CSCI-UA.0380-001
Programming Challenges

Sean McIntyre

Class 09: Graphs

Homework

Babelfish

- Given: key-pairs of words, the second of which translates to the first.
  - dog ogday
  - cat atcay
  - pig igpay
- Given: list of translated words, for each word output the original word
  - Or “eh” if it doesn't exist

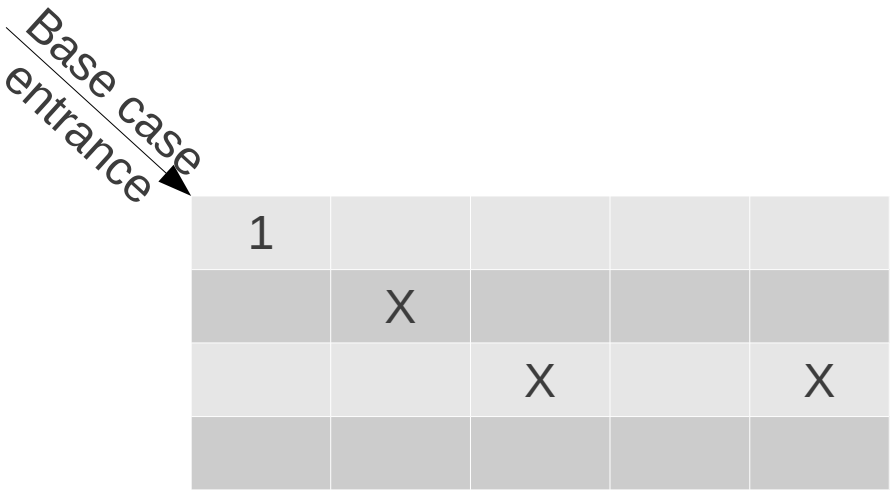
Walking on the Safe Side

- On a grid count how many ways there are to walk from (1, 1) to (N, M) in the quickest amount of time
  - Obstacles are placed along the way
- “Do not want to walk more than the required number of blocks”
  - i.e., no more than N+M blocks
  - Travel only down and right

Walking on the Safe Side

- Recurrence:
  - Each cell in dp[*row*][*column*] is the number of ways to get from (1, 1) to (*row*, *column*)
    - 1-based indices
  - dp[*row*][*column*] = dp[*row*-1][*column*] + dp[*row*][*column*-1]
    - i.e., the number of ways to get to (*row*, *column*) is the sum of the number of ways to the get to (*row*-1, *column*) and (*row*, *column*-1)

Walking on the Safe Side



Walking on the Safe Side

Base case entrance

1	1	1	1	1
	X			
		X		X

Walking on the Safe Side

Base case entrance

1	1	1	1	1
1	X	1	2	2
		X		X

Walking on the Safe Side

Base case entrance

1	1	1	1	1
1	X	1	2	2
1	1	X	2	X

Walking on the Safe Side

Base case entrance

1	1	1	1	1
1	X	1	2	2
1	1	X	2	X
1	2	2	4	4

Dividing Coins

- Given *m* coins, split the coins in two piles as evenly as possible
  - Output the minimal difference between the two piles
  - m <= 100, coins are between 1 and 500
- Observations:
  - Same problem as Bowen's
  - Each pile can be at most 25,000 cents

Dividing Coins

- Recurrence:
  - 2D table dp[coinIndex][totalAmount]
  - Mark dp[coinIndex][totalAmount] true if and only if it is possible to reach totalAmount in bag A by adding some combination of coins *i* = 1, 2, …, coinIndex
  - Easy to deduce bag B's totalAmount by knowing bag A's totalAmount

Dividing Coins

Total amount in bag A

Coins used
(1-based index)

	0	1	2	3	4	5	6	7	8	9	10
0	T										
1											
2											
3											

Coins: { 2, 3, 5 }

Dividing Coins

Total amount in bag A

Coins used
(1-based index)

	0	1	2	3	4	5	6	7	8	9	10
0	T										
1	T		T								
2											
3											

Coins: { 2, 3, 5 }

Dividing Coins

Total amount in bag A

Coins used
(1-based index)

	0	1	2	3	4	5	6	7	8	9	10
0	T										
1	T		T								
2	T		T	T		T					
3											

Coins: { 2, 3, 5 }

Dividing Coins

Total amount in bag A

Coins used
(1-based index)

	0	1	2	3	4	5	6	7	8	9	10
0	T										
1	T		T								
2	T		T	T		T					
3	T		T	T		T		T	T		T

Coins: { 2, 3, 5 }

Dividing Coins

Total amount in bag A

Coins used
(1-based index)

	0	1	2	3	4	5	6	7	8	9	10
0	T										
1	T		T								
2	T		T	T		T					
3	T		T	T		T		T	T		T

Coins: { 2, 3, 5 }

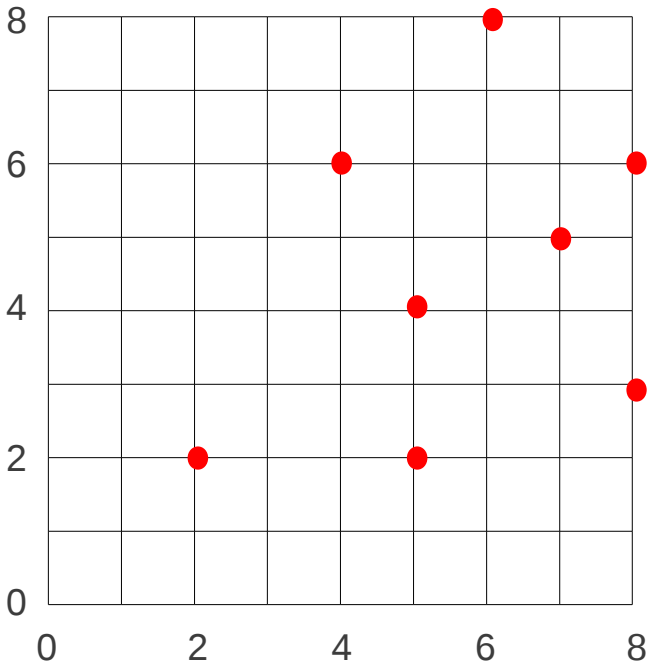
It's possible to place
half of the value of
the coins in bag A

Forming Quiz Teams

- Given 2*N points on a grid, make N pairs so that the sum of the distances of the paired points is minimized
  - 1 <= N <= 8, so 16 points

Forming Quiz Teams

- Example:



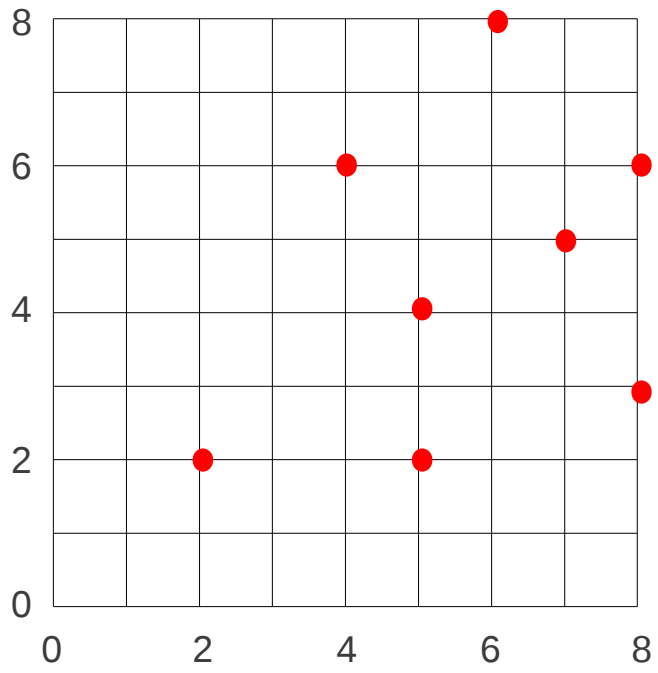
(2, 2)
(4, 6)
(5, 2)
(5, 4)
(6, 8)
(7, 5)
(8, 3)
(8, 6)

Forming Quiz Teams

- Recurrence:
  - dp[used grid points] = the minimum sum of distances between all remaining grid points
  - The answer is dp[all grid points]
  - The recursive step is to find the minimum sum by trying matching each pair of remaining grid points
  - There are a lot of overlapping states, so store the subresults

Forming Quiz Teams

- Example:



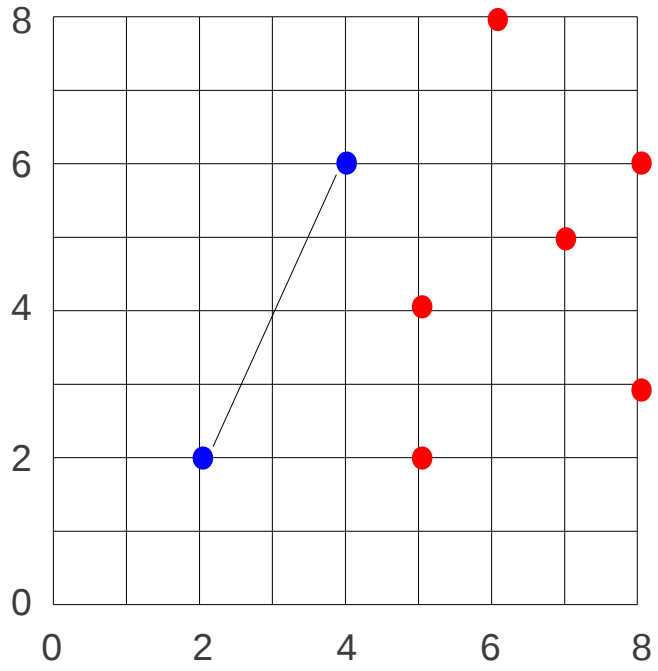
(2, 2)
(4, 6)
(5, 2)
(5, 4)
(6, 8)
(7, 5)
(8, 3)
(8, 6)

Recursion depth 0

What is the minimum sum for all the points?

Forming Quiz Teams

- Example:



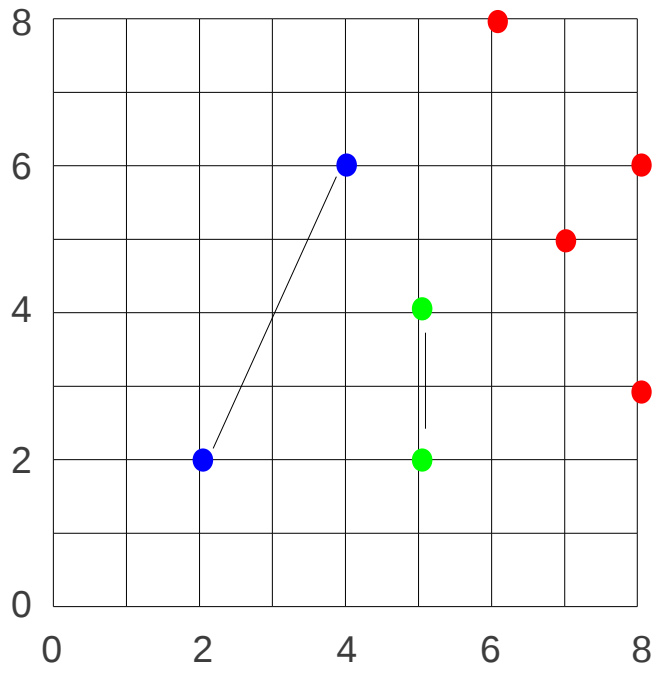
(2, 2)
(4, 6)
(5, 2)
(5, 4)
(6, 8)
(7, 5)
(8, 3)
(8, 6)

Recursion depth 1

What is the minimum sum for the 2nd, 3rd, 4th, 5th, 6th, and 7th points?

Forming Quiz Teams

- Example:



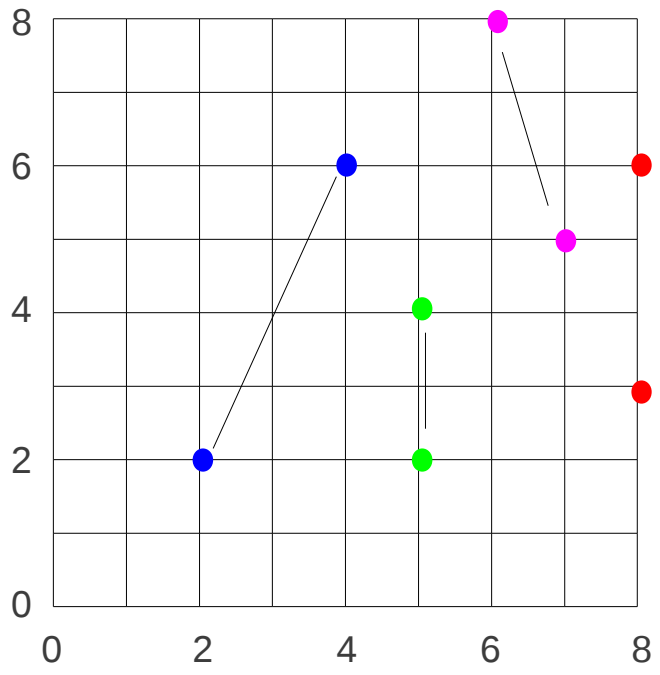
(2, 2)
(4, 6)
(5, 2)
(5, 4)
(6, 8)
(7, 5)
(8, 3)
(8, 6)

Recursion depth 2

What is the minimum sum for the 4th, 5th, 6th, and 7th points?

Forming Quiz Teams

- Example:



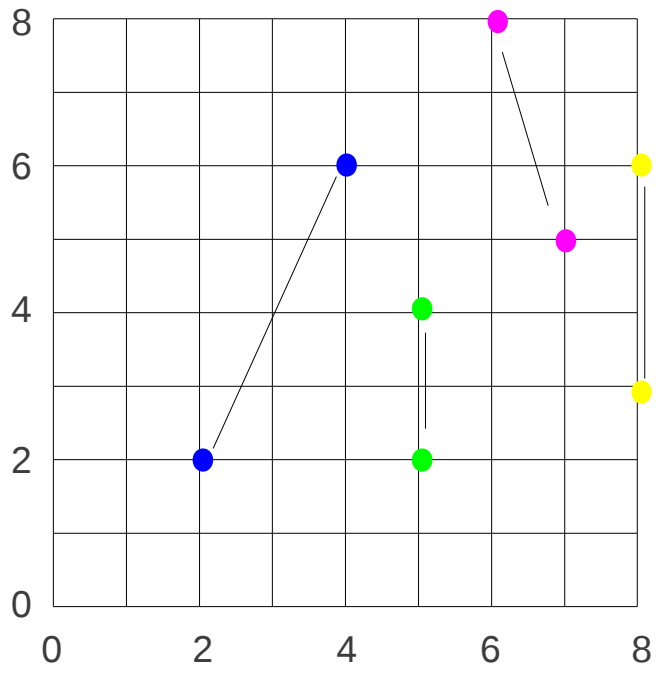
(2, 2)
(4, 6)
(5, 2)
(5, 4)
(6, 8)
(7, 5)
(8, 3)
(8, 6)

Recursion depth 3

What is the minimum sum for the 6th and 7th points?

Forming Quiz Teams

- Example:



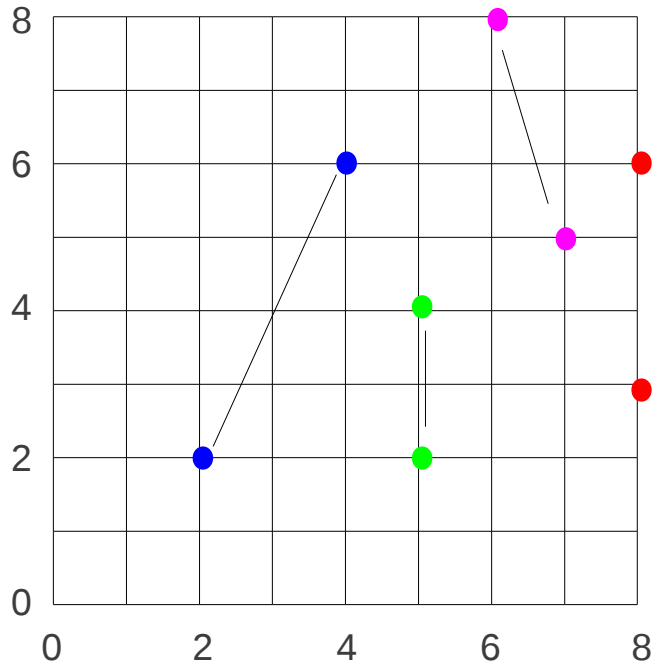
(2, 2)
(4, 6)
(5, 2)
(5, 4)
(6, 8)
(7, 5)
(8, 3)
(8, 6)

Recursion depth 4

What is the minimum sum no points?
Answer: Base case, 0

Forming Quiz Teams

- Example:



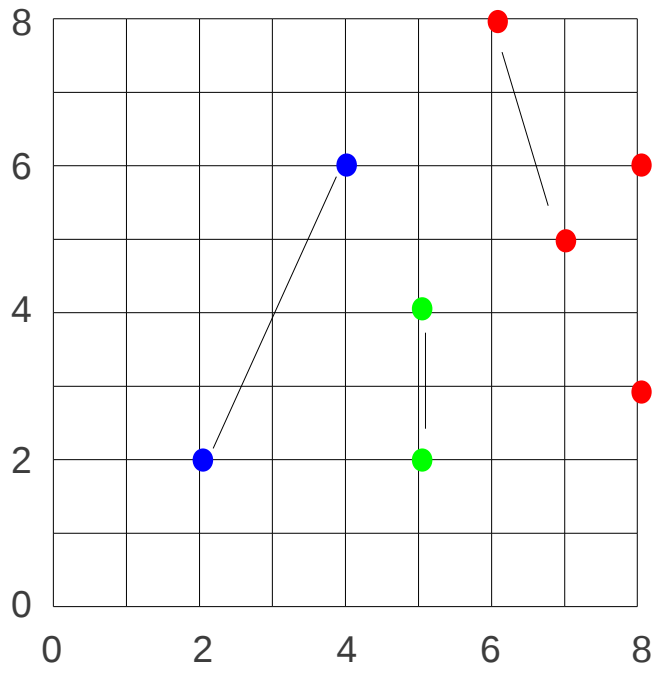
(2, 2)
(4, 6)
(5, 2)
(5, 4)
(6, 8)
(7, 5)
(8, 3)
(8, 6)

Recursion depth 3

What is the minimum sum for the 6th and 7th points?
Answer: 0 + dist(P[6], P[7]) = 3

Forming Quiz Teams

- Example:



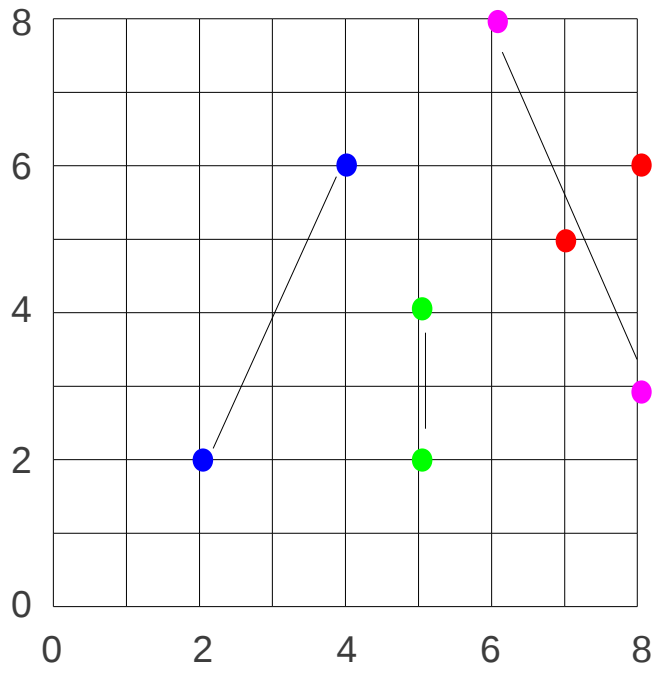
(2, 2)
(4, 6)
(5, 2)
(5, 4)
(6, 8)
(7, 5)
(8, 3)
(8, 6)

Recursion depth 2

What is the minimum sum for the 4th, 5th, 6th, and 7th points?
So far: 3 + dist(P[4], P[5]) = 3 + 3.16 = 6.16

Forming Quiz Teams

- Example:



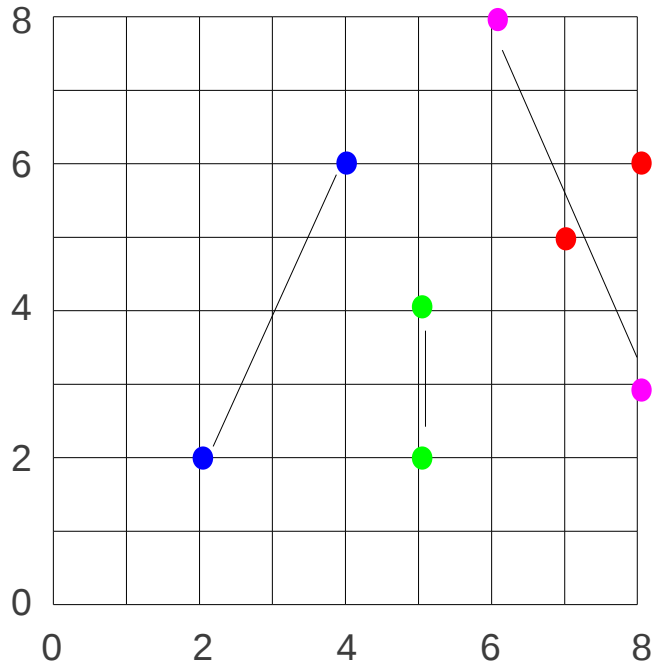
(2, 2)
(4, 6)
(5, 2)
(5, 4)
(6, 8)
(7, 5)
(8, 3)
(8, 6)

Recursion depth 3

What is the minimum sum for the 5th and 7th points?

Forming Quiz Teams

- Example:



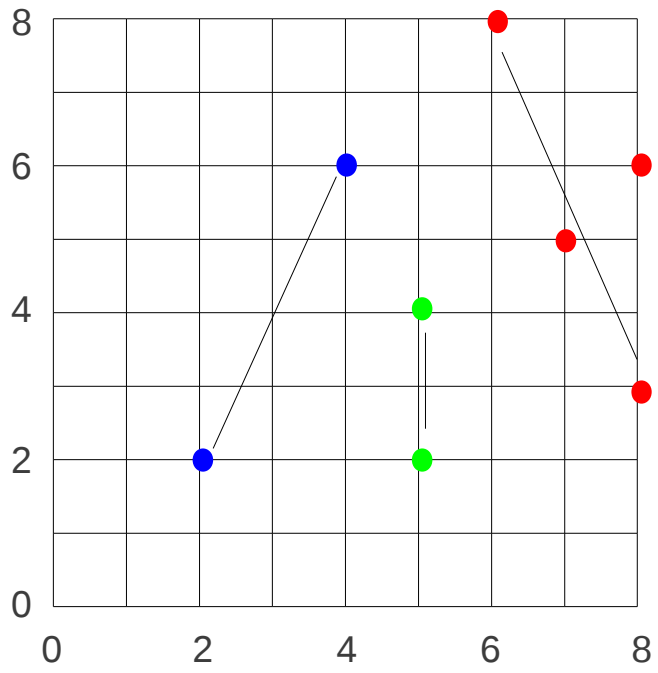
(2, 2)
(4, 6)
(5, 2)
(5, 4)
(6, 8)
(7, 5)
(8, 3)
(8, 6)

Recursion depth 3

What is the minimum sum for the 5th and 7th points?
Answer: 0 + dist(P[5], P[7]) = 1.41

Forming Quiz Teams

- Example:



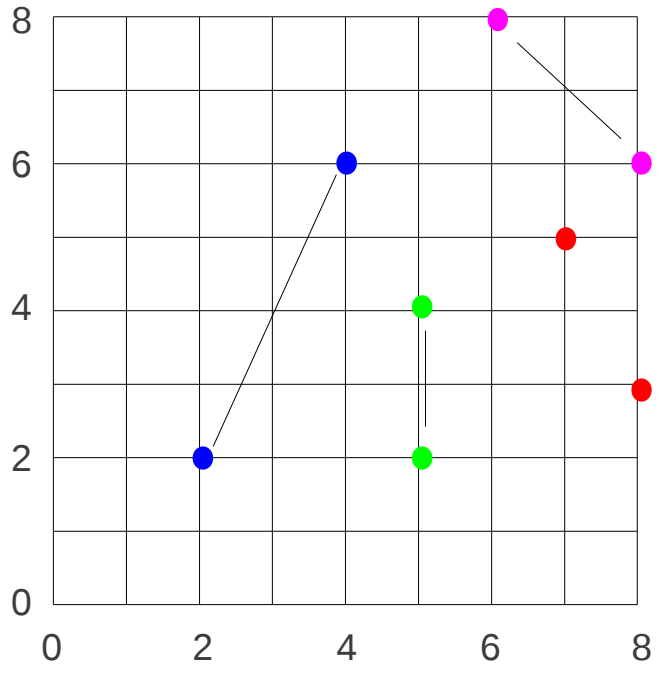
(2, 2)
(4, 6)
(5, 2)
(5, 4)
(6, 8)
(7, 5)
(8, 3)
(8, 6)

Recursion depth 2

What is the minimum sum for the 4th, 5th, 6th, and 7th points?
From before: 6.16
Just computed: 1.41 + dist(P[4], P[6]) = 1.41 + 5.39 = 6.80
Still: 6.16

Forming Quiz Teams

- Example:



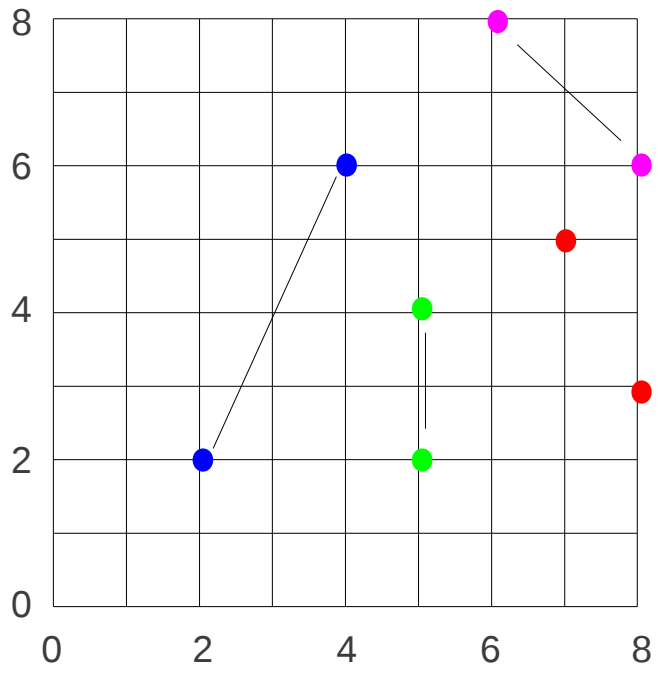
(2, 2)
(4, 6)
(5, 2)
(5, 4)
(6, 8)
(7, 5)
(8, 3)
(8, 6)

Recursion depth 3

What is the minimum sum for the 5th and 6th points?

Forming Quiz Teams

- Example:



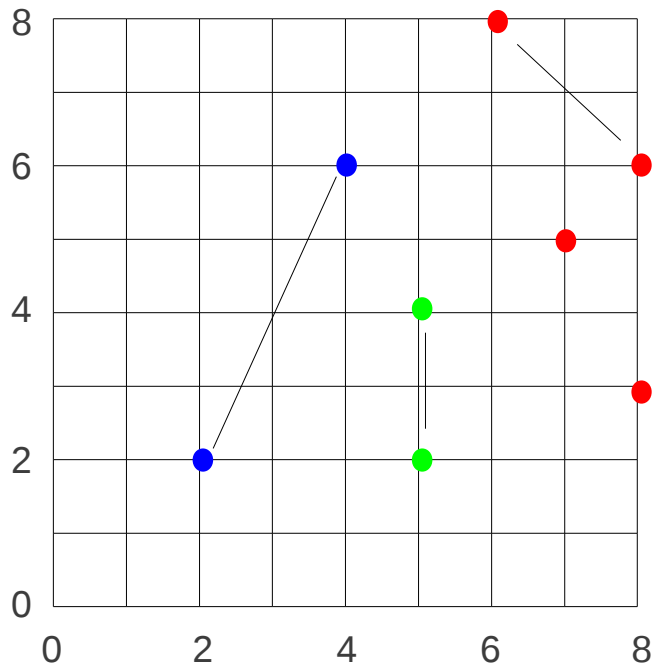
(2, 2)
(4, 6)
(5, 2)
(5, 4)
(6, 8)
(7, 5)
(8, 3)
(8, 6)

Recursion depth 3

What is the minimum sum for the 5th and 6th points?
Answer: **2.24**

Forming Quiz Teams

- Example:



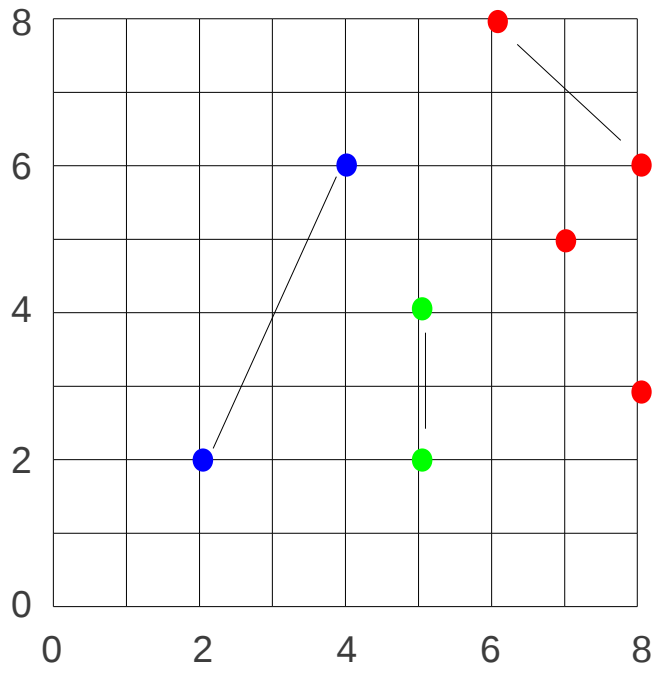
(2, 2)
(4, 6)
(5, 2)
(5, 4)
(6, 8)
(7, 5)
(8, 3)
(8, 6)

Recursion depth 2

What is the minimum sum for the 4th, 5th, 6th, and 7th points?
From before: 6.16
Just computed: 2.24 + dist(P[4], P[7]) = 5.07
Now: 5.07

Forming Quiz Teams

- Example:



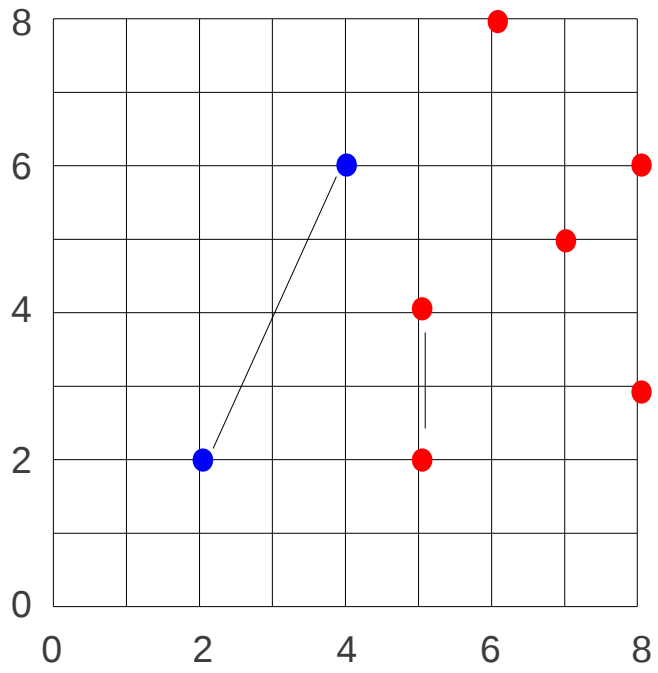
(2, 2)
(4, 6)
(5, 2)
(5, 4)
(6, 8)
(7, 5)
(8, 3)
(8, 6)

Recursion depth 2

What is the minimum sum for the 4th, 5th, 6th, and 7th points?
We have tried everything, so we can definitively answer 5.07.

Forming Quiz Teams

- Example:



(2, 2)
(4, 6)
(5, 2)
(5, 4)
(6, 8)
(7, 5)
(8, 3)
(8, 6)

Recursion depth 1

What is the minimum sum for the 2nd, 3rd, 4th, 5th, 6th, and 7th points?

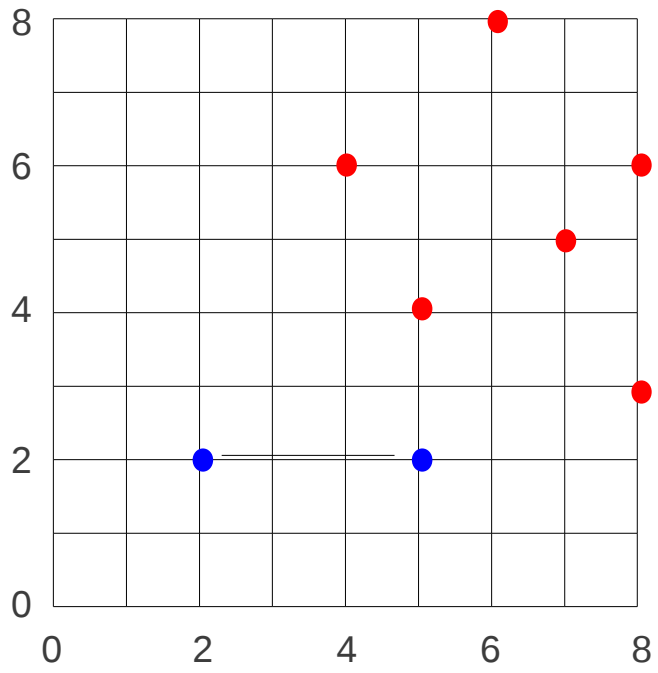
So far: 5.07 + dist(P[2], P[3]) = 7.07

Forming Quiz Teams

- Fast-forward...

Forming Quiz Teams

- Example:



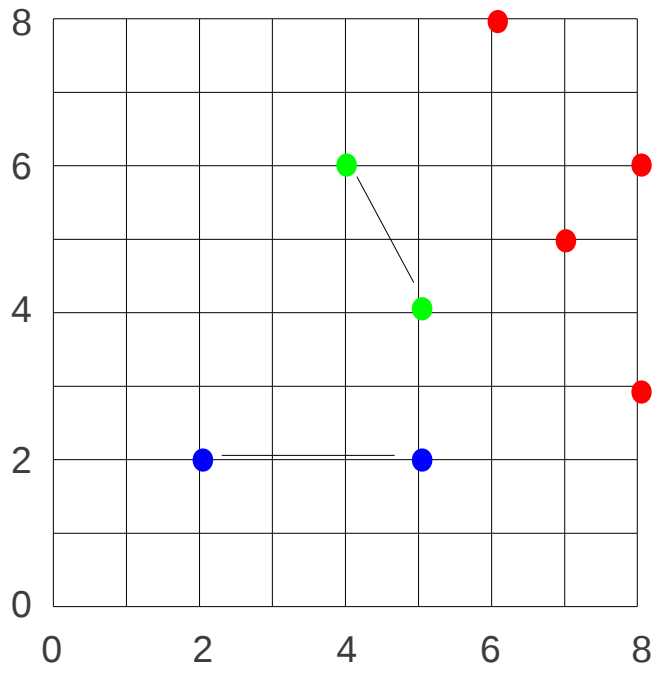
(2, 2)
(4, 6)
(5, 2)
(5, 4)
(6, 8)
(7, 5)
(8, 3)
(8, 6)

Recursion depth 1

What is the minimum sum for the 1st, 3rd, 4th, 5th, 6th, and 7th points?

Forming Quiz Teams

- Example:



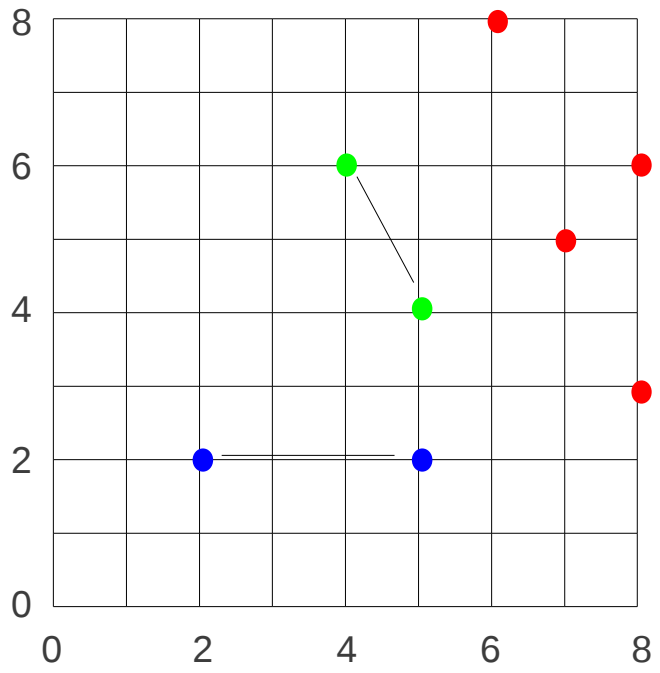
(2, 2)
(4, 6)
(5, 2)
(5, 4)
(6, 8)
(7, 5)
(8, 3)
(8, 6)

Recursion depth 2

What is the minimum sum for the 4th, 5th, 6th, and 7th points?

Forming Quiz Teams

- Example:



(2, 2)
(4, 6)
(5, 2)
(5, 4)
(6, 8)
(7, 5)
(8, 3)
(8, 6)

Recursion depth 2

What is the minimum sum for the 4th, 5th, 6th, and 7th points?
Answer: 5.07 (from the memoization table)

Forming Quiz Teams

- How to implement:
  - Use bitmasks!
  - Use a memoization table with 2^16 elements
  - Each entry in the table is considered a bitmask representing the set of all grid points chosen
- Another similar DP solution is the O(2^n * n) solution to the Traveling Salesman Problem

Forming Quiz Teams

```
int N;
int x[] = new int[16], y[] = new int[16]; // grid coordinates
double dp[] = new double[1 << 16];        // 2^16 entries

public double solve(int mask) {
    if (dp[mask] >= 0) return dp[mask];   // memoization step
    double res = INFINITY;

    for (int i = 0; i < 2*N; i++) {
        for (int j = i+1; j < 2*N; j++) { // filters out permutations
            if ((((1 << i) | (1 << j)) & mask) == 0) { // unused set elmnts
                double dist = sqrt(pow(x[i] - x[j], 2)
                                   + pow(y[i] - y[j], 2));
                res = min(res, solve(mask | (1 << i) | (1 << j)));
            }
        }
    }

    return dp[mask] = res; // store the solution in memo table
}

public void main() {            // left out the parsing details
    dp[(1 << (N*2)) - 1] = 0.0; // base case: all points used = 0 min dist
    System.out.printf(solve(0)); // 0 = empty bit mask = all points rem
}
```

Happy Number

- Square the digits of a number
  - If the result is 1, it's a happy number
  - If the result loops, it's an unhappy number
  - If it's unclear, then repeat the process
- Have to store the subresults as you go

Super Number

- If the x leftmost digits of N divide x for $x = n$, $n+1$, $n+2$, …, m, then N is a super number
- Find the <u>lexicographically</u> smallest N for n and m
  - $0 < n < m < 30$
- Kind of a trick question
  - The brute force approach is a little out of the runtime, but the number of possible inputs is 30x30 = 900
    - so pre-compute the answers and submit a table :)

Lecture

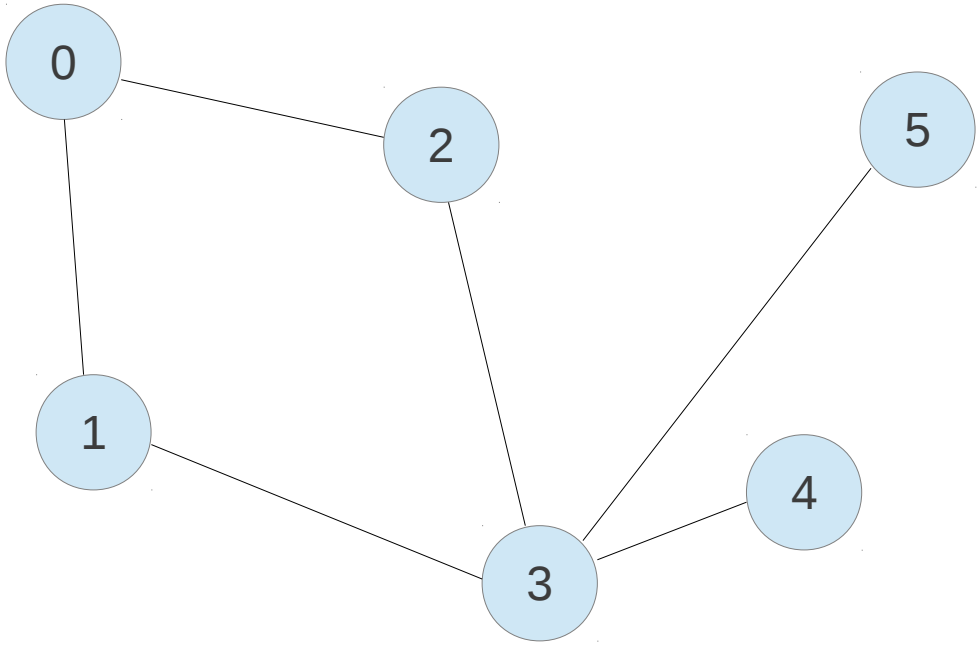
Graph Traversal Algorithms

- Many problems rely on traversing elements in a graph
  - e.g., UVa 469 – Wetlands of Florida
    - You're given a 2D grid, each cell of which can be “water” or “land”
    - Cells adjacent on the major axes or diagonals are adjacent
    - For a given water (x, y) coordinate on the grid, determine the area of the connected water
  - These problems look hard if you're not familiar with graph traversals

Graph Traversal Algorithms

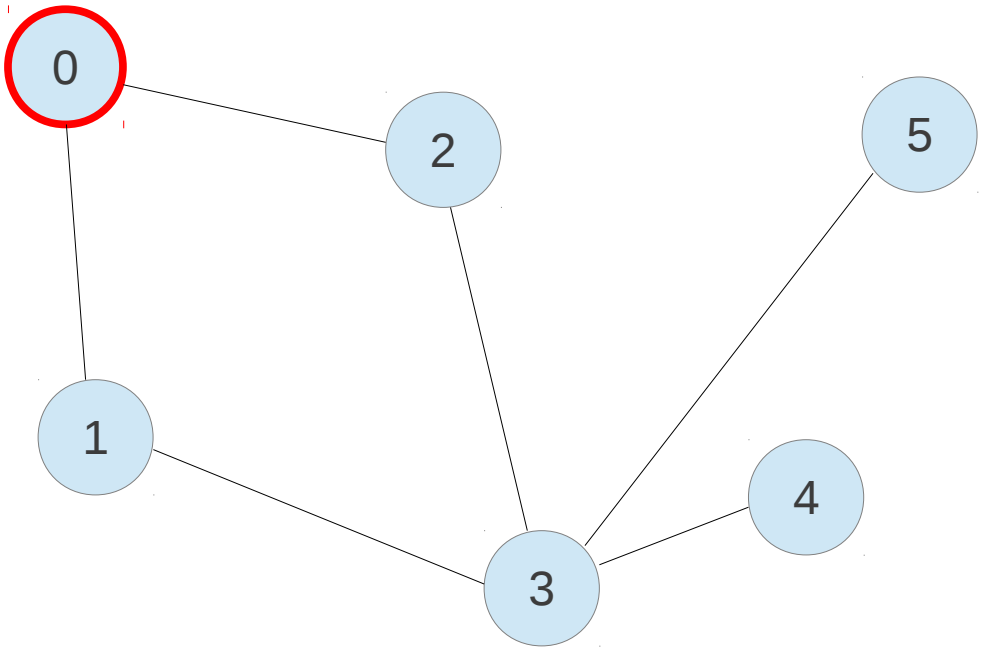
- Depth-first search
  - The first and most natural way to solve this problem is by visiting every node using recursion
  - As the name implies, visit the furthest nodes from the originating node
  - Perform backtracking

Graph Traversal Algorithms



<u>Adjacency list</u>
0: 1, 2
1: 1, 3
2: 0, 3
3: 1, 2, 4, 5
4: 3
5: 3

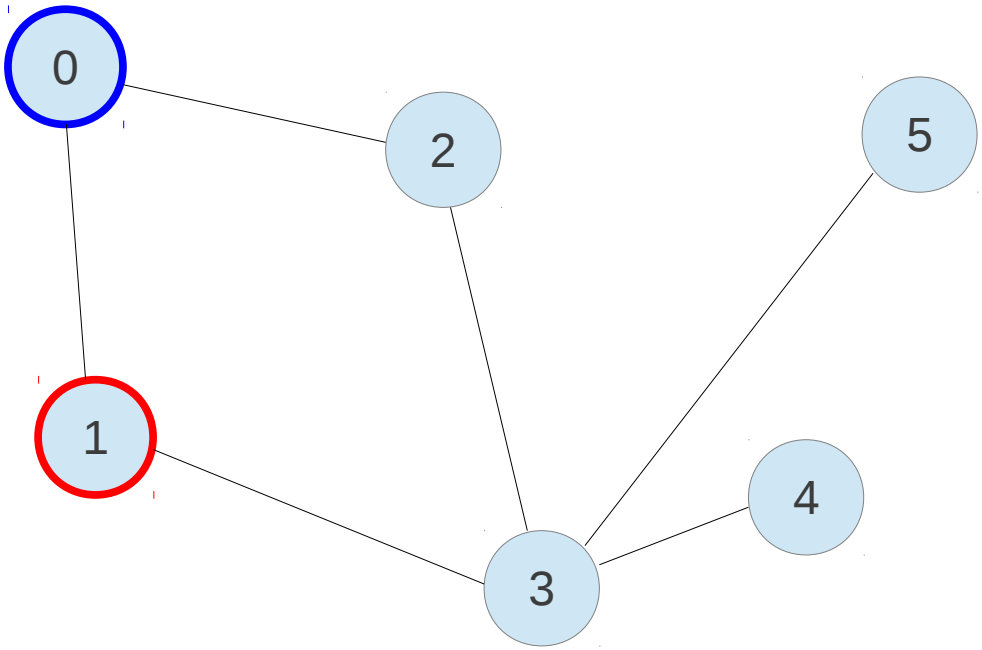
Graph Traversal Algorithms



<u>Adjacency list</u>
0: 1, 2
1: 1, 3
2: 0, 3
3: 1, 2, 4, 5
4: 3
5: 3

<u>Stack</u>
dfs(0)

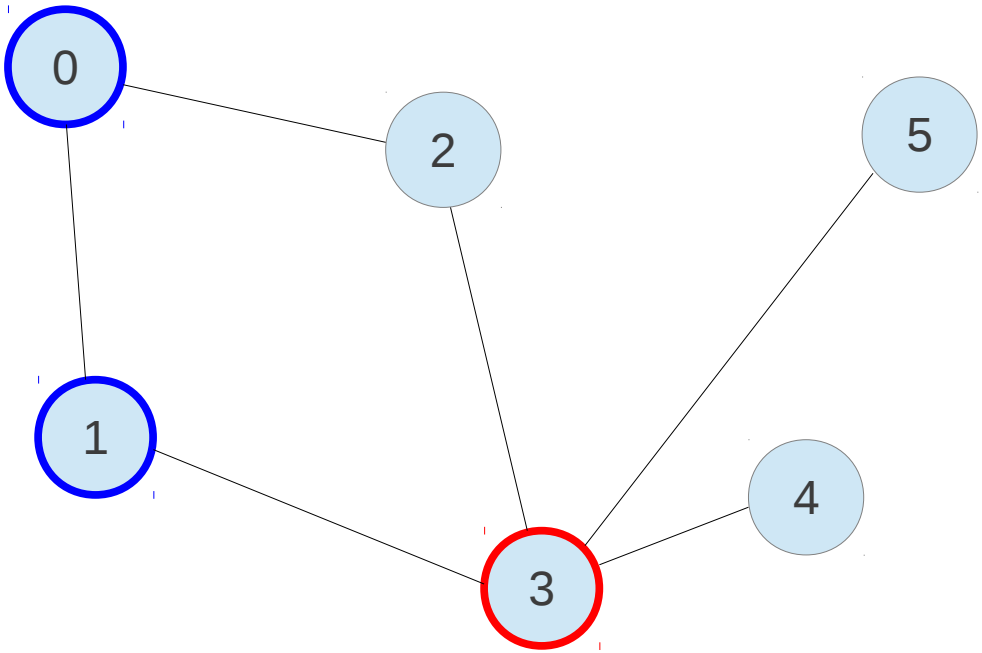
Graph Traversal Algorithms



Adjacency list
0: 1, 2
1: 1, 3
2: 0, 3
3: 1, 2, 4, 5
4: 3
5: 3

Stack
dfs(0)
dfs(1)

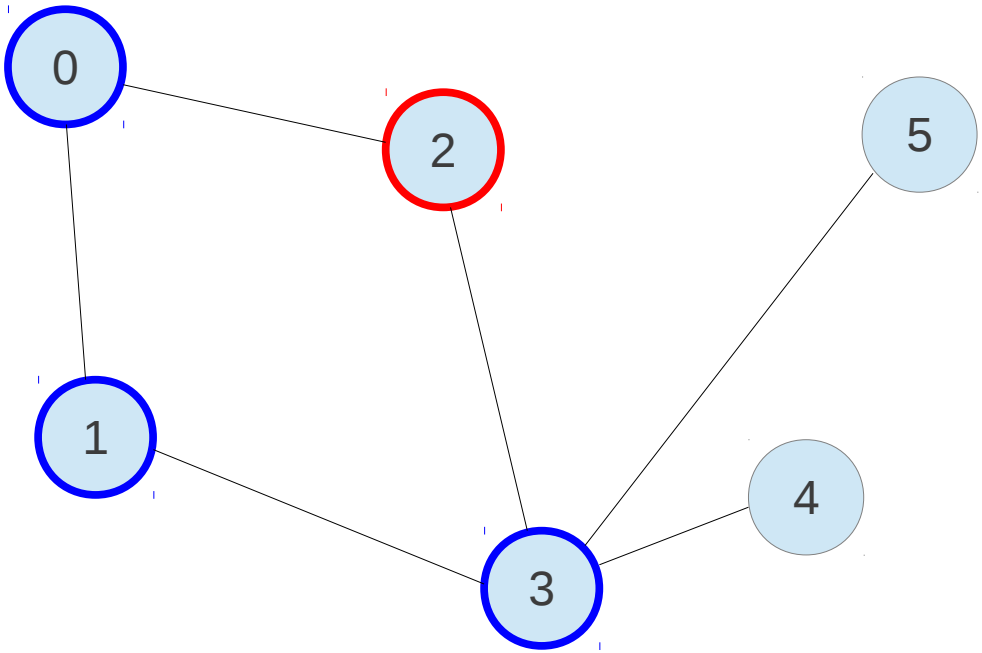
Graph Traversal Algorithms



<u>Adjacency list</u>
0: 1, 2
1: 1, 3
2: 0, 3
3: 1, 2, 4, 5
4: 3
5: 3

<u>Stack</u>
dfs(0)
dfs(1)
dfs(3)

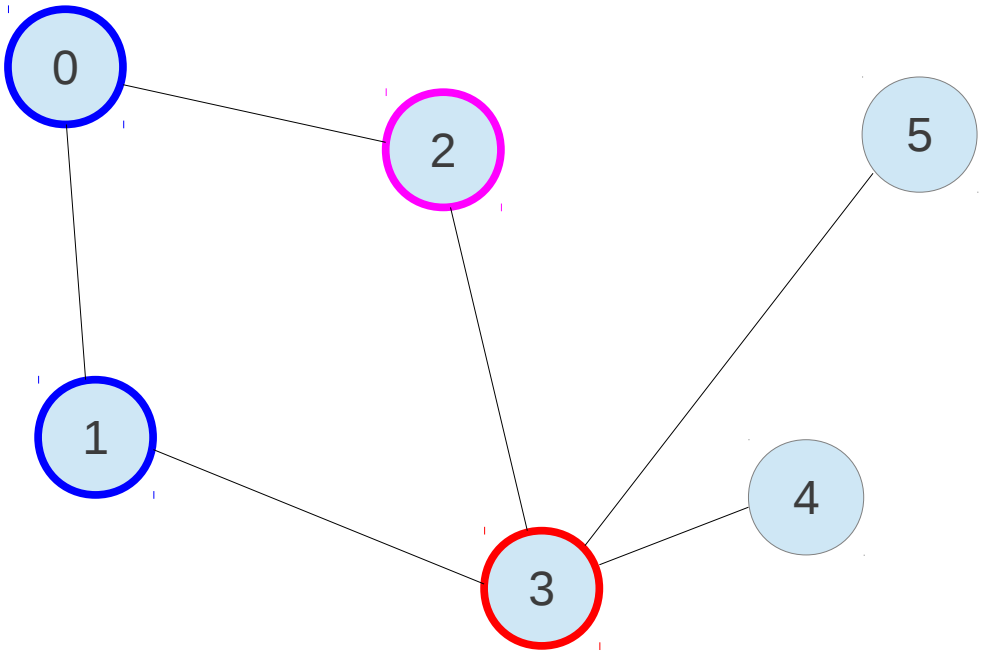
Graph Traversal Algorithms



Adjacency list
0: 1, 2
1: 1, 3
2: 0, 3
3: 1, 2, 4, 5
4: 3
5: 3

Stack
dfs(0)
dfs(1)
dfs(3)
dfs(2)

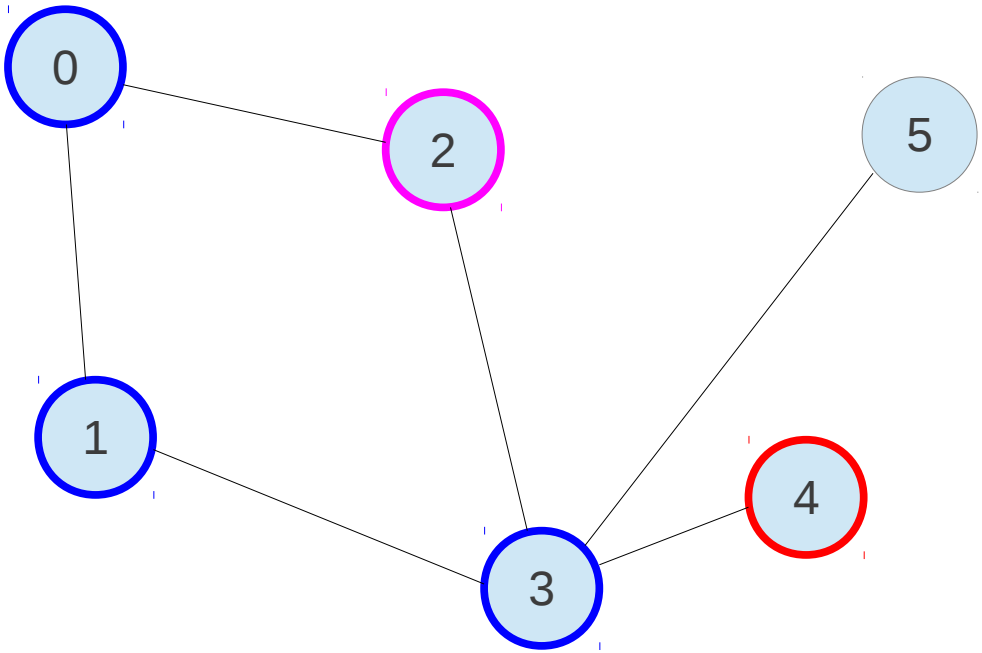
Graph Traversal Algorithms



Adjacency list
0: 1, 2
1: 1, 3
2: 0, 3
3: 1, 2, 4, 5
4: 3
5: 3

Stack
dfs(0)
dfs(1)
dfs(3)

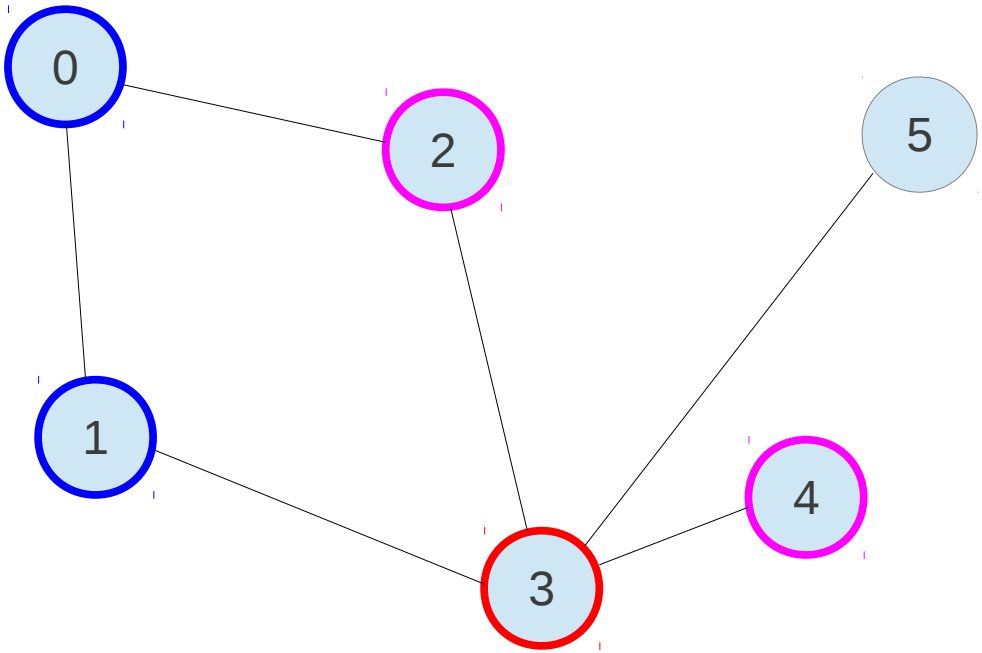
Graph Traversal Algorithms



<u>Adjacency list</u>
0: 1, 2
1: 1, 3
2: 0, 3
3: 1, 2, 4, 5
4: 3
5: 3

<u>Stack</u>
dfs(0)
dfs(1)
dfs(3)
dfs(4)

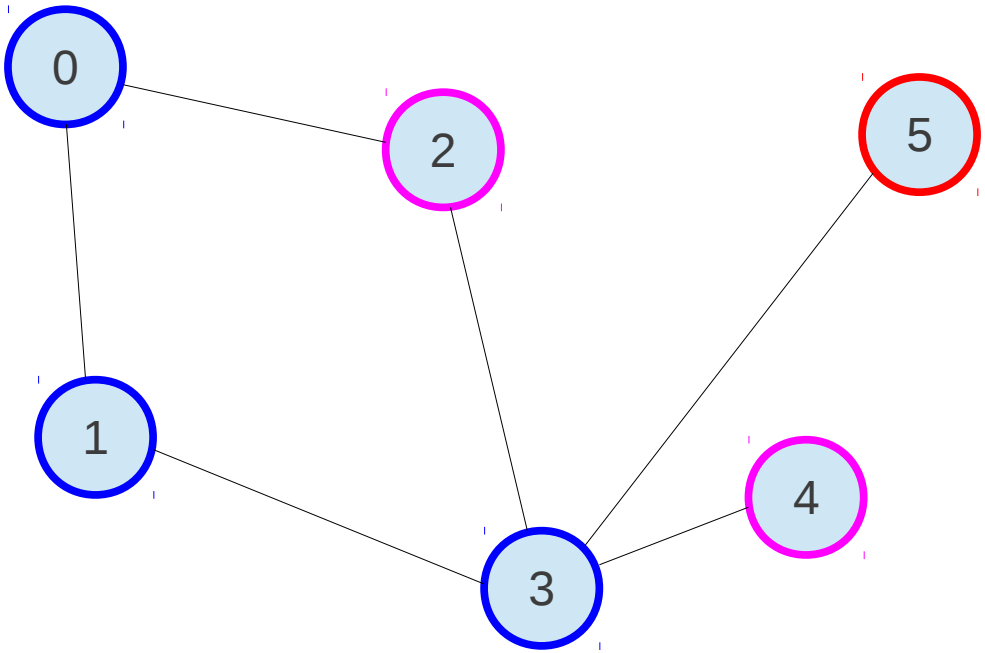
Graph Traversal Algorithms



<u>Adjacency list</u>
0: 1, 2
1: 1, 3
2: 0, 3
3: 1, 2, 4, 5
4: 3
5: 3

<u>Stack</u>
dfs(0)
dfs(1)
dfs(3)

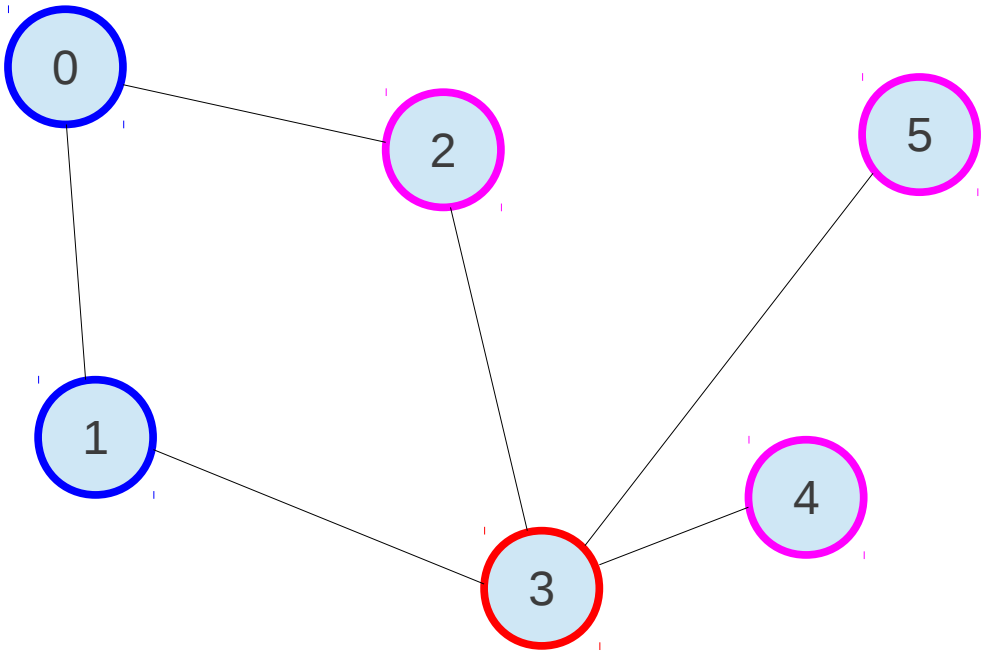
Graph Traversal Algorithms



Adjacency list
0: 1, 2
1: 1, 3
2: 0, 3
3: 1, 2, 4, 5
4: 3
5: 3

Stack
dfs(0)
dfs(1)
dfs(3)
dfs(5)

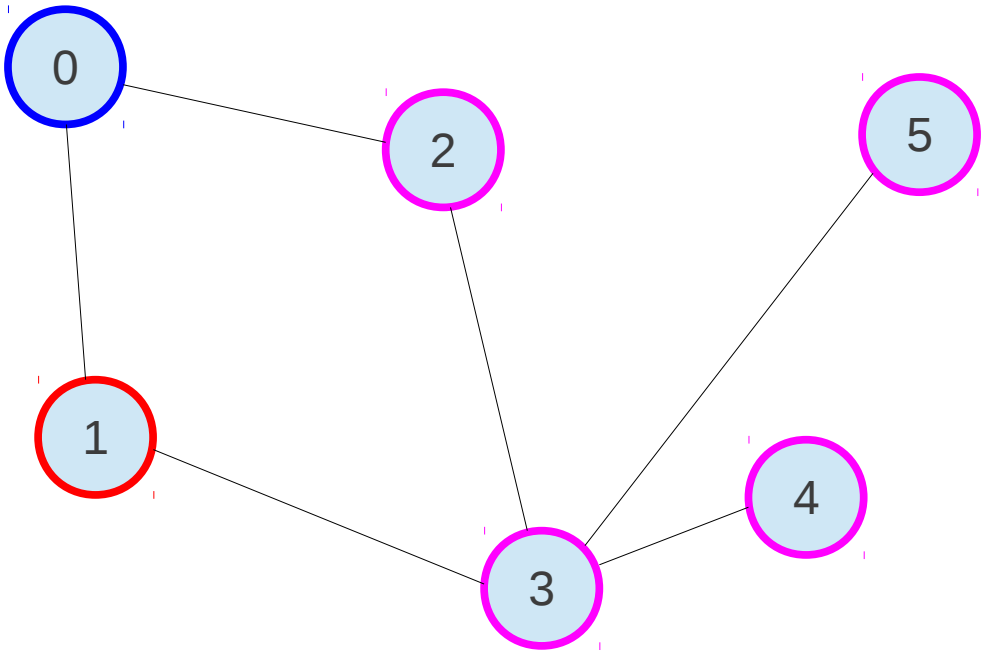
Graph Traversal Algorithms



Adjacency list
0: 1, 2
1: 1, 3
2: 0, 3
3: 1, 2, 4, 5
4: 3
5: 3

Stack
dfs(0)
dfs(1)
dfs(3)

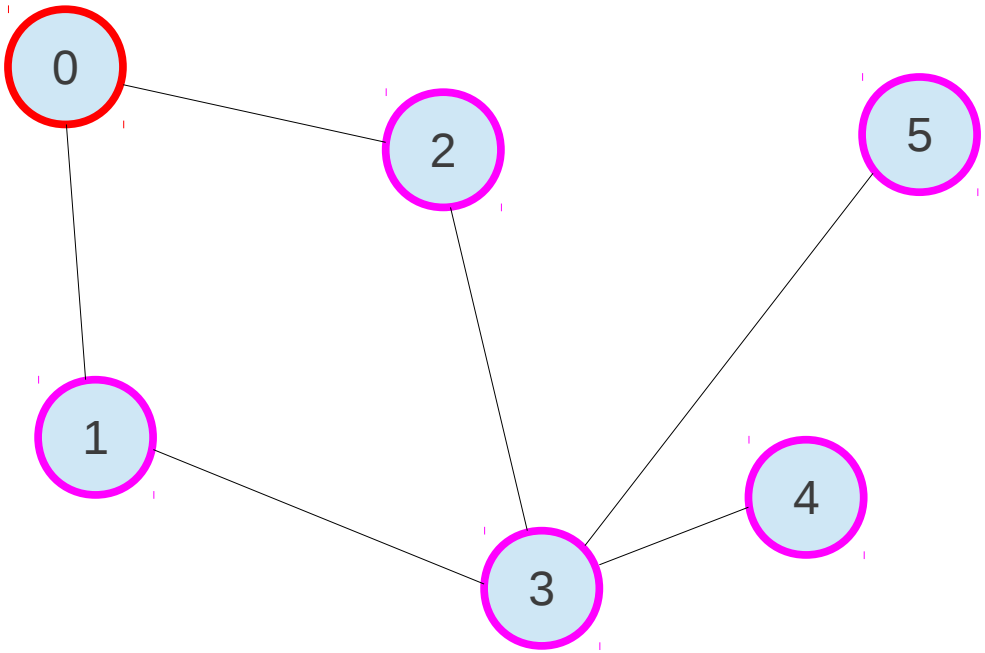
Graph Traversal Algorithms



<u>Adjacency list</u>
0: 1, 2
1: 1, 3
2: 0, 3
3: 1, 2, 4, 5
4: 3
5: 3

<u>Stack</u>
dfs(0)
dfs(1)

Graph Traversal Algorithms



<u>Adjacency list</u>
0: 1, 2
1: 1, 3
2: 0, 3
3: 1, 2, 4, 5
4: 3
5: 3

<u>Stack</u>
dfs(0)

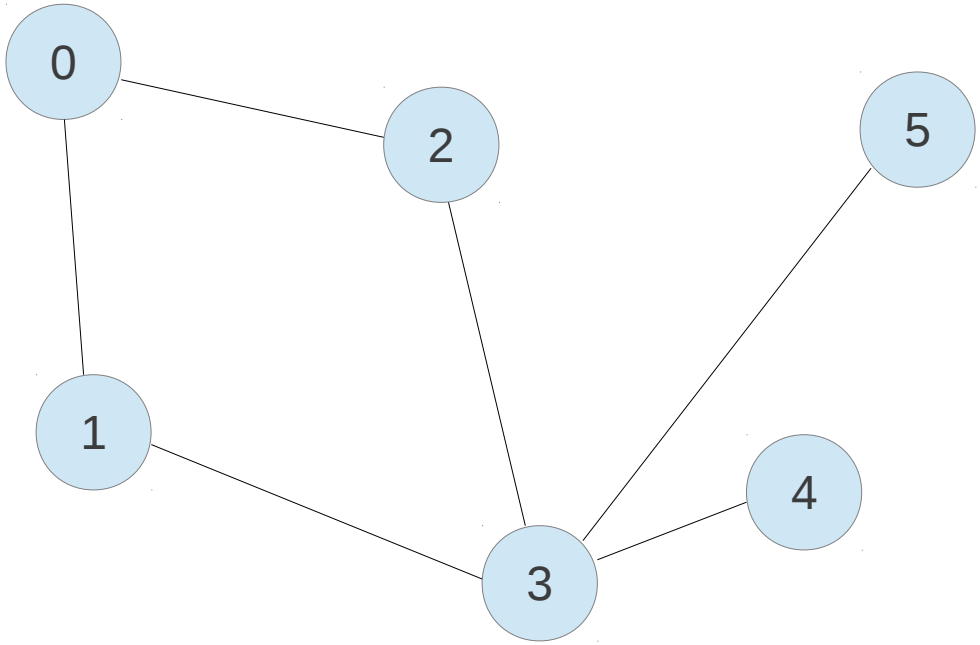
Graph Traversal Algorithms

```
ArrayList<ArrayList<Integer>> adjList; // prefilled with adjacents

int dfs(int node) { // returns # of nodes visited from node idx
    int res = 0;

    visited[node] = true; // mark this node as visited

    for (int i = 0; i < adjList.get(node).size(); i++) {
        int neighbor = adjList.get(node).get(i);
        if (!visited[neighbor]) {
            res += dfs(neighbor); // add number of dfs nodes visited
        }
    }

    return res+1; // the +1 refers to visiting the present node
}

System.out.println(dfs(0));
```

Graph Traversal Algorithms

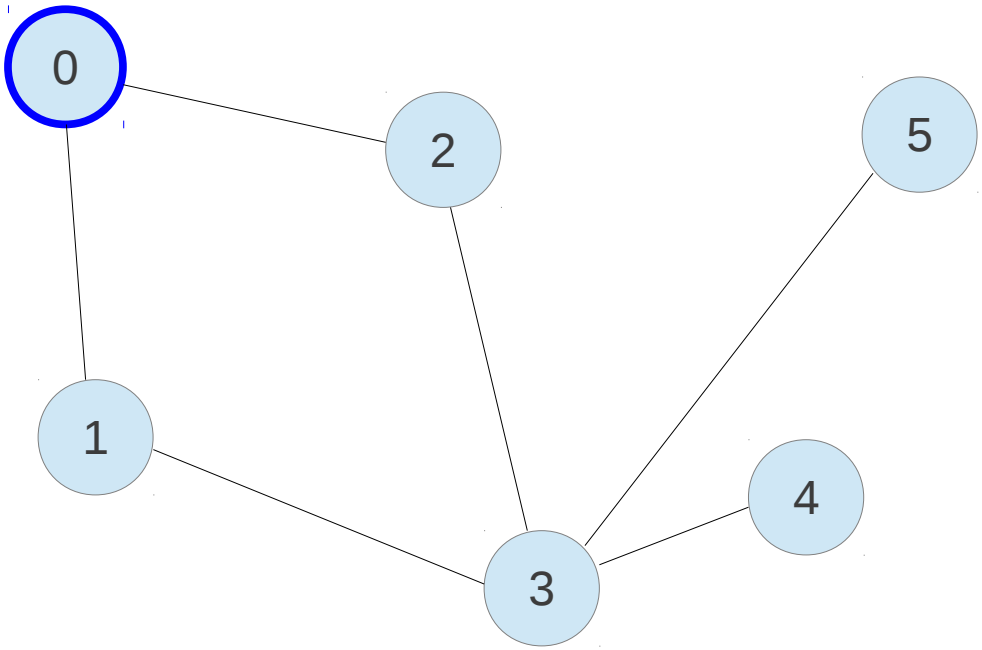
- Breadth-first search
  - Visit nodes closest to the originating node before diving down into the tree
  - Implemented using Queue
  - Typically used to solve “shortest path” problems in special cases
    - More later

Graph Traversal Algorithms



<u>Adjacency list</u>
0: 1, 2
1: 1, 3
2: 0, 3
3: 1, 2, 4, 5
4: 3
5: 3

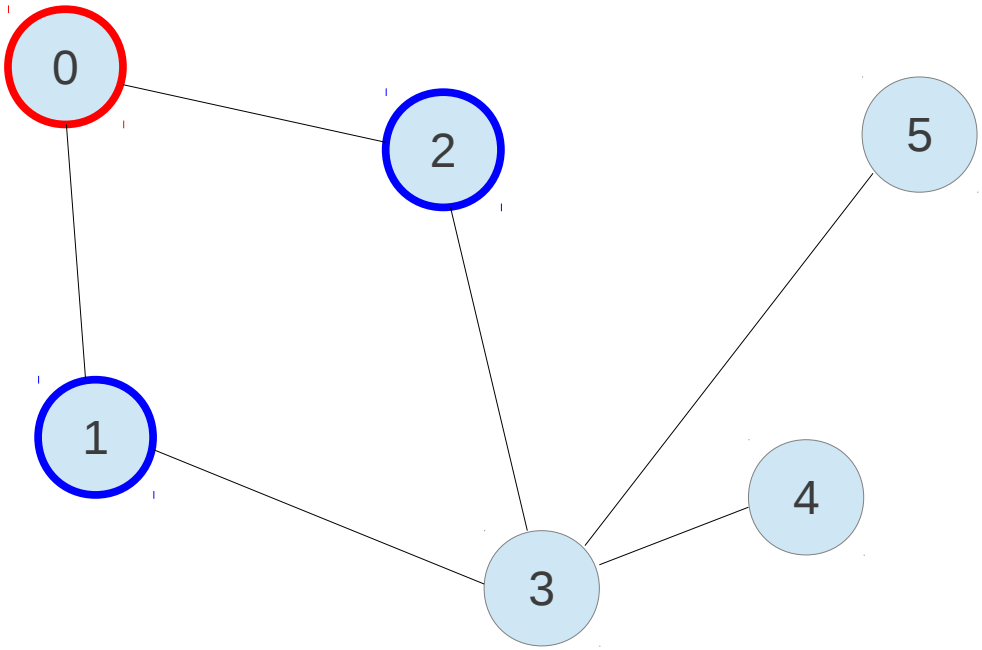
Graph Traversal Algorithms



<u>Adjacency list</u>
0: 1, 2
1: 1, 3
2: 0, 3
3: 1, 2, 4, 5
4: 3
5: 3

<u>Queue</u>
0

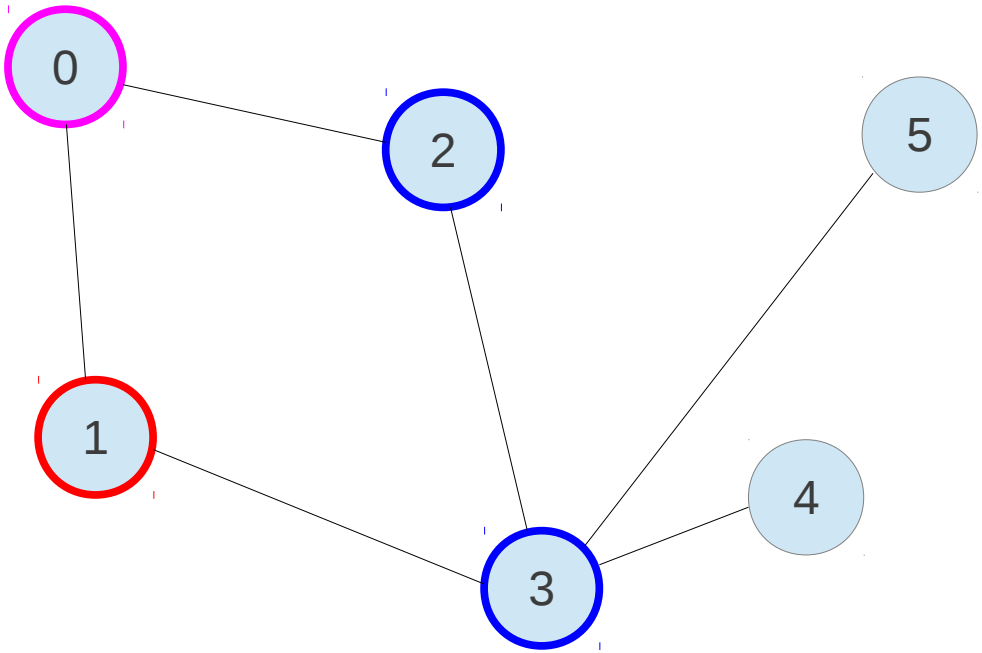
Graph Traversal Algorithms



<u>Adjacency list</u>
0: 1, 2
1: 1, 3
2: 0, 3
3: 1, 2, 4, 5
4: 3
5: 3

<u>Queue</u>
1
2

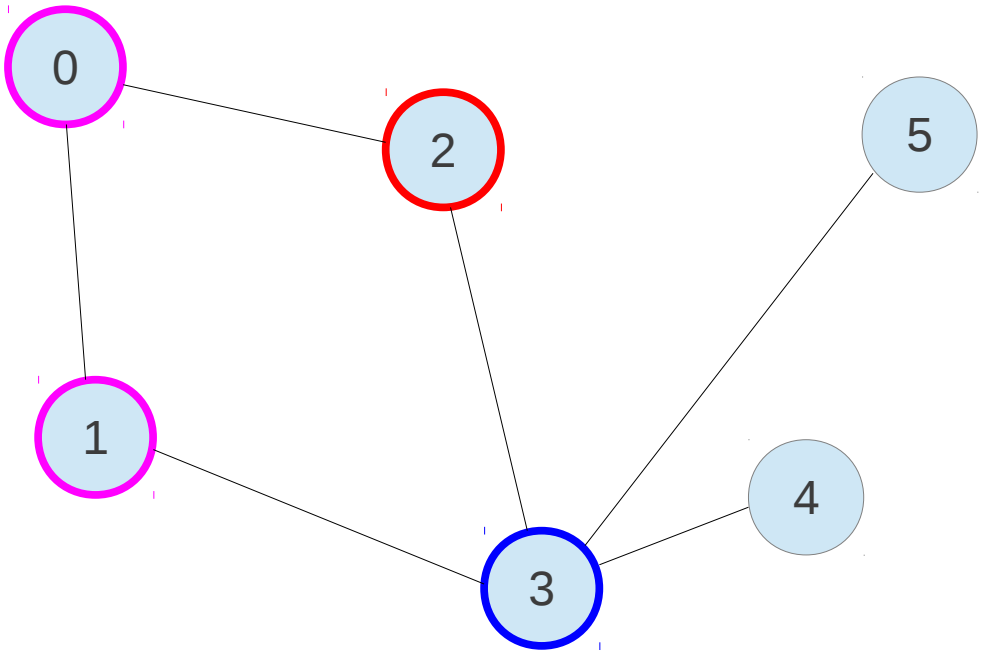
Graph Traversal Algorithms



Adjacency list
0: 1, 2
1: 1, 3
2: 0, 3
3: 1, 2, 4, 5
4: 3
5: 3

Queue
2
3

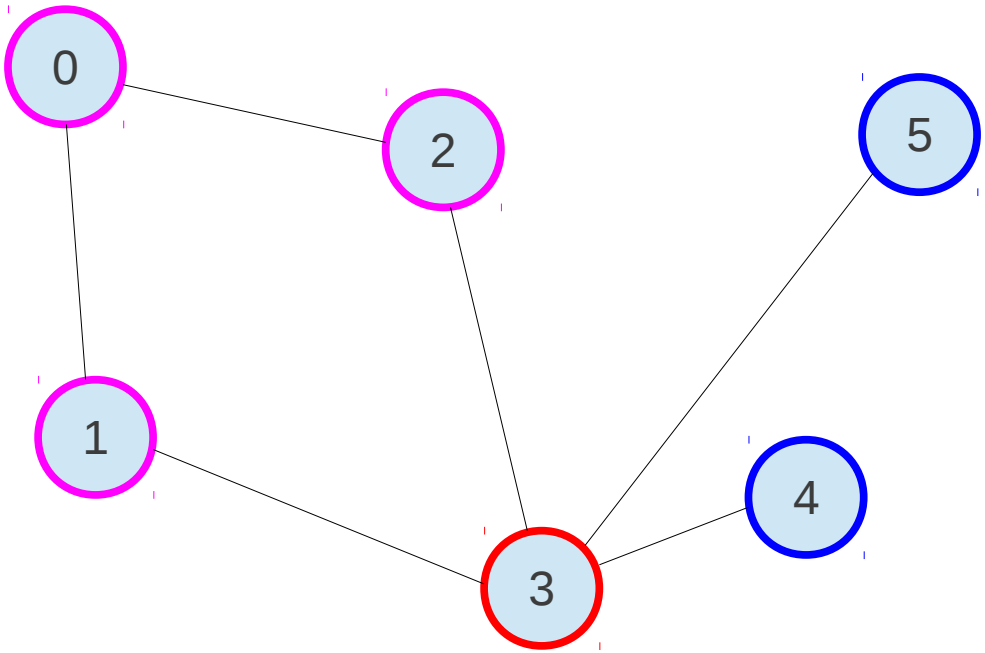
Graph Traversal Algorithms



Adjacency list
0: 1, 2
1: 1, 3
2: 0, 3
3: 1, 2, 4, 5
4: 3
5: 3

Queue
3

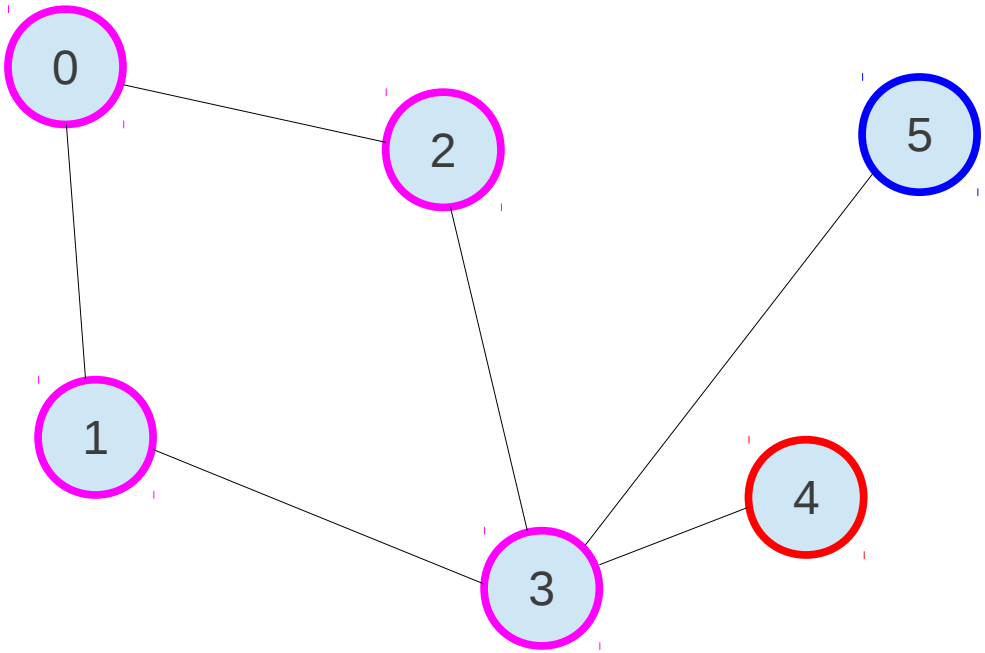
Graph Traversal Algorithms



Adjacency list
0: 1, 2
1: 1, 3
2: 0, 3
3: 1, 2, 4, 5
4: 3
5: 3

Queue
4
5

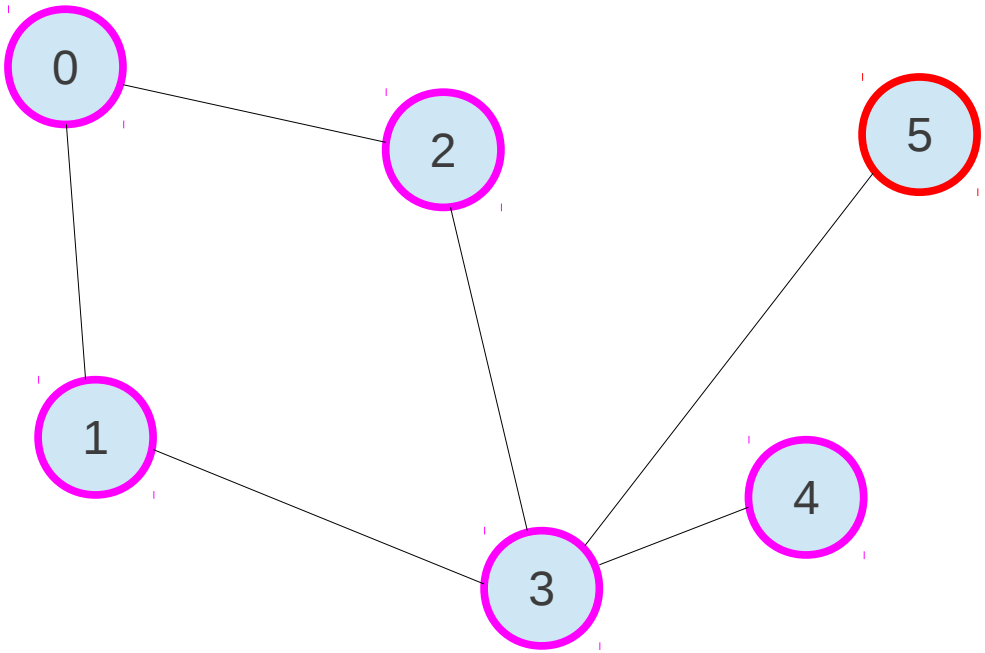
Graph Traversal Algorithms



Adjacency list
0: 1, 2
1: 1, 3
2: 0, 3
3: 1, 2, 4, 5
4: 3
5: 3

Queue
5

Graph Traversal Algorithms



Adjacency list
0: 1, 2
1: 1, 3
2: 0, 3
3: 1, 2, 4, 5
4: 3
5: 3

Queue

Graph Traversal Algorithms

```
ArrayList<ArrayList<Integer>> adjList; // prefilled with adjacents

Queue<Integer> q = new LinkedList<Integer>();
boolean visited[] = new boolean[N]; // keep track of visited nodes

q.add(0); visited[0] = true; // add to traversal queue and mark
int count = 1; // example: keep count of nodes traversed

while (!q.isEmpty()) {
    int node = q.poll();

    for (int i = 0; i < adjList.get(node).size(); i++) {
        int neighbor = adjList.get(node).get(i);
        if (!visited[neighbor]) { // do not visit nodes twice
            q.add(neighbor); // add to traversal queue
            visited[neighbor] = true; // mark as visited
            count++; // visited a new node! Keep count
        }
    }
}
```

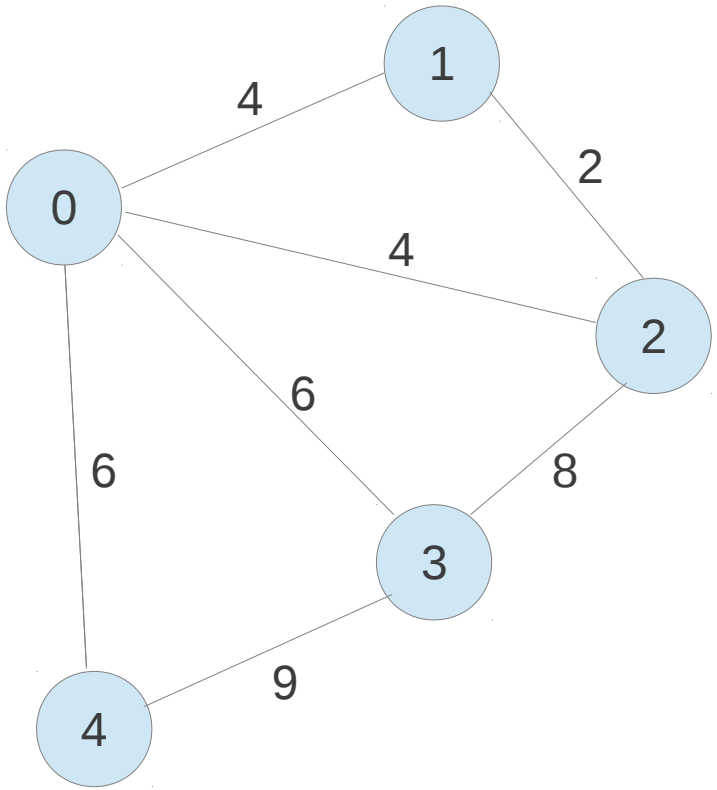
Exercise

- UVa 352: The Seasonal War
  - Given a graph, find how many blobs of connected 1s there are
    - Connections are on the major axes or diagonals
  - Major hint: Use DFS or BFS
    - Both work, pick the one more natural to you
  - 10 minutes

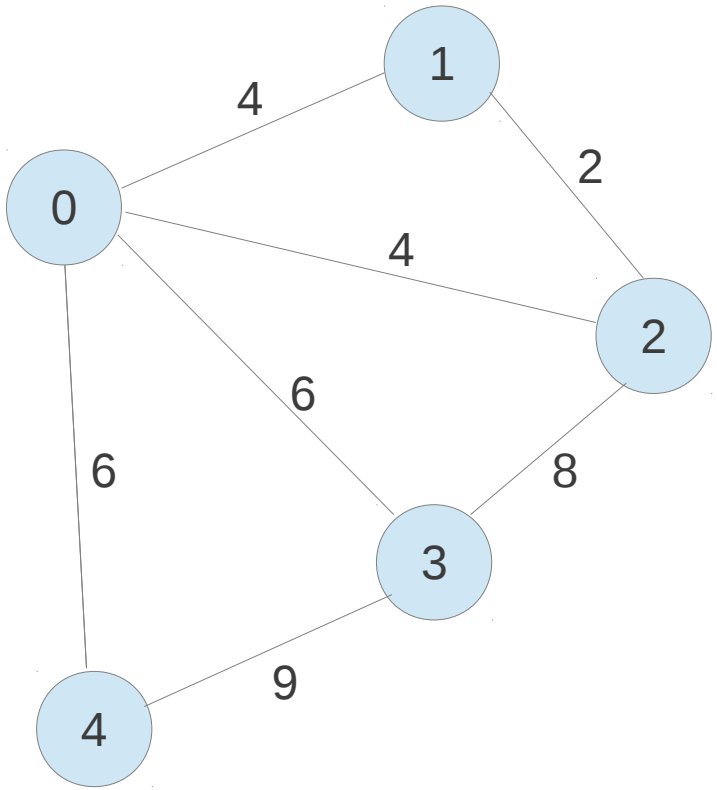
Minimum Spanning Trees

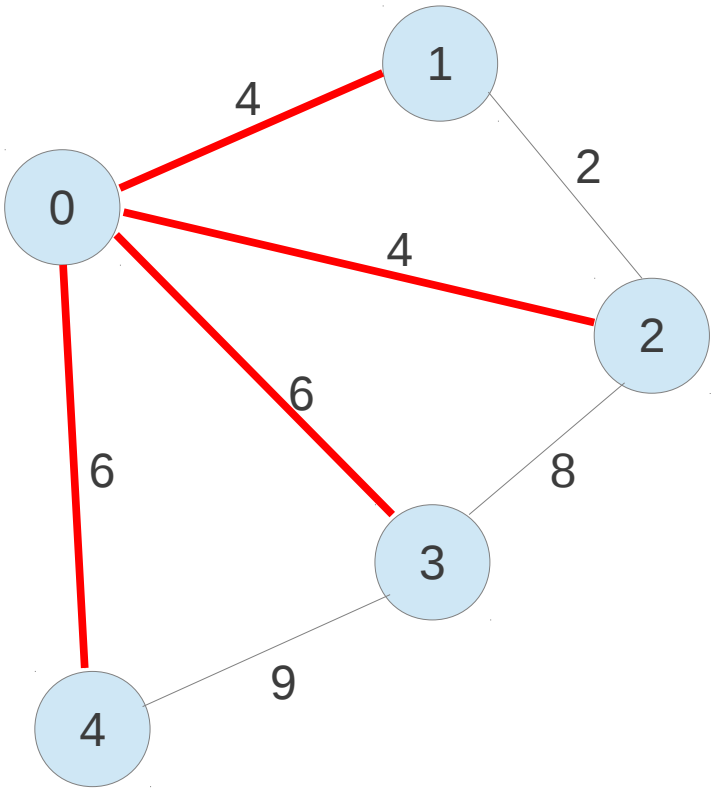
- Spanning tree
  - Given: a connected, undirected graph $G = (V, E)$
    - V is the set of vertices
    - E is the set of edges
  - A spanning tree is a set of edges that is a tree and “covers” all vertices V
    - There can be several trees
  - The spanning tree with the minimum cost (sum of edge weights) is called the **Minimum Spanning Tree**

Minimum Spanning Trees



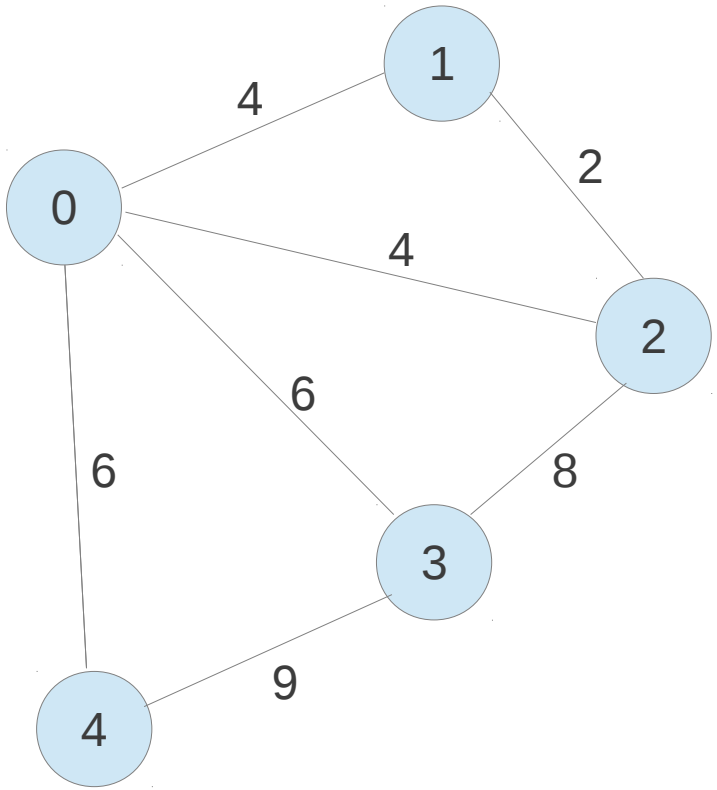
Minimum Spanning Trees

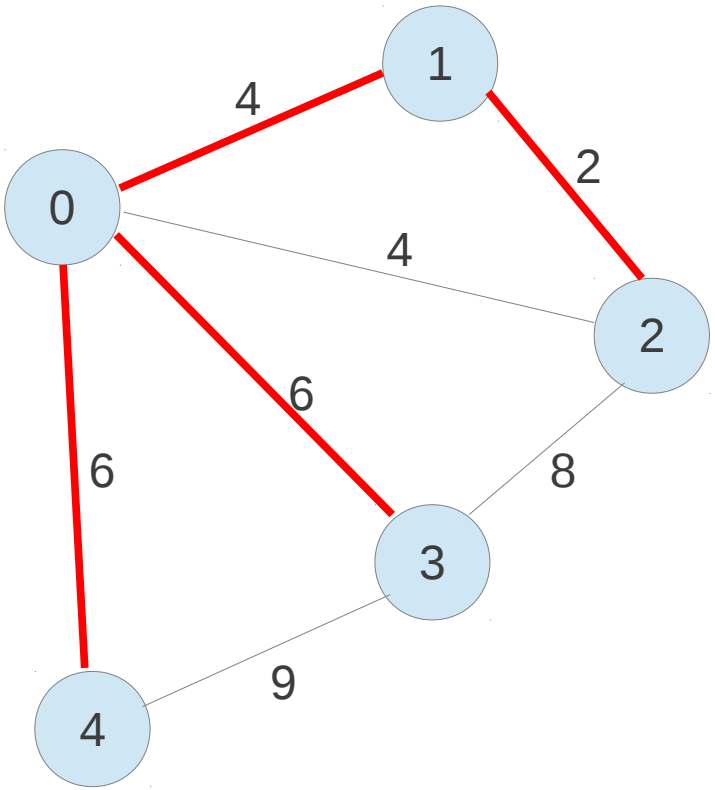




A spanning tree
Cost: 4 + 4 + 6 + 6 = 20

Minimum Spanning Trees





Minimum spanning tree
Cost: 4 + 2 + 6 + 6 = 18

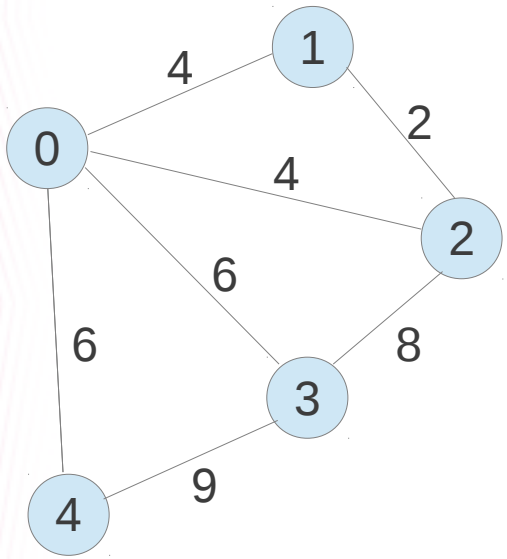
Minimum Spanning Trees

- Algorithms for finding the MST
  - Prim's algorithm
    - Not covered, see Wikipedia for a good overview
  - Kruskal's algorithm
    - Repeatedly finds edges with minimum costs that does not form a cycle
    - Greedy algorithm, provably correct
    - In a nutshell...

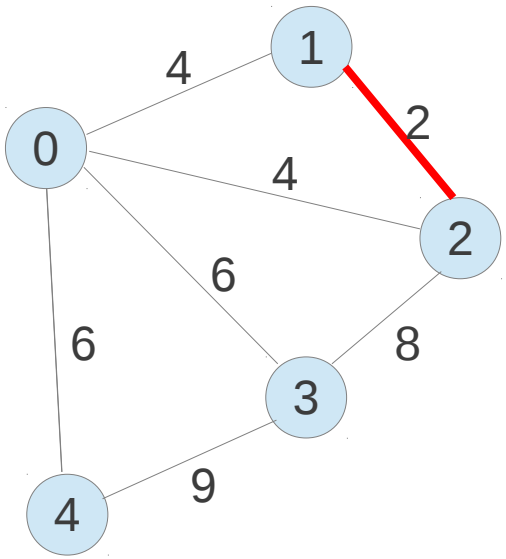
Minimum Spanning Trees

- Kruskal's algorithm pseudocode
  1) Sort edges by increasing weight
  2) While there are unprocessed edges left
     1) Pick an edge *e* **with minimum cost**
     2) If adding *e* to the MST **does not form a cycle,** add *e* to MST
  - Test for cycles using disjoint sets and union-find
  - Store all edges in an edge list, or possibly priority queue
  - Runtime
    - Sort: $O(|E| \log |E|)$; Processing: $O(|E|)$

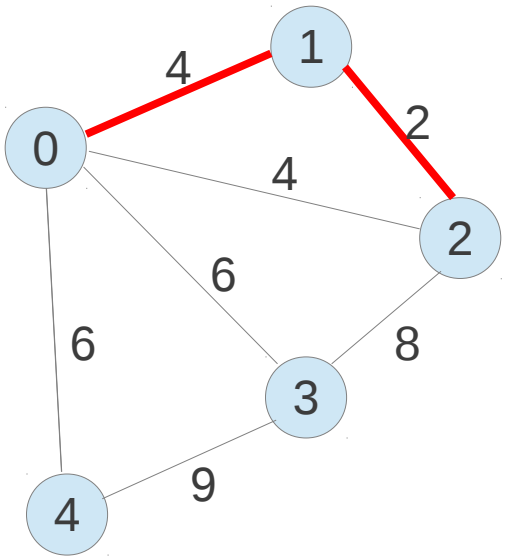
Minimum Spanning Trees



Pick smallest edge



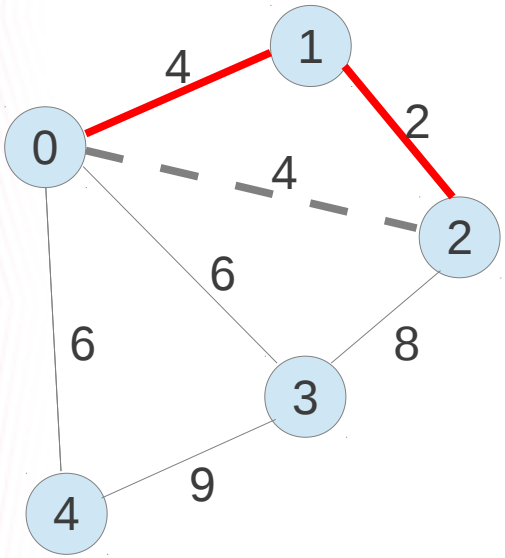
Pick smallest edge
No cycle



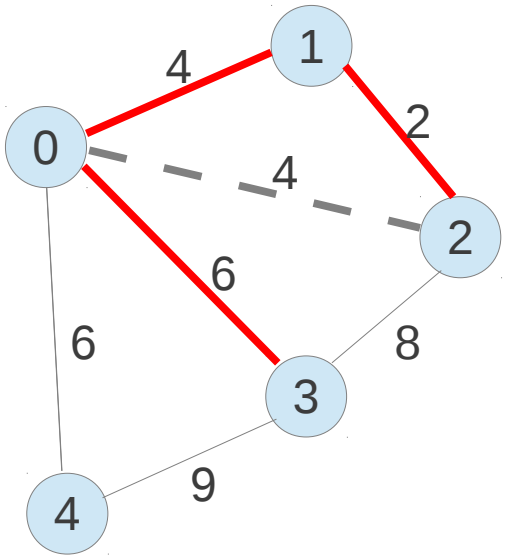
<u>Weighted adjacency list by (index, weight)</u>
0: (1, 4), (2, 4), (3, 6), (4, 6)
1: (0, 4), (2, 2)
2: (0, 4), (1, 2), (3, 8)
3: (0, 6), (2, 8), (4, 9)
4: (0, 6), (3, 9)

Minimum Spanning Trees

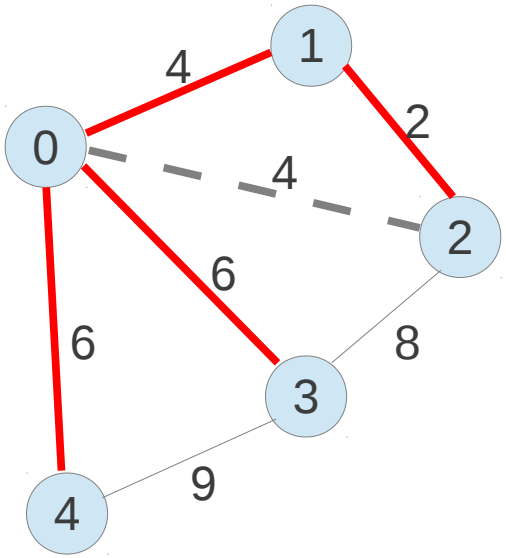
Pick smallest edge
Cycle formed, ignore



Pick smallest edge
No cycle



Pick smallest edge
No cycle

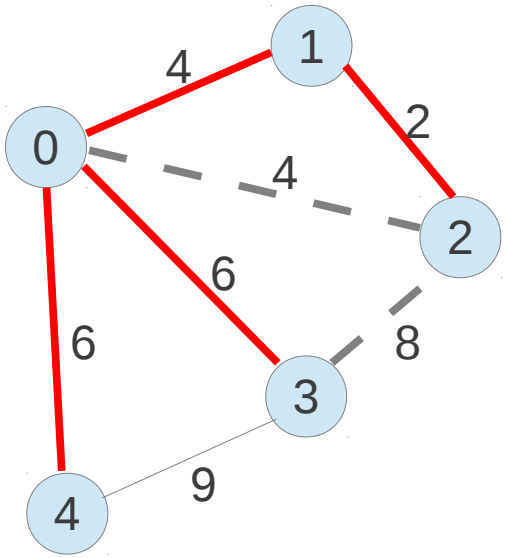


<u>Weighted adjacency list by (index, weight)</u>
0: (1, 4), (2, 4), (3, 6), (4, 6)
1: (0, 4), (2, 2)
2: (0, 4), (1, 2), (3, 8)
3: (0, 6), (2, 8), (4, 9)
4: (0, 6), (3, 9)

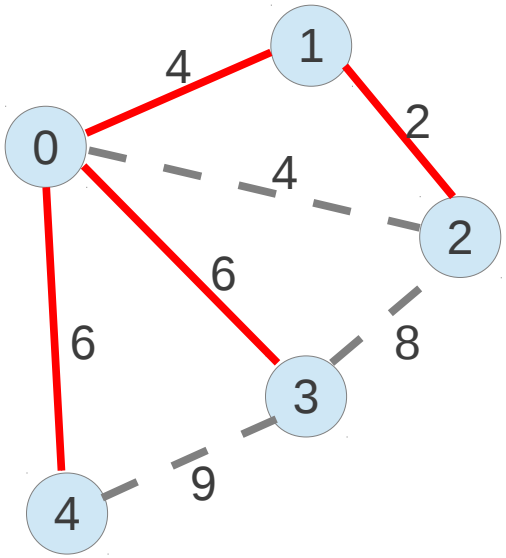
Minimum Spanning Trees

Algorithm not done!
The edge list hasn't
yet been exhausted

Pick smallest edge
Cycle formed, ignore



Pick smallest edge
Cycle formed, ignore



<u>Weighted adjacency list by (index, weight)</u>
0: (1, 4), (2, 4), (3, 6), (4, 6)
1: (0, 4), (2, 2)
2: (0, 4), (1, 2), (3, 8)
3: (0, 6), (2, 8), (4, 9)
4: (0, 6), (3, 9)

Minimum Spanning Tree

- More on Wikipedia
- Code will be on the class website

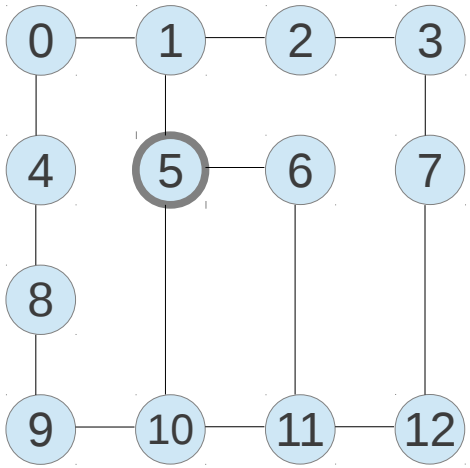
Single source shortest paths

- Classic problem in computer science
  - Given a node on a graph, find the shortest paths to all other nodes

Single source shortest paths

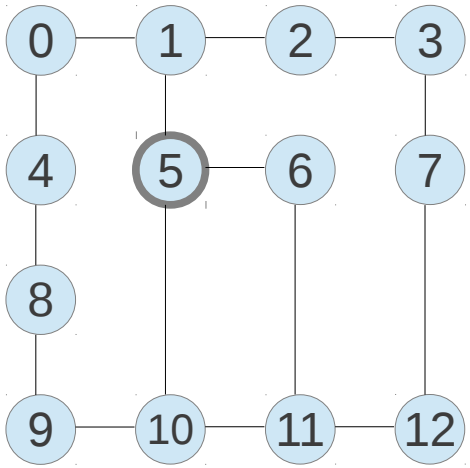
- For undirected, unweighted graphs:
  - Use BFS!
  - E.g., Uva 336 (A Node Too Far)
    - Given an undirected and unweighted graph G = (V, E) and a vertex v in V, find the number of nodes unreachable in n hops

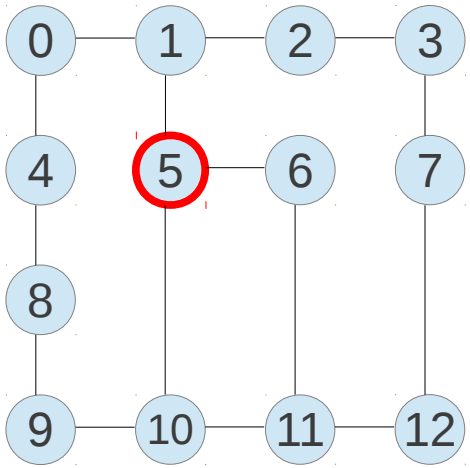
Single source shortest paths



From node 5, find # of nodes > 3 hops away

Single source shortest paths

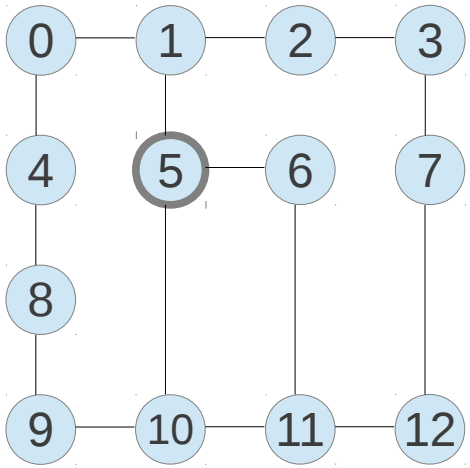


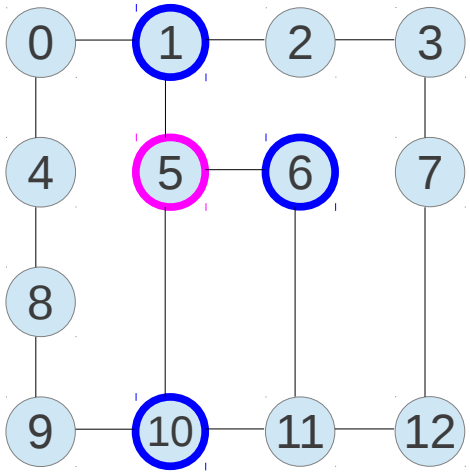


Queue	Distances
5	D[5] = 0

From node 5, find # of nodes > 3 hops away

Single source shortest paths

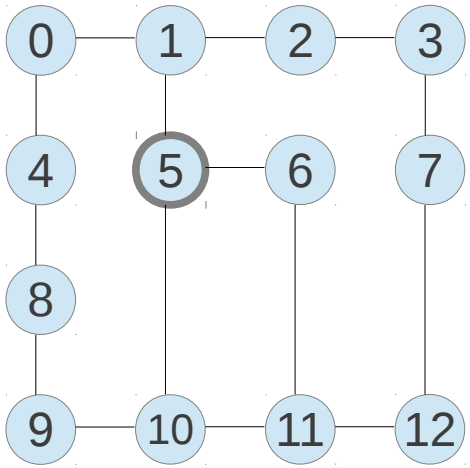


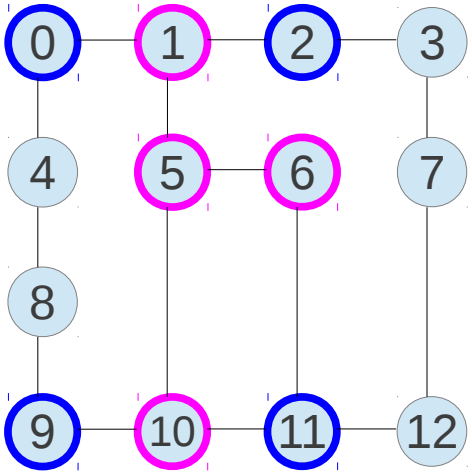


Queue	Distances
1	D[5] = 0
6	D[1] = D[5] + 1 = 1
10	D[6] = D[5] + 1 = 1
	D[10] = D[5] + 1 = 1

From node 5, find # of nodes > 3 hops away

Single source shortest paths

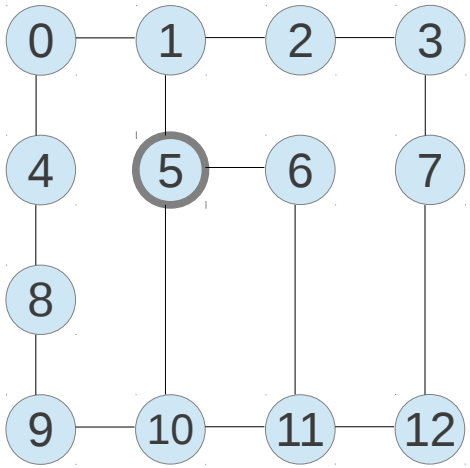


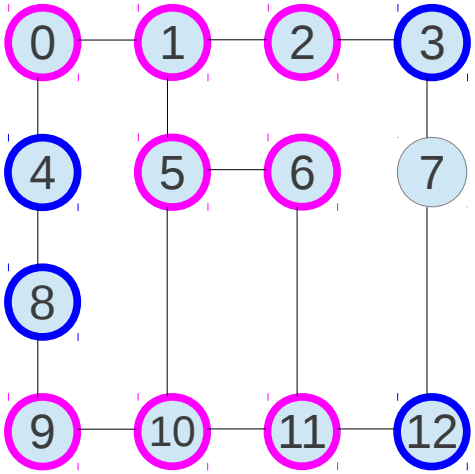


Queue	Distances
0	D[5] = 0
2	D[1] = 1
9	D[6] = 1
11	D[10] = 1
	D[0] = D[1] + 1 = 2
	D[2] = D[1] + 1 = 2
	D[9] = D[10] + 1 = 2
	D[11] = D[10] + 1 = 2

From node 5, find # of nodes > 3 hops away

Single source shortest paths

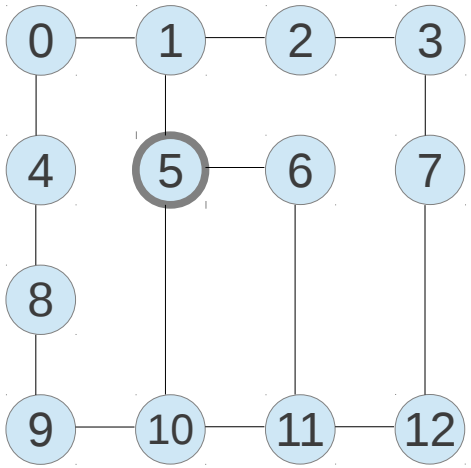


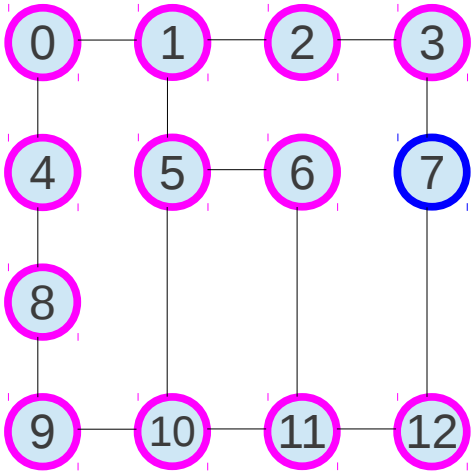


Queue	Distances
3	D[5] = 0
4	D[1] = 1
8	D[6] = 1
12	D[10] = 1
	D[0] = 2
	D[2] = 2
	D[9] = 2
	D[11] = 2
	D[3] = D[2] + 1 = 3
	D[4] = D[0] + 1 = 3
	D[8] = D[9] + 1 = 3
	D[12] = D[11] + 1 = 3

From node 5, find # of nodes > 3 hops away

Single source shortest paths



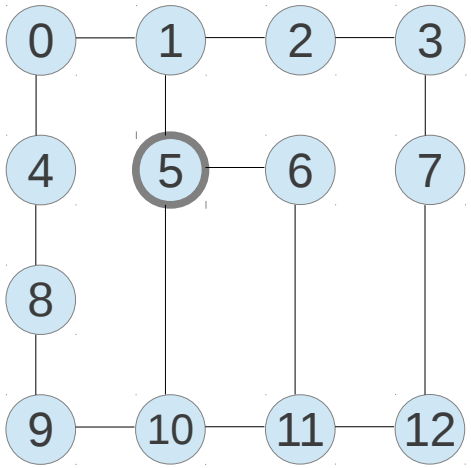


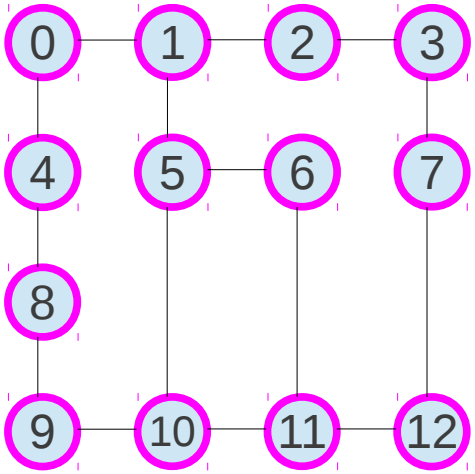
<u>Queue</u>
7

<u>Distances</u>
D[5] = 0
D[1] = 1
D[6] = 1
D[10] = 1
D[0] = 2
D[2] = 2
D[9] = 2
D[11] = 2
D[3] = 3
D[4] = 3
D[8] = 3
D[12] = 3
D[7] = D[3] + 1 = 4

From node 5, find # of nodes > 3 hops away

Single source shortest paths





Queue
0

Distances
D[5] = 0
D[1] = 1
D[6] = 1
D[10] = 1
D[0] = 2
D[2] = 2
D[9] = 2
D[11] = 2
D[3] = 3
D[4] = 3
D[8] = 3
D[12] = 3
D[7] = 4

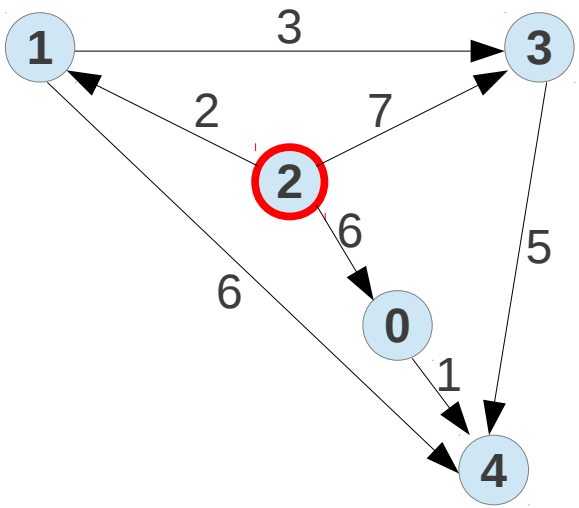
Answer: 1

From node 5, find # of nodes > 3 hops away

Single source shortest paths

- For directed, weighted graphs:
  - Use Dijkstra's! $O((|V| + |E|) \log |V|)$
  - This can be done using a priority queue
    - Works kind of like a greedy, modified BFS
  - Shown by example...

Single source shortest paths



Question: Shortest paths from 2 to all other nodes?

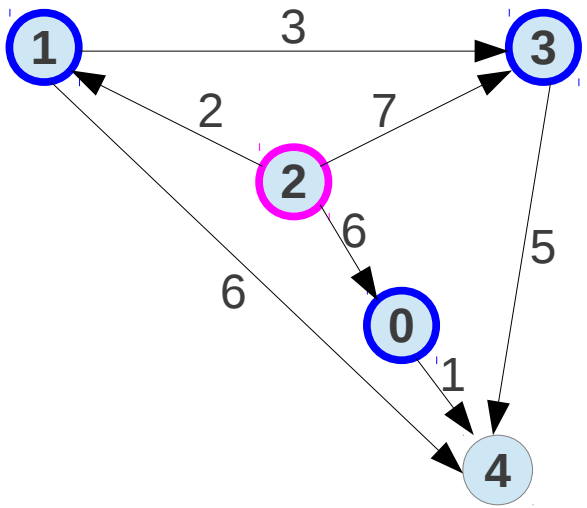
<u>Priority Queue (distance, node index)</u>
{ (0, 2) }

i	0	1	2	3	4
d[i]	INF	INF	0	INF	INF

Distance table

Start from node 2

Single source shortest paths



Question: Shortest paths from 2 to all other nodes?

Priority Queue (distance, node index)

~~{ (0, 2) }~~

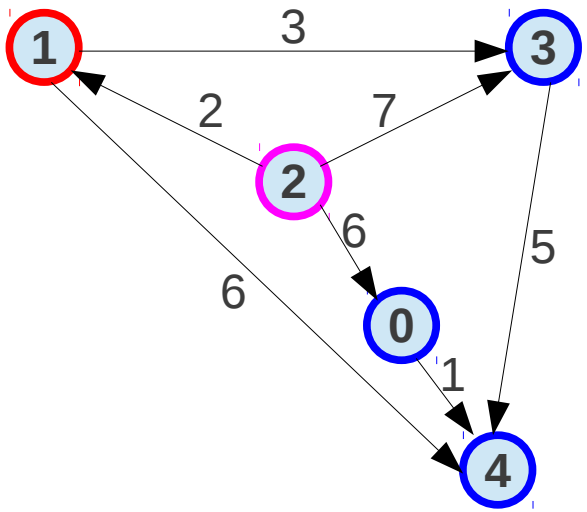
{ (2, 1), (6, 0), (7, 3) }

i	0	1	2	3	4
d[i]	6	2	0	7	INF

Distance table

Add all unvisited nodes from node 2 to the priority queue.
The PQ sorts the distances so the “next closest” node floats to the top.
Right now the closest node is 1, followed by 0, then 3.

Single source shortest paths



Question: Shortest paths from 2 to all other nodes?

Priority Queue (distance, node index)

~~{ (0, 2) }~~

{ ~~(2, 1)~~, (6, 0), (7, 3) }

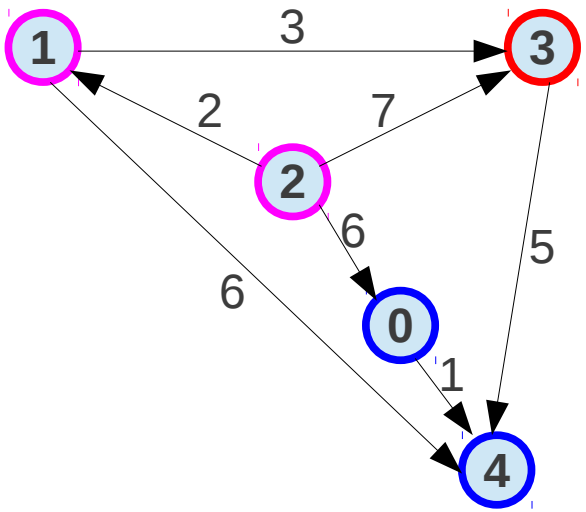
{ (5, **3**), (6, 0), (7, **3**), (8, 4) }

i	0	1	2	3	4
d[i]	6	2	0	5	8

Distance table

Poll from the PQ to get node 1.
Add all neighboring nodes to node 1 that haven't been polled yet.
BUT be sure to add all nodes that may already be in the queue with longer distances – there may be a shorter way to reach them

Single source shortest paths



Question: Shortest paths from 2 to all other nodes?

Priority Queue (distance, node index)

~~{ (0, 2) }~~

{ ~~(2, 1)~~, (6, 0), (7, 3) }

{ ~~(5, 3)~~, (6, 0), (7, 3), (8, 4) }

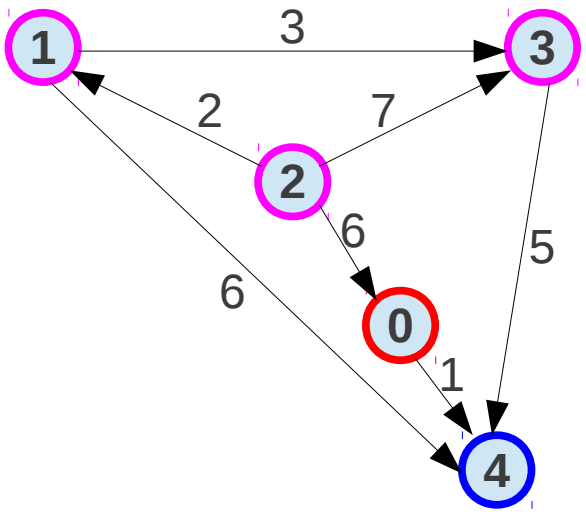
{ (6, 0), (7, 3), (8, 4) }

i	0	1	2	3	4
d[i]	6	2	0	5	8

Distance table

Poll from the PQ to get node 3.
Since we know there is a faster way to get node 4, don't bother adding node 4 to PQ

Single source shortest paths



Question: Shortest paths from 2 to all other nodes?

Priority Queue (distance, node index)

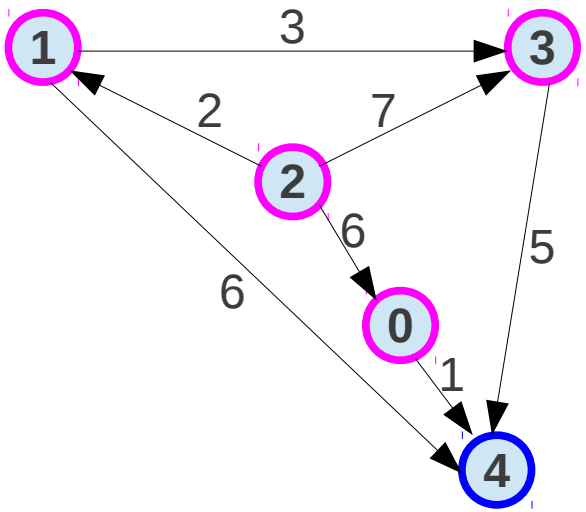
~~{ (0, 2) }~~
{ ~~(2, 1)~~, (6, 0), (7, 3) }
{ ~~(5, 3)~~, (6, 0), (7, 3), (8, 4) }
{ ~~(6, 0)~~, (7, 3), (8, 4) }
{ (7, 3), (7, 4), (8, 4) }

i	0	1	2	3	4
d[i]	6	2	0	5	7

Distance table

Poll from the PQ to get node 0.
Since we know there is a faster way to get node 4, don't bother adding node 4 to PQ

Single source shortest paths



Question: Shortest paths from 2 to all other nodes?

Priority Queue (distance, node index)

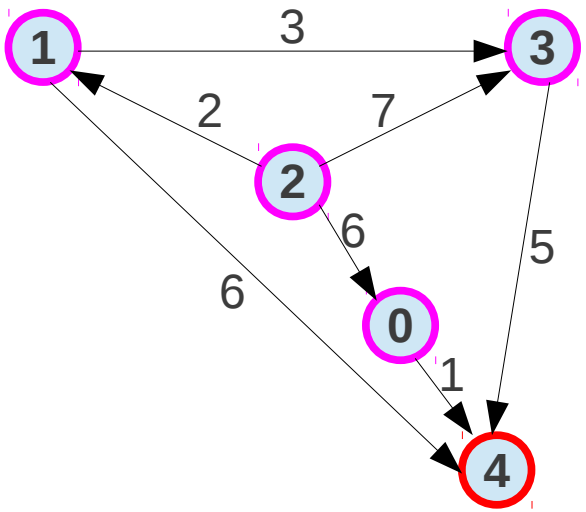
~~{ (0, 2) }~~
{ ~~(2, 1)~~, (6, 0), (7, 3) }
{ ~~(5, 3)~~, (6, 0), (7, 3), (8, 4) }
{ ~~(6, 0)~~, (7, 3), (8, 4) }
{ ~~(7, 3)~~, (7, 4), (8, 4) }
{ (7, 4), (8, 4) }

i	0	1	2	3	4
d[i]	6	2	0	5	7

Distance table

Now the (7, 3) state is ignored because it's been determined that 7 is a longer path than another existing path to node 3

Single source shortest paths



Question: Shortest paths from 2 to all other nodes?

Priority Queue (distance, node index)

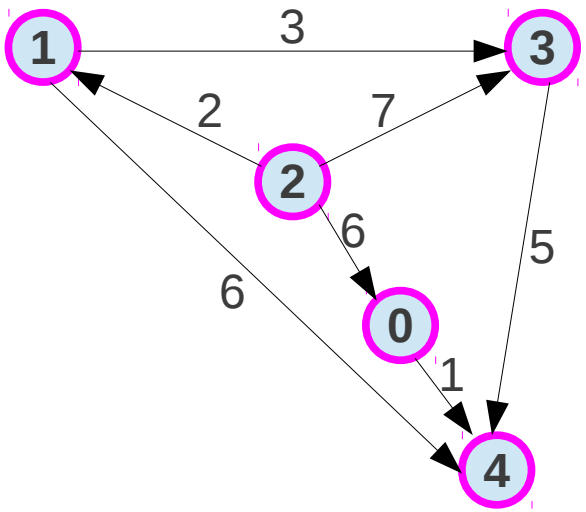
~~{ (0, 2) }~~
{ ~~(2, 1)~~, (6, 0), (7, 3) }
{ ~~(5, 3)~~, (6, 0), (7, 3), (8, 4) }
{ ~~(6, 0)~~, (7, 3), (8, 4) }
{ ~~(7, 3)~~, (7, 4), (8, 4) }
{ ~~(7, 4)~~, (8, 4) }
{ (8, 4) }

i	0	1	2	3	4
d[i]	6	2	0	5	7

Distance table

Nowhere to go, so nothing is added to the PQ

Single source shortest paths



Question: Shortest paths from 2 to all other nodes?

Priority Queue (distance, node index)

~~{ (0, 2) }~~
{ ~~(2, 1)~~, (6, 0), (7, 3) }
{ ~~(5, 3)~~, (6, 0), (7, 3), (8, 4) }
{ ~~(6, 0)~~, (7, 3), (8, 4) }
{ ~~(7, 3)~~, (7, 4), (8, 4) }
{ ~~(7, 4)~~, (8, 4) }
{ ~~(8, 4)~~ }
{ }

i	0	1	2	3	4
d[i]	6	2	0	5	7

Distance table

State (8, 4) is ignored because 8 > 7

Dijkstra's in code

```
LinkedHashMap<Integer, LinkedHashMap<Integer, Integer>> adj;
// State is a pair (dist, index)
PriorityQueue<State> pq = new PriorityQueue<State>();
int dist[] = new int[V]; Arrays.fill(dist, 1 << 20); // INF
pq.add(new State(2, 0)); // Initial state

while (!pq.isEmpty()) {
    State s = pq.poll();
    if (s.dist == dist[s.index]) { // true if has not been updated
        LinkedHashMap<Integer, Integer> nbors = adj.get(s.index);
        for (Map.Entry<Integer, Integer> e : nbors.getEntrySet()) {
            int nbor = e.getKey();
            int nborDist = e.getValue();

            if (nborDist + dist[s.index] < dist[nbor]) {
                // have found a closer path
                dist[nbor] = nborDist + dist[s.index];
                pq.add(new State(nbor, nborDist + dist[s.index]));
            }
        }
    }
}
```

Single source shortest paths

- For directed, weighted graphs:
  - With negative cycles? Need to detect those, use Bellman Ford algorithm
    - Read about this in the text!

All pairs shortest paths

- What happens if you want to find the shortest distance between **all** pairs of nodes?
  - On a weighted, connected graph, use Floyd Warshall algorithm
  - Implement in ~4 lines of code
  - O(V^3) instead of N Dijkstra's algorithm, which would be O(V^3 log V)
  - Explained in detail in the book

Floyd Warshall in code

```
// inside int main()
// precondition: m[i][j] contains the weight of edge (i, j)
// or INF (1B) if there is no such edge
// (m is an adjacency matrix)

for (int k = 0; k < V; k++)
    for (int i = 0; i < V; i++)
        for (int j = 0; j < V; j++)
            m[i][j] = min(m[i][j], m[i][k] + m[k][j]);

// common error: remember that loop order is k->i->j
```

Graph algorithms

- Now you've seen the bread and butter of graph algorithms
  - There are many more problems associated with graphs
    - Find the width of a graph, find strongly connected components
  - There are special kinds of graphs and smarter algorithms for them
    - Trees, directed acyclic graphs (DAGs), bipartite graphs, eulerian graphs

Practice

For next class

- Readings:
  - Sections 4.1-4.5
    - Mostly what we went over in class, plus more
  - Section 4.7
    - Review this, special cases of graphs and algorithms on them
- Exercises:
  - Homework on the website