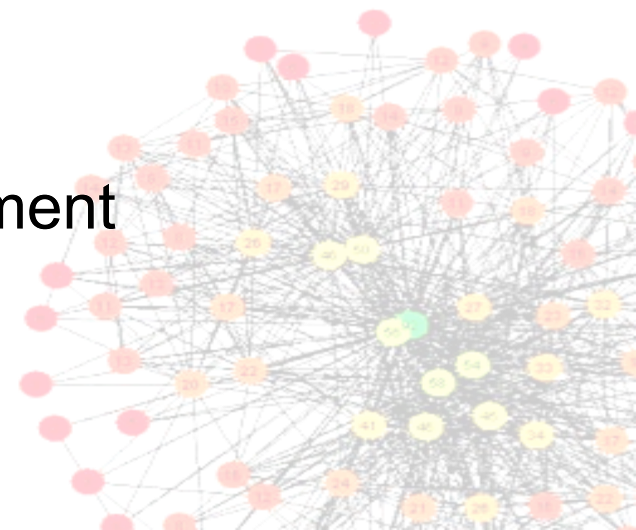
Graph Traversals:
Breadth-First and Depth-First Search

Eric Eaton

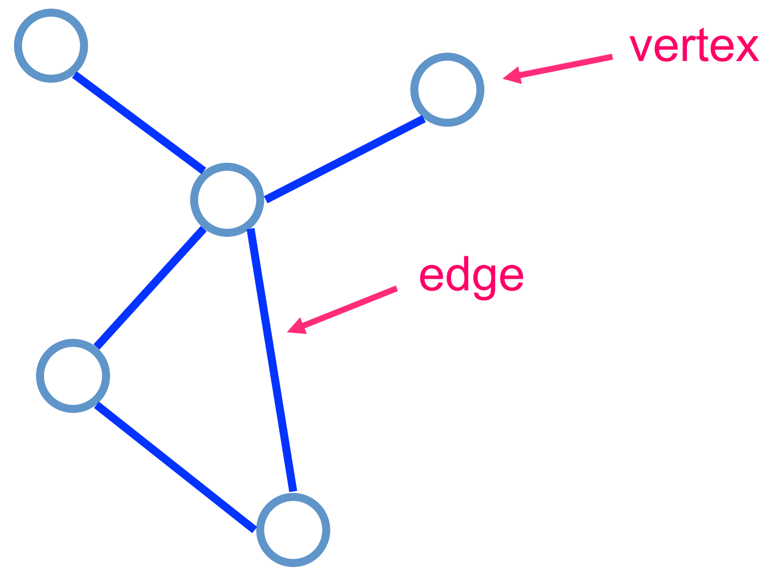
Bryn Mawr College
Computer Science Department



What is a Graph?

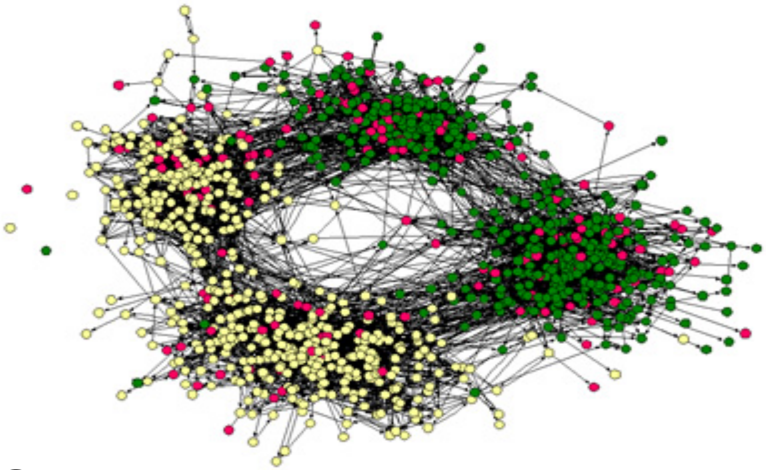
Graphs are collections of vertices joined by edges

"Graph" ≡ "Network"

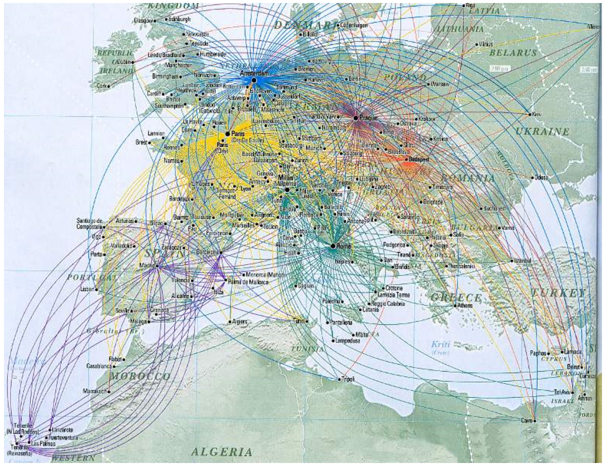


Vertices	Edges	
vertices	edges, arcs	math
nodes	links, relations	computer science
sites	bonds	physics
actors	ties, relations	sociology

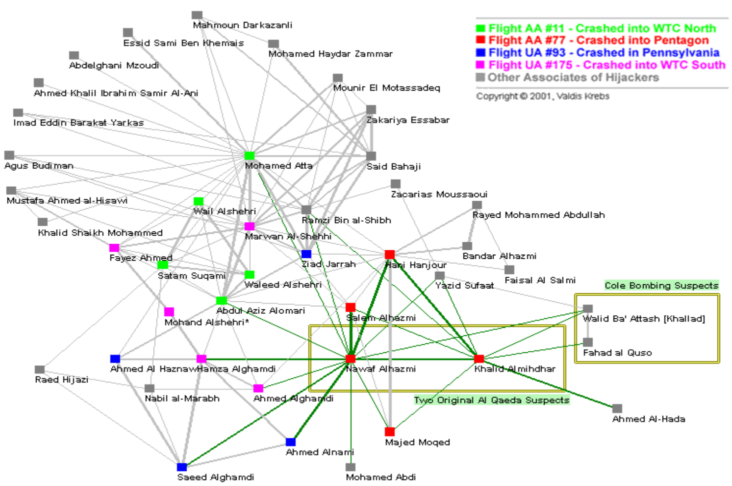
Example Networks



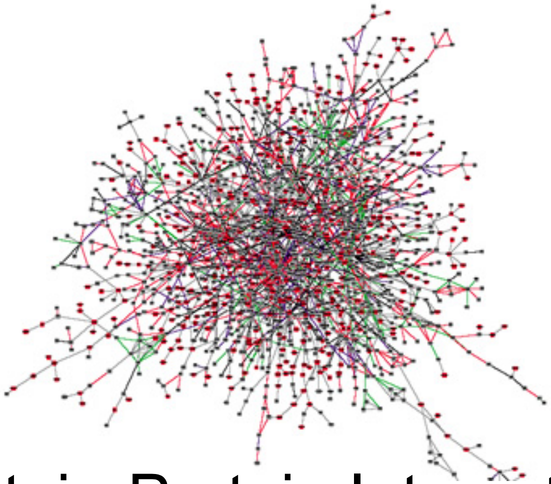
School Friendship Network
(from Moody 2001)



Airline Network
(Source: Northwest Airlines)



Terrorist Network
(by Valdis Krebs, Orgnet.com)



Protein-Protein Interactions
(by Peter Uetz)

Other Applications

- Intersections and streets within a city
- Computer networks
- Electronic circuits
- Food webs
- Gene regulatory networks
- Steps to solve a puzzle
- many more...

Outline

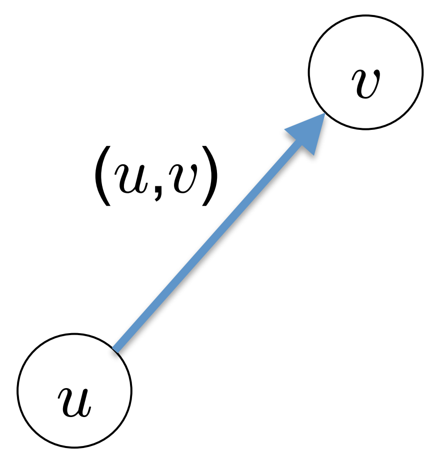
- **Introduction**
- Graph Basics
- Graph Search Problem
  - Breadth-First Search
  - Depth-First Search
- Complexity Analysis

Outline

- Introduction
- **Graph Basics**
- Graph Search Problem
  - Breadth-First Search
  - Depth-First Search
- Complexity Analysis

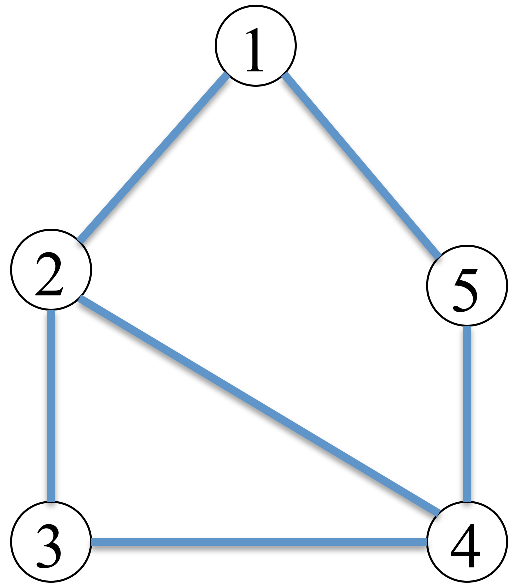
Basic Graph Definitions

- A ***graph*** $G = (V,E)$ consists of a finite set of ***vertices*** V and a finite set of ***edges*** E
- Each edge is a pair (u,v) where $u,v \in V$
  - V and E are sets, so each vertex $u \in V$ is unique, and each edge $e \in E$ is unique
  - v is **adjacent to** u
- We will focus on two types:
  - Undirected graphs
  - Directed graphs



Undirected Graph

All edges are two-way

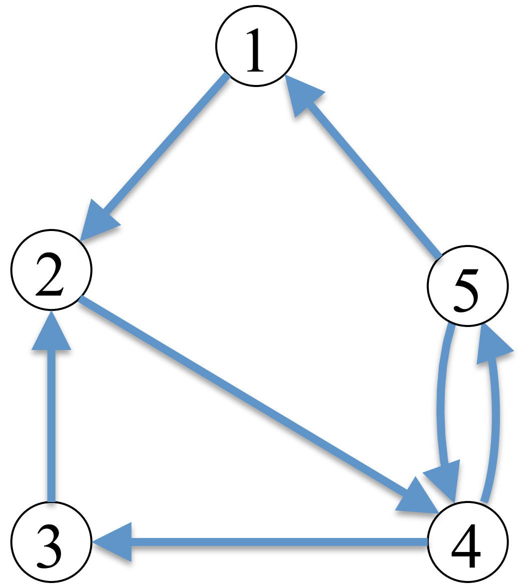


- V = { 1, 2, 3, 4, 5 }
- Edges are unordered pairs:

  E = { {1,2}, {2,3}, {3,4}, {2,4}, {4,5}, {5,1} }

Directed Graph

All edges are "one-way" as indicated by the arrows



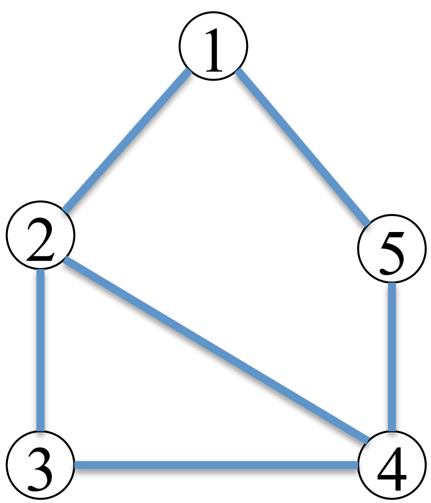
- V = { 1, 2, 3, 4, 5 }
- Edges are ordered pairs:

  E = { (1,2), (2,4), (3,2), (4,3), (4,5), (5,1), (5,4) }

Degree

Undirected Graphs

degree(u): the number of edges $\{u,v\}$ for all $v \in V$



Degree

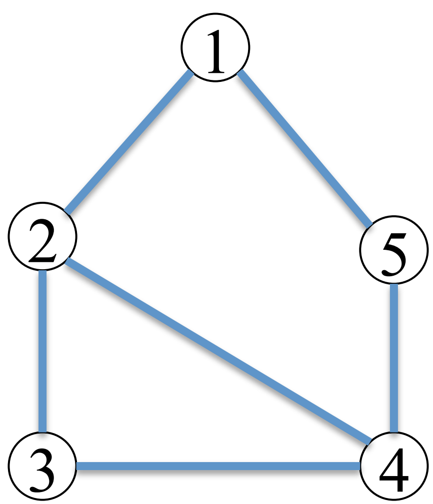
Undirected Graphs

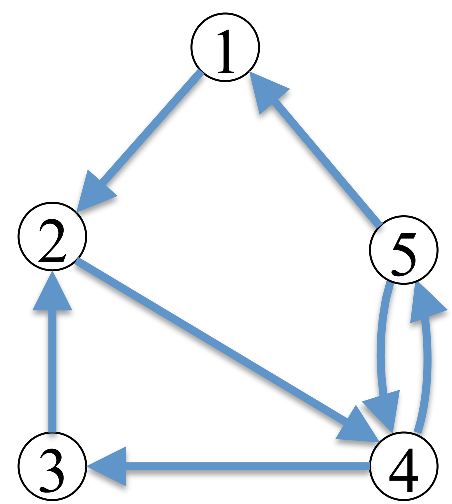
degree(u): the number of edges $\{u,v\}$ for all $v \in V$

Directed Graphs

in-degree(u): the number of edges (v,u) for all $v \in V$

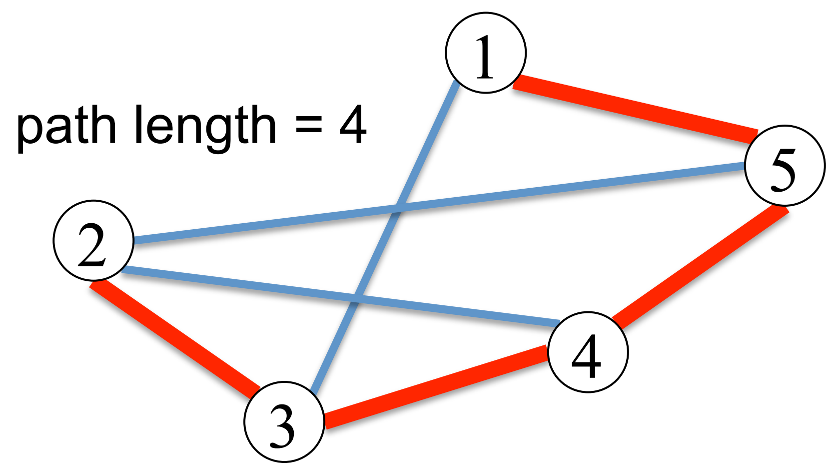
out-degree(u): the number of edges (u,v) for all $v \in V$





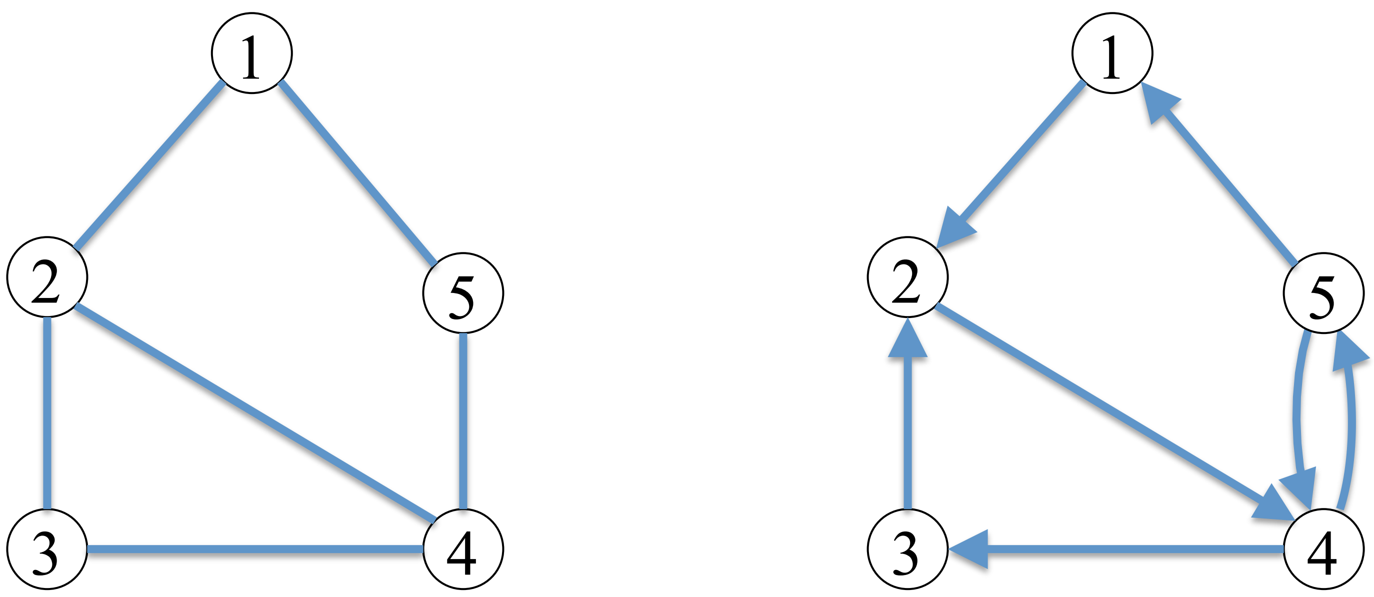
Paths in Graphs

- A ***path*** in a graph is a sequence of vertices $w_1, w_2, \ldots, w_n$ s.t. $(w_i, w_{i+1}) \in E$ for $1 \leq i < n$
- The path's ***length*** is the number of edges on the path
  - The length of the path from a vertex to itself is 0
- In a ***simple path***, all vertices are distinct
  - The first and last vertices may be the same



Paths in Graphs

- How many simple paths are there from 1 to 4 and what are their lengths?



Outline

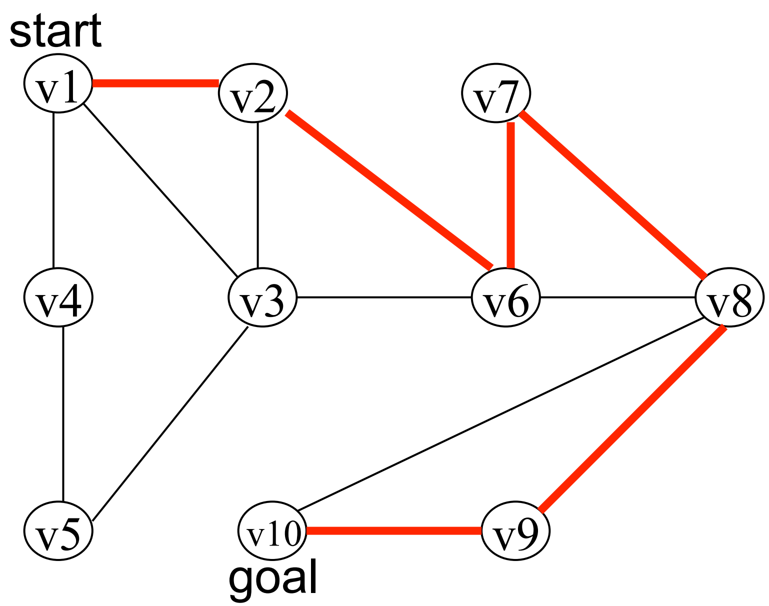
- Introduction
- **Graph Basics**
- Graph Search Problem
  - Breadth-First Search
  - Depth-First Search
- Complexity Analysis

Outline

- Introduction
- Graph Basics
- **Graph Search Problem**
  - Breadth-First Search
  - Depth-First Search
- Complexity Analysis

Graph Search Problem

- **Goal**: Find a simple path from a starting vertex to a goal vertex



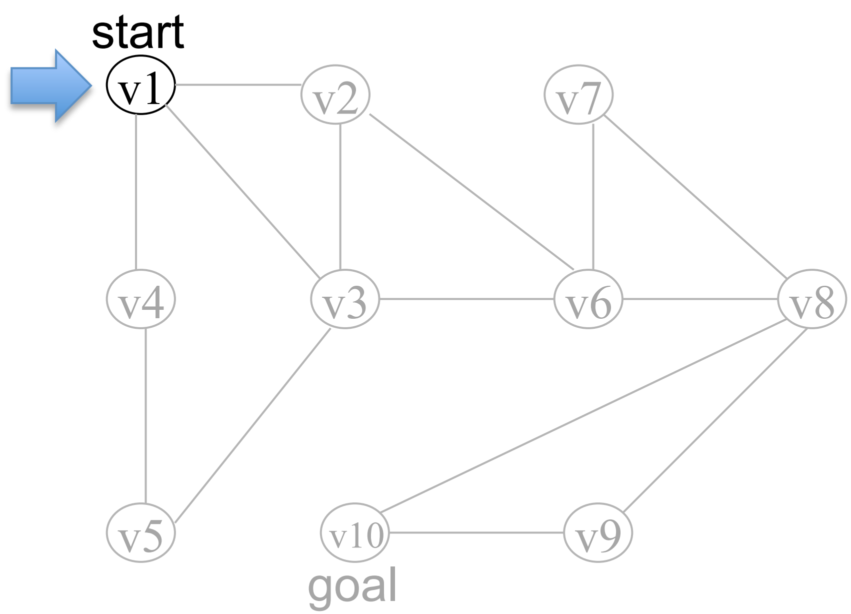
- What applications can be framed as instances of this problem?

Intuition

- From starting vertex, keep expanding vertices until we find the goal

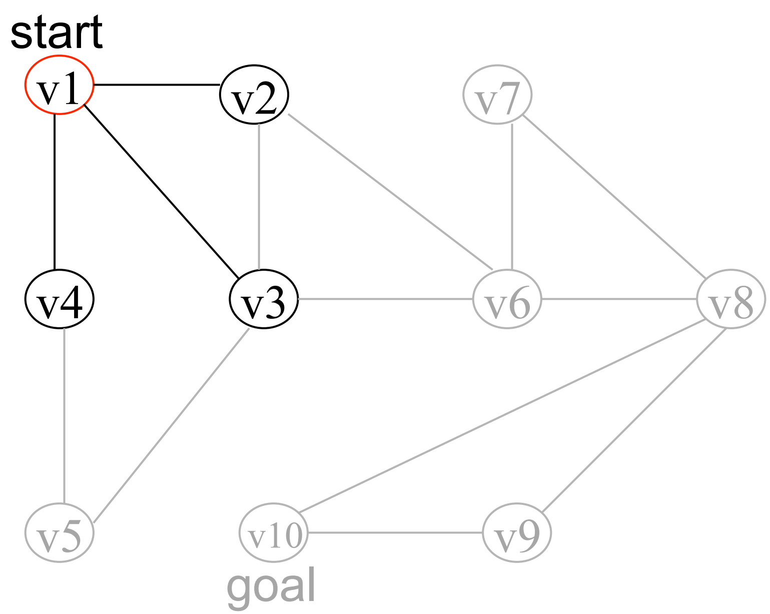
Intuition

- From starting vertex, keep expanding vertices until we find the goal



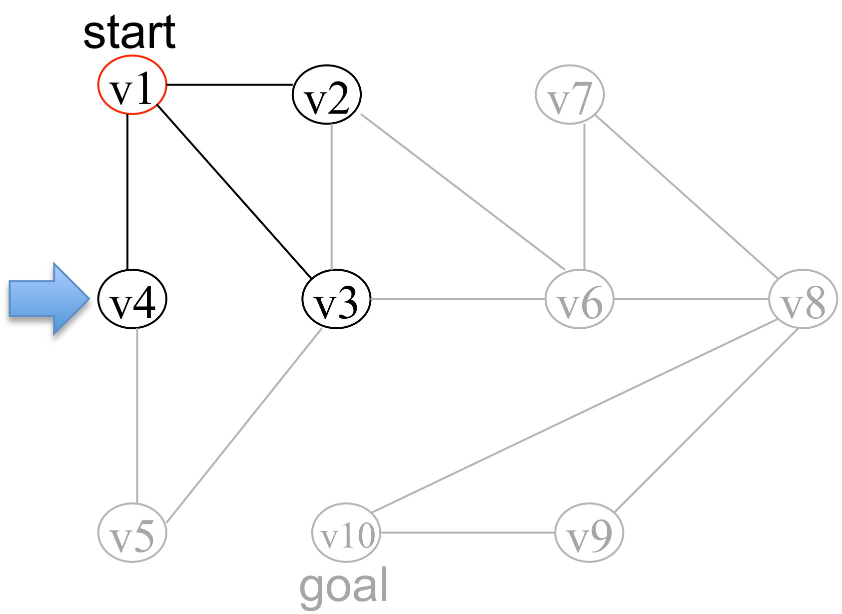
Intuition

- From starting vertex, keep expanding vertices until we find the goal



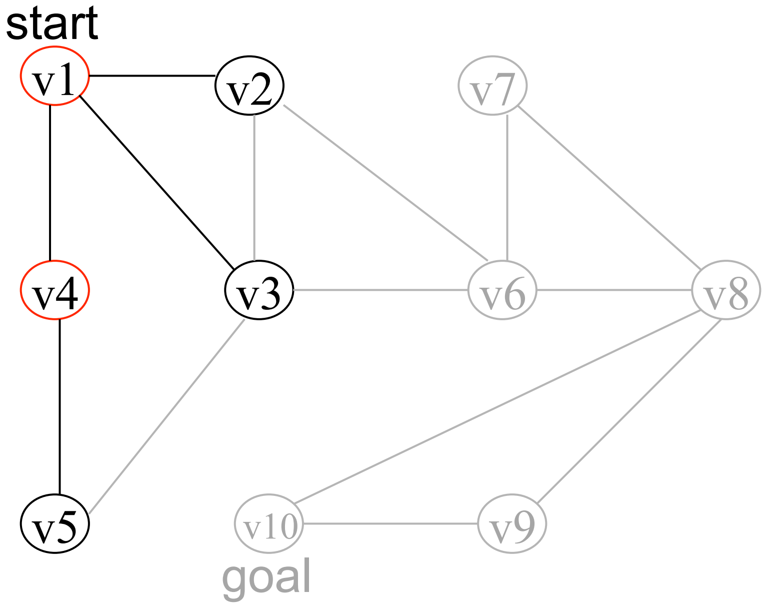
Intuition

- From starting vertex, keep expanding vertices until we find the goal



Intuition

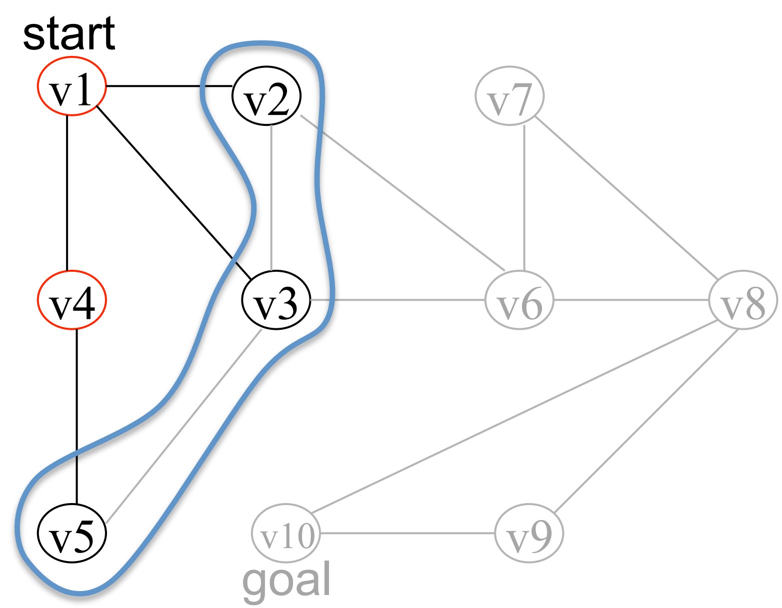
- From starting vertex, keep expanding vertices until we find the goal



- Breadth-First: expand shallowest unexpanded vertex
- Depth-First: expand deepest unexpanded vertex

Queuing Function

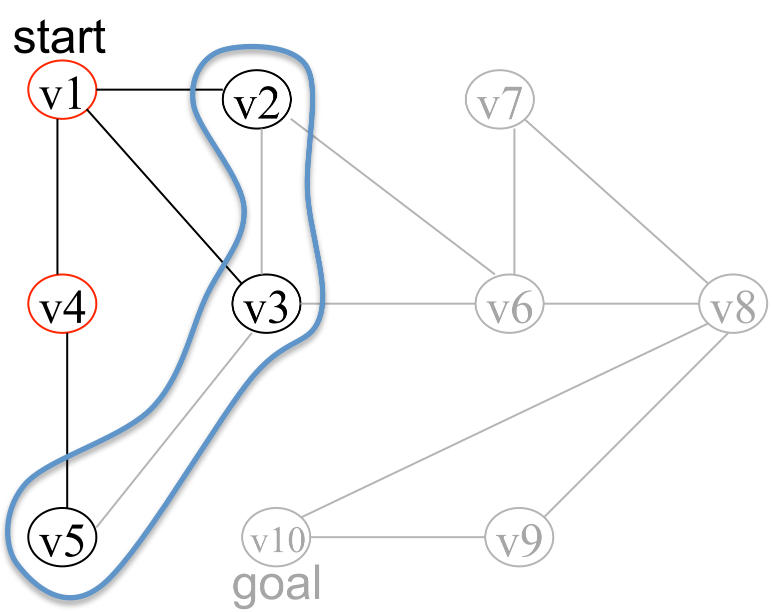
- Used to maintain a ranked list of nodes that are candidates for expansion
  - Called the “fringe”



- Substituting different queuing functions yields different searches

Protection Against Cycles

- We need to guard against cycles
  - Mark each vertex as "closed" when we encounter it
  - Do not consider closed vertices again



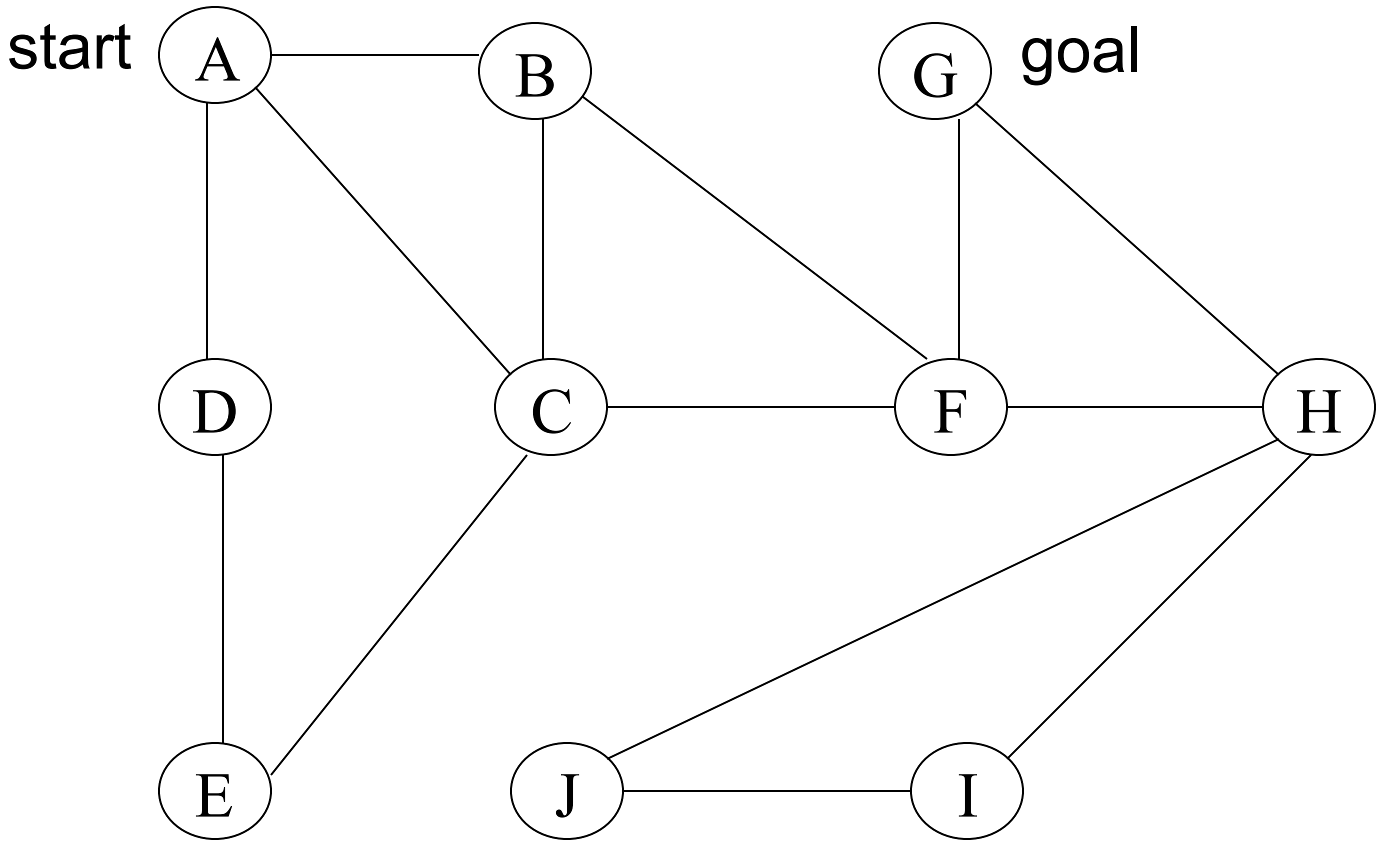
Bookkeeping Structures

- **Node**:
  - vertex ID
  - predecessor node
  - path length
- **Problem**:
  - graph
  - starting vertex
  - goalTest(Vertex v) – tests if vertex is a goal state

General Graph Search

```
// problem describes the graph, start vertex, and goal test
// queueingfn is a comparator function that ranks two states
// graphSearch returns either a goal node or failure
graphSearch(problem, queuingFn) {
    open = {}, closed = {}                        //empty lists
    queuingFn(open, new Node(problem.startvertex))  //init
    loop {
      if empty(open) then return FAILURE        //no nodes remain
      c = removeFront(open)                     //get current node
      if problem.goalTest(c.vertex)             //goal test
         return c
      if c.vertex is not in closed    {        //avoid duplicates
         add c.vertex to closed
         for each Vertex w adjacent to c.vertex  //expand node
             if w is not in closed
                queuingFn(open, new Node(w,c));
      }
    }
}
```

Application: Route Finding



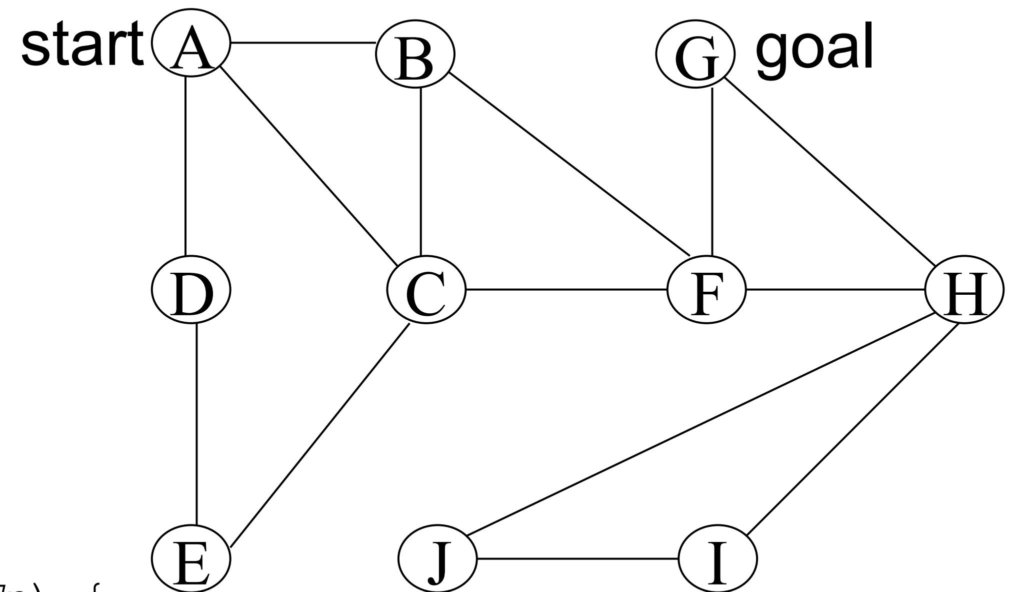
Breadth-First Search

Expands the “shallowest” vertex

Application: Route Finding (BFS)

open list

closed list



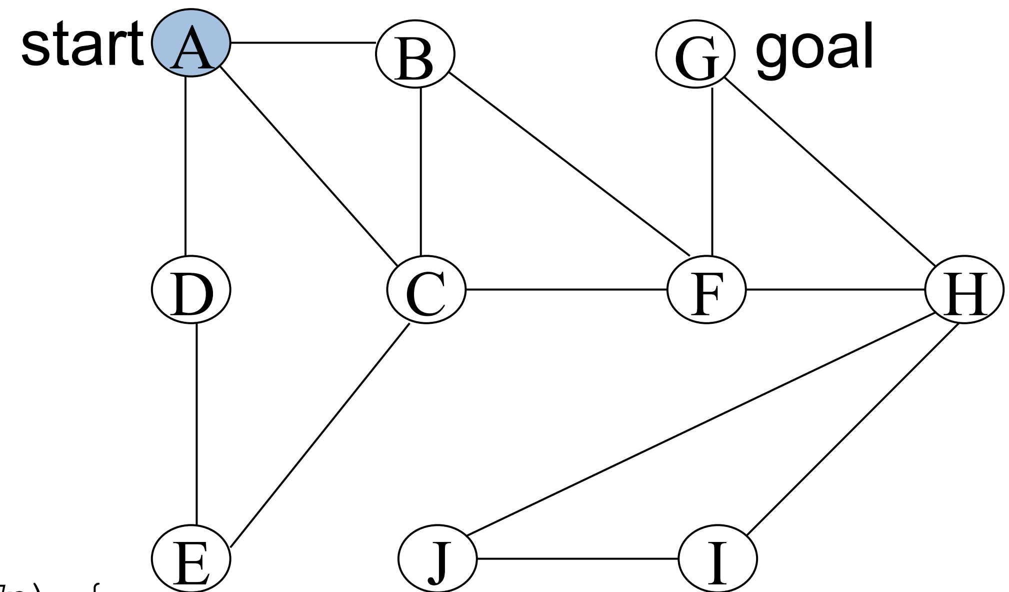

```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
}}}
```

Application: Route Finding (BFS)

open list

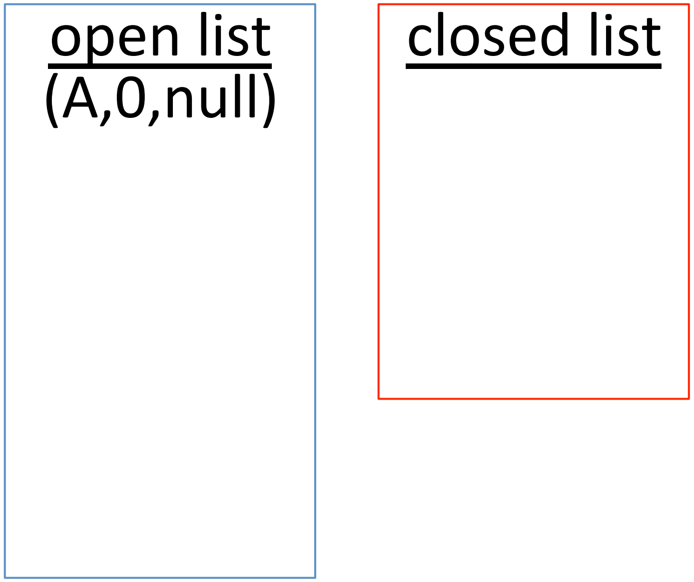
(A,0,null)

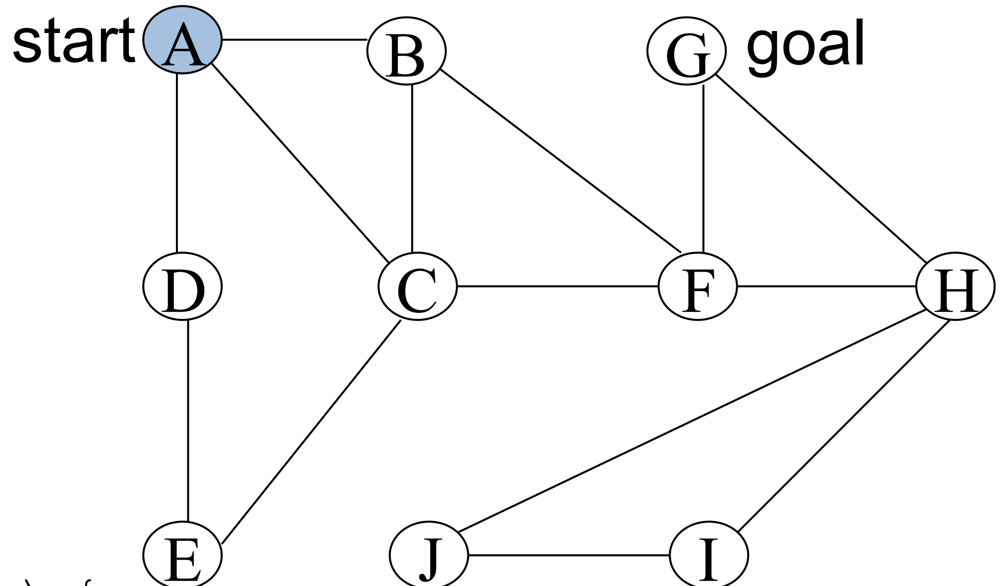
closed list




```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
    }}}
```

Application: Route Finding (BFS)





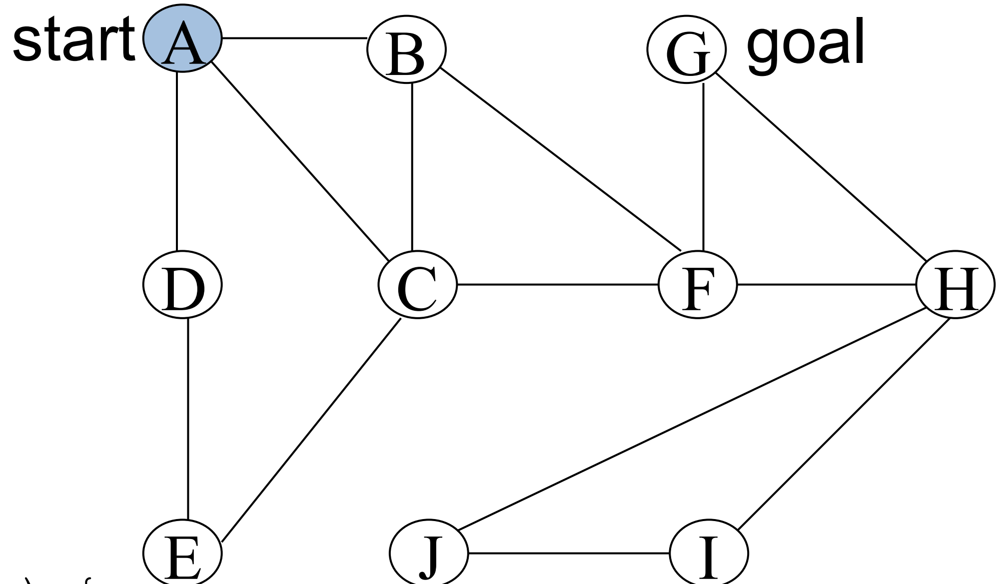

```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
}}}
```

Application: Route Finding (BFS)

open list

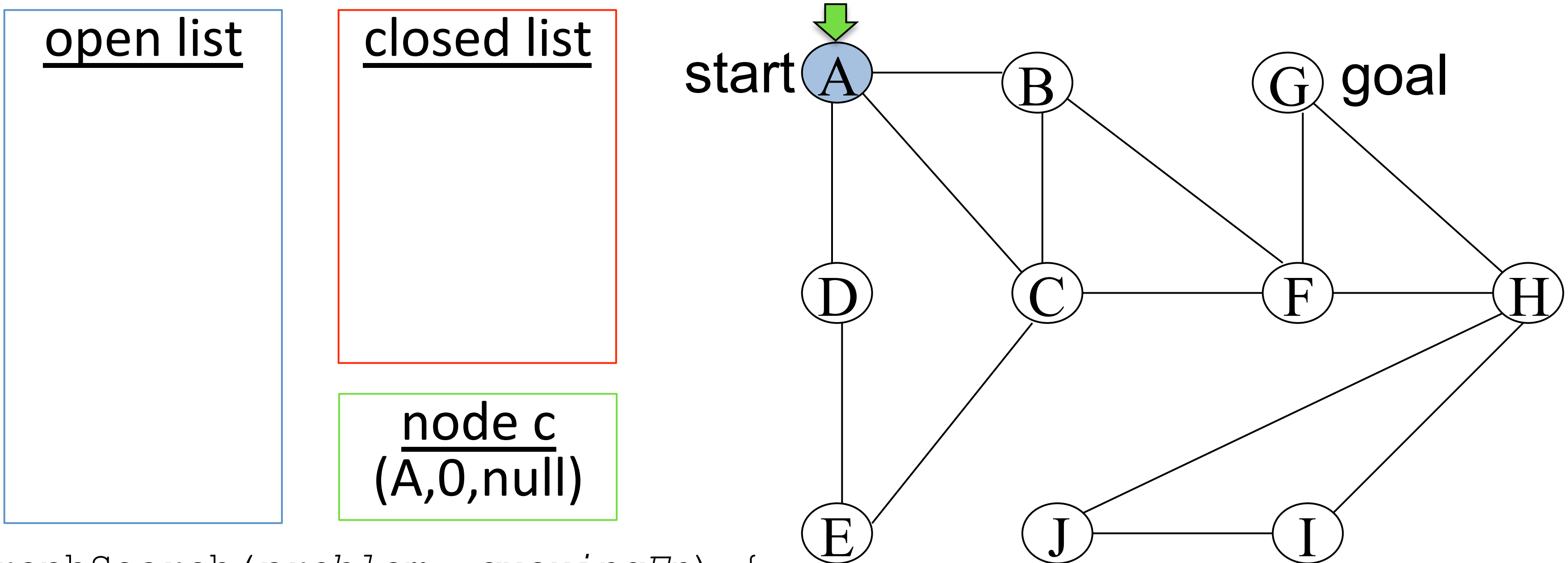
(A,0,null)

closed list




```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
}}}
```

Application: Route Finding (BFS)



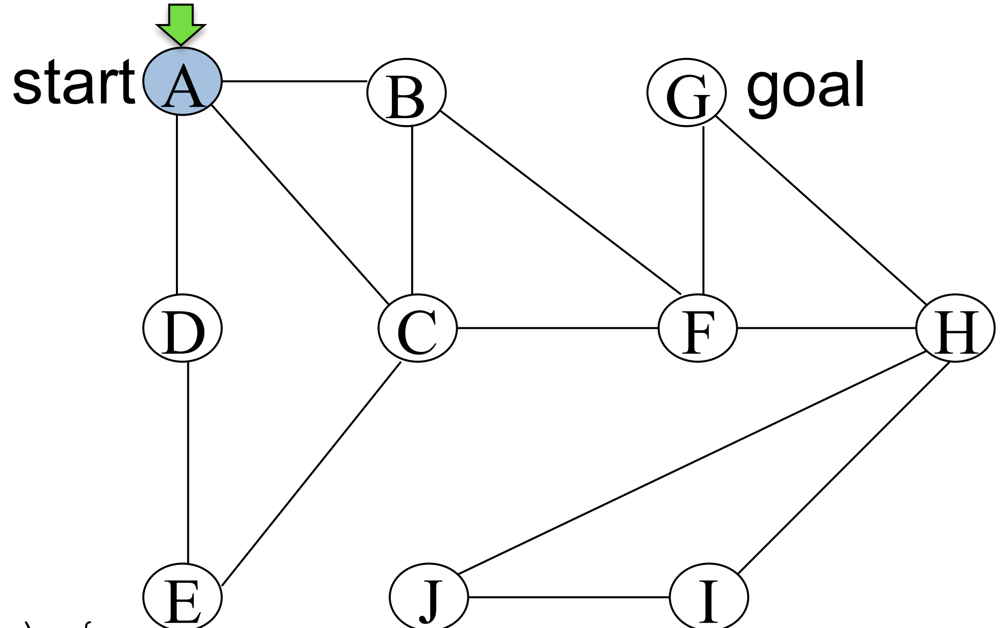

```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
}}}
```

Application: Route Finding (BFS)

open list

closed list

node c
(A,0,null)



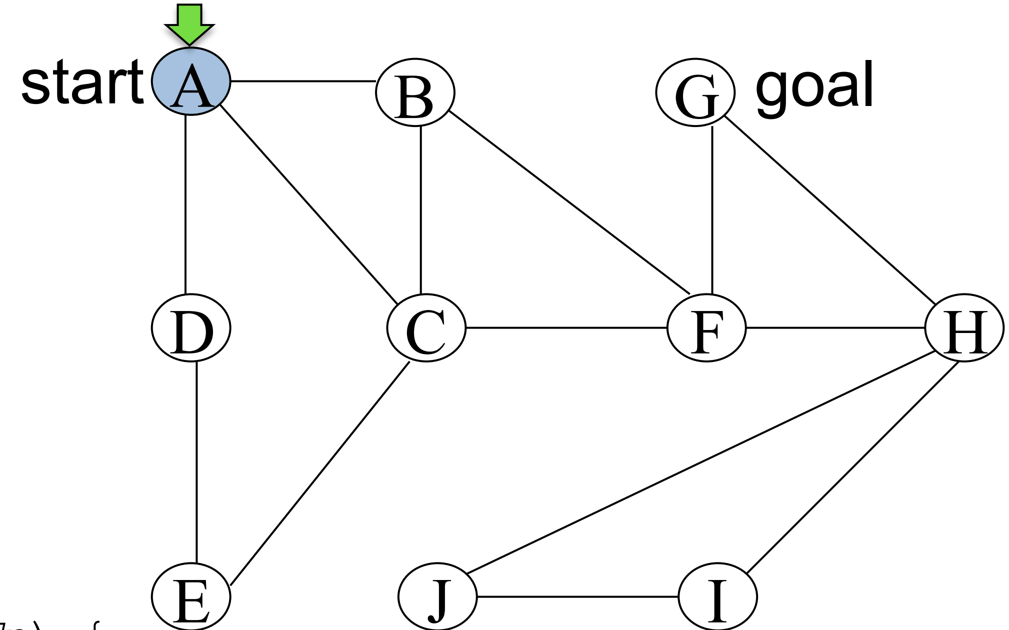

```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
}}}
```

Application: Route Finding (BFS)

open list

closed list

node c
(A,0,null)




```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
}}}
```

Application: Route Finding (BFS)

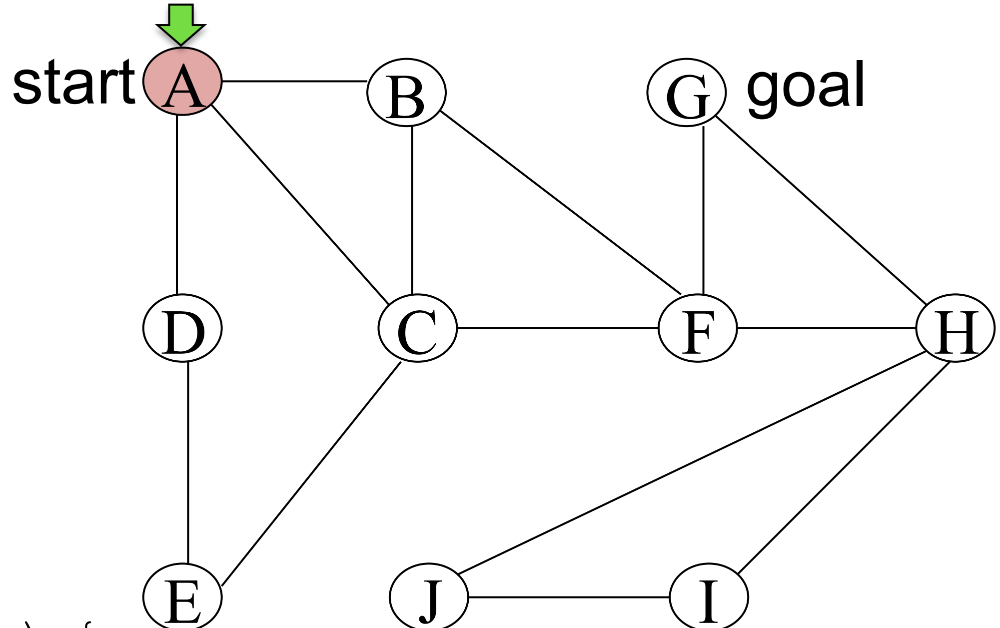
open list

closed list

A

node c

(A,0,null)



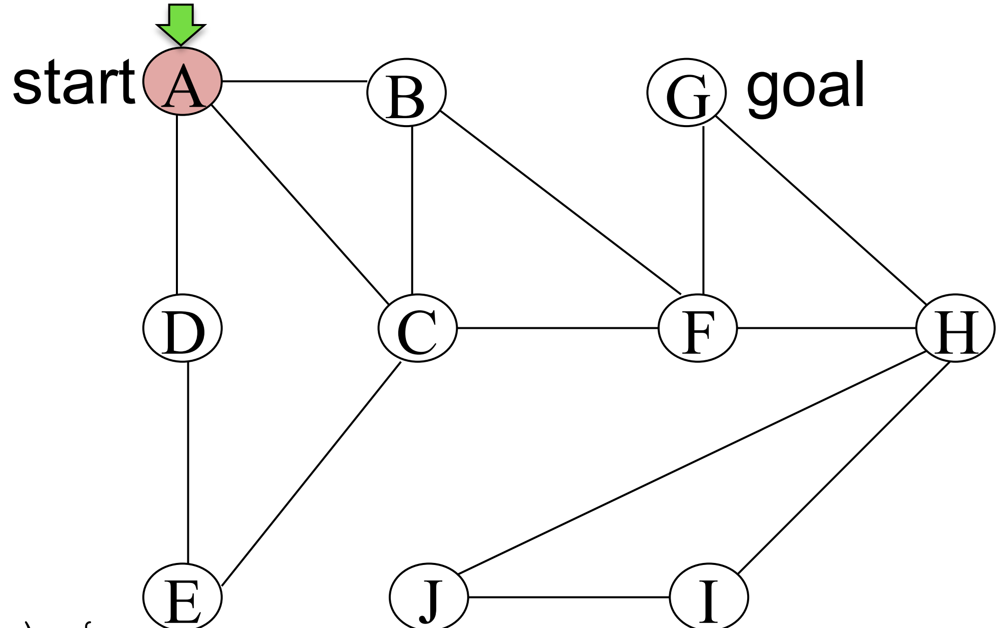

```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
}}}
```

Application: Route Finding (BFS)

open list

closed list
A

node c
(A,0,null)



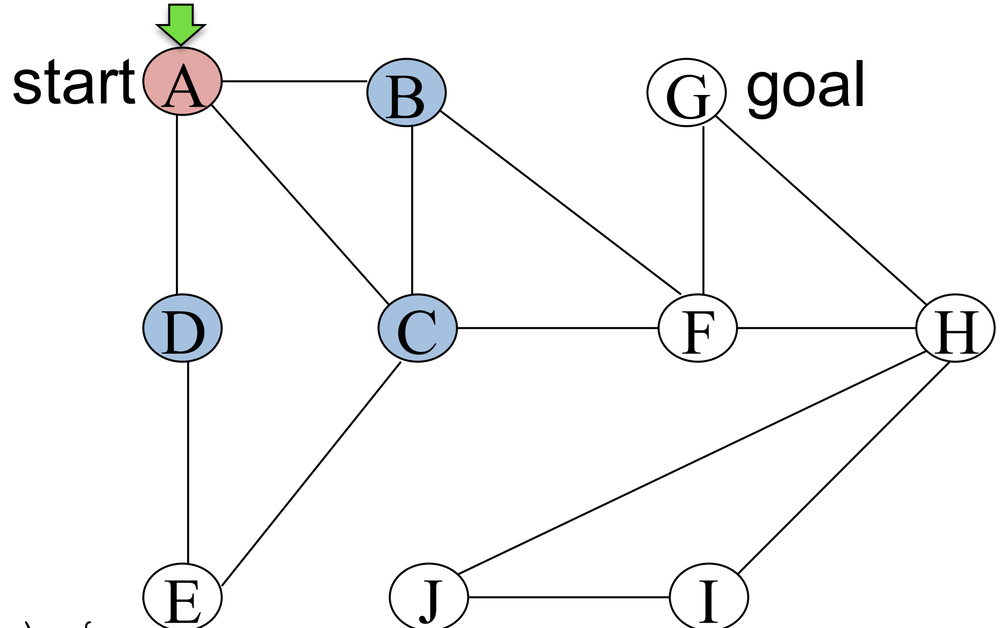

```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
}}}
```

Application: Route Finding (BFS)

open list
(B,1,A)
(C,1,A)
(D,1,A)

closed list
A

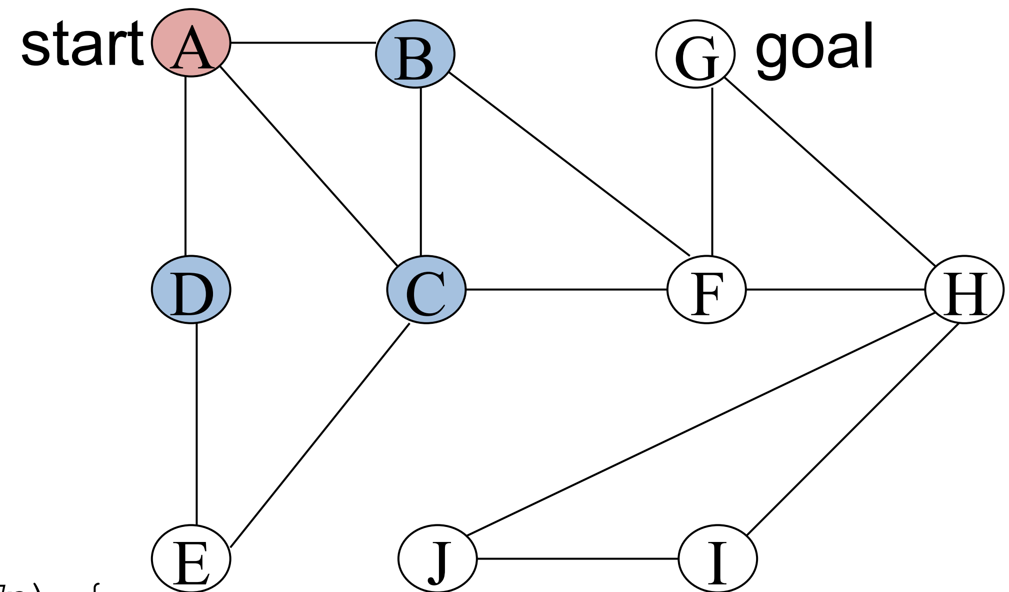
node c
(A,0,null)




```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
}}}
```

Application: Route Finding (BFS)

open list	closed list
(B,1,A)	A
(C,1,A)	
(D,1,A)	



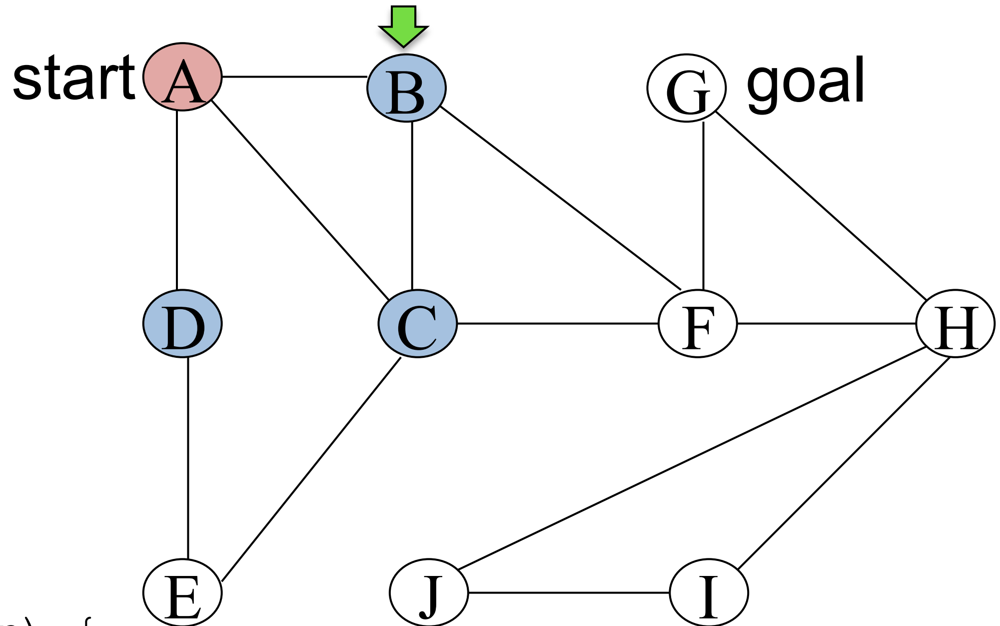

```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
}}}
```

Application: Route Finding (BFS)

open list
(C,1,A)
(D,1,A)

closed list
A

node c
(B,1,A)



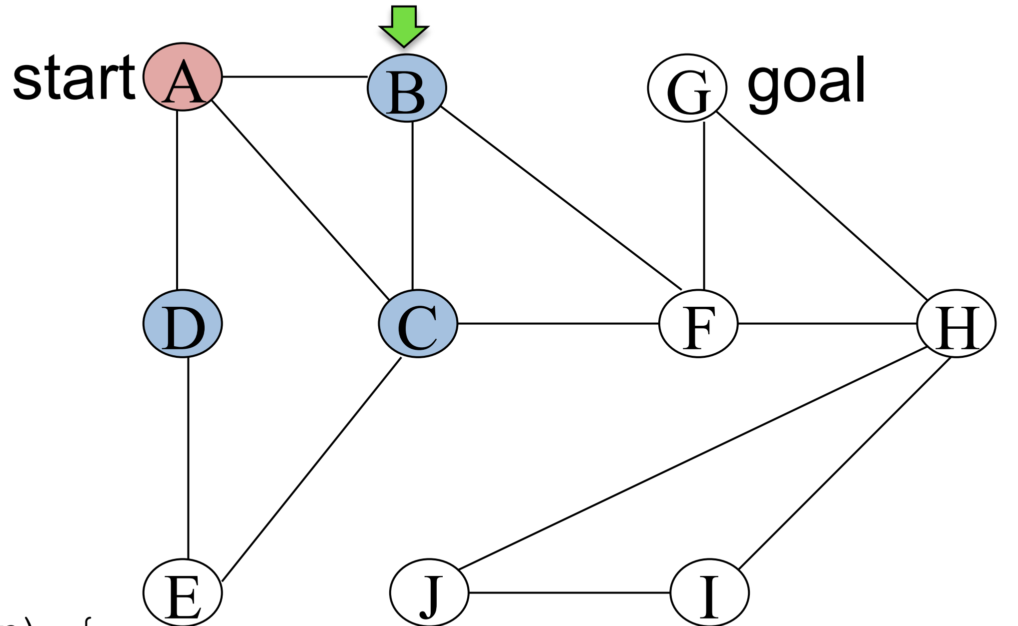

```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
}}}
```

Application: Route Finding (BFS)

open list
(C,1,A)
(D,1,A)

closed list
A

node c
(B,1,A)



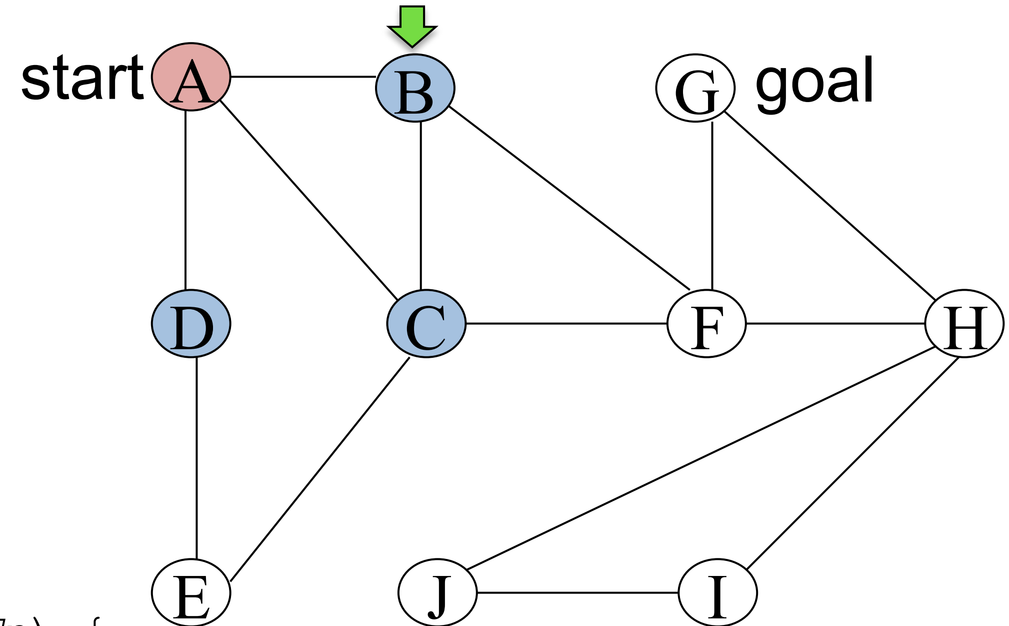

```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
    }}}
```

Application: Route Finding (BFS)

open list
(C,1,A)
(D,1,A)

closed list
A

node c
(B,1,A)



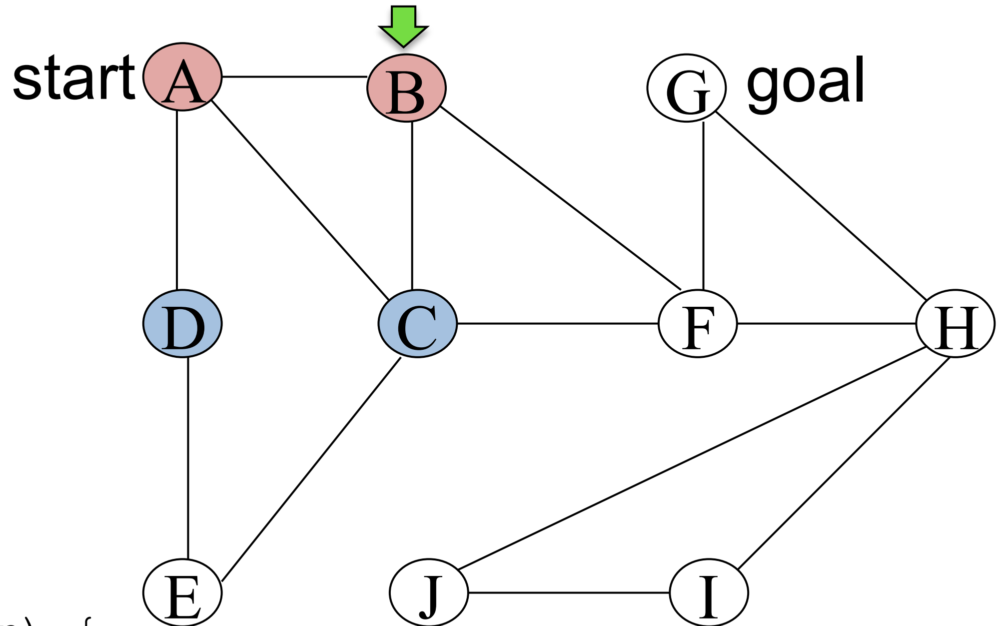

```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
}}}
```

Application: Route Finding (BFS)

open list
(C,1,A)
(D,1,A)

closed list
A
B

node c
(B,1,A)



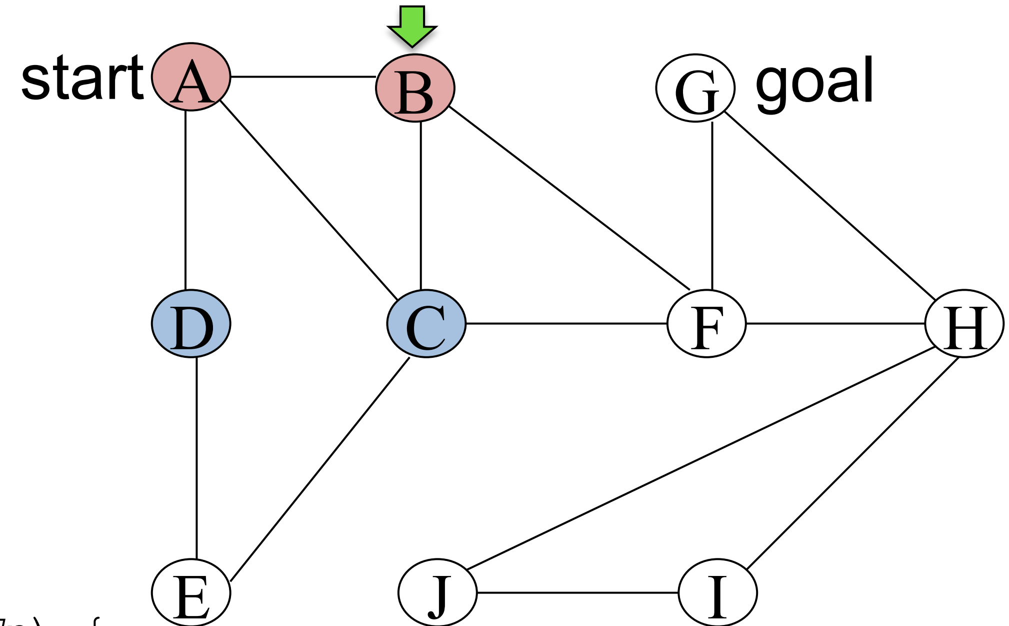

```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
}}}
```

Application: Route Finding (BFS)

open list
(C,1,A)
(D,1,A)

closed list
A
B

node c
(B,1,A)



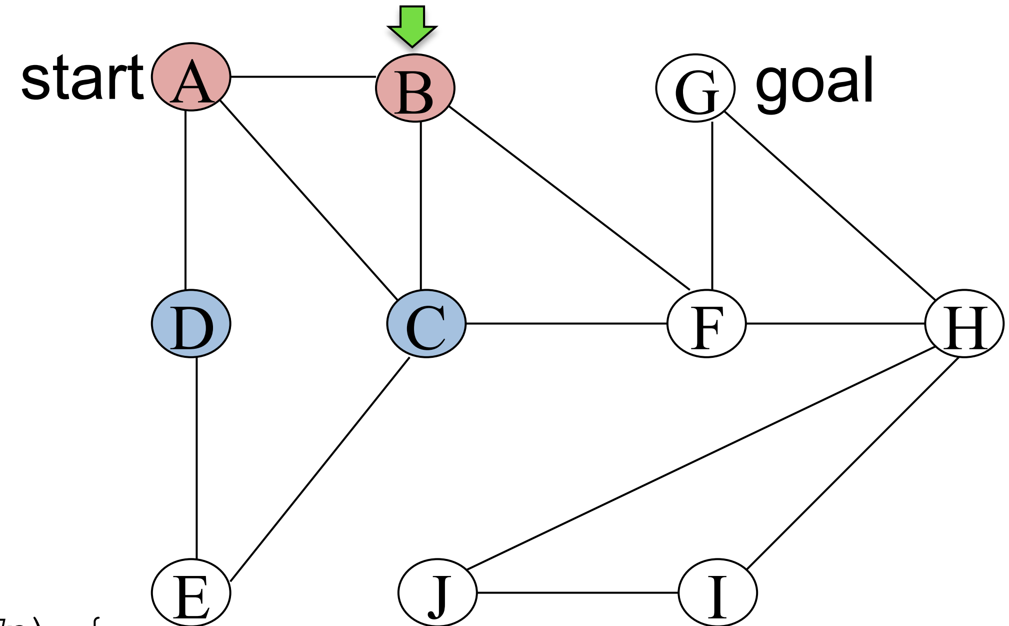

```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
}}}
```

Application: Route Finding (BFS)

open list
(C,1,A)
(D,1,A)

closed list
A
B

node c
(B,1,A)




```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(pro
  loop {
    if empty(open) then return
    c = removeFront(open)
    if problem.goalTest(c.vert
    if c.vertex is not in clos
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
}}}
```

Need to add (C,2,B) and (F,2,B) to open

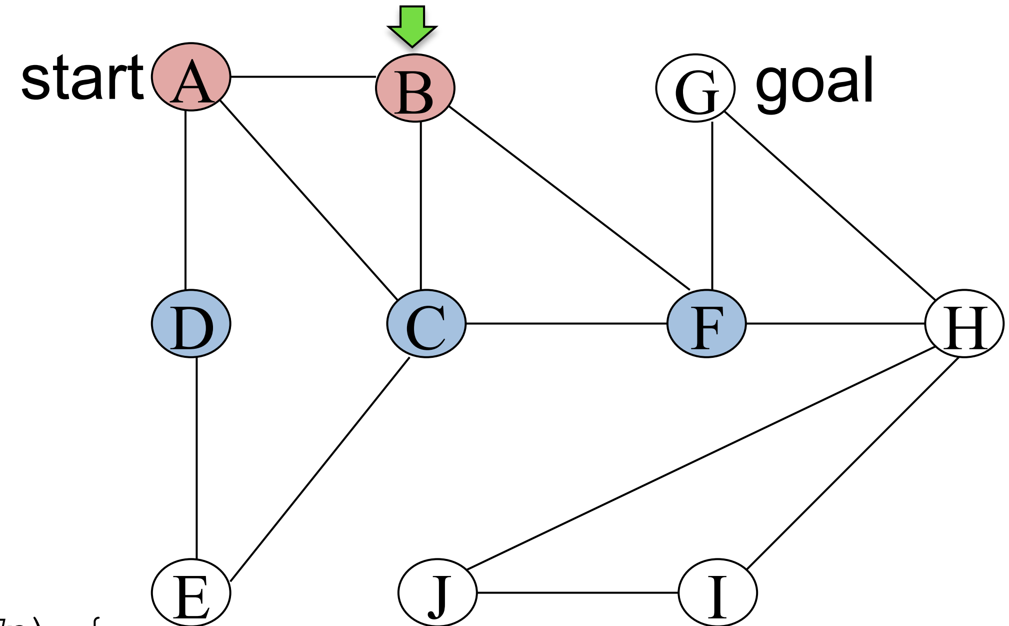
Since BFS expands the <u>shallowest</u> node, what must we insure about the open list?

What queuing function should we use?

Application: Route Finding (BFS)

open list	closed list
(C,1,A)	A
(D,1,A)	B
(C,2,B)	
(F,2,B)	

node c
(B,1,A)




```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
}}}
```

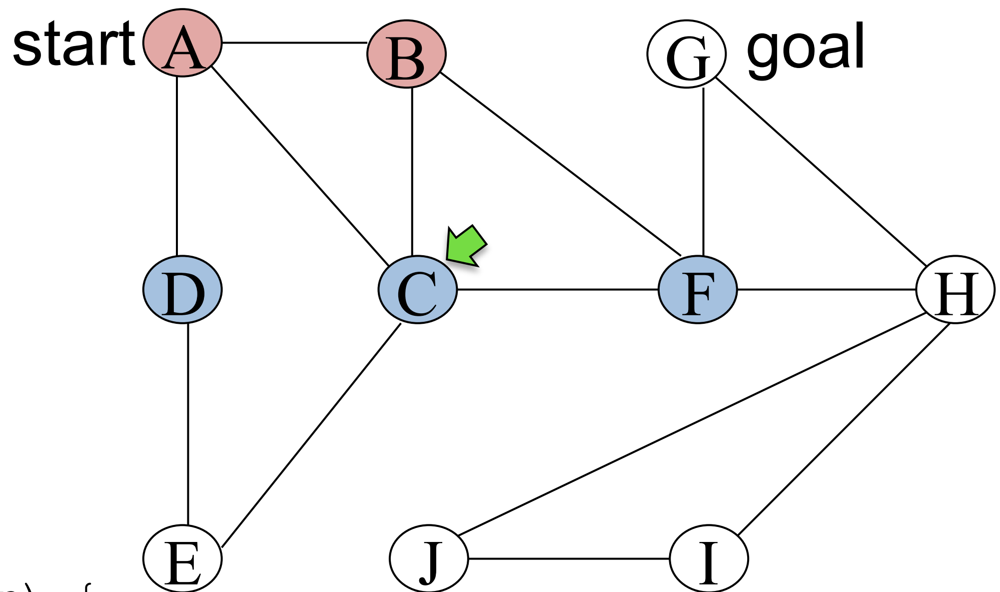
BFS uses a FIFO Queue!

Application: Route Finding (BFS)

open list
(D,1,A)
(C,2,B)
(F,2,B)

closed list
A
B

node c
(C,1,A)



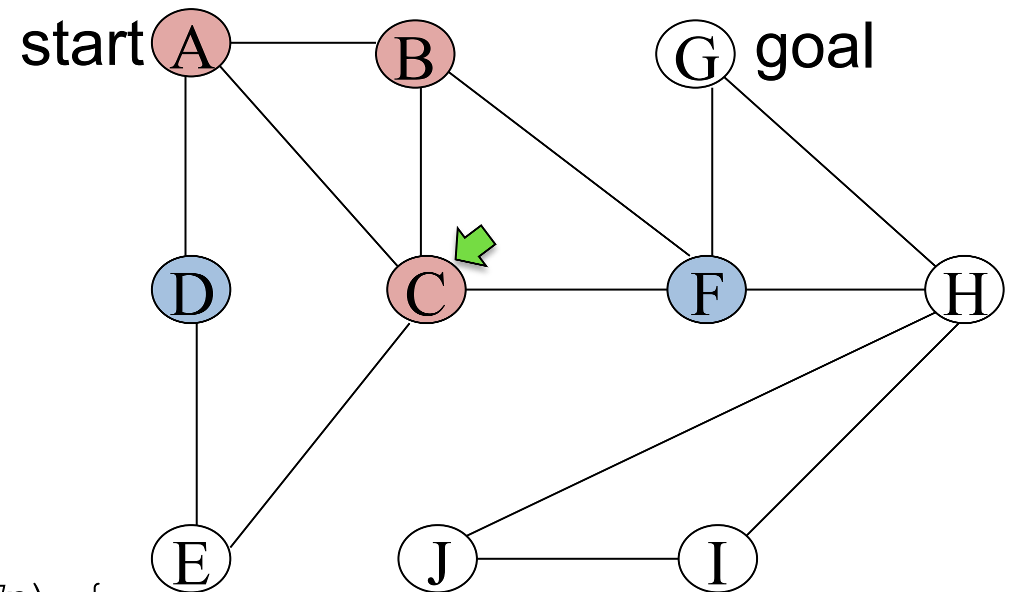

```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
    }}}
```

Application: Route Finding (BFS)

open list
(D,1,A)
(C,2,B)
(F,2,B)

closed list
A
B
C

node c
(C,1,A)

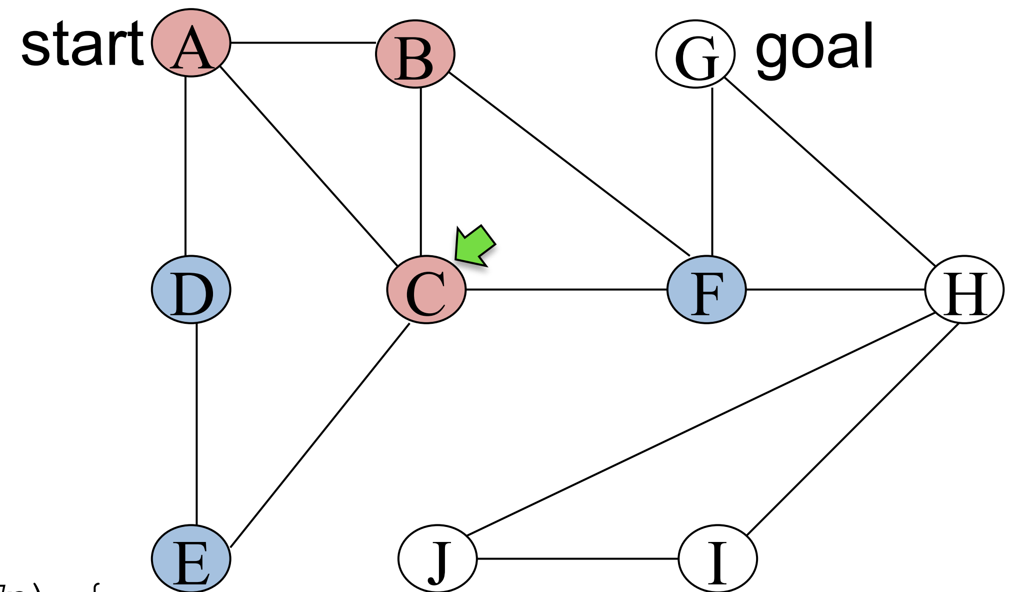



```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
}}}
```

Application: Route Finding (BFS)

open list	closed list
(D,1,A)	A
(C,2,B)	B
(F,2,B)	C
(F,2,C)	
(E,2,C)	

node c
(C,1,A)

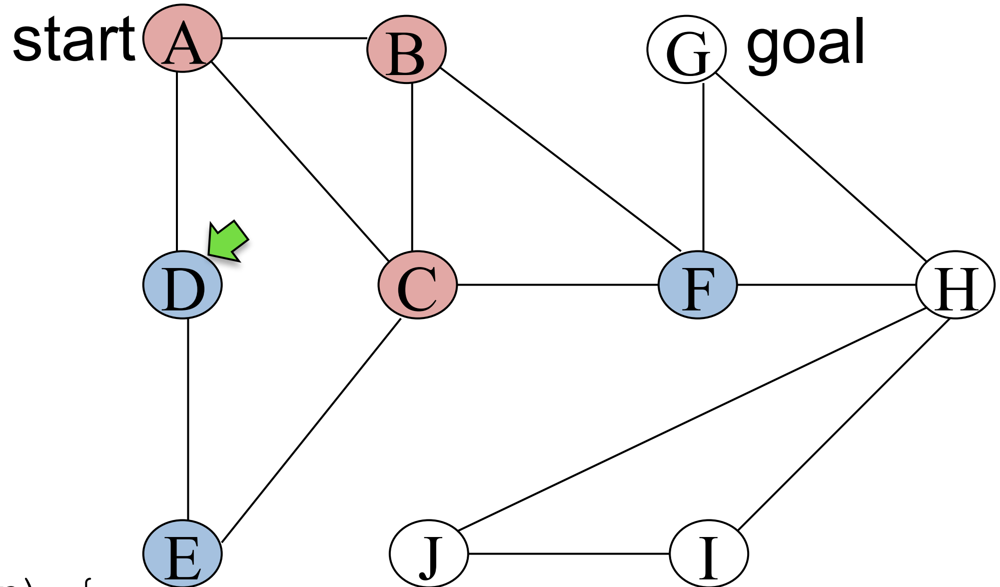



```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
}}}
```

Application: Route Finding (BFS)

open list	closed list
(C,2,B)	A
(F,2,B)	B
(F,2,C)	C
(E,2,C)	

node c
(D,1,A)

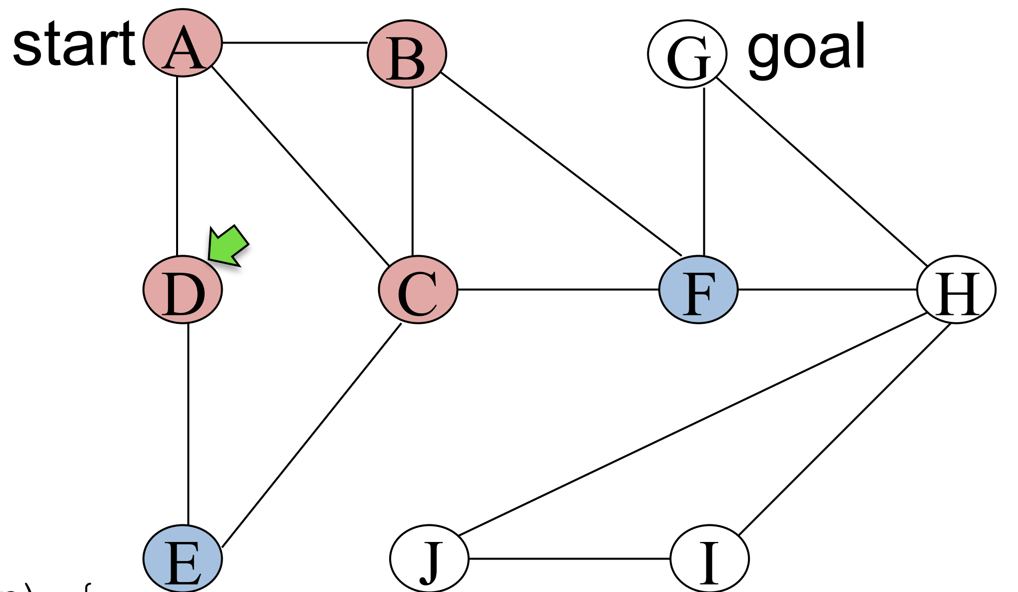



```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
}}}
```

Application: Route Finding (BFS)

open list	closed list
(C,2,B)	A
(F,2,B)	B
(F,2,C)	C
(E,2,C)	D

node c
(D,1,A)



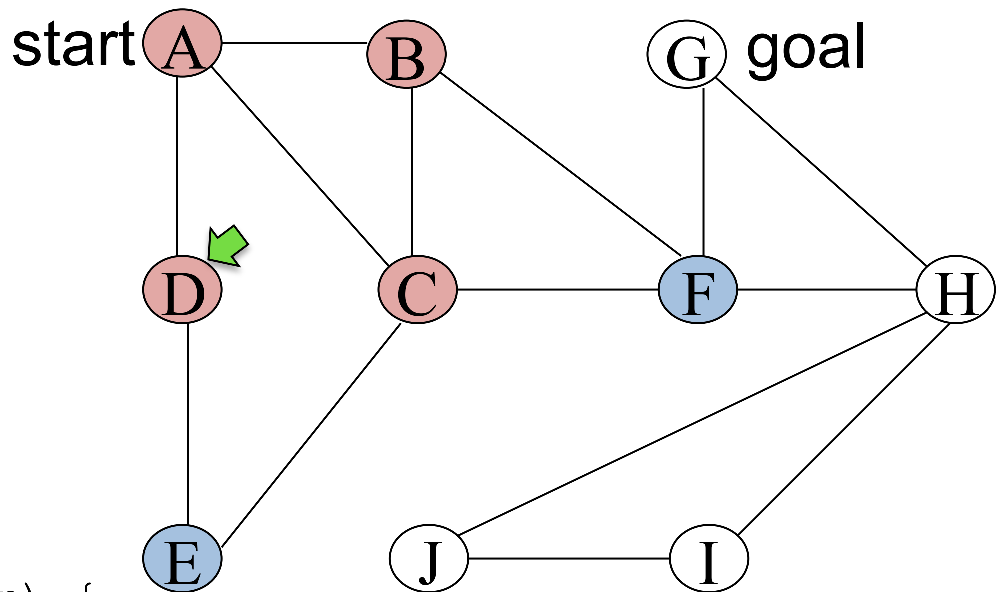

```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
}}}
```

Application: Route Finding (BFS)

open list
(C,2,B)
(F,2,B)
(F,2,C)
(E,2,C)
(E,2,D)

closed list
A
B
C
D

node c
(D,1,A)

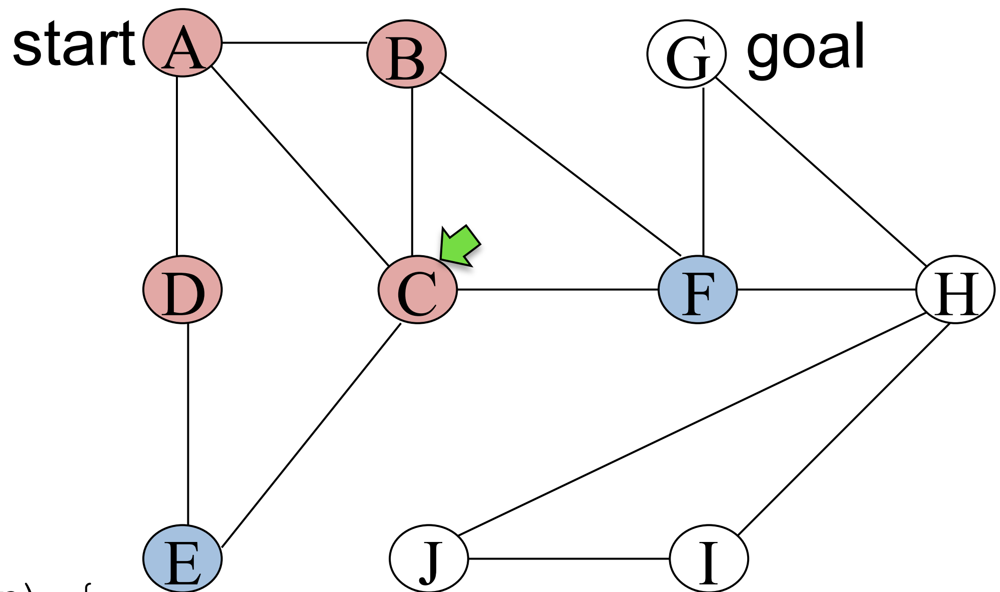



```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
}}}
```

Application: Route Finding (BFS)

open list	closed list
(F,2,B)	A
(F,2,C)	B
(E,2,C)	C
(E,2,D)	D

node c
(C,2,B)



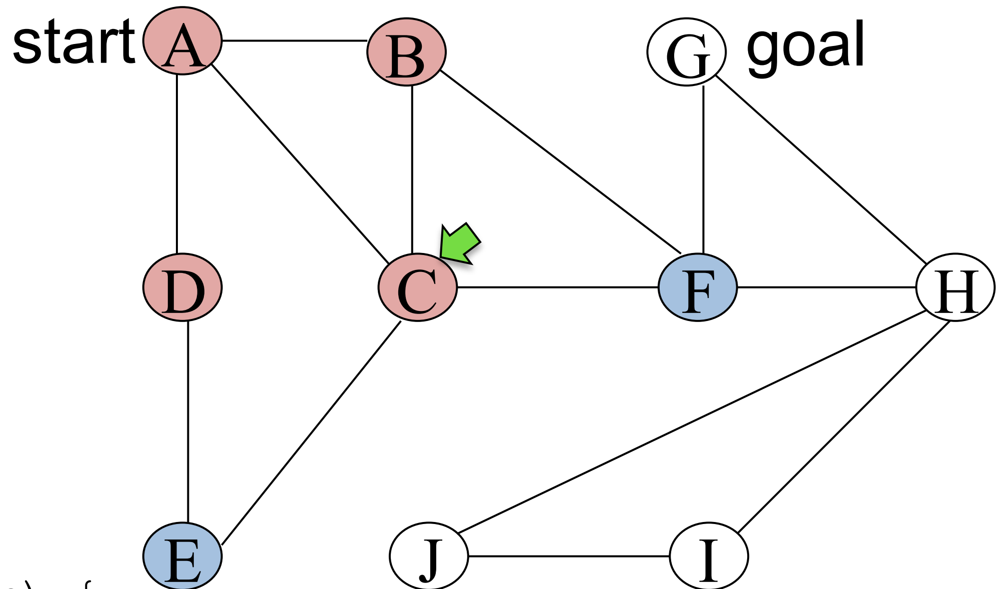

```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
    }}}
```

Application: Route Finding (BFS)

open list
(F,2,B)
(F,2,C)
(E,2,C)
(E,2,D)

closed list
A
B
C
D

node c
(C,2,B)



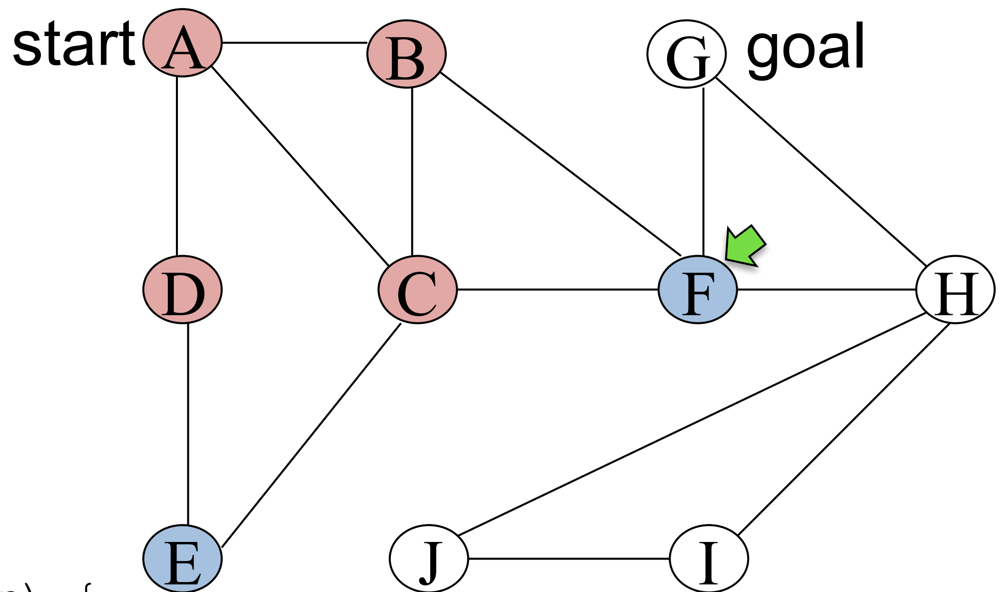

```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
}}}
```

Application: Route Finding (BFS)

open list
(F,2,C)
(E,2,C)
(E,2,D)

closed list
A
B
C
D

node c
(F,2,B)



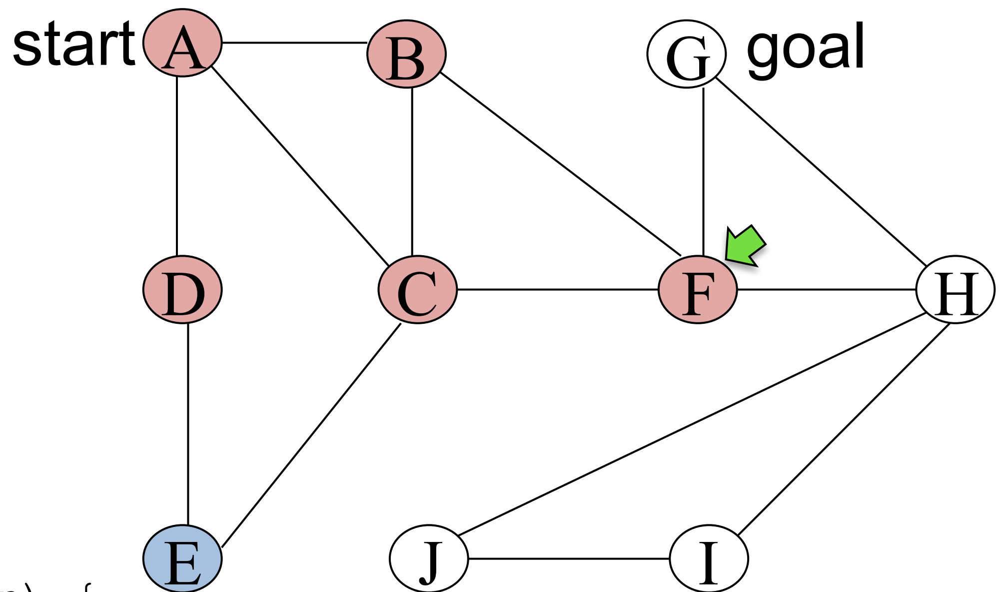

```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
}}}
```

Application: Route Finding (BFS)

open list
(F,2,C)
(E,2,C)
(E,2,D)

closed list
A
B
C
D
F

node c
(F,2,B)



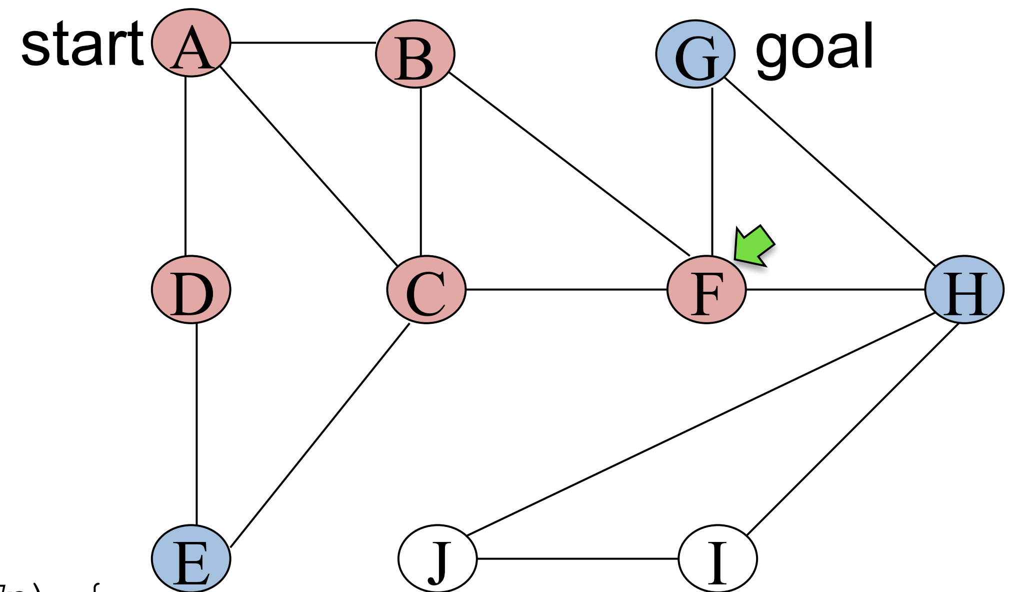

```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
}}}
```

Application: Route Finding (BFS)

open list
(F,2,C)
(E,2,C)
(E,2,D)
(G,3,F)
(H,3,F)

closed list
A
B
C
D
F

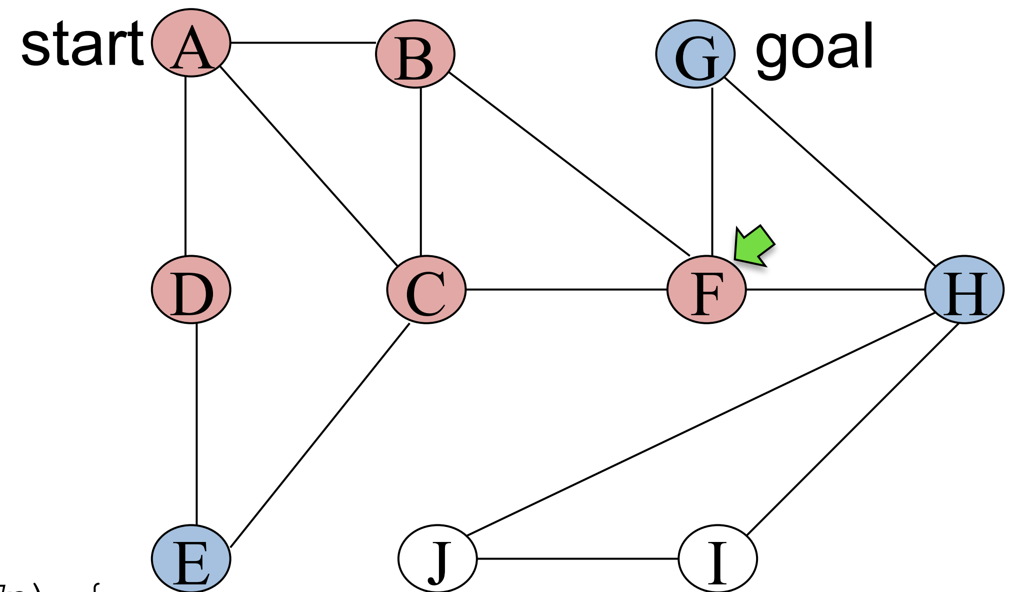
node c
(F,2,B)




```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
    }}}
```

Application: Route Finding (BFS)

open list	closed list	node c
(E,2,C)	A	(F,2,C)
(E,2,D)	B	
(G,3,F)	C	
(H,3,F)	D	
	F	

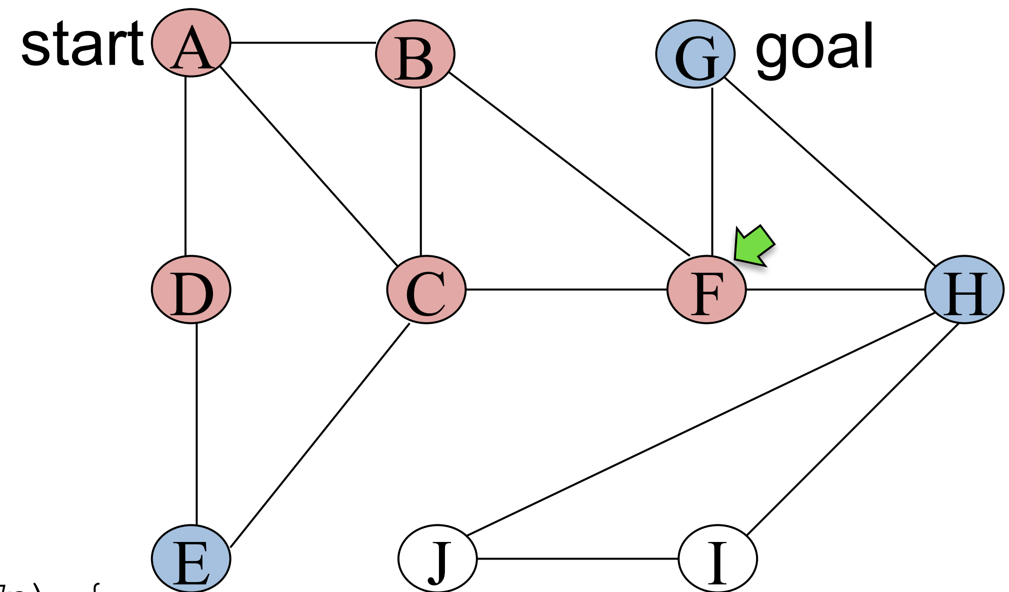



```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
    }}}
```

Application: Route Finding (BFS)

open list	closed list
(E,2,C)	A
(E,2,D)	B
(G,3,F)	C
(H,3,F)	D
	F

node c
(F,2,C)



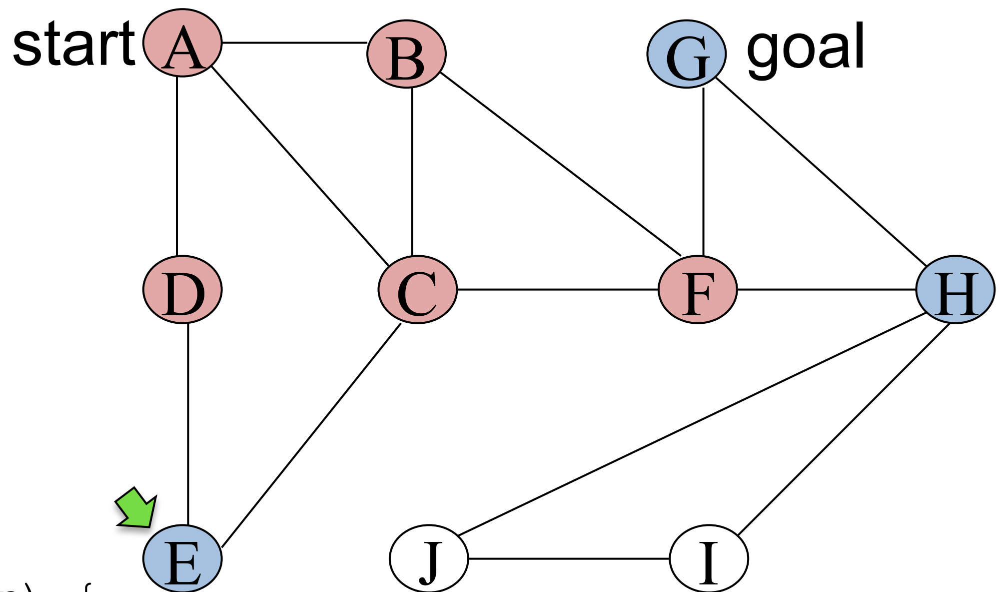

```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
}}}
```

Application: Route Finding (BFS)

open list
(E,2,D)
(G,3,F)
(H,3,F)

closed list
A
B
C
D
F

node c
(E,2,C)

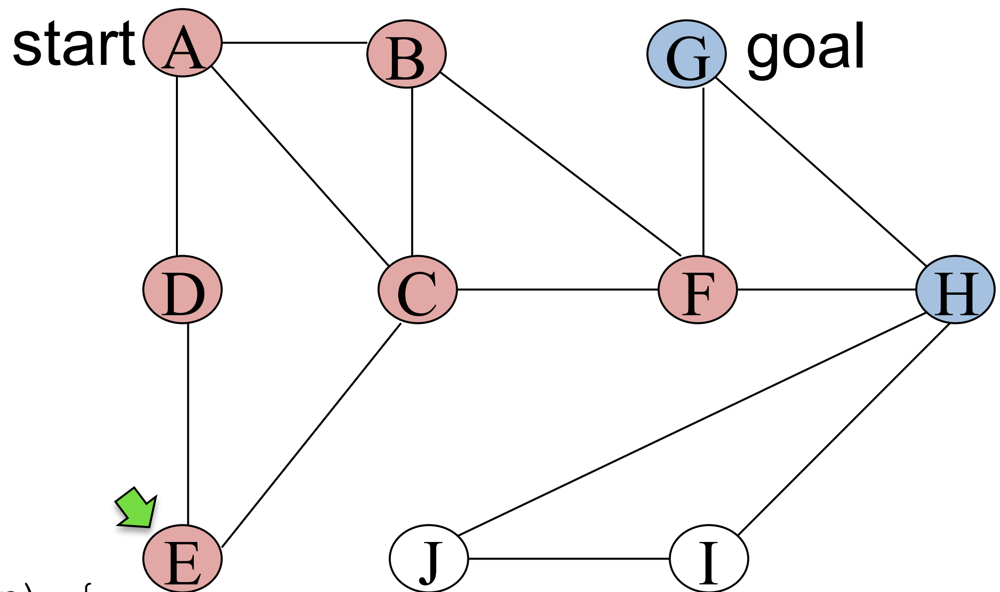



```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
}}}
```

Application: Route Finding (BFS)

open list	closed list
(E,2,D)	A
(G,3,F)	B
(H,3,F)	C
	D
	E
	F

node c
(E,2,C)



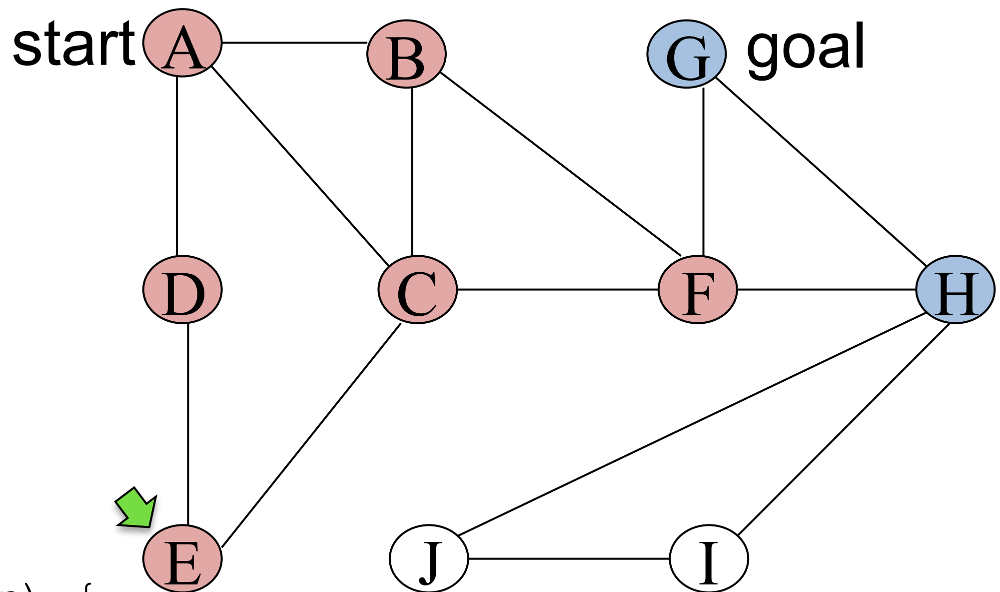

```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
}}}
```

Application: Route Finding (BFS)

open list
(G,3,F)
(H,3,F)

closed list
A
B
C
D
E
F

node c
(E,2,D)



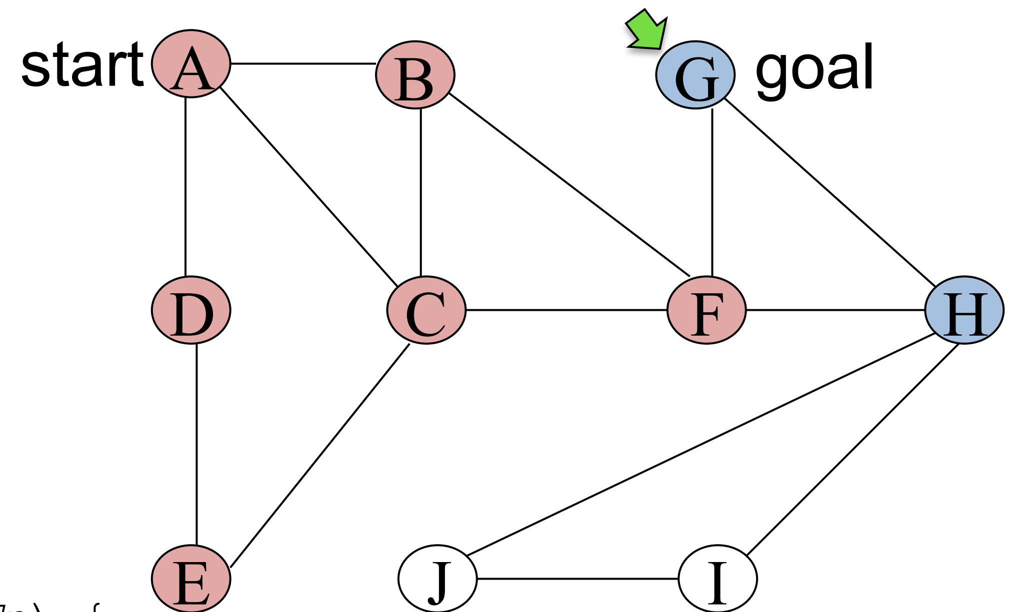

```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
}}}
```

Application: Route Finding (BFS)

open list
(H,3,F)

closed list
A
B
C
D
E
F

node c
(G,3,F)



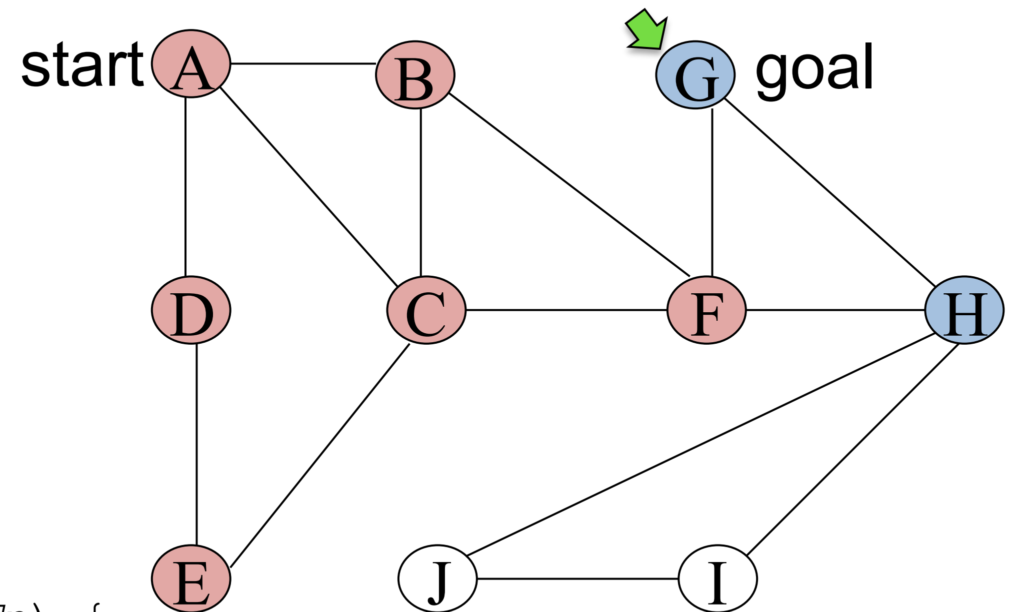

```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
}}}
```

Application: Route Finding (BFS)

open list
(H,3,F)

closed list
A
B
C
D
E
F

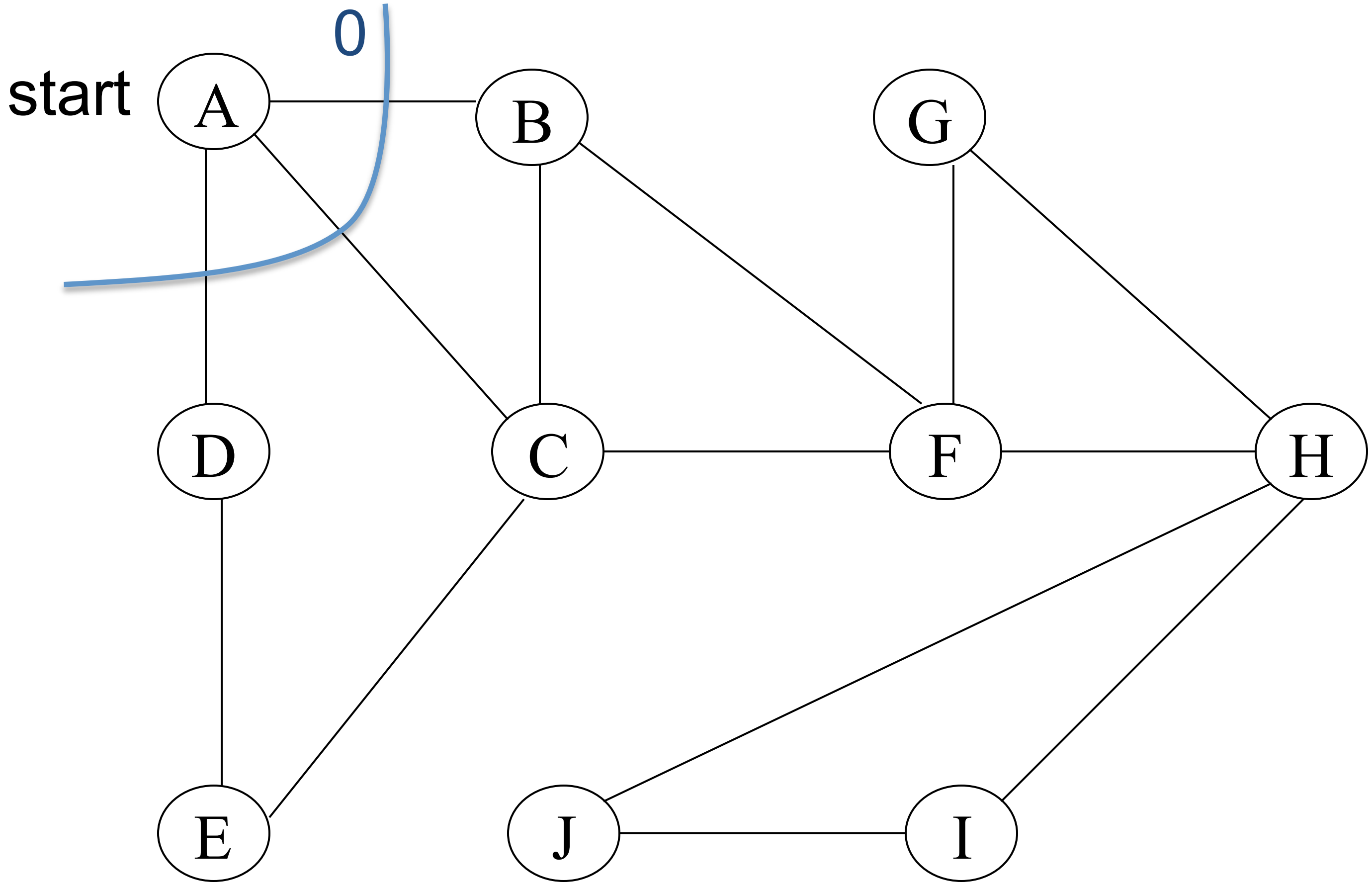
node c
(G,3,F)



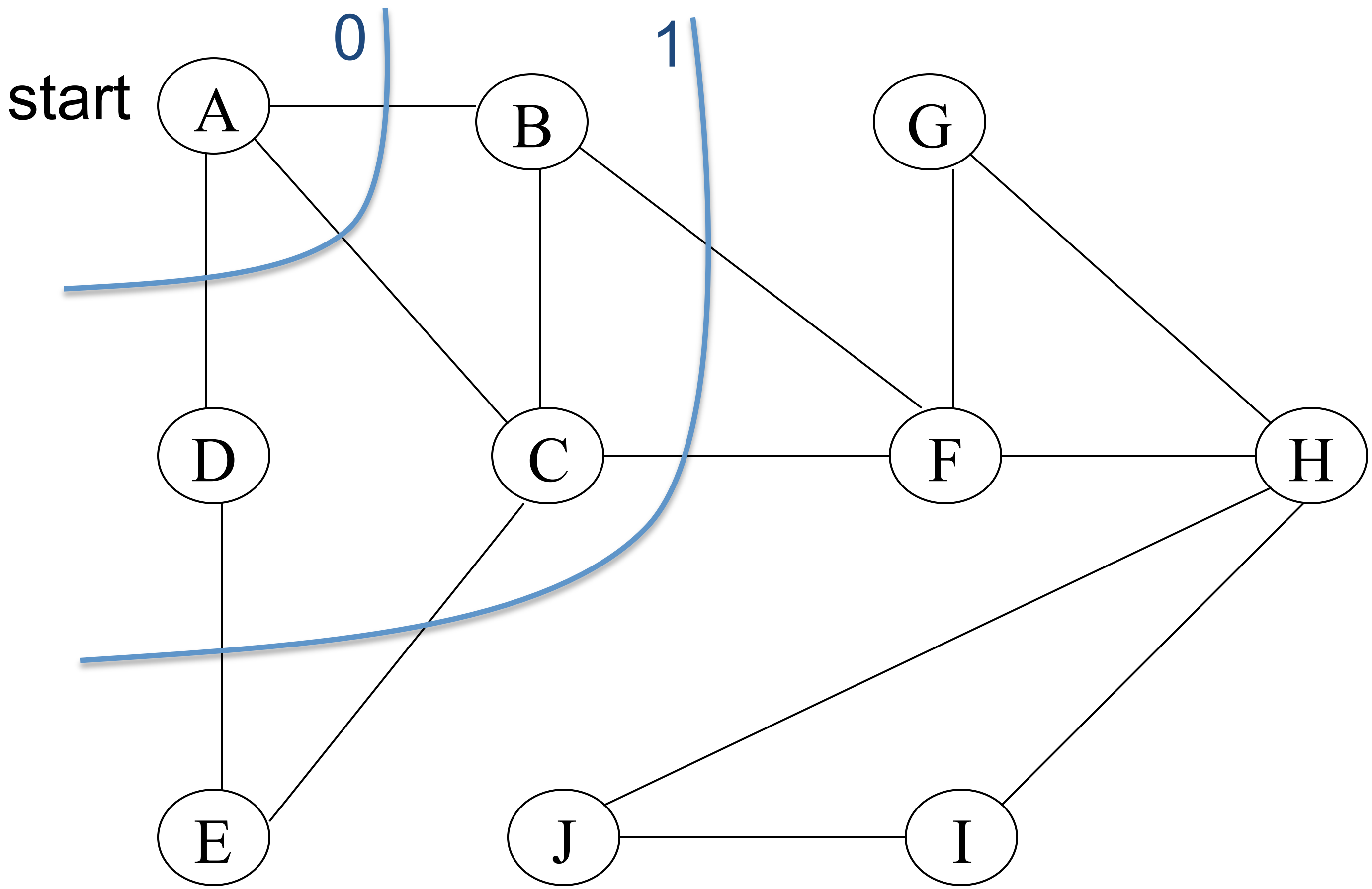

```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
}}}
```

Goal!

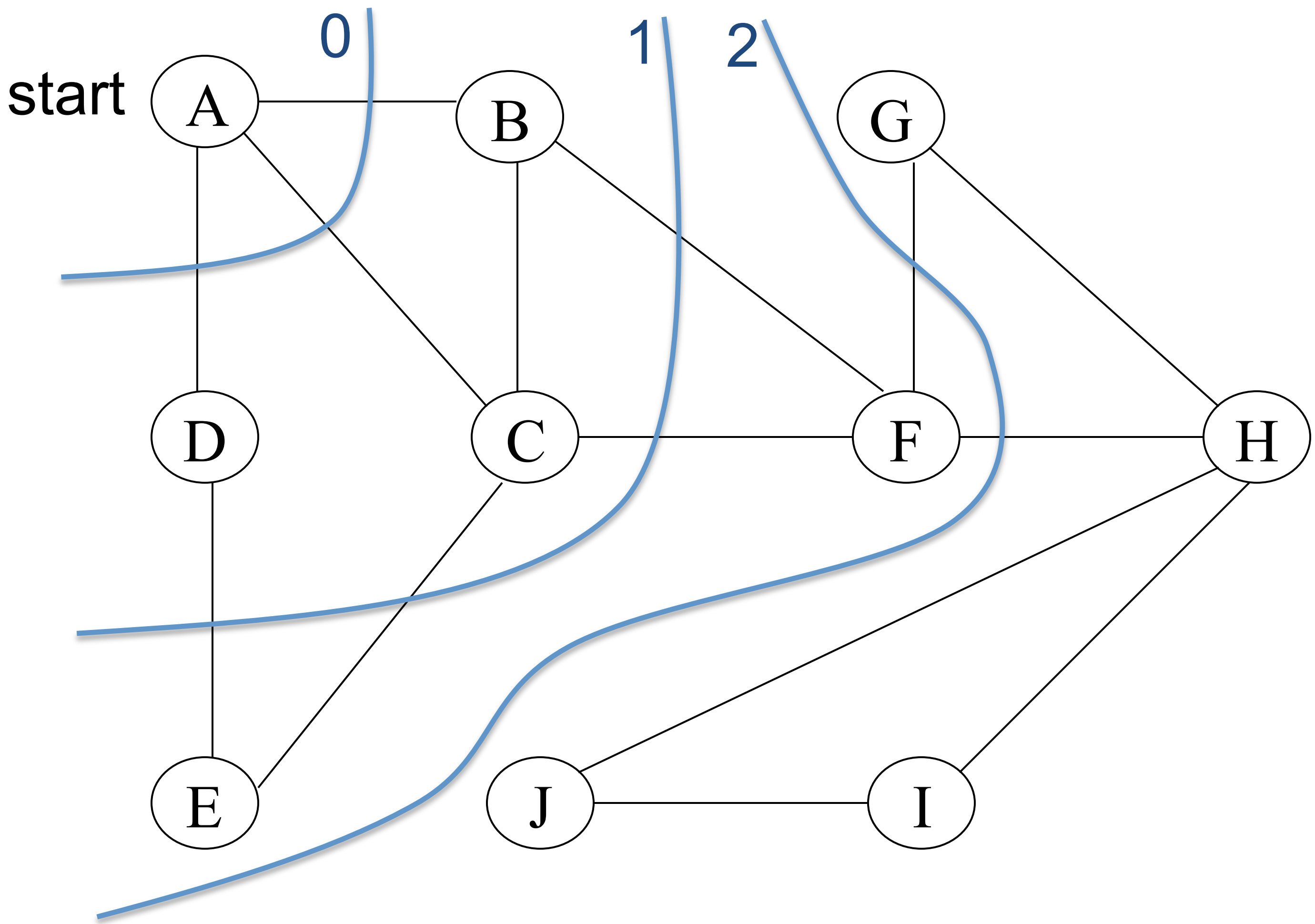
Breadth-First Search



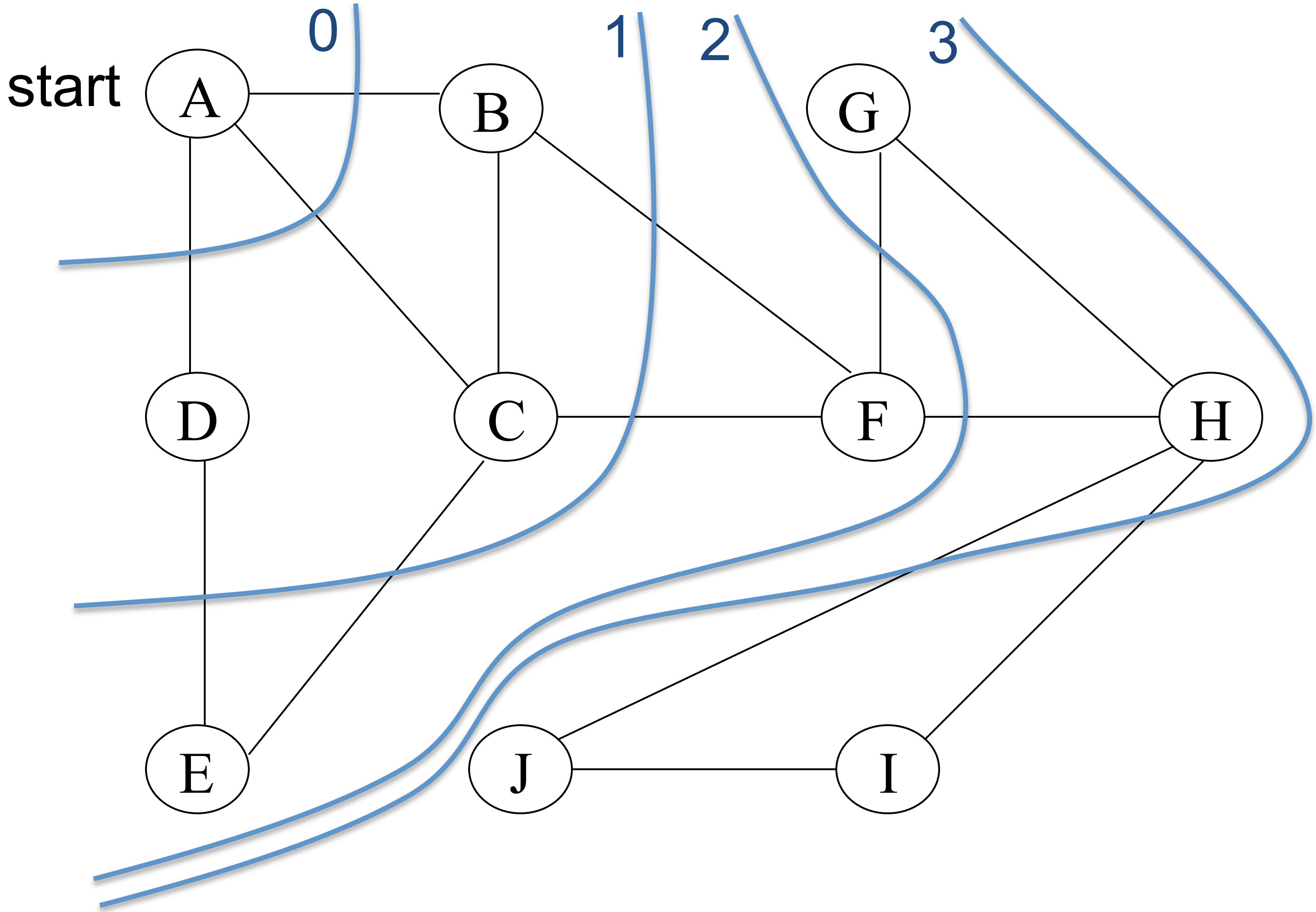
Breadth-First Search



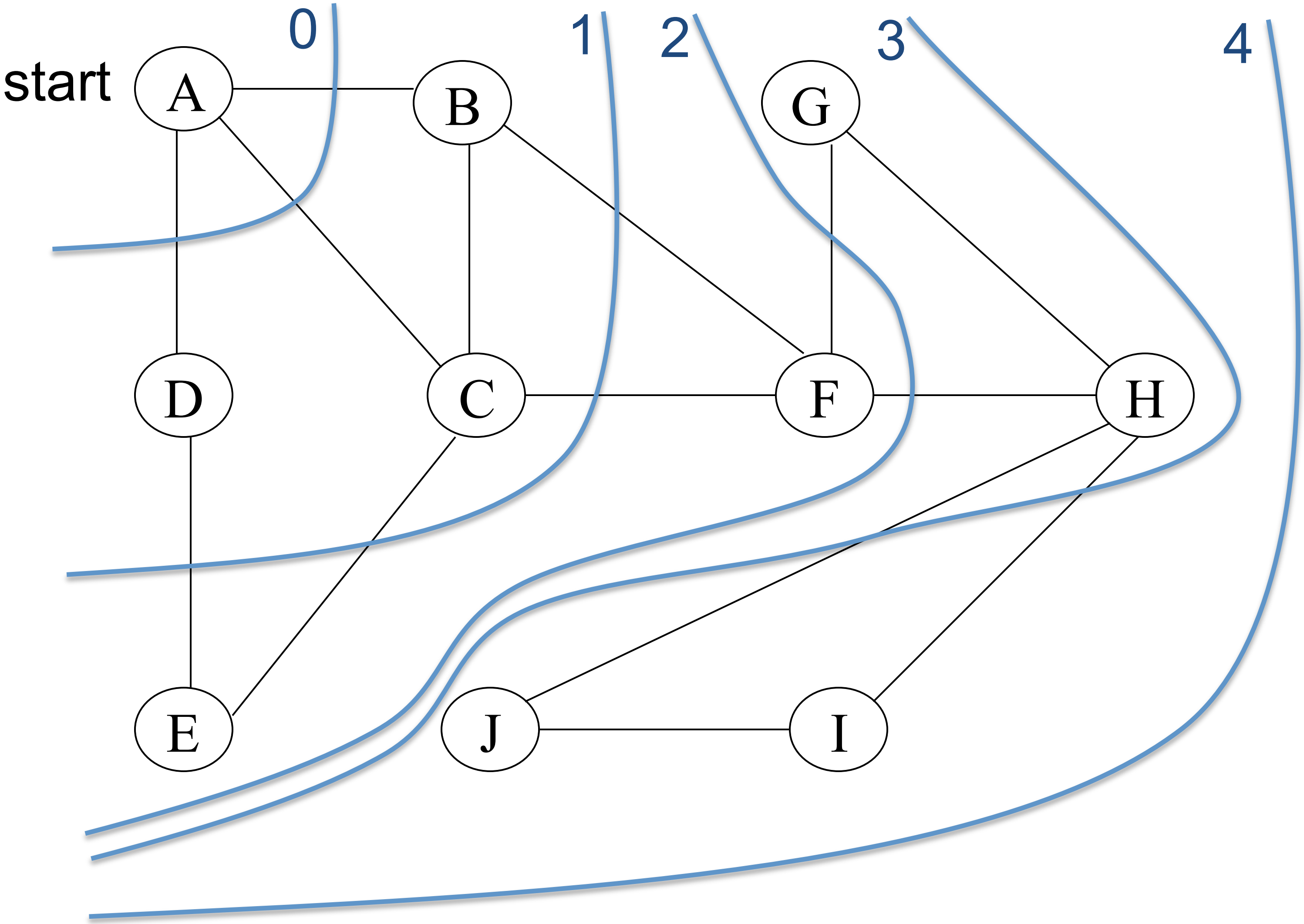
Breadth-First Search



Breadth-First Search



Breadth-First Search



Depth-First Search

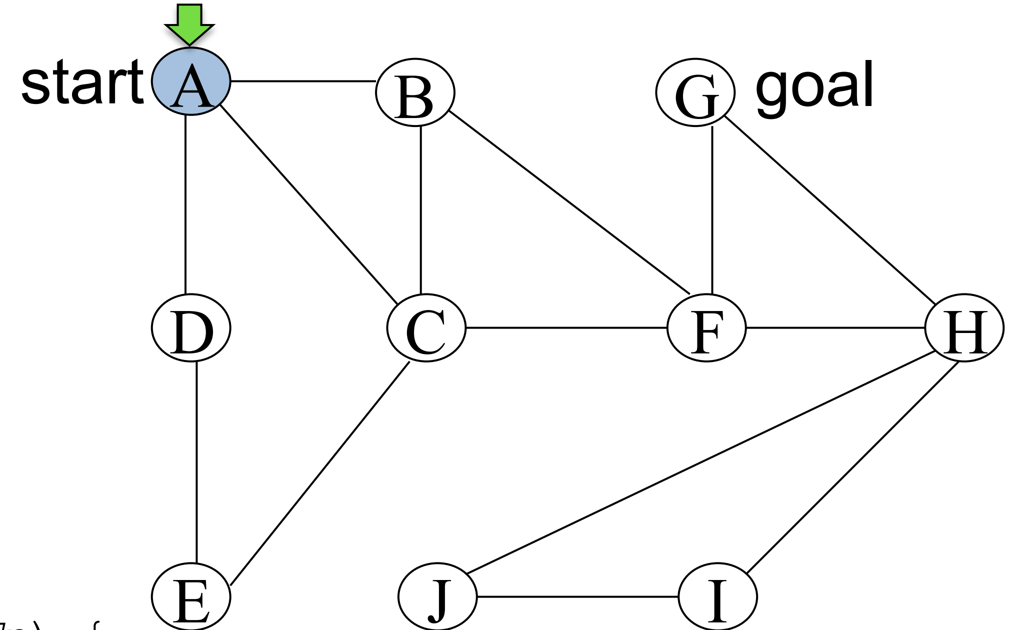
Expands the “deepest” vertex

Application: Route Finding (DFS)

open list

closed list

node c
(A,0,null)



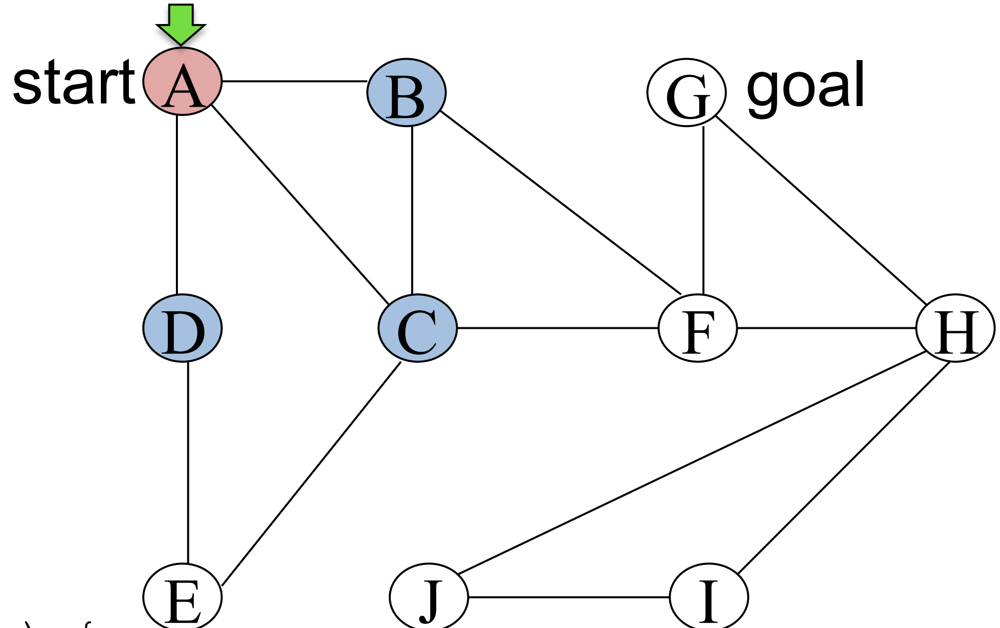

```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
    }}}
```

Application: Route Finding (DFS)

open list
(B,1,A)
(C,1,A)
(D,1,A)

closed list
A

node c
(A,0,null)



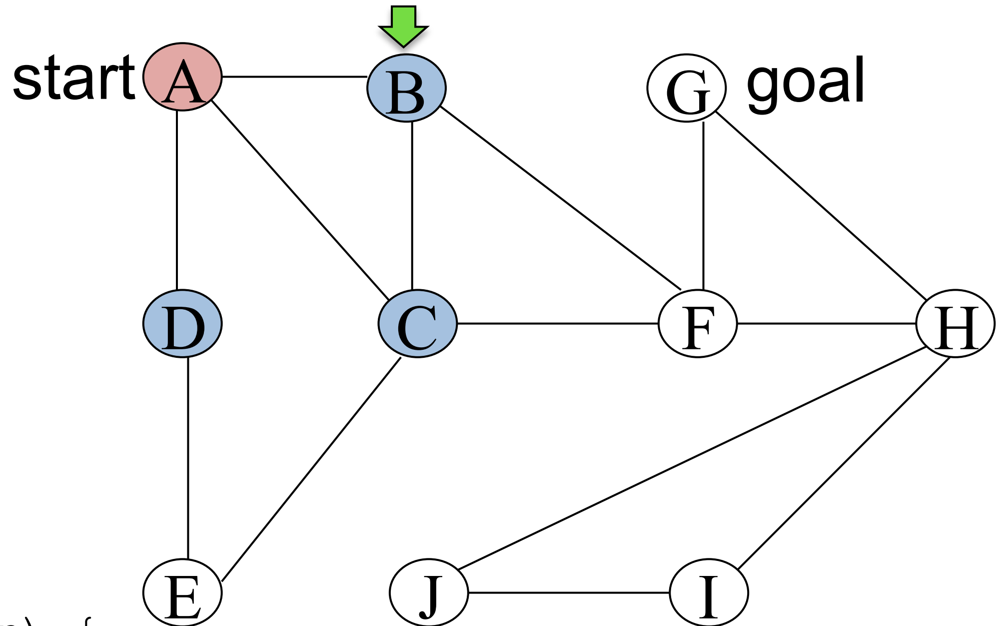

```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
}}}
```

Application: Route Finding (DFS)

open list
(C,1,A)
(D,1,A)

closed list
A

node c
(B,1,A)



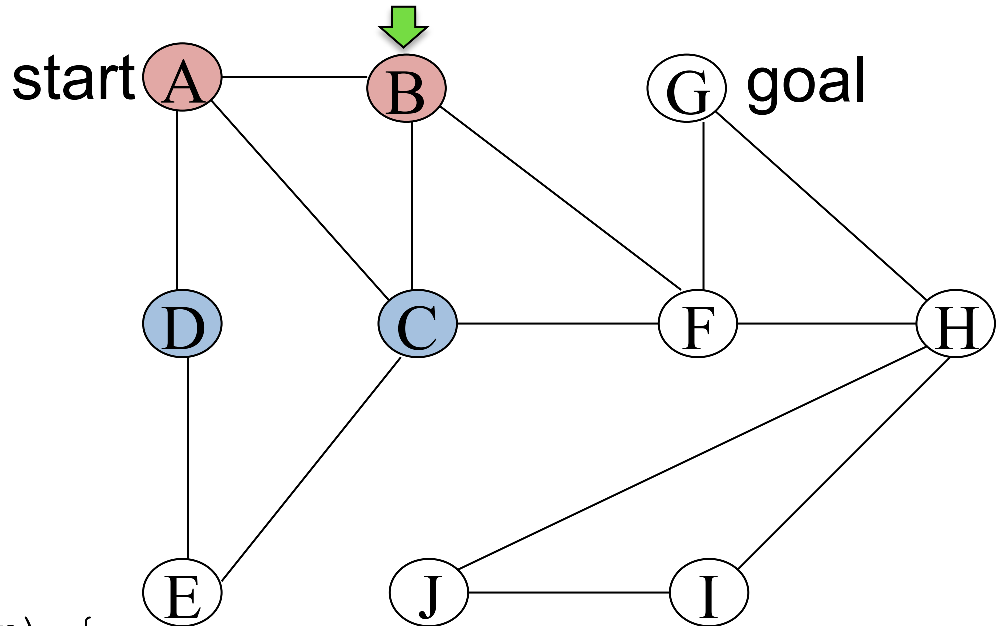

```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
}}}
```

Application: Route Finding (DFS)

open list
(C,1,A)
(D,1,A)

closed list
A
B

node c
(B,1,A)



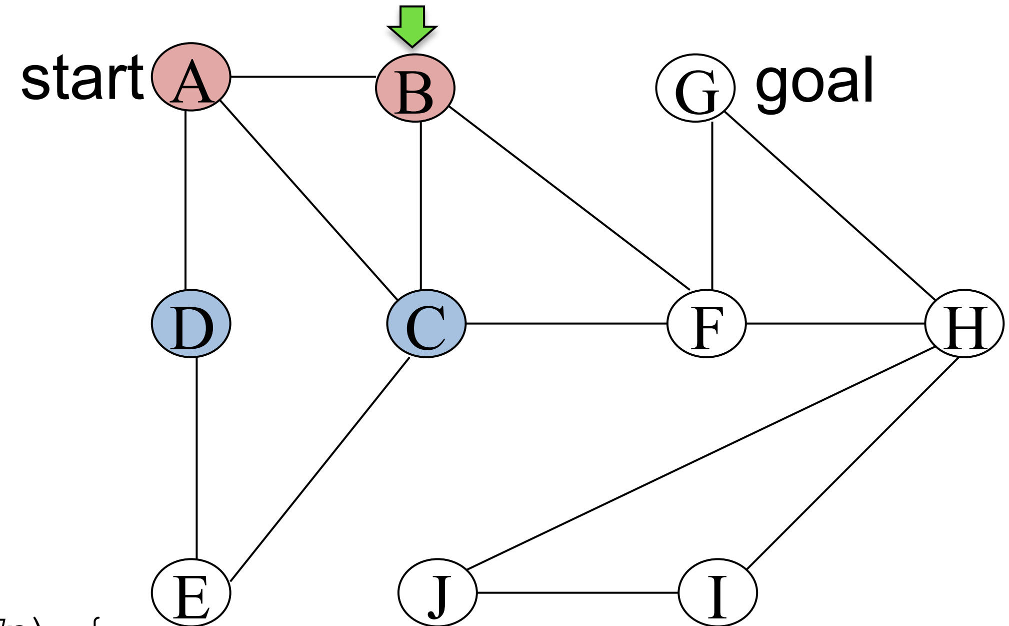

```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
}}}
```

Application: Route Finding (DFS)

open list
(C,1,A)
(D,1,A)

closed list
A
B

node c
(B,1,A)




```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(pro
  loop {
    if empty(open) then return
    c = removeFront(open)
    if problem.goalTest(c.vert
    if c.vertex is not in clos
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
}}}
```

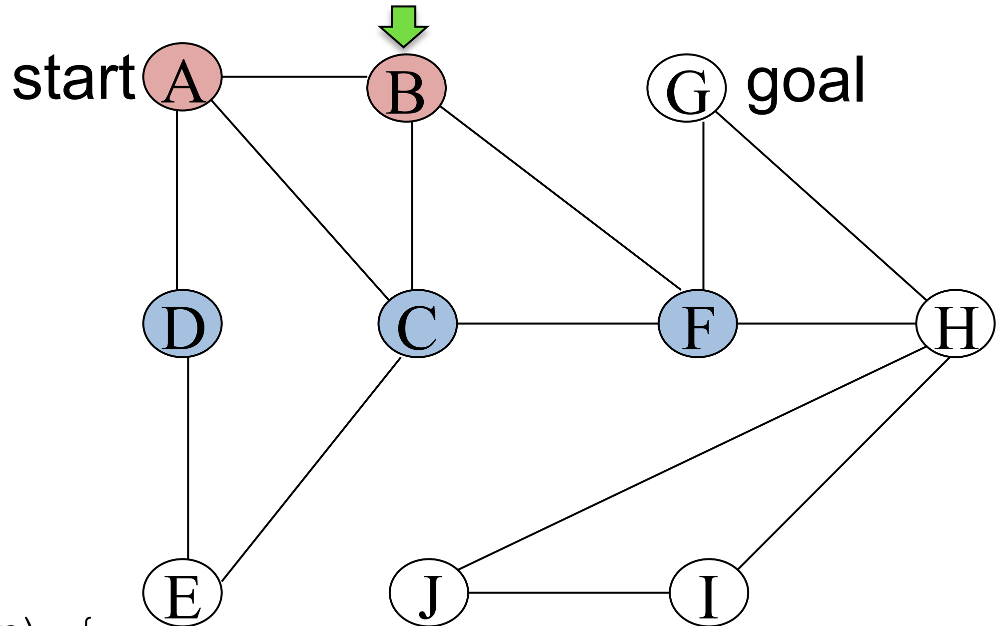
Need to add (C,2,B) and (F,2,B) to open

Since DFS expands the deepest node, what must we insure about the open list?

What queuing function should we use?

Application: Route Finding (DFS)

open list	closed list	node c
(F,2,B)	A	(B,1,A)
(C,2,B)	B	
(C,1,A)		
(D,1,A)		




```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
  }}}
```

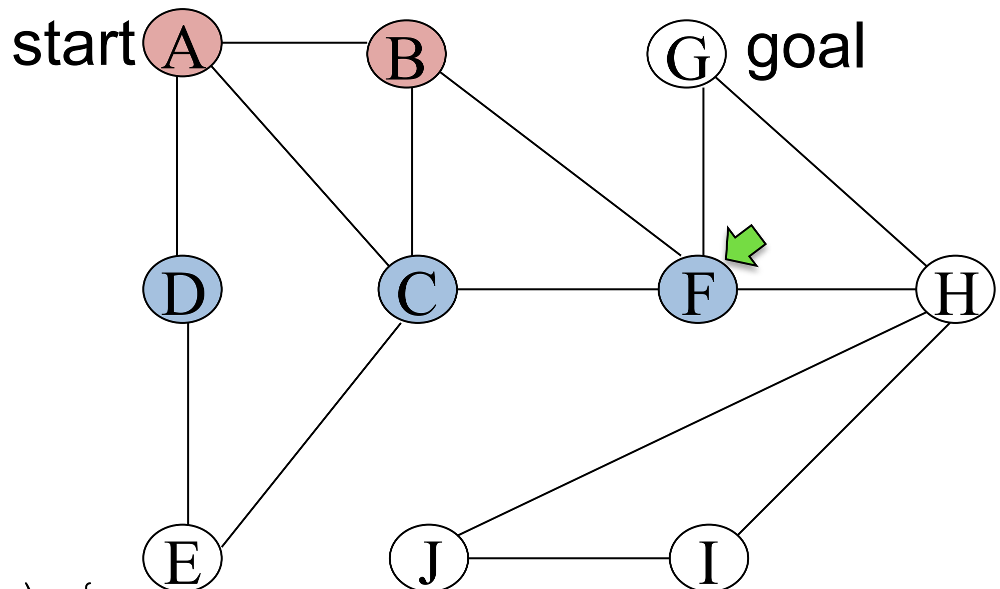
DFS uses a LIFO Stack!

Application: Route Finding (DFS)

open list
(C,2,B)
(C,1,A)
(D,1,A)

closed list
A
B

node c
(F,2,B)



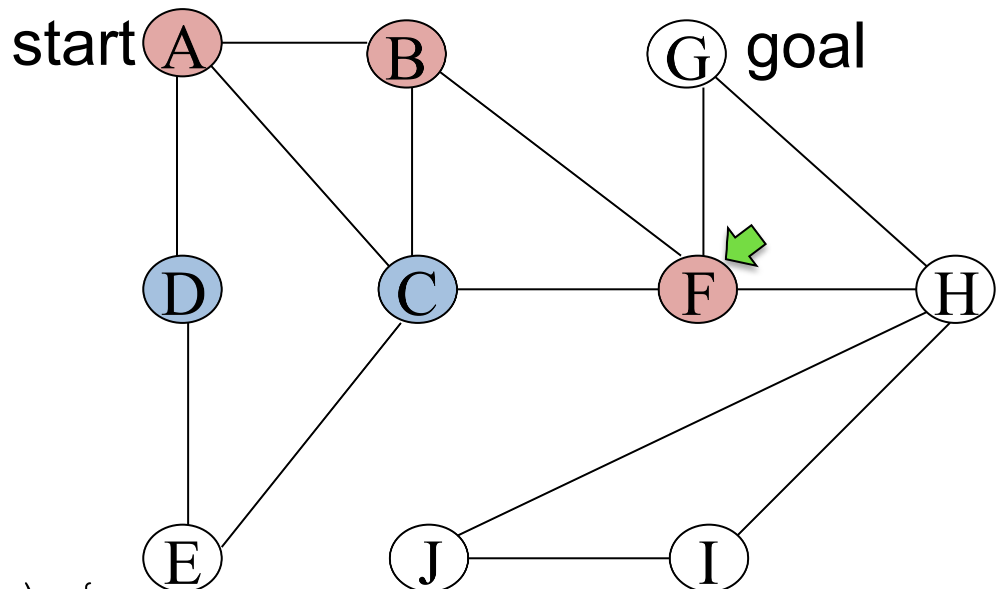

```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
}}}
```

Application: Route Finding (DFS)

open list
(C,2,B)
(C,1,A)
(D,1,A)

closed list
A
B
F

node c
(F,2,B)

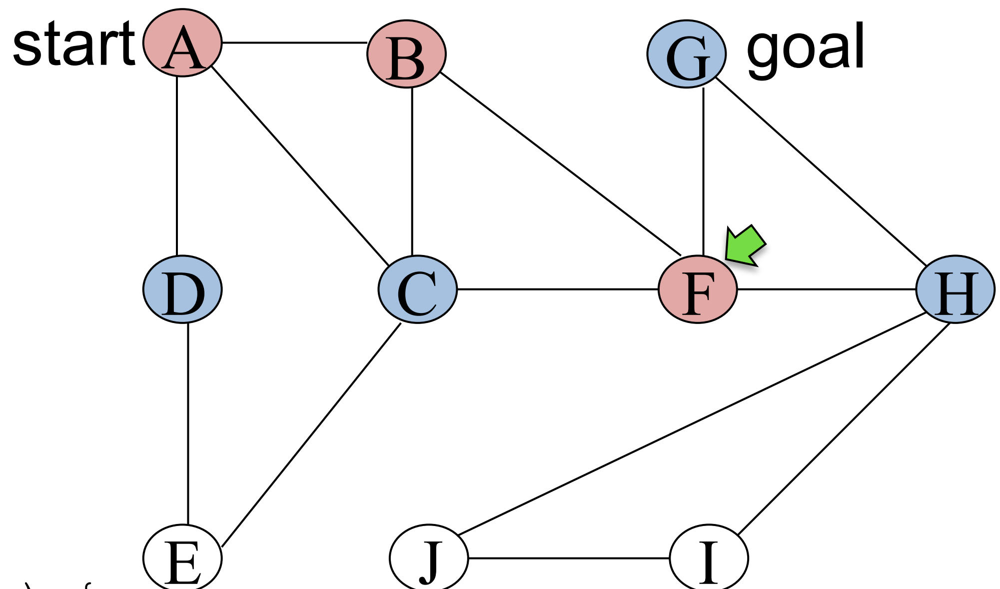



```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
}}}
```

Application: Route Finding (DFS)

open list	closed list
(H,3,F)	A
(G,3,F)	B
(C,2,B)	F
(C,1,A)	
(D,1,A)	

node c
(F,2,B)




```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
}}}
```

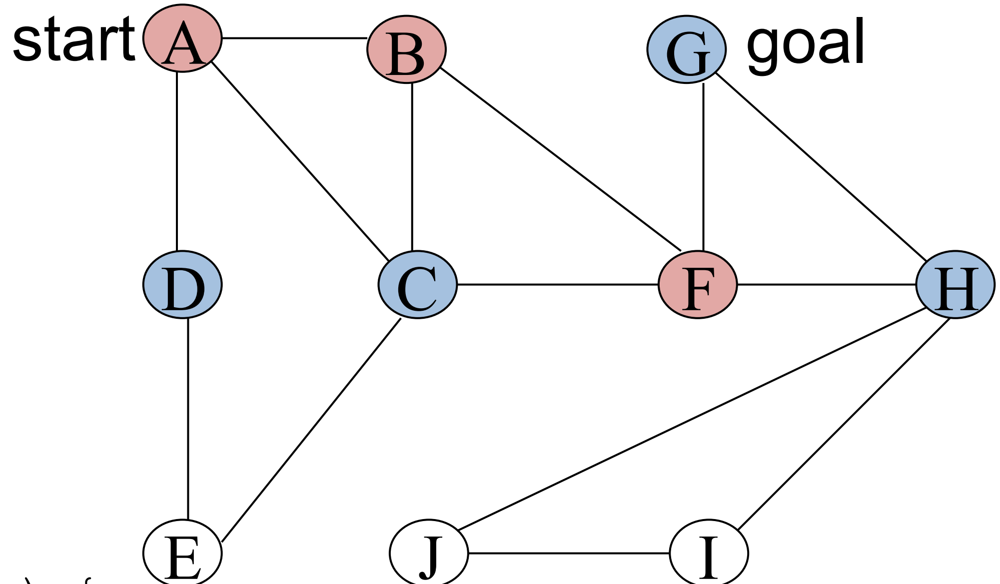
Application: Route Finding (DFS)

open list

(H,3,F)
(G,3,F)
(C,2,B)
(C,1,A)
(D,1,A)

closed list

A
B
F




```
graphSearch(problem, queuingFn) {
  open = {}, closed = {}
  queuingFn(open, new Node(problem.startvertex))
  loop {
    if empty(open) then return FAILURE
    c = removeFront(open)
    if problem.goalTest(c.vertex) then return c
    if c.vertex is not in closed {
      add c.vertex to closed
      for each w adjacent to c.vertex
        if w is not in closed
          queuingFn(open, new Node(w,c));
    }}}
```

What we've found so far...

Breadth-First Search (FIFO Queue)

- Solves **unweighted shortest path problem**:

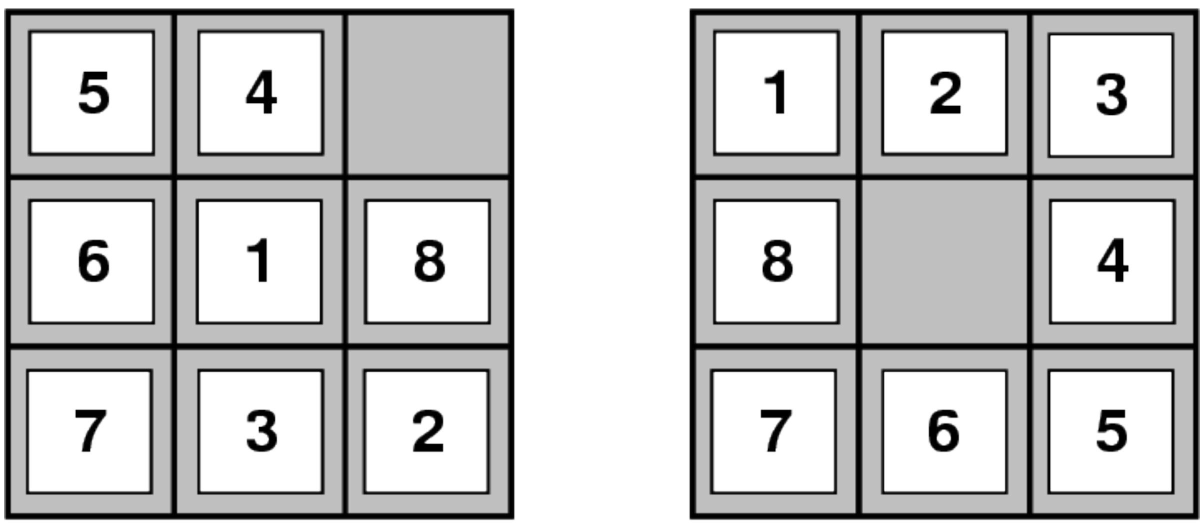
  Finds the shortest path between vertices if edges are unweighted (or equal cost)

Depth-First Search (LIFO Stack)

- Finds nearby goals quickly if lucky
- If unlucky, finds nearby goals very slowly

Application: 8-Puzzle

Given an initial configuration of 8 numbered tiles on a 3 x 3 board, move the tiles as to produce a desired goal configuration

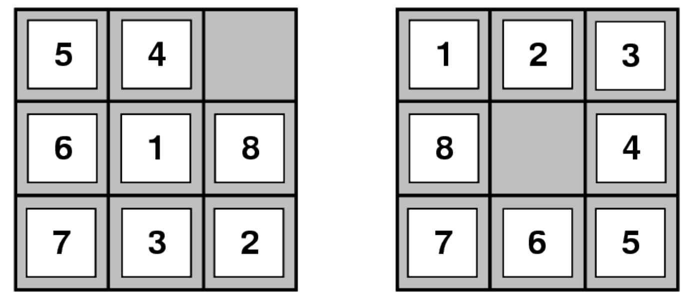


Start State Goal State

Slide adapted from materials by Stuart Russell

Application: 8-Puzzle

- **What are the vertices?**
- **What are the edges?**
- **Starting vertex?**
- **Goal vertex / vertices?**



Start State Goal State

Slide adapted from materials by Stuart Russell

Application: 8-Puzzle

- **What are the vertices?** Each vertex corresponds to a particular tile configuration
- **What are the edges?** Consider four operators: Move Blank Square Left, Right, Up or Down
  – This is a more efficient encoding than considering each of 4 moves for each tile

  The edges signify applying an operator to a board configuration
- **Initial state?** A particular board configuration
- **Goal vertex?** A particular board configuration

Slide adapted from materials by Stuart Russell

Outline

- Introduction
- Graph Basics
- **Graph Search Problem**
  - Breadth-First Search
  - Depth-First Search
- Complexity Analysis

Outline

- Introduction
- Graph Basics
- Graph Search Problem
  - Breadth-First Search
  - Depth-First Search
- **Complexity Analysis**

What is the Time Complexity of BFS & DFS?

- In the worst case, the goal vertex won't be found

```
graphSearch(problem, queuingFn) {
     open = {}, closed = {}

     queuingFn(open, new Node(problem.startvertex))

     loop {
        if empty(open) then return FAILURE

        c = removeFront(open)

        if problem.goalTest(c.vertex) return c

        if c.vertex is not in closed      {
            add c.vertex to closed
            for each Vertex w adjacent to c.vertex
               if w is not in closed
                  queuingFn(open, new Node(w,c));
        }
     }
}
```

What is the Time Complexity of BFS & DFS?

- In the worst case, the goal vertex won't be found

Each vertex is in the queue at most once, so the outer loop runs at most $|V|$ iterations

```
graphSearch(problem, queuingFn) {
     open = {}, closed = {}

     queuingFn(open, new Node(problem.startvertex))

     loop {
        if empty(open) then return FAILURE

        c = removeFront(open)

        if problem.goalTest(c.vertex) return c

        if c.vertex is not in closed     {
            add c.vertex to closed
            for each Vertex w adjacent to c.vertex
               if w is not in closed
                  queuingFn(open, new Node(w,c));
        }
     }
}
```

What is the Time Complexity of BFS & DFS?

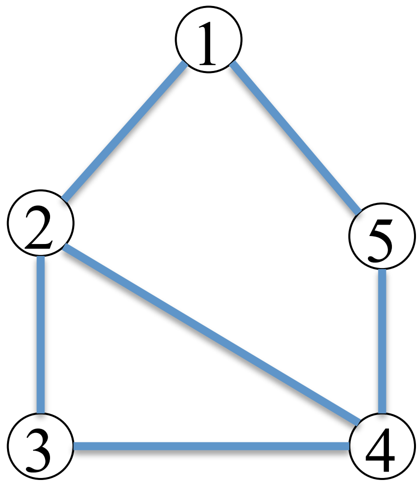
- In the worst case, the goal vertex won't be found

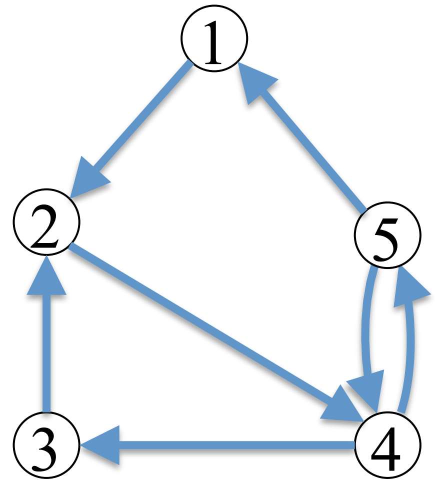
Each vertex is in the queue at most once, so the outer loop runs at most $|V|$ iterations

Performance will depend on the time for getAdjacent()

```
graphSearch(problem, queuingFn) {
     open = {}, closed = {}

     queuingFn(open, new Node(problem.startvertex))

     loop {
        if empty(open) then return FAILURE

        c = removeFront(open)

        if problem.goalTest(c.vertex) return c

        if c.vertex is not in closed     {
            add c.vertex to closed
            for each Vertex w adjacent to c.vertex
               if w is not in closed
                  queuingFn(open, new Node(w,c));
        }
     }
}
```

Graph Representation: Adjacency Matrix

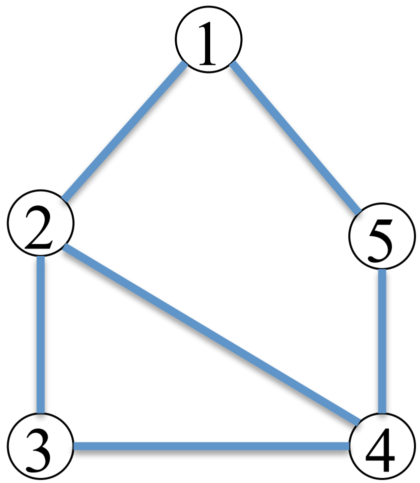


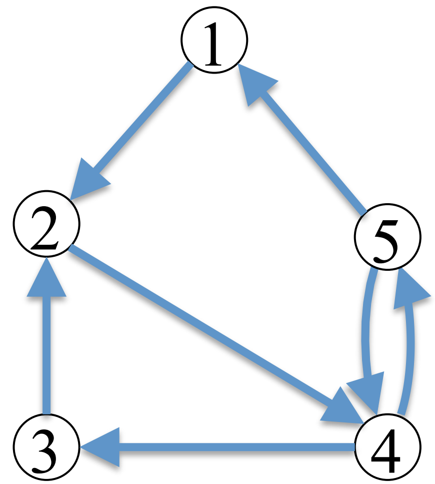


	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	0
3	0	1	0	1	0
4	0	1	1	0	1
5	1	0	0	1	0

	1	2	3	4	5
1	0	1	0	0	0
2	0	0	0	1	0
3	0	1	0	0	0
4	0	0	1	0	1
5	1	0	0	1	0

Graph Representation: Adjacency Matrix



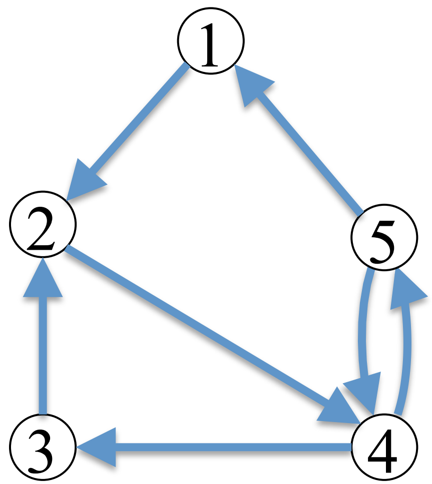


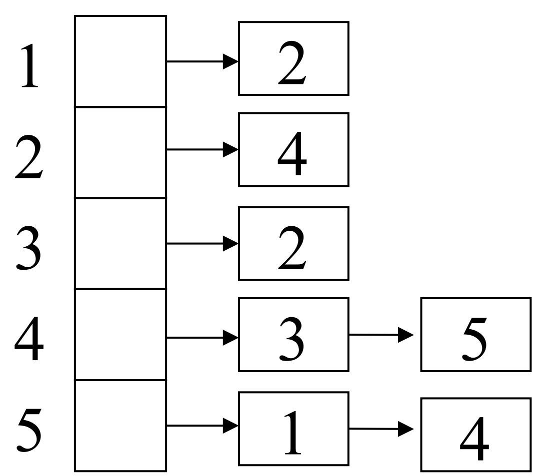
	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	0
3	0	1	0	1	0
4	0	1	1	0	1
5	1	0	0	1	0

	1	2	3	4	5
1	0	1	0	0	0
2	0	0	0	1	0
3	0	1	0	0	0
4	0	0	1	0	1
5	1	0	0	1	0

What is the performance of getAdjacent(u)?

Graph Representation: Adjacency List





What is the performance of getAdjacent(u)?

What is the Time Complexity of BFS & DFS?

- Using an adjacency matrix:

$|V|$ iterations

$O(|V|)$

```
graphSearch(problem, queuingFn) {
    open = {}, closed = {}

    queuingFn(open, new Node(problem.startvertex))

    loop {
       if empty(open) then return FAILURE

       c = removeFront(open)

       if problem.goalTest(c.vertex) return c

       if c.vertex is not in closed     {
           add c.vertex to closed
           for each Vertex w adjacent to c.vertex
              if w is not in closed
                 queuingFn(open, new Node(w,c));
       }
    }
}
```

What is the Time Complexity of BFS & DFS?

- Using an adjacency matrix: $O(|V|^2)$

$|V|$ iterations

$O(|V|)$

```
graphSearch(problem, queuingFn) {
     open = {}, closed = {}

     queuingFn(open, new Node(problem.startvertex))

     loop {
        if empty(open) then return FAILURE

        c = removeFront(open)

        if problem.goalTest(c.vertex) return c

        if c.vertex is not in closed     {
            add c.vertex to closed
            for each Vertex w adjacent to c.vertex
               if w is not in closed
                  queuingFn(open, new Node(w,c));
        }
     }
}
```

What is the Time Complexity of BFS & DFS?

- Using an adjacency list:

$|V|$ iterations

O(out-degree(c.vertex))

```
graphSearch(problem, queuingFn) {
     open = {}, closed = {}

     queuingFn(open, new Node(problem.startvertex))

     loop {
        if empty(open) then return FAILURE

        c = removeFront(open)

        if problem.goalTest(c.vertex) return c

        if c.vertex is not in closed     {
            add c.vertex to closed
            for each Vertex w adjacent to c.vertex
               if w is not in closed
                  queuingFn(open, new Node(w,c));
        }
     }
}
```

What is the Time Complexity of BFS & DFS?

- For an adjacency list, looping over all adjacent vertices of u will be O(out-degree(u))

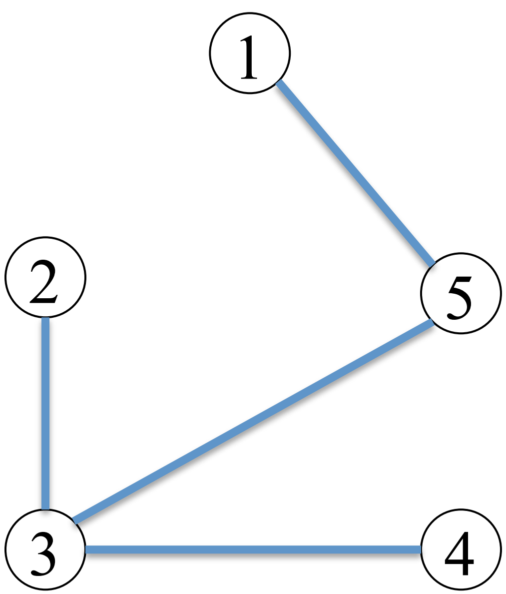
- Therefore, the traversal performance is
$$O\left(\sum_{i=1}^{|V|} \text{out-degree}(v_i)\right) = O(|E|)$$
since the inner loop is repeated O($|V|$) times

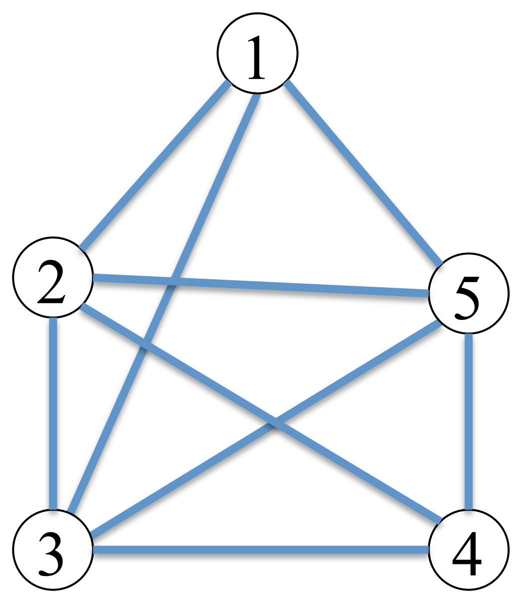
- However, in a disconnected graph, we must still look at every vertex, so the performance is O($|V|$ + $|E|$)

How do these terms compare?

Sparse vs Dense Graphs

- A ***sparse graph*** is one with "few" edges. That is $|E| = O\left(|V|\right)$
- A ***dense graph*** is one with "many" edges. That is $|E| = O\left(|V|^2\right)$





What is the Time Complexity of BFS & DFS?

- For an adjacency list, getAdjacent(u) will be O(out-degree(u))
- Therefore, the traversal performance is

$$O\left(\sum_{i=1}^{|V|} \text{out-degree}(v_i)\right) = O(|E|)$$

since getAdjacent is done O($|V|$) times

- However, in a disconnected graph, we must still look at every vertex, so the performance is O($|V| + |E|$)

Ranges from O($|V|$) to O($|V|^2$), depending on density

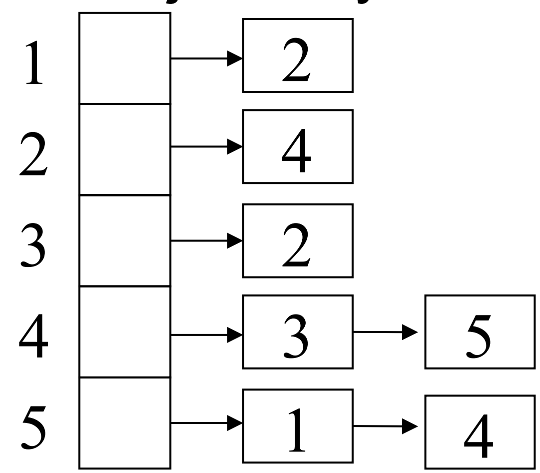
What is the Space Complexity of BFS & DFS?

- Really depends on the graph representation

Adjacency Matrix

	1	**2**	**3**	**4**	**5**
1	0	1	0	0	0
2	0	0	0	1	0
3	0	1	0	0	0
4	0	0	1	0	1
5	1	0	0	1	0

Adjacency List



What is the Space Complexity of BFS & DFS?

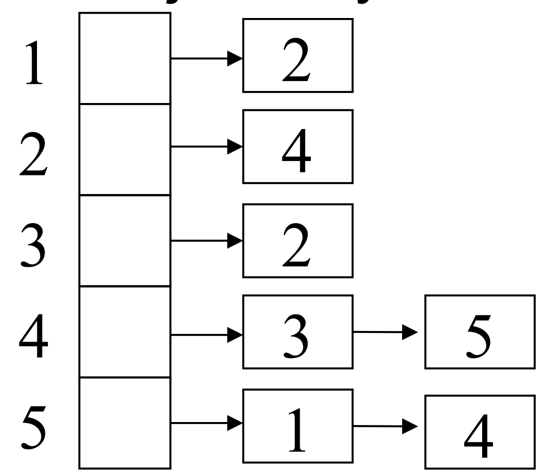
- Really depends on the graph representation

Adjacency Matrix

	1	**2**	**3**	**4**	**5**
1	0	1	0	0	0
2	0	0	0	1	0
3	0	1	0	0	0
4	0	0	1	0	1
5	1	0	0	1	0

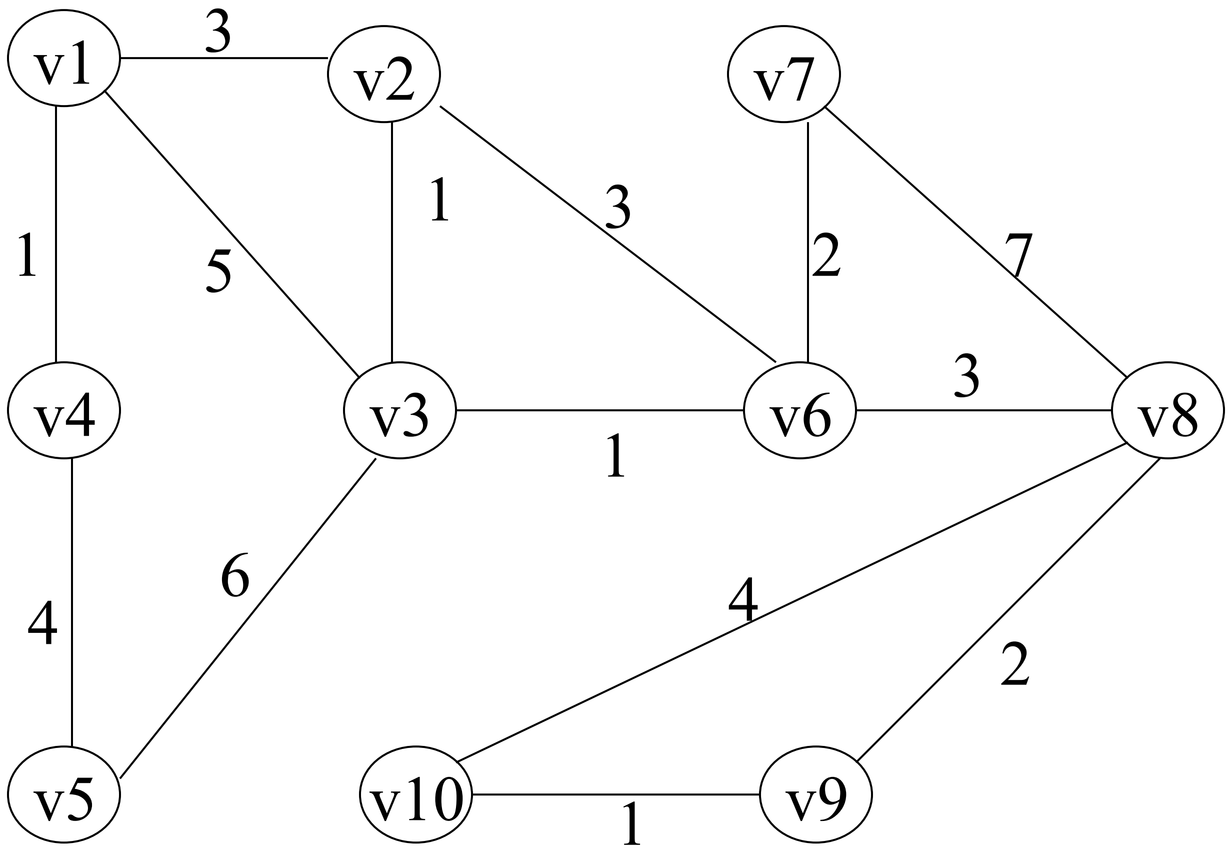
Space Complexity
O($|V|^2$)

Adjacency List



Space Complexity
O($|V|$ + $|E|$)

Does BFS find Shortest Paths in Weighted Graphs?



Summary

- Breadth-First Search
  - Solves unweighted shortest path problem
  - Uses FIFO queue
  - Traverses the graph in level-order
- Depth-First Search
  - Uses LIFO stack
  - Takes a “deep-dive” into the graph
- Time/Space Complexity: O($|V|$ + $|E|$)