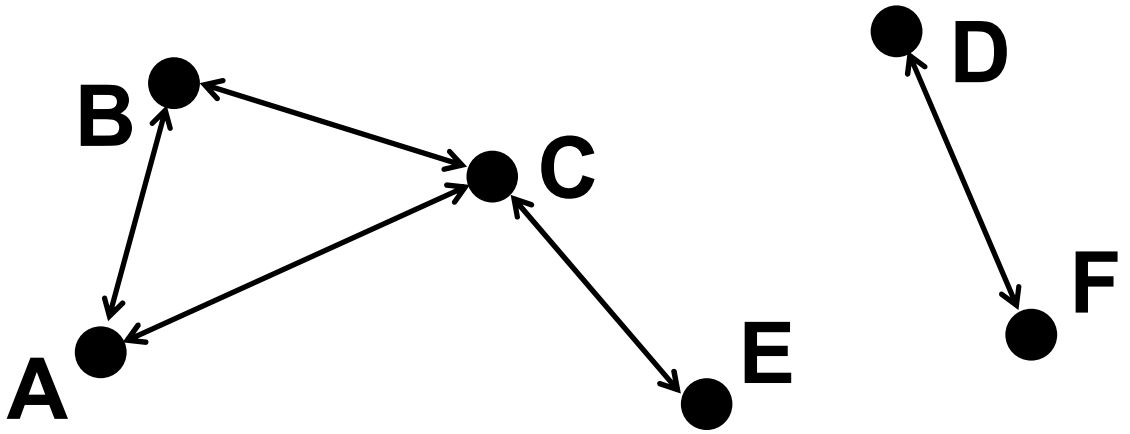
Graphs and Search Algorithms

Social Network Backend

Want to build social network app tracking:

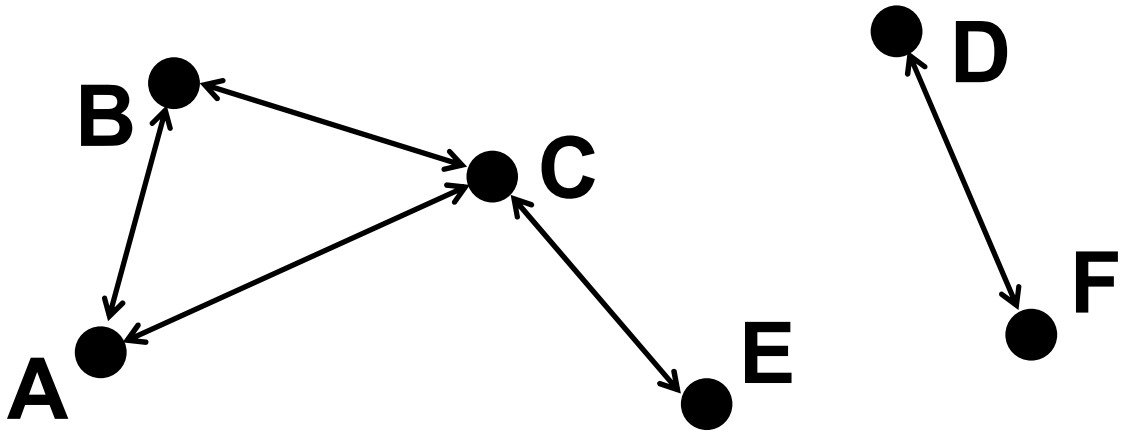
- users
- *friend* relationship between some pairs of users



Graph Basics

Network is a **graph**:

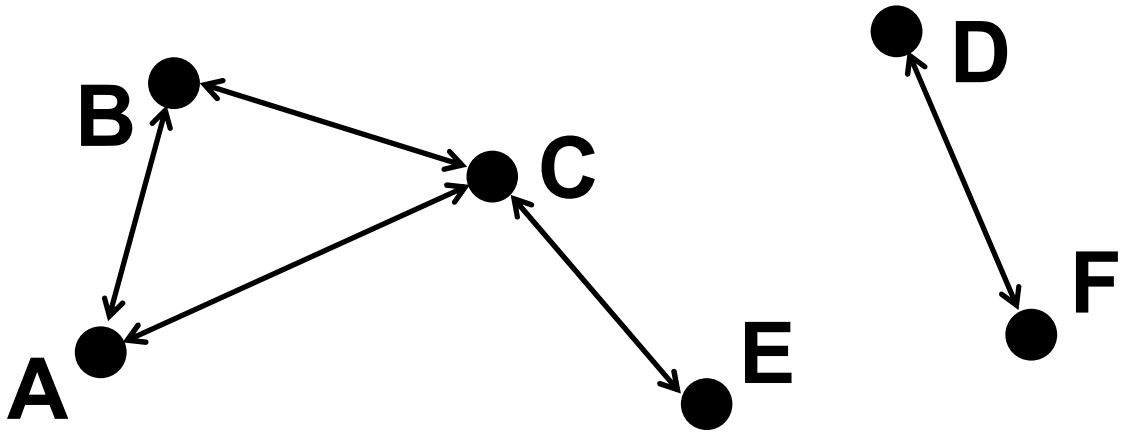
- **nodes** or **vertices** $V = \{A, B, C, D, E, F\}$



Graph Basics

Network is a **graph**:

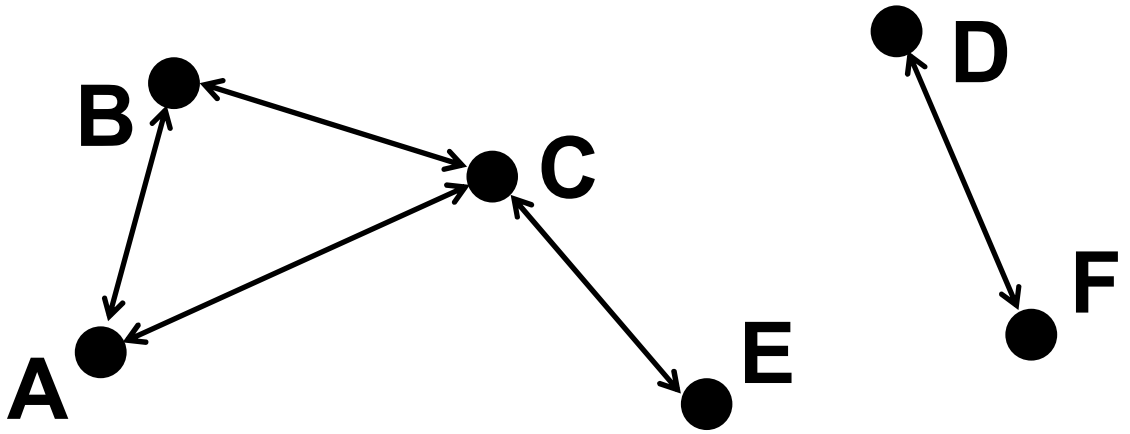
- **nodes** or **vertices** $V = \{A, B, C, D, E, F\}$
- **edges** $E = \{\{A, B\}, \{A, C\}, \{B, C\}, \ldots\}$



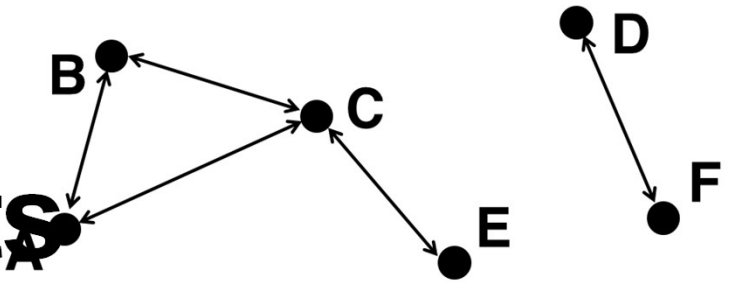
Graph Basics

Network is a **graph**:

- **nodes** or **vertices** $V = \{A, B, C, D, E, F\}$
- **edges** $E = \{\{A, B\}, \{A, C\}, \{B, C\}, \ldots\}$
  - can be **directed** (one-way) or **undirected**



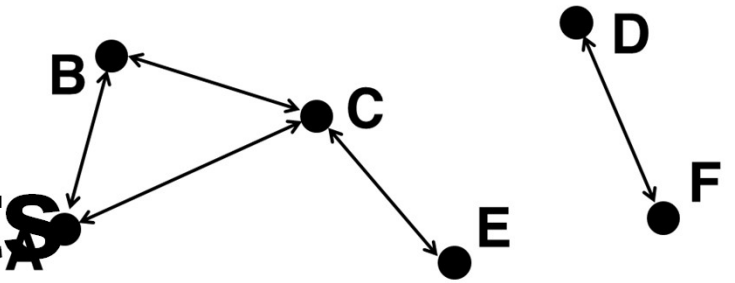
Graph Data Structs



1. Raw vertex/edge lists

$$V = \{A, B, C, D, E, F\}$$
$$E = \{\{A, B\}, \{A, C\}, \{B, C\}, \ldots\}$$

Graph Data Structs



1. Raw vertex/edge lists

$$V = \{A, B, C, D, E, F\}$$

$$E = \{\{A, B\}, \{A, C\}, \{B, C\}, \ldots\}$$

2. Adjacency list

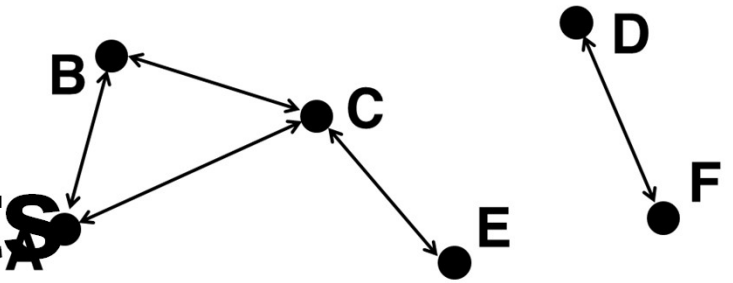
$$A : \{B, C\}$$

$$B : \{A, C\}$$

$$C : \{A, B, E\}$$

$$\ldots$$

Graph Data Structs



1. Raw vertex/edge lists

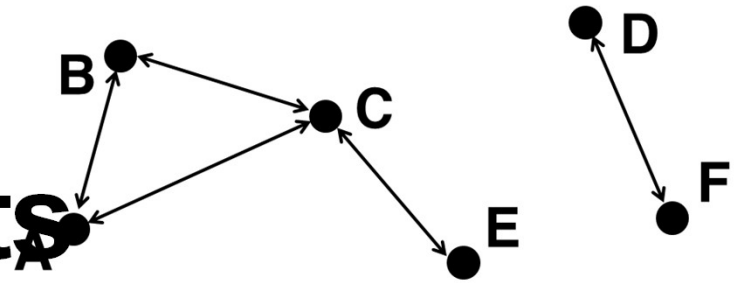
$$V = \{A, B, C, D, E, F\}$$

$$E = \{\{A, B\}, \{A, C\}, \{B, C\}, \ldots\}$$

2. Adjacency list
3. Adjacency matrix

	A	B	C	D	E	F
A	0	1	1	0	0	0
B						
C						
D						
E						
F						

Graph Data Structs



1. Raw vertex/edge lists

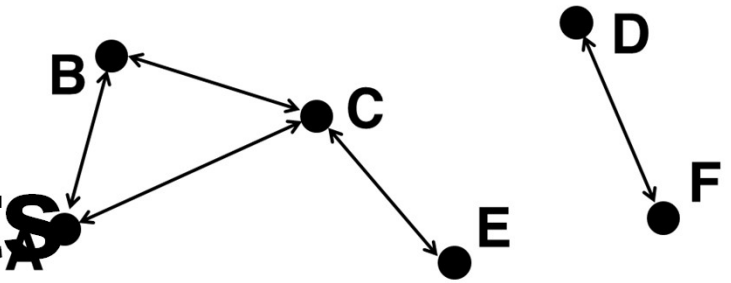
$$V = \{A, B, C, D, E, F\}$$

$$E = \{\{A, B\}, \{A, C\}, \{B, C\}, \ldots\}$$

2. Adjacency list
3. Adjacency matrix

	A	B	C	D	E	F
A	0	1	1	0	0	0
B	1	0	1	0	0	0
C	1	1	0	0	1	0
D	0	0	0	0	0	1
E	0	0	1	0	0	0
F	0	0	0	1	0	0

Graph Data Structs



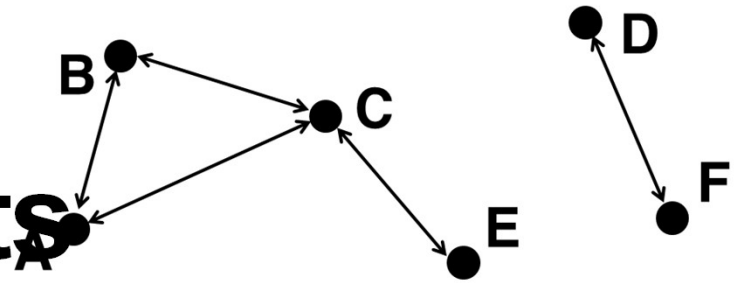
What is the space cost of each option?

Raw lists:

Adjacency list:

Adjacency matrix:

Graph Data Structs



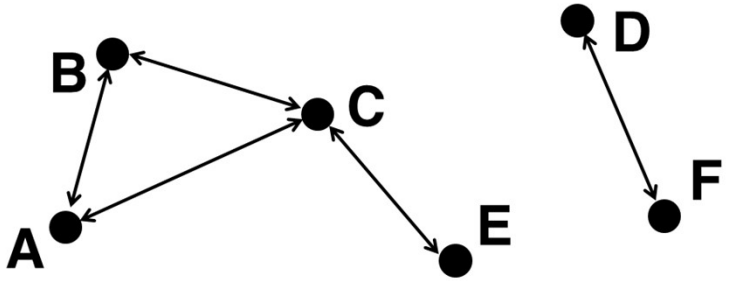
What is the space cost of each option?

Raw lists: $O(|V| + |E|)$

Adjacency list: $O(|V| + |E|)$

Adjacency matrix: $O(|V|^2)$

Graph Operations



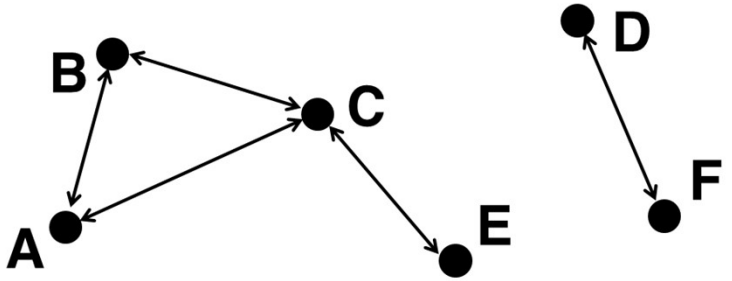
Given two vertices, are they neighbors?

Raw lists:

Adjacency list:

Adjacency matrix:

Graph Operations



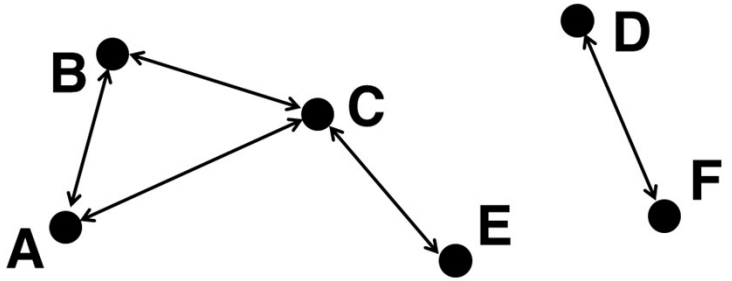
Given two vertices, are they neighbors?

Raw lists: search entire edge list $O(|E|)$

Adjacency list:

Adjacency matrix:

Graph Operations



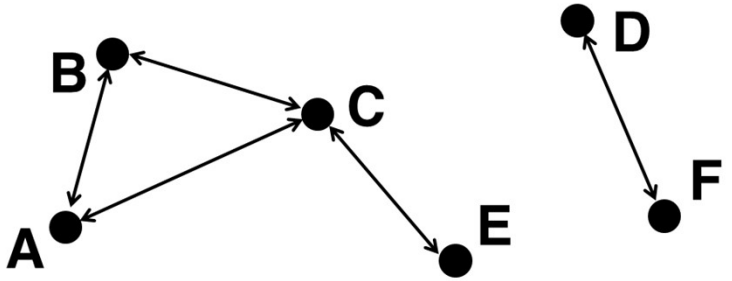
Given two vertices, are they neighbors?

Raw lists: search entire edge list $O(|E|)$

Adjacency list: search one adjacency list (technically $O(|V|)$)

Adjacency matrix:

Graph Operations



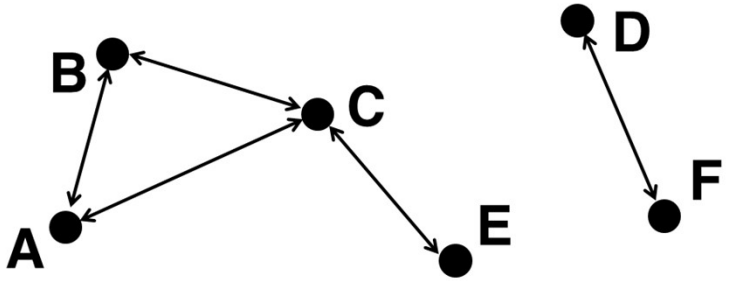
Given two vertices, are they neighbors?

Raw lists: search entire edge list $O(|E|)$

Adjacency list: search one adjacency list (technically $O(|V|)$)

Adjacency matrix: look up entry $O(1)$

Graph Operations



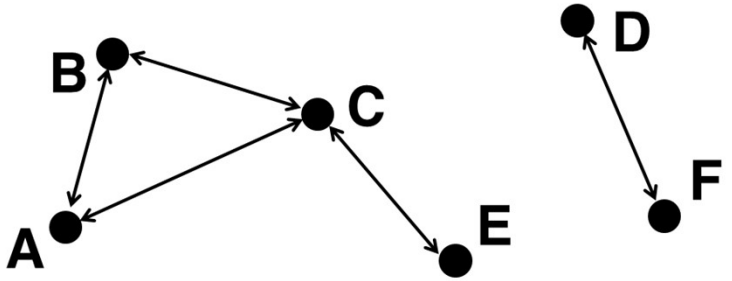
Given a vertex, who are the neighbors?

Raw lists: search entire edge list $O(|E|)$

Adjacency list:

Adjacency matrix:

Graph Operations



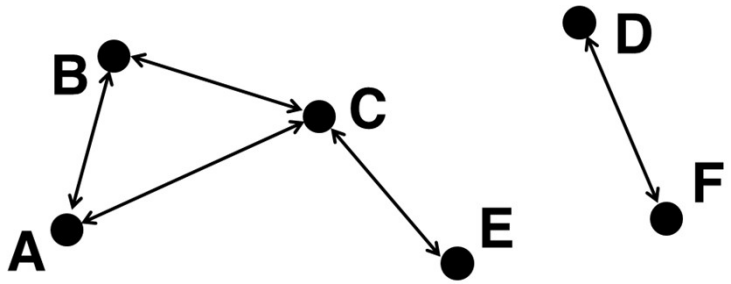
Given a vertex, who are the neighbors?

Raw lists: search entire edge list $O(|E|)$

Adjacency list: nothing to do… $O(1)$

Adjacency matrix: search row of matrix $O(|V|)$

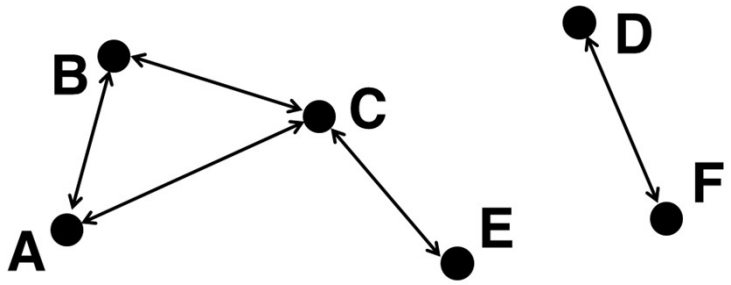
Friend Network



Given a social network containing people (vertices) and friend relationships (edges), **A** is in the same friend network as **B** if

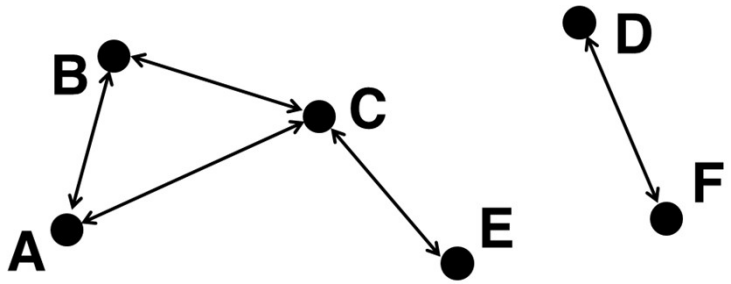
- they are the same person
- **A** is friends with someone that is in the same friend network as **B**

Friend Network



Are **A** and **E** in the same friend network?

Friend Network

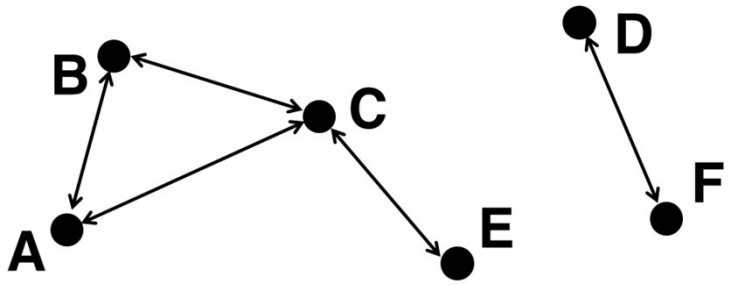


Are **A** and **E** in the same friend network?

Basic idea: start at **A** and "flood fill" along edges, and see if we ever hit **E**

(We will need to create a "visited" flag for vertices)

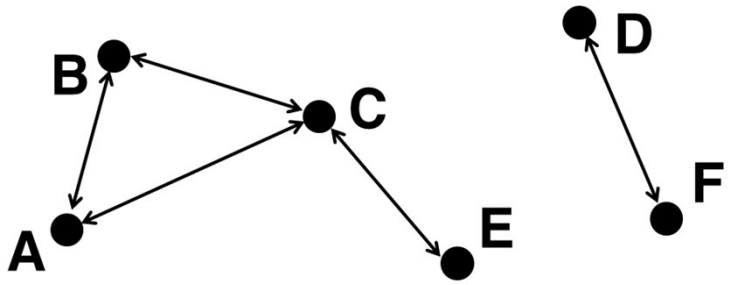
Friend Network




```
friendNetwork(A,B)
for each vertex v:
   v.visited = false;
return search(A);
```

```
search(v)
if(v == B) return true;
if(v.visited) return false;
v.visited = true;
for each neighbor w:
  if(search(w))
    return true;
return false;
```

Friend Network



friendNetwork(**A**,**B**)

```
for each vertex v:
   v.visited = false;
return search(A);
```

search(**v**)

```
if(v == B) return true;
if(v.visited) return false;
v.visited = true;
for each neighbor w:
  if(search(w))
    return true;
return false;
```

(are there potential issues?)

Iterative Version

```
friendNetwork(A,B)
for each vertex v:
   v.visited = false;
stack S = {A};
while(!S.empty())
  v = S.pop();
  if(v == B) return
   true;
  if(v.visited) continue;
  v.visited = true;
  for each neighbor w:
    S.push(w);
return false;
```

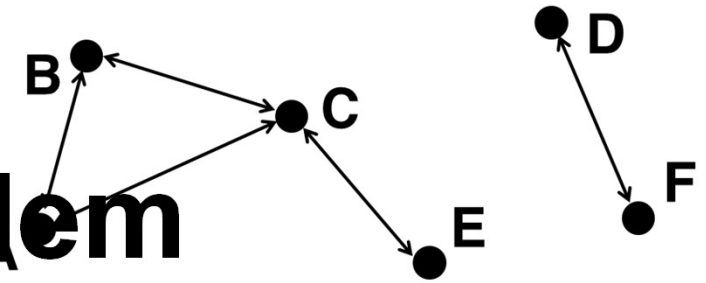
Iterative Version

```
friendNetwork(A,B)
for each vertex v:
   v.visited = false;
stack S = {A};
```

depth-first search (DFS)

```
while(!S.empty())
  v = S.pop();
  if(v == B) return
   true;
  if(v.visited) continue;
  v.visited = true;
  for each neighbor w:
    S.push(w);
return false;
```

Kevin Bacon Problem



Given a social network and two people **A**, **B**, what is the shortest chain of friends from **A** to **B**?

Ex:

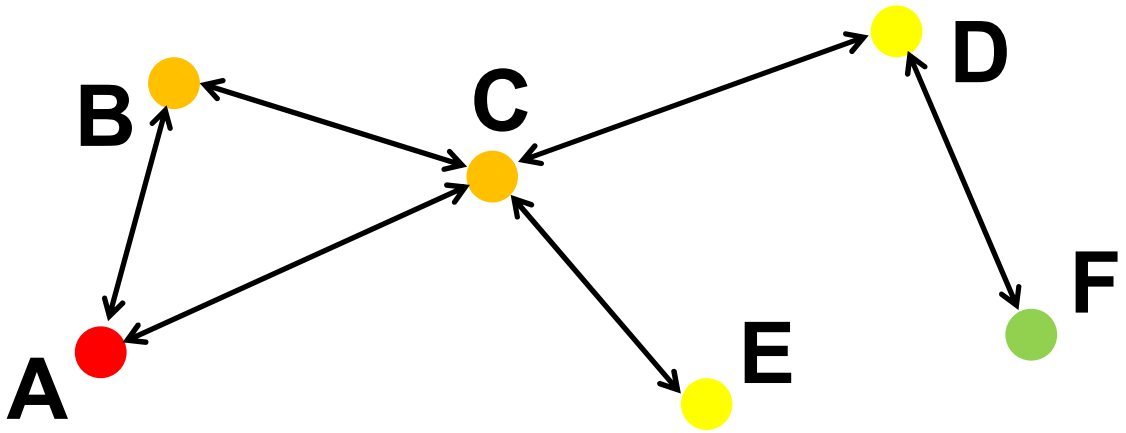
bacon(**A, A**) = 0

bacon(**A, E**) = 2

bacon(**C, D**) = infinity

Kevin Bacon Problem

Intuition: when calculating bacon(**A**, *) we still want to flood-fill, but we need to guarantee we search friends **before** friends-of-friends



Breadth-First Search

```
friendNetwork(A,B)
for each vertex v:
   v.visited = false;
queue Q = {A};
```

```
while(!Q.empty())
  v = Q.pop();
  if(v == B) return
   true;
  if(v.visited) continue;
  v.visited = true;
  for each neighbor w:
    Q.push(w);
return false;
```

Kevin Bacon Problem

```
bacon(A,B)
for each vertex v:
    v.visited = false;
   v.dist = infinity;
queue Q = {A};
A.dist = 0;
```

```
while(!Q.empty())
  v = Q.pop();
  if(v == B) return v.dist;
  if(v.visited) continue;
  v.visited = true;
  for each neighbor w:
    Q.push(w);
    w.dist = v.dist + 1;
return infinity;
```