CS 344: Design and Analysis of Computer Algorithms

(Spring 2022 — Sections 5,6,7,8)

Lecture 15: Graph Reductions, Graph Search: DFS

Graph Reductions

Graph Reductions

- Model the given problem as a graph problem
  - Specify exactly how you can get your graph from the input of the original problem
- Run any **black-box** algorithm for the graph problem you specified
- Translate back the solution to the graph problem to the solution of your original problem

Example: Fill Coloring Problem

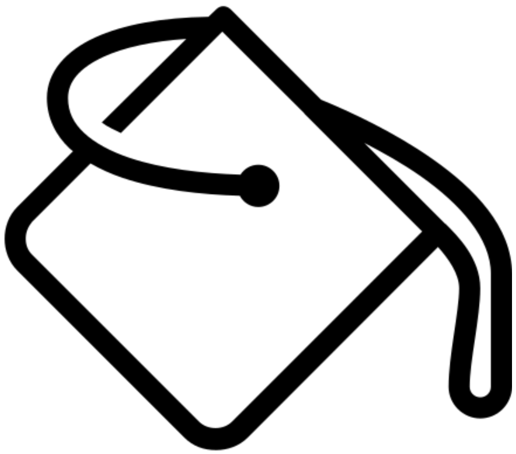
Example: Fill Coloring Problem

0	1	0	1	0	0	0	1	0	1
0	0	0	1	0	0	1	1	0	1
0	0	1	1	1	1	1	1	1	1
0	0	0	0	0	1	1	1	1	1
0	1	0	0	0	1	0	1	1	0
1	1	1	0	0	1	1	1	0	0
0	0	0	1	0	1	0	0	0	0
0	0	0	1	1	1	0	0	0	0
0	0	0	1	1	1	1	1	1	0
0	0	0	0	1	1	1	0	0	0

Example: Fill Coloring Problem

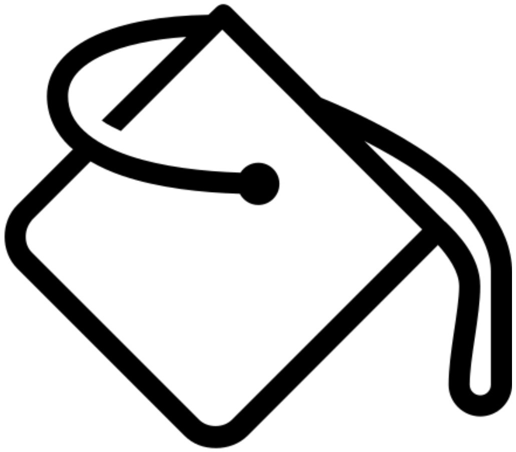
0	1	0	1	0	0	0	1	0	1
0	0	0	1	0	0	1	1	0	1
0	0	1	1	1	1	1	1	1	1
0	0	0	0	0	1	1	1	1	1
0	1	0	0	0	1	0	1	1	0
1	1	1	0	0	1	1	1	0	0
0	0	0	1	0	1	0	0	0	0
0	0	0	1	1	1	0	0	0	0
0	0	0	1	1	1	1	1	1	0
0	0	0	0	1	1	1	0	0	0

Example: Fill Coloring Problem



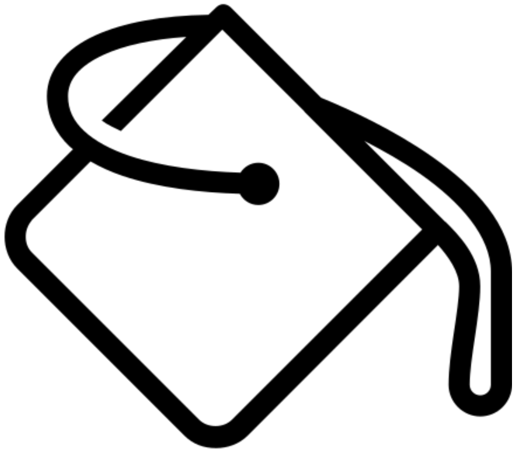
0	1	0	1	0	0	0	1	0	1
0	0	0	1	0	0	1	1	0	1
0	0	1	1	1	1	1	1	1	1
0	0	0	0	0	1	1	1	1	1
0	1	0	0	0	1	0	1	1	0
1	1	1	0	0	1	1	1	0	0
0	0	0	1	0	1	0	0	0	0
0	0	0	1	1	1	0	0	0	0
0	0	0	1	1	1	1	1	1	0
0	0	0	0	1	1	1	0	0	0

Example: Fill Coloring Problem



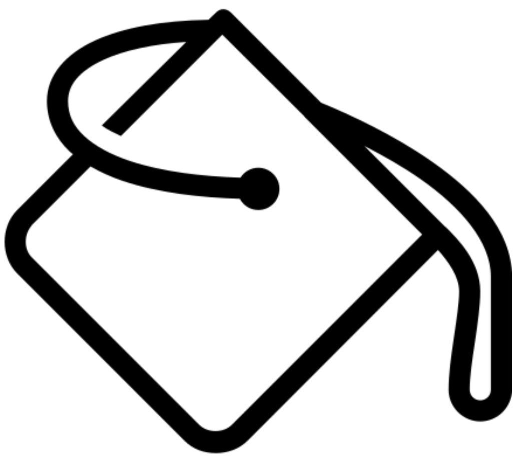
0	1	0	1	0	0	0	1	0	1
0	0	0	1	0	0	1	1	0	1
0	0	1	1	1	1	1	1	1	1
0	0	0	0	0	1	1	1	1	1
0	1	0	0	0	1	0	1	1	0
1	1	1	0	0	1	1	1	0	0
0	0	0	1	0	1	0	0	0	0
0	0	0	1	1	1	0	0	0	0
0	0	0	1	1	1	1	1	1	0
0	0	0	0	1	1	1	0	0	0

Example: Fill Coloring Problem



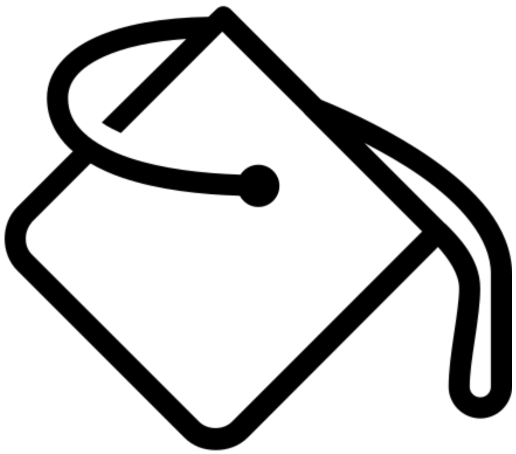
0	1	0	1	0	0	0	1	0	1
0	0	0	1	0	0	1	1	0	1
0	0	1	1	1	1	1	1	1	1
0	0	0	0	0	1	1	1	1	1
0	1	0	0	0	1	0	1	1	0
1	1	1	0	0	1	1	1	0	0
0	0	0	1	0	1	0	0	0	0
0	0	0	1	1	1	0	0	0	0
0	0	0	1	1	1	1	1	1	0
0	0	0	0	1	1	1	0	0	0

Example: Fill Coloring Problem



0	1	0	1	0	0	0	1	0	1
0	0	0	1	0	0	1	1	0	1
0	0	1	1	1	1	1	1	1	1
0	0	0	0	0	1	1	1	1	1
0	1	0	0	0	1	0	1	1	0
1	1	1	0	0	1	1	1	0	0
0	0	0	1	0	1	0	0	0	0
0	0	0	1	1	1	0	0	0	0
0	0	0	1	1	1	1	1	1	0
0	0	0	0	1	1	1	0	0	0

Example: Fill Coloring Problem



0	1	0	1	0	0	0	1	0	1
0	0	0	1	0	0	1	1	0	1
0	0	1	1	1	1	1	1	1	1
0	0	0	0	0	1	1	1	1	1
0	1	0	0	0	1	0	1	1	0
1	1	1	0	0	1	1	1	0	0
0	0	0	1	0	1	0	0	0	0
0	0	0	1	1	1	0	0	0	0
0	0	0	1	1	1	1	1	1	0
0	0	0	0	1	1	1	0	0	0

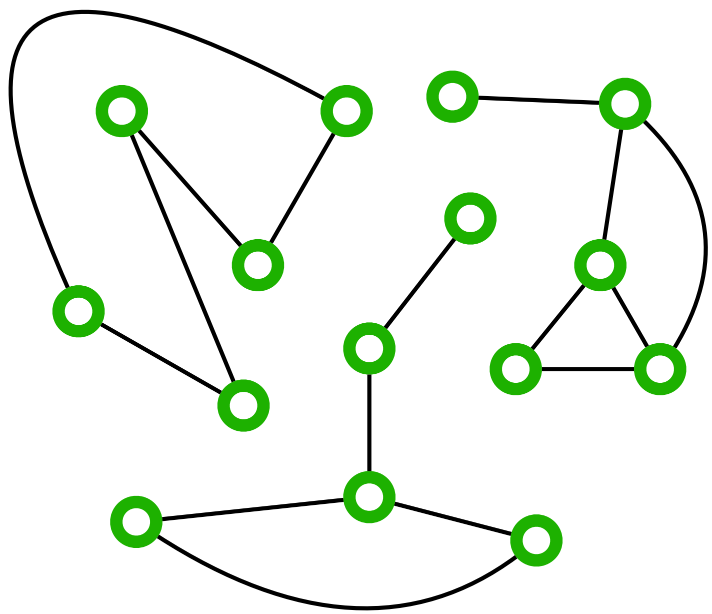
Example: Fill Coloring Problem

- **Input:** A pixel map and a starting point
  - An input 2D-array A[1:k][1:k] with entries in {0,1}.
  - A single cell i_s, j_s with $A[i_s][j_s] = 0$ as the starting point
- **Output:** Find the cells colored by the fill coloring of this pixel map
  - Color starting point,
  - go to any neighboring cell (left, right, top, down) with 0-value in the matrix and color them, go to the neighbor of any cell colored
  - continue until no new cell can be colored

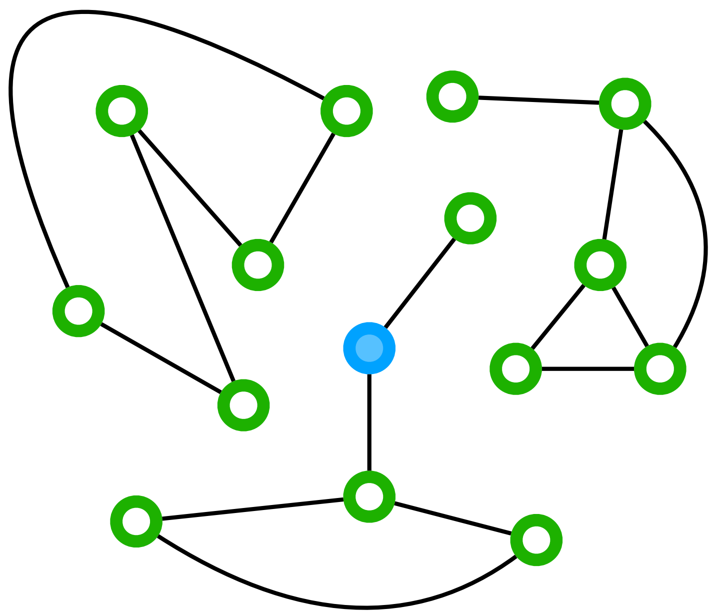
Graph Search

- **Input:**
  - An undirected graph $G=(V,E)$
  - A starting vertex s
- **Output**
  - All vertices in the connected component of G that contains s
    - Vertices reachable from s in the graph via some path

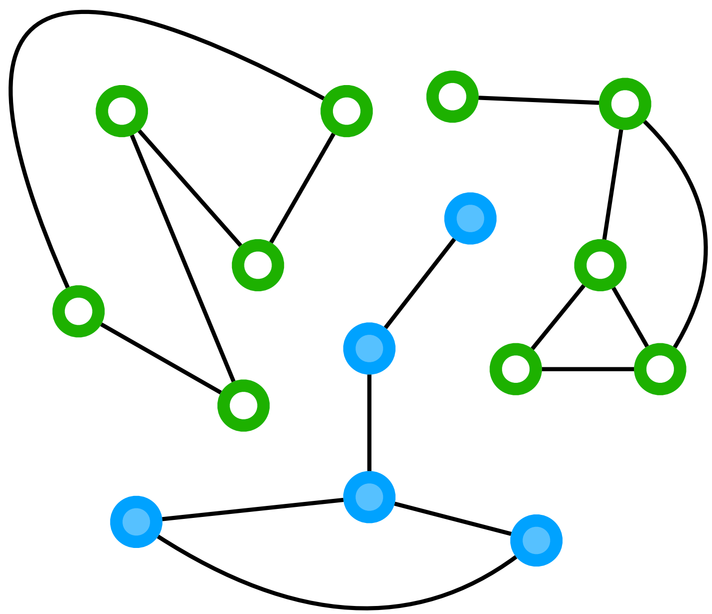
Graph Search: Example



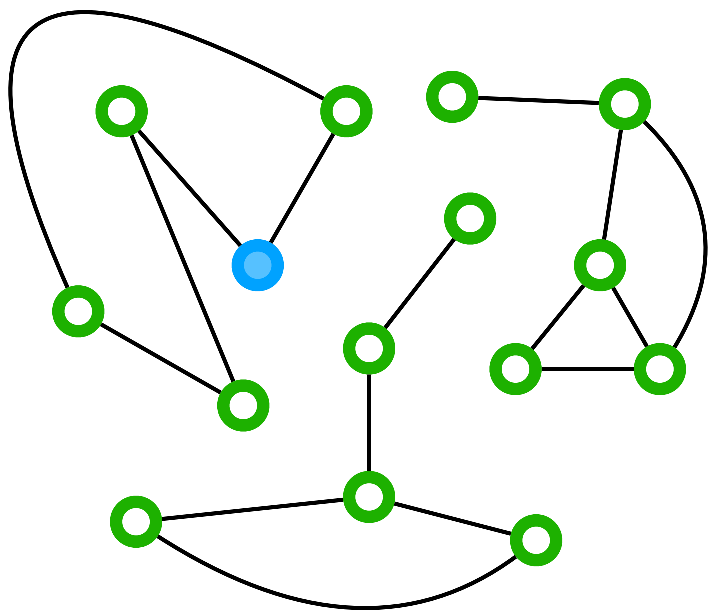
Graph Search: Example



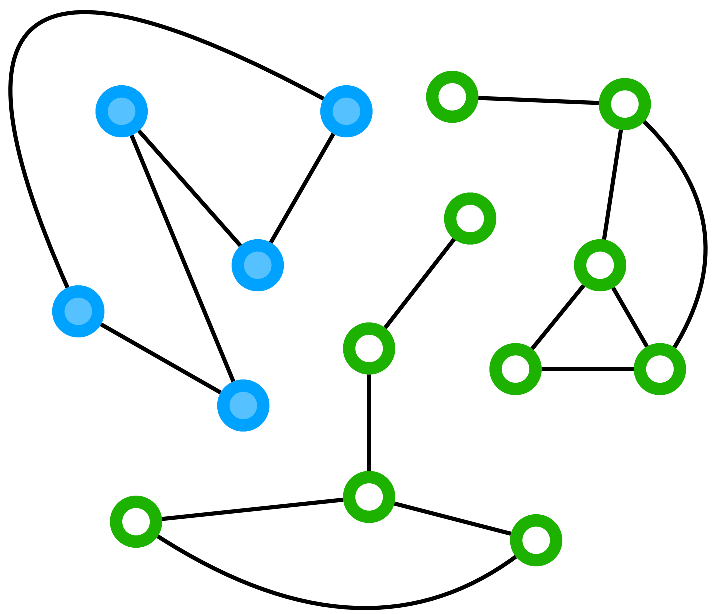
Graph Search: Example



Graph Search: Example



Graph Search: Example



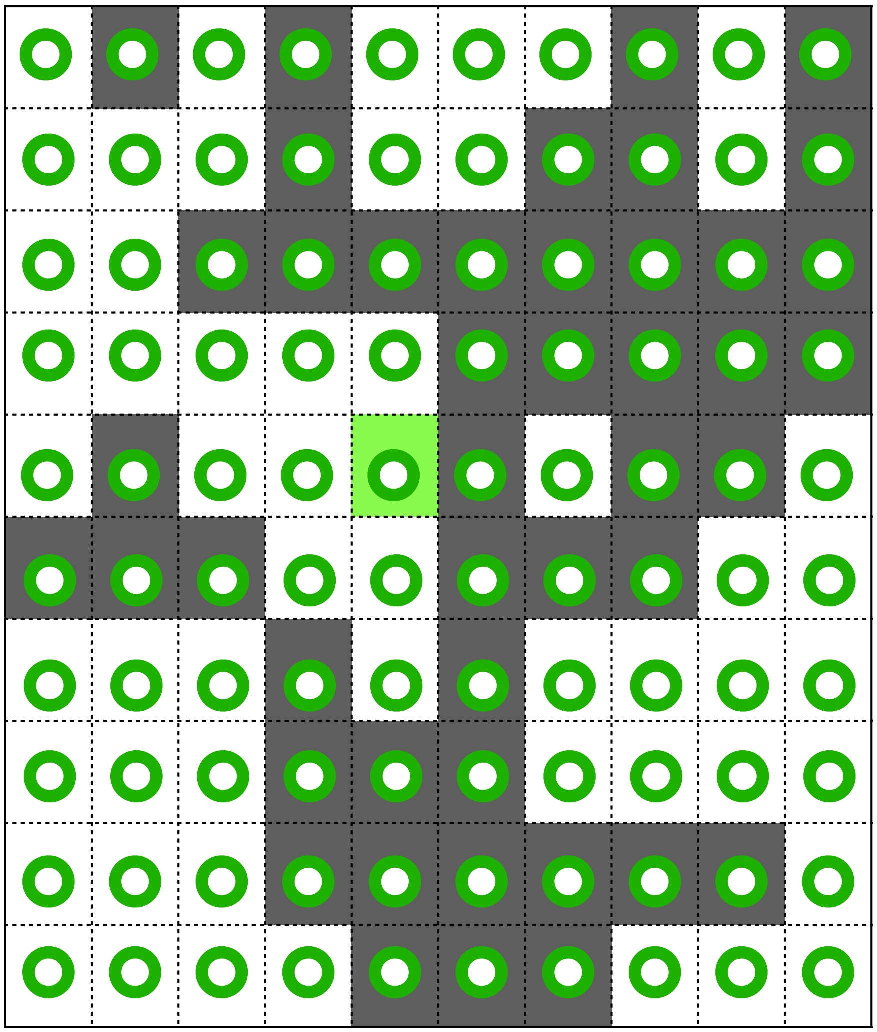
Reducing Fill Coloring to Graph Search

- Algorithm for Fill Coloring (Reduction):
  - Create a graph $G = (V, E)$ with $n = k^2$ vertices as follows:
    - For any cell (i, j) in the array A create a vertex $v_{i,j}$
    - For any neighboring cells (i, j) and (x, y) if both $A[i][j] = A[x][y] = 0$ then add an edge $(v_{i,j}, v_{x,y})$.
  - Define the starting vertex of search as $s = v_{i_s,j_s}$ where (i_s, j_s) is the starting point in the fill coloring problem
  - Run any graph search algorithm on G and s and return all cells (i, j) where their corresponding vertex $v_{i,j}$ is output by the search algorithm

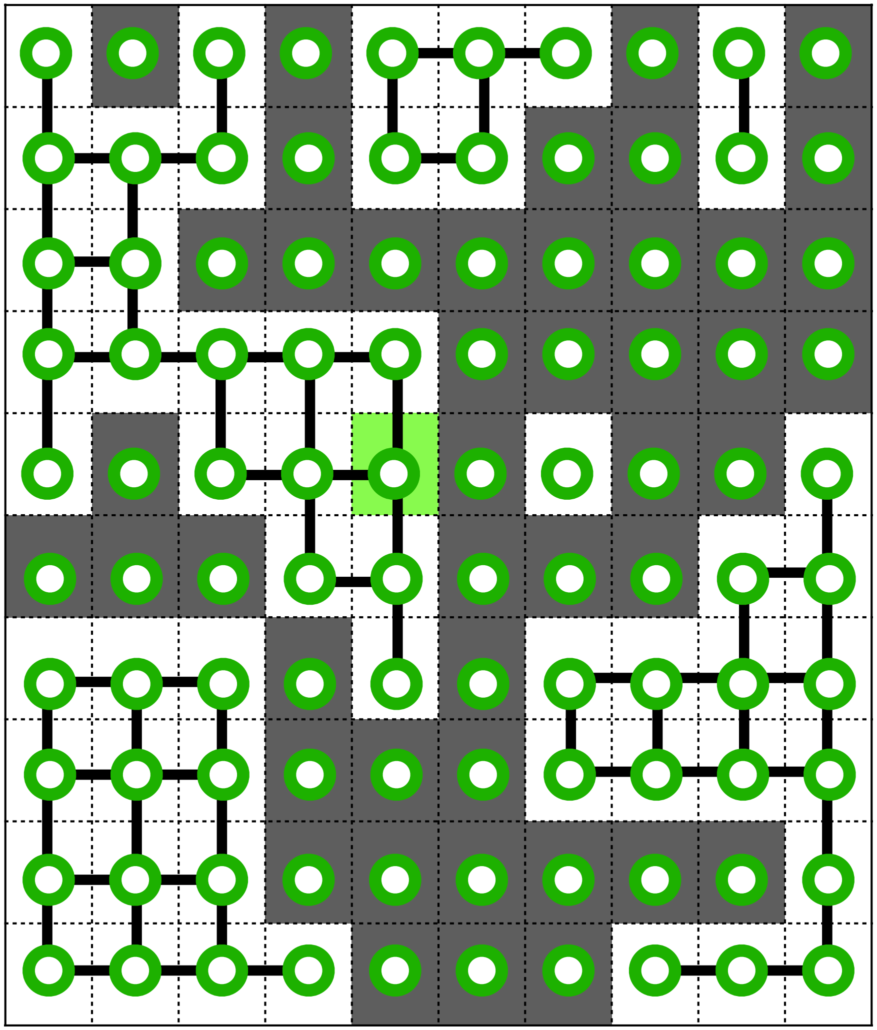
Example of the Reduction

0	1	0	1	0	0	0	1	0	1
0	0	0	1	0	0	1	1	0	1
0	0	1	1	1	1	1	1	1	1
0	0	0	0	0	1	1	1	1	1
0	1	0	0	0	1	0	1	1	0
1	1	1	0	0	1	1	1	0	0
0	0	0	1	0	1	0	0	0	0
0	0	0	1	1	1	0	0	0	0
0	0	0	1	1	1	1	1	1	0
0	0	0	0	1	1	1	0	0	0

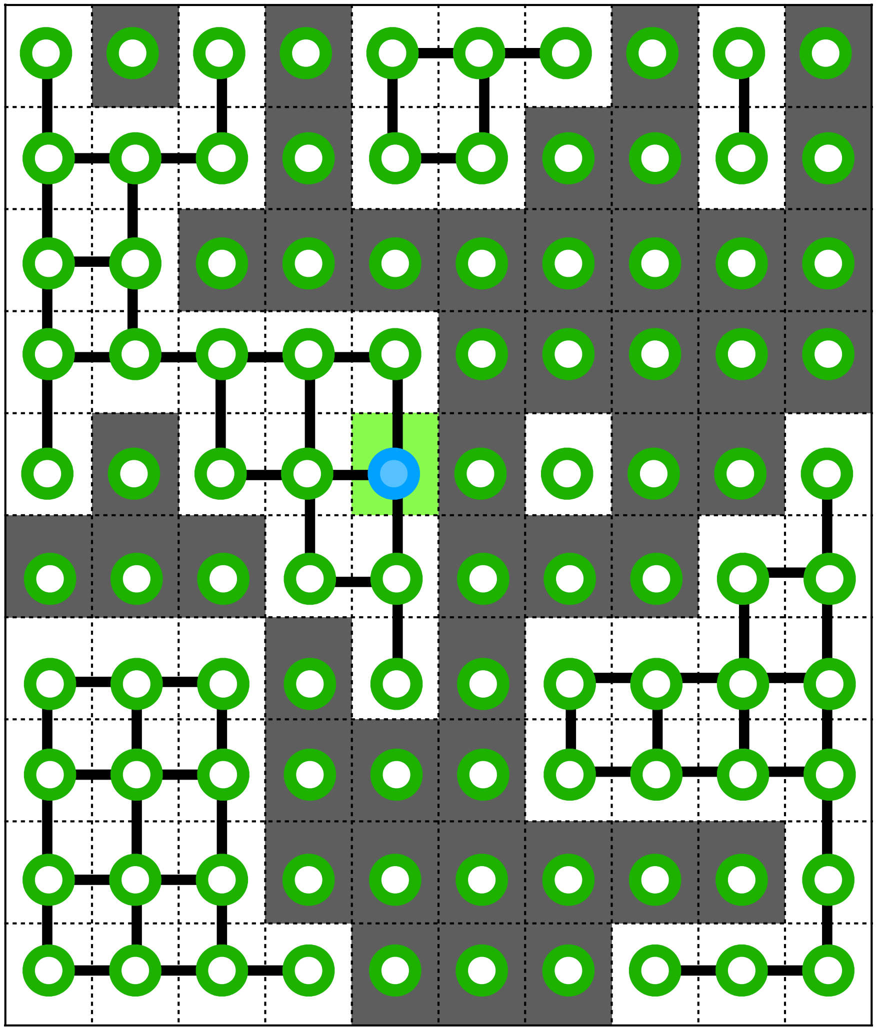
Example of the Reduction



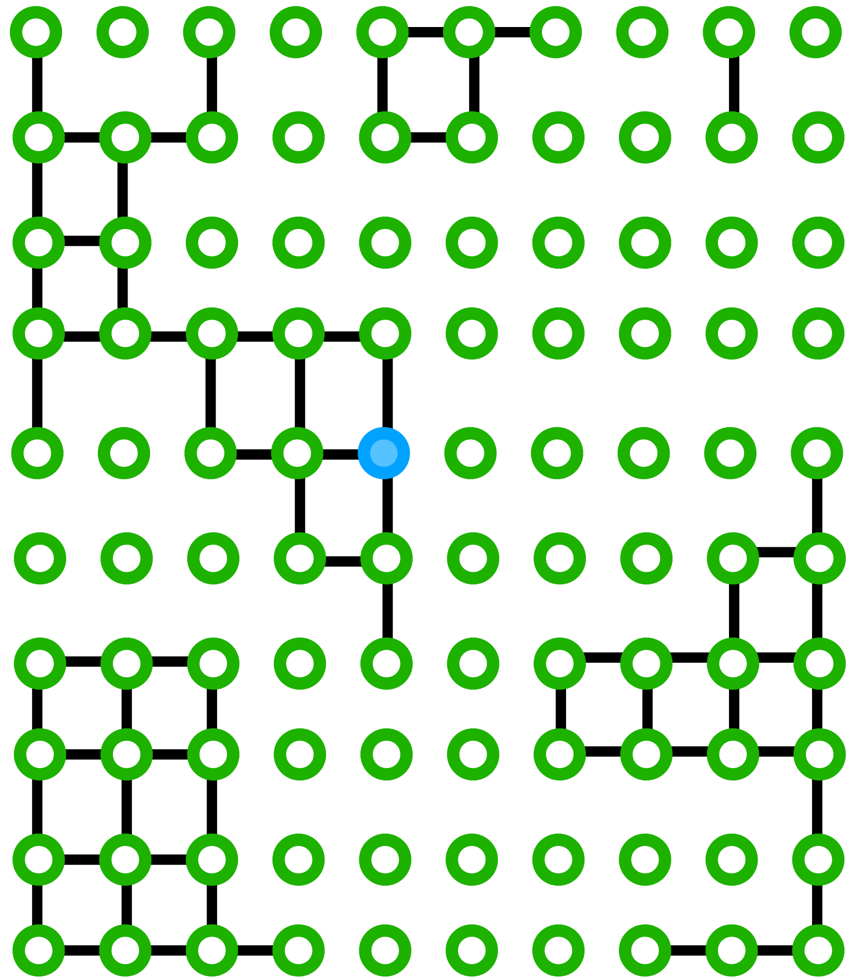
Example of the Reduction



Example of the Reduction

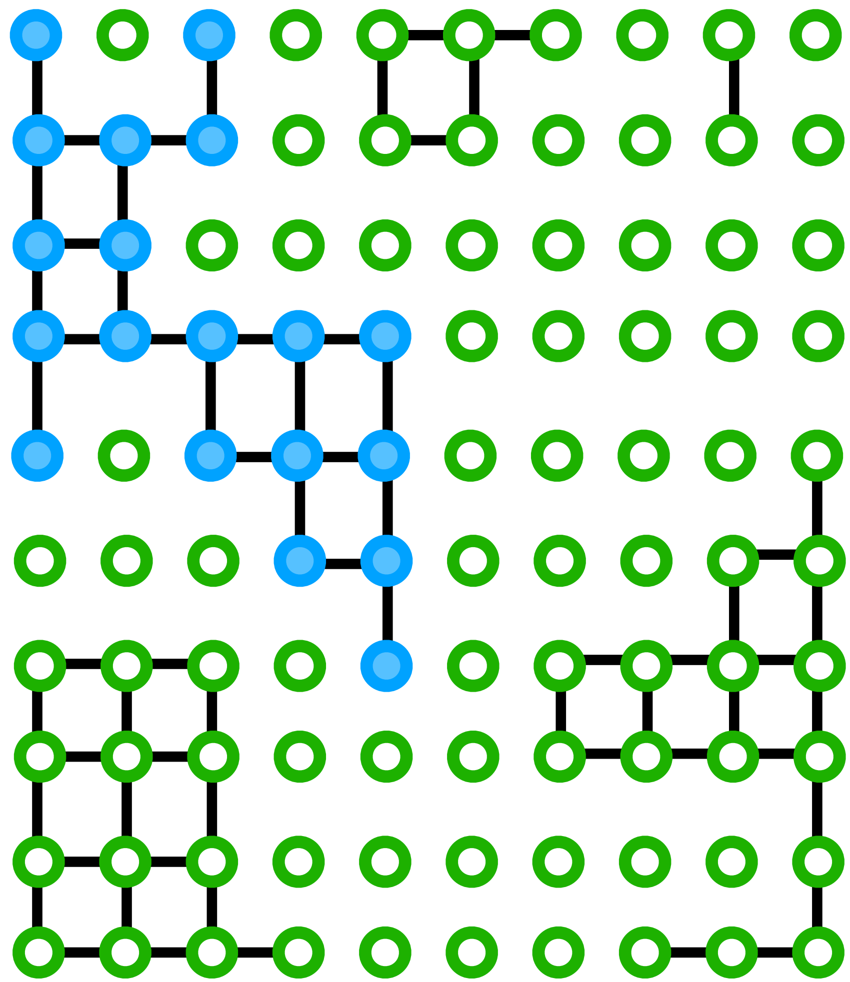


Example of the Reduction

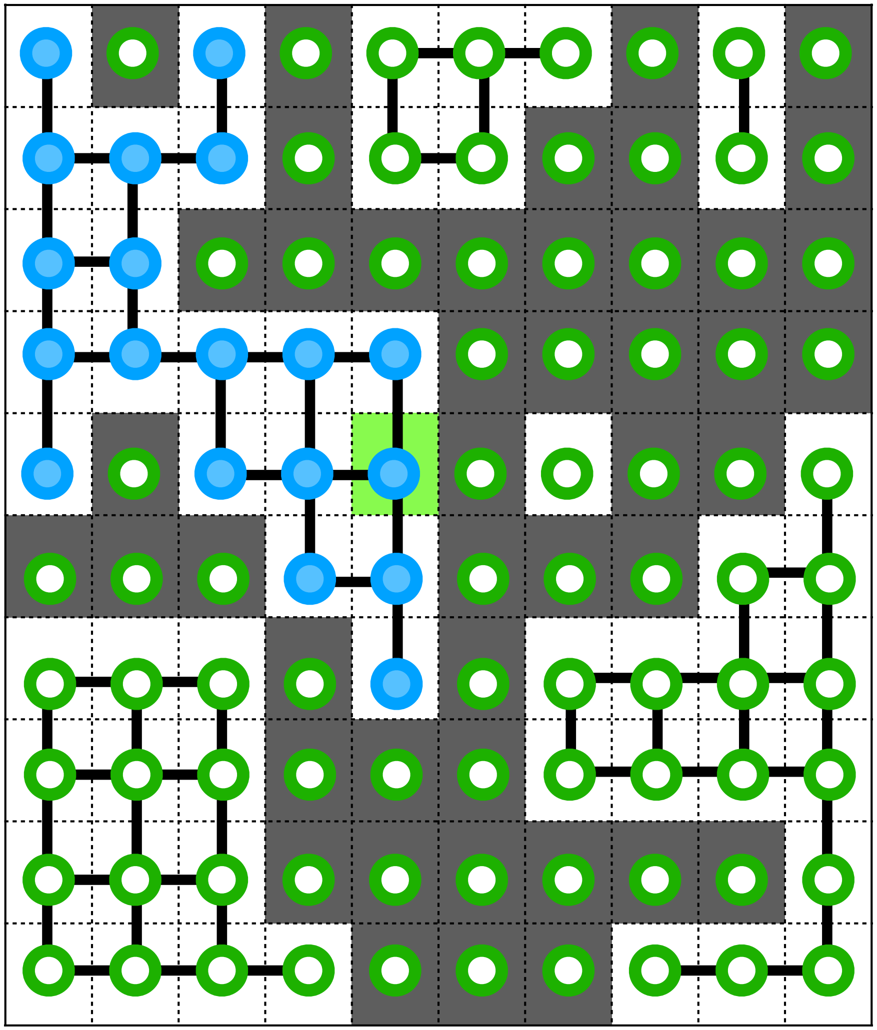


Example of the Reduction

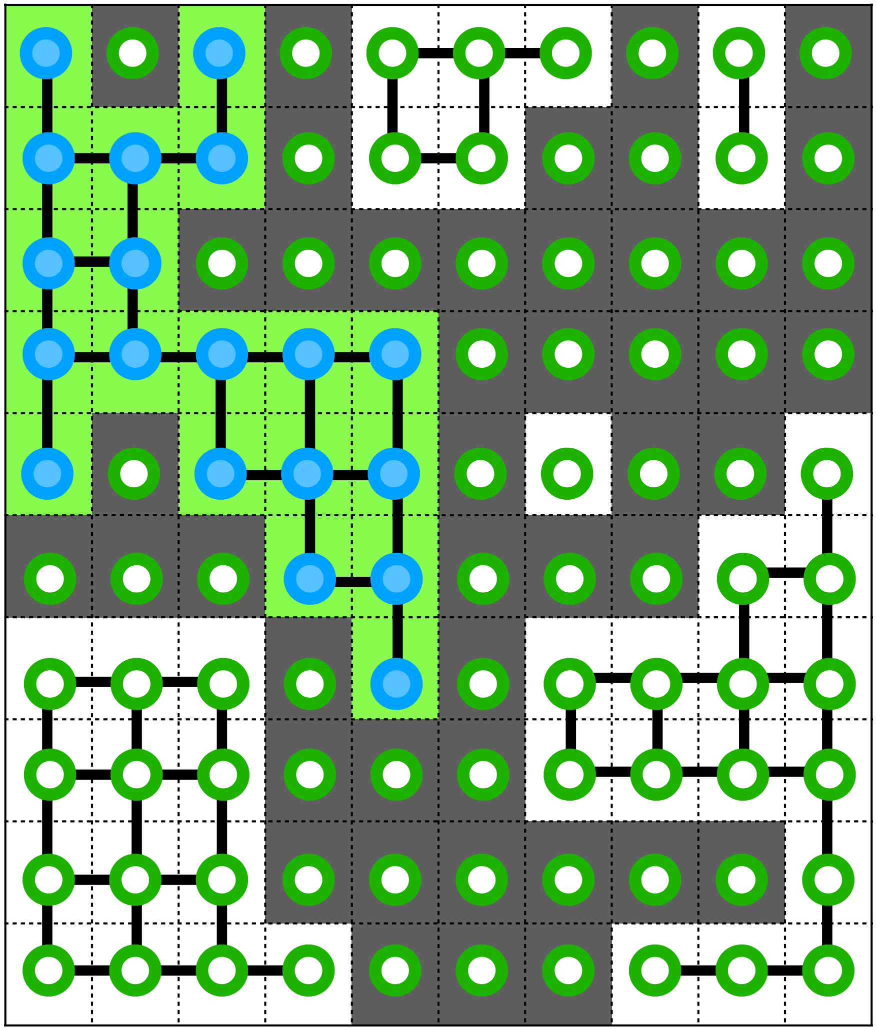
Graph Search Algorithm



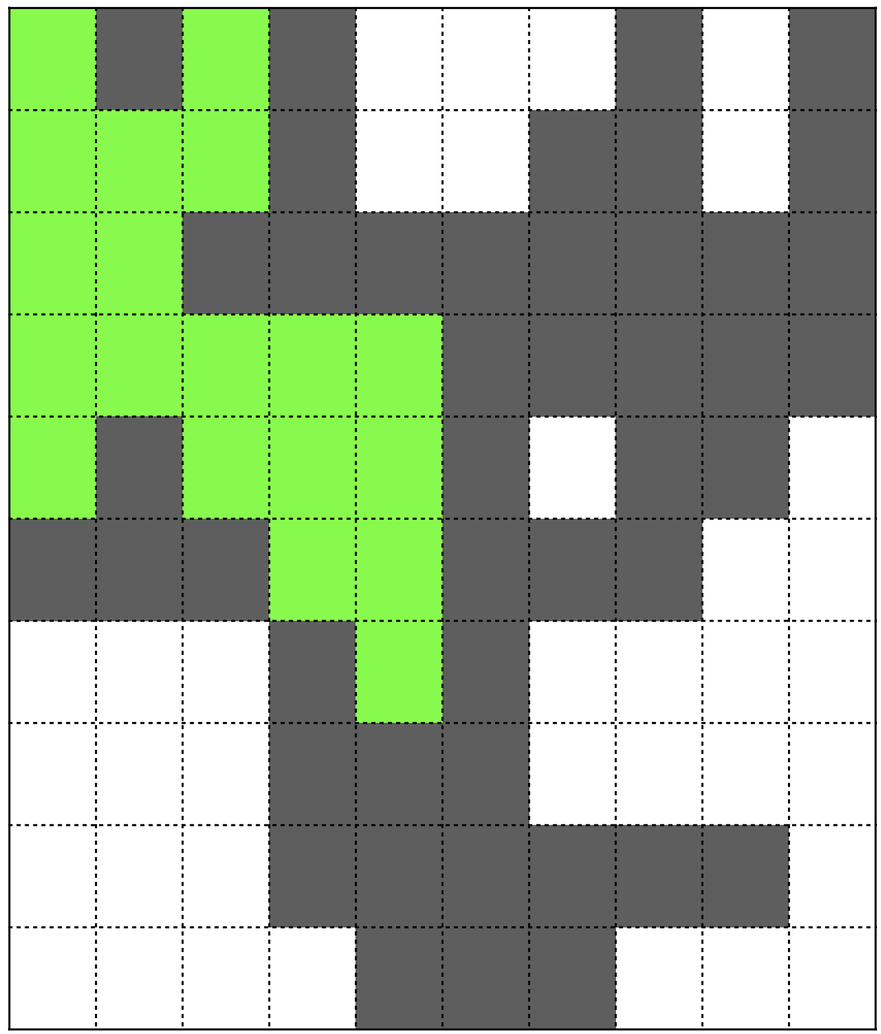
Example of the Reduction



Example of the Reduction



Example of the Reduction



Proof of Correctness

- We prove that the set of cells that should be fill colored is exactly the same as the connected component of the vertex v_{i_s, j_s}

Proof of Correctness

- We prove that the set of cells that should be fill colored is exactly the same as the connected component of the vertex v_{i_s,j_s}
- Part one: any cell that needs to be colored corresponds to a vertex in the connected component of v_{i_s,j_s}
- Part two: any vertex in the connected component of v_{i_s,j_s} corresponds to a cell that needs to be colored

Proof of Correctness

- We prove that the set of cells that should be fill colored is exactly the same as the connected component of the vertex v_{i_s,j_s}
- Part one: any cell that needs to be colored corresponds to a vertex in the connected component of v_{i_s,j_s}
- Part two: any vertex in the connected component of v_{i_s,j_s} corresponds to a cell that needs to be colored
- We should remember to prove **both parts**!

Proof of Correctness

- Part one: any cell that needs to be colored corresponds to a vertex in the connected component of v_{i_s,j_s}
- **Proof:**

Proof of Correctness

- Part two: any vertex in the connected component of v_{i_s,j_s} corresponds to a cell that needs to be colored
- **Proof:**

Runtime Analysis

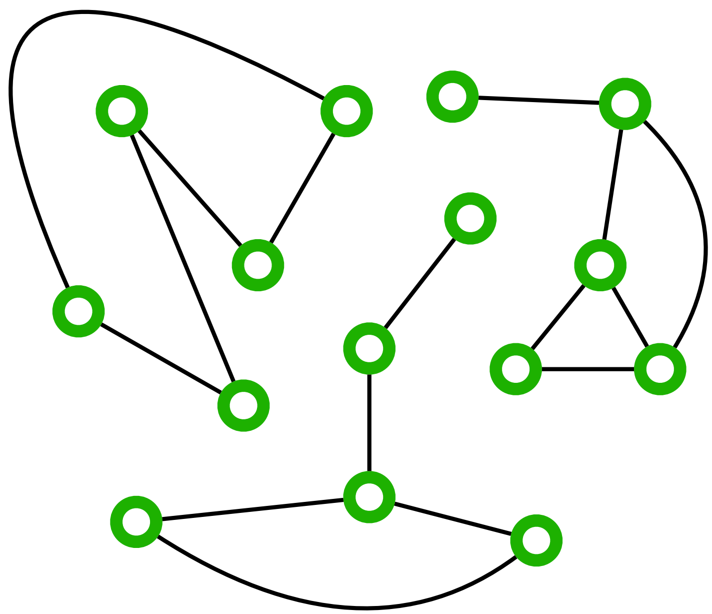
- Creating the graph $G = (V, E)$ takes $O(k^2)$ time
- This graph has $n = \Theta(k^2)$ & $m = \Theta(k^2)$
- Let $Search(n, m)$ denote the runtime of the best algorithm for doing a graph search on a graph with n vertices and m edges
- Runtime of our algorithm is $O(k^2 + Search(\Theta(k^2), \Theta(k^2)))$
- In the next lecture, we design different algorithms for graph search and show that $Search(n, m) = O(n + m)$
- This makes our runtime $O(k^2)$

Graph Search Algorithm 1: Depth-First-Search (DFS)

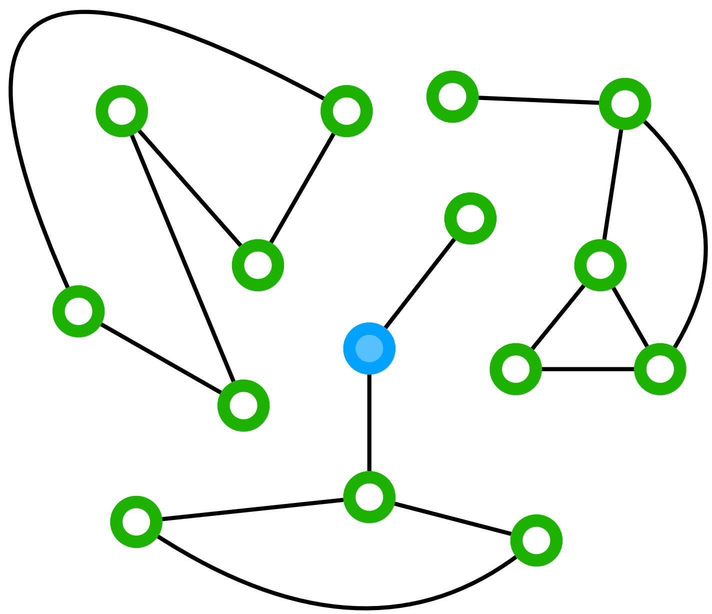
Graph Search

- **Input:**
  - An undirected graph $G=(V,E)$
  - A starting vertex s
- **Output**
  - All vertices in the connected component of G that contains s
    - Vertices reachable from s in the graph via some path

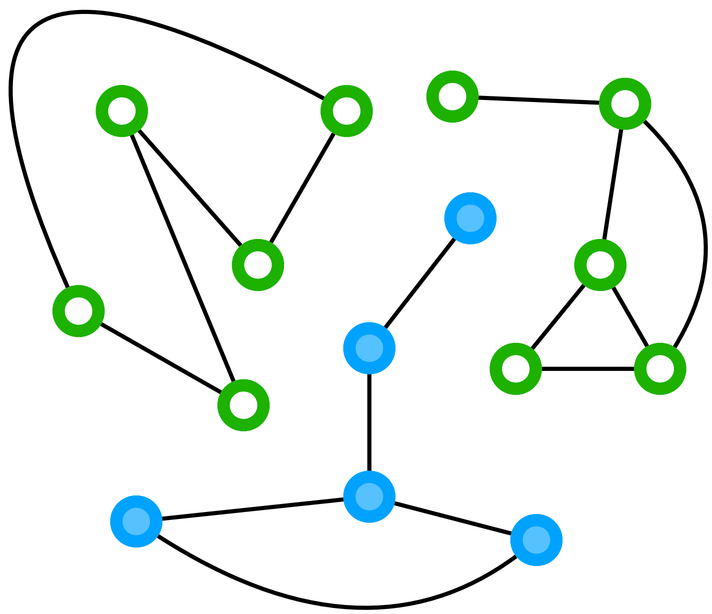
Graph Search: Example



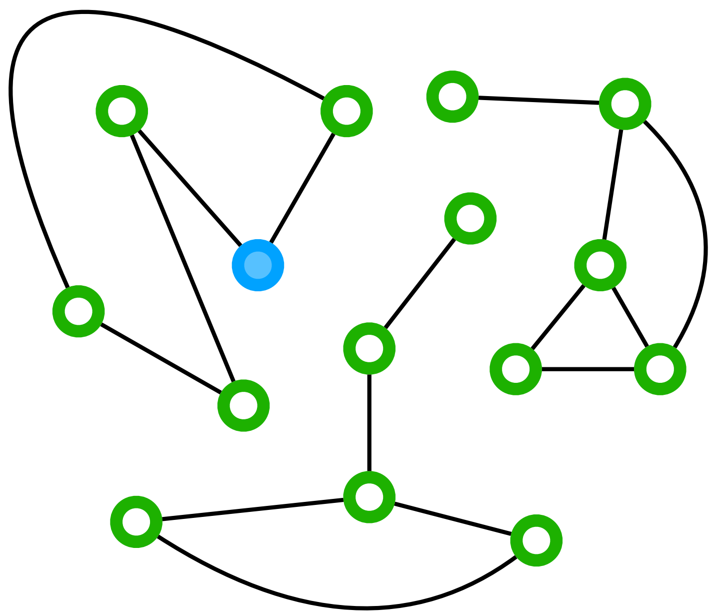
Graph Search: Example



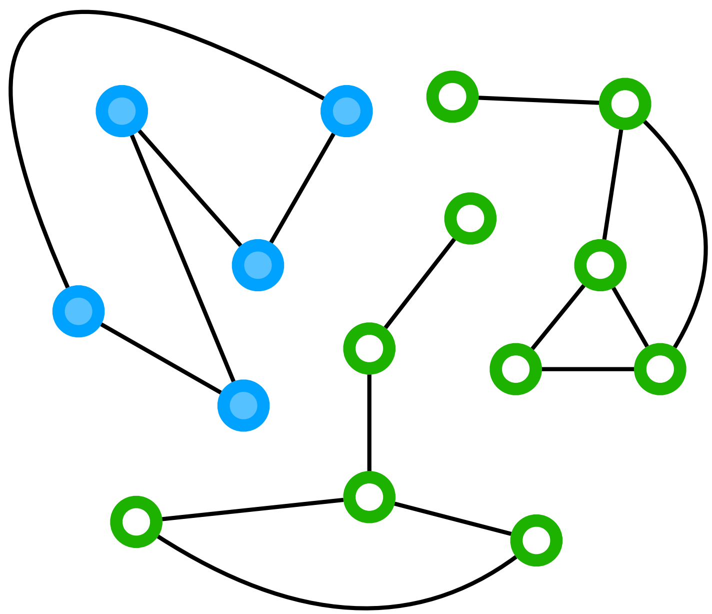
Graph Search: Example



Graph Search: Example



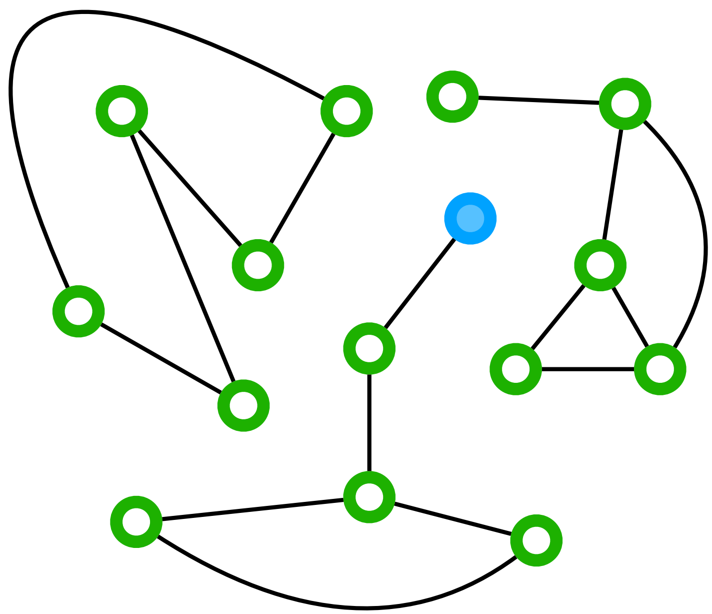
Graph Search: Example



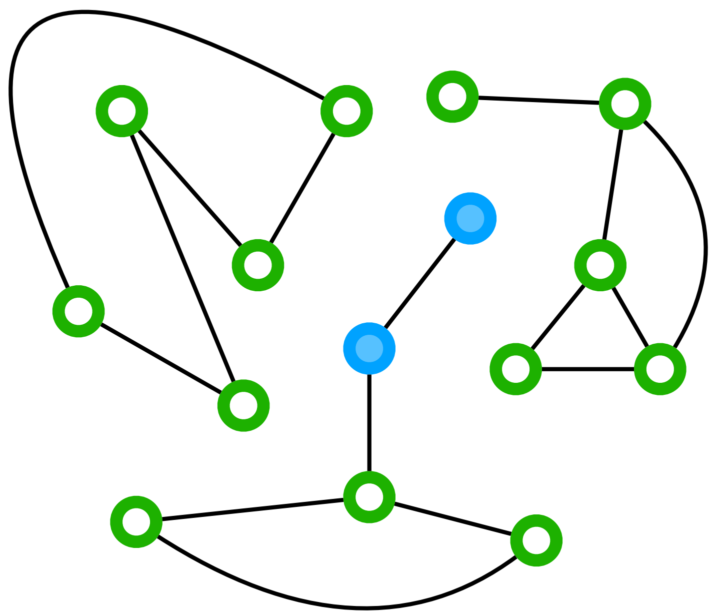
A Simple Solution?

- Run the following algorithm on s recursively, i.e., **DFS**(s)
- **DFS**(u):
  - Output u as part of the connected component
  - For $v \in N(u)$ recursively run **DFS**(v)

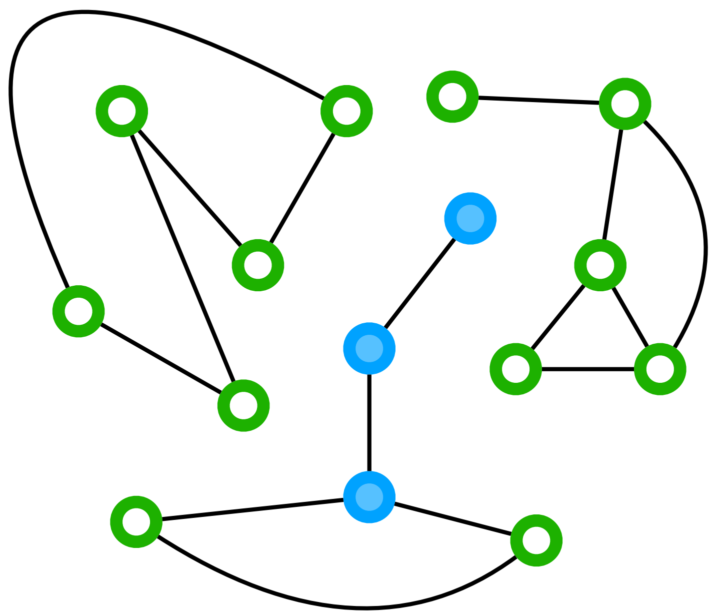
Example



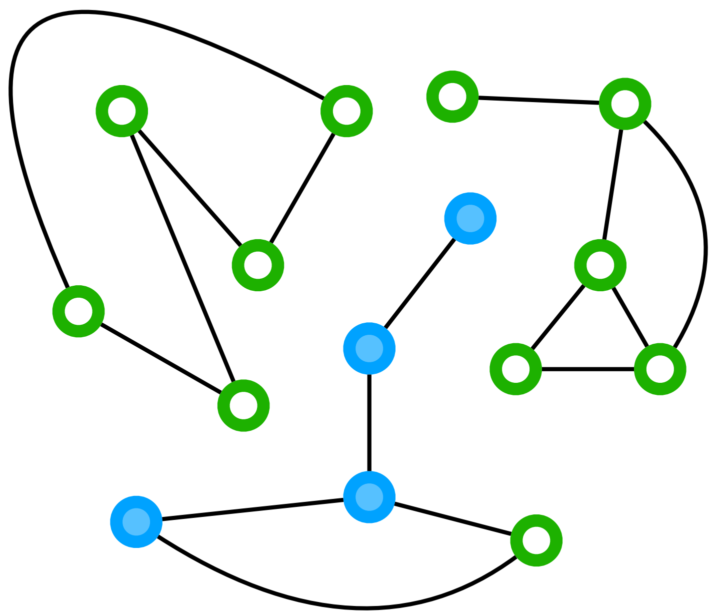
Example



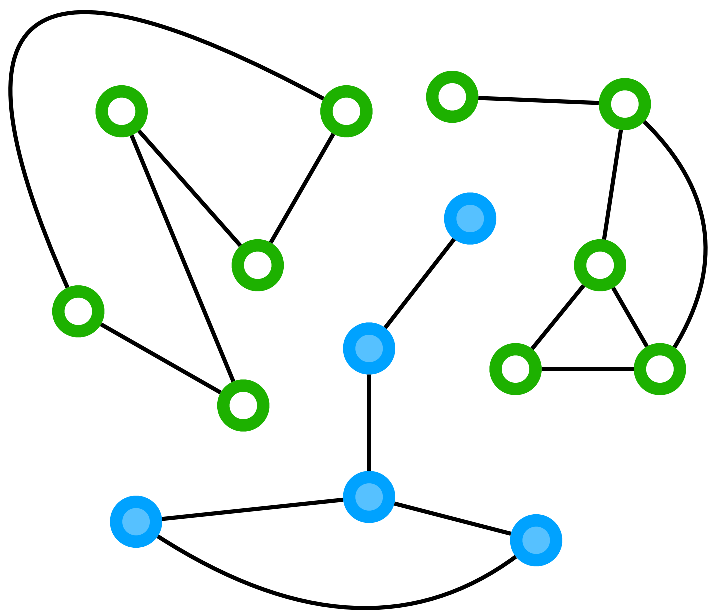
Example



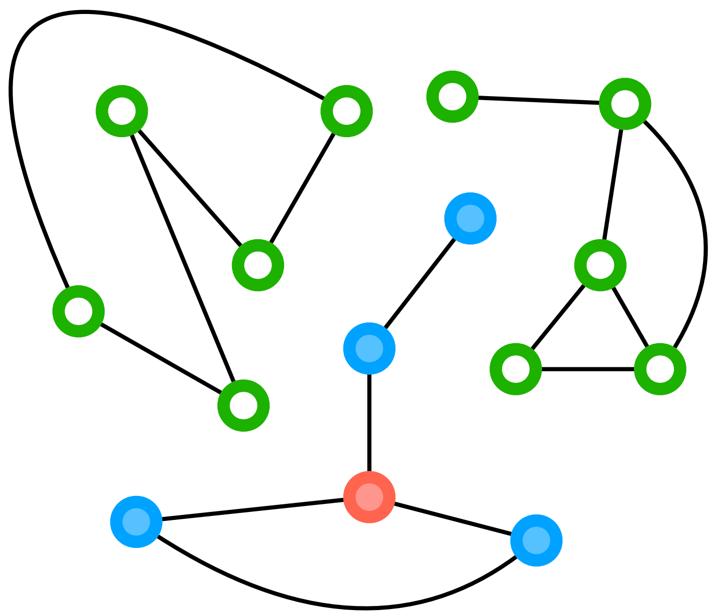
Example



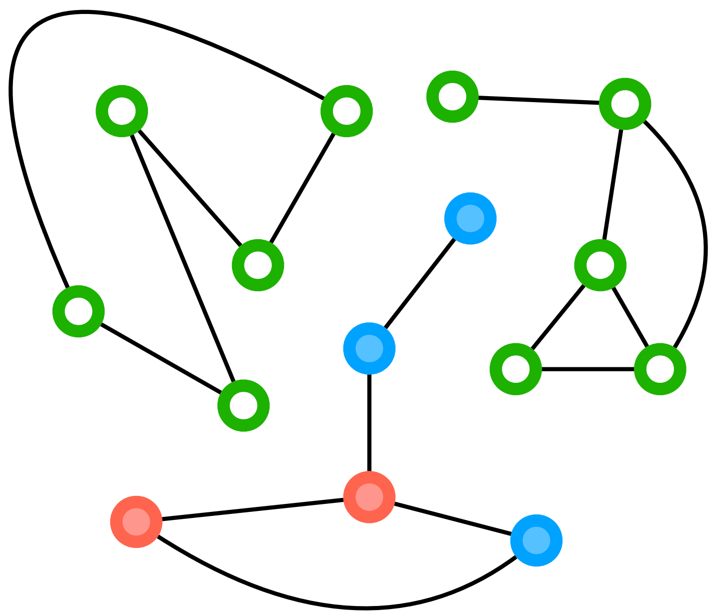
Example



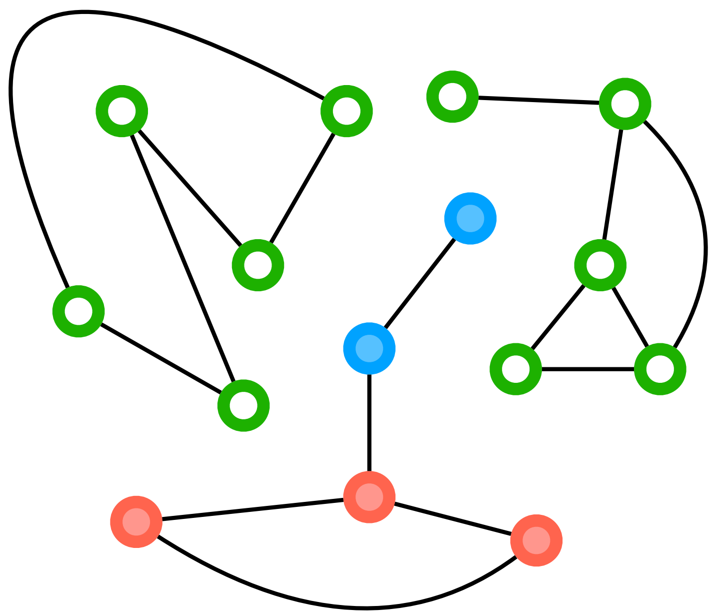
Example



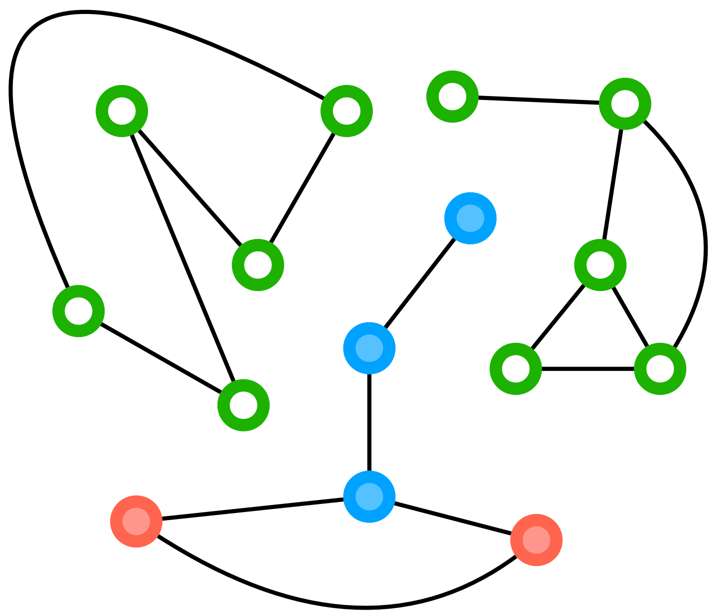
Example



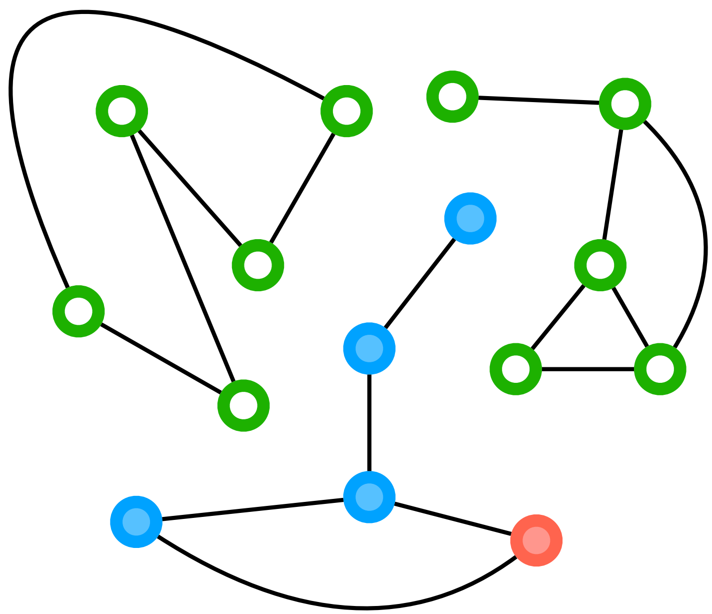
Example



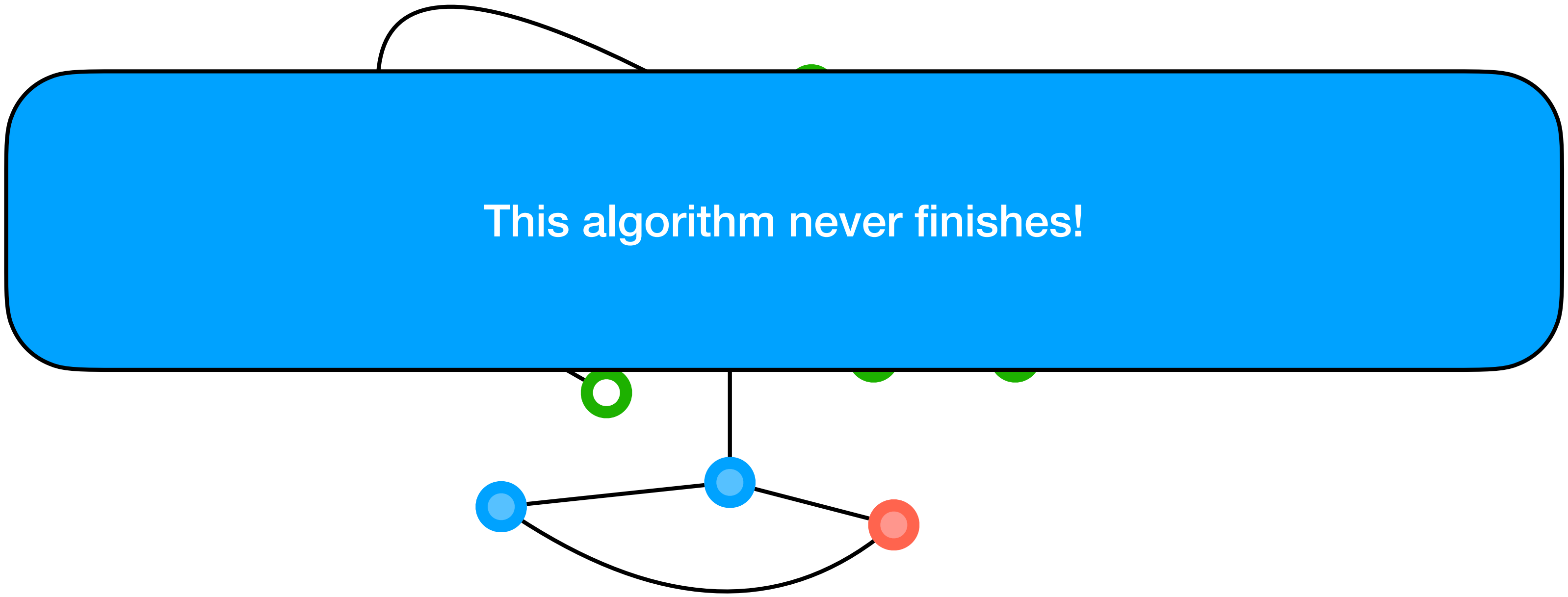
Example



Example



Example



Example

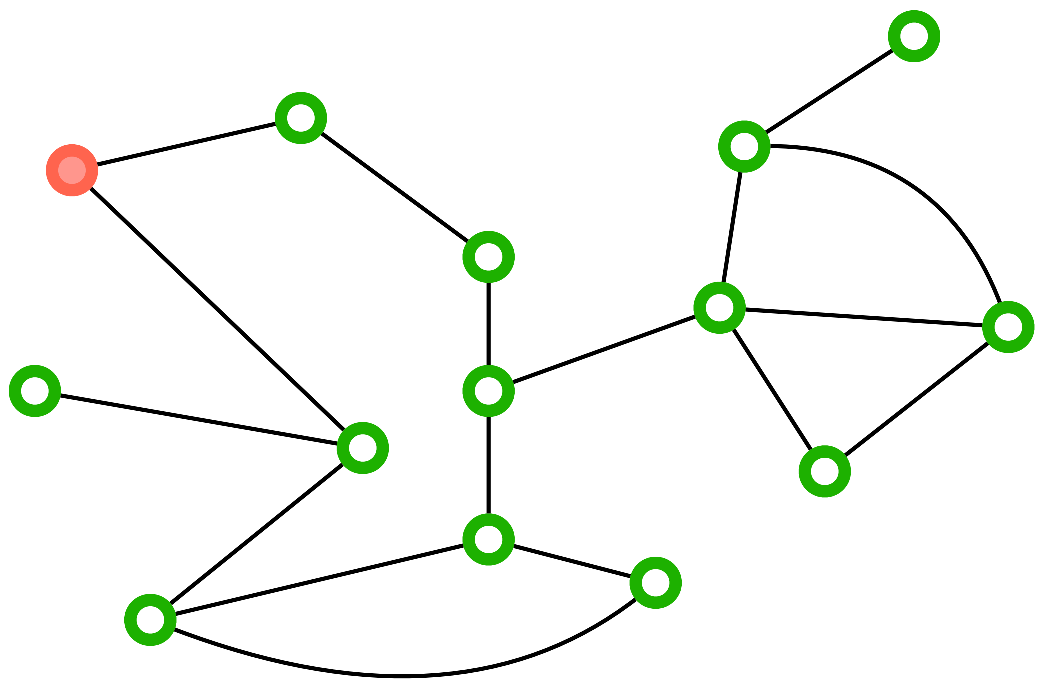
This algorithm never finishes!

Fix? Use Dynamic Programming!

The Actual DFS

- Create an array mark[1:n] = 'false' initially
- Run the following algorithm on s recursively, i.e., **DFS**(s)
- **DFS**(u):
  - If mark[u] = 'true', terminate; otherwise mark[u] = 'true'
  - For $v \in N(u)$ recursively run **DFS**(v)
- Output all marked vertices as the connected component of s

DFS: Example

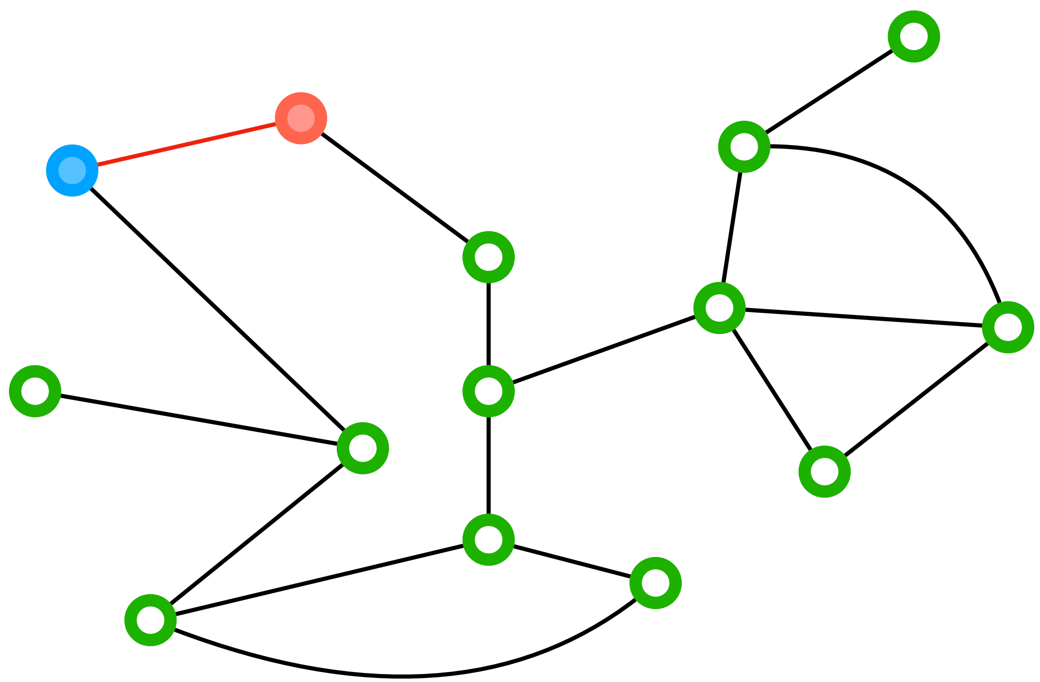


marked vertices

unmarked vertices

current recursive call

DFS: Example

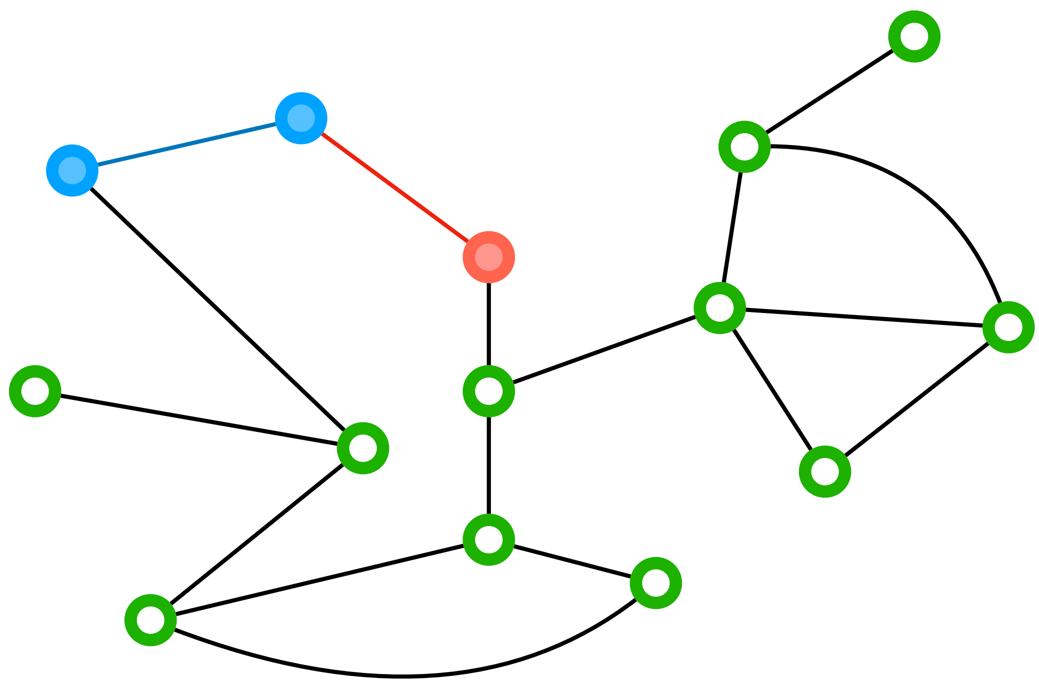


marked vertices

unmarked vertices

current recursive call

DFS: Example

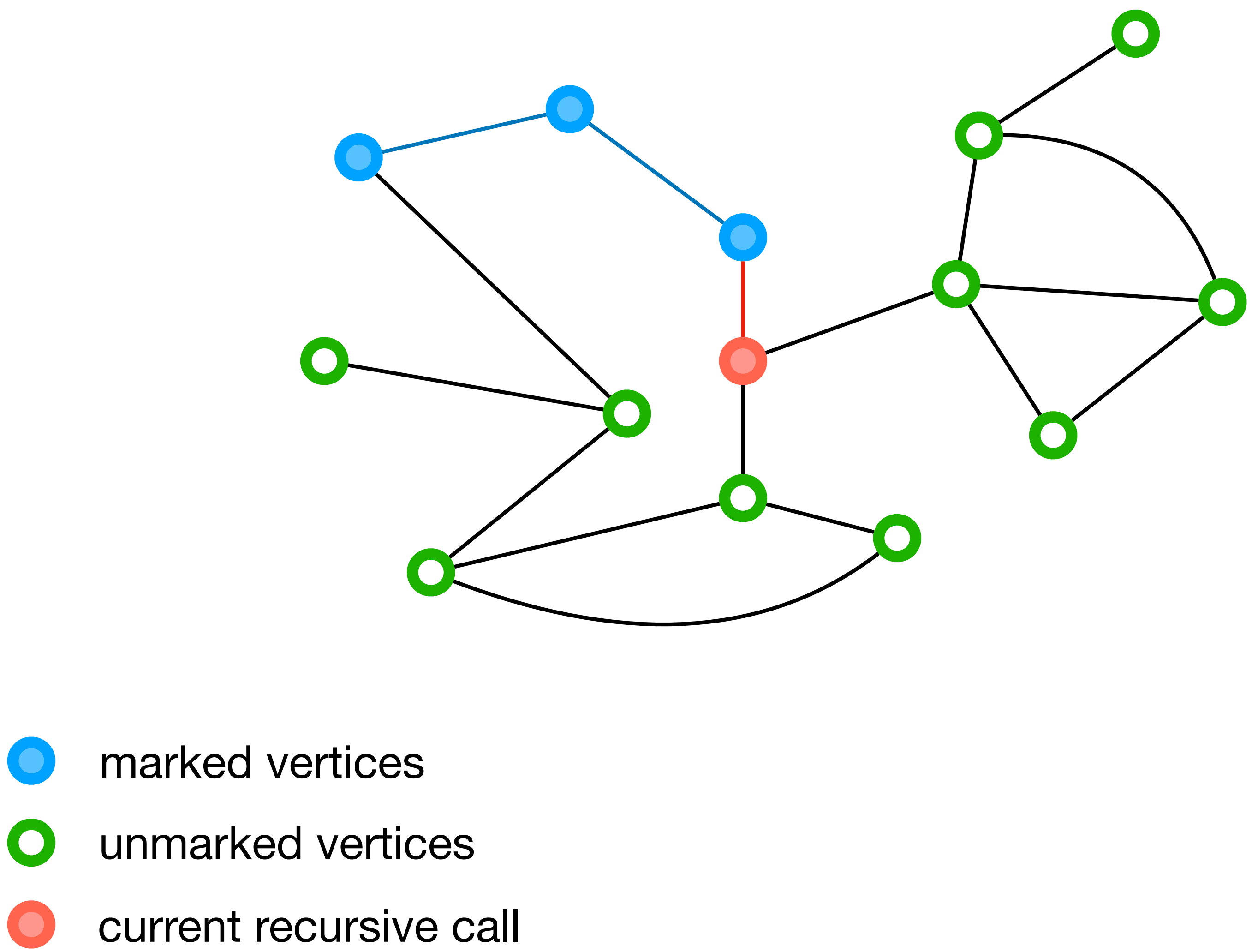


marked vertices

unmarked vertices

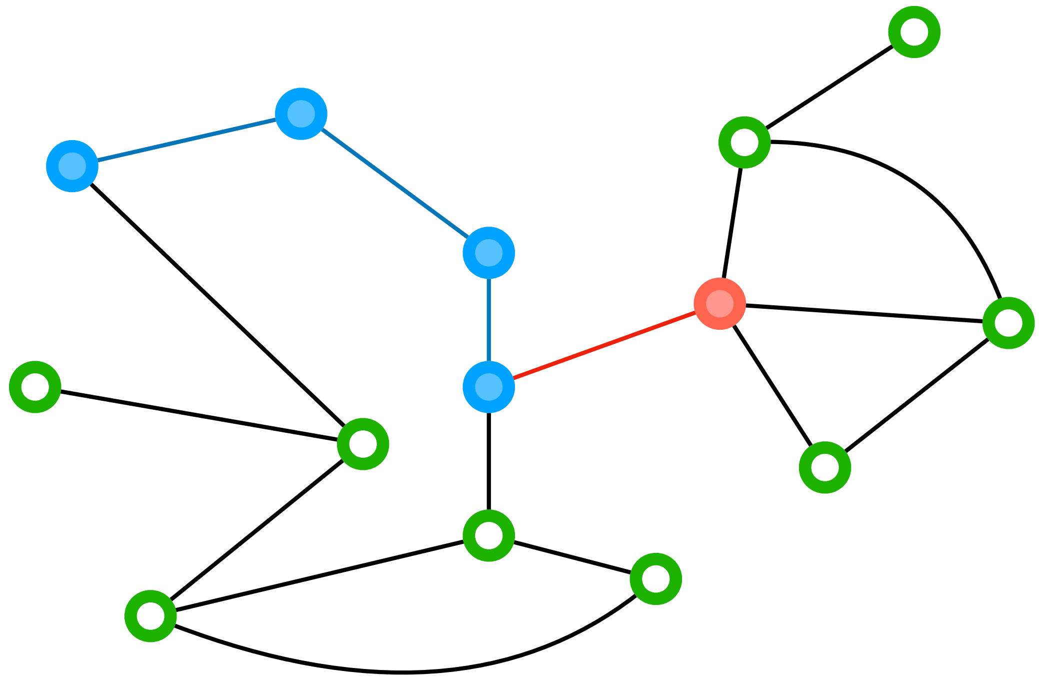
current recursive call

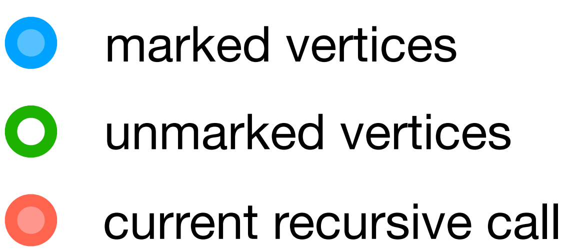
DFS: Example



marked vertices
unmarked vertices
current recursive call

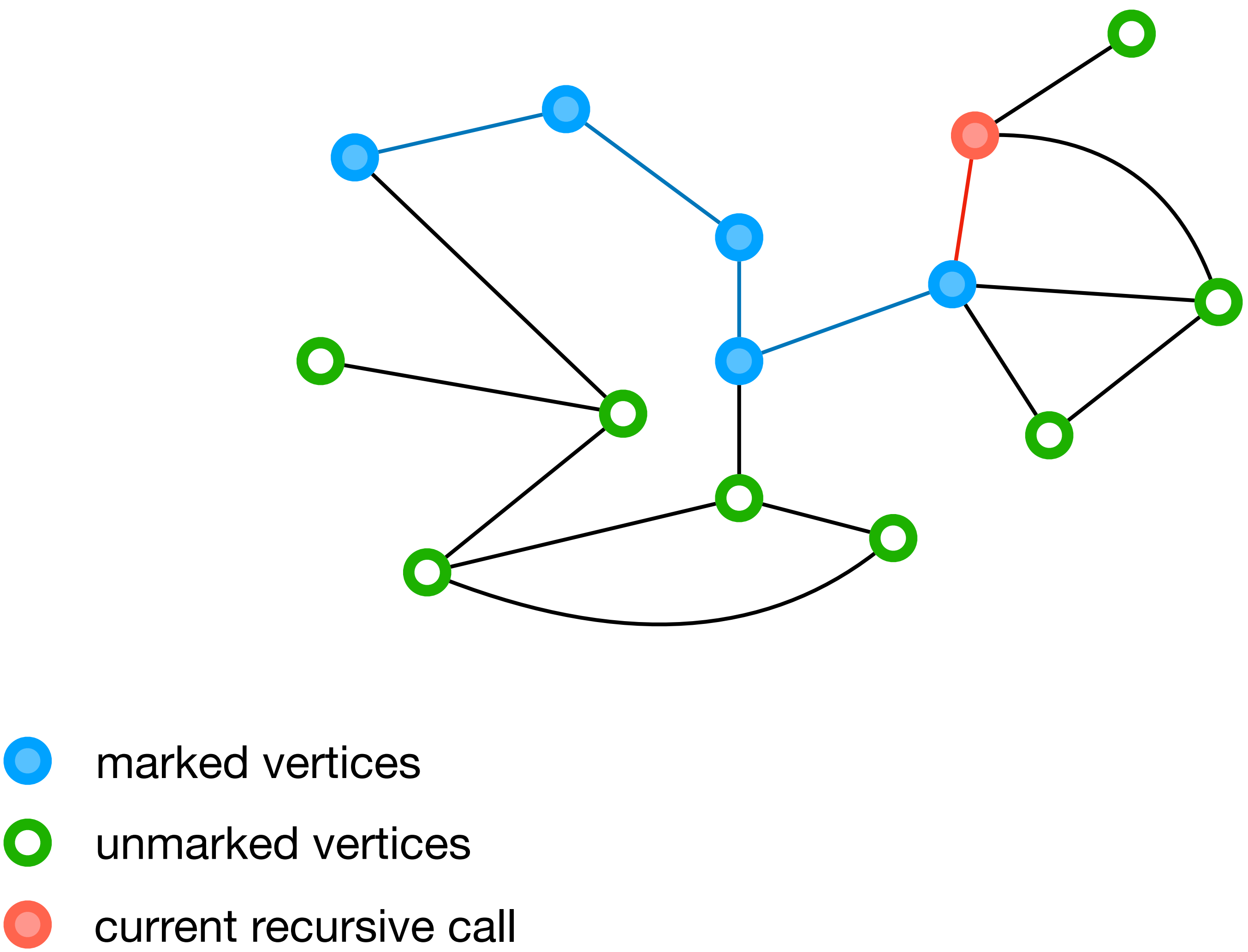
DFS: Example





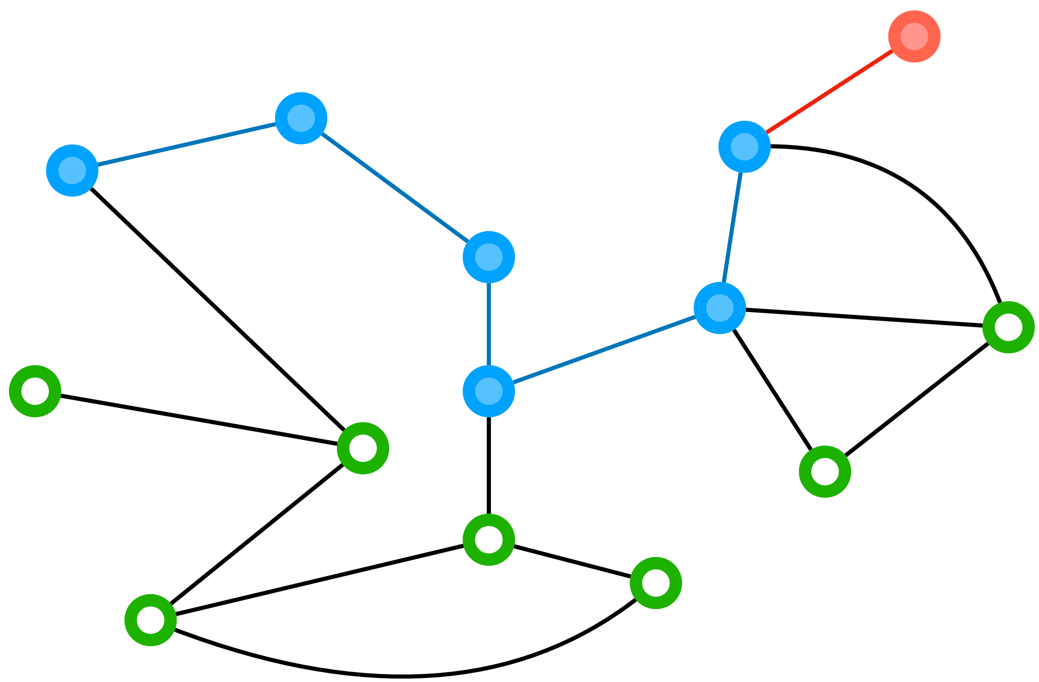
marked vertices
unmarked vertices
current recursive call

DFS: Example



marked vertices
unmarked vertices
current recursive call

DFS: Example

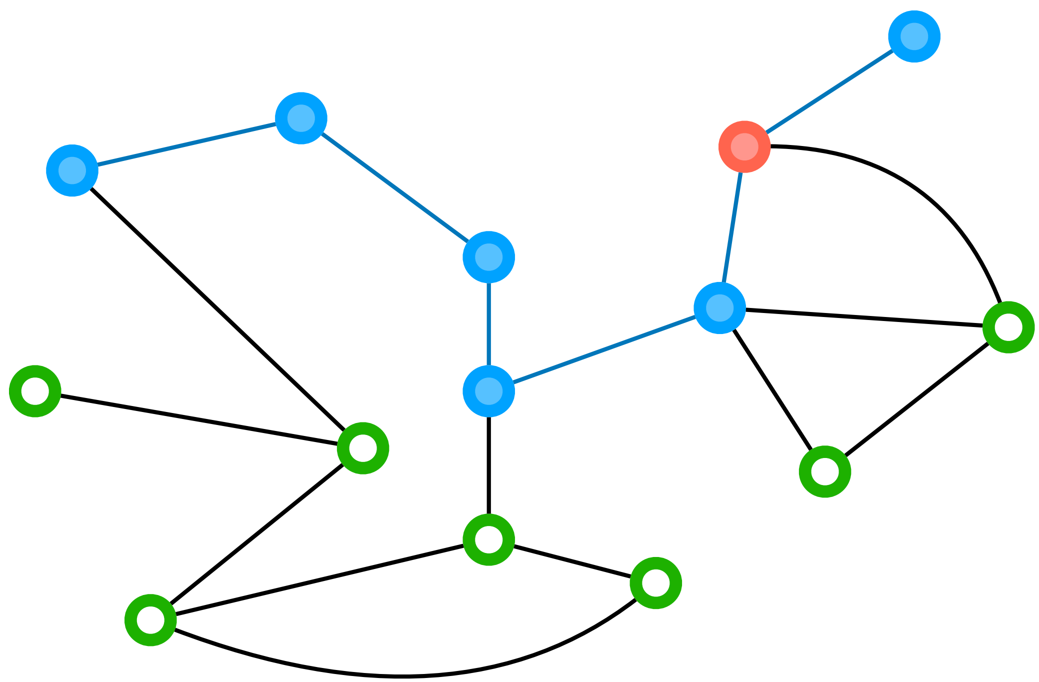


marked vertices

unmarked vertices

current recursive call

DFS: Example

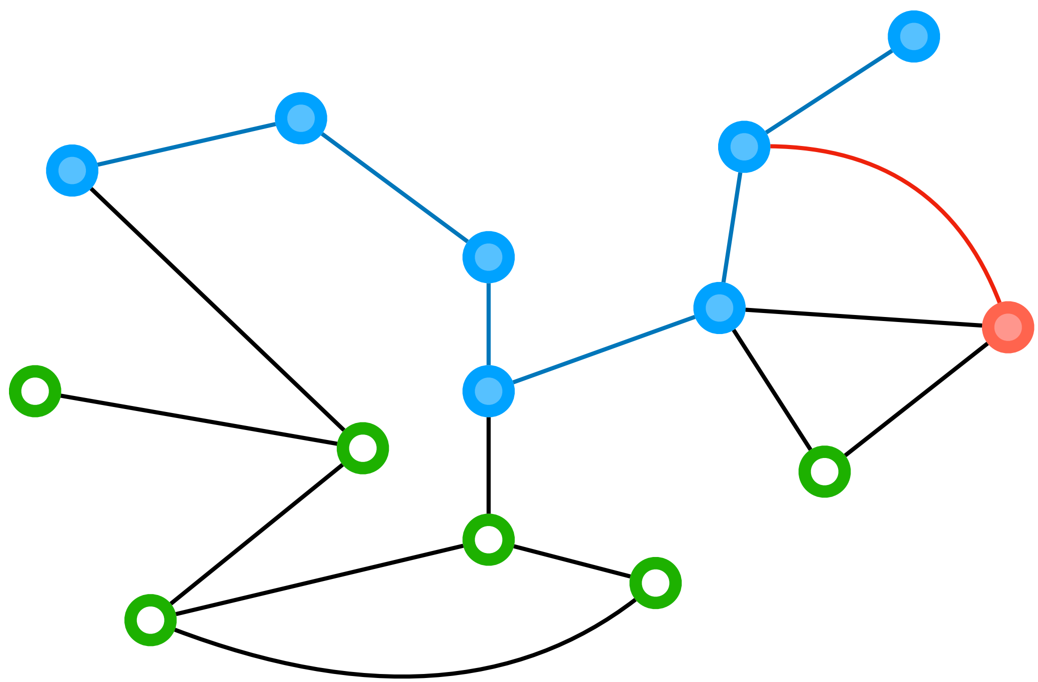


marked vertices

unmarked vertices

current recursive call

DFS: Example

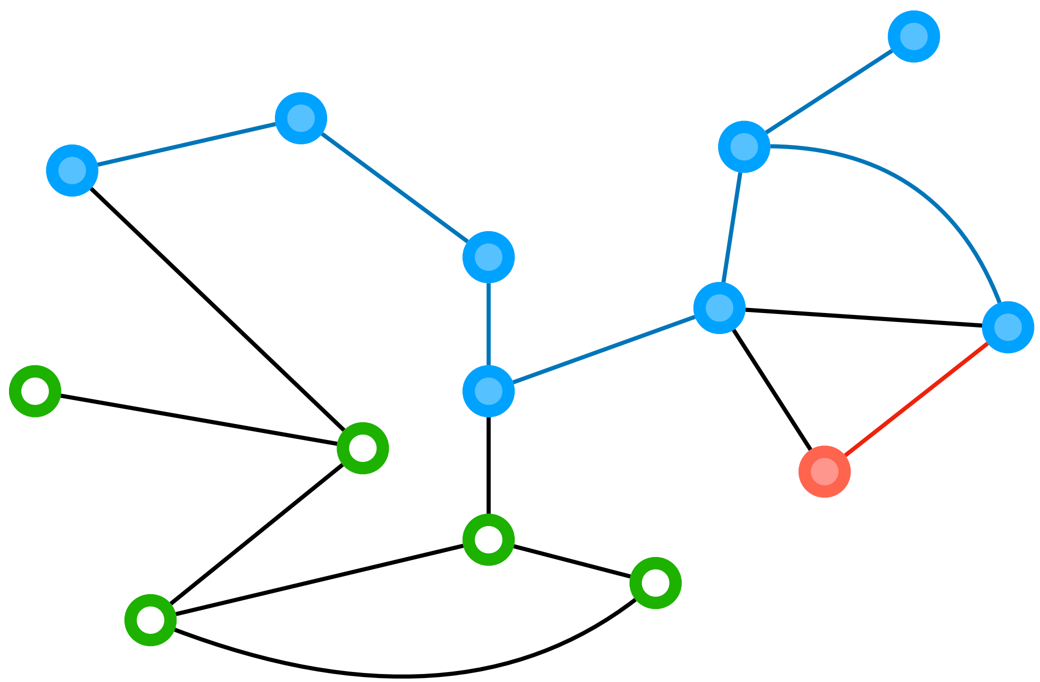


marked vertices

unmarked vertices

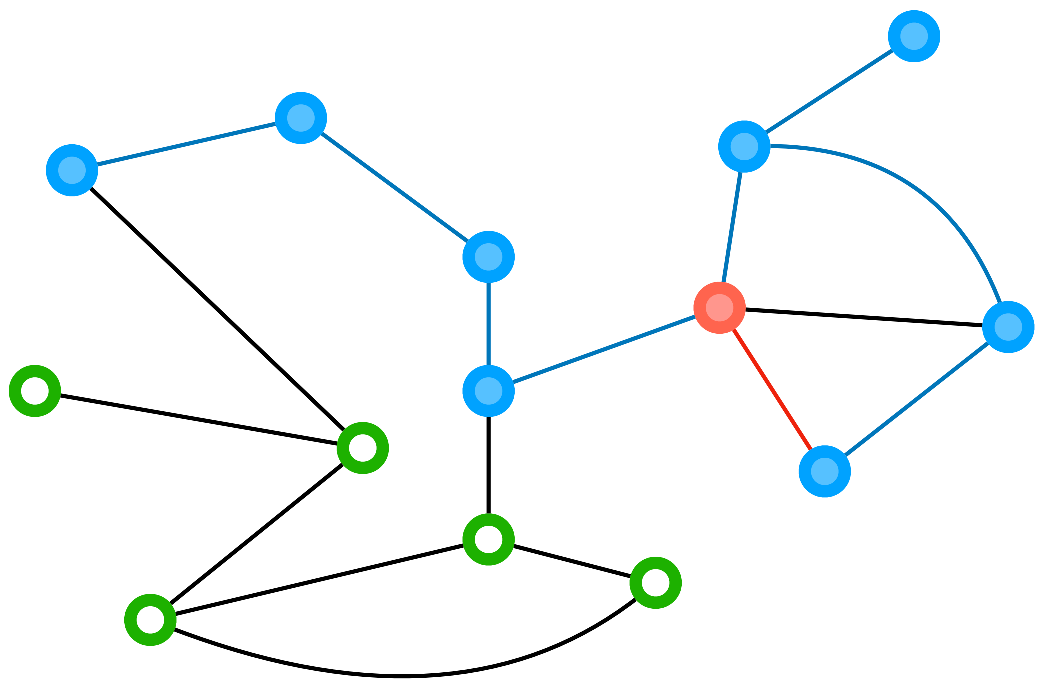
current recursive call

DFS: Example



- marked vertices
- unmarked vertices
- current recursive call

DFS: Example

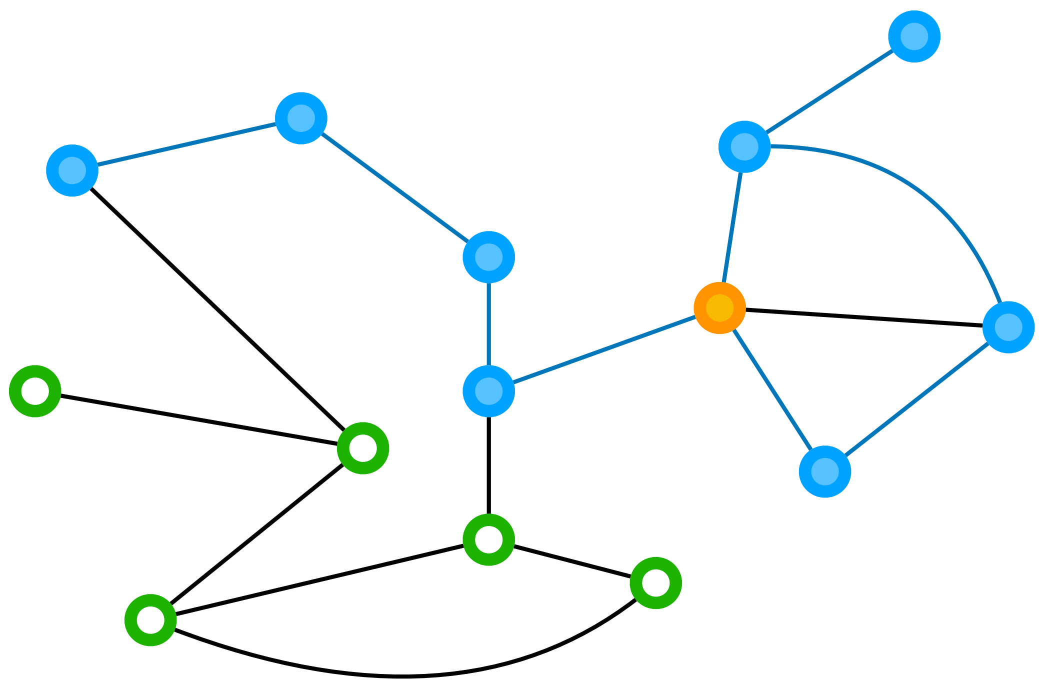


marked vertices

unmarked vertices

current recursive call

DFS: Example



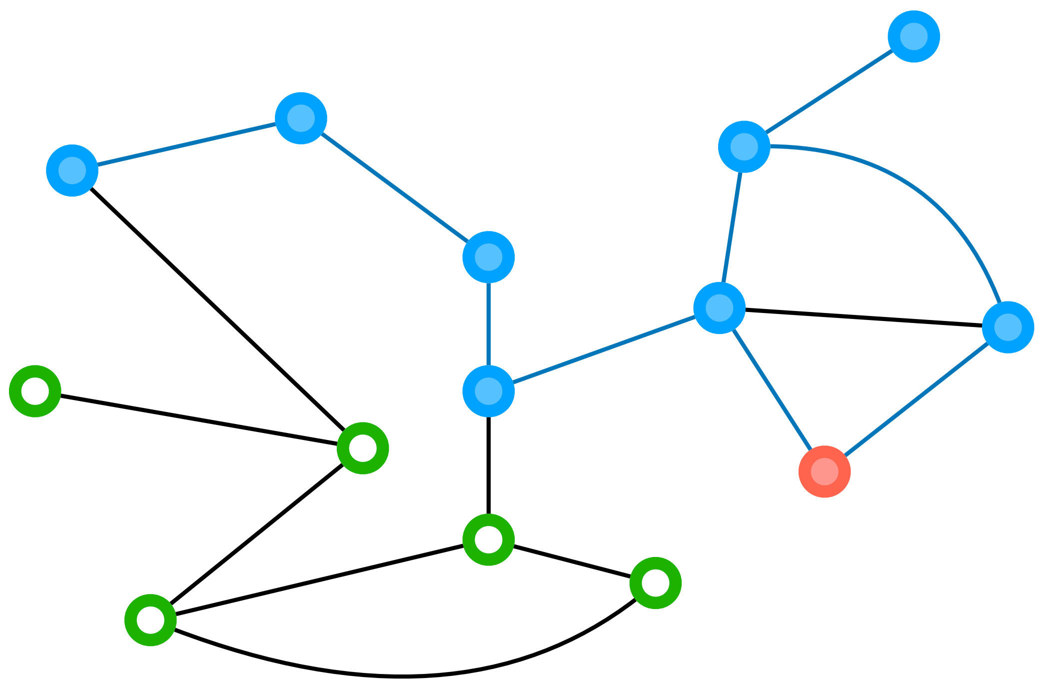
marked vertices

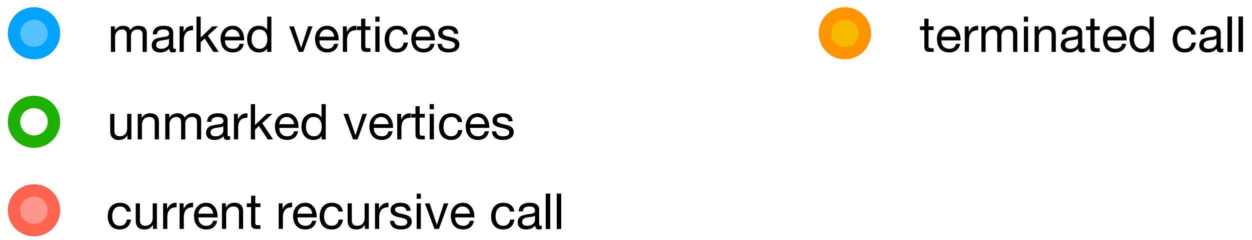
unmarked vertices

current recursive call

terminated call

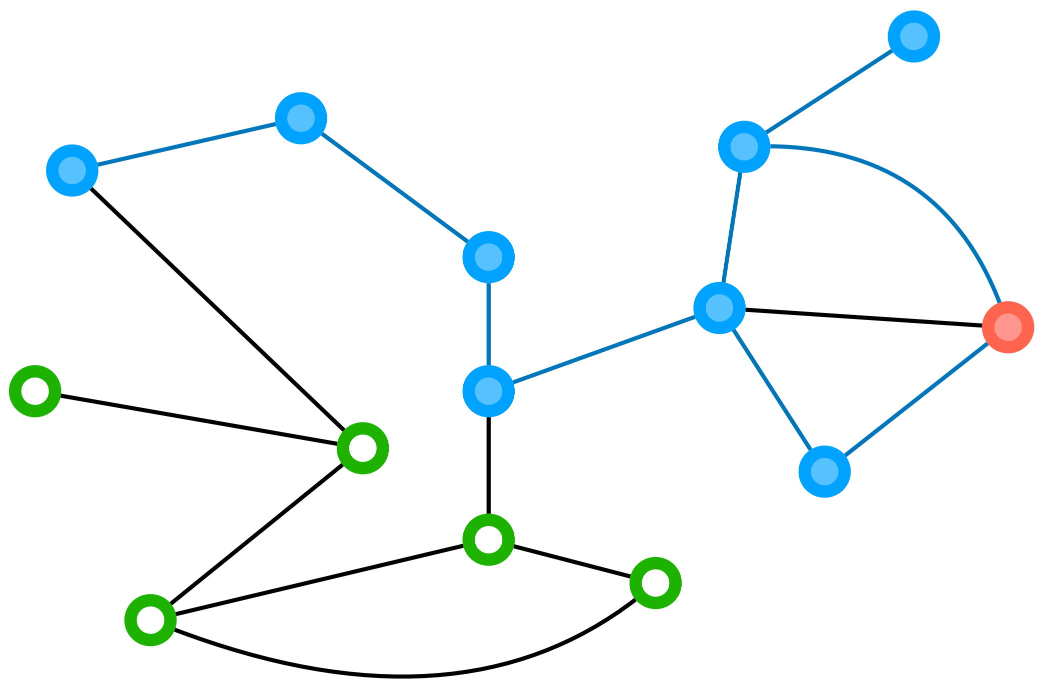
DFS: Example





marked vertices
unmarked vertices
current recursive call
terminated call

DFS: Example



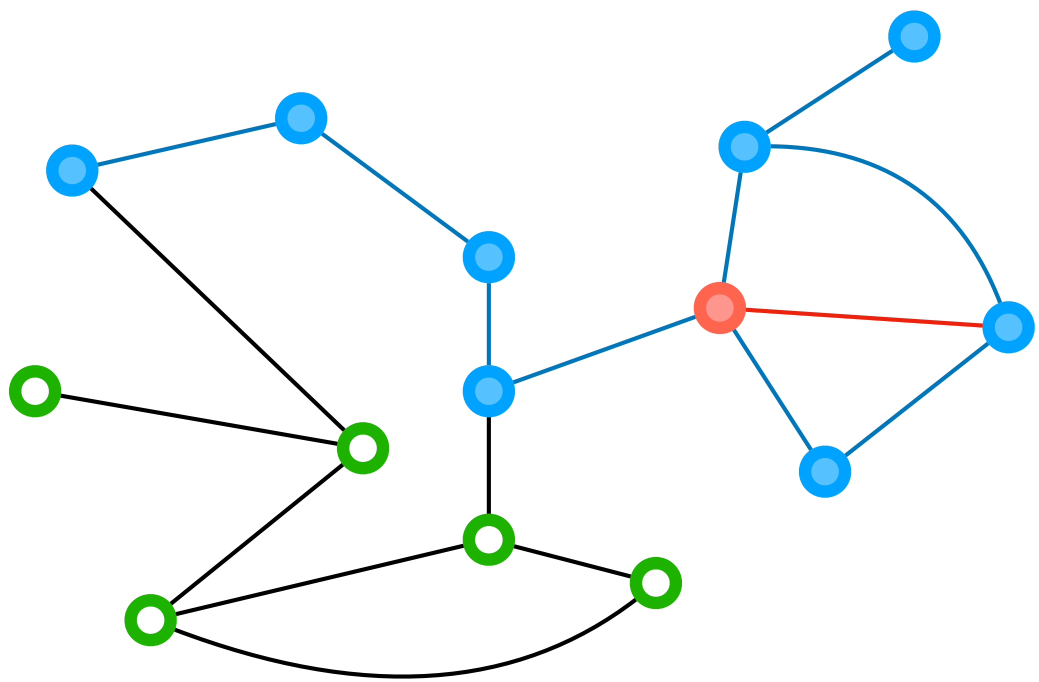
marked vertices

unmarked vertices

current recursive call

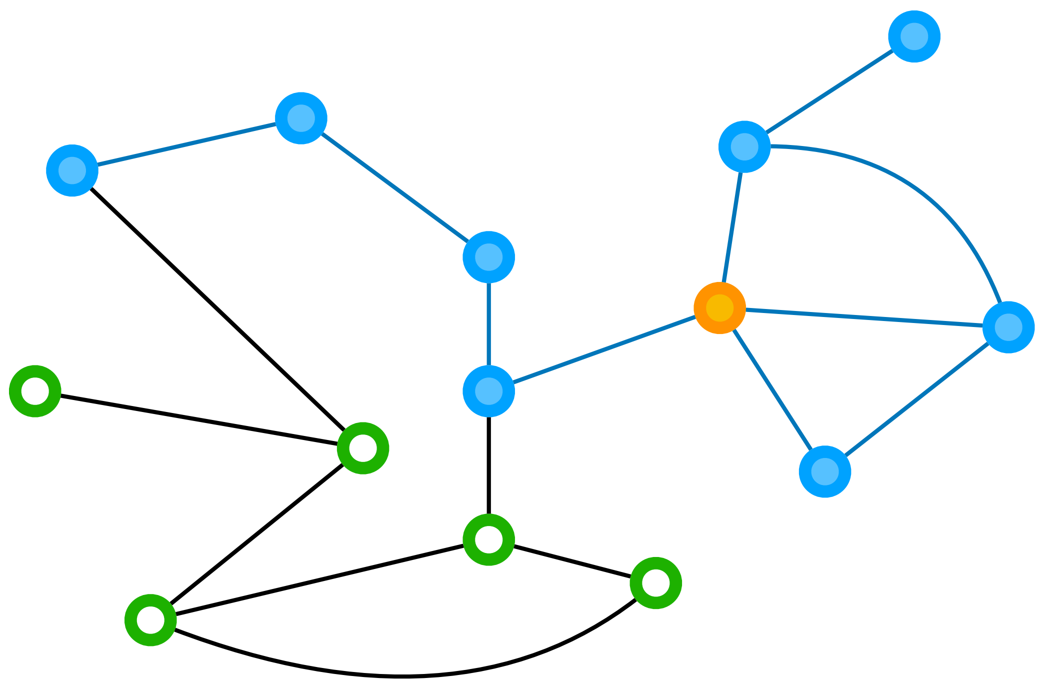
terminated call

DFS: Example



marked vertices
unmarked vertices
current recursive call
terminated call

DFS: Example



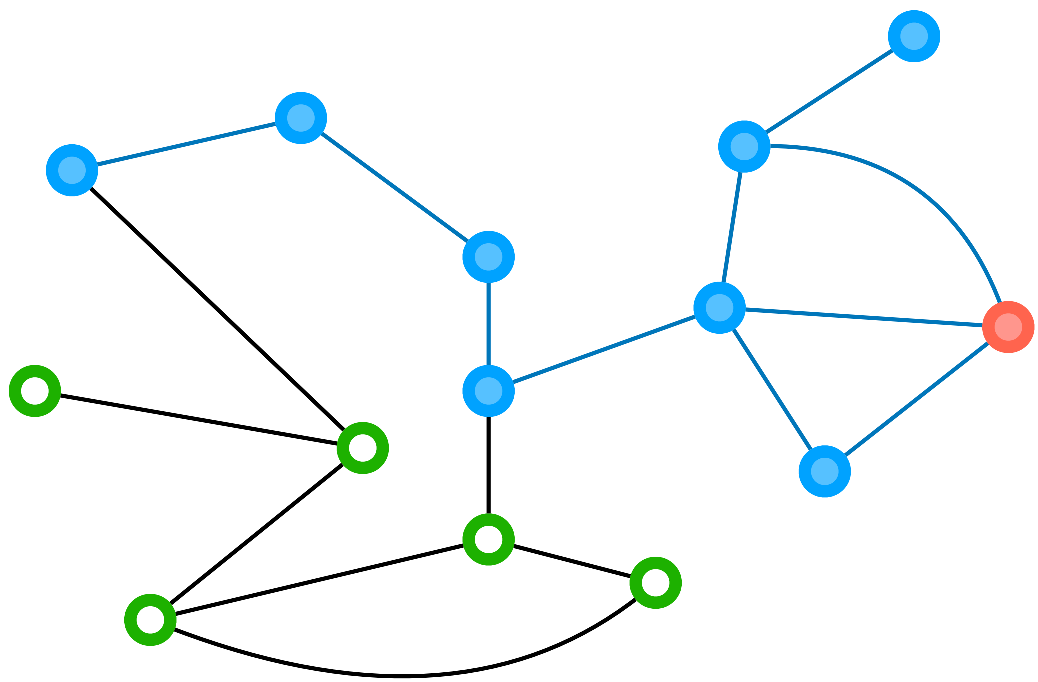
marked vertices

unmarked vertices

current recursive call

terminated call

DFS: Example



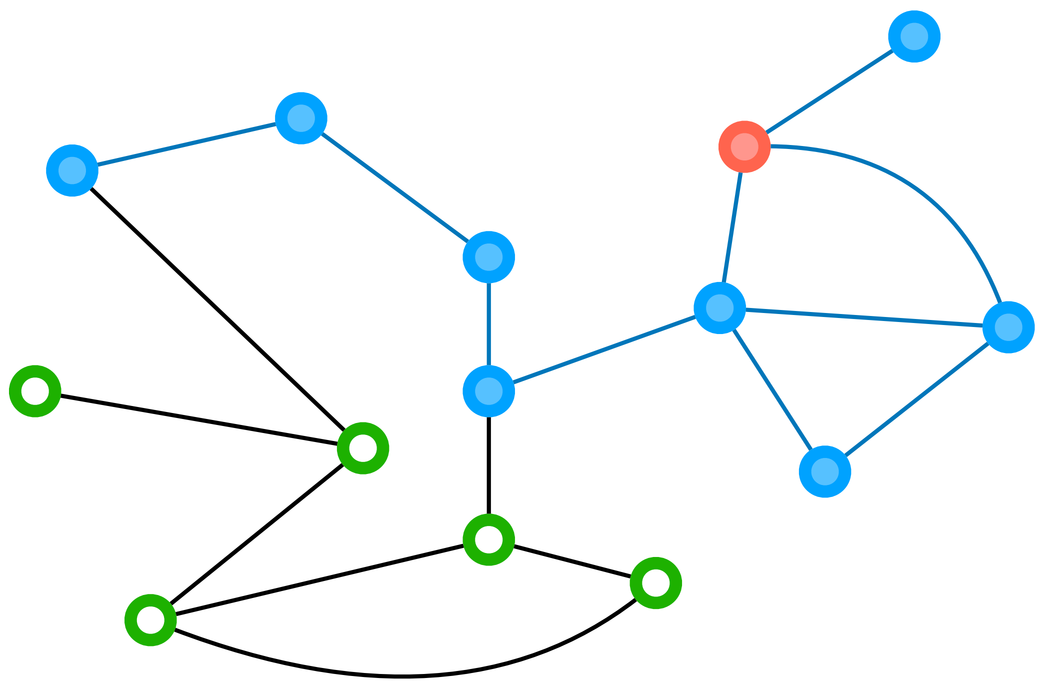
marked vertices

unmarked vertices

current recursive call

terminated call

DFS: Example



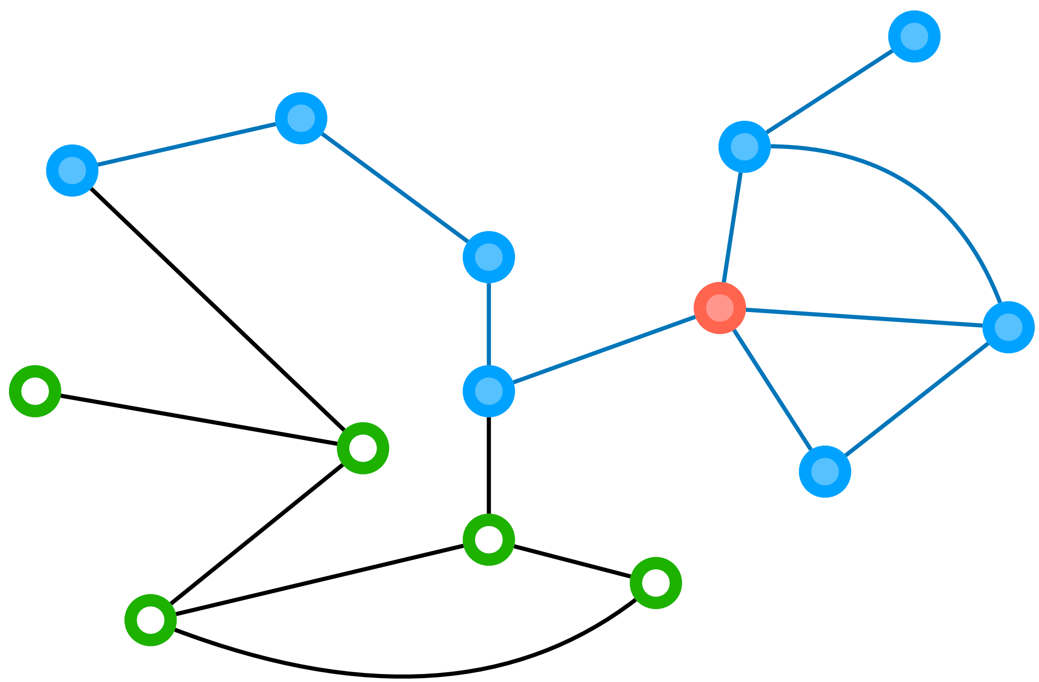
marked vertices

unmarked vertices

current recursive call

terminated call

DFS: Example



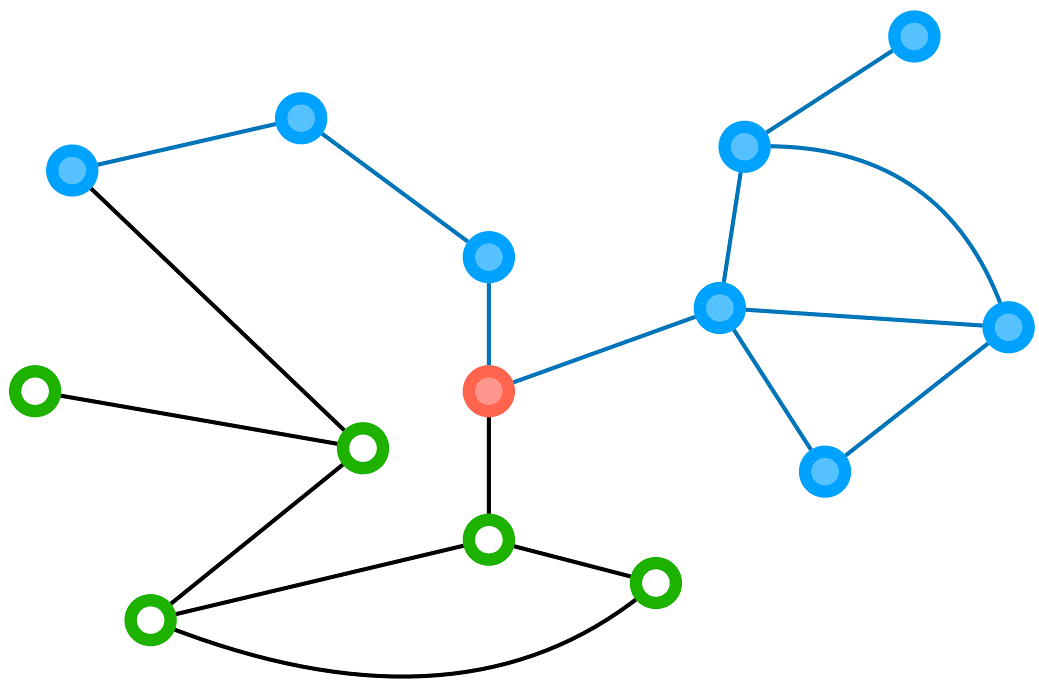
marked vertices

unmarked vertices

current recursive call

terminated call

DFS: Example



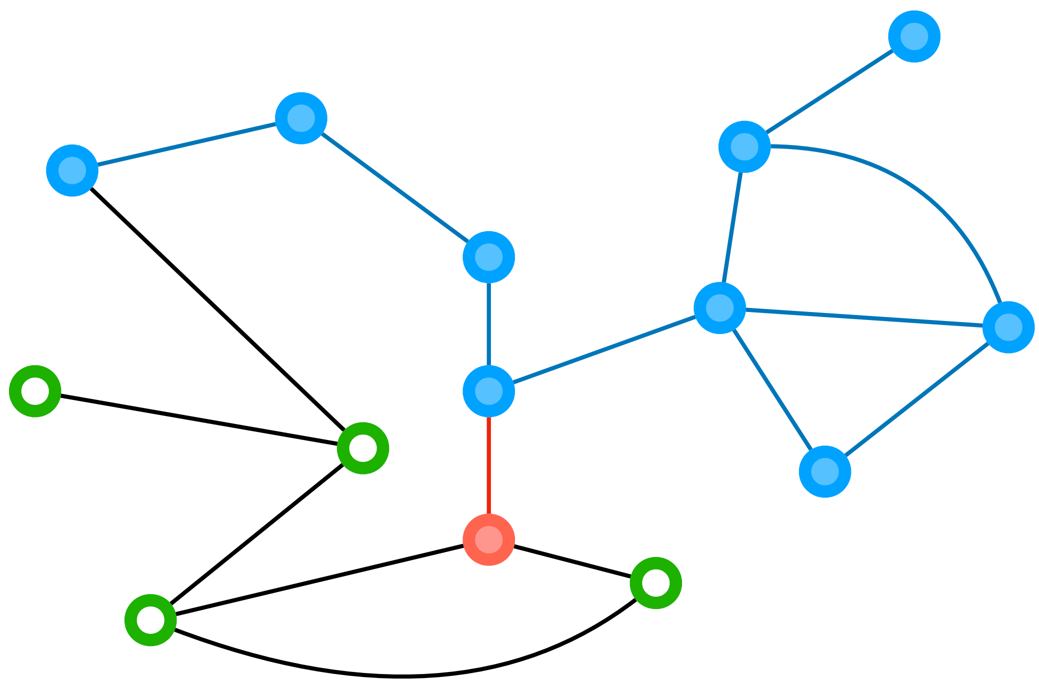
marked vertices

unmarked vertices

current recursive call

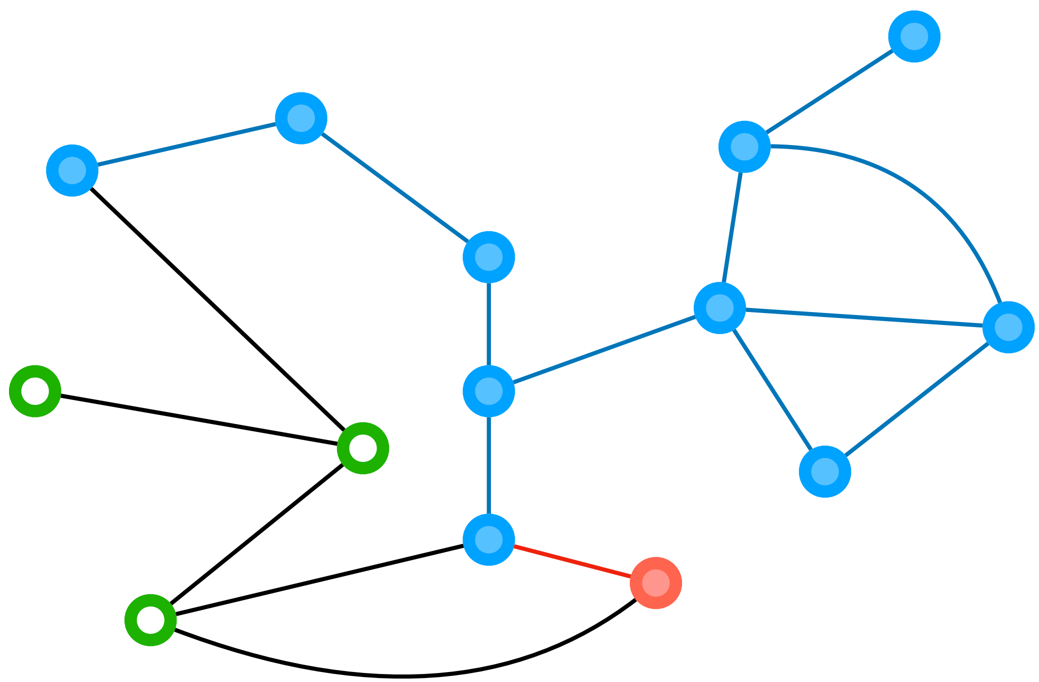
terminated call

DFS: Example



- marked vertices
- unmarked vertices
- current recursive call
- terminated call

DFS: Example



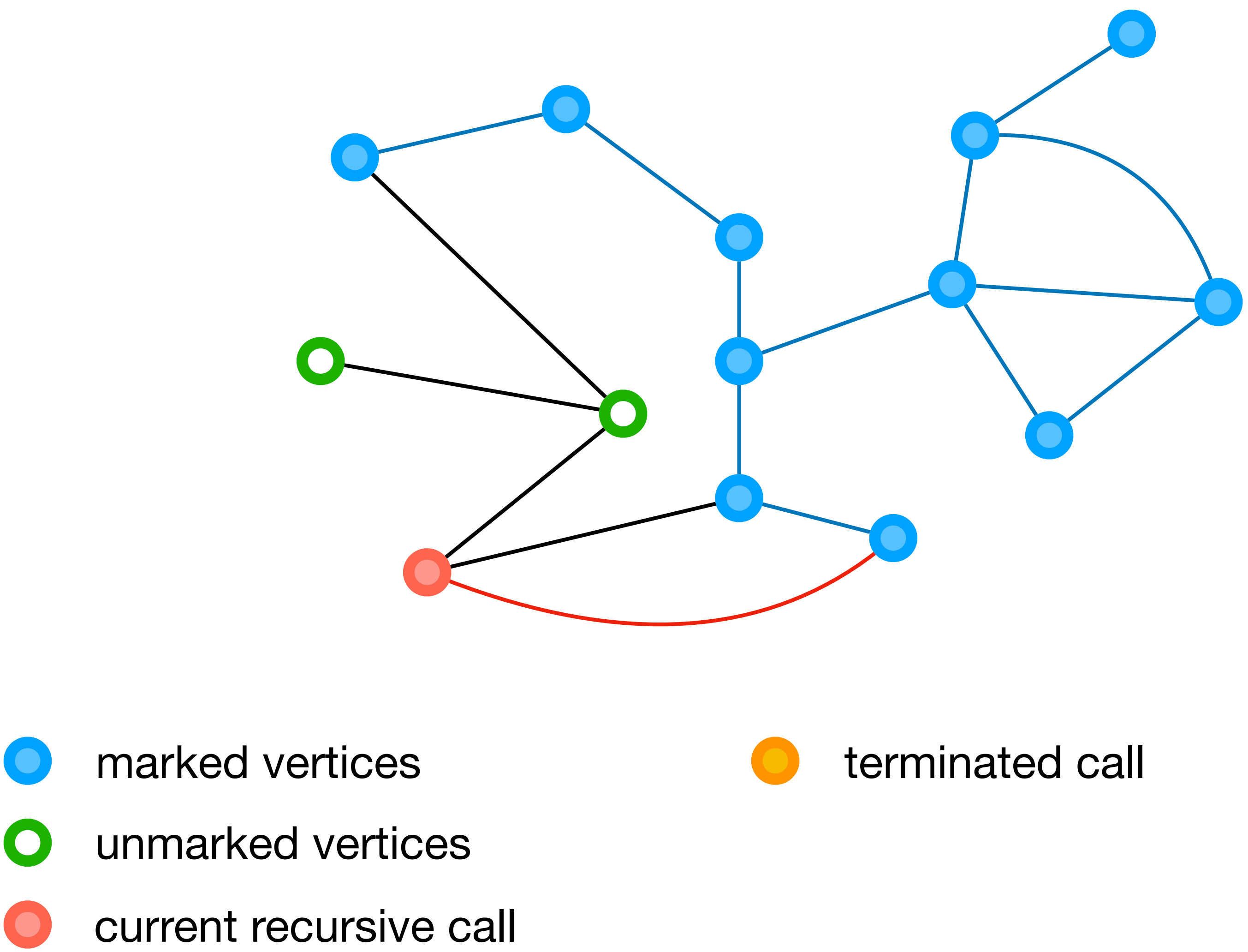
marked vertices

unmarked vertices

current recursive call

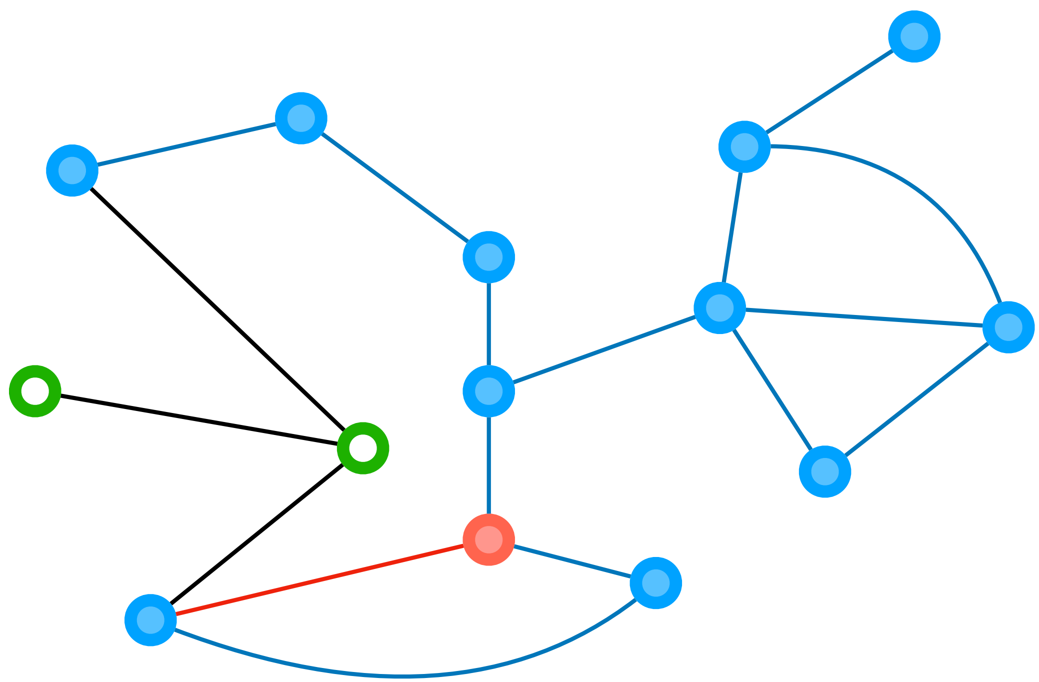
terminated call

DFS: Example



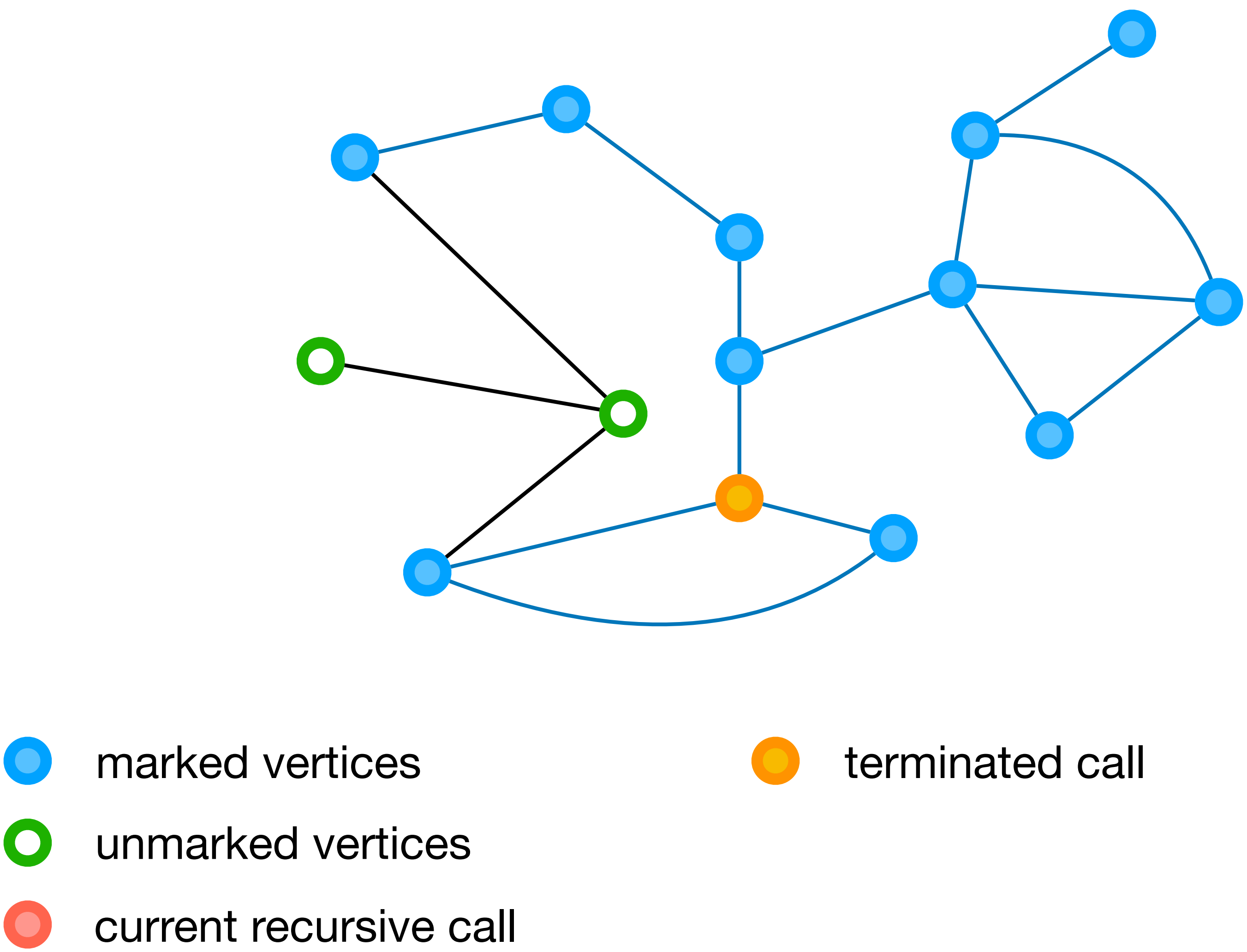
marked vertices
unmarked vertices
current recursive call
terminated call

DFS: Example



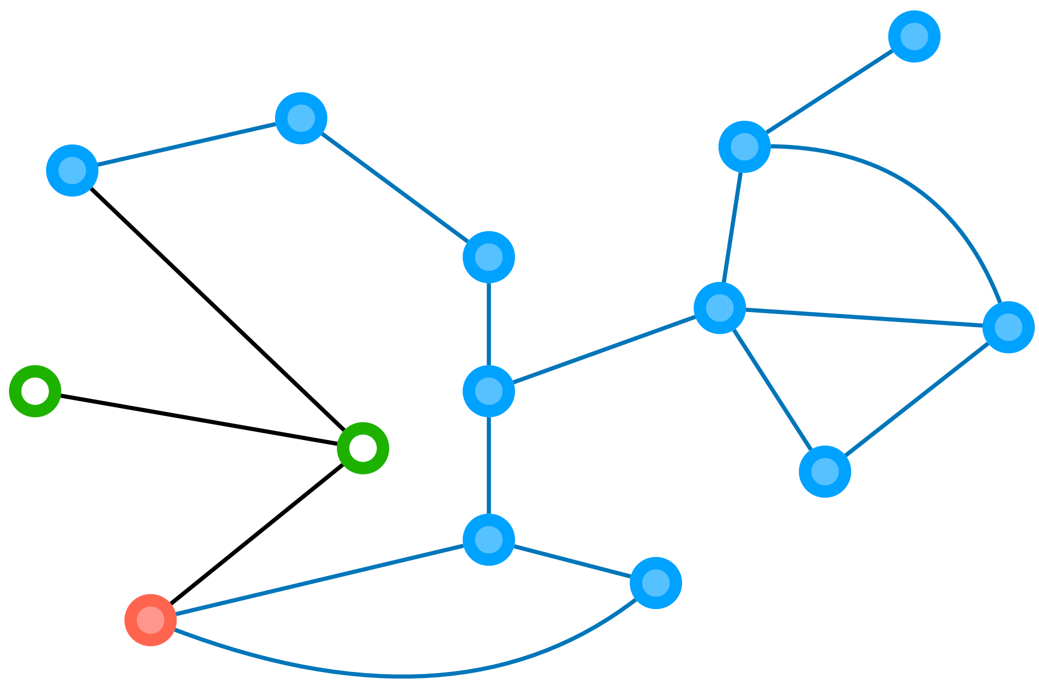
- marked vertices
- unmarked vertices
- current recursive call
- terminated call

DFS: Example



marked vertices
unmarked vertices
current recursive call
terminated call

DFS: Example



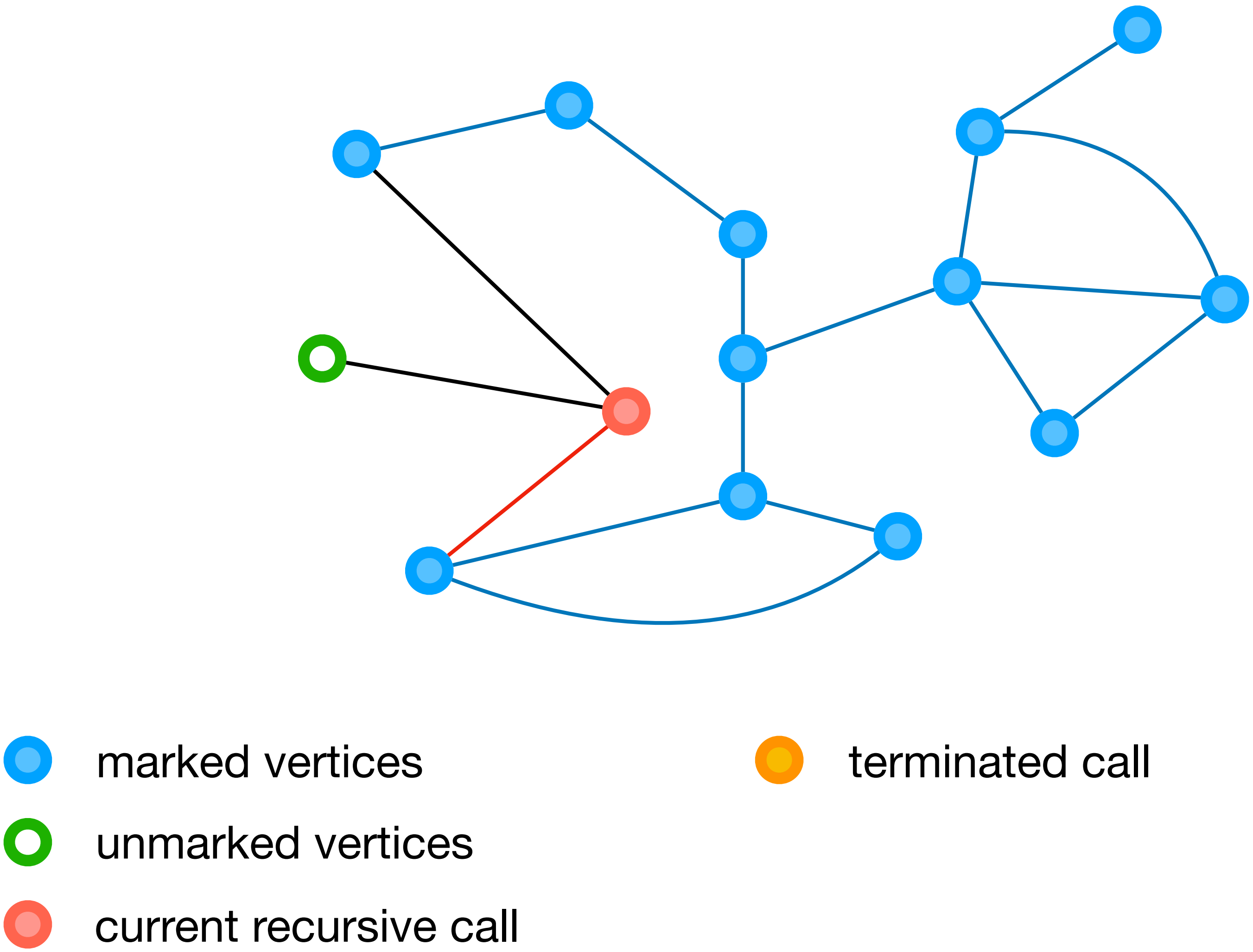
marked vertices

unmarked vertices

current recursive call

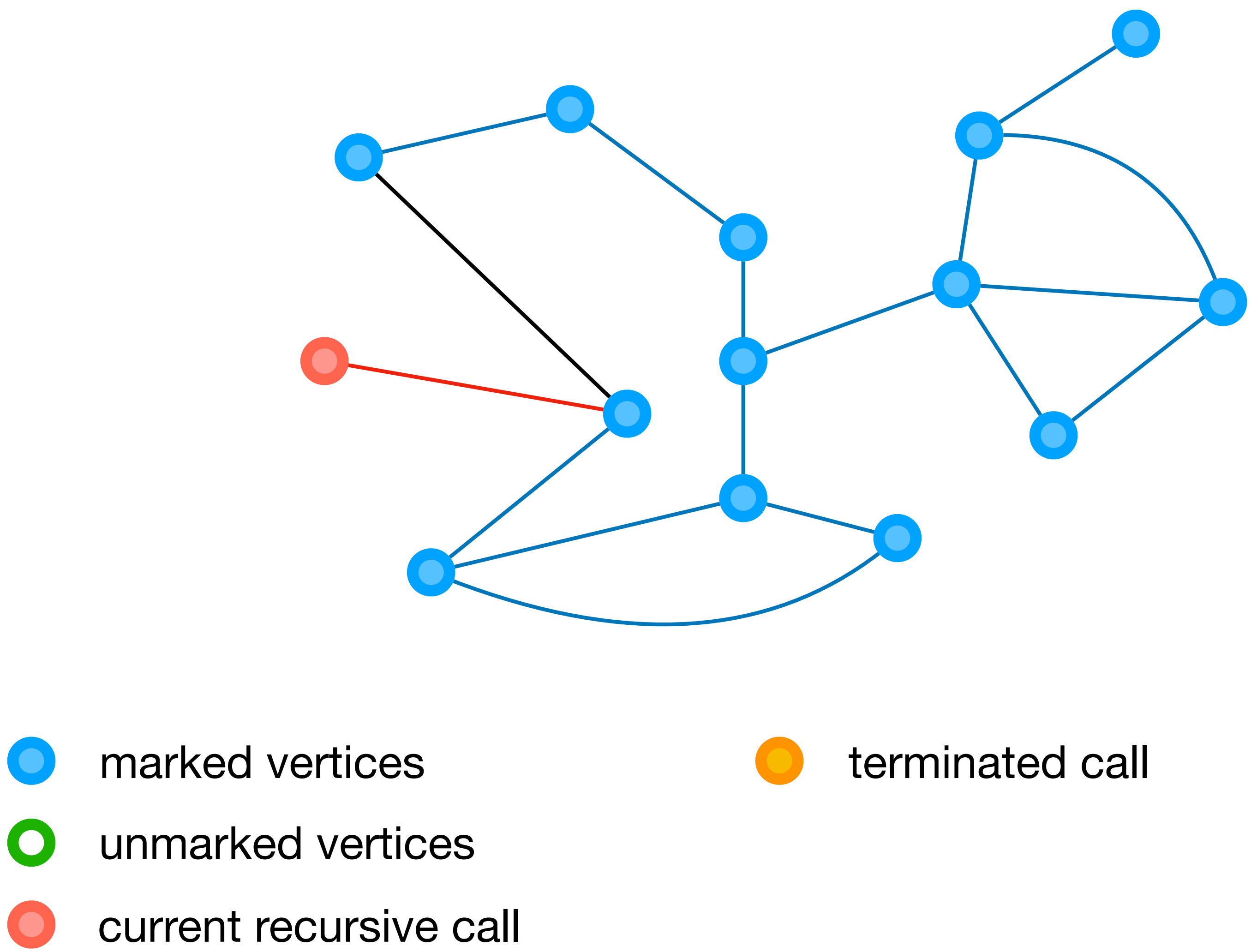
terminated call

DFS: Example



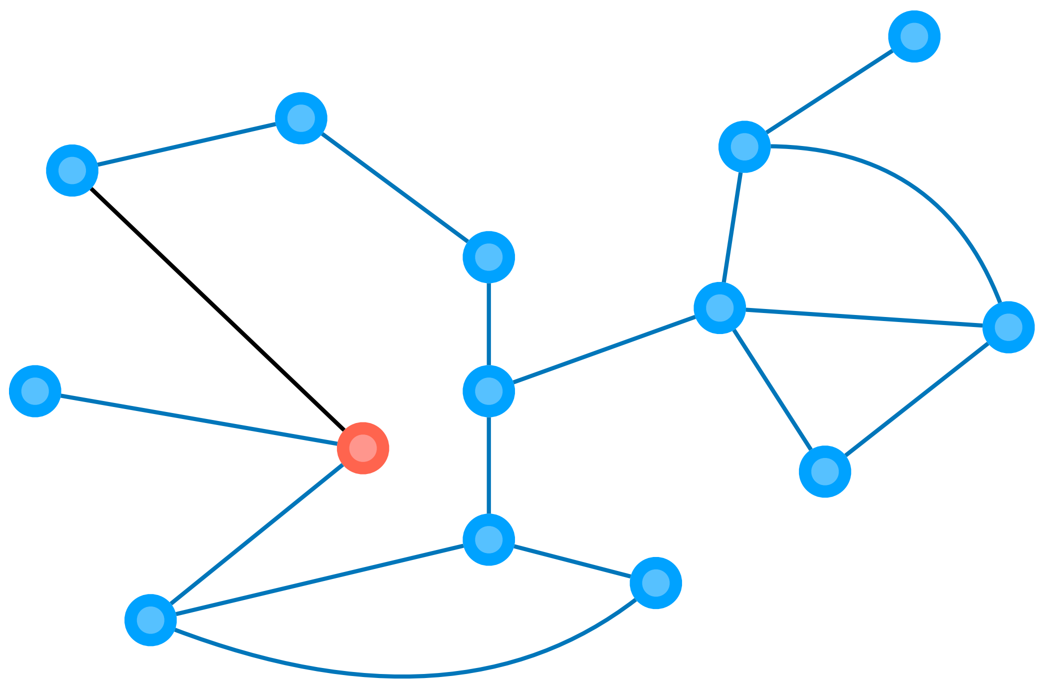
marked vertices
unmarked vertices
current recursive call
terminated call

DFS: Example



marked vertices
unmarked vertices
current recursive call
terminated call

DFS: Example



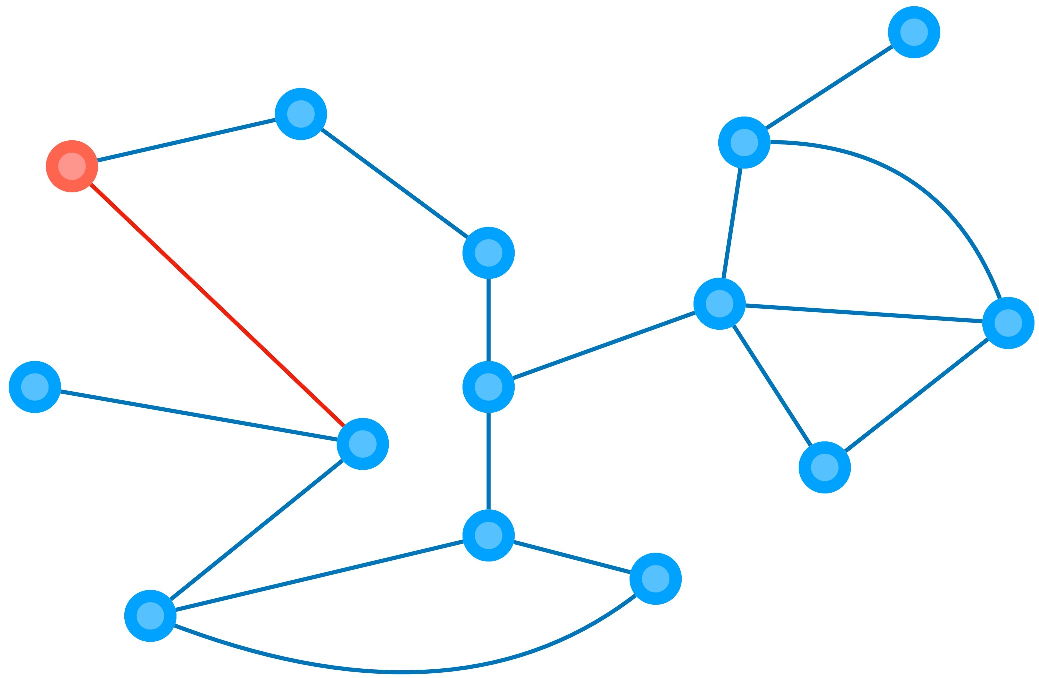
marked vertices

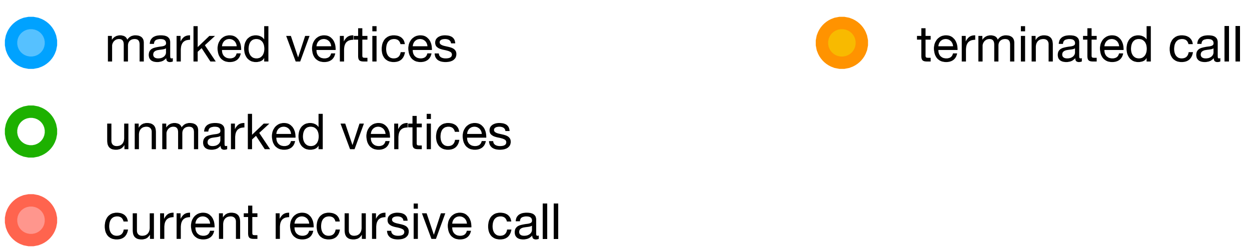
unmarked vertices

current recursive call

terminated call

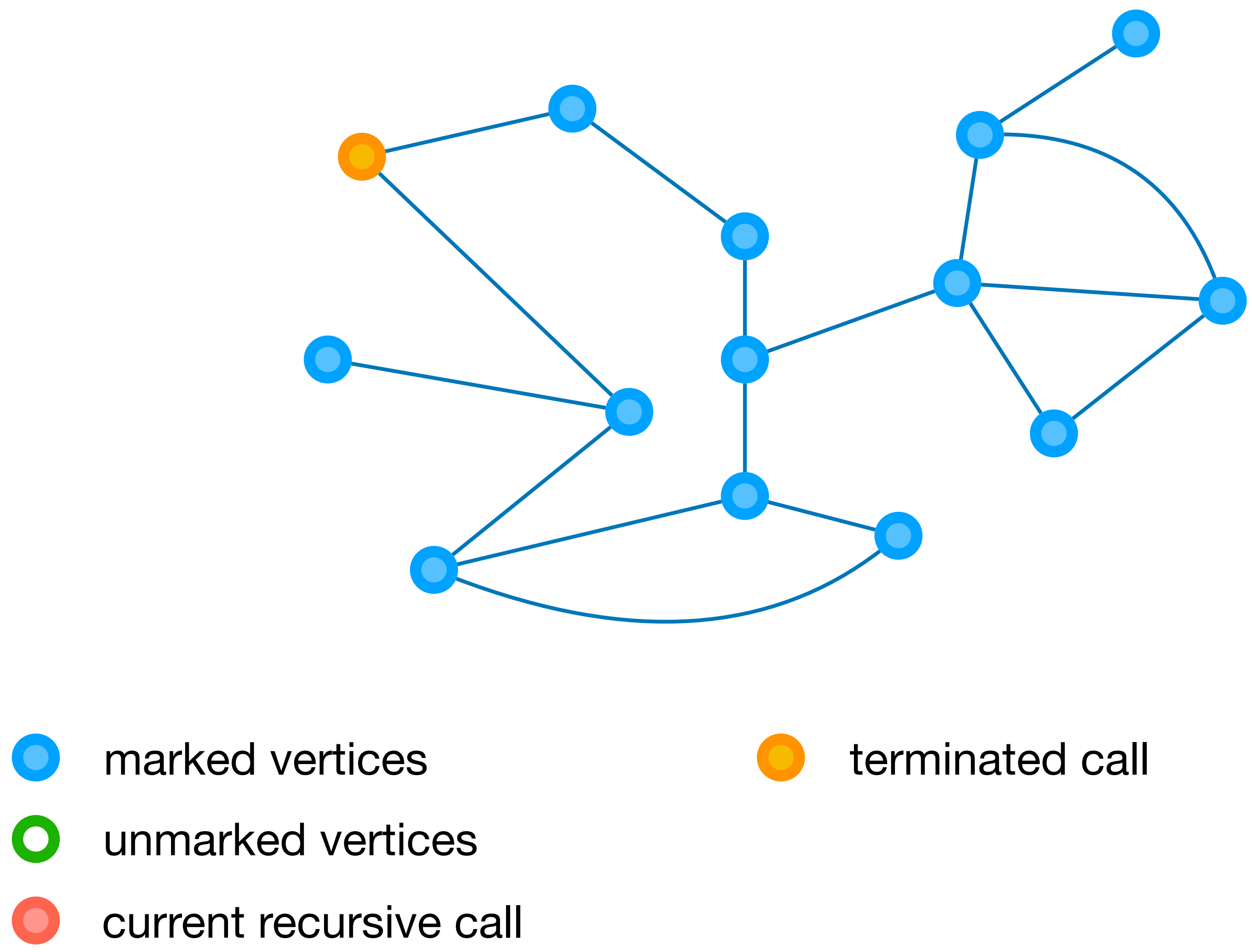
DFS: Example





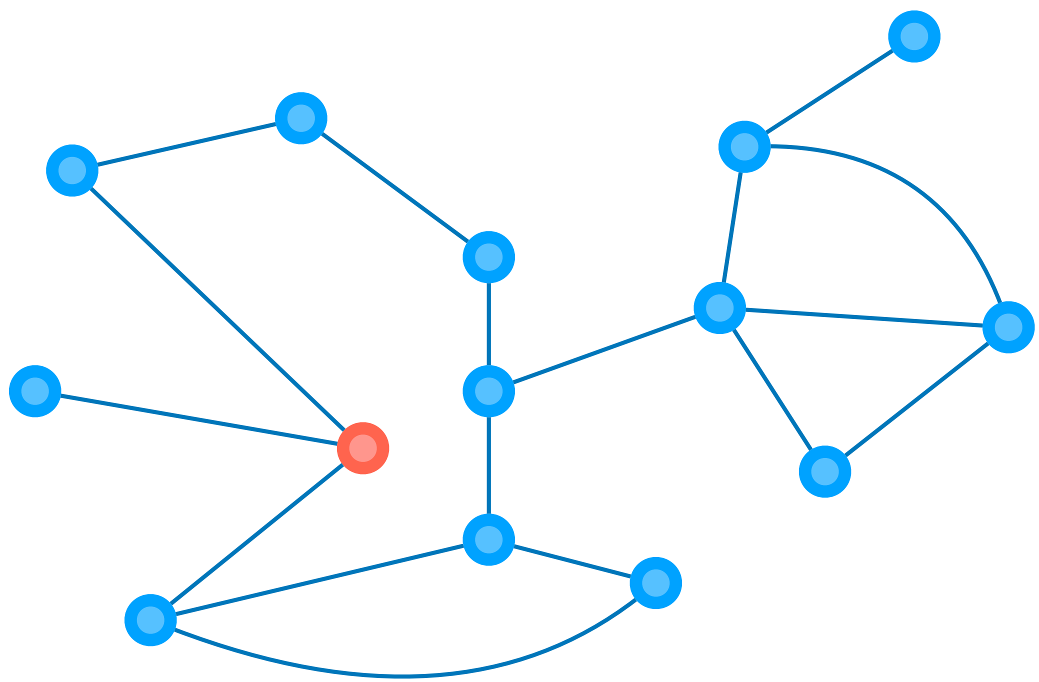
marked vertices
unmarked vertices
current recursive call
terminated call

DFS: Example



marked vertices
unmarked vertices
current recursive call
terminated call

DFS: Example



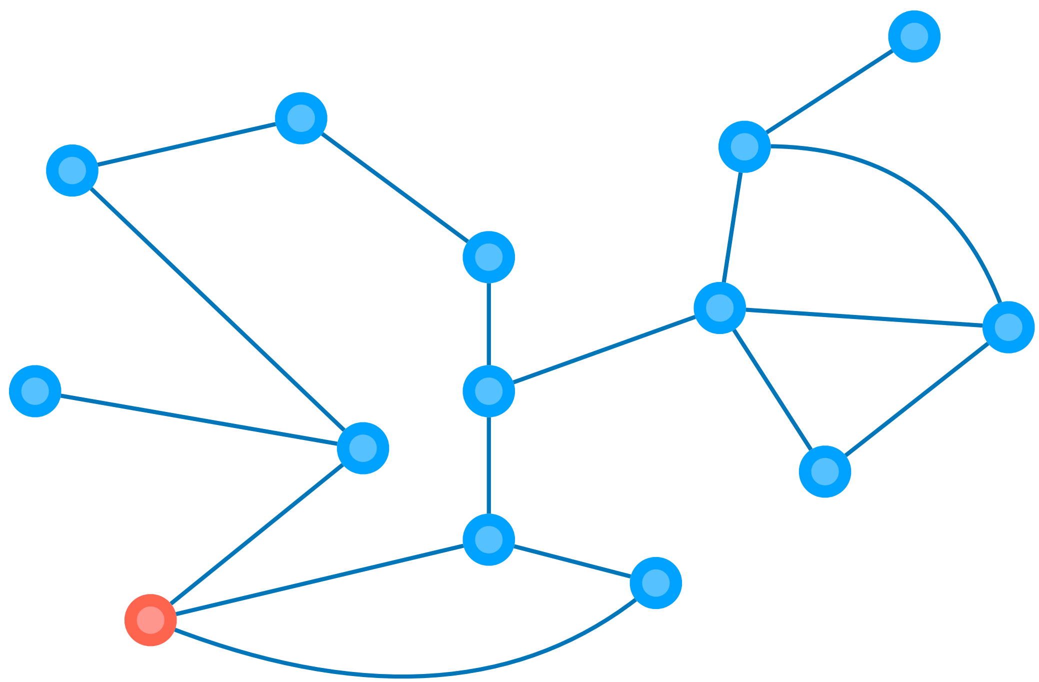
marked vertices

unmarked vertices

current recursive call

terminated call

DFS: Example



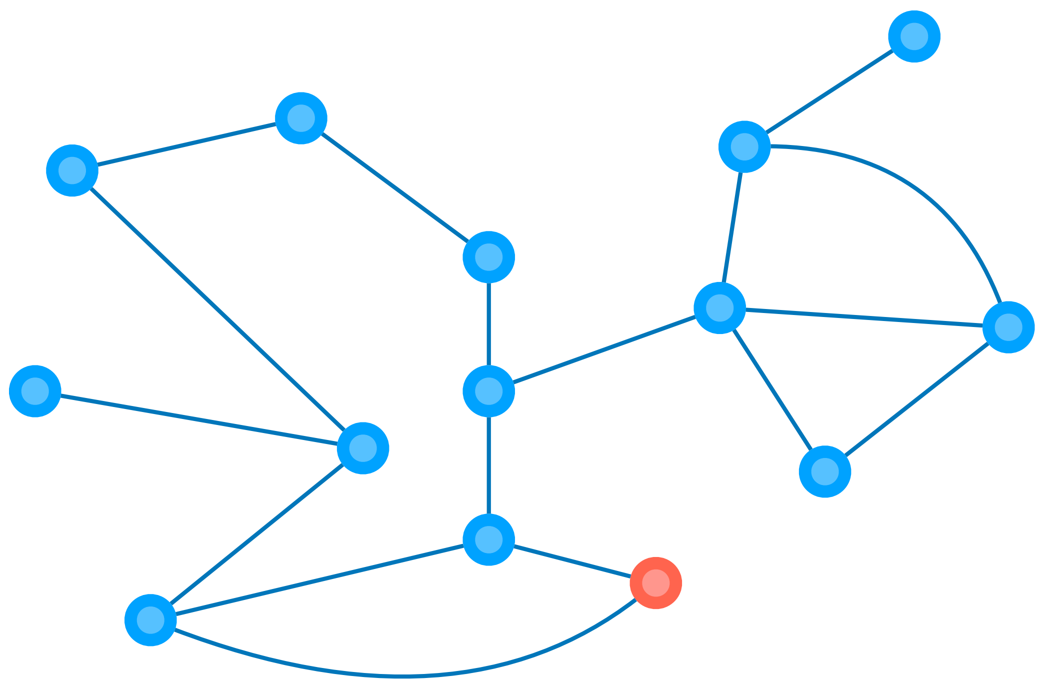
marked vertices

unmarked vertices

current recursive call

terminated call

DFS: Example



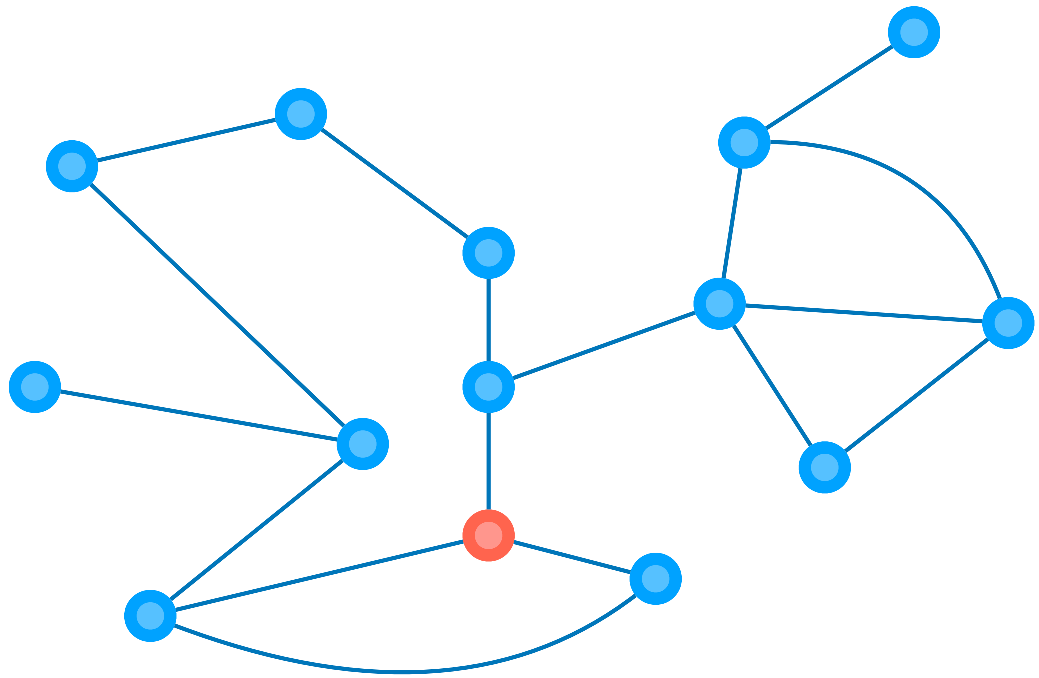
marked vertices

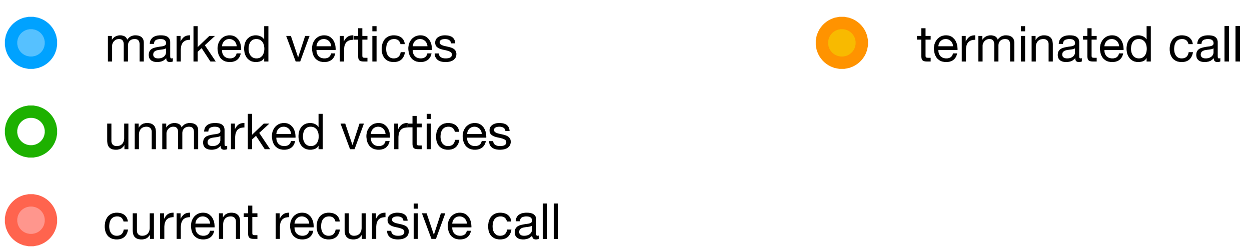
unmarked vertices

current recursive call

terminated call

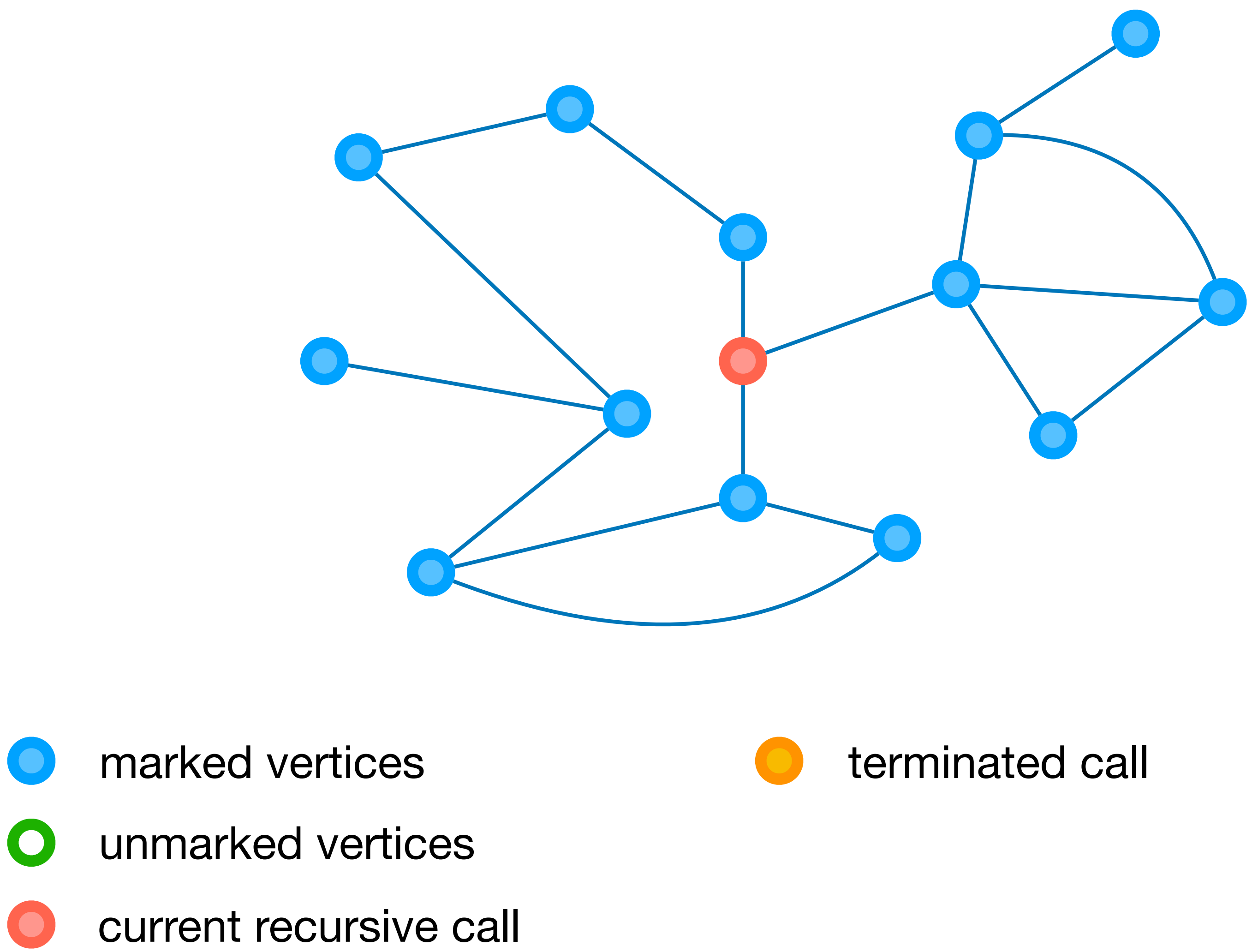
DFS: Example





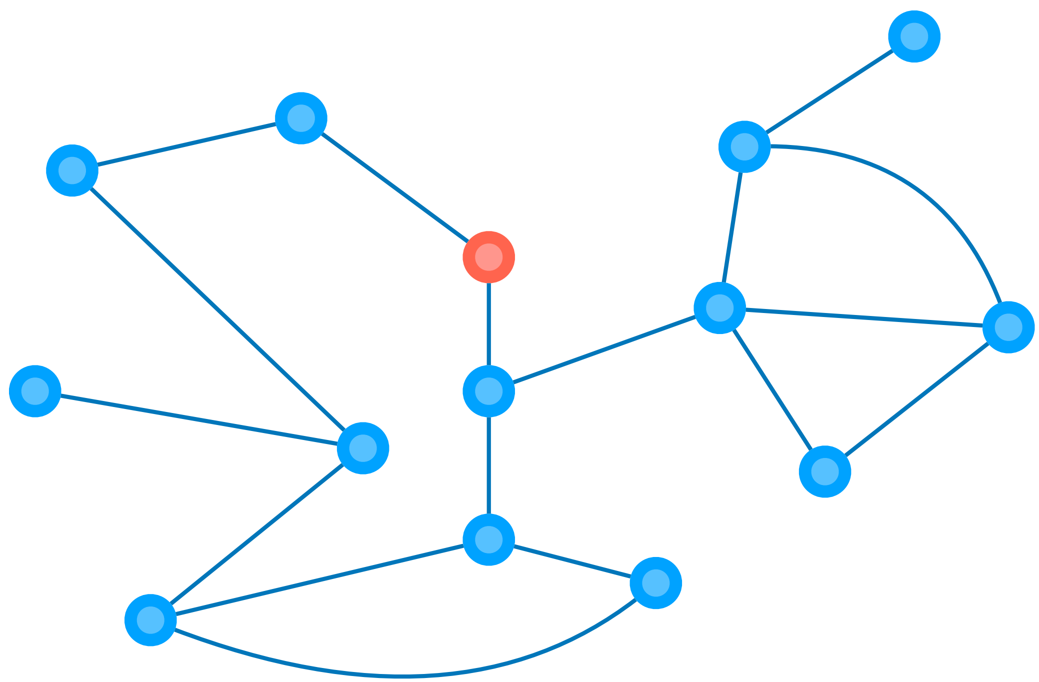
marked vertices
unmarked vertices
current recursive call
terminated call

DFS: Example



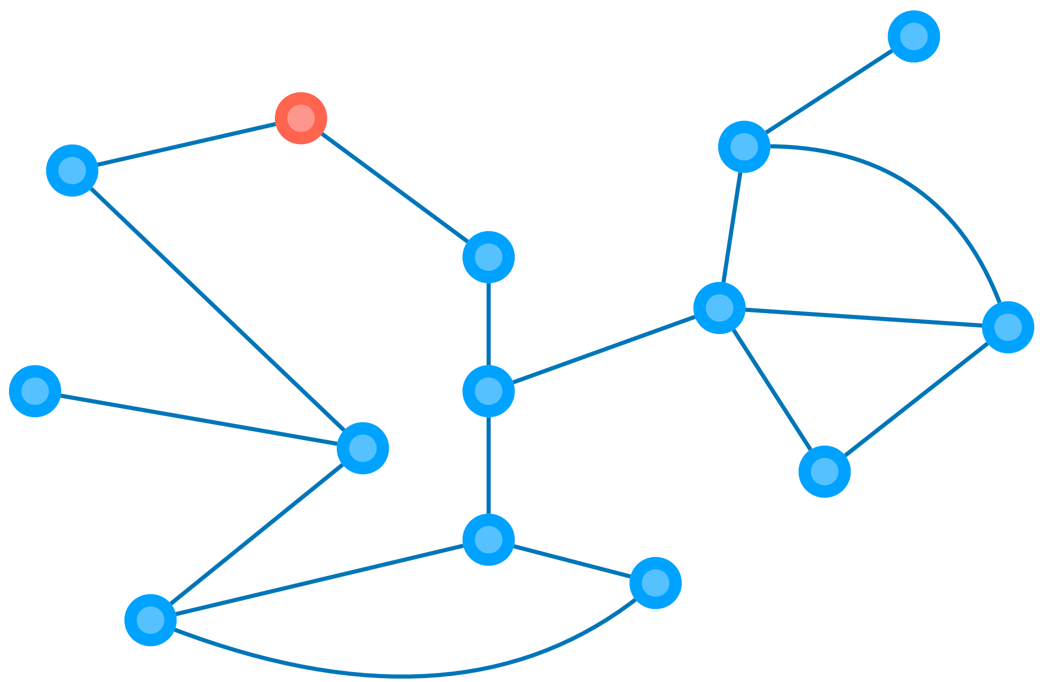
marked vertices
unmarked vertices
current recursive call
terminated call

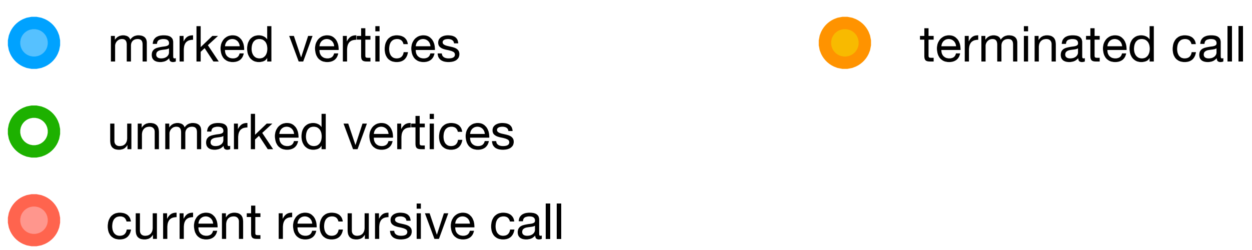
DFS: Example



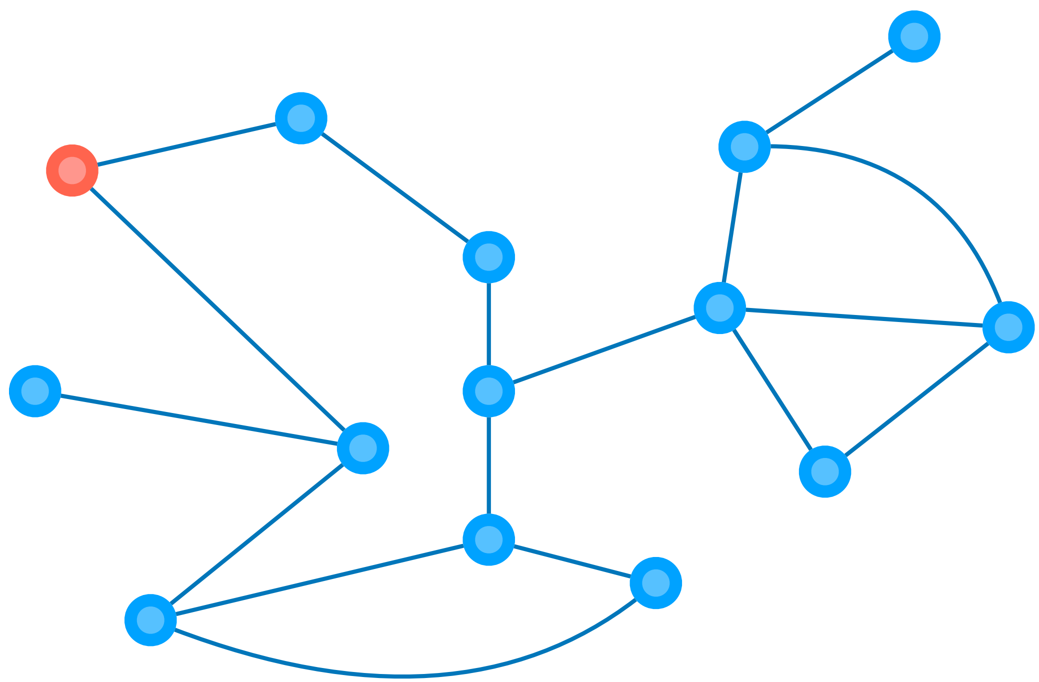
- marked vertices
- unmarked vertices
- current recursive call
- terminated call

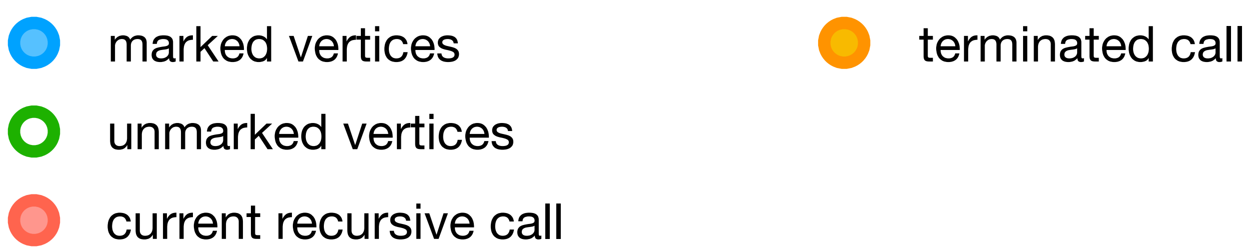
DFS: Example




marked vertices
unmarked vertices
current recursive call
terminated call

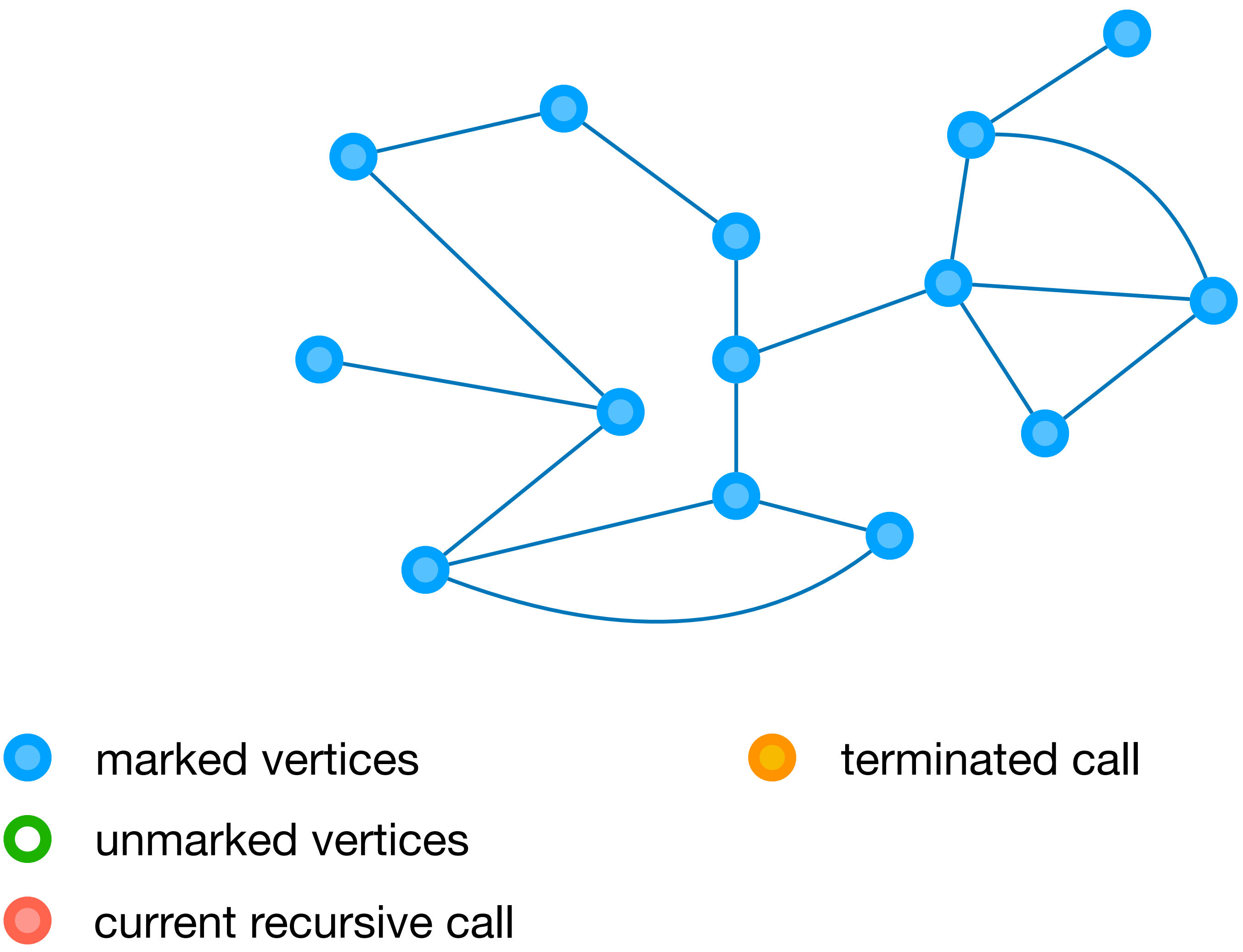
DFS: Example





marked vertices
unmarked vertices
current recursive call
terminated call

DFS: Example



marked vertices
unmarked vertices
current recursive call
terminated call

The Actual DFS

- Create an array mark[1:n] = ‘false’ initially
- Run the following algorithm on s recursively, i.e., **DFS**(s)
- **DFS**(u):
  - If mark[u] = ‘true’, terminate; otherwise mark[u] = ‘true’
  - For $v \in N(u)$ recursively run **DFS**(v)
- Output all marked vertices as the connected component of s

Proof of Correctness

- Create an array mark[1:n] = 'false' initially
- Run the following algorithm on s recursively, i.e., **DFS**(s)
- **DFS**(u):
  - If mark[u] = 'true', terminate; otherwise mark[u] = 'true'
  - For $v \in N(u)$ recursively run **DFS**(v)
- Output all marked vertices as the connected component of s

- Set of marked vertices is exactly the CC of s

Proof of Correctness

Runtime Analysis

- Create an array mark[1:n] = 'false' initially
- Run the following algorithm on s recursively, i.e., **DFS**(s)
- **DFS**(u):
  - If mark[u] = 'true', terminate; otherwise mark[u] = 'true'
  - For $v \in N(u)$ recursively run **DFS**(v)
- Output all marked vertices as the connected component of s

- We have n subproblems
- The subproblem for vertex v takes O(1+deg(v)) time
- So total runtime is $O(\sum_{v \in V} (1 + \deg(v)))$
- We have $\sum_{v \in V} \deg(v) = 2m$
- So runtime is O(n+m)

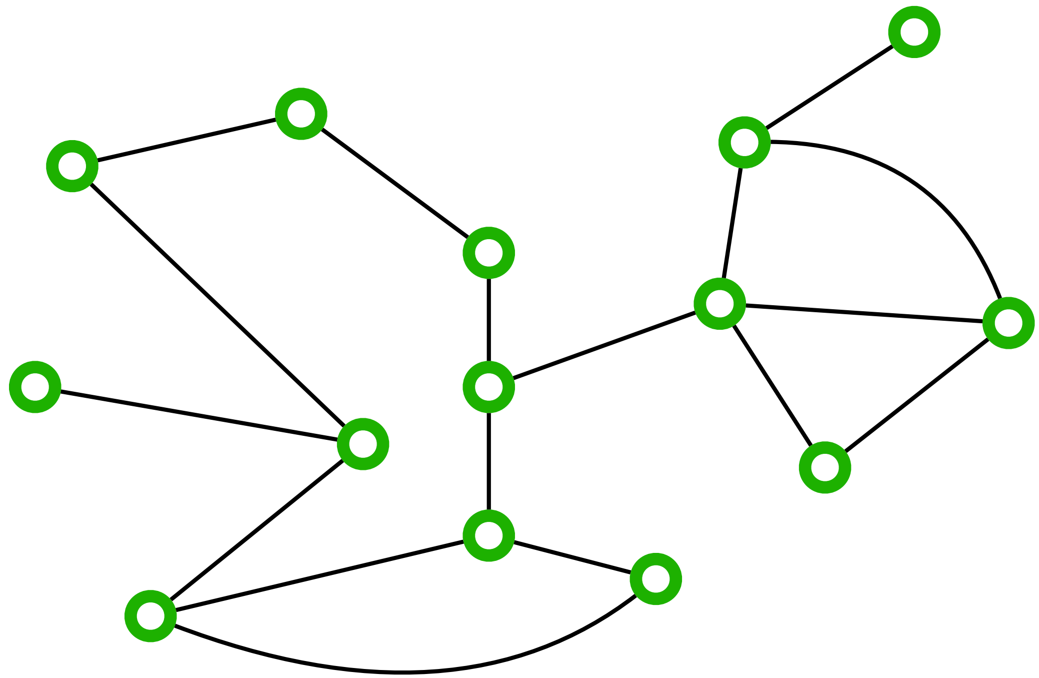
Further Extensions

- Can we find a path between all pairs of vertices in the connected components?

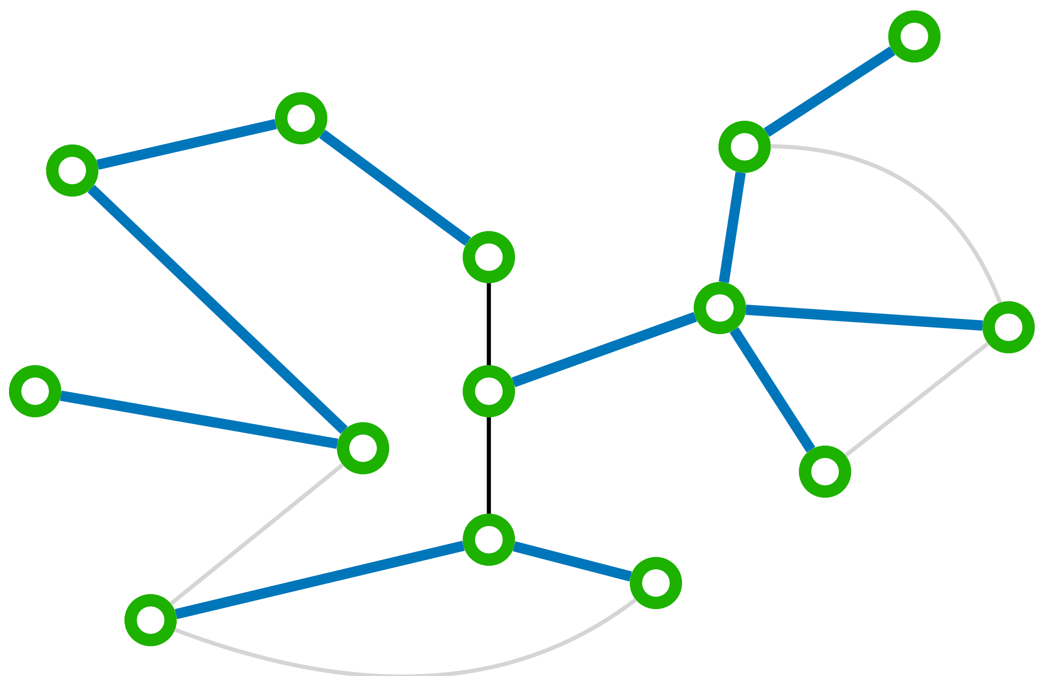
- A spanning tree: a subgraph of G which connects all vertices in the connected component of s and is additionally a tree

  - tree: a graph with no cycles

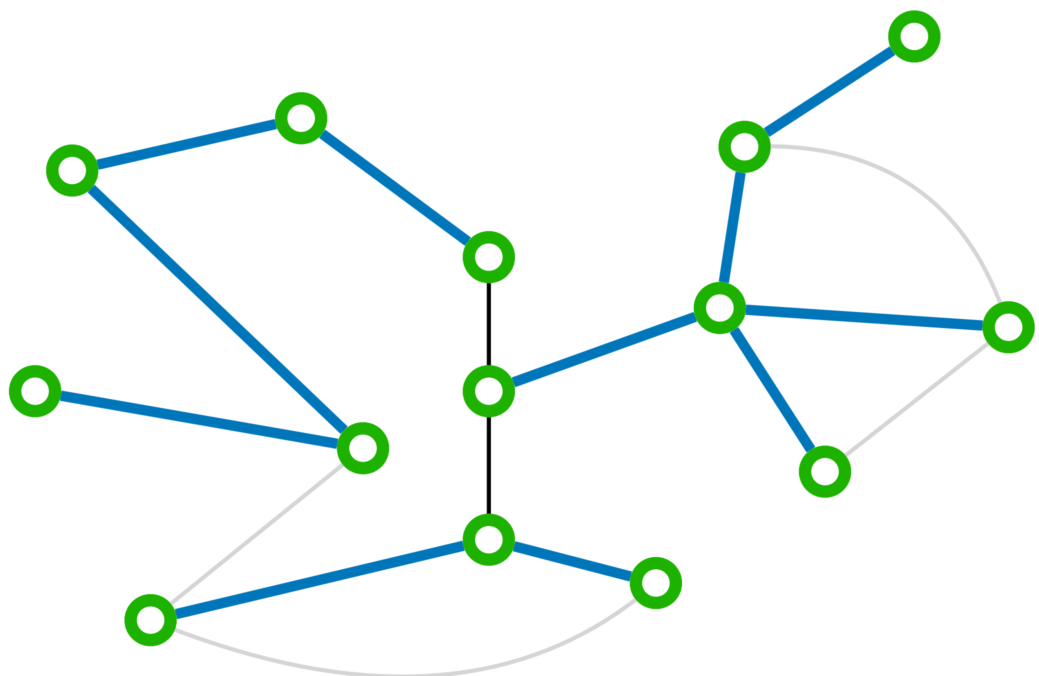
Spanning Tree: Example



Spanning Tree: Example



Spanning Tree: Example

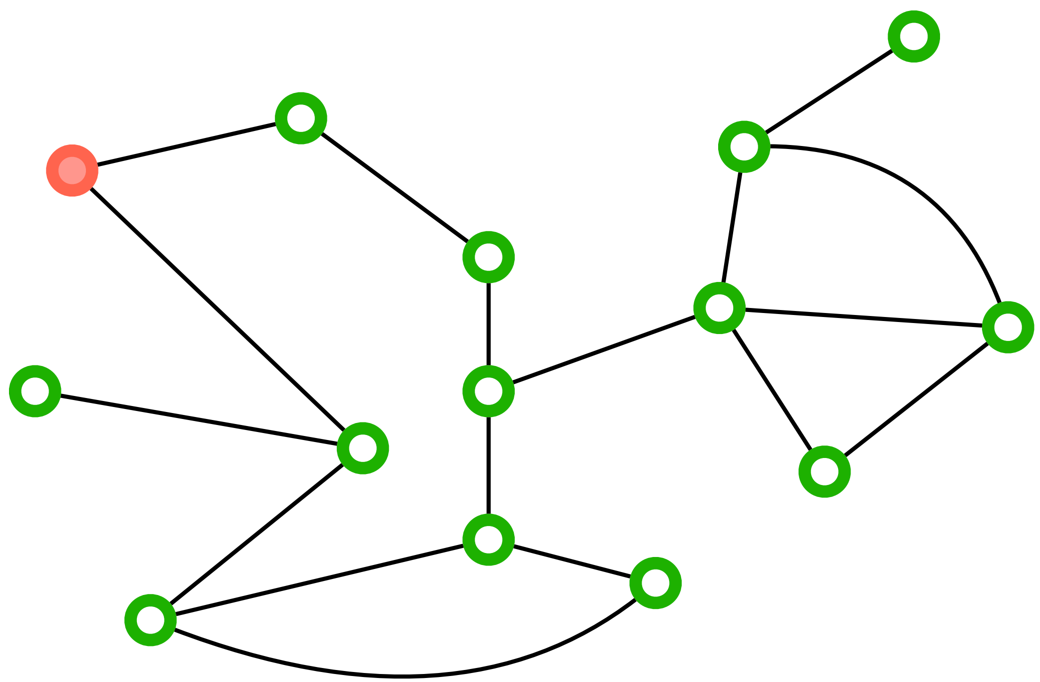


In every tree, there is a unique path between every pairs of vertices

The DFS-Tree Algorithm

- Create an array mark[1:n] = ‘false’ initially
- Run the following algorithm on s recursively, i.e., **DFS-Tree**(s)
- **DFS-Tree**(u):
    - Set mark[u] = ‘true’
    - For $v \in N(u)$:
        - If mark[v] = ‘false’, add the edge (u,v) to the tree (or add v as child-node of u in the tree) and recursively run **DFS-Tree**(v)

DFS-Tree: Example

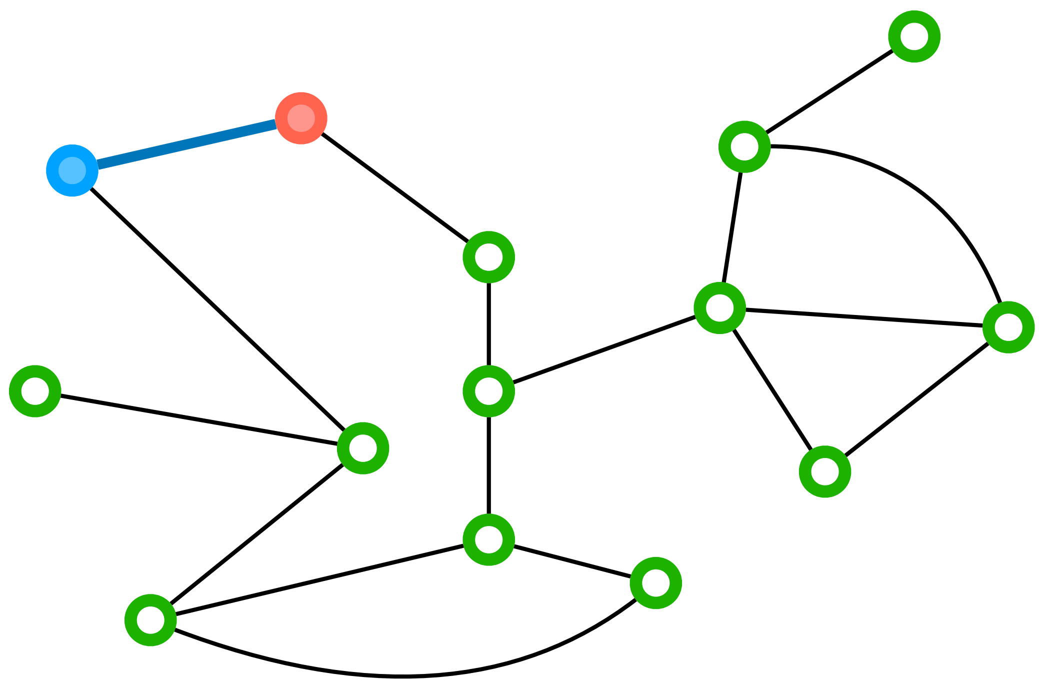


marked vertices

unmarked vertices

current recursive call

DFS-Tree: Example

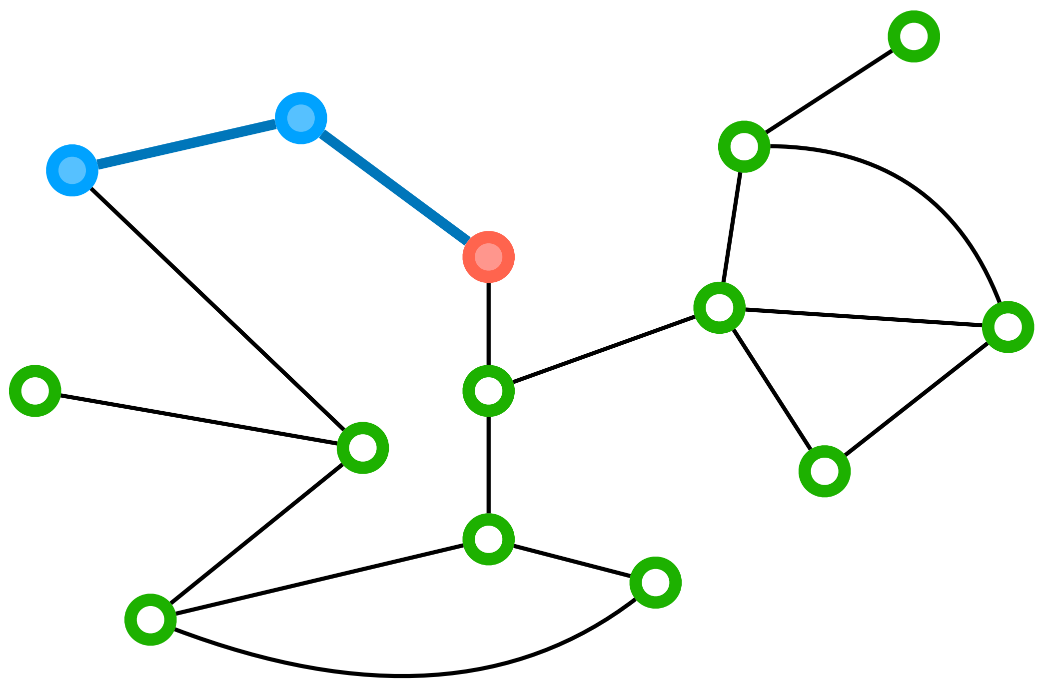


marked vertices

unmarked vertices

current recursive call

DFS-Tree: Example

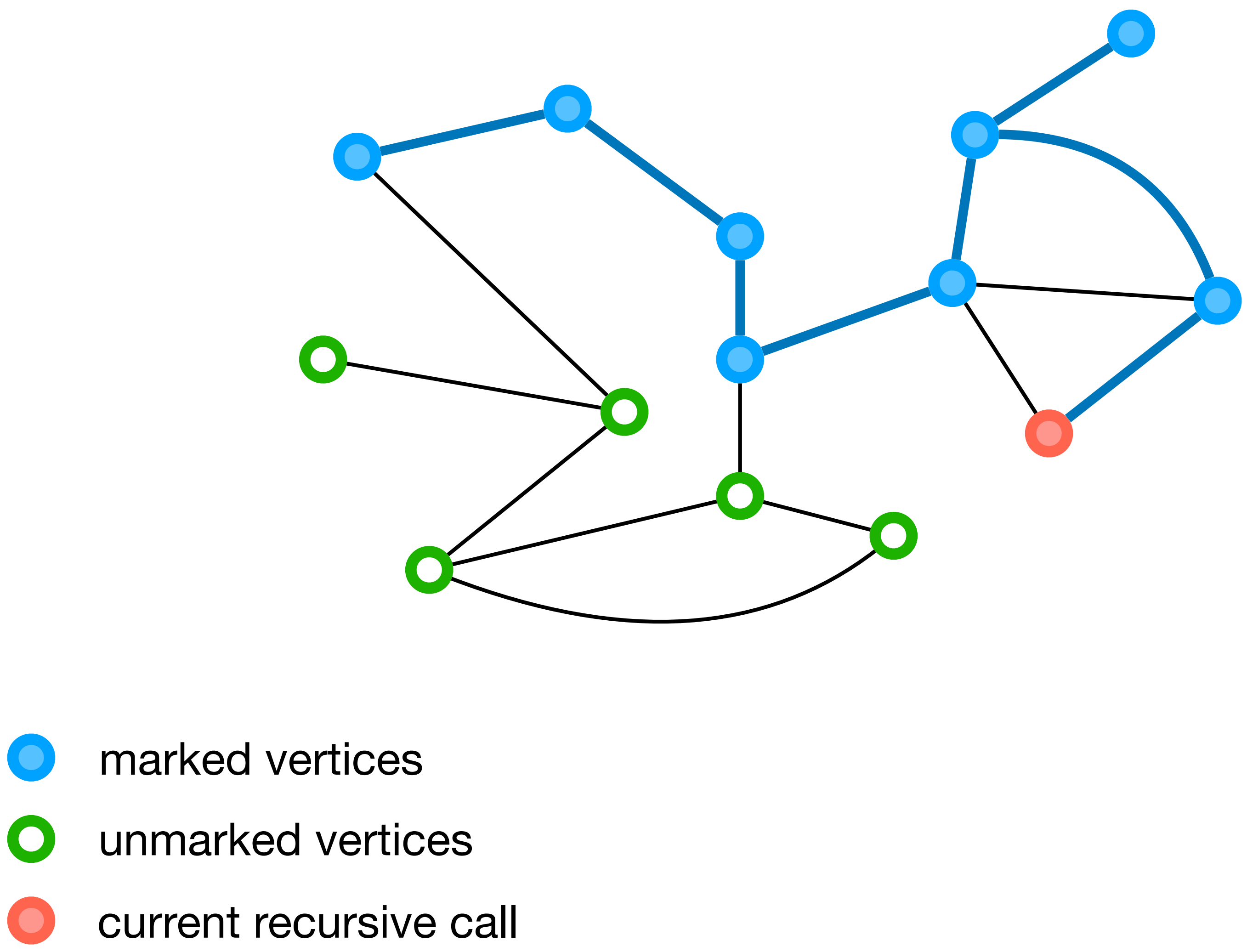



marked vertices

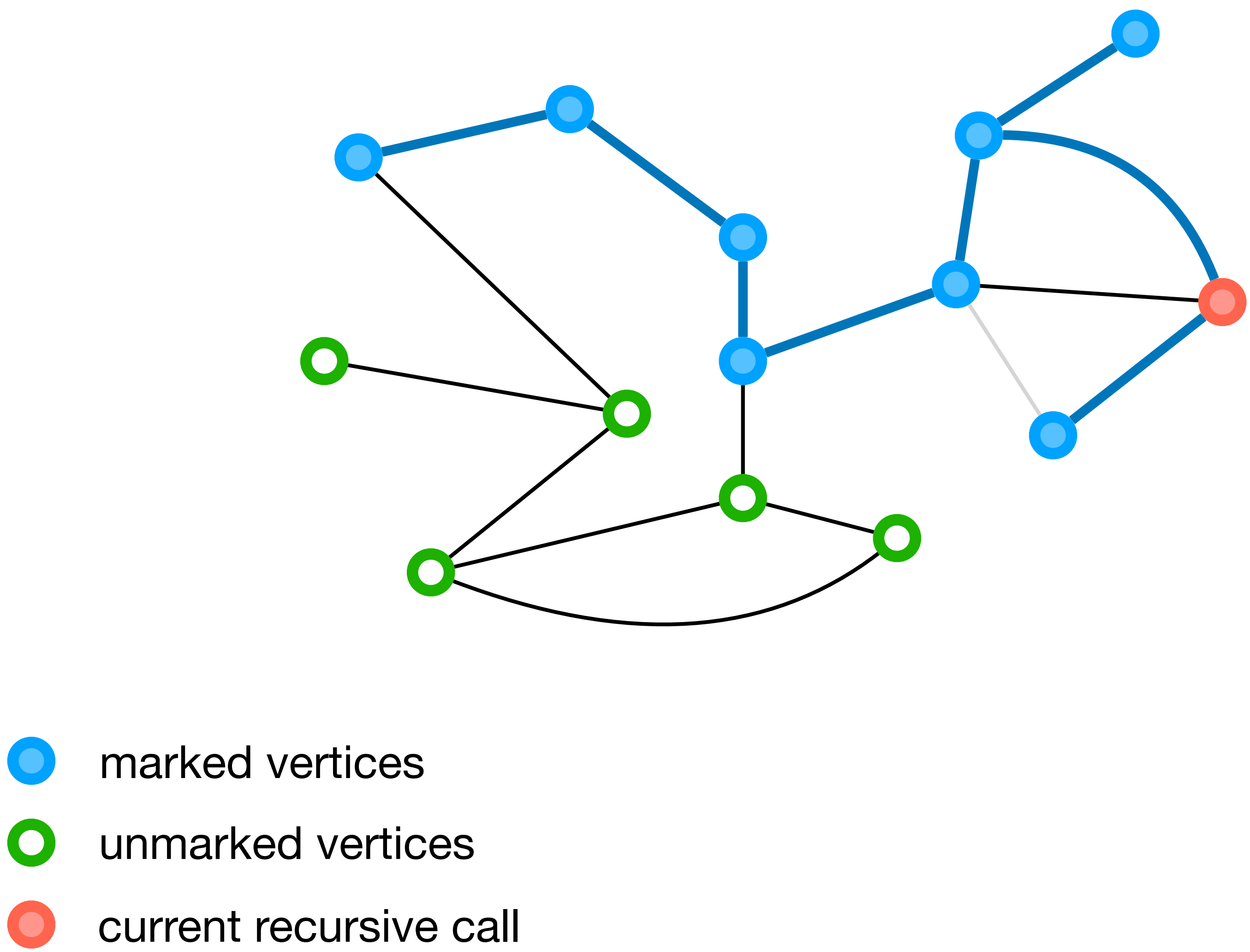
unmarked vertices

current recursive call

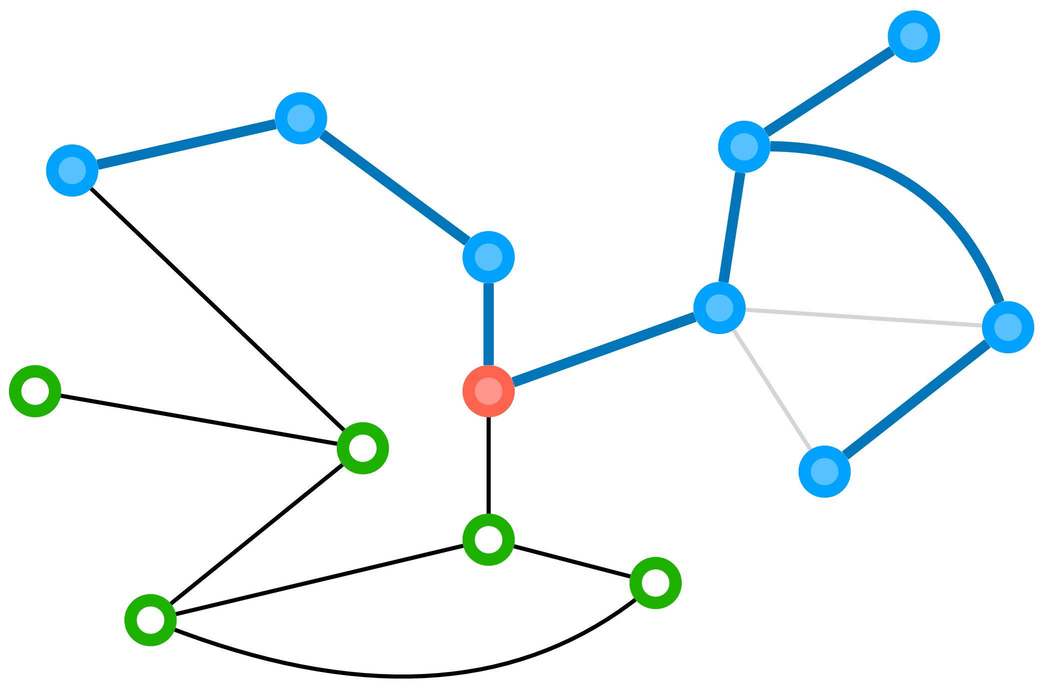
DFS-Tree: Example



DFS-Tree: Example



DFS-Tree: Example

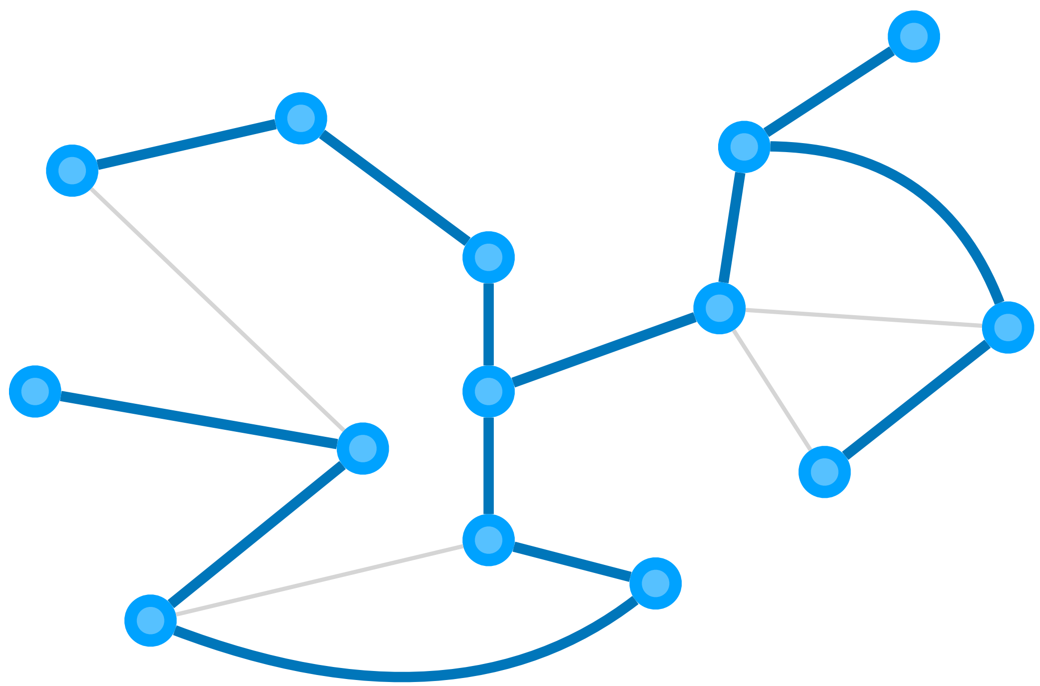


marked vertices

unmarked vertices

current recursive call

DFS-Tree: Example

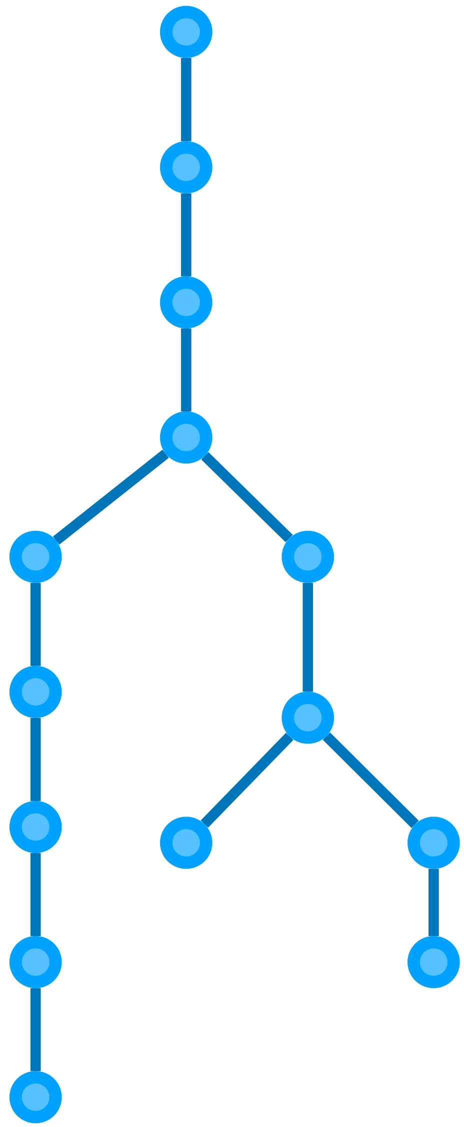


marked vertices

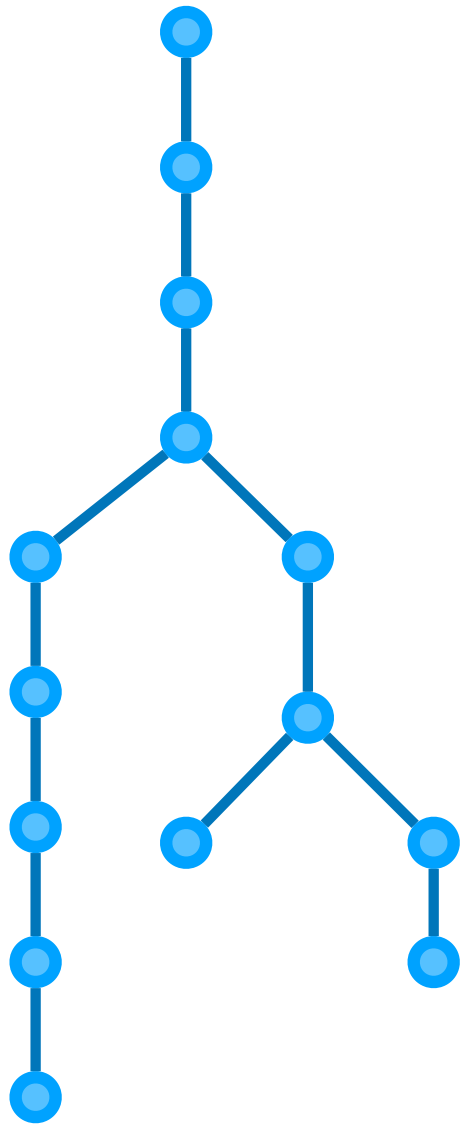
unmarked vertices

current recursive call

DFS-Tree: Example



DFS-Tree: Example



DFS tends to create very deep trees

Further Extensions

- Can we find a path between all pairs of vertices in the connected components?
- A spanning tree: a subgraph of G which connects all vertices in the connected component of s and is additionally a tree
  - tree: a graph with no cycle
- DFS can also be used on directed graphs
  - Set of all vertices reachable from s
  - A path from s to all these vertices