If you have any question, please ask via

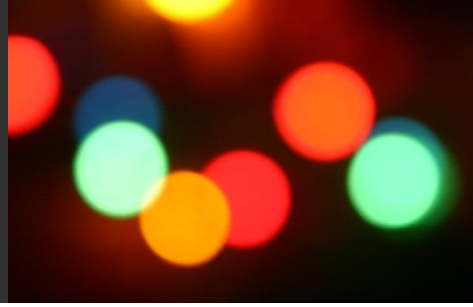
Email: garywong612@gmail.com

MSN: gary_wong612@hotmail.com

Graph Representation, DFS and BFS

Gary Wong

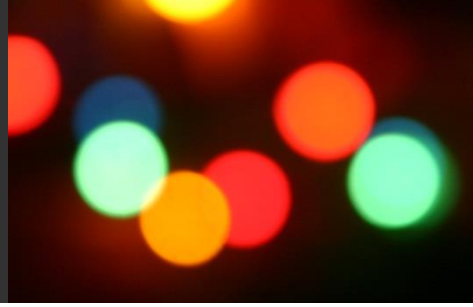
WARNING!



Today's topics are:

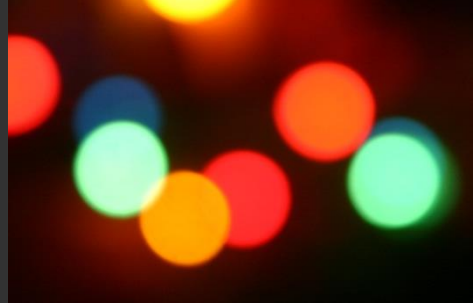
- Tough
  - Probably the hardest among all intermediate topics
- Essential
  - Foundation of many advanced algorithms/concepts, such as the topics in the afternoon session

But don’t panic…



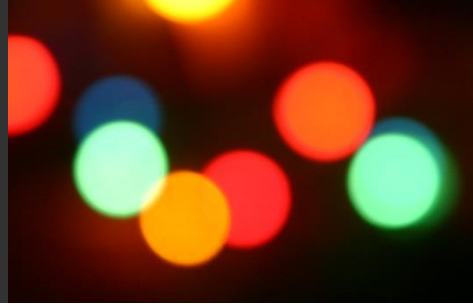
- I will guide you throughout the training
- Don’t be shy to interrupt and ask ^^

Pre-requisites



- Before we start, I would like to ensure that you know the followings:
  - Queue
  - Linked List
  - Recursion

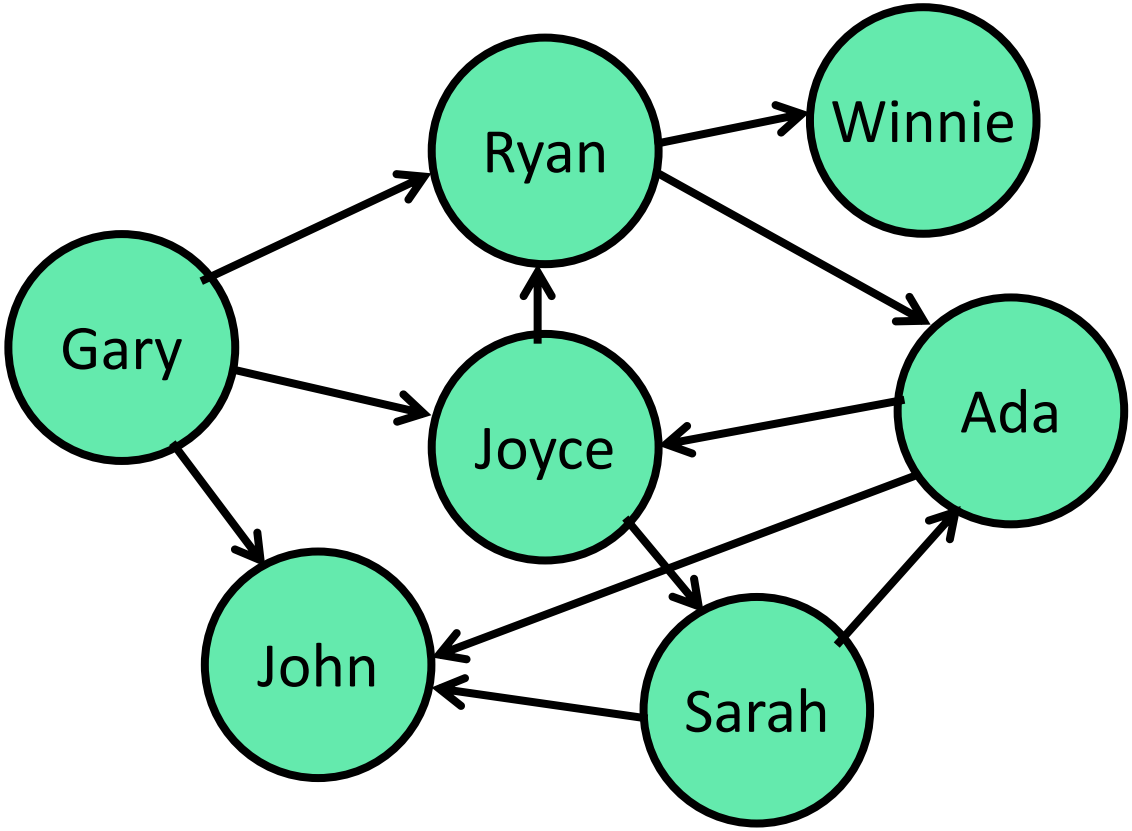
A classical problem



- Now I have N friends, but I have the phone numbers of some of them only
- I want to deliver a message to Sarah by repeated calling
- My memory is so good that I remember who can be contacted by my N friends
- Give me a calling plan for me and my friends

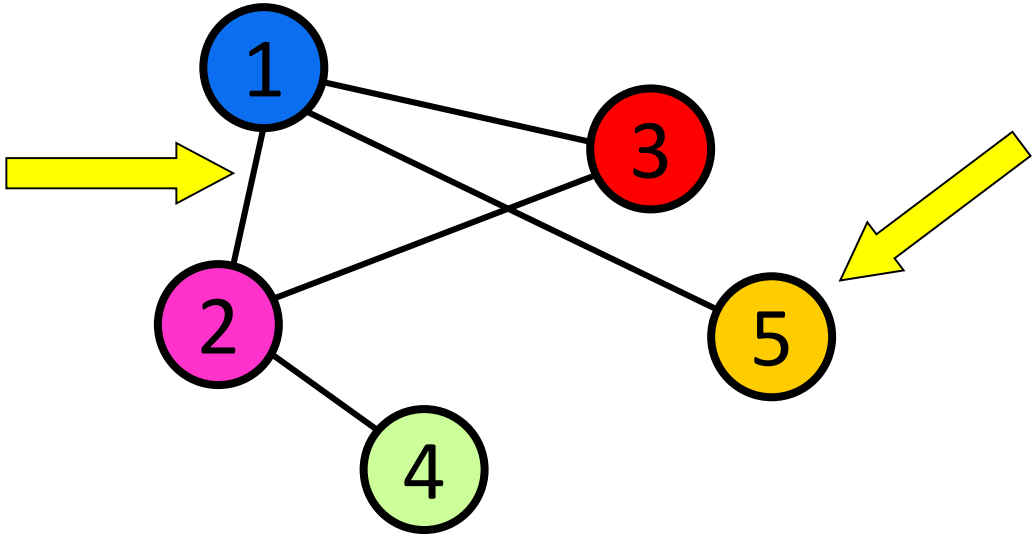
A classical problem

- This is actually a graph!



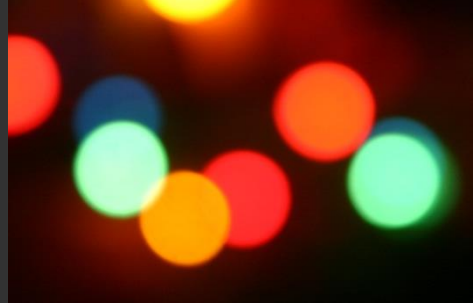
What is a graph?

- A set of vertices (or nodes) linked by edges

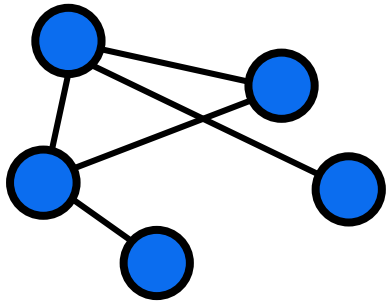


- Mathematically, we often write G = (V,E)
  - V: set of vertices, so |V| = number of vertices
  - E: set of edges, so |E| = number of edges

Why do we need graphs?



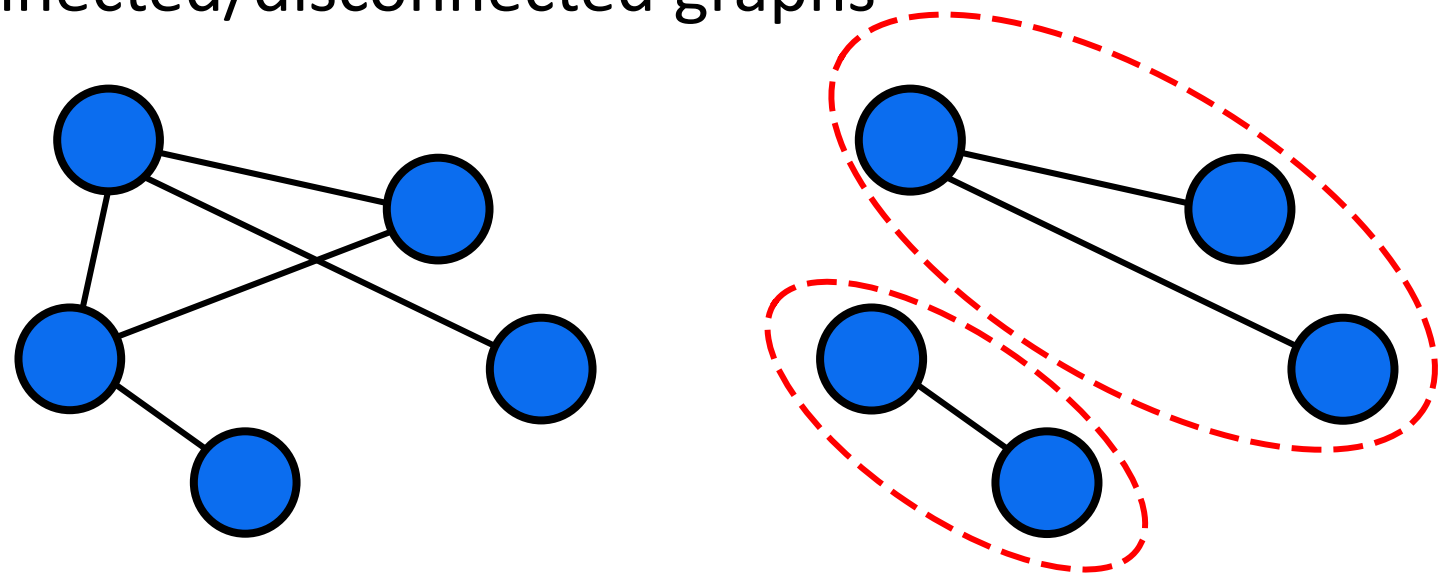
- To present the relationships between different objects/elements in a mathematical way
- Examples:
  - Friendship
  - Local Area Network (LAN)
  - Map of a country



- What could the vertices and edges represent in the above examples?

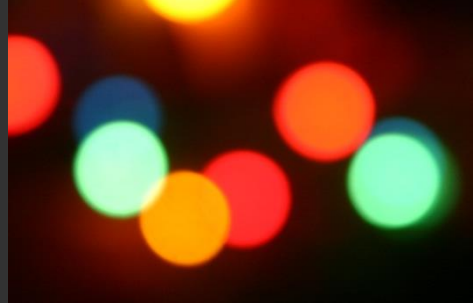
Various types of graphs

- Connected/disconnected graphs

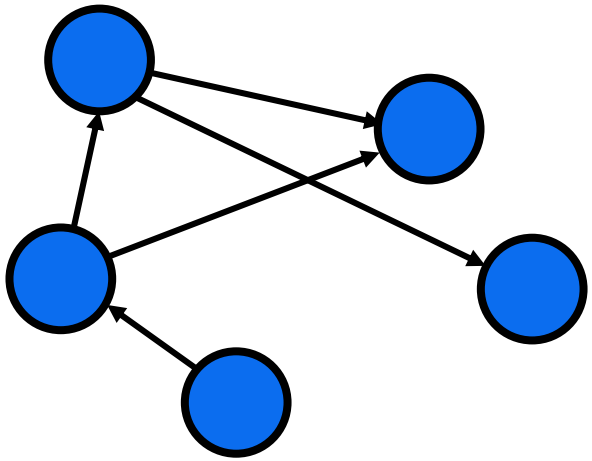


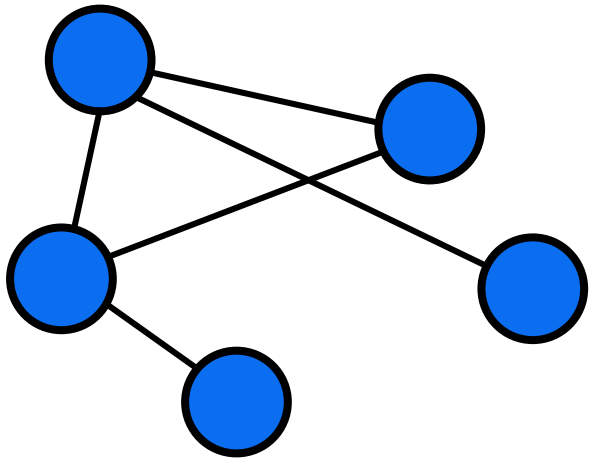
- The circled subgraphs are also known as connected components

Various types of graphs



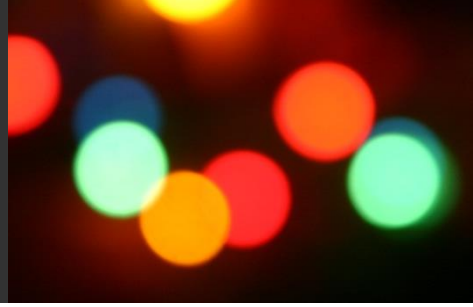
- Directed/undirected graphs



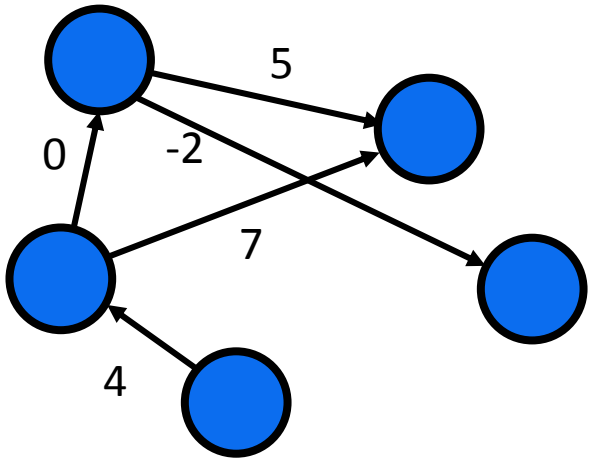


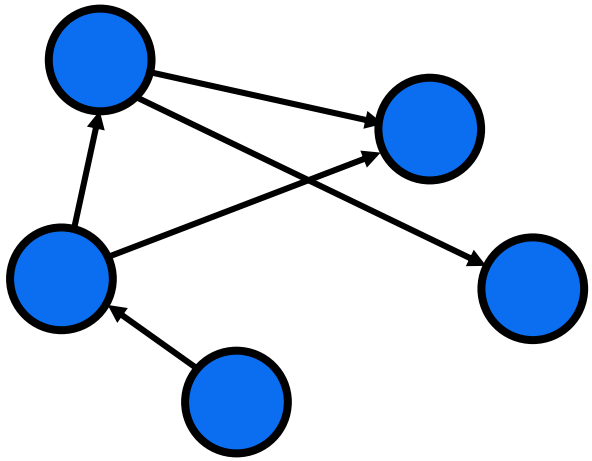
- You may treat each undirected edge as two directed edges in opposite directions

Various types of graphs



- Weighted/unweighted graphs

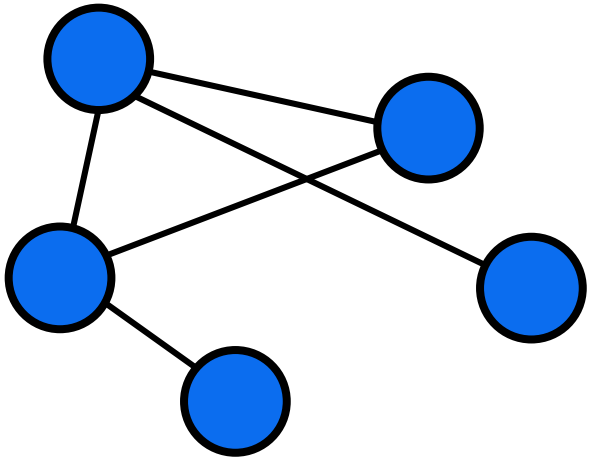


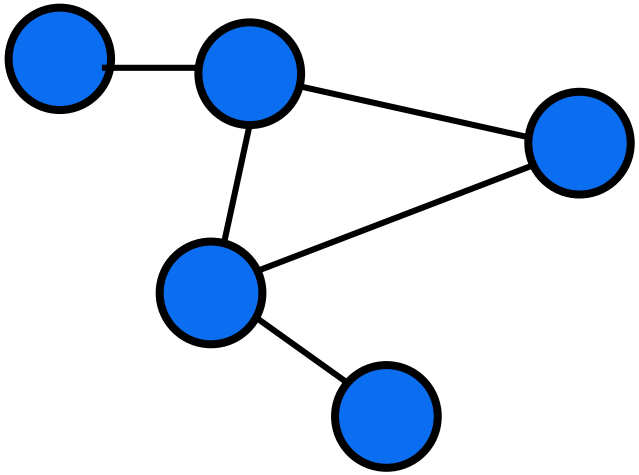


- You may treat unweighted edges to be weighted edges of equal weights

Special graphs

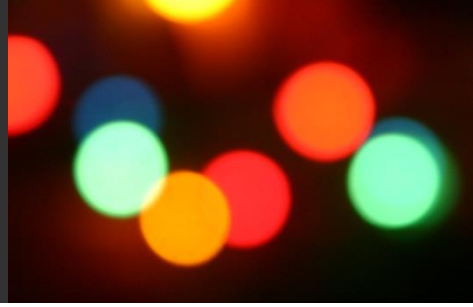
- Planar graphs
  - A graph that can be drawn on a plane without edge intersections
  - The following two graphs are equivalent and planar:





- To be discussed in details in Graph (III) ☺

Special graphs

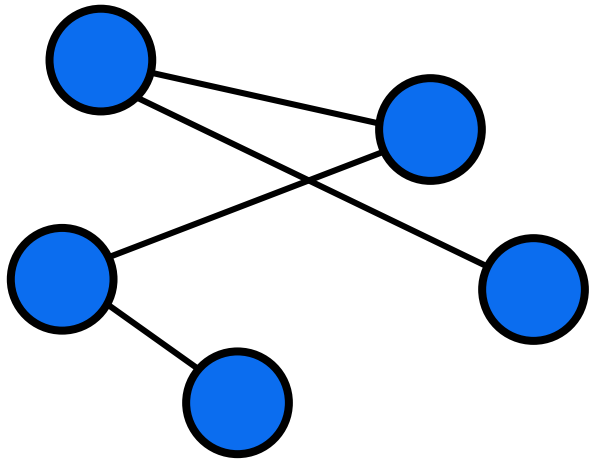


- Tree

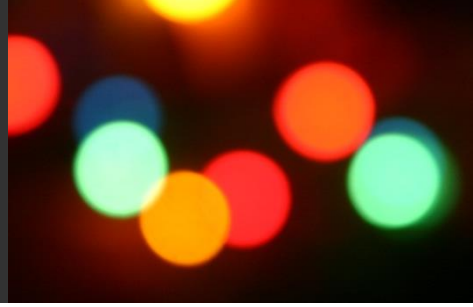
NO!

Special graphs

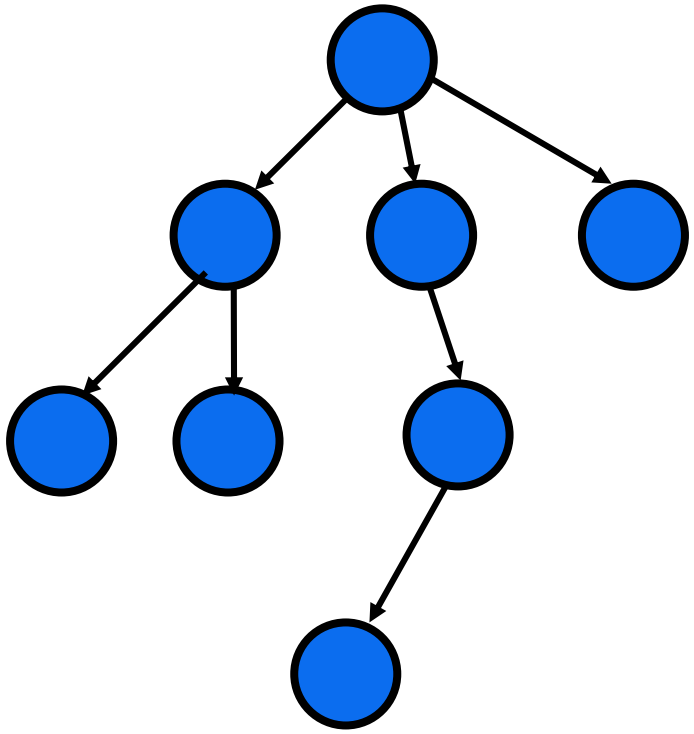
- Tree: either one of the followings is the definition
  - A connected graph with |V|-1 edges
  - A connected graph without cycles
  - A graph with exactly one path between every pair of vertices



Special graphs

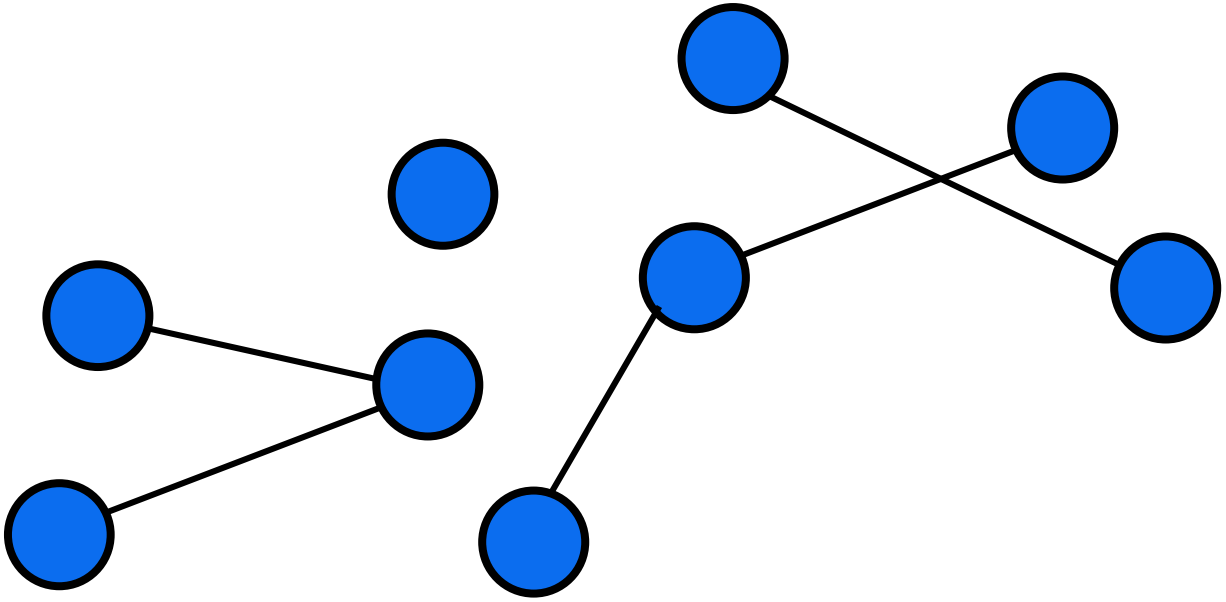


- Tree edges could be directed or undirected
- For trees with directed edges, a root usually exists



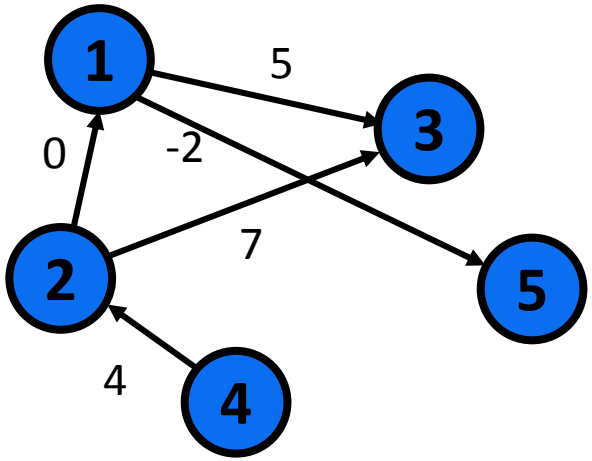
Special graphs

- Forest
  - All connected component(s) is/are tree(s)
- How many trees are there in the following forest?



How to store graphs in the program?

- Usually, the vertices are labeled beforehand

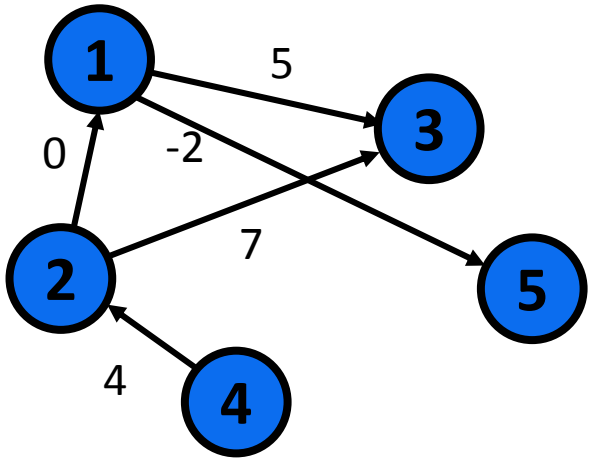


- 3 types of graph representations:
  - Adjacency matrix
  - Adjacency list
  - Edge list

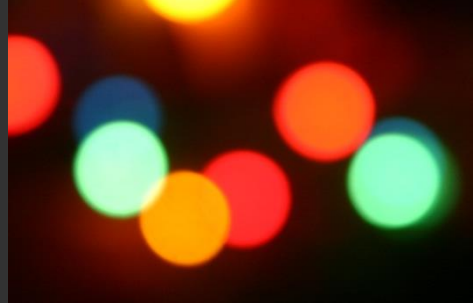
Adjacency matrix

- Use a 2D array

s\t	1	2	3	4	5
1			5		-2
2	0		7		
3					
4		4			
5					



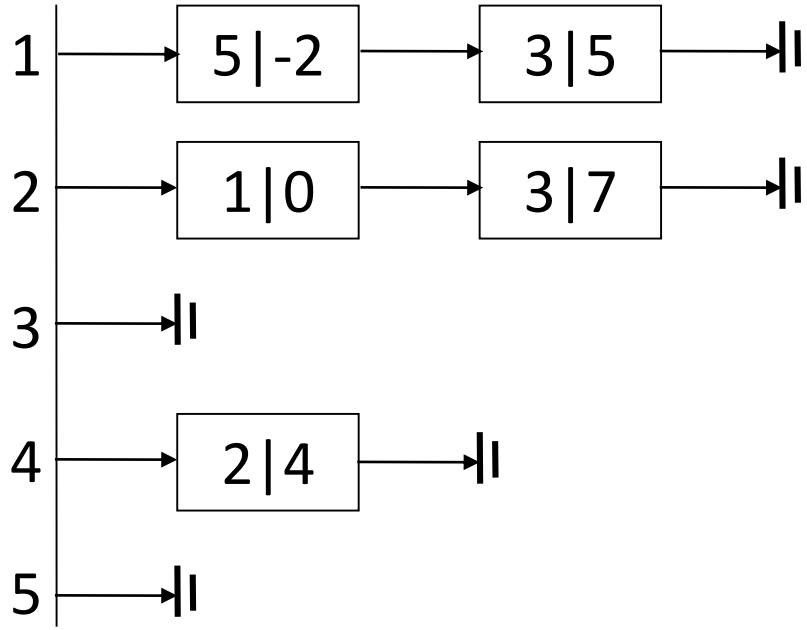
Adjacency matrix

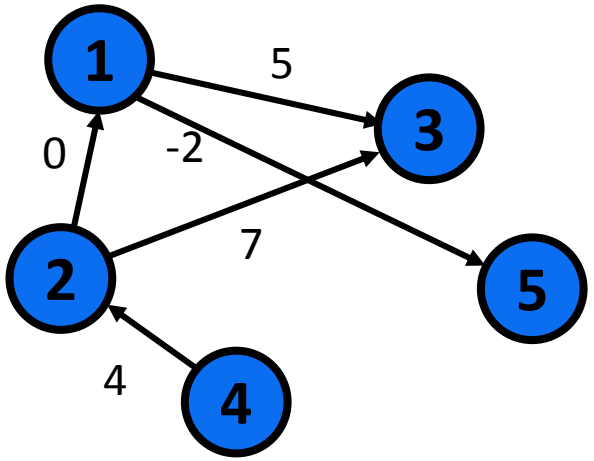


- Memory complexity?
- Time complexity for:
  - Checking the weight of an edge between 2 given nodes?
  - Querying all adjacent nodes of a given node?

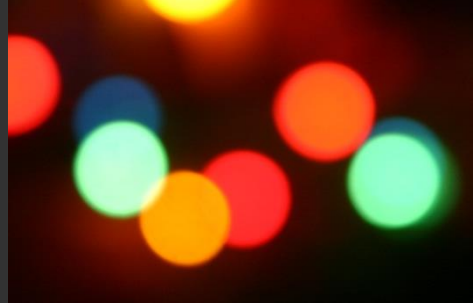
Adjacency list

- N vertices, N linked lists
- Each list stores its adjacent vertices





Adjacency list

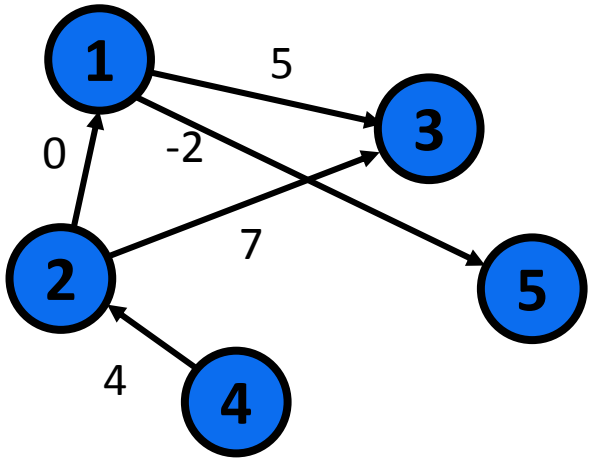


- Memory complexity?
- Time complexity for:
  - Checking the weight of an edge between 2 given nodes?
  - Querying all adjacent nodes of a given node?

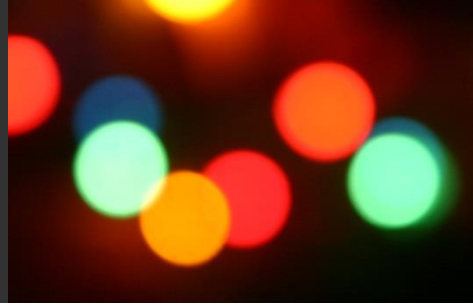
Edge list

- A list of edges

id	x	y	w
0	1	5	-2
1	2	1	0
2	1	3	5
3	2	3	7
4	4	2	4

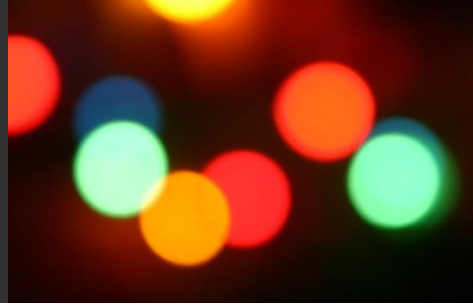


Edge list



- Memory complexity?
- Time complexity for:
  - Checking the weight of an edge between 2 given nodes?
  - Querying all adjacent nodes of a given node?

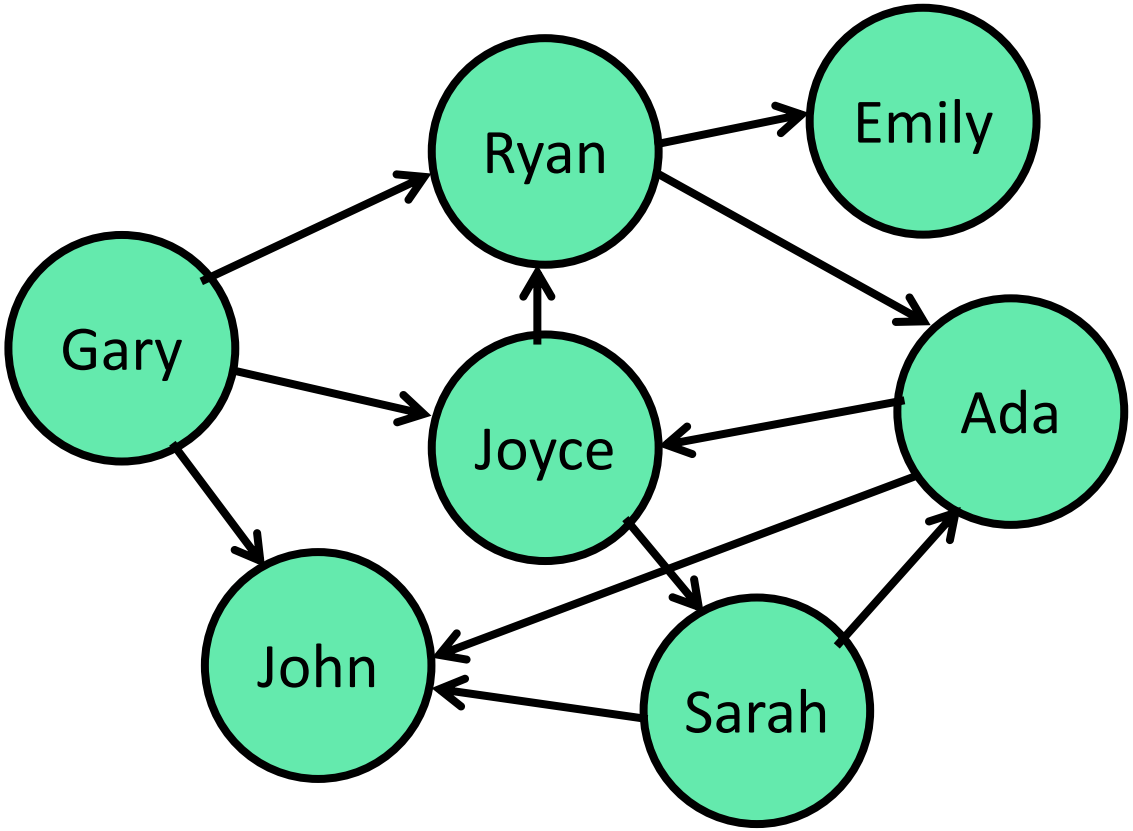
Which one should be used?



- It depends on:
  - Constraints
  - Time Limit
  - Memory Limit
  - What algorithm is used

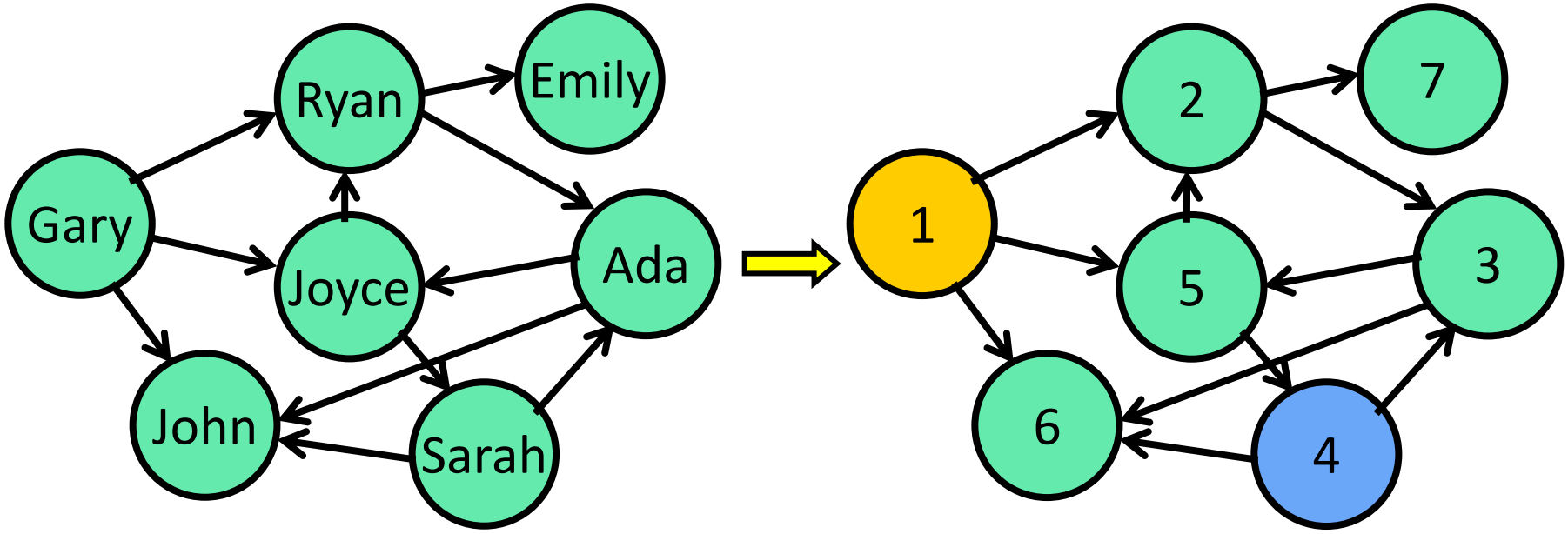
Back to the classical problem…

- How can we solve this?
- Equivalent to finding a path from “Gary” to “Sarah”

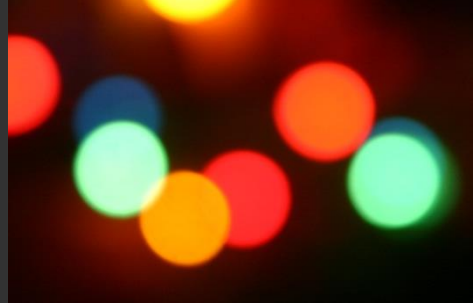


Back to the classical problem…

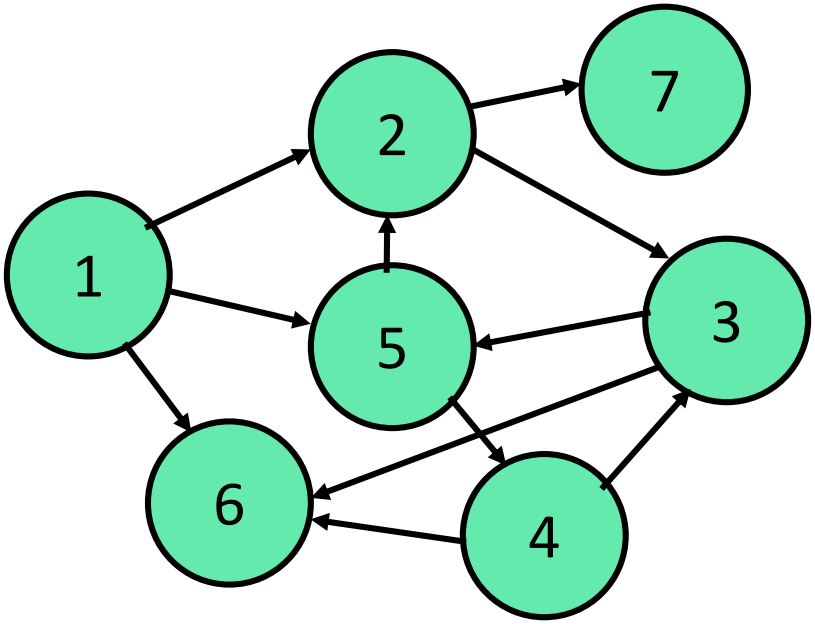
- For convenience, we label the nodes by numbers instead of names



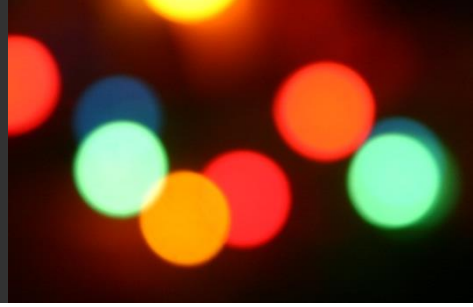
Let’s use recursion!



- Go as far as you can
- If it is a blind end, go back and walk through another edge



Let's use recursion!



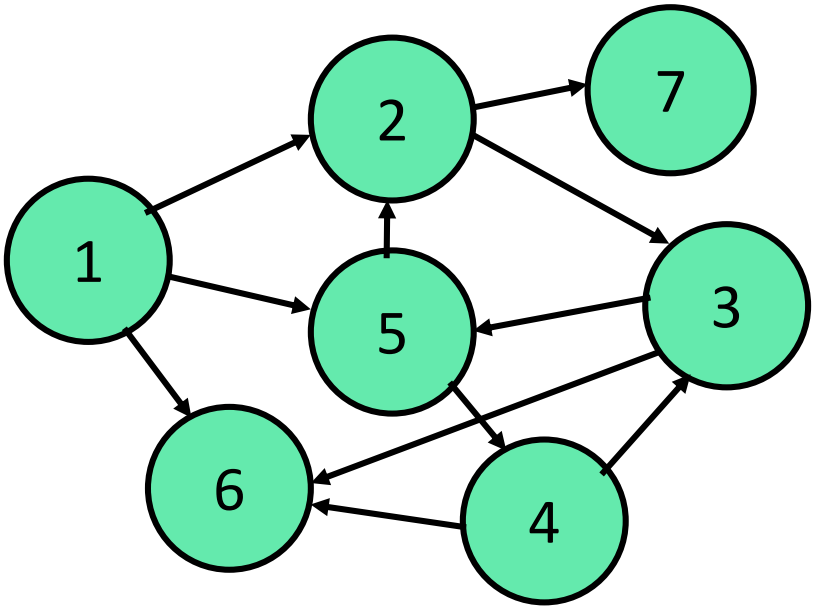
- This is also known as Depth-First-Search (DFS)

```
DFS (node u) {
    mark u as visited
    for each adjacent node v from u
        if (v is unvisited) DFS (v)
}
```

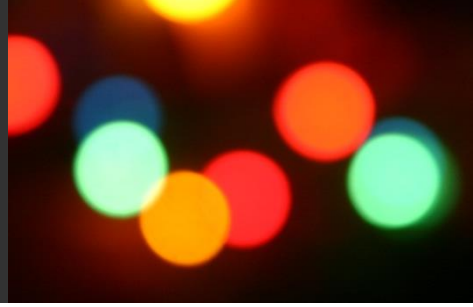
- Initialize all nodes as unvisited

Depth First Search (DFS)

- Let's review the graph, and note the color changes
  - Green: Unvisited
  - Grey: Visited
  - Black: Dead (No more unvisited adjacent nodes)

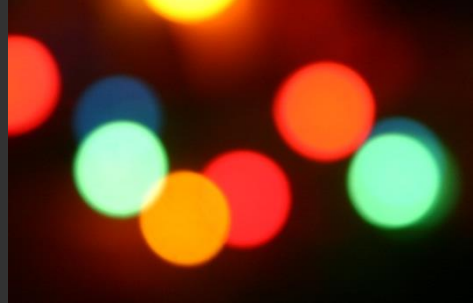


Depth First Search (DFS)



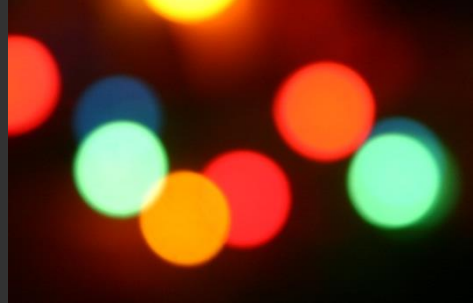
- Advantages
  - Useful for checking whether 2 nodes are connected
- Drawbacks
  - Finding a shortest path using DFS is difficult
  - Stack overflow

A similar classical problem



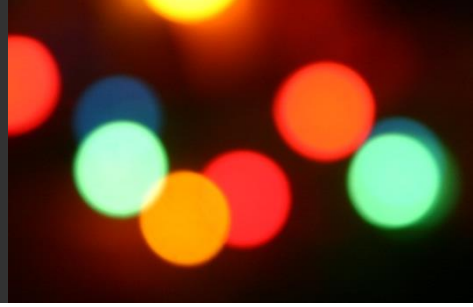
- Same situation, but I want the calling plan with the least number of calls
- Equivalent to finding the shortest path from “Gary” to “Sarah”

Breadth First Search (BFS)



- Go to all nearest nodes first
- The data structure “queue” is used to store the visited nodes
- Expand from visited (but not dead) nodes

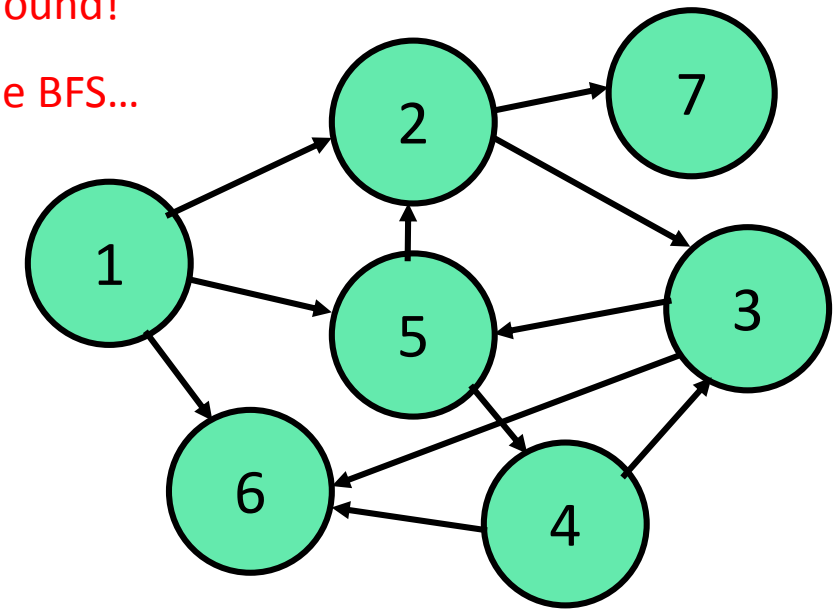
Breadth First Search (BFS)



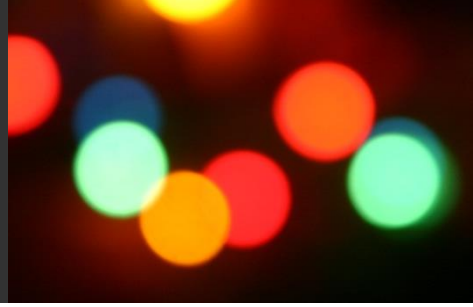
Queue	1	6	2	5	7	3	4

The path has been found!

But let's complete the BFS…



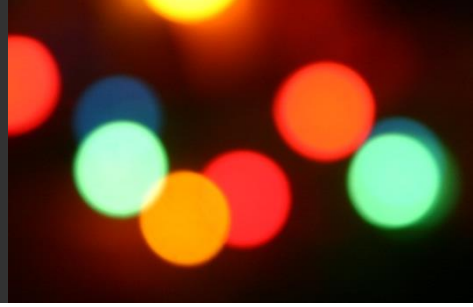
Breadth First Search (BFS)



```
while queue Q not empty
    dequeue the first node u from Q
    for each adjacent node v from u
        if v is unvisited
            enqueue v to Q
            mark v as visited
```

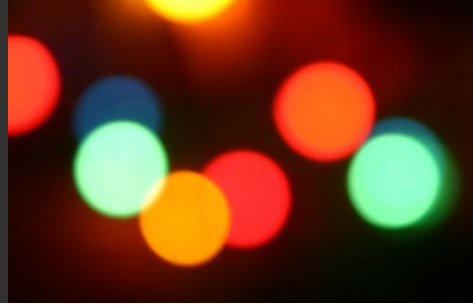
- Initialize all nodes as unvisited, except the starting node

Breadth First Search (BFS)



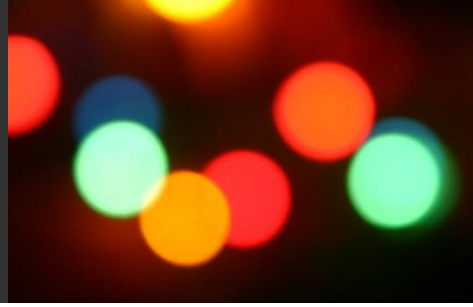
- Advantages
  - Shortest route is guaranteed on unweighted graphs
    - How about weighted graphs?
  - Avoid stack overflow
- Why don’t we always use BFS instead of DFS?

Question



- Yes, you can show that there exists a path to the destination, but how to trace the path?

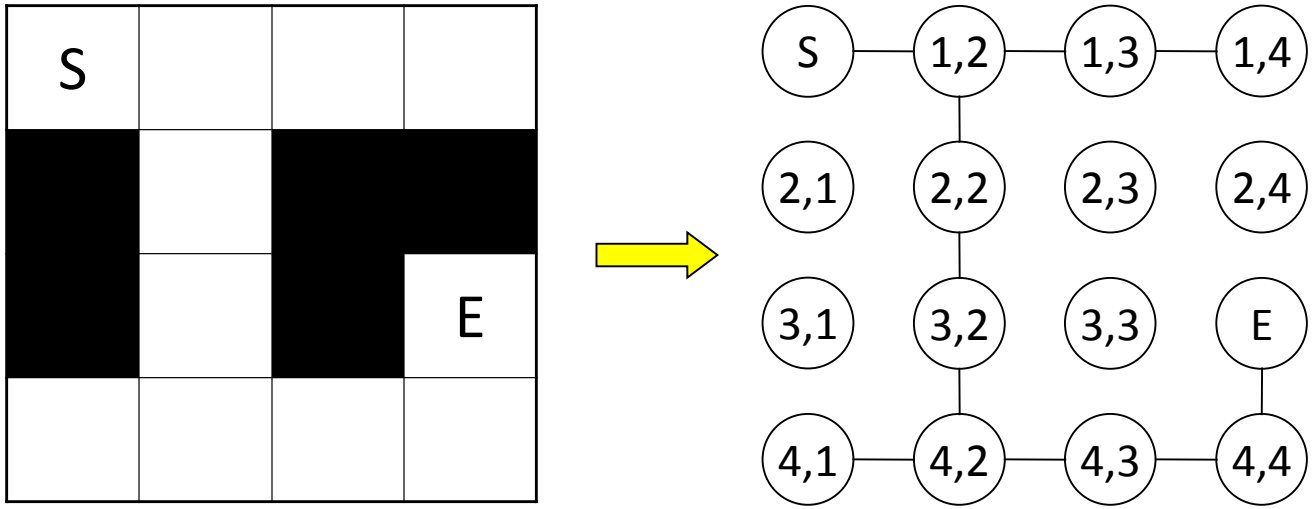
Let’s have a 10-minute break!



- We will discuss some basic applications of graphs

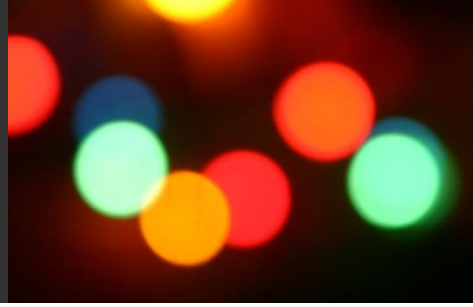
Maze

- Given an N x N maze, find the shortest route from S to E.



- BFS

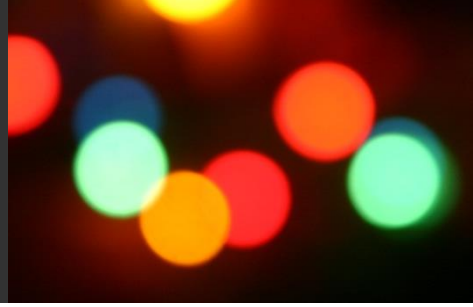
Maze



- In fact you don’t even need to store the graph in this way
- The changes are the current coordinates only
- Just directly mark on the map!

- Use the coordinates to represent the ***states*** **(狀態)** when you do BFS

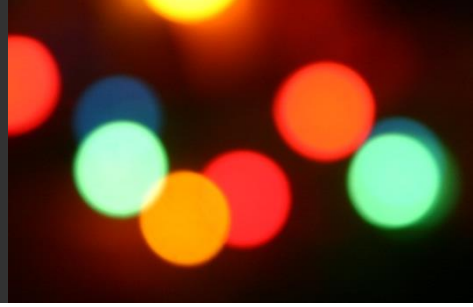
Variants of maze problems



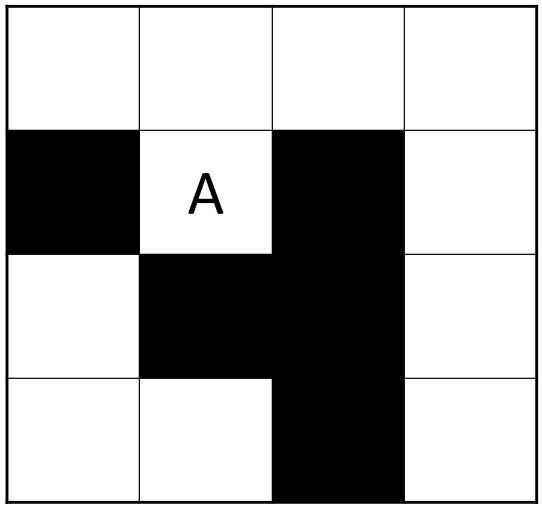
- Different scenarios:
  - You may have a few bombs to destroy walls…
  - Someone is chasing after you, you need to escape without being caught…
  - Walls can be pushed forward…

- No matter how complicated it is, the main concern is how you should represent a ***state***

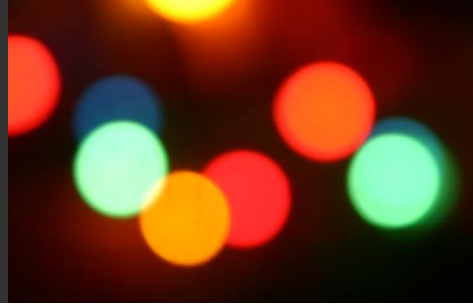
Finding area



- Find the area that are reachable from A.

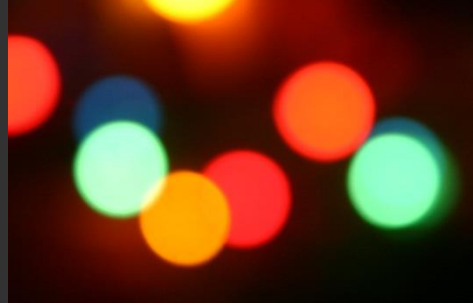


Flood fill



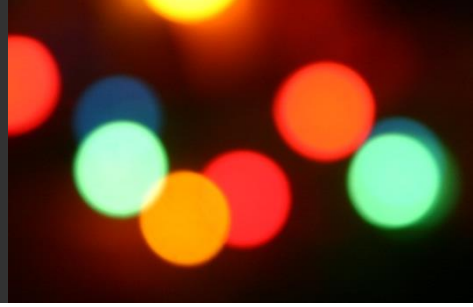
- Starting from 1 vertex, use DFS / BFS to repeatedly visit (or fill) the adjacent vertices
- Count the number of visited vertices while you do DFS / BFS

Teacher's Problem



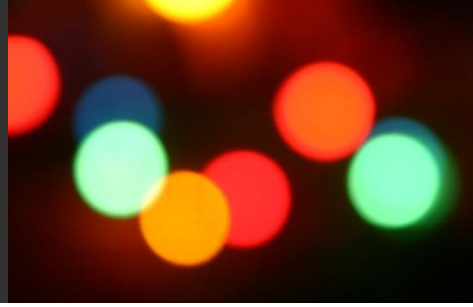
- HKOI 2004 Senior
- Emily wants to distribute candies to *N* students one by one, with a rule that if student A is teased by B, A can receive candy before B.
- Given lists of students teased by each students, find a possible sequence to give the candies

Teacher's Problem



- In short, in a **directed** graph,
  - We want to give a label to the vertices
  - So that if there is an edge from u to v, then u < v
  - Is it always possible?
- Finding such order is called "topological sort"

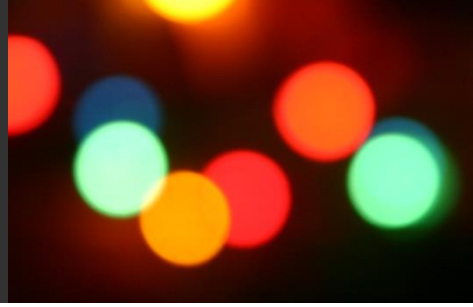
Topological sort



- Find a vertex with no incoming edges. Number it and remove all outgoing edges from it.
- Repeat the process.

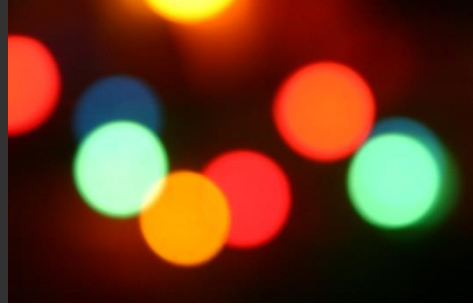
- How can it be implemented by:
  - DFS?
  - BFS?

More (if time allows)



- Iterative Deepening Search (IDS)
- Bidirectional Search (BDS)

Exercises & Sample Code



01035 Patrol Area (http://pastebin.com/Skw0aM93)

01067 Maze (http://pastebin.com/78UyTd1V)

01075 Sokoban

M0911 Theseus and the Minotaur

S034 Largest Continuous Region

S042 Teacher's Problem

T013 OIMan

T022 Bomber Man

T041 Amazing Robot II