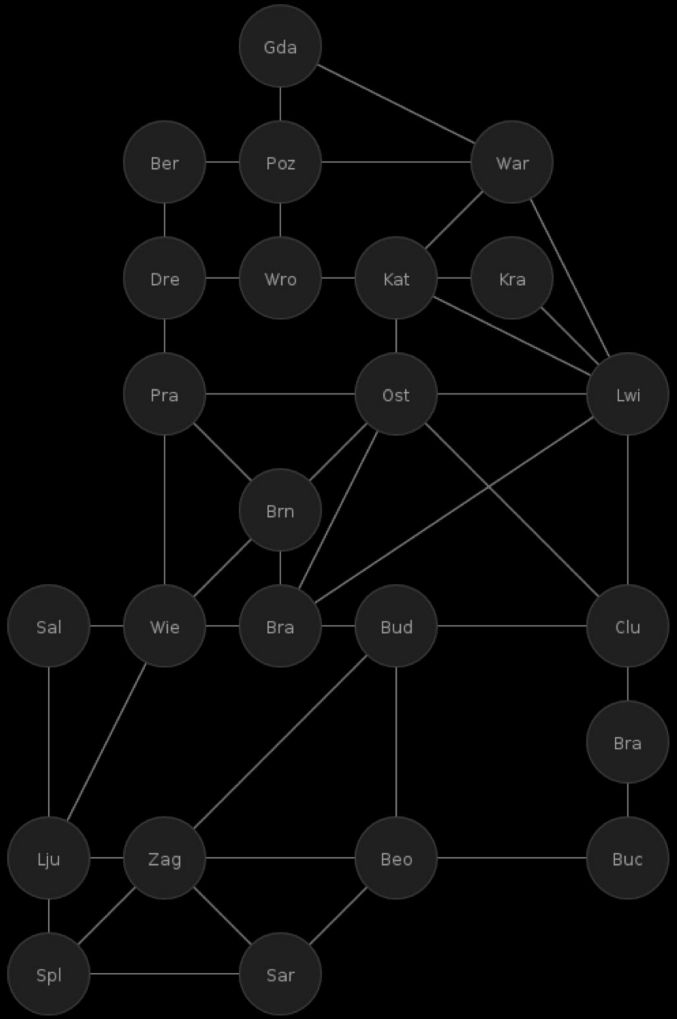
*Maximum number of cities to stockpile*

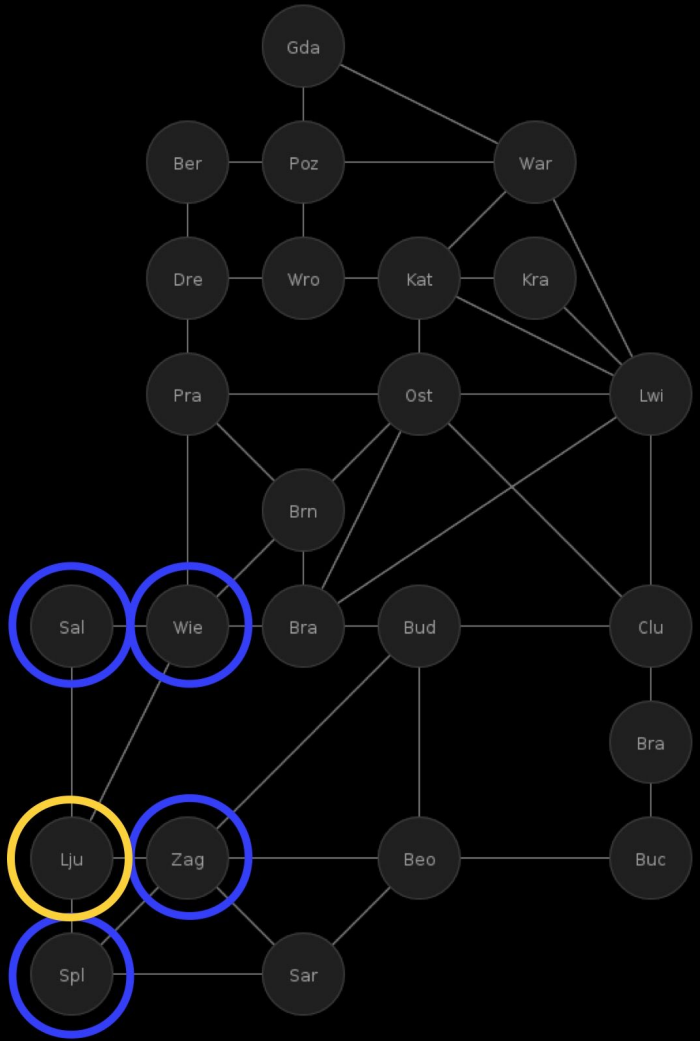
*If possible, fill with all locations we want to stockpile*

# There are different ways of thinking of the problem

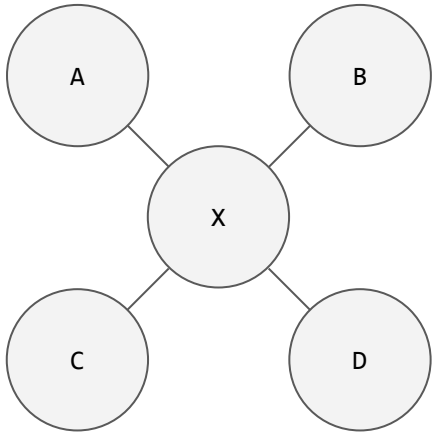
# **Option 1**: Enumerate all possibilities

$$\binom{\text{totalCities}}{\text{numCities}} = \binom{100}{16} \approx \text{grains of sand on earth}$$

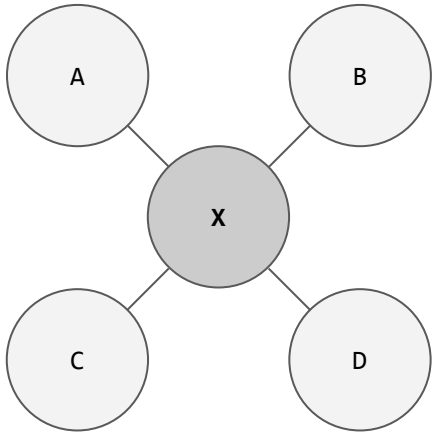

Gda
Ber
Poz
War
Dre
Wro
Kat
Kra
Pra
Ost
Lwi
Brn
Sal
Wie
Bra
Bud
Clu
Bra
Lju
Zag
Beo
Buc
Spl
Sar


Gda
Ber
Poz
War
Dre
Wro
Kat
Kra
Pra
Ost
Lwi
Brn
Sal
Wie
Bra
Bud
Clu
Bra
Lju
Zag
Beo
Buc
Spl
Sar

# **Option 2**: Choose by city cover

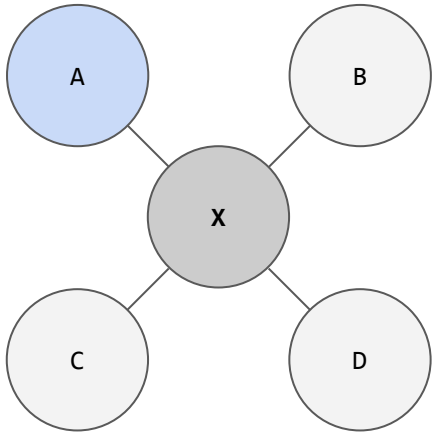


# **Option 2**: Choose by city cover



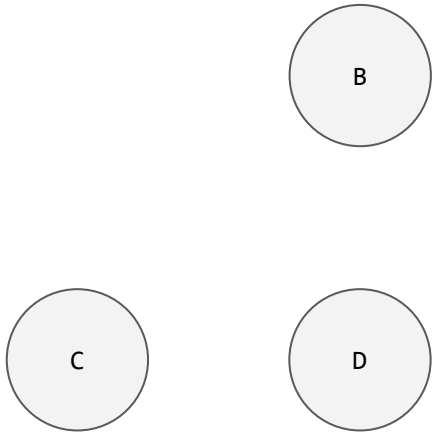

Choose to cover **X**

# **Option 2**: Choose by city cover



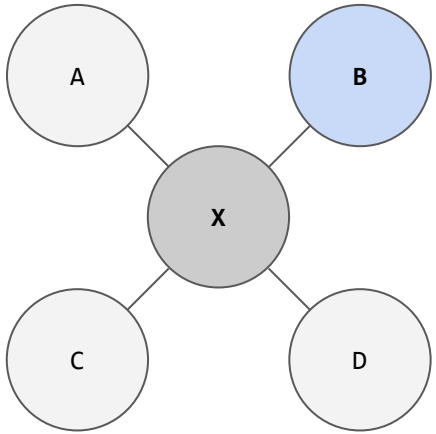

Choose to cover **X**
Pick **A**?

# **Option 2**: Choose by city cover



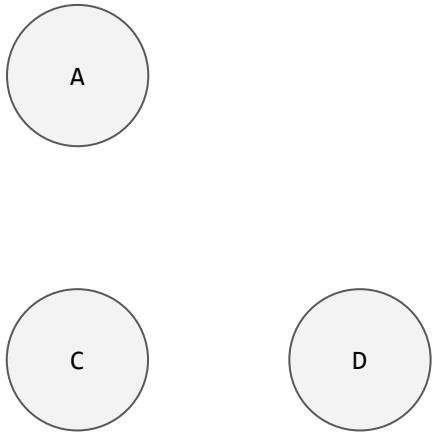

Choose to cover **X**
Pick **A**?
*Explore resulting graph*

# **Option 2**: Choose by city cover



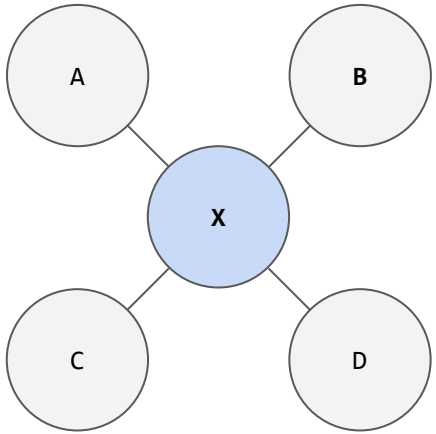

Choose to cover **X**
Pick **B**?

# **Option 2**: Choose by city cover




Choose to cover **X**
Pick **B**?
*Explore resulting graph*

# **Option 2**: Choose by city cover




Choose to cover **X**
Pick **X**?

# **Option 2**: Choose by city cover

Choose to cover **X**
Pick **X**?
*Explore resulting graph*

# **Option 2**: Choose by city cover

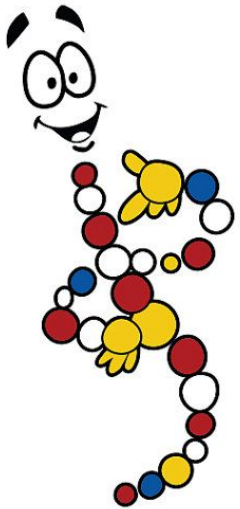
Choose to cover **X**
Pick **X**?
*Explore resulting graph*

# Tips and Tricks

- The road is bidirectional (if A → B then B → A)
- Every city appears as a key in the map
- It’s fine if you find a way to solve using fewer cities!
- Some of the test files have a *lot* of cities; your code may take up to two minutes to complete

# Questions?

# DNA Detective



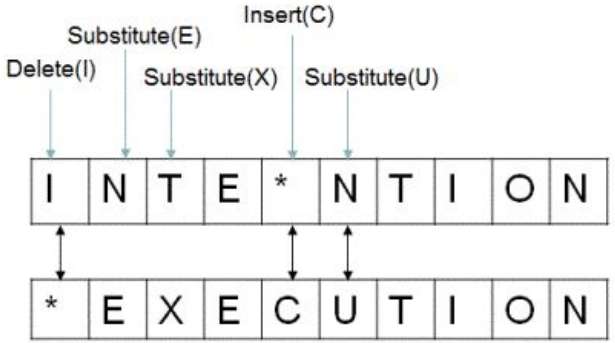
ACTGTACTGACTGACTG
CATGCATGACTATGCATC

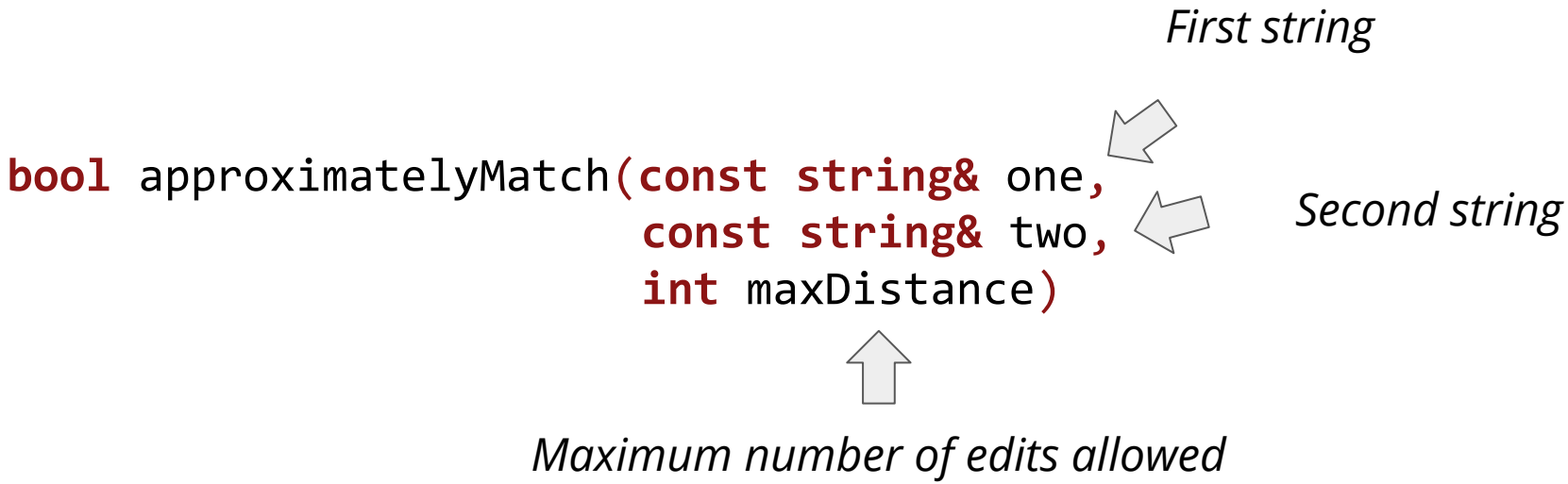
-ACTGTACTGAC--TGACTG
CA-TGCA-TGACTATGCATC

# Edit Distance

- Minimum number of operations that need to be performed to one string into another string.

- **Operations:**
  - Insert a character into one of the strings
  - Delete a character from one of the strings
  - Replace a character in one of the strings




First string
bool approximatelyMatch(const string& one,
const string& two,
int maxDistance)
Second string
Maximum number of edits allowed

```
editDistance(table, maple) → 2
editDistance(rate, pirate) → 2
editDistance(cat, dog)     → 3
```

**Hint:** Look at first char!

ATTACA
AGATACT

ATTACA
AGATACT

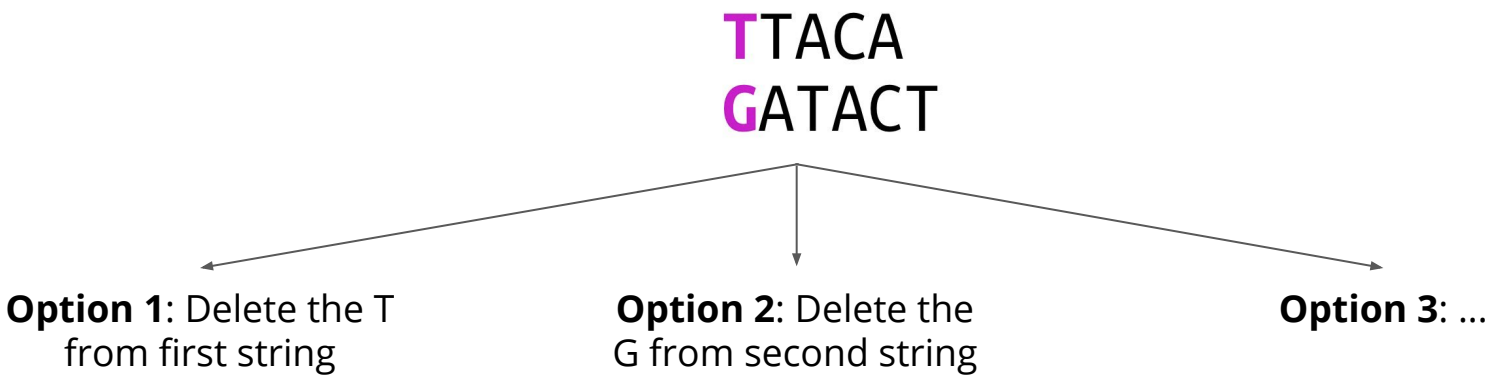
ATTACA
AGATACT



Remove A from start
of both strings

TTACA
GATACT

TTACA
GATACT

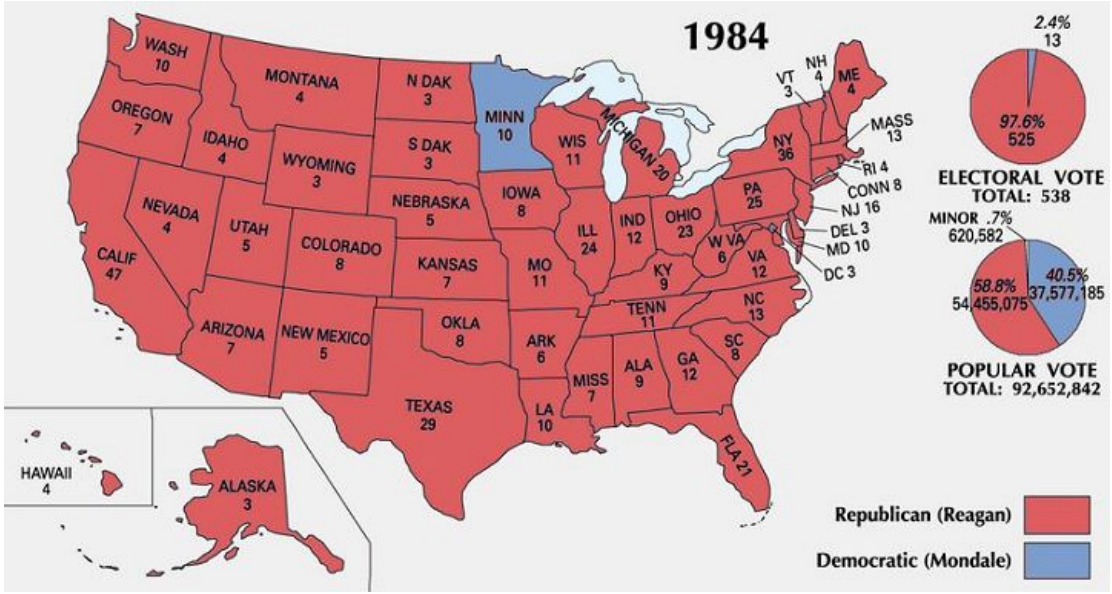

TTACA
GATACT
**Option 1**: Delete the T from first string
**Option 2**: Delete the G from second string
**Option 3**: ...

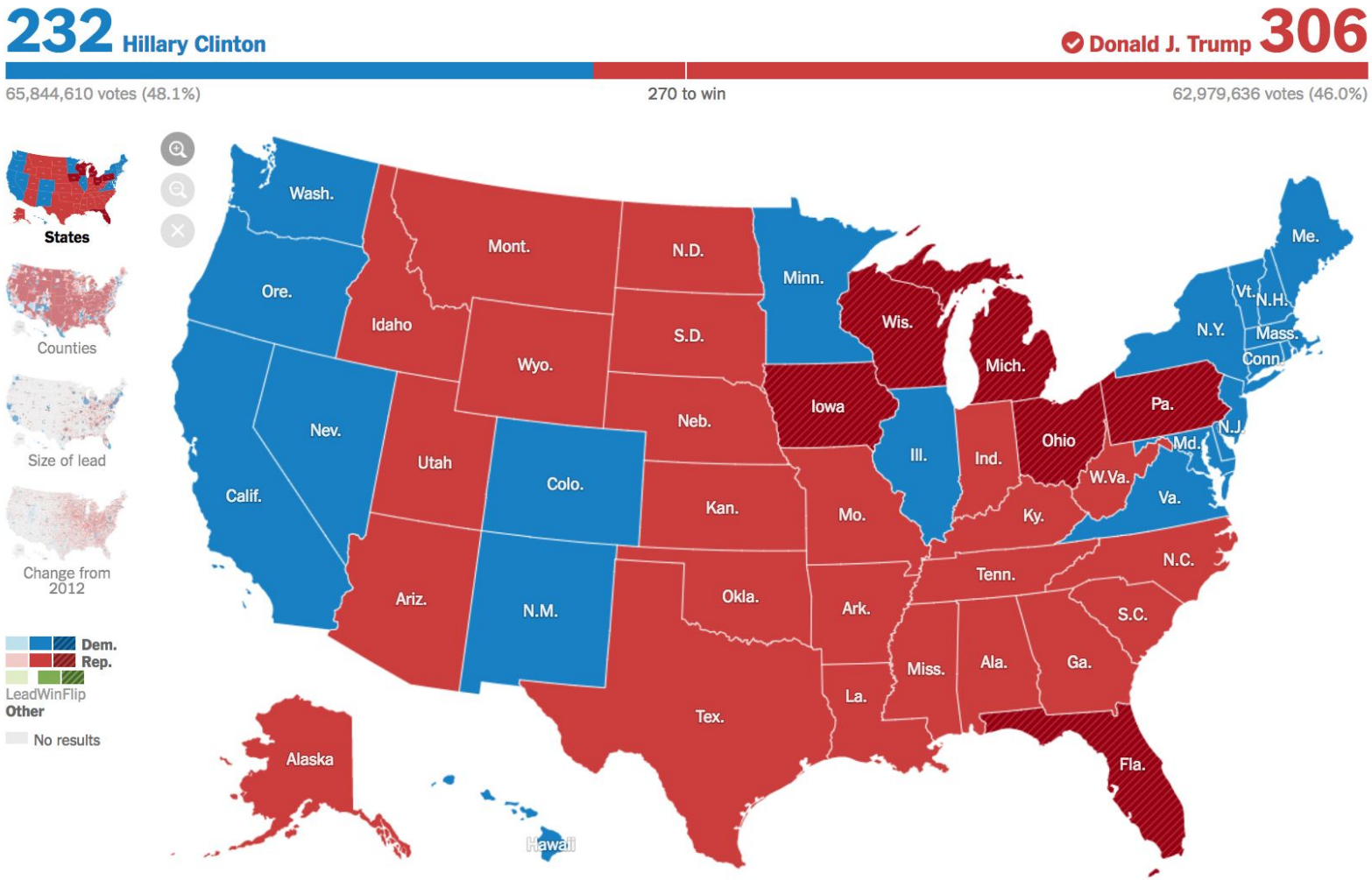
# Questions?

# Winning the Presidency

# US Presidency

- Every state has a certain number of electors and a certain population
- If you win **majority** of popular votes in a state, you get all electors
- You need a **majority** of electoral votes to become president!




232 Hillary Clinton
Donald J. Trump 306
65,844,610 votes (48.1%)
270 to win
62,979,636 votes (46.0%)
States
Counties
Size of lead
Change from 2012
Dem.
Rep.
LeadWinFlip
Other
No results
Wash.
Ore.
Calif.
Idaho
Nev.
Mont.
Wyo.
Utah
Ariz.
Colo.
N.M.
N.D.
S.D.
Neb.
Kan.
Okla.
Tex.
Minn.
Iowa
Mo.
Ark.
La.
Wis.
Ill.
Mich.
Ind.
Ohio
Ky.
Tenn.
Miss.
Ala.
W.Va.
Pa.
N.Y.
Va.
N.C.
S.C.
Ga.
Fla.
Me.
Vt.
N.H.
Mass.
Conn.
N.J.
Md.
Alaska
Hawaii

```
struct State {
    string name;
    int electoralVotes;
    int popularVotes;
};
```

```
State michigan = {“Michigan”,
                  16,
                  9,910,000};
```

If **4,955,001** Michiganders vote for you, you gain **16** electoral votes!

If **4,955,000** Michiganders vote for you, you gain **0** electoral votes…

# **Question**: What's the fewest number of votes needed to be elected president?

```
MinInfo minPopularVoteToWin(Vector<State> states)
```

```
struct MinInfo {
   int popularVotesNeeded;
   Vector<State> statesUsed;
};
```

# Demo!

# There are different ways of thinking of the problem

**Think about this problem instead:**

What is the minimum number of popular votes needed to get at least **V** electoral votes, using only states from index **i** and greater in the **Vector** of states?

*If you can solve this problem, you solve the original problem!*

**What if it's impossible?**
(e.g. one state left, and 75 electoral votes needed)

Use **MAX_INT** sentinel value!

**Memoization is required!**
(or else your code will never finish...)

# Questions?