Applied Algorithms

Lecture 2: Meet in the Middle

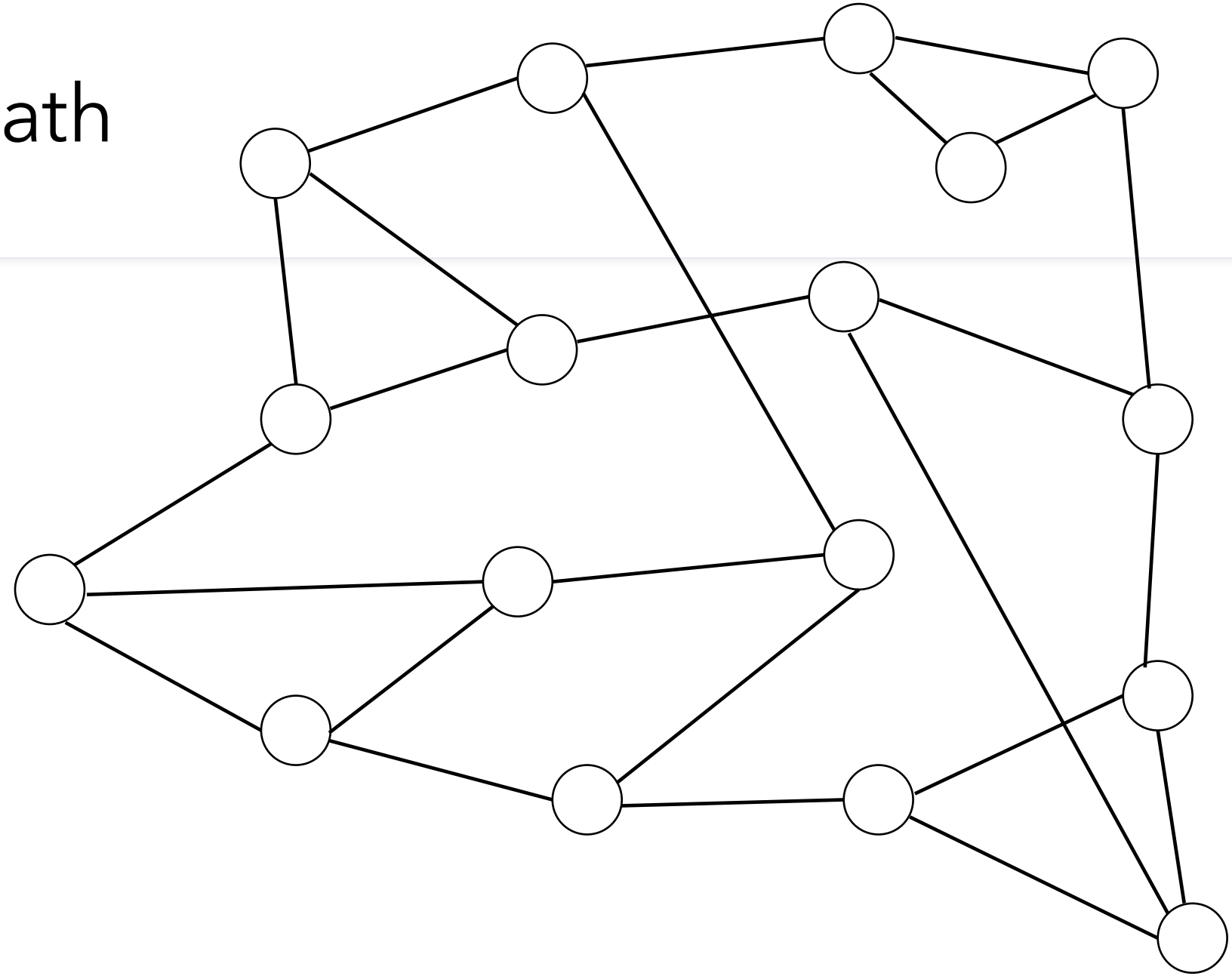
(and more C)

Admin

- Google form due midnight tonight (please fill!)
- Email me if you're still not registered
- Names during questions
- Assignment 1 will be posted soon; we'll go over instructions on Thursday

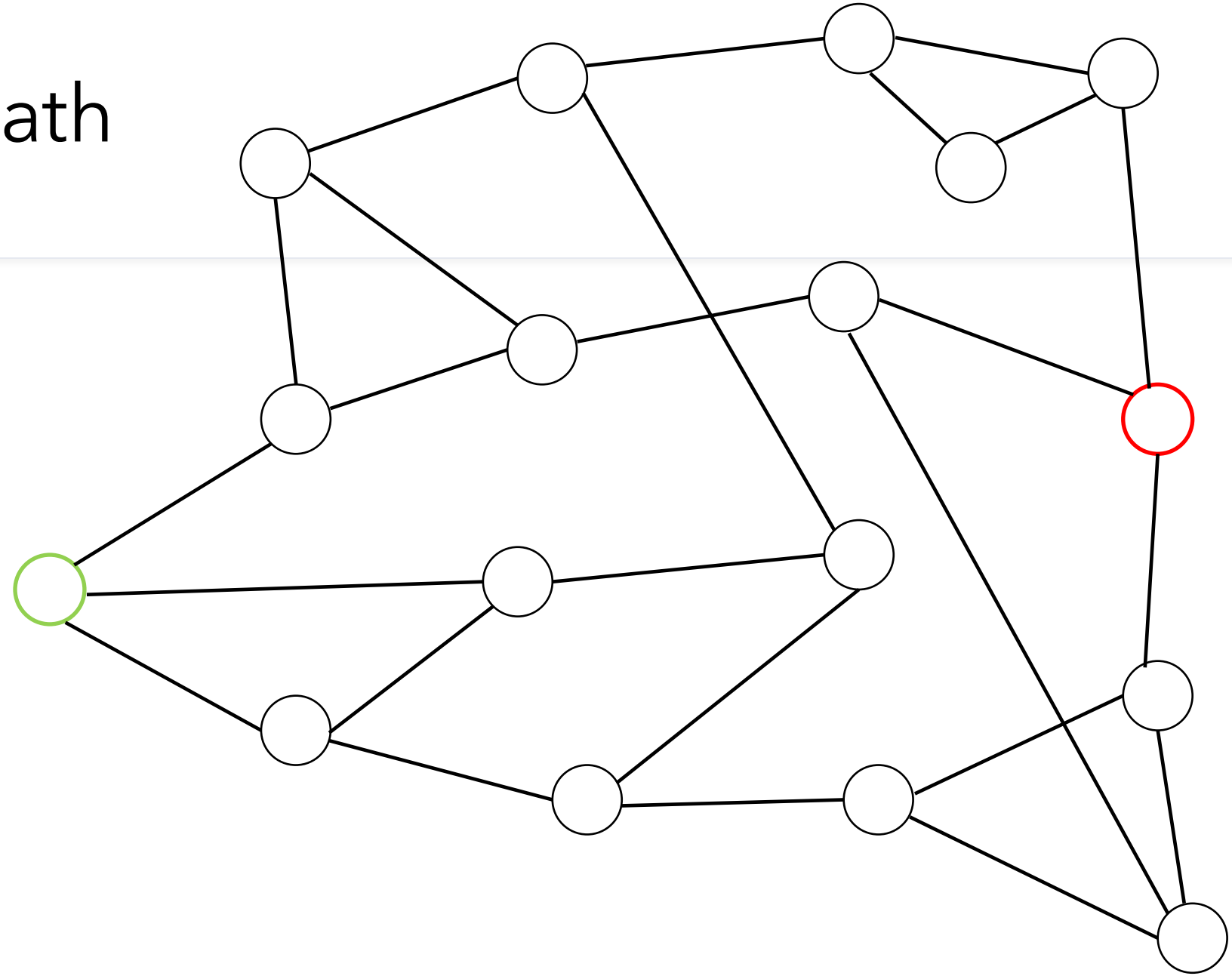
Meet in the middle

Shortest path



Shortest path

- How quickly can we find the shortest path in this graph?
- What algorithm should we use?



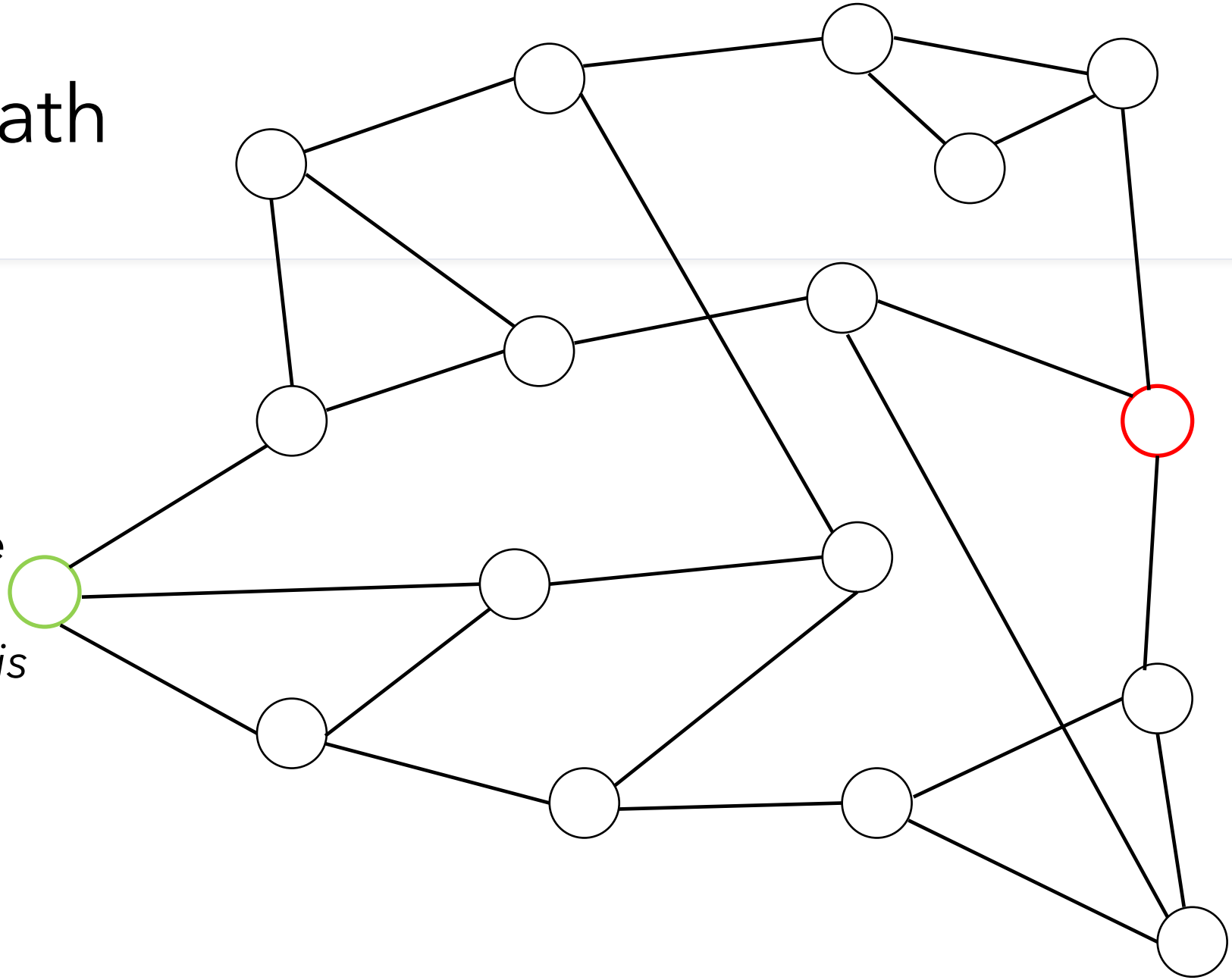
Shortest path

n nodes, m edges

Dijkstra's alg:

- O(m + n log n) time

Is there structure in *this* graph that lets us improve this?



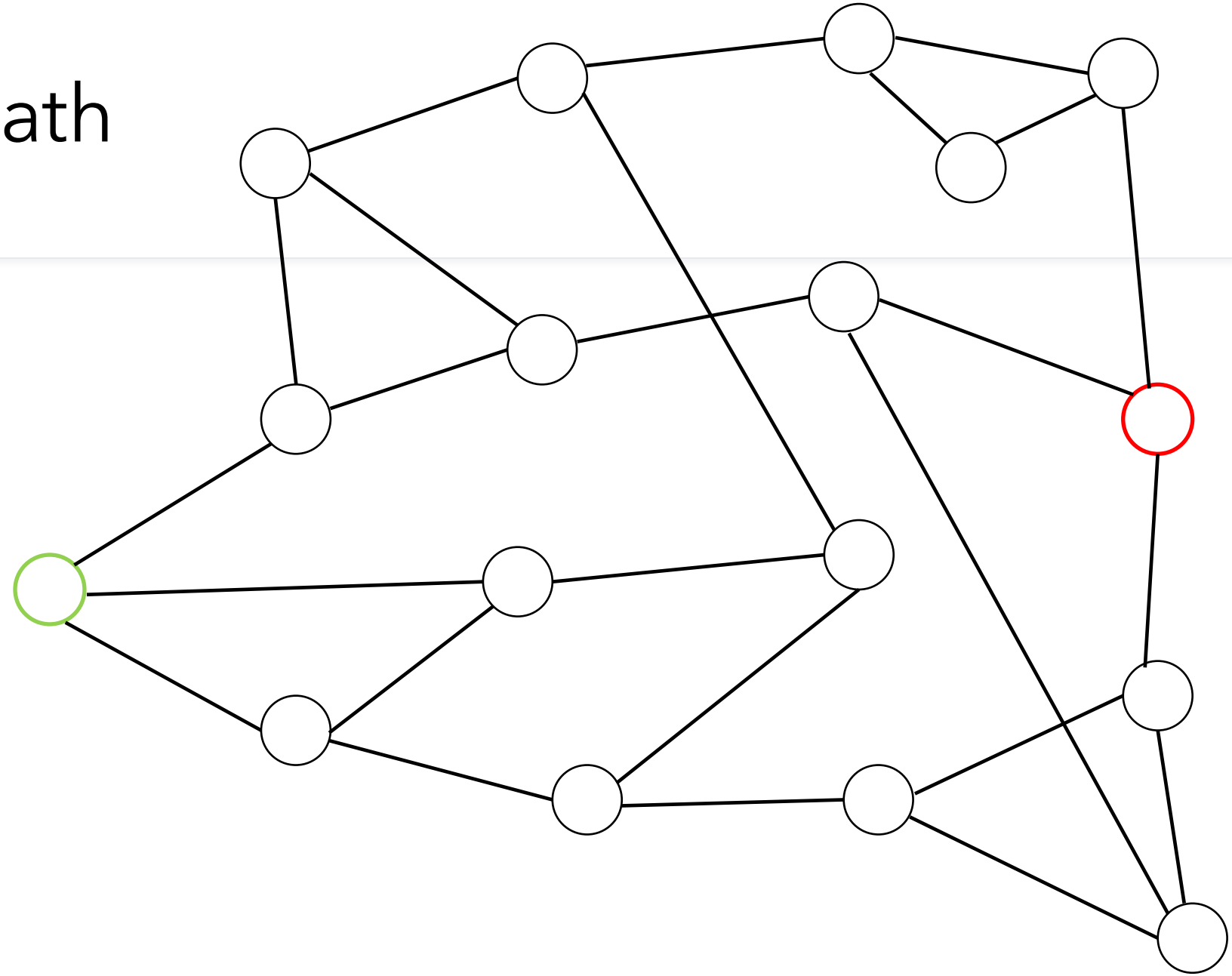
Shortest path

Unweighted!

BFS is sufficient

Time?

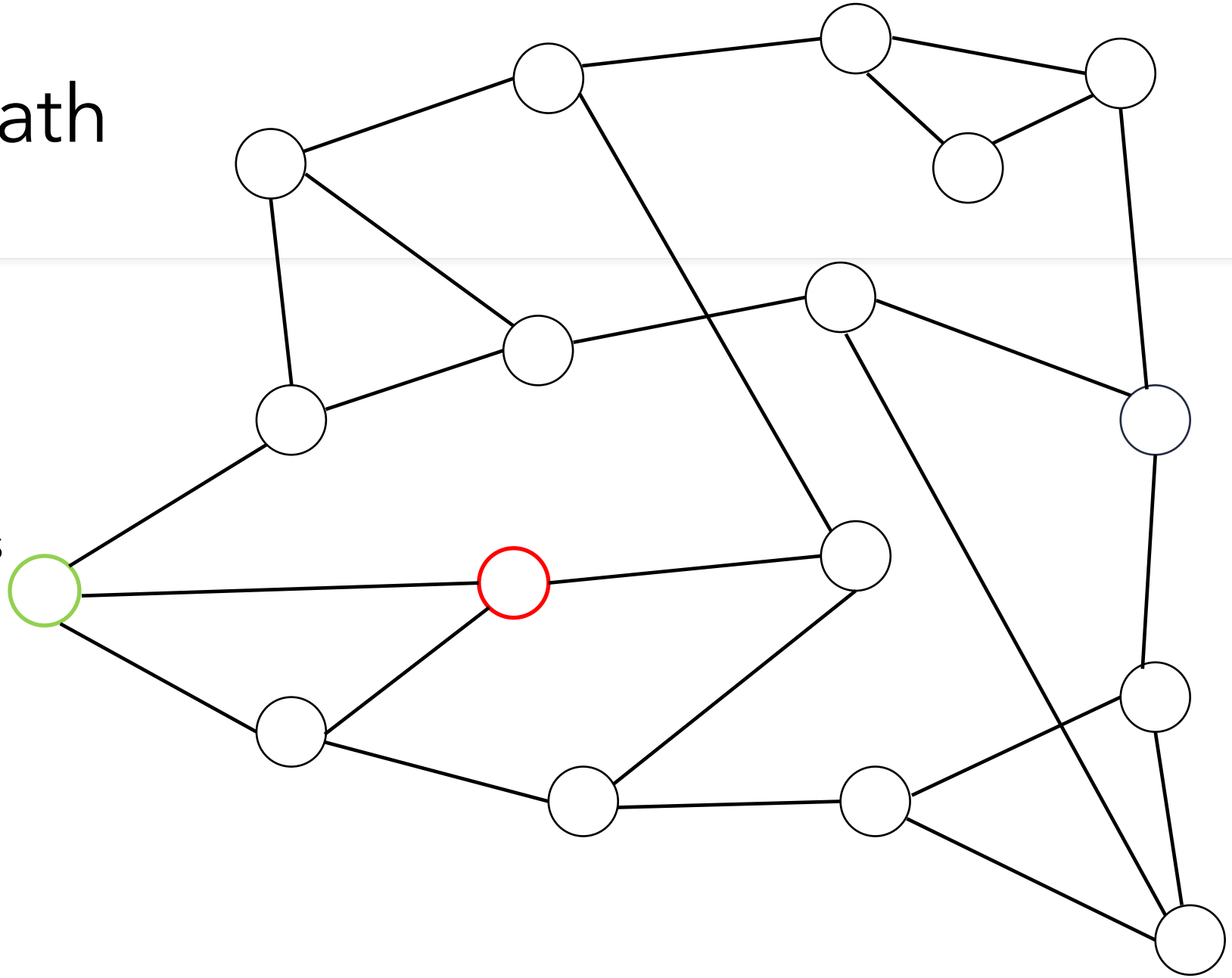
- O(m + n)



Shortest path

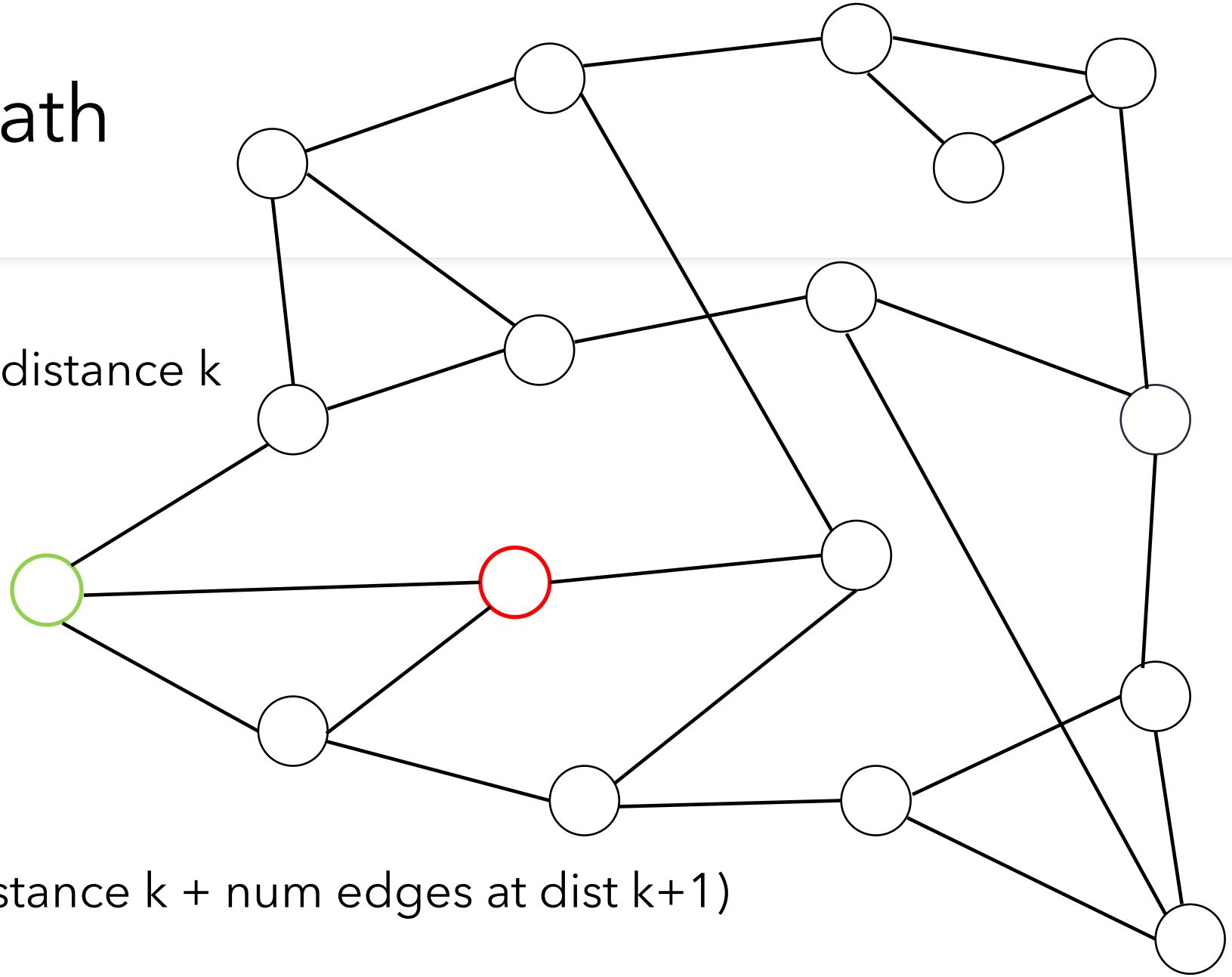
How about now?

I claim that this case is easier. Why?



Shortest path

Let's say nodes are at distance k

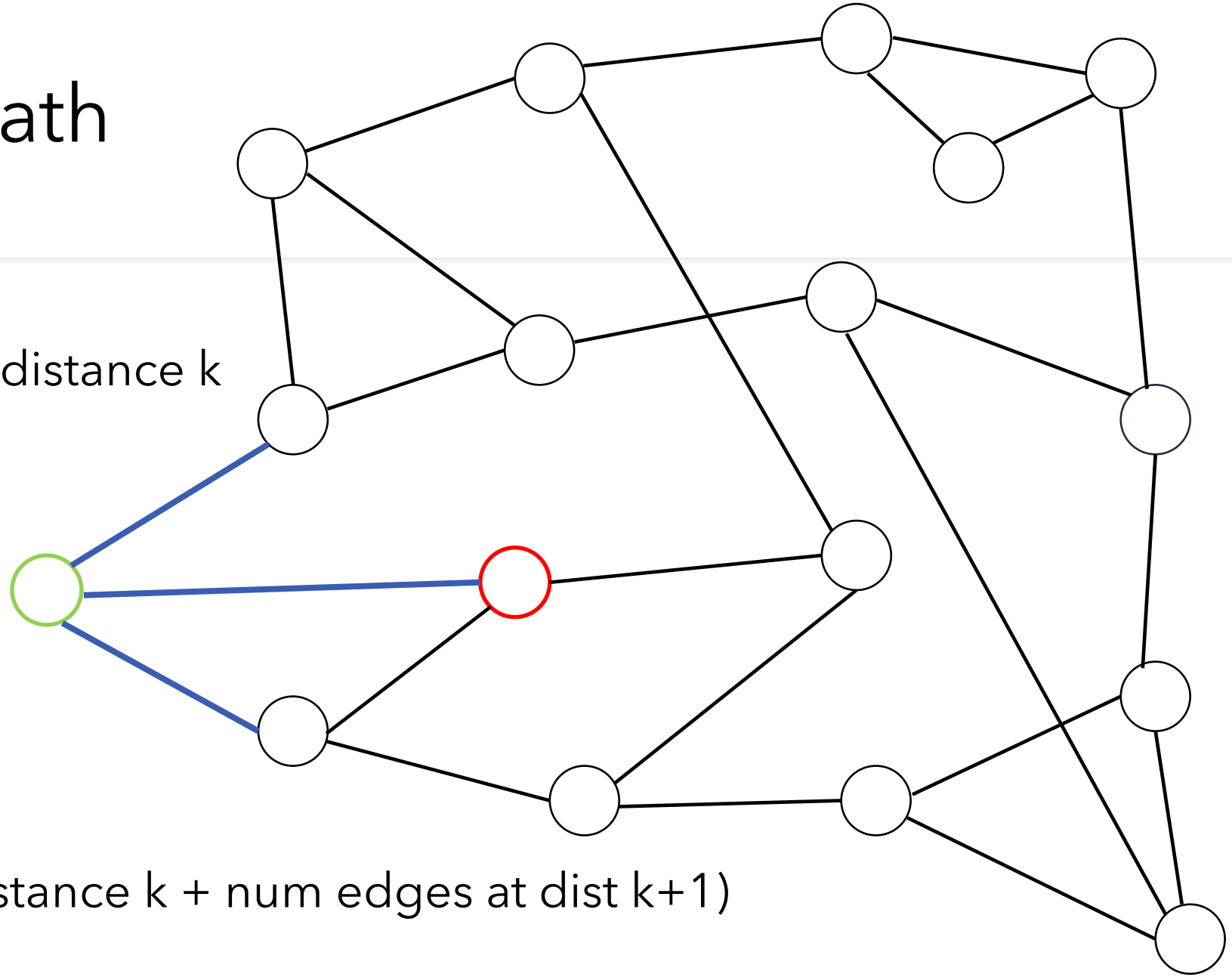


- Time for BFS?
  - (High level)

- O(num nodes at distance k + num edges at dist k+1)

Shortest path

Let's say nodes are at distance k

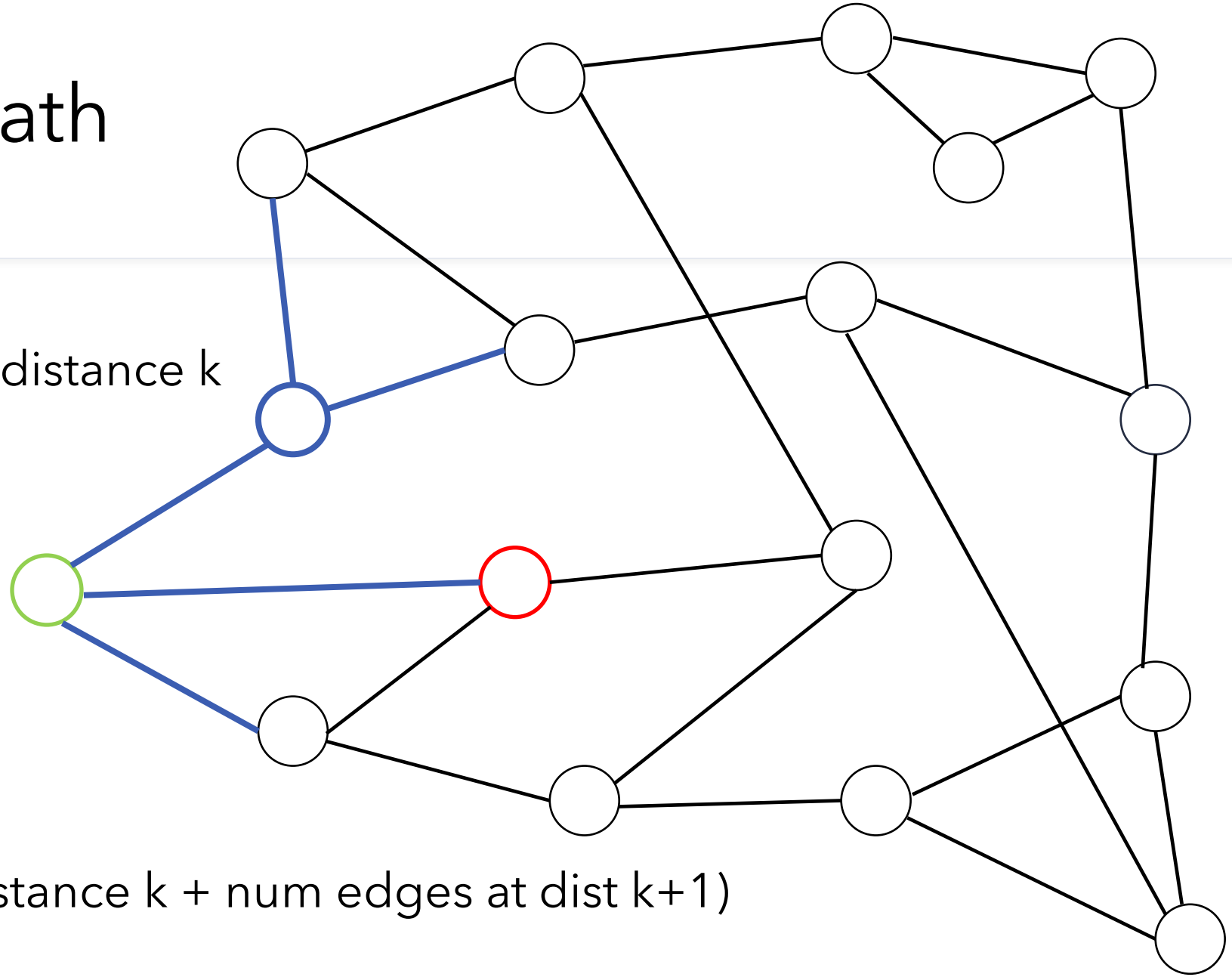


- Time for BFS?
  - (High level)

- O(num nodes at distance k + num edges at dist k+1)

Shortest path

Let's say nodes are at distance k



- Time for BFS?
  - (High level)

- O(num nodes at distance k + num edges at dist k+1)

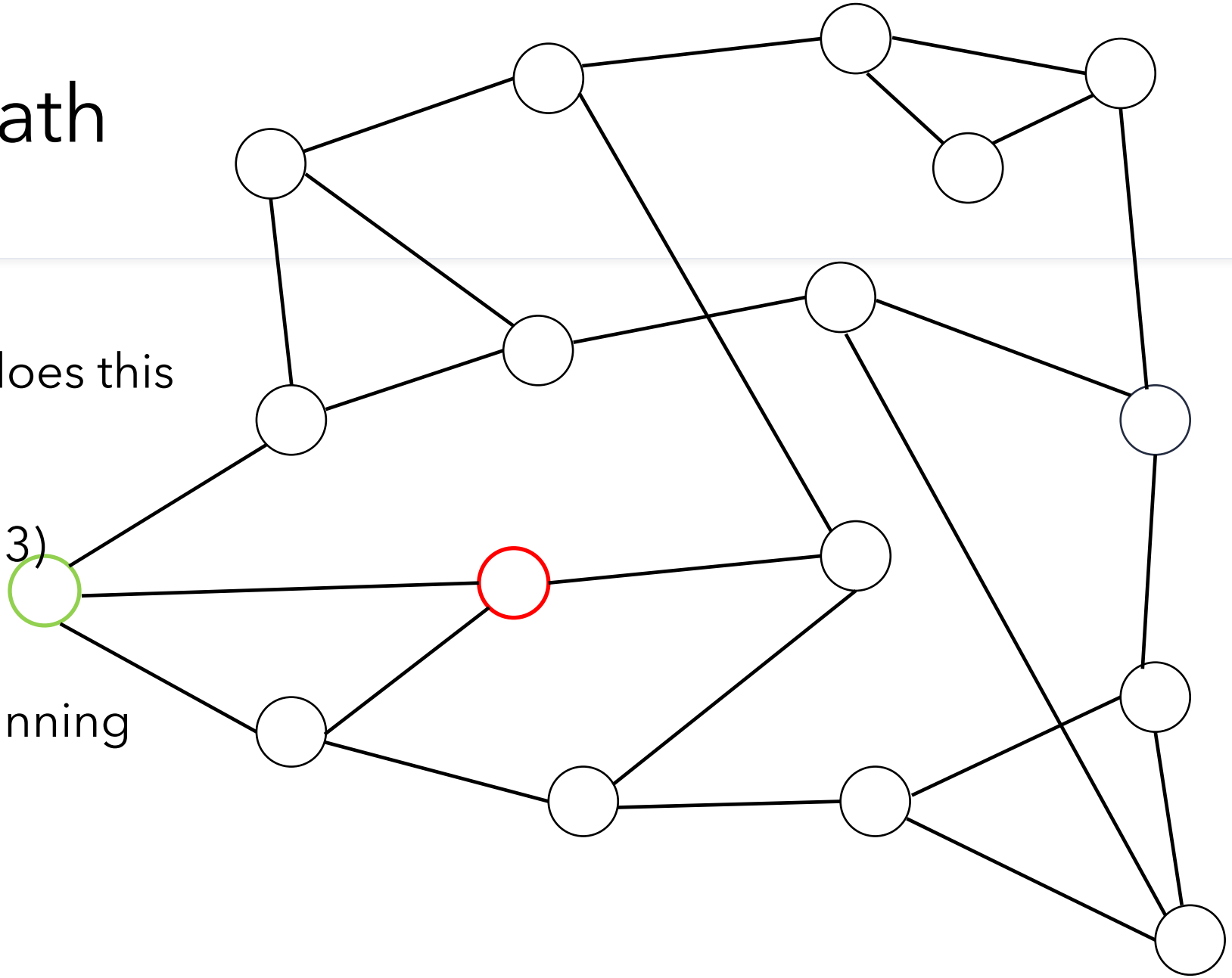
Shortest path

What other structure does this graph have?

- 3-regular!
- (All vertices degree 3)

Now can we say the running time in terms of k?

- $O(2^k)$



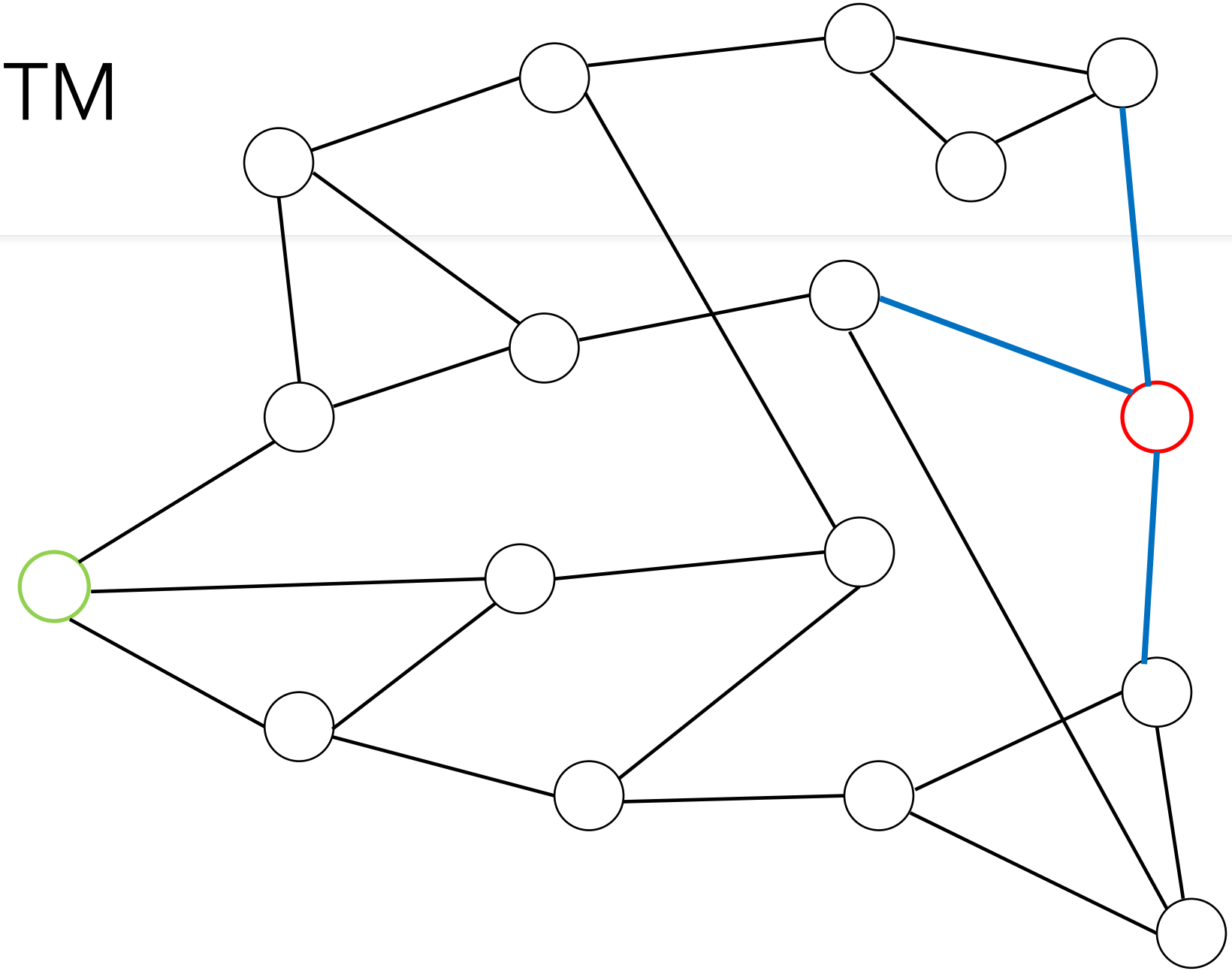
Any ideas for how to improve this?

- What is the topic for today again?

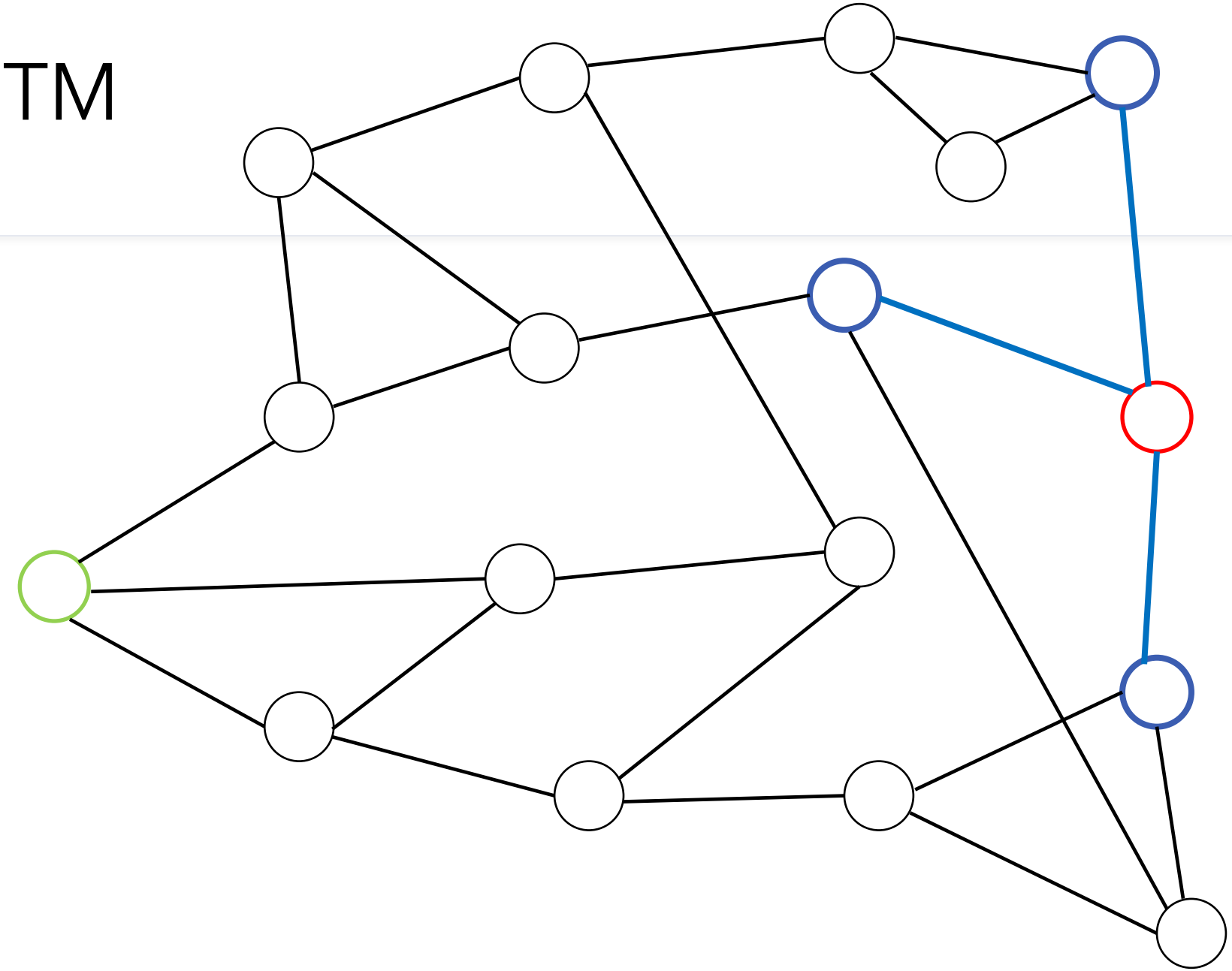
Meet in the middle

- Let's say we know k ahead of time, and all vertices are unvisited
- Go backwards k/2 steps from the target node. Mark resulting vertices as visited
  - (How do we do this? How much time does it take?)
- Go forwards k/2 steps from the start node. If you find a marked vertex, you have your path

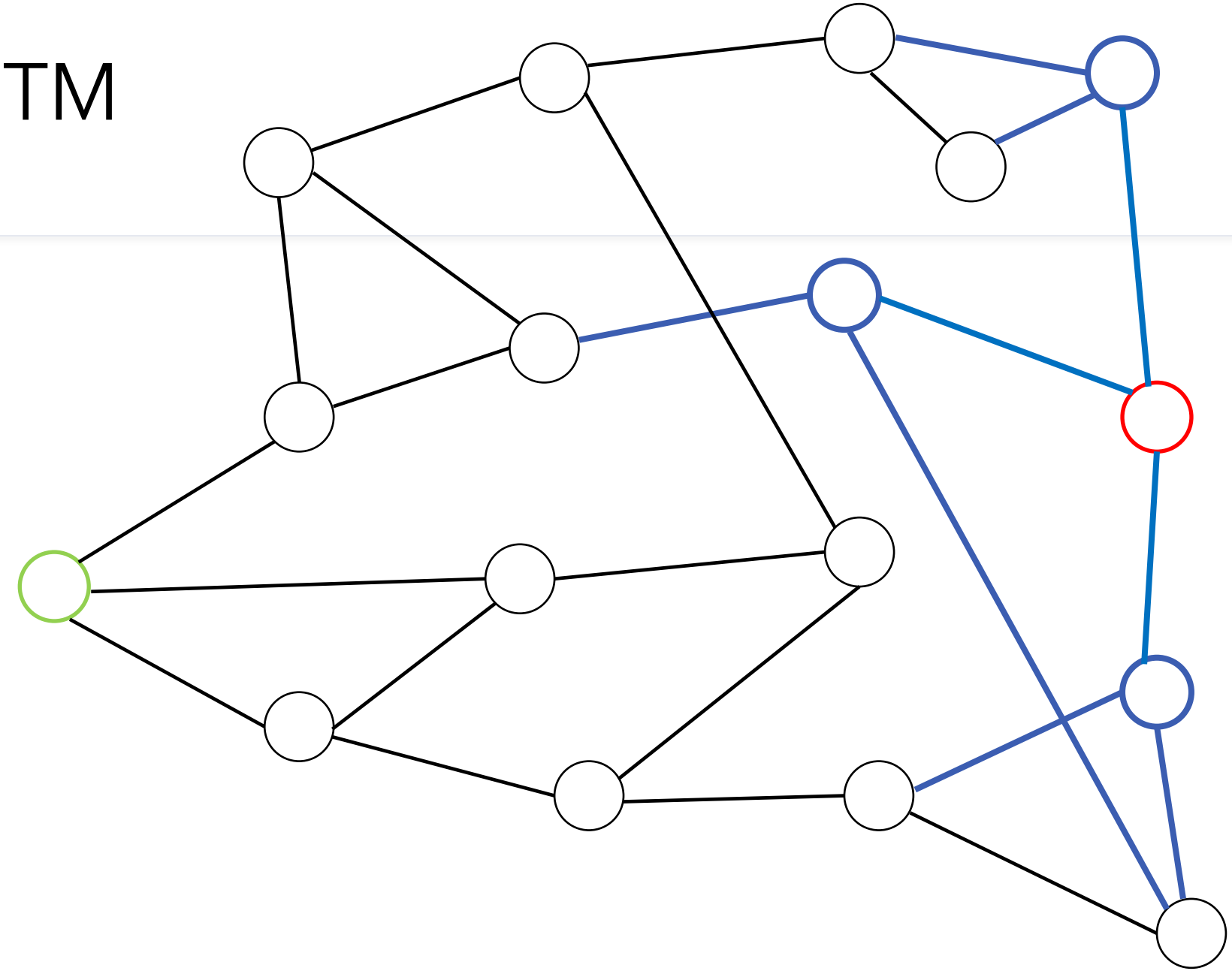
MITM



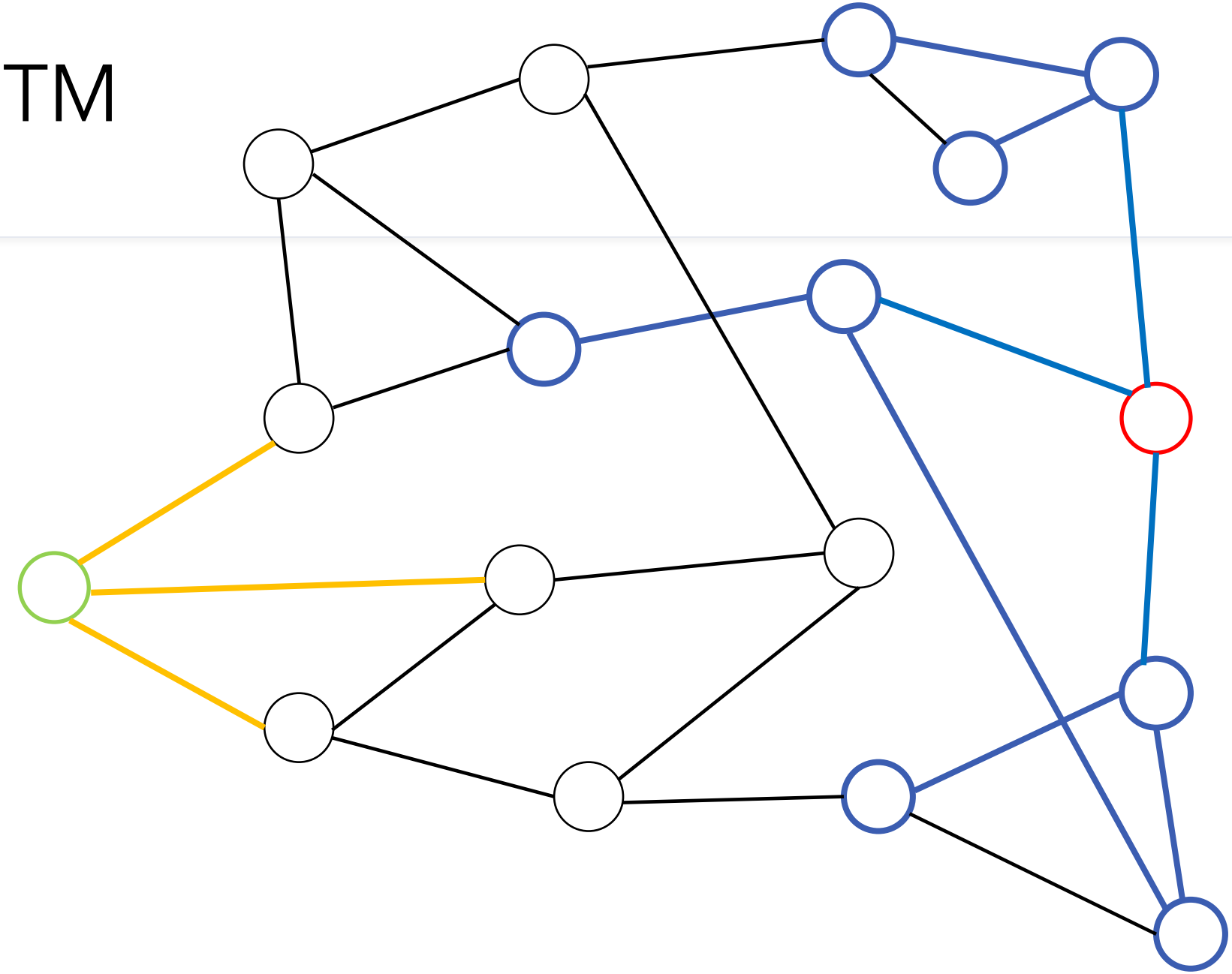
MITM



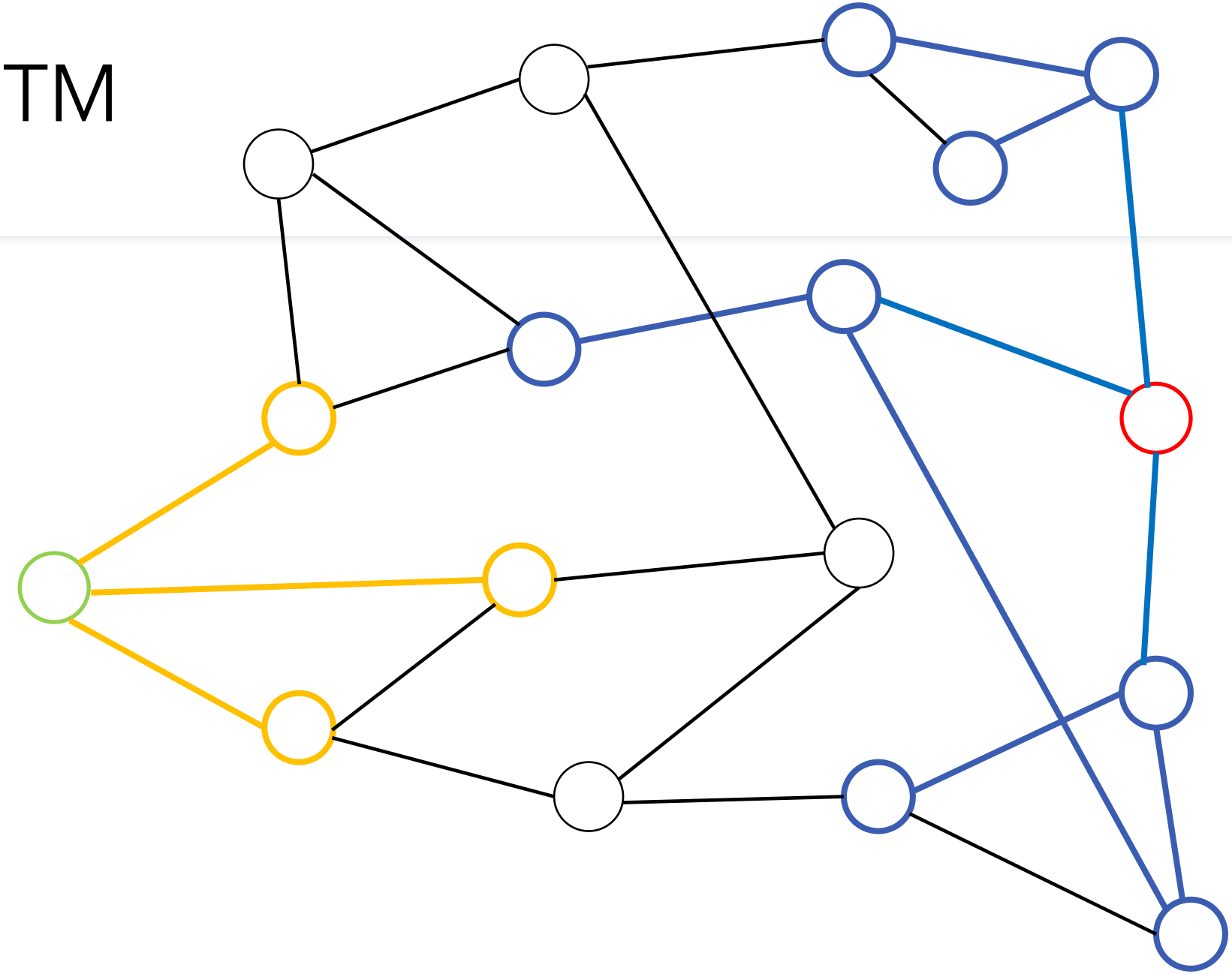
MITM



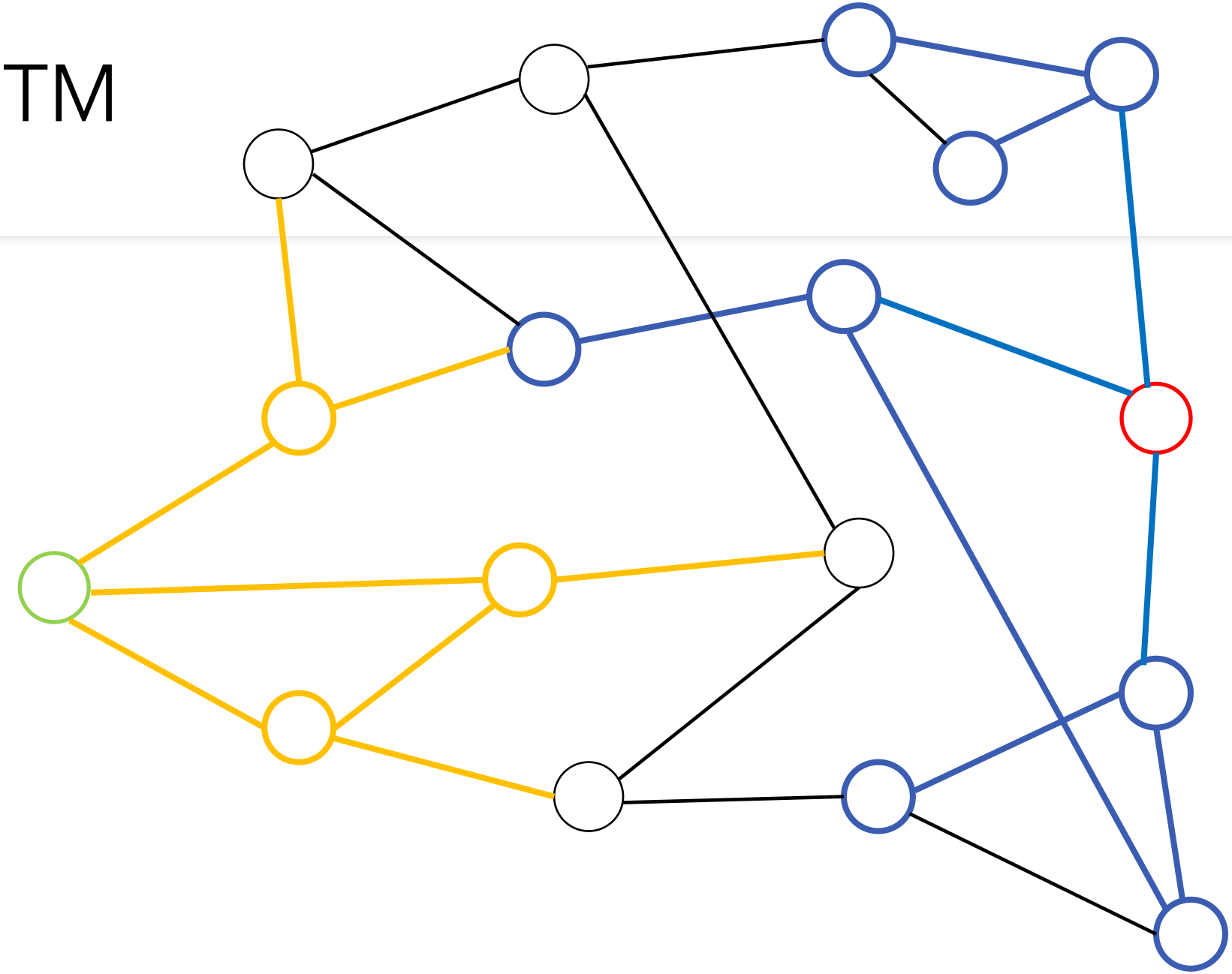
MITM



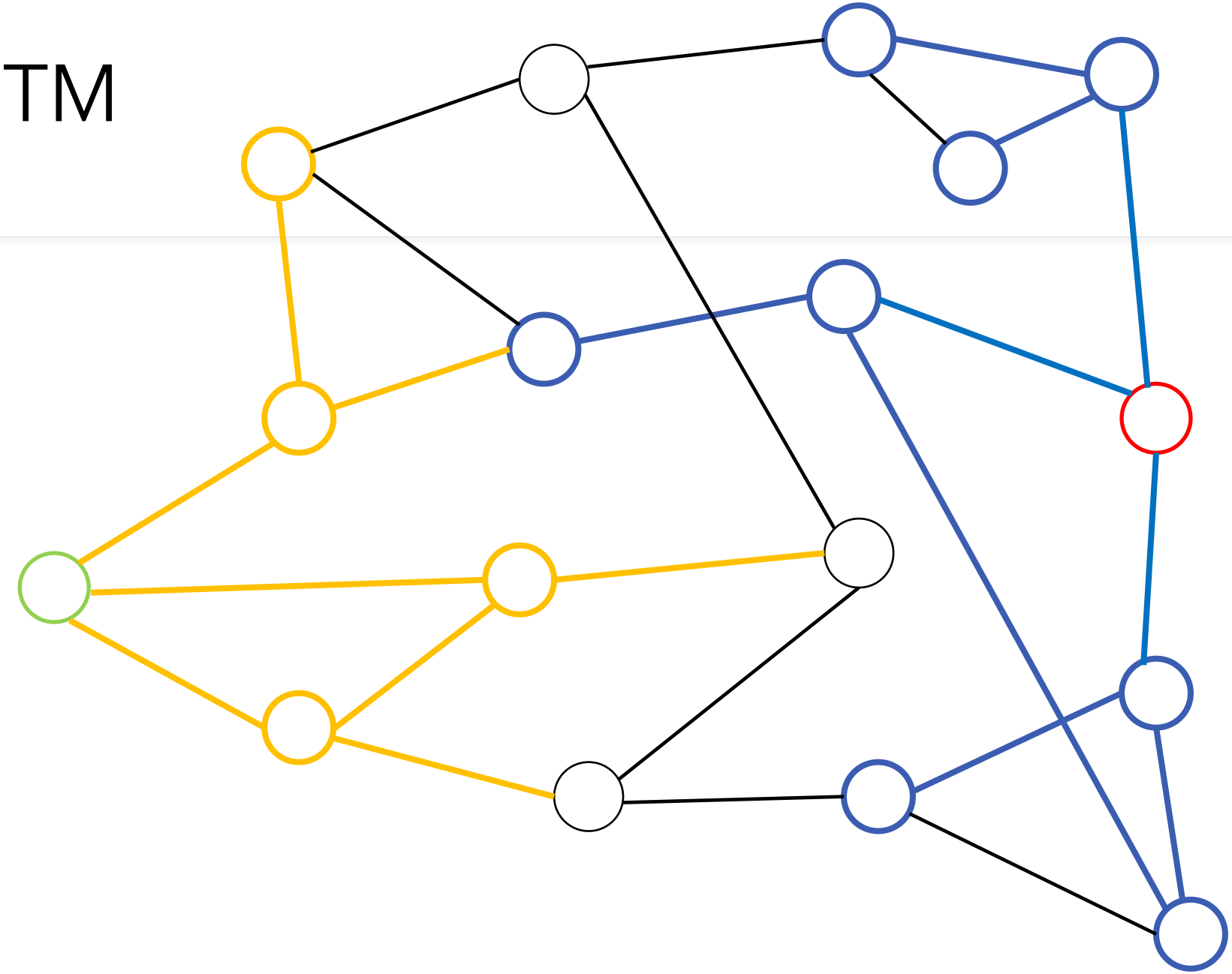
MITM



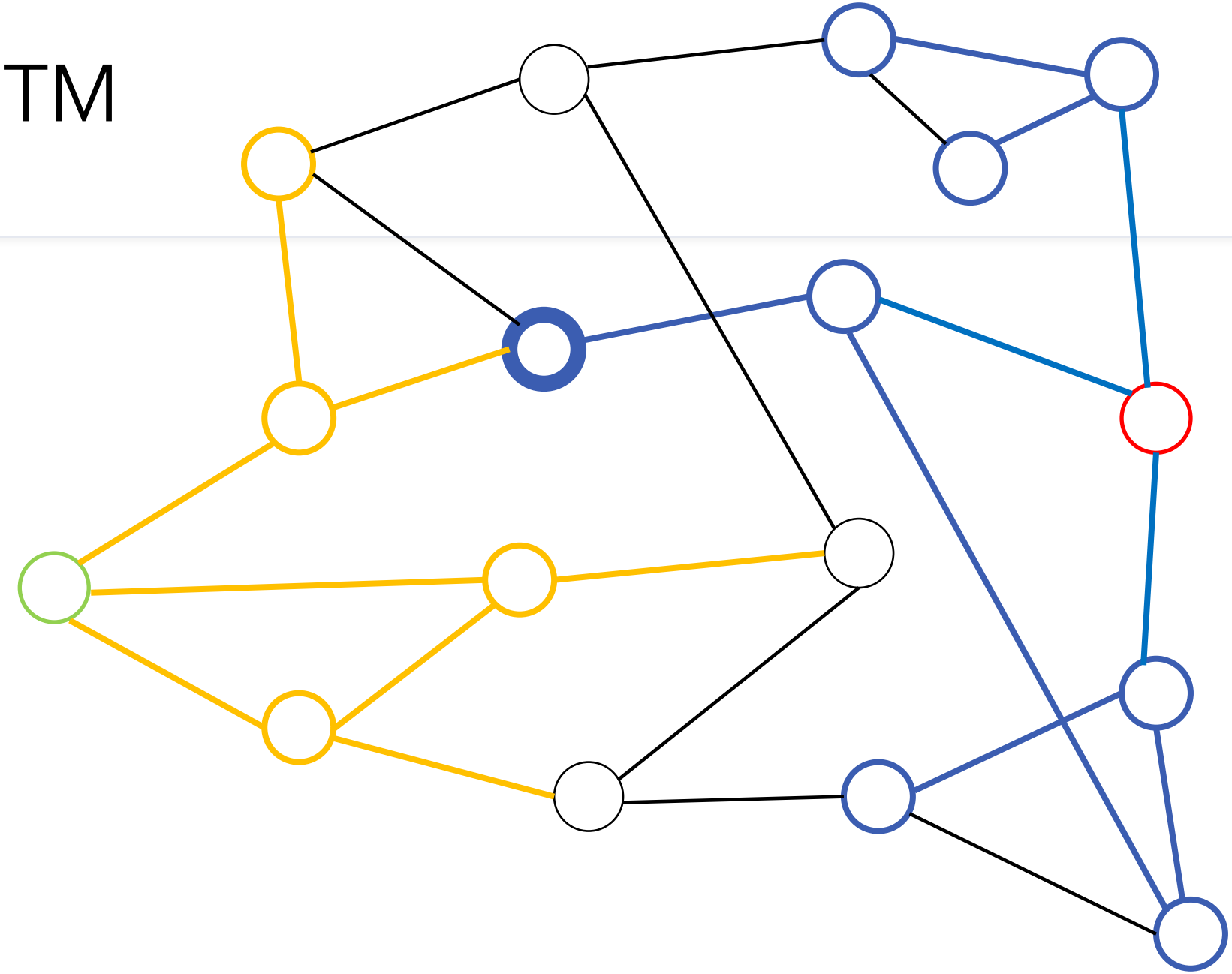
MITM



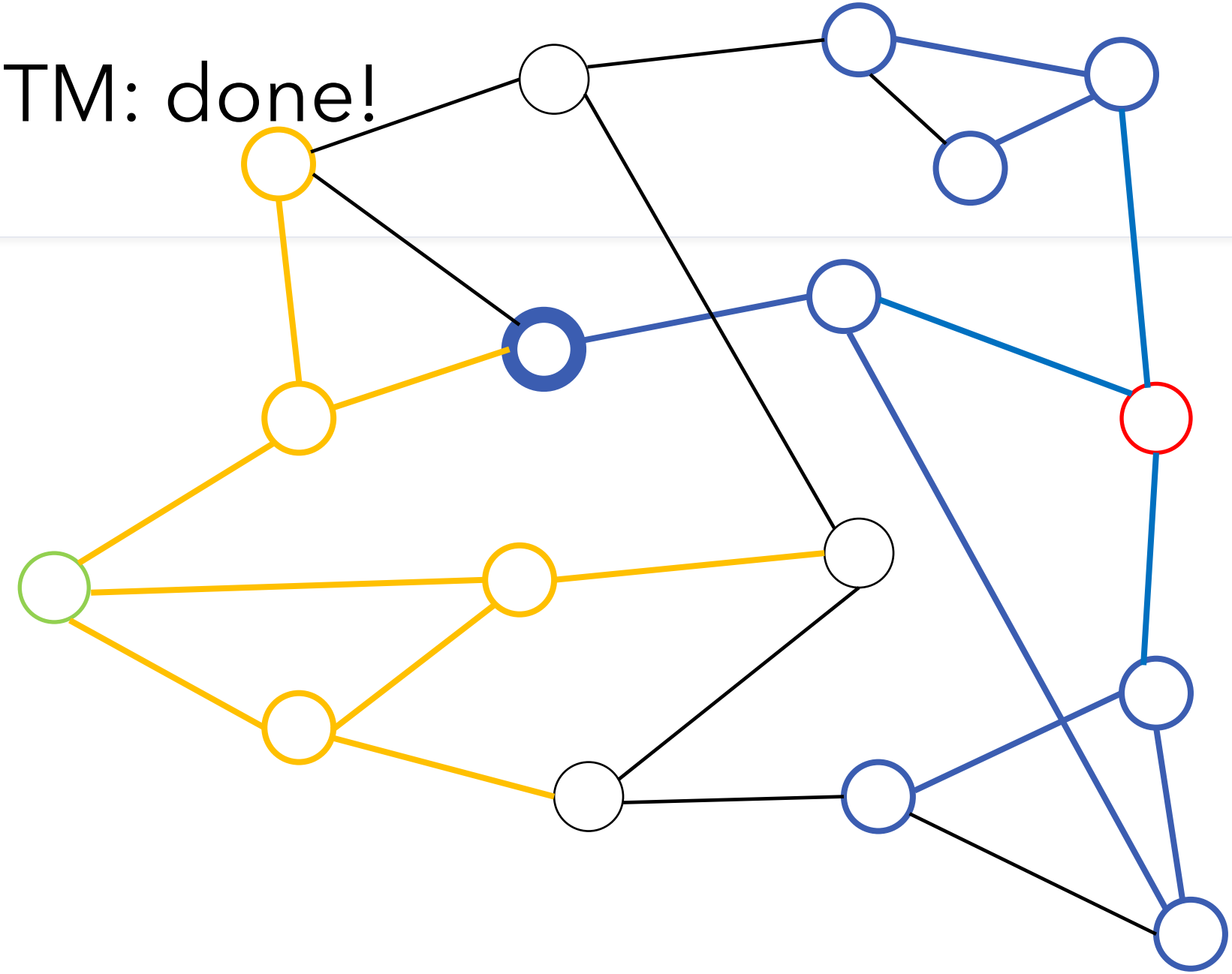
MITM



MITM



MITM: done!

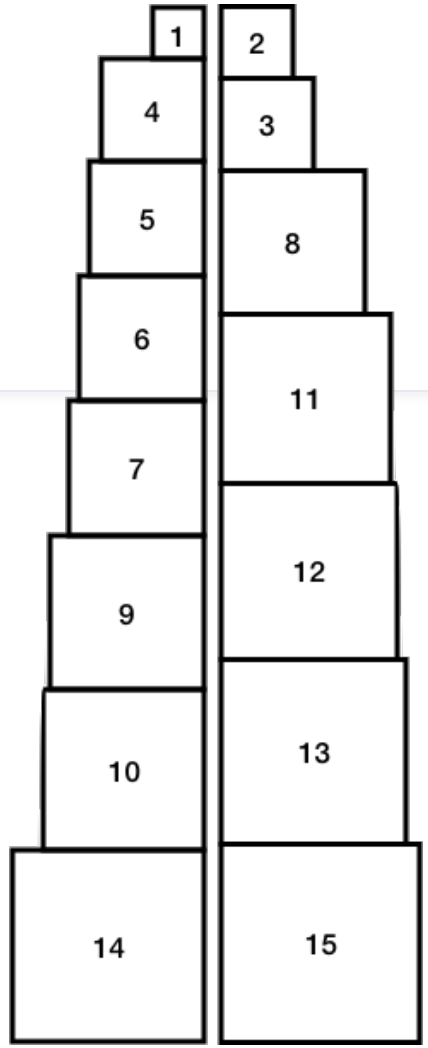


Meet in the middle analysis

- Need to go k/2 steps of BFS backward in a 3-regular graph
  - $O(2^{k/2})$ time
- Need to go k/2 steps of BFS forward in a 3-regular graph
  - $O(2^{k/2})$ time
- Total time and space?
  - $O(2^{k/2})$
- When is this useful?
  - Exponential time is best

Assignment 1: Two towers

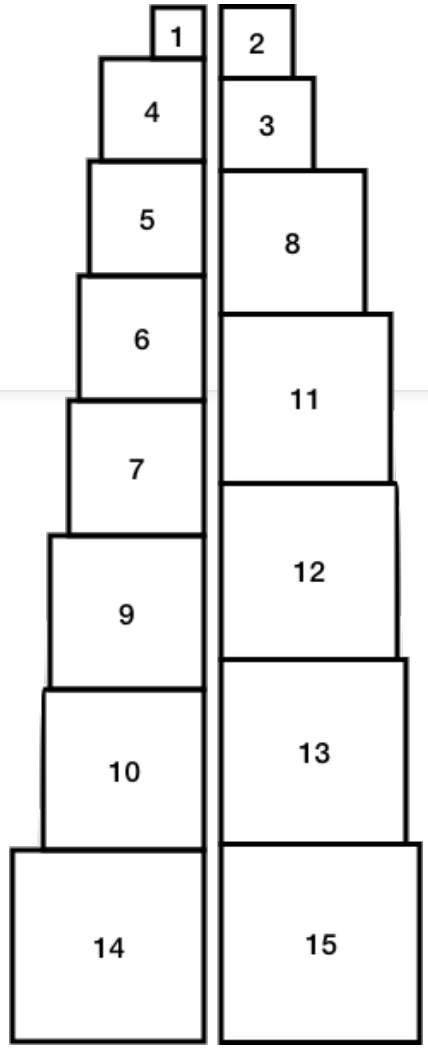
- Given: a sequence of “areas” of blocks
- Goal: stack the blocks into two towers that are as equal as possible



The second-best solution if the blocks have areas {1, 2, …, 15}

Assignment 1: Two towers

- Given: a sequence of “areas” of blocks
- Equivalent goal:
  - find half the total height of the blocks (“target” height)
  - What subset of the blocks is closest to this target without going over?



The second-best solution if the blocks have areas {1, 2, …, 15}

How to solve two towers?

- Solution 1: iterate through all possible subsets of blocks
  - Find the best

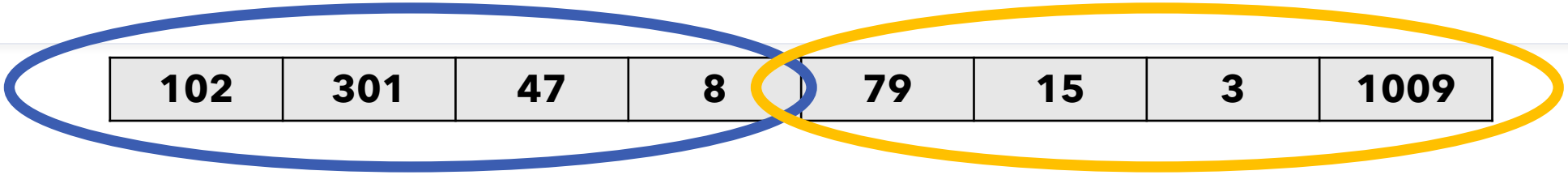
- Time with n blocks? Space?
  - $O(n2^n)$ time, $O(n)$ space

How can we “meet in the middle”?

102	301	47	8	79	15	3	1009

- Need to divide problem into two halves
- Partial solution from each half
- Partial solutions can be combined into a full solution

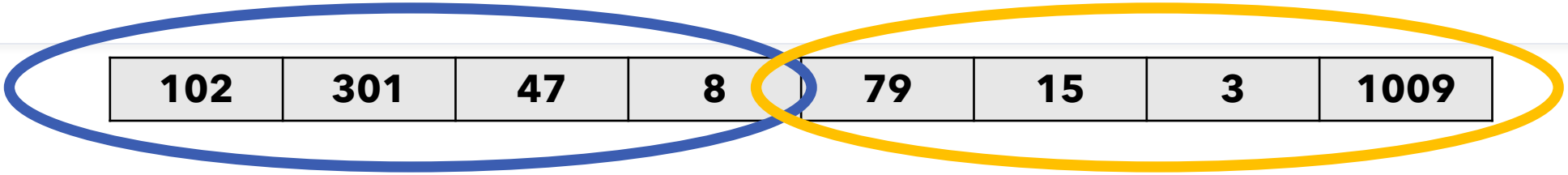
Meet in the middle



- Let's say we have a partial solution of cost X in the left half
- What do we need from the right half?
  - Largest partial solution of cost (target – X)
- We don't have nodes to mark as "visited." How can we keep track of this instead?

Meet in the middle

Target = 41.7



102	301	47	8	79	15	3	1009

10.01	17.34	6.86	2.83	8.88	3.87	1.73	31.76

0	0000
31.76	0001
1.73	0010
33.49	0011
3.87	0100
35.63	0101
5.6	0110
37.36	0111

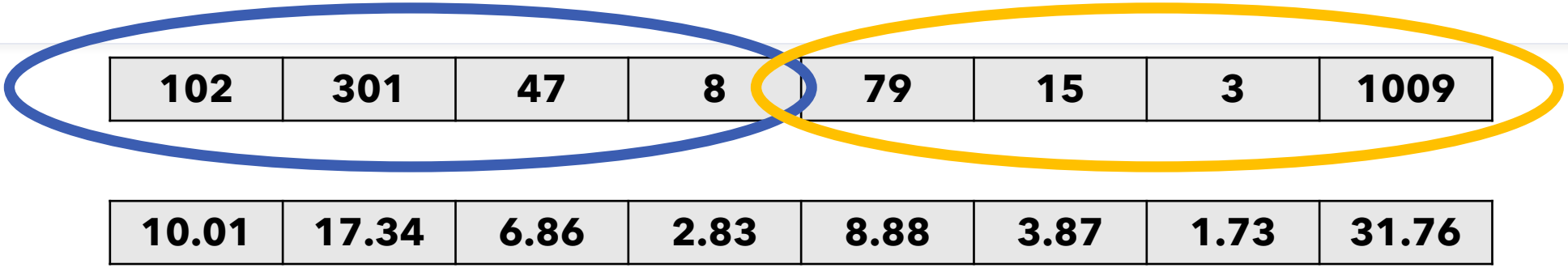
(Top half of table)

What do we want out of this table?

- Query: what is the largest value no larger than Y
  - Y = Target – X
- What is this called?
  - Predecessor query
- What is the simplest data structure to use for this?
  - Sorted array. (Other options too!)

Meet in the middle

Target = 41.7

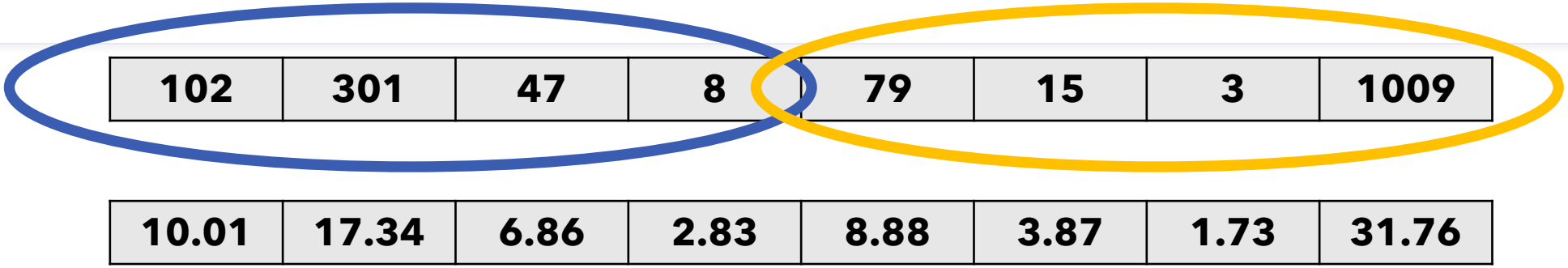


0	0000
31.76	0001
1.73	0010
33.49	0011
3.87	0100
35.63	0101
5.6	0110
37.36	0111

8.88	1000
40.64	1001
10.61	1010
42.37	1011
12.75	1100
44.51	1101
14.48	1110
46.24	1111

Meet in the middle

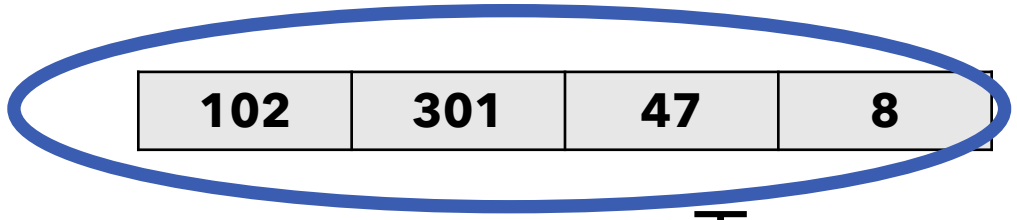
Target = 41.7



0	0000
1.73	0010
3.87	0100
5.6	0110
8.88	1000
10.61	1010
12.75	1100
14.48	1110

31.76	0001
33.49	0011
35.63	0101
37.36	0111
40.64	1001
42.37	1011
44.51	1101
46.24	1111

Meet in the middle



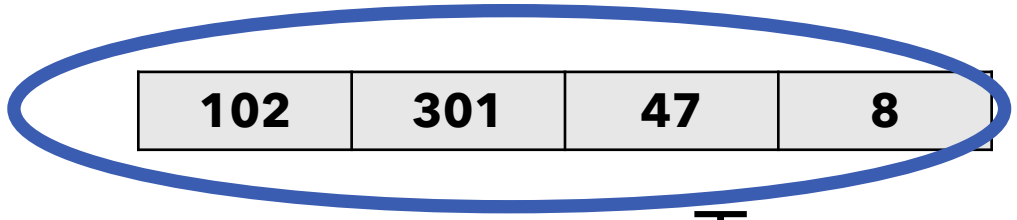
Target = 41.7

- Go through each subset of remaining half
- Look up best solution in the table!
- {102, 47} -> {10.10, 6.86}
- Sum = 16.96
- Want sln less than 41.7 – 16.96 = 24.74

0	0000
1.73	0010
3.87	0100
5.6	0110
8.88	1000
10.61	1010
12.75	1100
14.48	1110

31.76	0001
33.49	0011
35.63	0101
37.36	0111
40.64	1001
42.37	1011
44.51	1101
46.24	1111

Meet in the middle



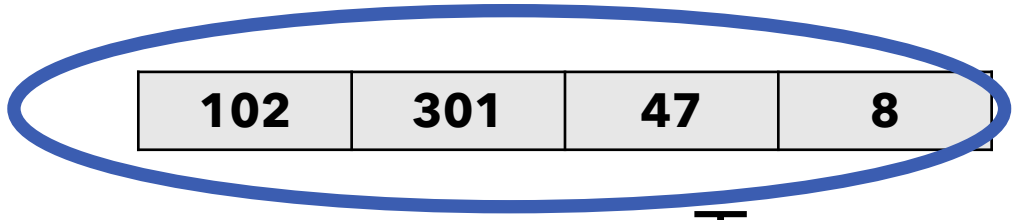
Target = 41.7

- Go through each subset of remaining half
- Look up best solution in the table!
- {102, 47} -> {10.10, 6.86}
- Sum = 16.96
- Want sln less than 41.7 – 16.96 = 24.74

0	0000
1.73	0010
3.87	0100
5.6	0110
8.88	1000
10.61	1010
12.75	1100
14.48	1110

31.76	0001
33.49	0011
35.63	0101
37.36	0111
40.64	1001
42.37	1011
44.51	1101
46.24	1111

Meet in the middle



Target = 41.7

0	0000
1.73	0010
3.87	0100
5.6	0110
8.88	1000
10.61	1010
12.75	1100
14.48	1110

31.76	0001
33.49	0011
35.63	0101
37.36	0111
40.64	1001
42.37	1011
44.51	1101
46.24	1111

- What is the guarantee for {102, 47}?
- Of all solutions that contain exactly 102 and 47 from the first 4 numbers, the best solution is {102, 47, 79, 15, 3}
- Height = 14.48 + 16.96 = 31.44

Meet in the middle for two towers

- Take last n/2 items
- Calculate height of all subsets
  - How many subsets?
  - How long does calculating the height take?
- Store in sorted table
  - How long does sorting take?

Meet in the middle for two towers

- Take first n/2 items
- For all subsets:
  - Calculate their height X
  - Look up largest number smaller than (target – X) in the table
    - (How? How long does that take?)
  - If resulting total height is best so far, store it

Meet in the middle analysis

- Making the table time and space:
  - $O(n2^{n/2})$ time, $O(n2^{n/2})$ *bits* of space (how many 64-bit words?)
- Searching the table time and space:
  - $O(n2^{n/2})$ time, $O(1)$ words of space (assuming $n \leq 64$)

- Exponential speedup compared to $O(n\ 2^n)$ brute force

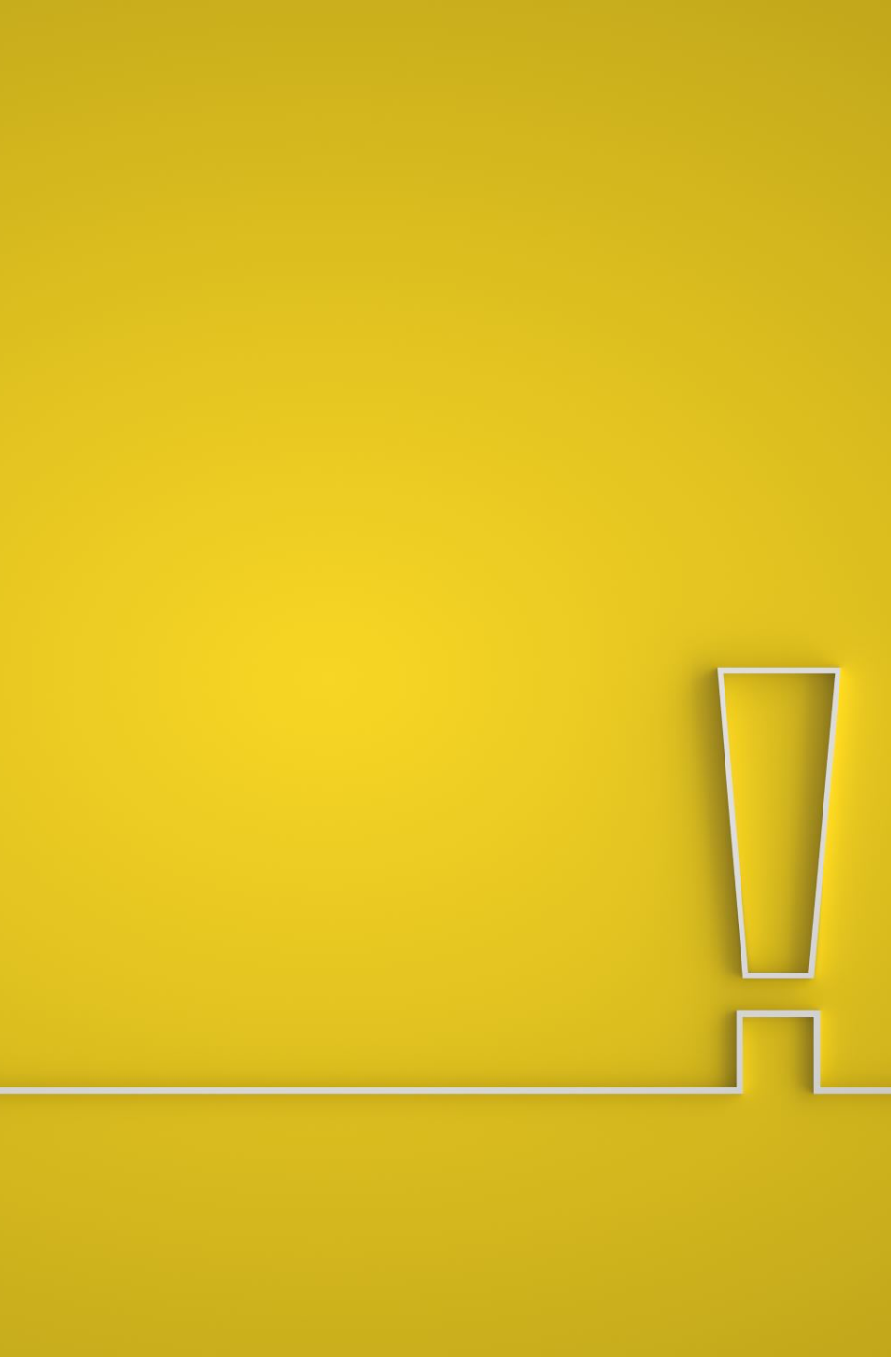
Can we do better than a sorted table?

- What do we need from this table?
- Can we use a hash table?
- Can we use a tree? What advantages might a tree provide?

MITM: Enormously useful trick

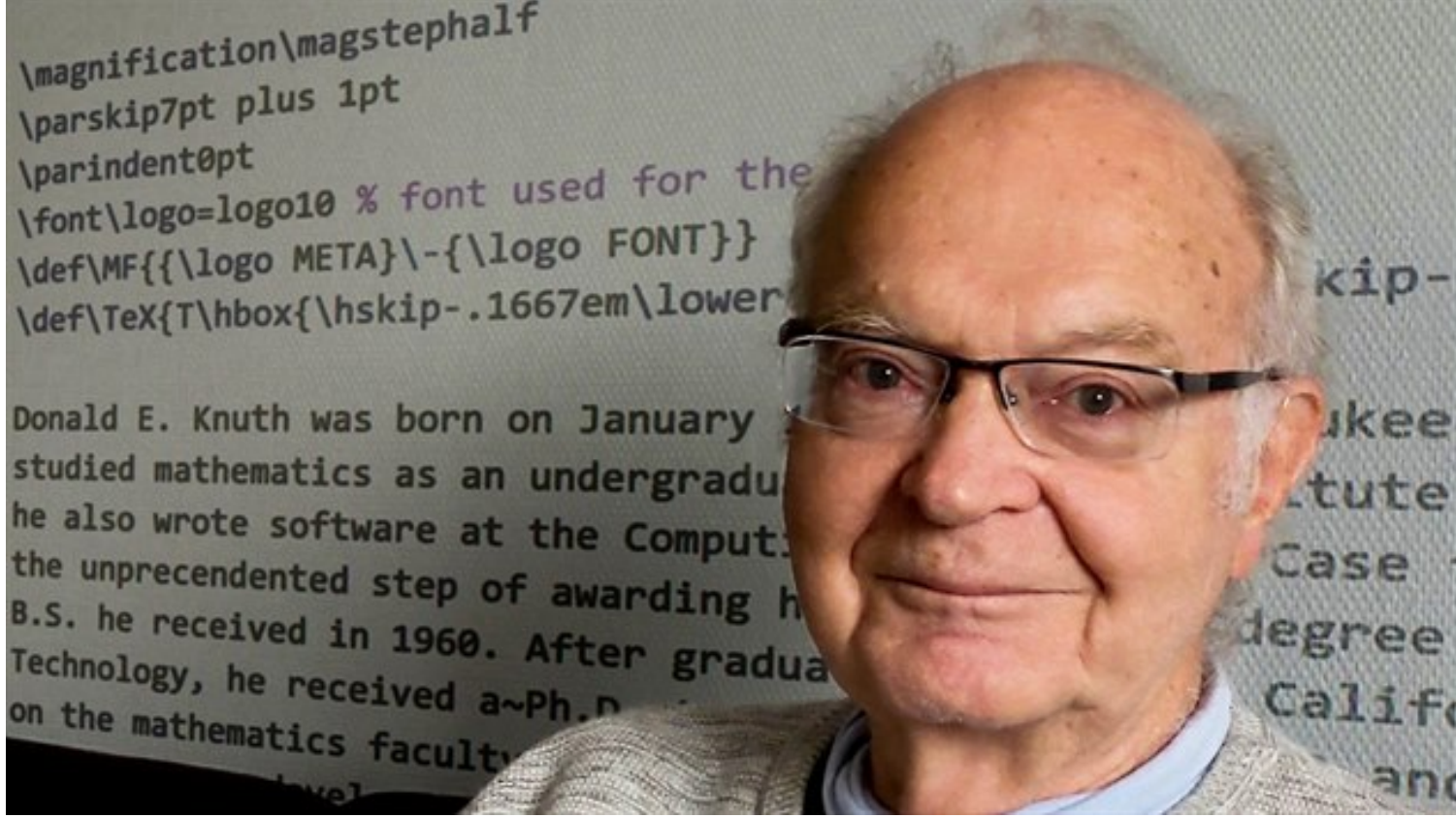
- Very common in crypto in particular
  - Big step, little step
- Use extra space, but take MUCH less time
  - Worth it?

Let’s get back to C



Careful coding

- Good coding practice is much much much more important than ever
- Include asserts to check array ranges
- Code, test, code, test
- Split into functions and test separately!
- Check your pointers!
- Corner cases! (Is this pointer null? Is this value 0?)
- Speed is not your first priority, correctness is



Course motto(s)

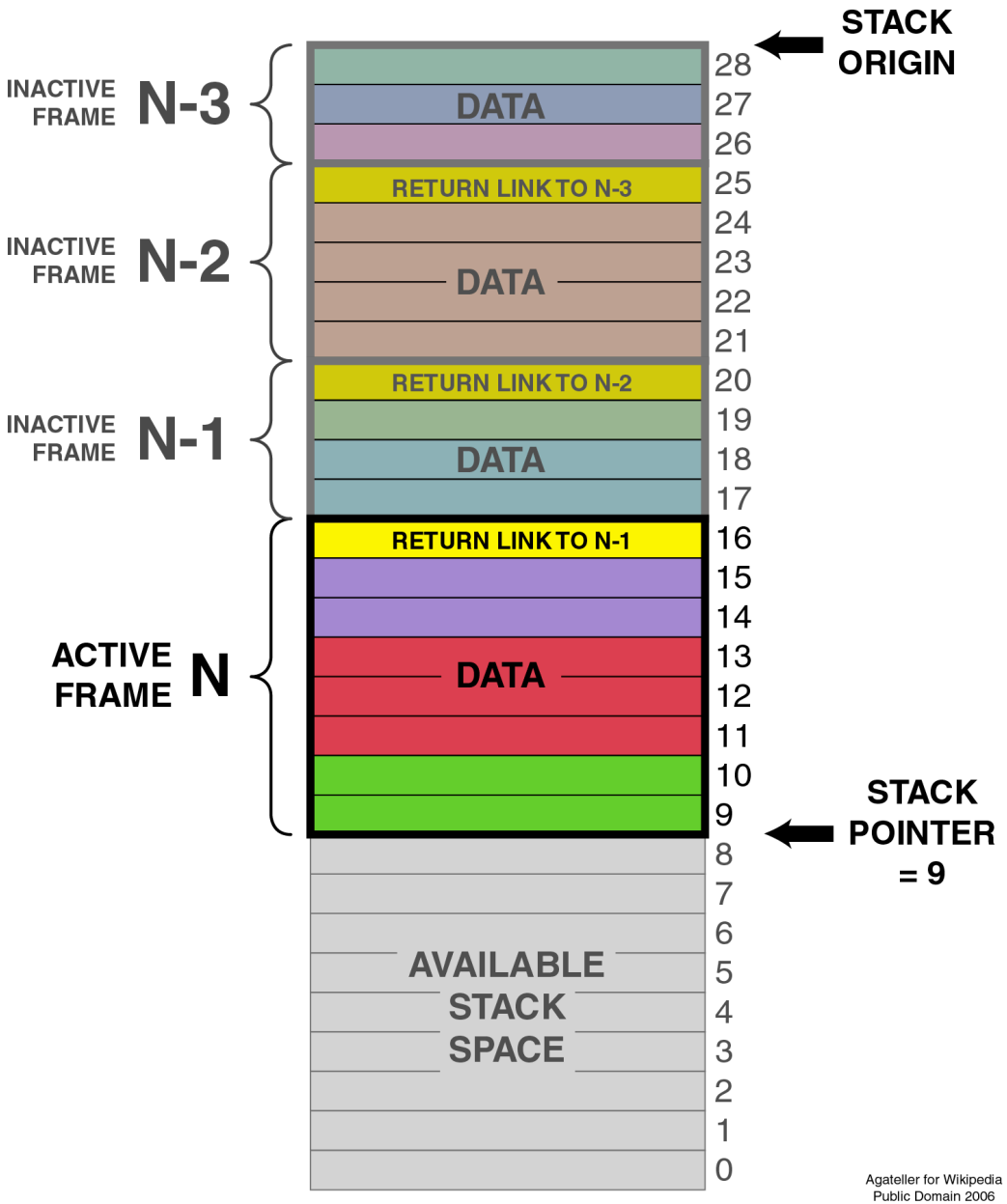
- “Premature optimization is the root of all evil”
- “We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%.”

Pointers, functions, and structs

- Creating function
- Passing is *always* by value. Can pass struct instances
- How do we change a variable inside a function?
  - Pass the address—the address doesn't change, but the value does!
- -> operator
- Structs stored contiguously in memory

Allocation

- “new” in Java and C++ allocates space for a new instance of a variable
- C uses “malloc”
- Very much user-controlled: you set the space, no garbage collection



Agateller for Wikipedia
Public Domain 2006

Where are things stored?

- First place: in CPU register, never in memory
  - Temporary variables like loop indices
  - Compiler decides this
- Second place: call stack
  - Small amount of dedicated memory to keep track of current function and local variables
  - Pop back to last function when done
  - Temporary!

Third place: the heap

- Very large amount of memory (basically all of RAM)
- Using new in Java or C++ puts variable on the heap
- We use malloc
  - Does not zero out memory. calloc does
  - C will not make you instantiate your variables
- Needs stdlib.h
- Returns pointer; don’t need to cast to pointer type

Ways to store things

- Speed: registers > stack > heap
- Size: heap > stack > registers
- Longevity: heap > stack > registers

- Java rules work out well: store ”objects” and arrays on heap, just declare small “primitive types” and let the compiler work it out (Remember scope!!)

Allocation, pointers, and arrays

- What is an array?
- Can we use arrays without using array-like things?
  - Using pointers and malloc instead?
- Does this allow us to allocate arrays dynamically?

- Pointers and arrays are (mostly) *equivalent* in C

Memory leaks

- C does not have a garbage collector
  - Fast, efficient, you actually really want to be able to control this
  - But, obviously, huge pain and difficult to debug
- free() releases memory
  - Can be used for another variable
  - Not zeroed out
- Every malloc() should have a free()!
- After your program ends all memory is released

Segmentation faults

- Access “illegal” memory
  - Address that the OS didn’t give your program
- Given very very little information
- Debug using gdb (checkpoints, etc.)
- valgrind is useful for checking memory
- We’ll see some examples of these Thursday

Compiling and building

- Include and function declarations
- Compile: convert code into machine-executable code
  - gcc –c [file name]
- Link: stitch together function calls between files
- Build: whole process
  - What gcc actually does when given file
  - Need to list compiled object files
- Student example

What happens when we change one file?

- Need to recompile that file
- Need to build final output file

- Can we do this automatically?

Makefile

- Lists dependencies
- Lists what you actually want to build
- Entire command: make
- If a file changes, compiles only what’s necessary

- Very very useful!

In this class

- I will give you makefile
- Don’t need to change unless you use multiple files
  - You can, but probably won’t ever need to
  - Projects in this class are fairly small and self-contained

Variable types

- Int, long, etc. not necessarily the same on different systems
  - (If you use Windows long is likely 32 bits, but on Mac and Unix it's generally 64 bits)
  - (long long is 64 bits)
- Include stdint.h
- Unsigned (?)

Variable types

- Ints are OK for things like small loops
- If you care at all about size, should use int64_t
  - Fixed platforms means you don’t NEED to for this class
  - Good to get in the habit
- Unsigned is up to you
  - Controversial if they’re a good idea