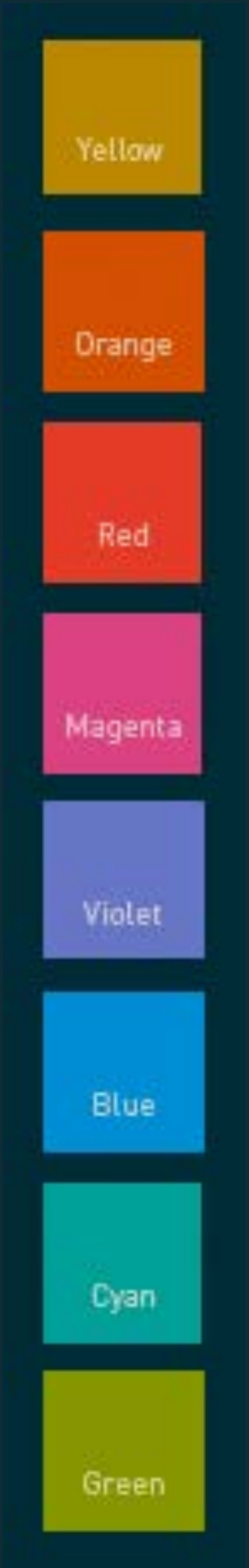
https://github.com/martinchapman/BuildX(.git)

BUILD X: ALGORITHMS

MARTIN CHAPMAN

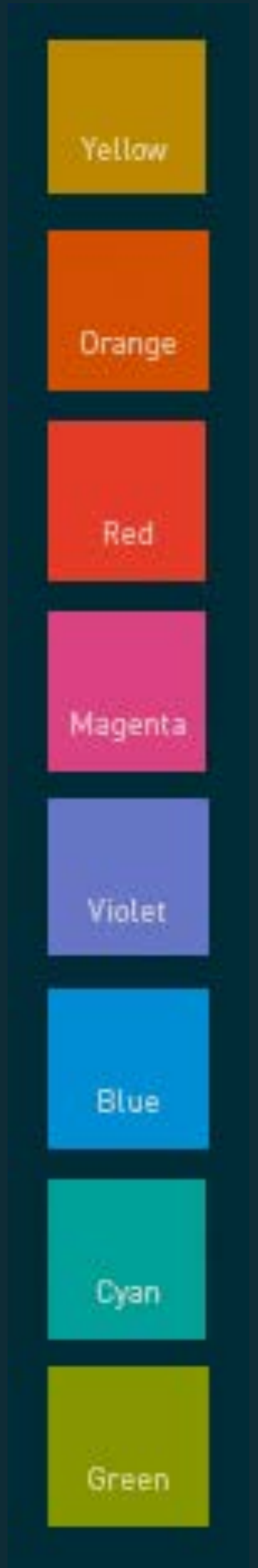
GRAPHS



BUILD X: ALGORITHMS

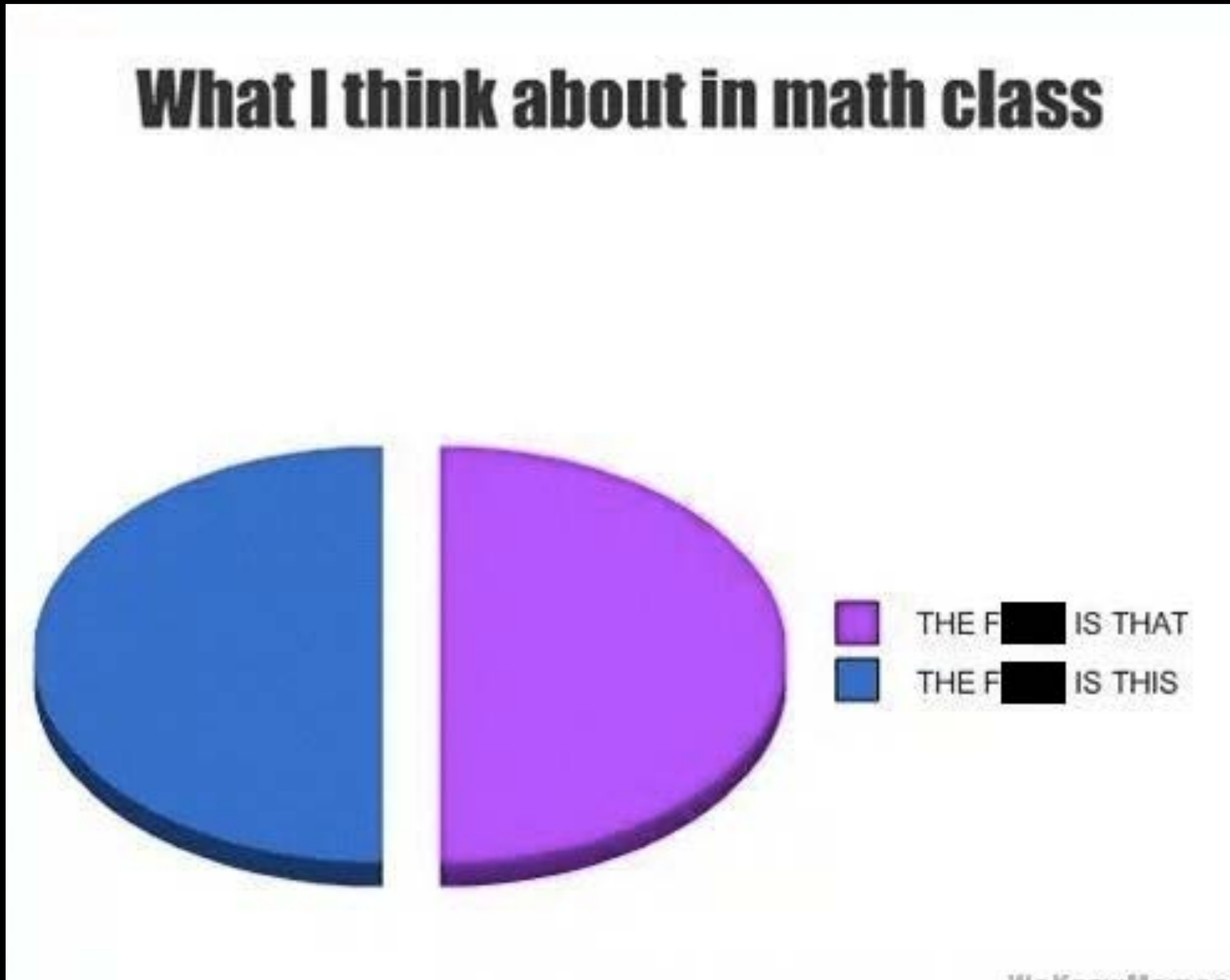
MARTIN CHAPMAN

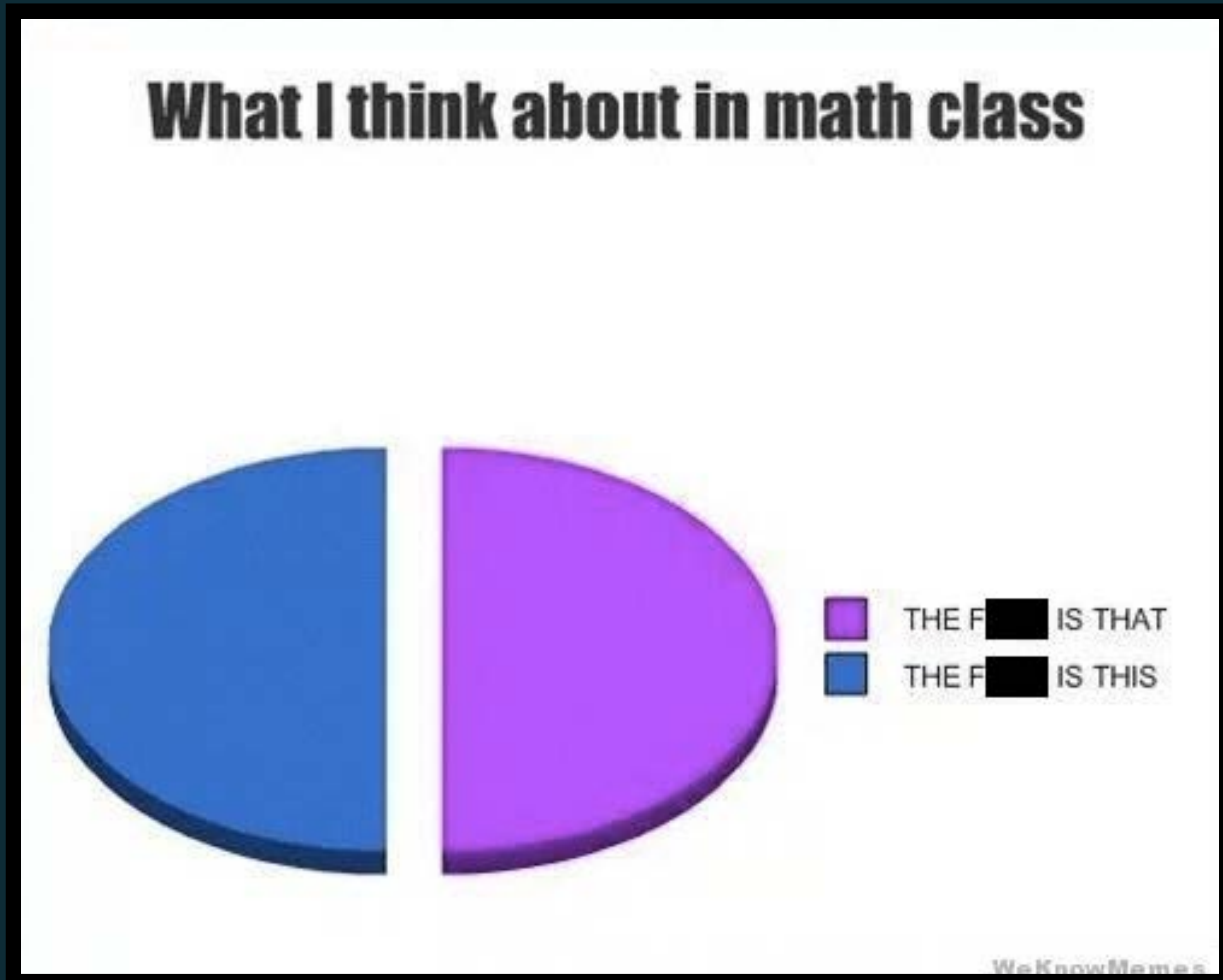
GRAPHS (sort of)



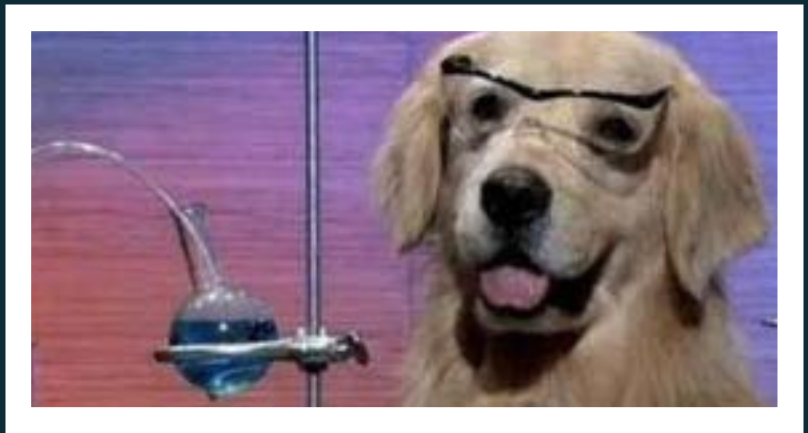

VIA 9GAG.COM
A MATT DAMON MATT DAMON
MATT
DAMON
MATT DAMON
MATT DAMON
THE
MATT DAMON
IN CINEMAS SEPTEMBER 30 IN 3D

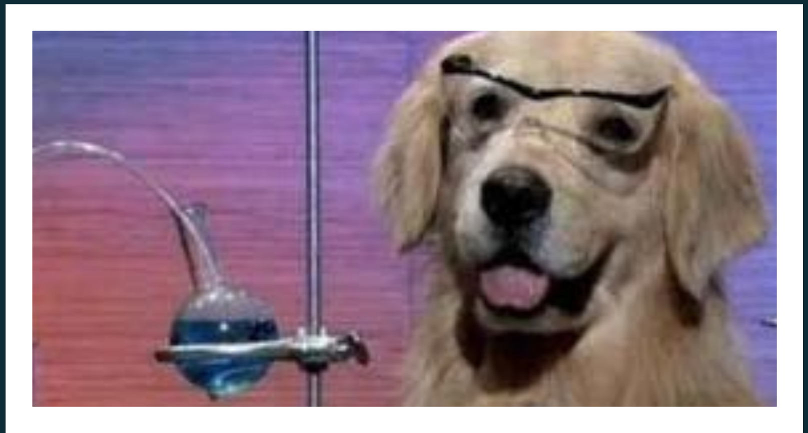

DON'T LISTEN TO THEM
I THINK YOU'RE FINTASTIC


What I think about in math class
THE F[redacted] IS THAT
THE F[redacted] IS THIS


What I think about in math class
THE F[redacted] IS THAT
THE F[redacted] IS THIS
WeKnowMemes


I HAVE NO
IDEA WHAT
I'M DOING



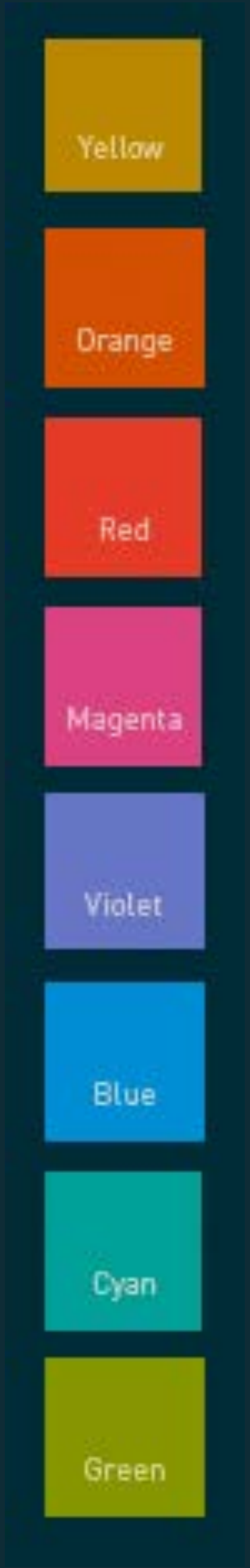


Magenta 211 54 130 #d33682

?

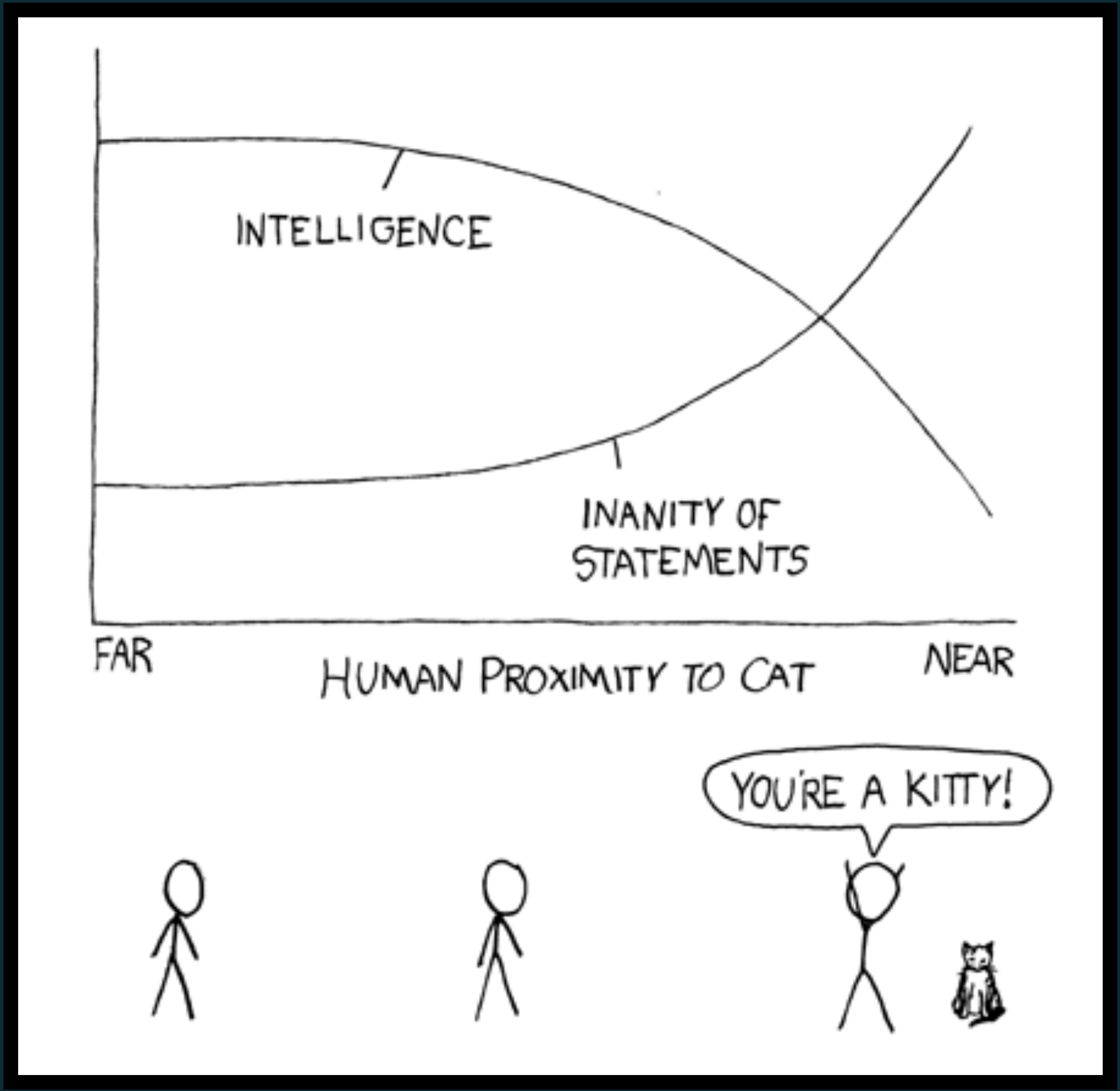
BUILD X: ALGORITHMS

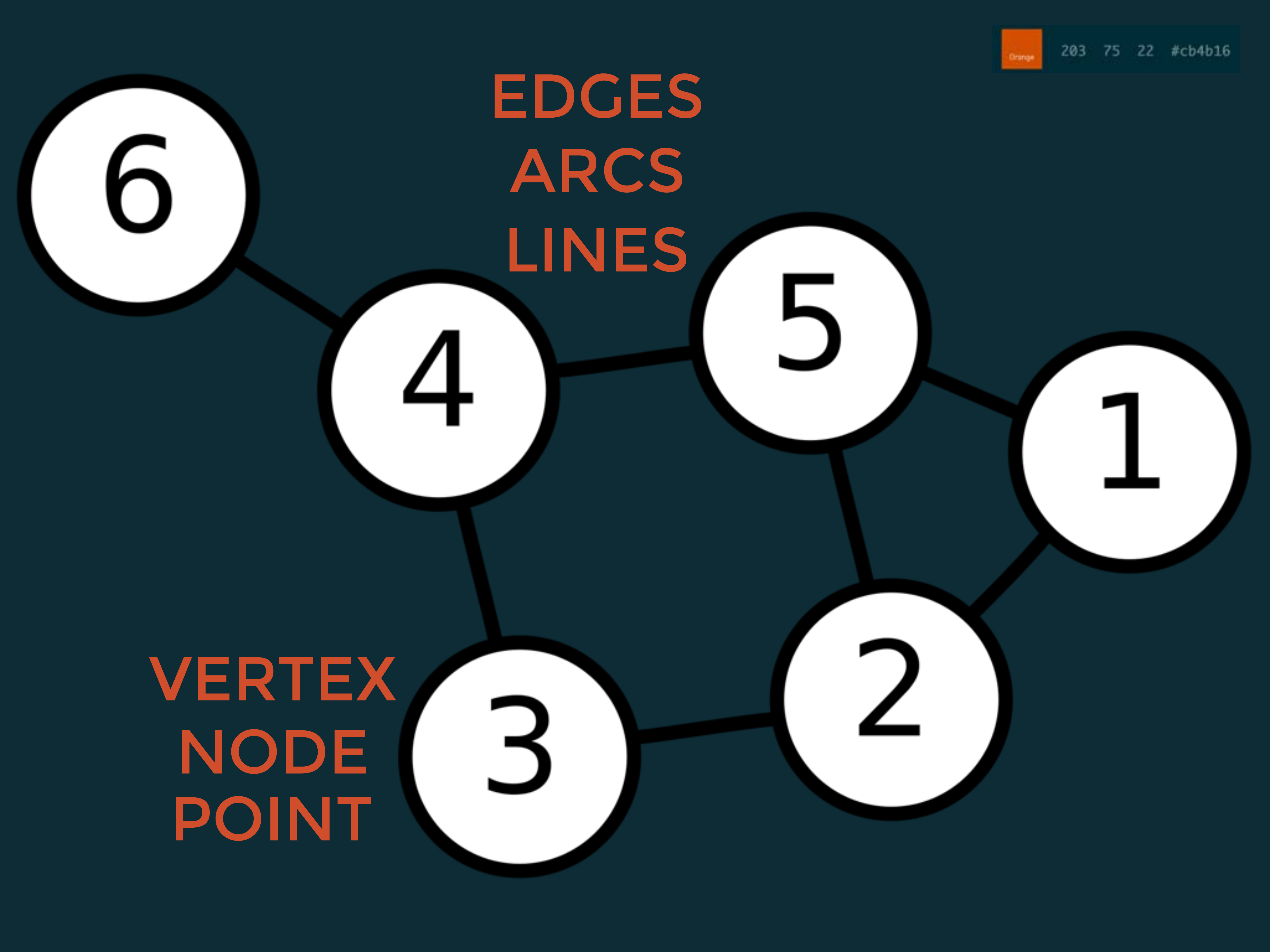
GRAPHS (sort of)



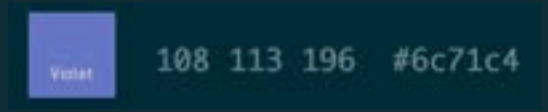
Yellow 181 137 0 #b58900

GRAPH

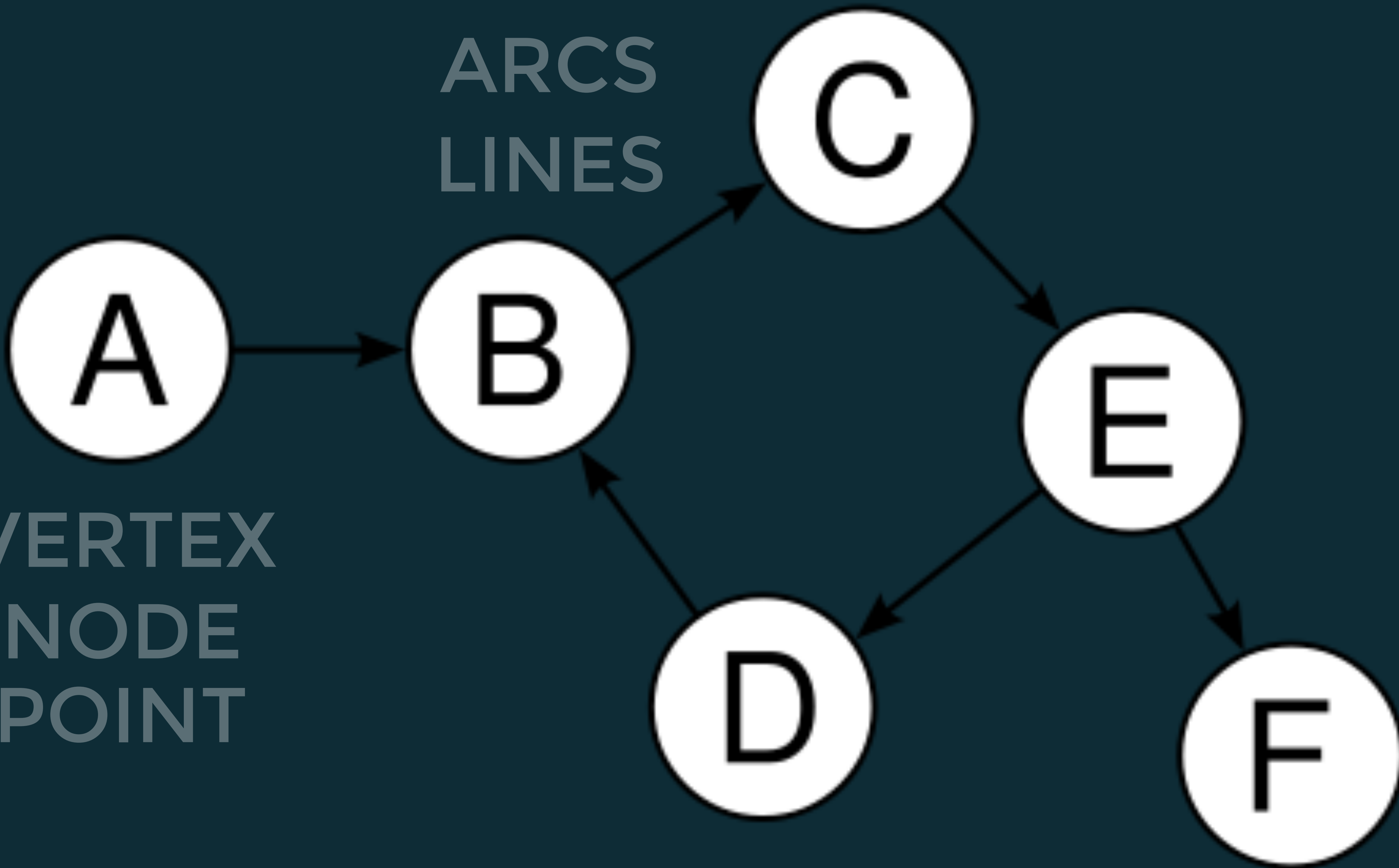

INTELLIGENCE
INANITY OF STATEMENTS
FAR
NEAR
HUMAN PROXIMITY TO CAT
YOU'RE A KITTY!



Orange
203 75 22 #cb4b16
EDGES
ARCS
LINES
6
4
5
1
2
3
VERTEX
NODE
POINT

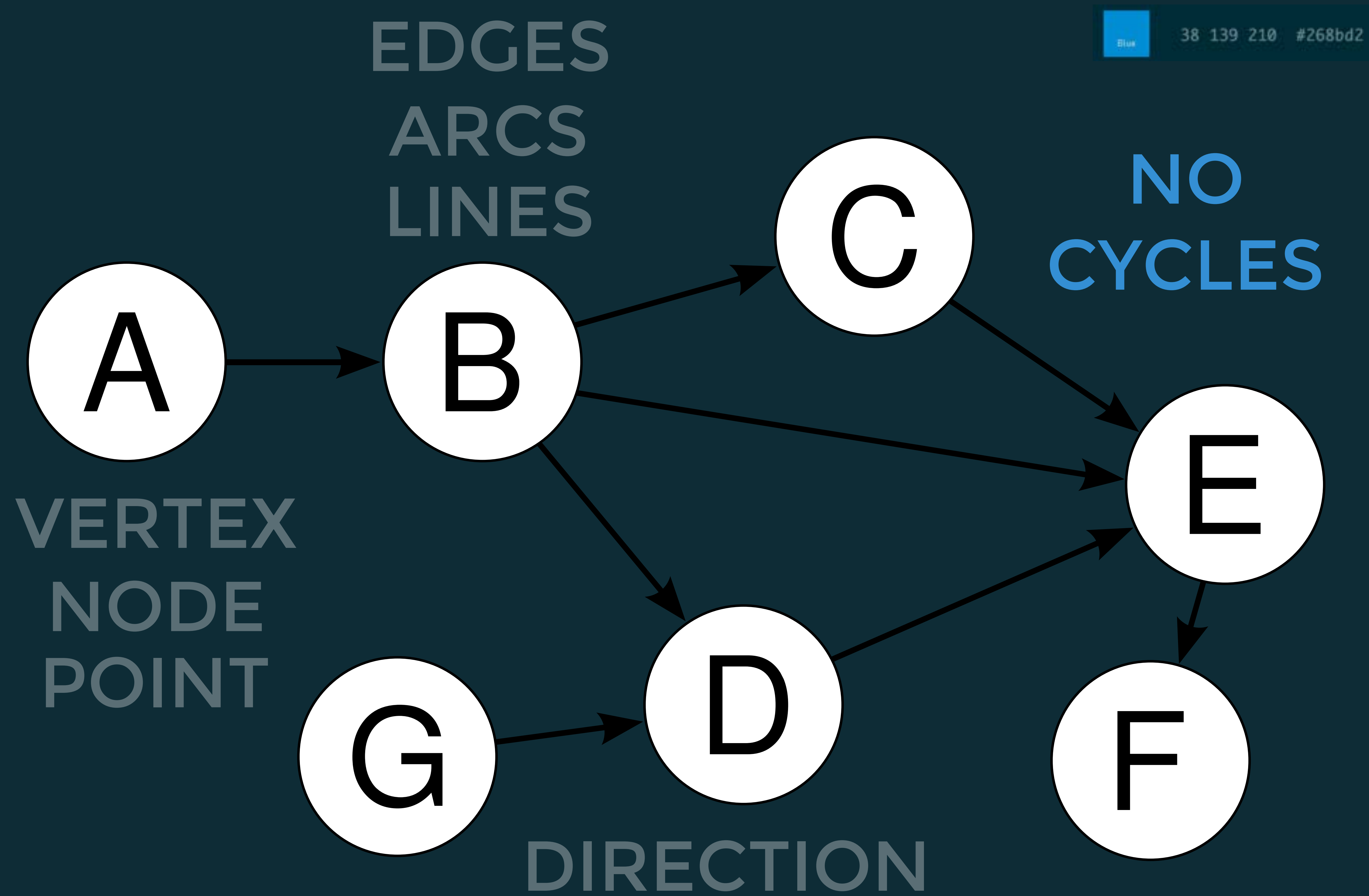


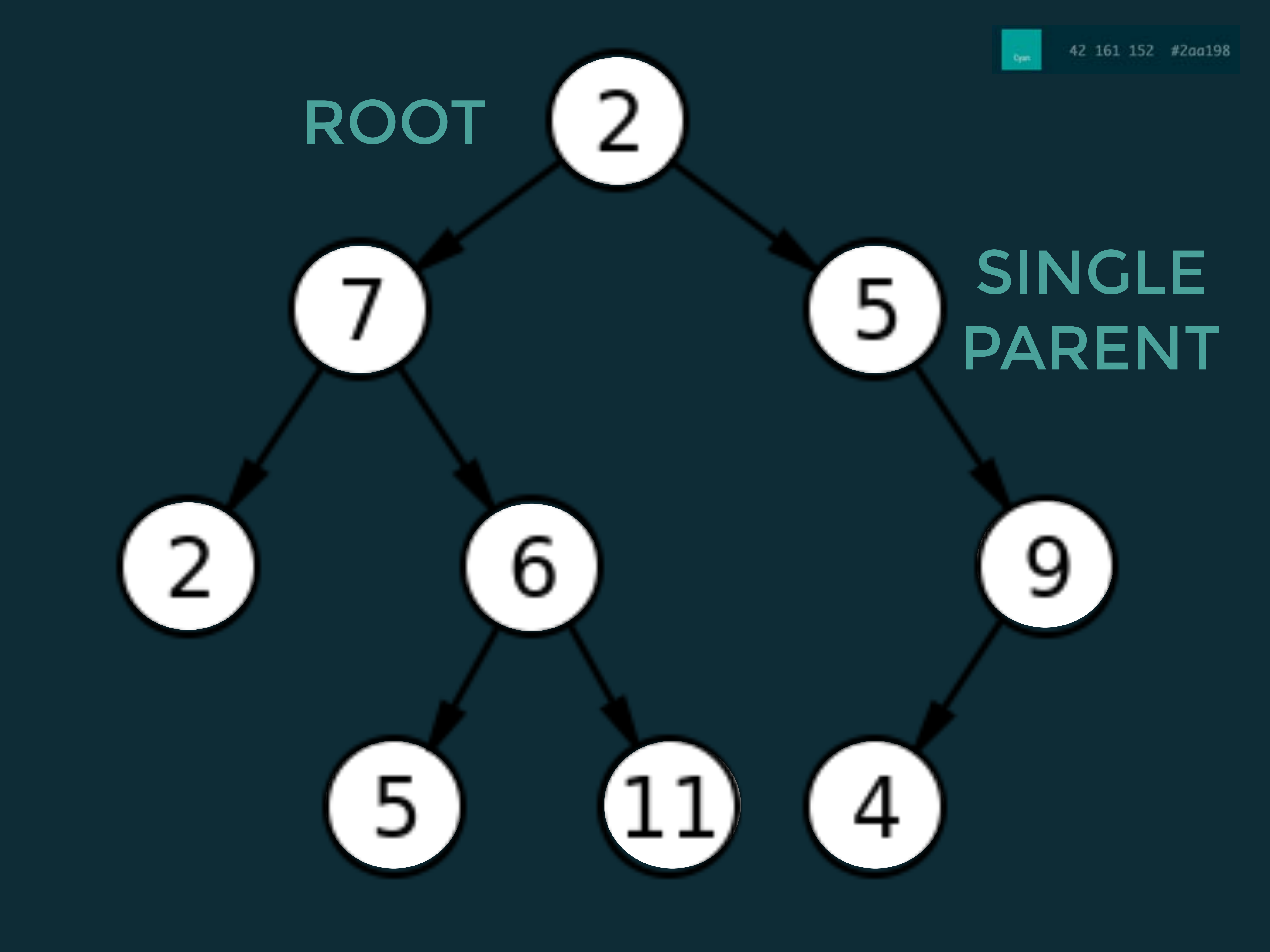
EDGES
ARCS
LINES



VERTEX
NODE
POINT

DIRECTION

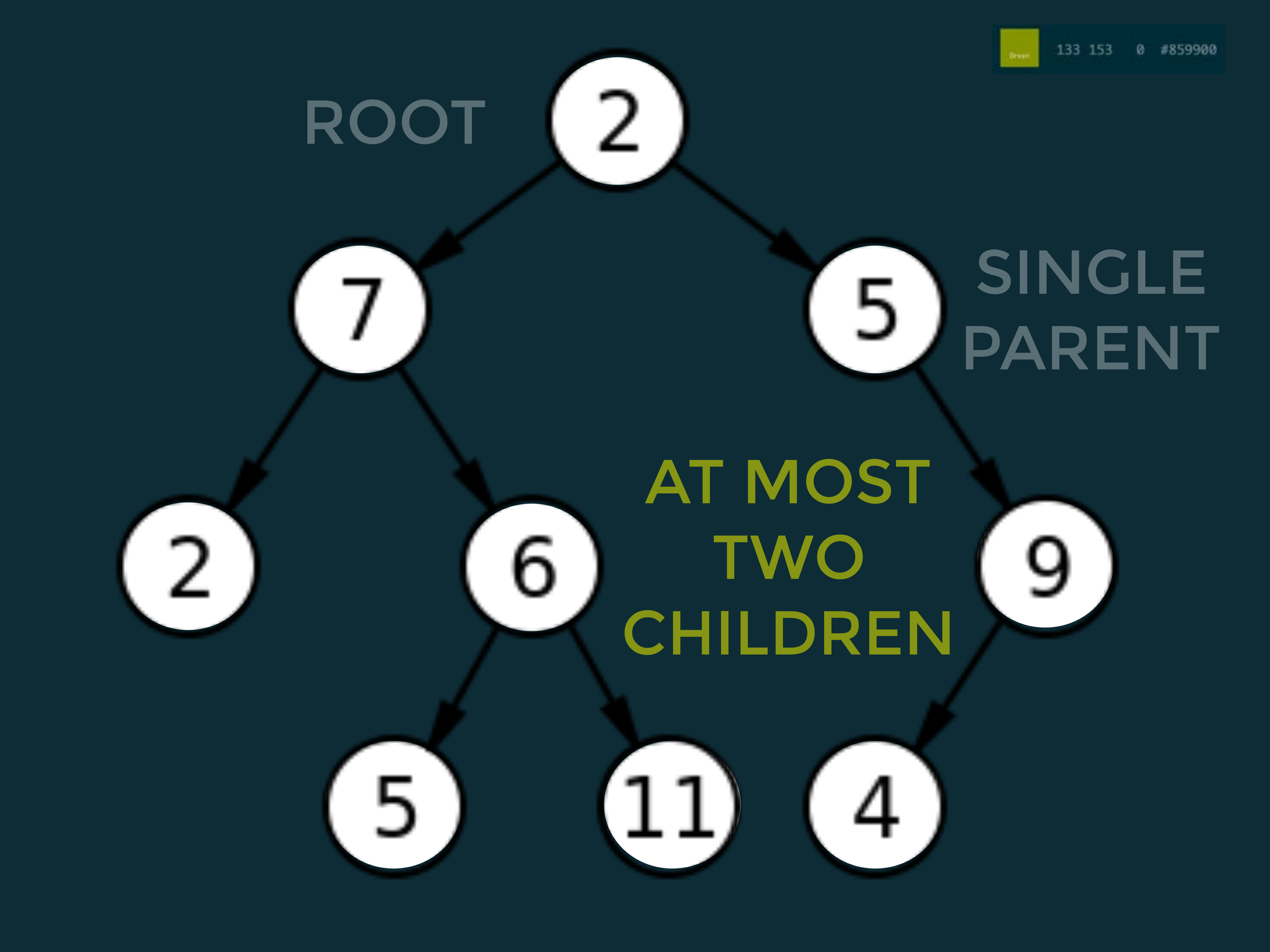

EDGES
ARCS
LINES
NO
CYCLES
38 139 210 #268bd2
A
B
C
D
E
F
G
VERTEX
NODE
POINT
DIRECTION

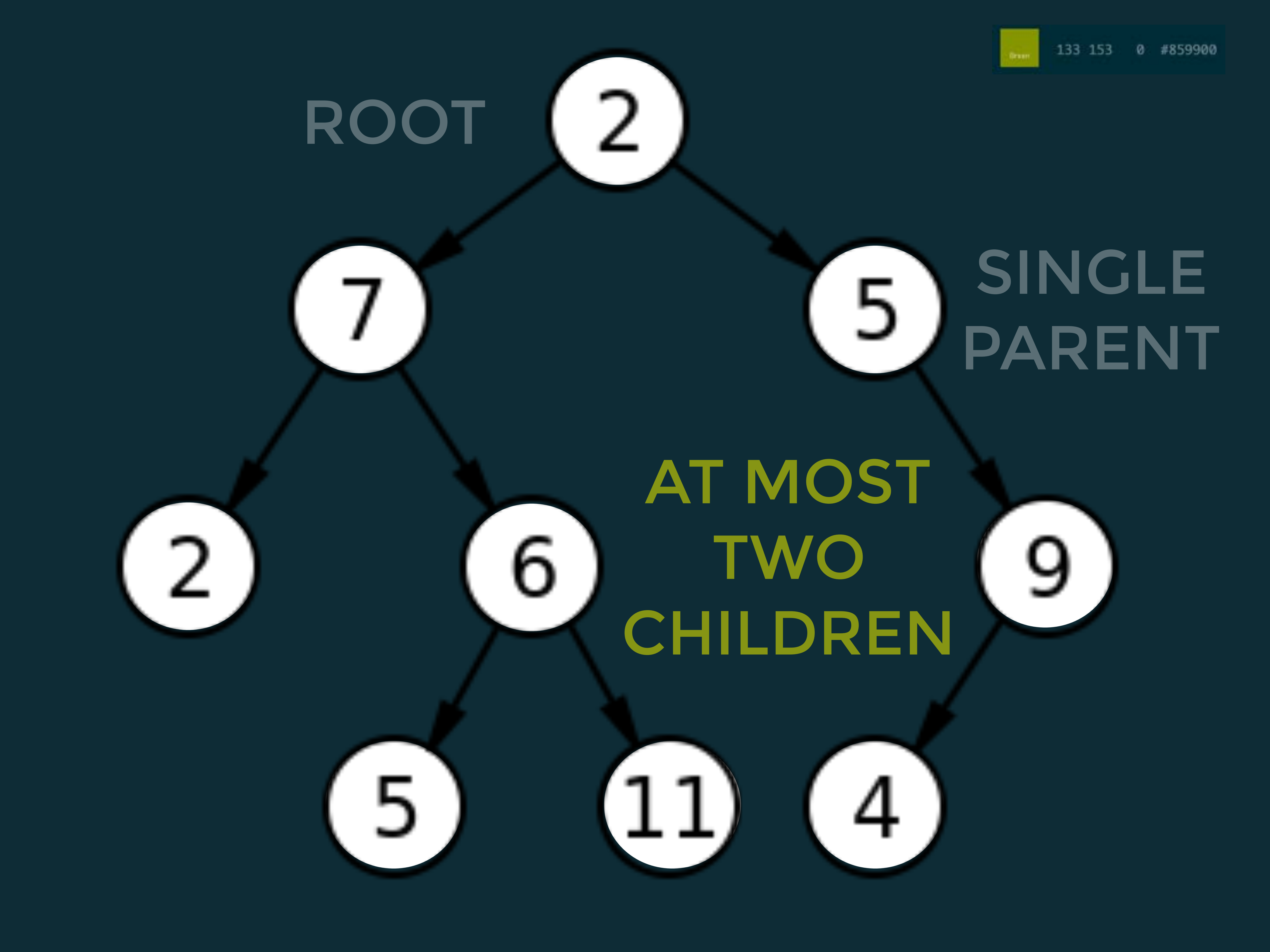

42 161 152 #2aa198
ROOT
SINGLE
PARENT
2
7
5
2
6
9
5
11
4

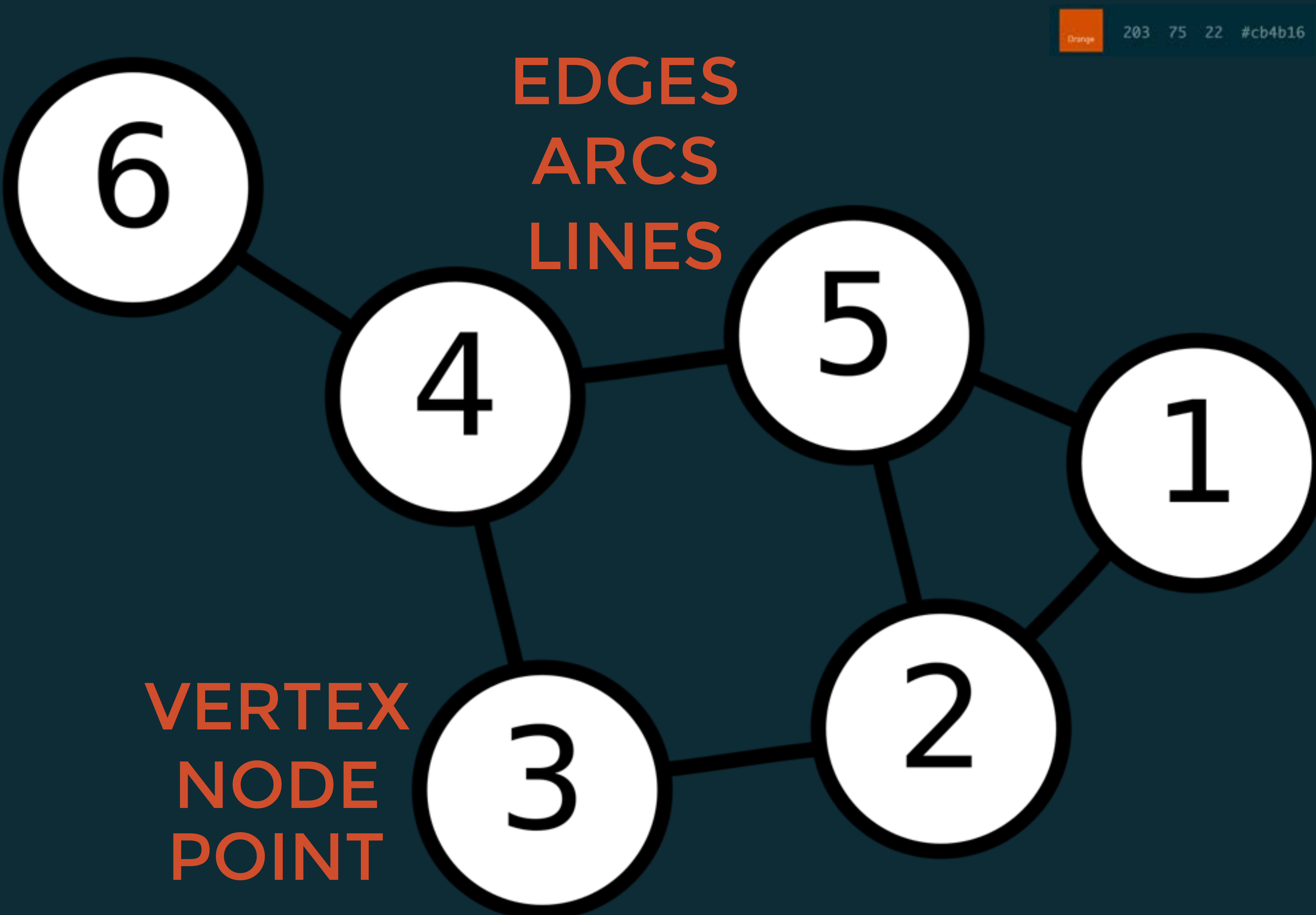
220 50 47 #dc322f

TREE

(other definitions exist)

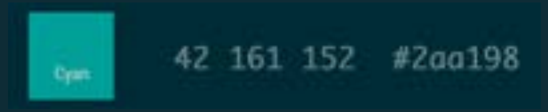

Green
133 153 0 #859900
ROOT
2
7
5
SINGLE PARENT
AT MOST TWO CHILDREN
2
6
9
5
11
4


133 153 0 #859900
ROOT
2
7
5
SINGLE PARENT
2
6
AT MOST TWO CHILDREN
9
5
11
4


203 75 22 #cb4b16
EDGES
ARCS
LINES
6
4
5
1
2
3
VERTEX
NODE
POINT

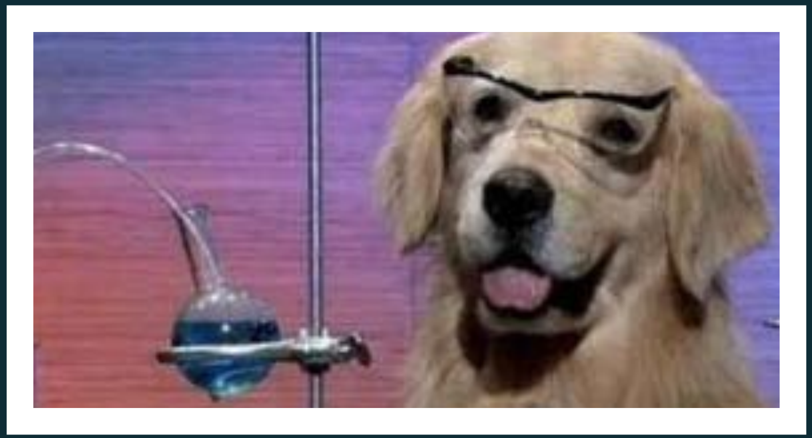
133 153 0 #859900

HOW DO WE CONSTRUCT A GRAPH?



TOPOLOGY AND GENERATORS

http://graphstream-project.org/doc/Generators/



Magenta 211 54 130 #d33682

RANDOM

Béla Bollobás. Random Graphs. Springer, 1998.

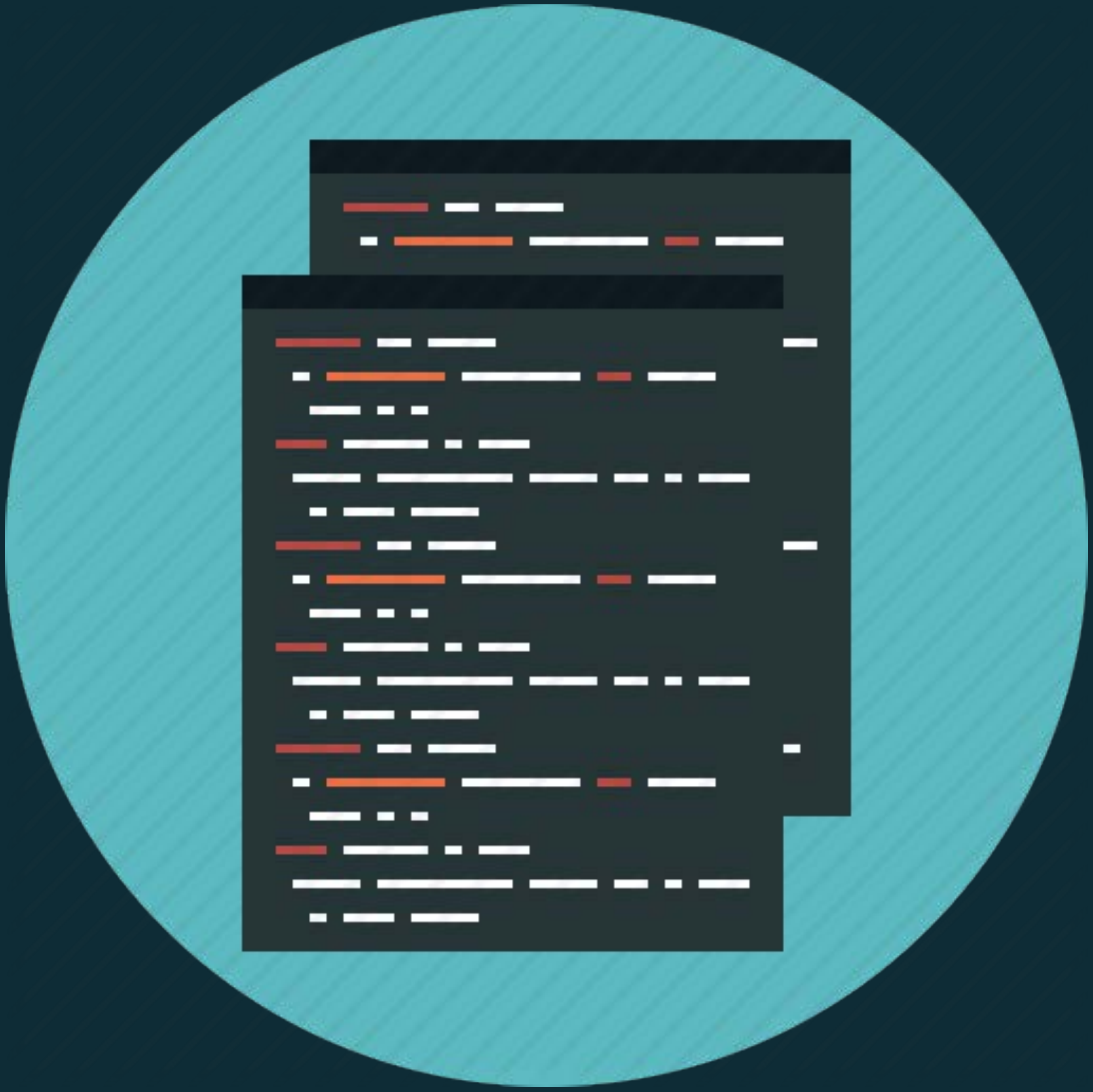
Magenta 211 54 130 #d33682

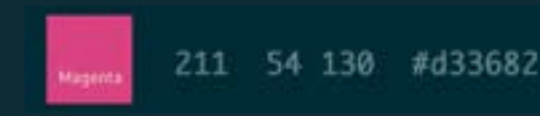
SCALE-FREE

Albert-László BY Barabási and Eric Bonabeau.
Scale-free Networks.
Scientific American, 5(5):50–59, 2003.

Yellow 181 137 0 #b58900

https://github.com/martinchapman/BuildX(.git)

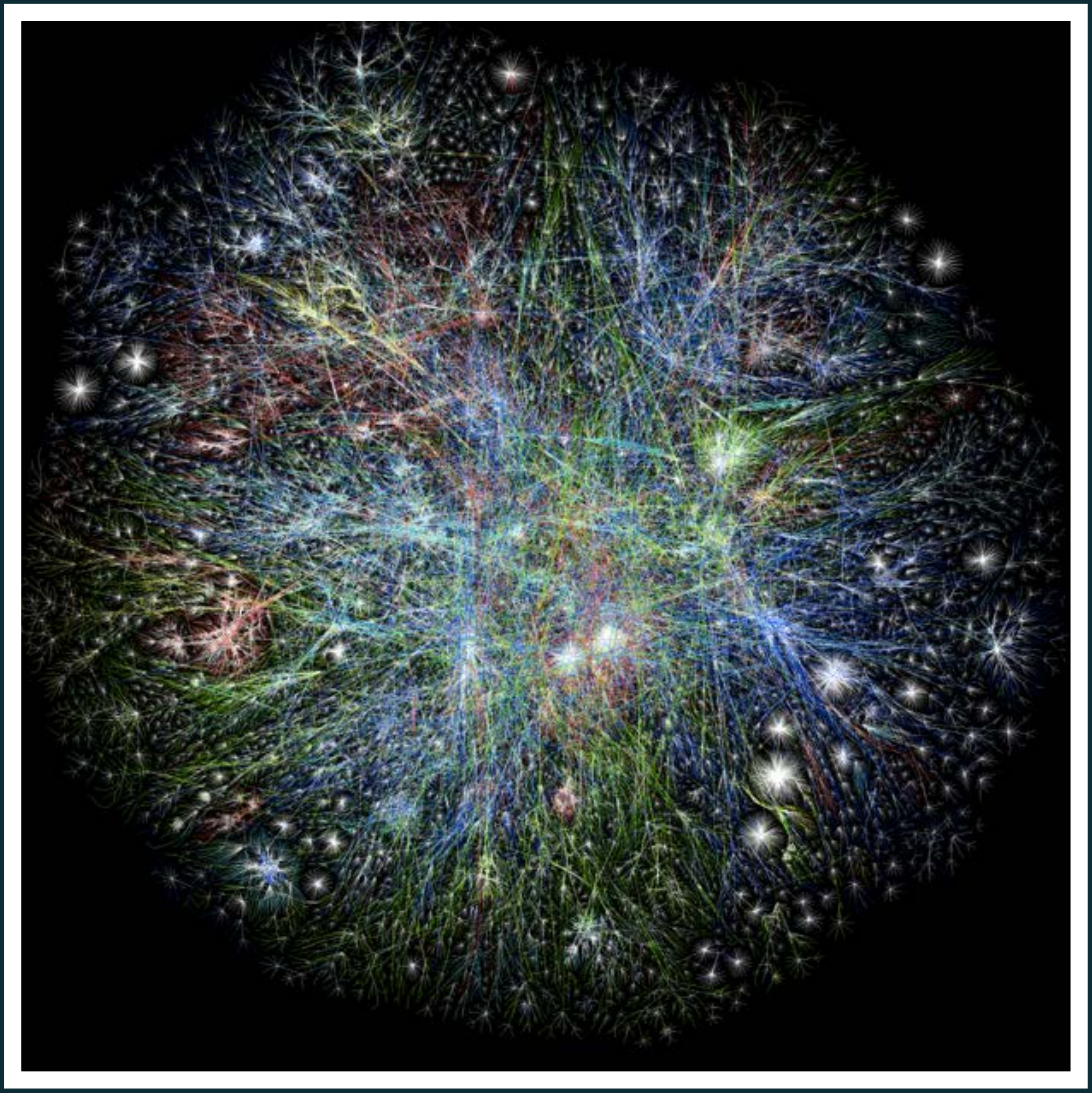




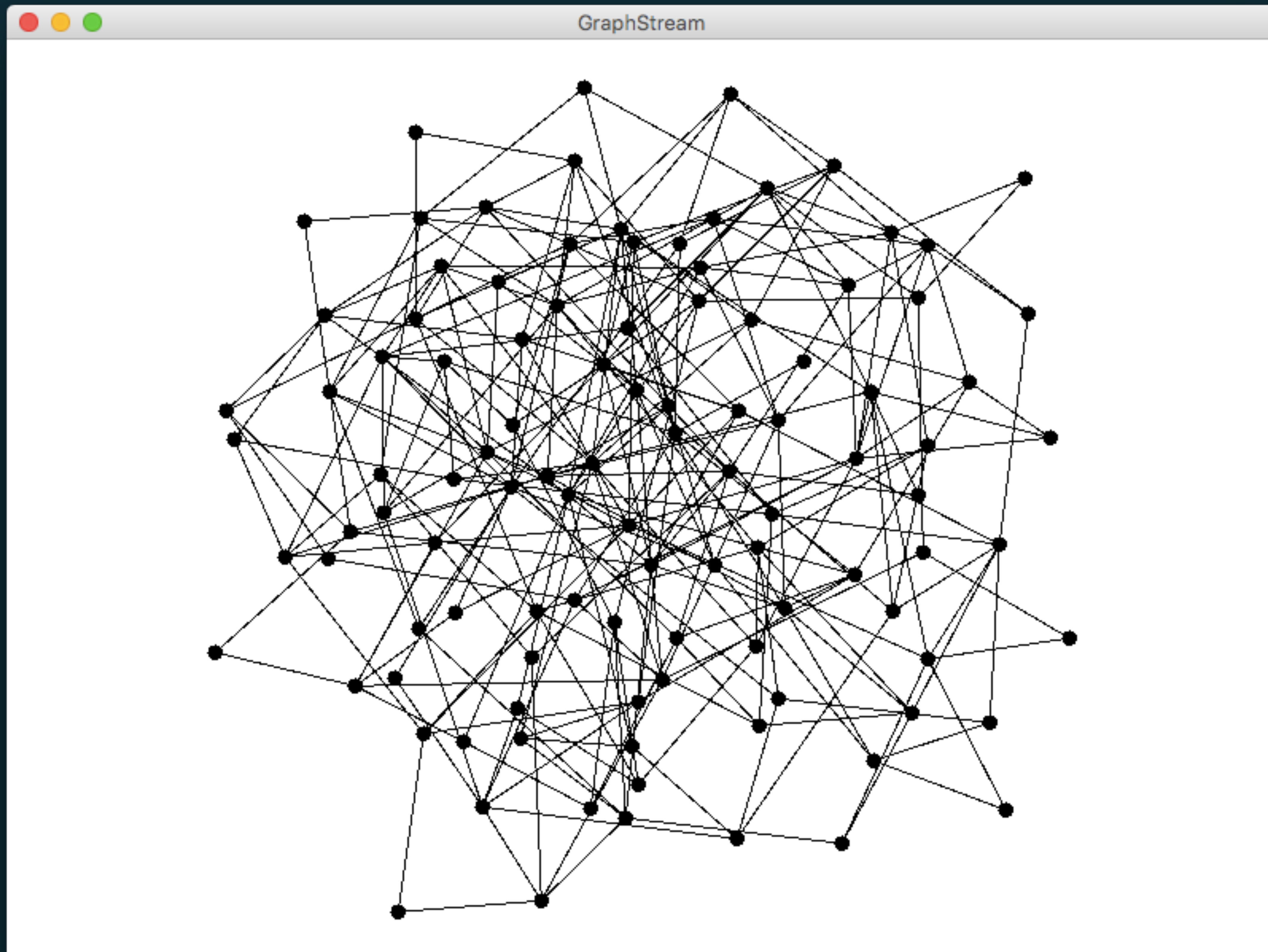
SCALE-FREE

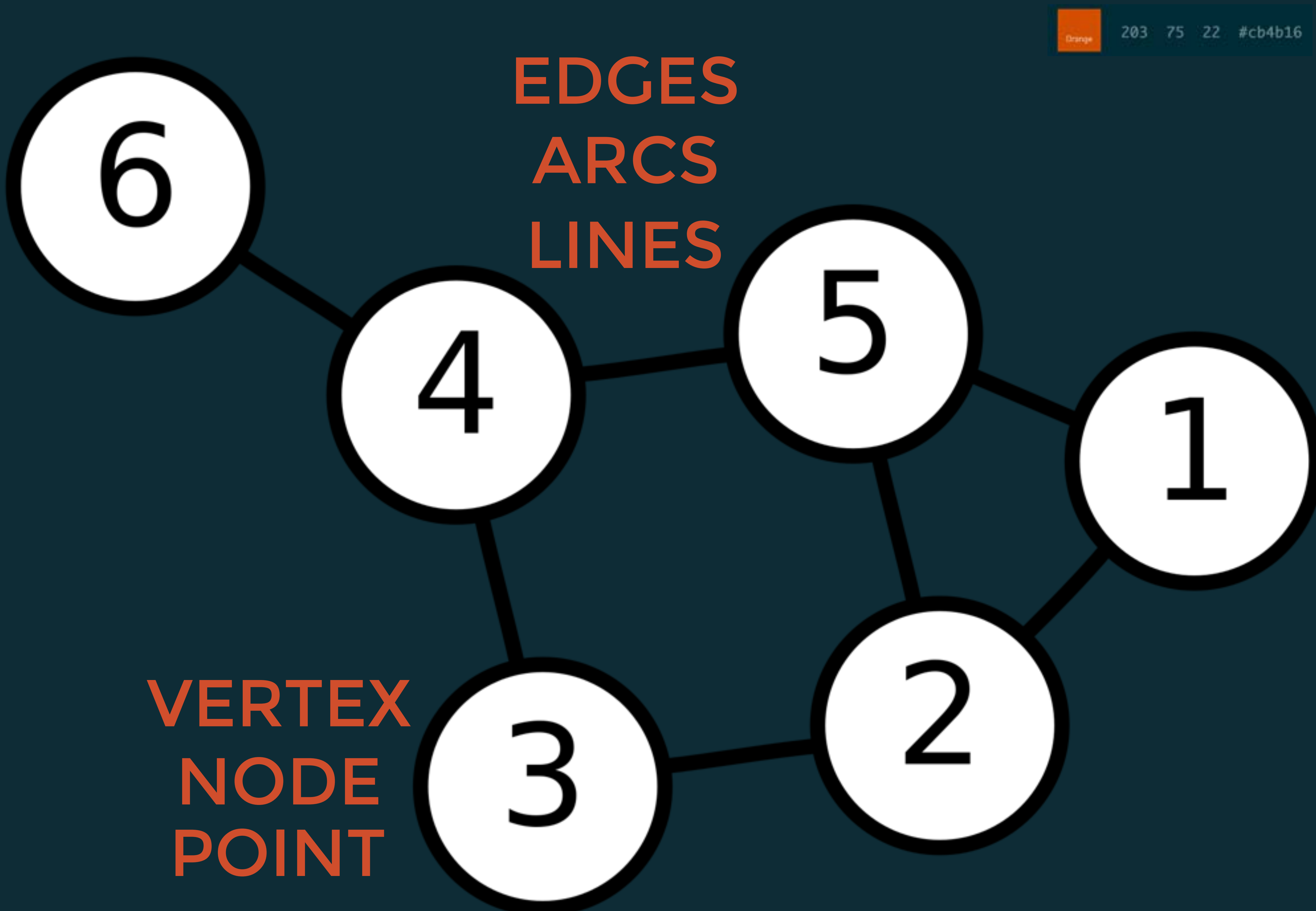
Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos.
On Power-law Relationships of the Internet Topology.
In Proceedings of
The 1999 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications
(SIGCOMM99), pages 251–262, 1999.

Yellow 181 137 0 #b58900



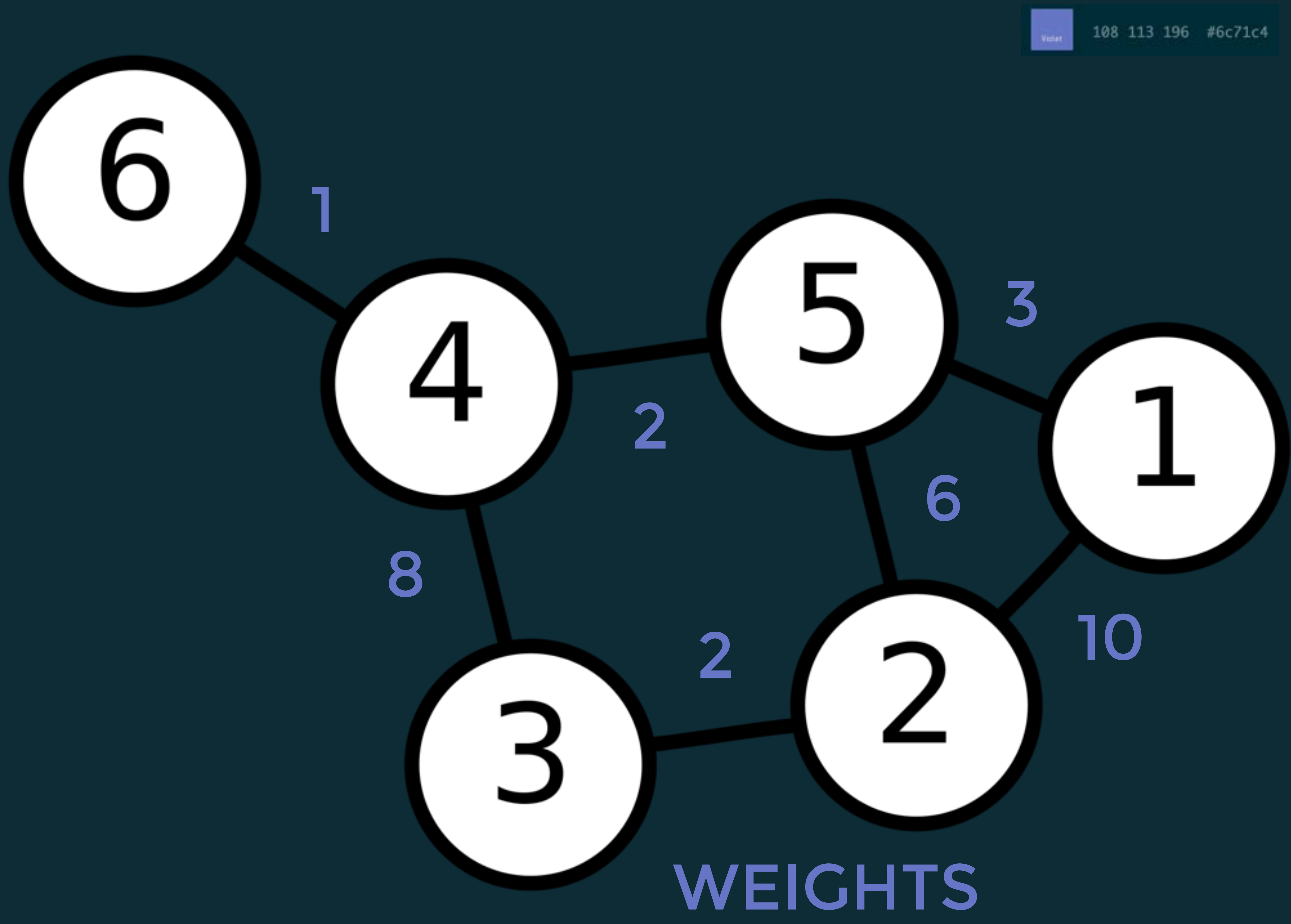
http://opte.org/maps/


GraphStream

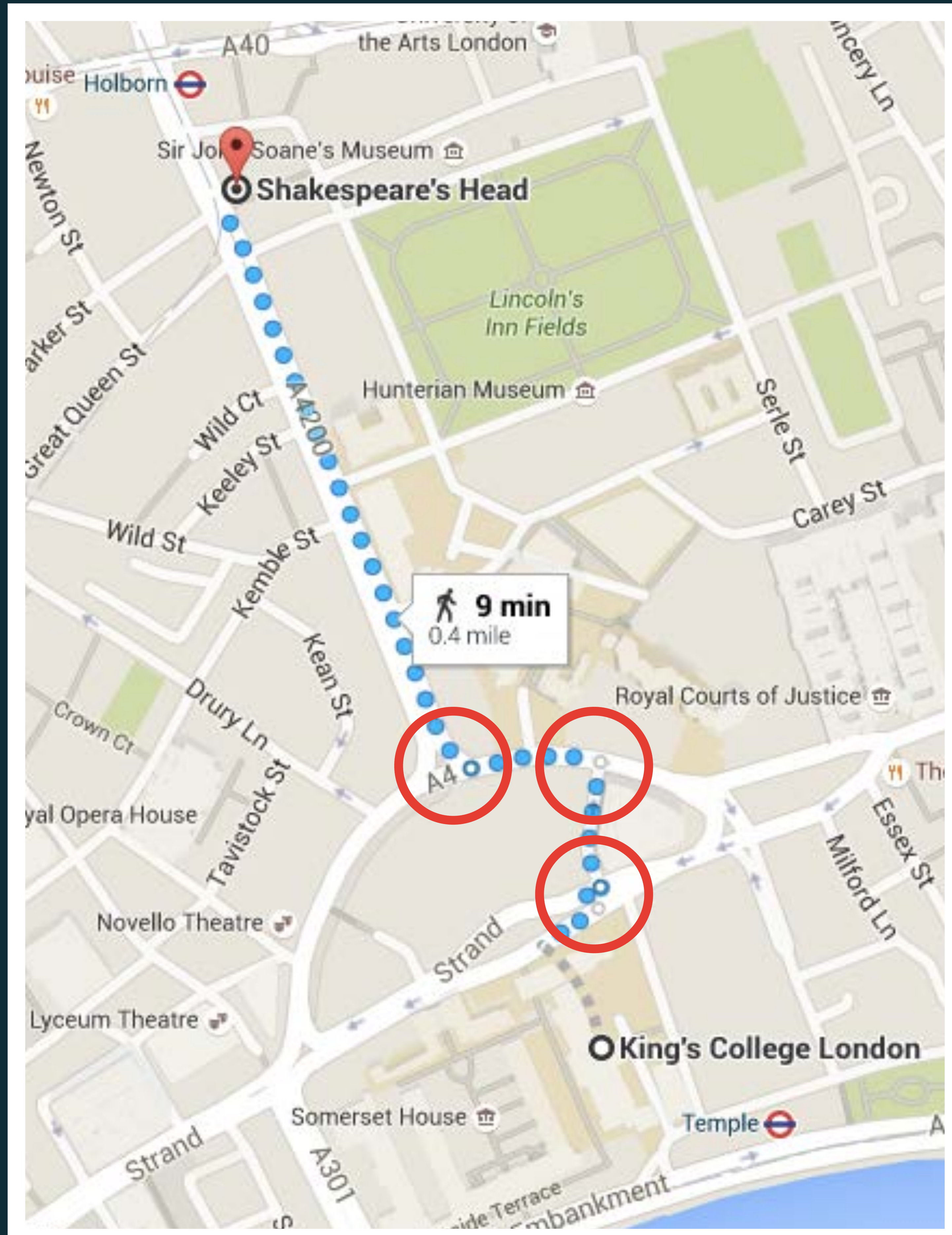

203 75 22 #cb4b16
EDGES
ARCS
LINES
6
4
5
1
2
3
VERTEX
NODE
POINT

Green 133 153 0 #859900

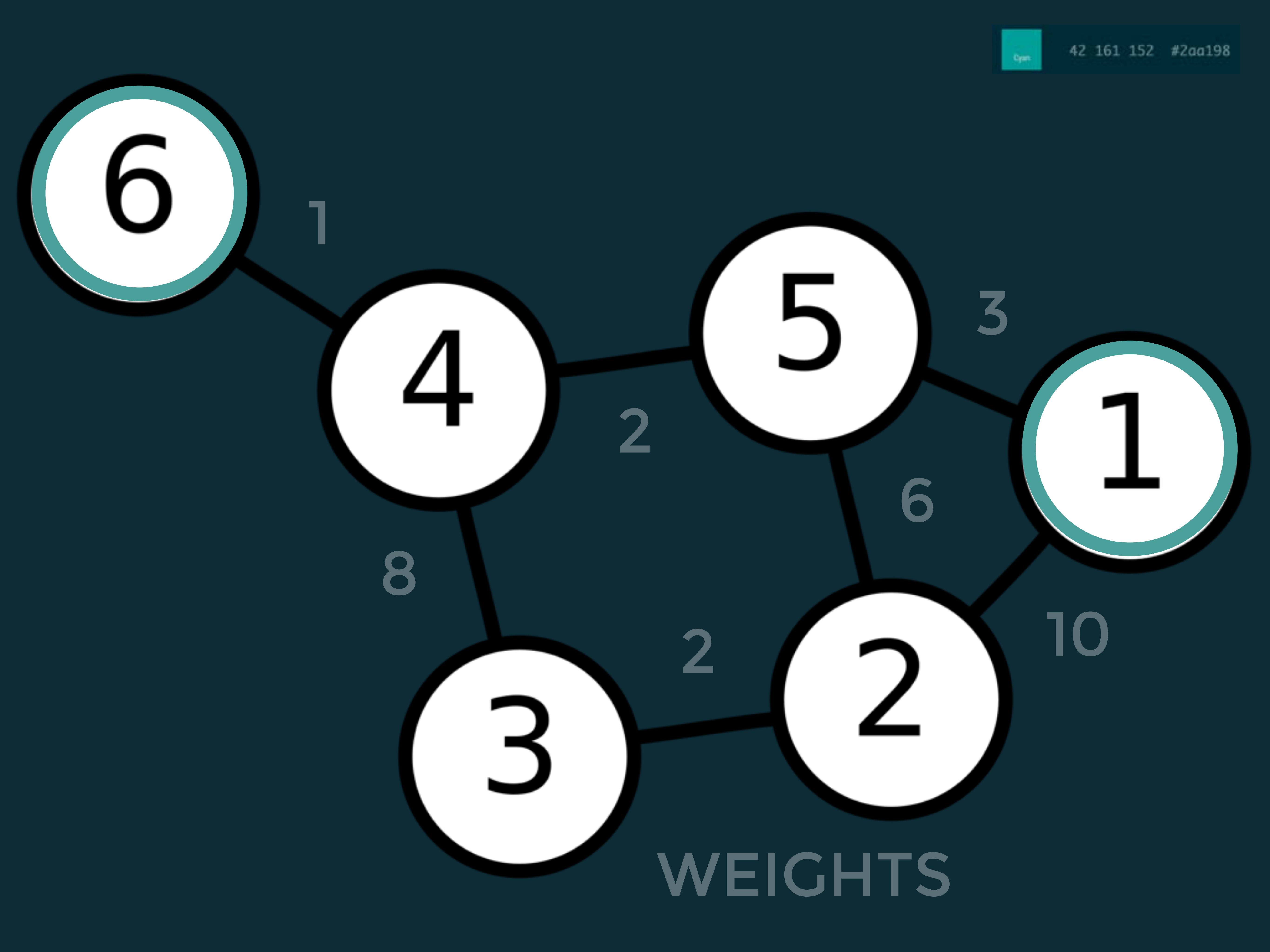
WHAT CAN WE DO WITH THIS?

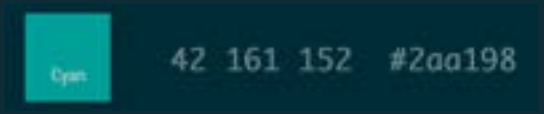

108 113 196 #6c71c4
6
1
4
5
3
1
2
6
8
10
2
3
2
WEIGHTS

220 50 47 #dc322f



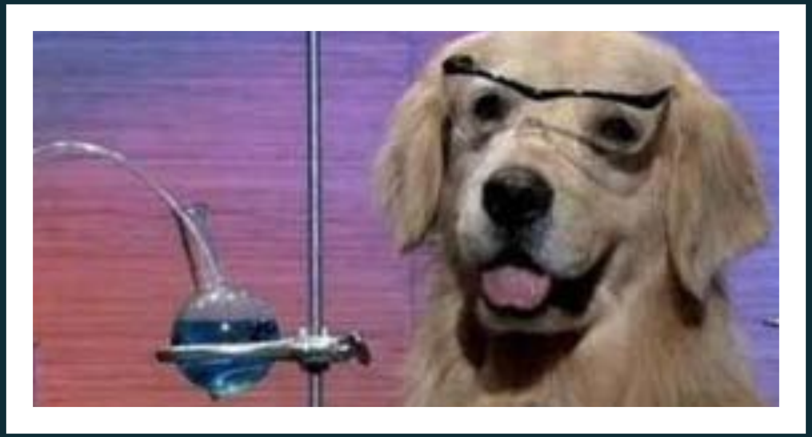
Decision points


42 161 152 #2aa198
6
1
4
2
5
3
1
6
8
2
2
10
3
WEIGHTS

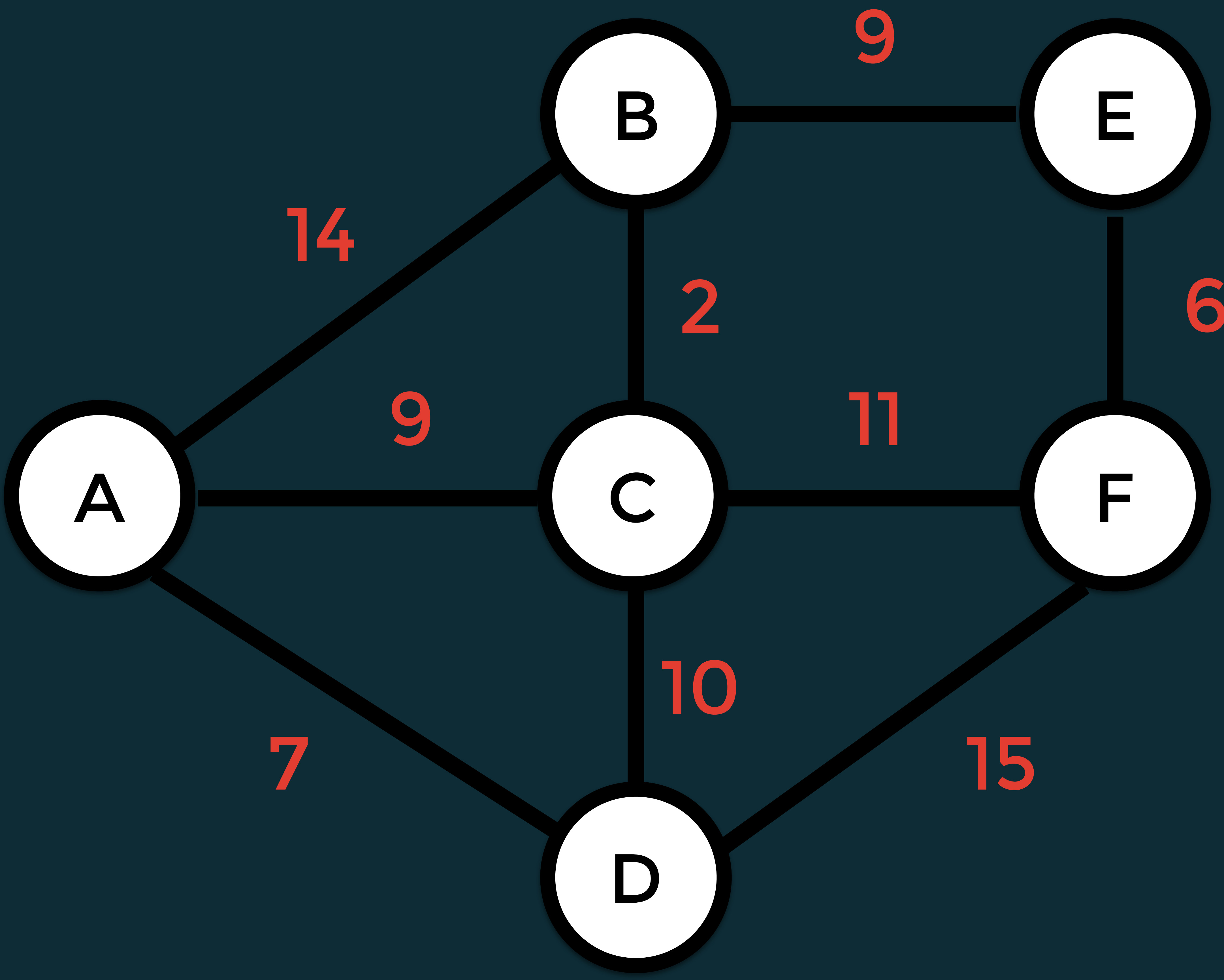


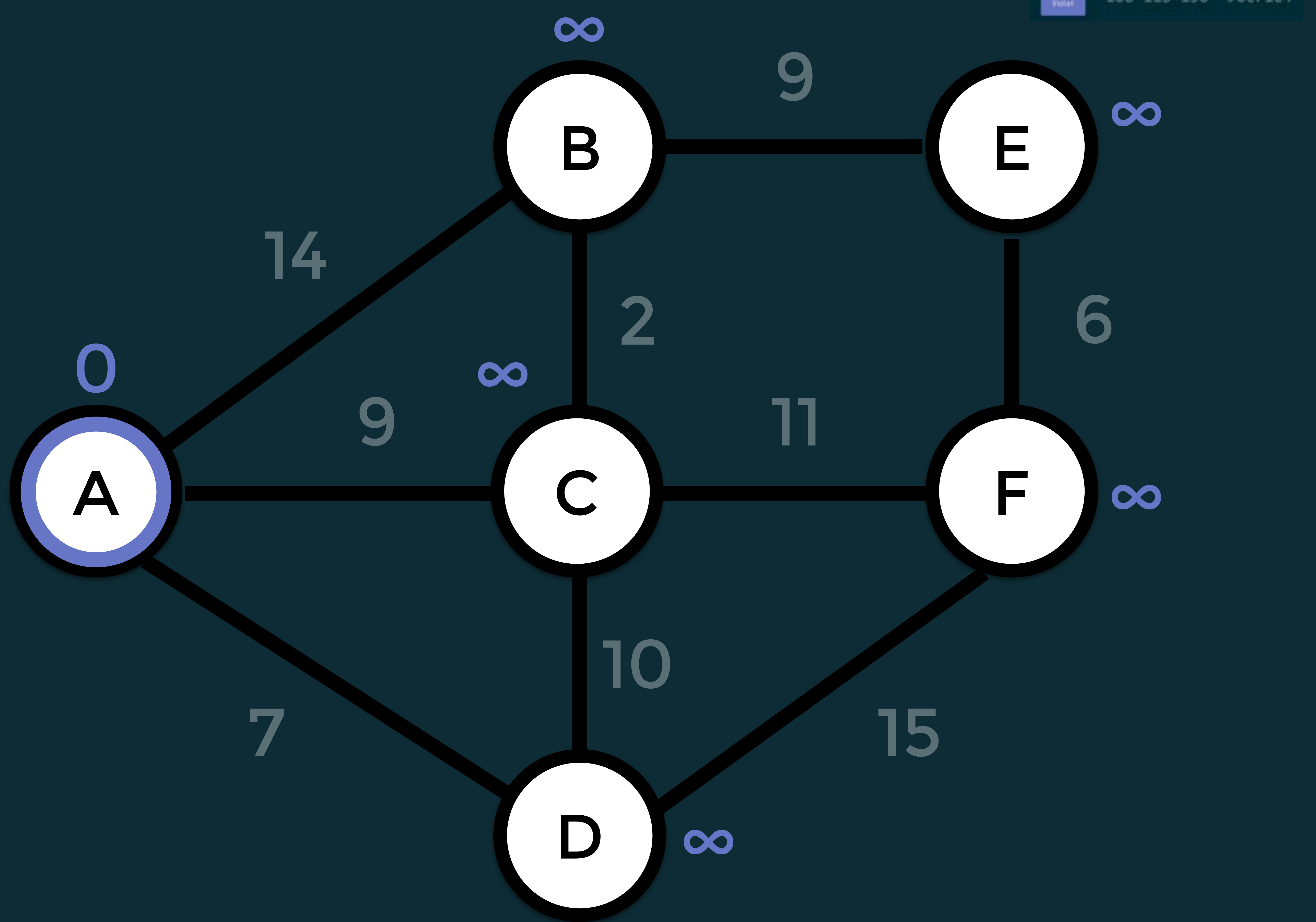
SHORTEST PATH

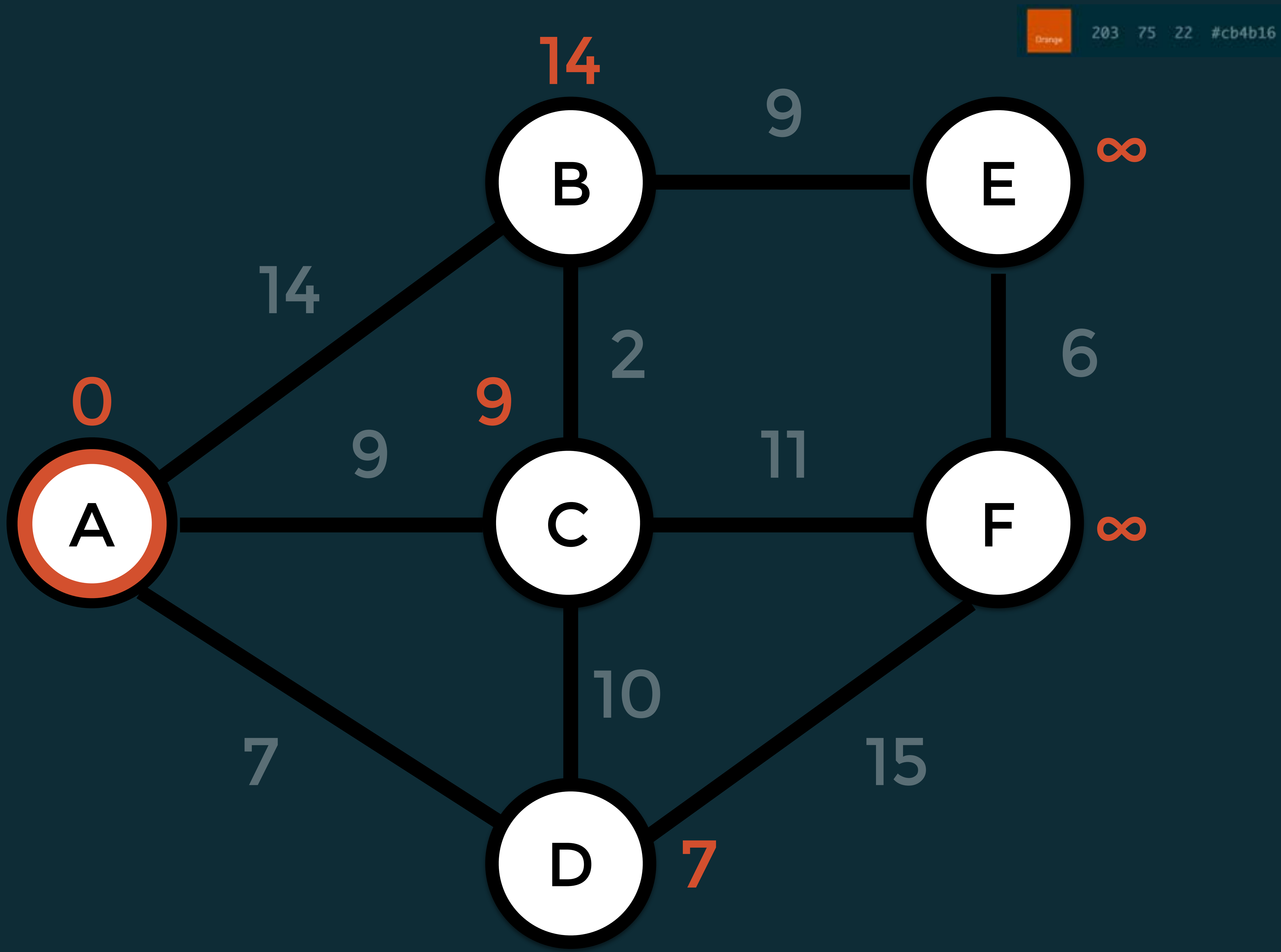
http://graphstream-project.org/doc/Algorithms/Shortest-path/



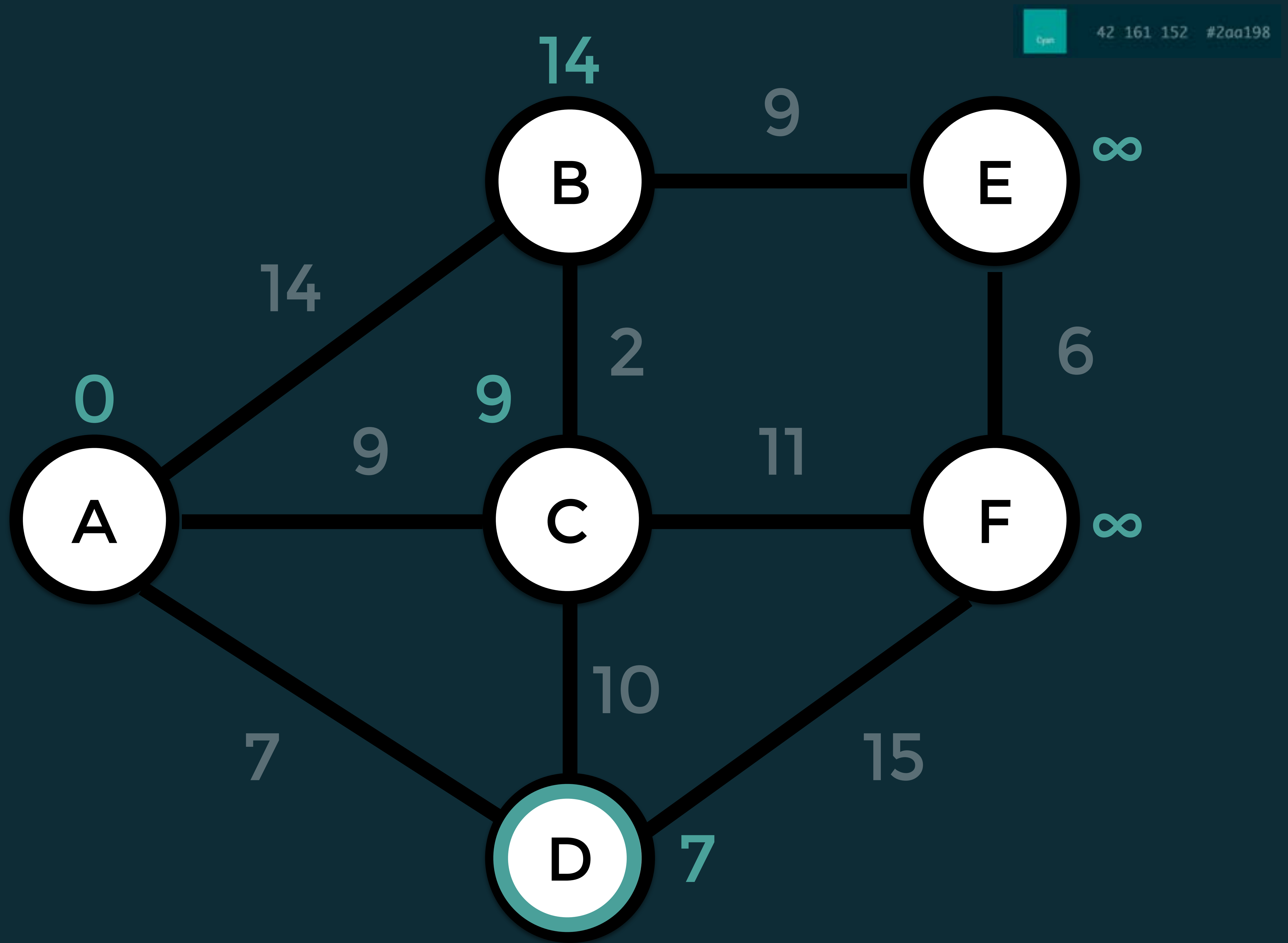

220 50 47 #dc322f



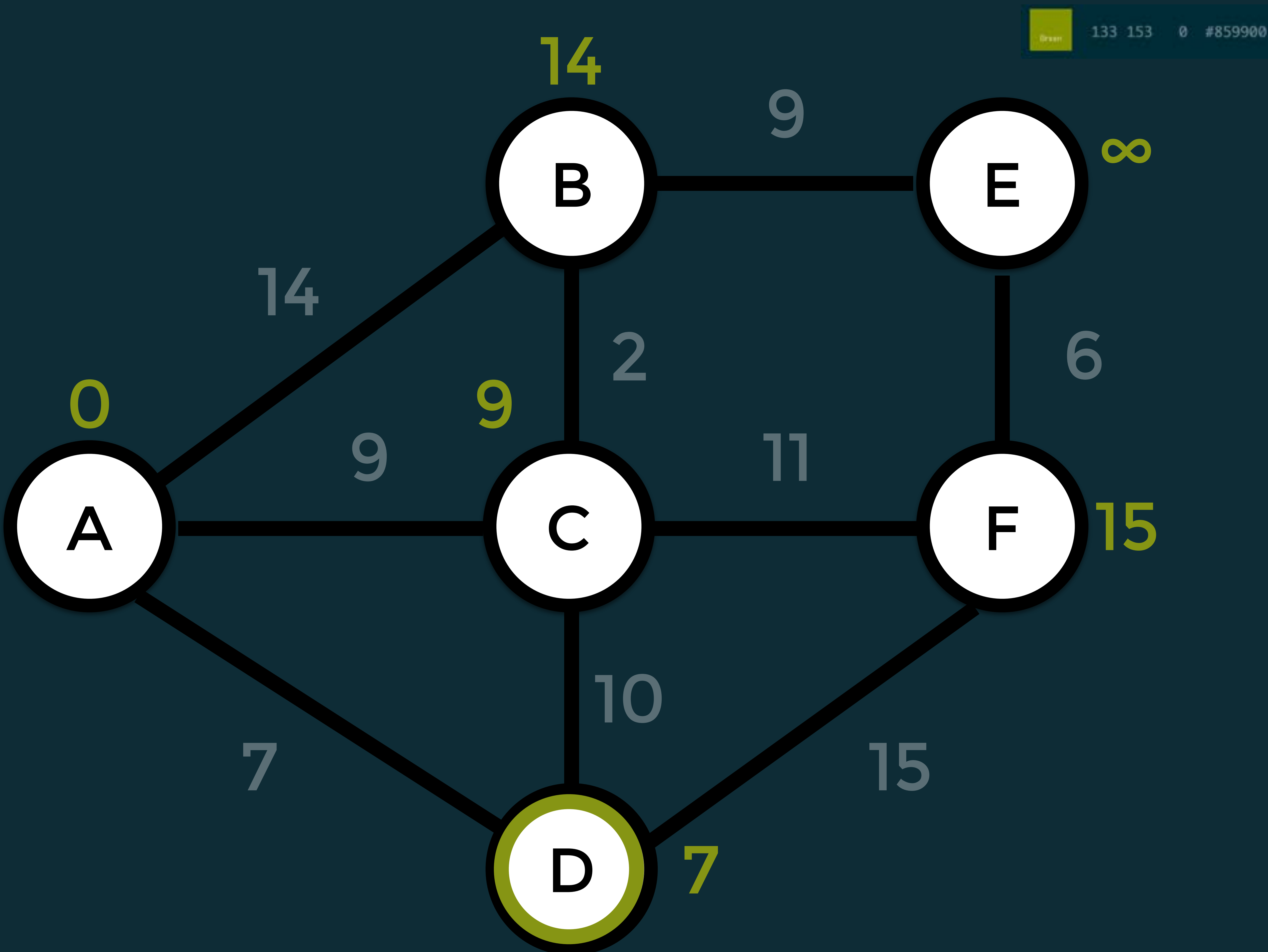

108 113 196 #6c71c4
∞
9
B
E
∞
14
2
6
0
∞
9
11
A
C
F
∞
10
7
15
D
∞



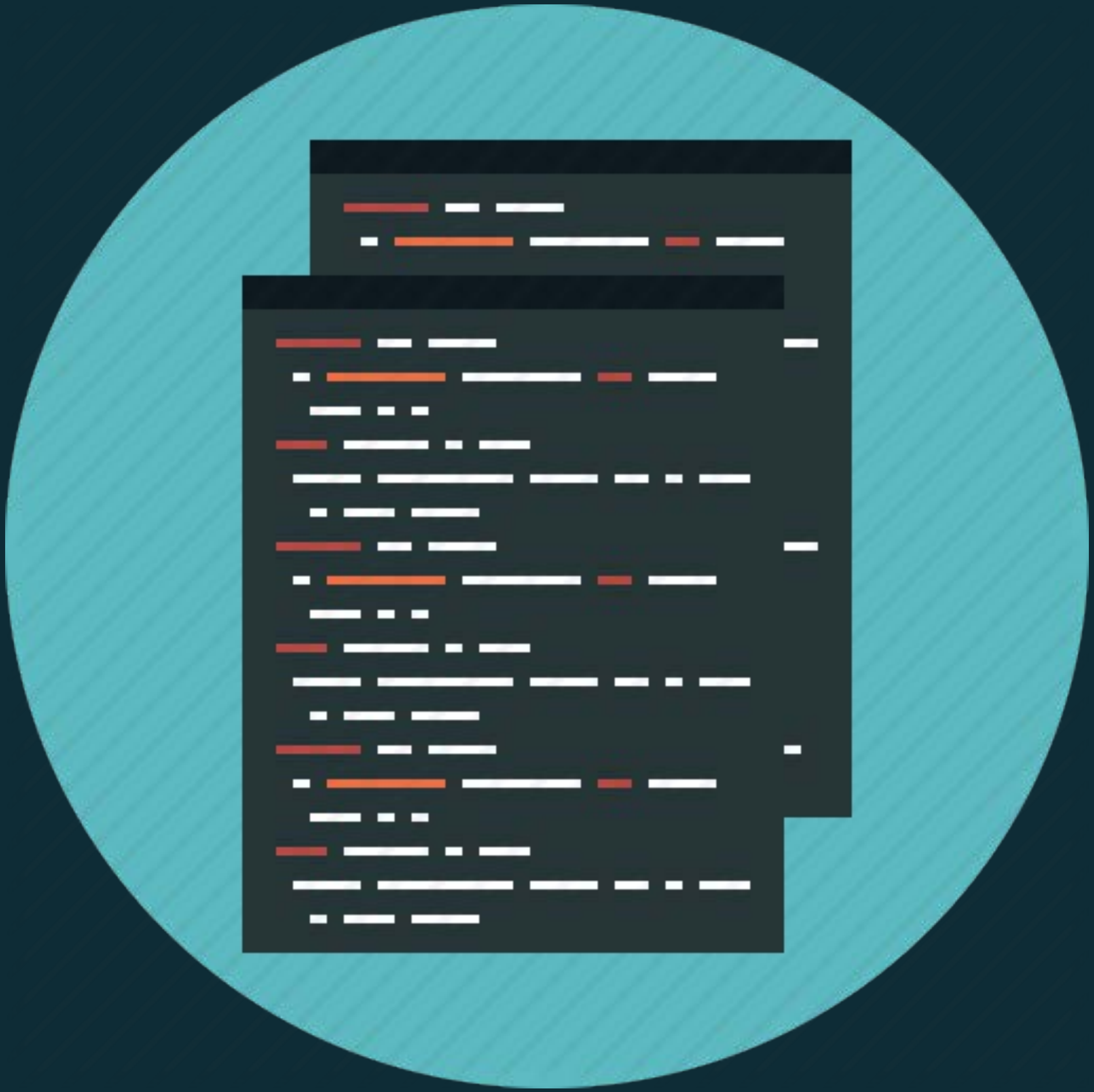
Orange
203 75 22 #cb4b16
14
B
9
E
∞
14
2
6
0
9
9
11
A
C
F
∞
10
7
15
D
7



42 161 152 #2aa198
14
B
9
E
∞
14
2
6
0
9
9
11
A
C
F
∞
10
7
15
D
7



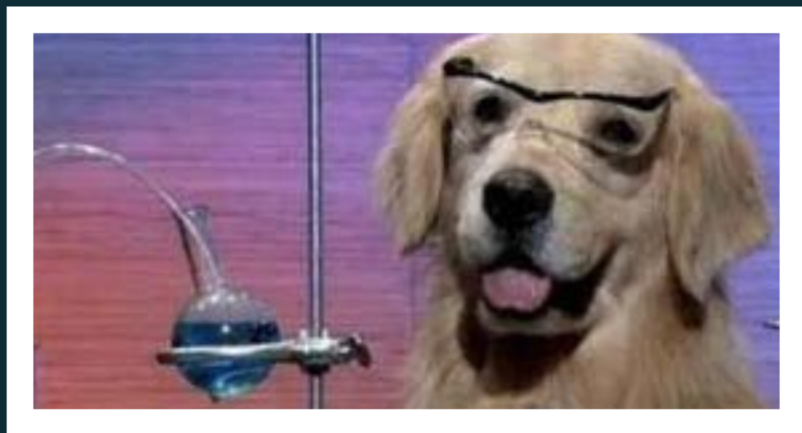
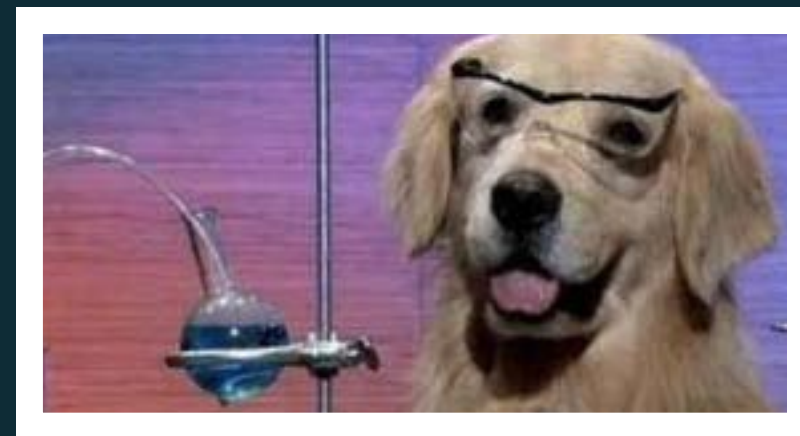
133 153 0 #859900
14
B
9
E
∞
14
2
6
0
9
9
11
A
C
F
15
10
7
15
D
7

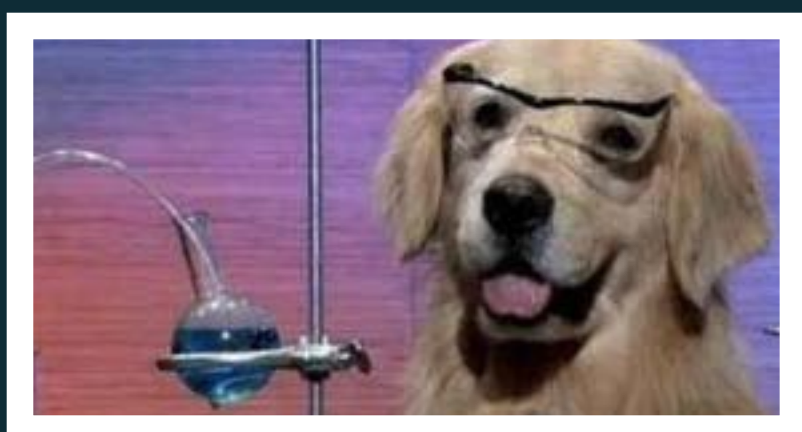
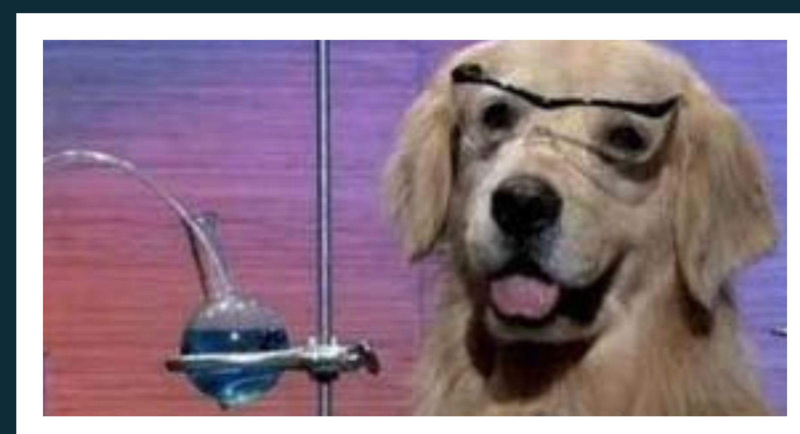


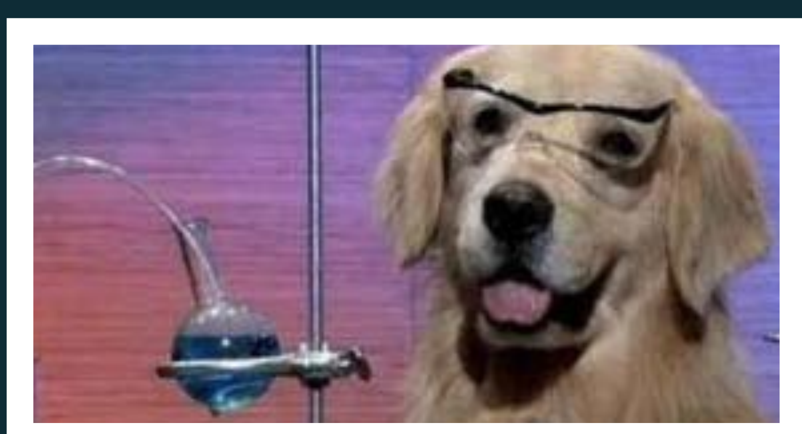
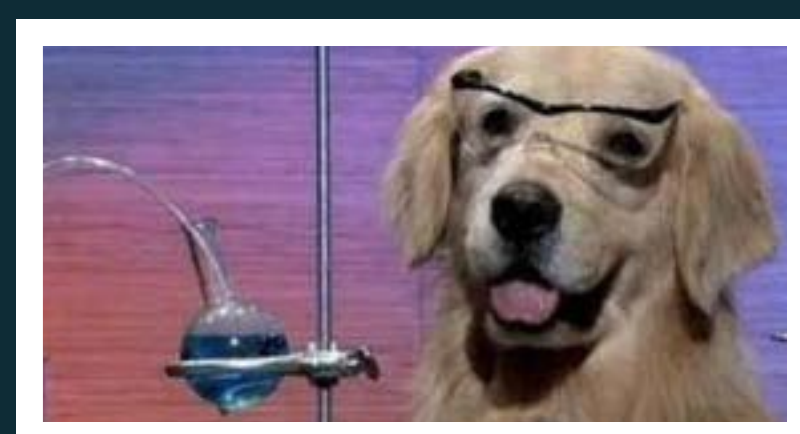
Violet 108 113 196 #6c71c4

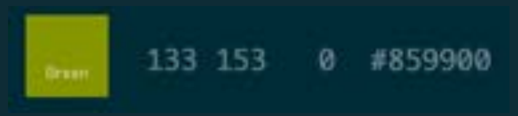
COMPLEXITY

Running time as a function of the input

$$(V \times \mathrm{LOG}\ V) + E$$

Many different permutations on this

https://www.cs.cornell.edu/courses/cs312/2002sp/lectures/lec20/lec20.htm

Blue 38 139 210 #268bd2

$$(V \times \log V) + E$$

```
(* Dijkstra's Algorithm *)
let val q: queue = new_queue()
  val visited: vertexMap = create_vertexMap()
  fun expand(v: vertex) =
    let val neighbors: vertex list = Graph.outgoing(v)
        val dist: int = valOf(get(visited, v))
        fun handle_edge(v': vertex, weight: int) =
          case get(visited, v') of
            SOME(d') =>
              if dist+weight < d'
              then ( add(visited, v', dist+weight);
                     incr_priority(q, v', dist+weight) )
              else ()
          | NONE => ( add(visited, v', dist+weight);
                      push(q, v', dist+weight) )
    in
      app handle_edge neighbors
    end
in
  add(visited, v0, 0);
  expand(v0);
  while (not (empty_queue(q)) do expand(pop(q))
end
```

```
(* Dijkstra's Algorithm *)
let val q: queue = new_queue()
  val visited: vertexMap = create_vertexMap()
  fun expand(v: vertex) =
    let val neighbors: vertex list = Graph.outgoing(v)
        val dist: int = valOf(get(visited, v))
        fun handle_edge(v': vertex, weight: int) =
          case get(visited, v') of
            SOME(d') =>
              if dist+weight < d'
              then ( add(visited, v', dist+weight);
                     incr_priority(q, v', dist+weight) )
              else ()
          | NONE => ( add(visited, v', dist+weight);
                      push(q, v', dist+weight) )
    in
      app handle_edge neighbors
    end
in
  add(visited, v0, 0);
  expand(v0);
  while (not (empty_queue(q)) do expand(pop(q))
end
```

Cyan 42 161 152 #2aa198

```
// Always deal with the next closest node first (via the estimate)
Node node = getClosestFromEstimate( unsettledNodes );
```

```
    let val neighbors: vertex list = Graph.outgoing(v)
```

```
if ( getDistanceEstimate( node ) < getDistanceEstimate( minimum ) ) {

    minimum = node;

}
```

```
              else ()
            | NONE => ( add(visited, v', dist+weight);
                        push(q, v', dist+weight) )
    in
      app handle_edge neighbors
    end
in
  add(visited, v0, 0);
  expand(v0);
  while (not (empty_queue(q)) do expand(pop(q))
end
```

So, in reality, our intuitive Java implementation is likely to differ in complexity.

220 50 47 #dc322f

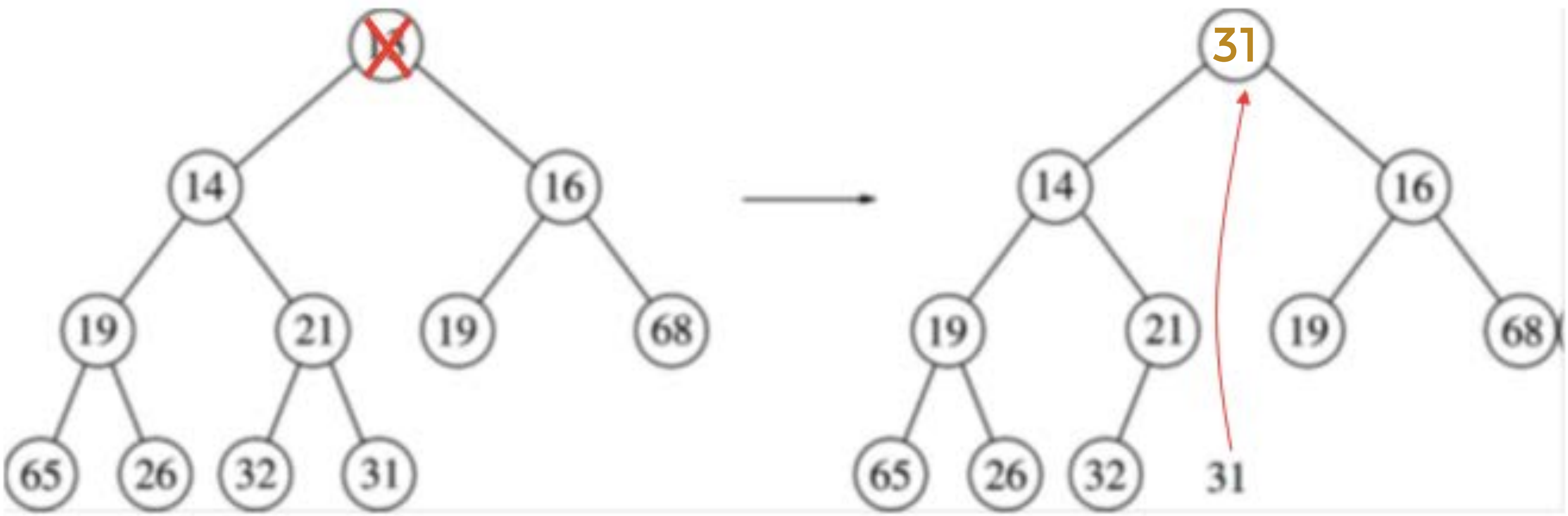
$$(V \times \log V) + E$$

Reshuffling after a pop

181 137 0 #b58900

Remove value at the root

Move 31 (last element) to the root



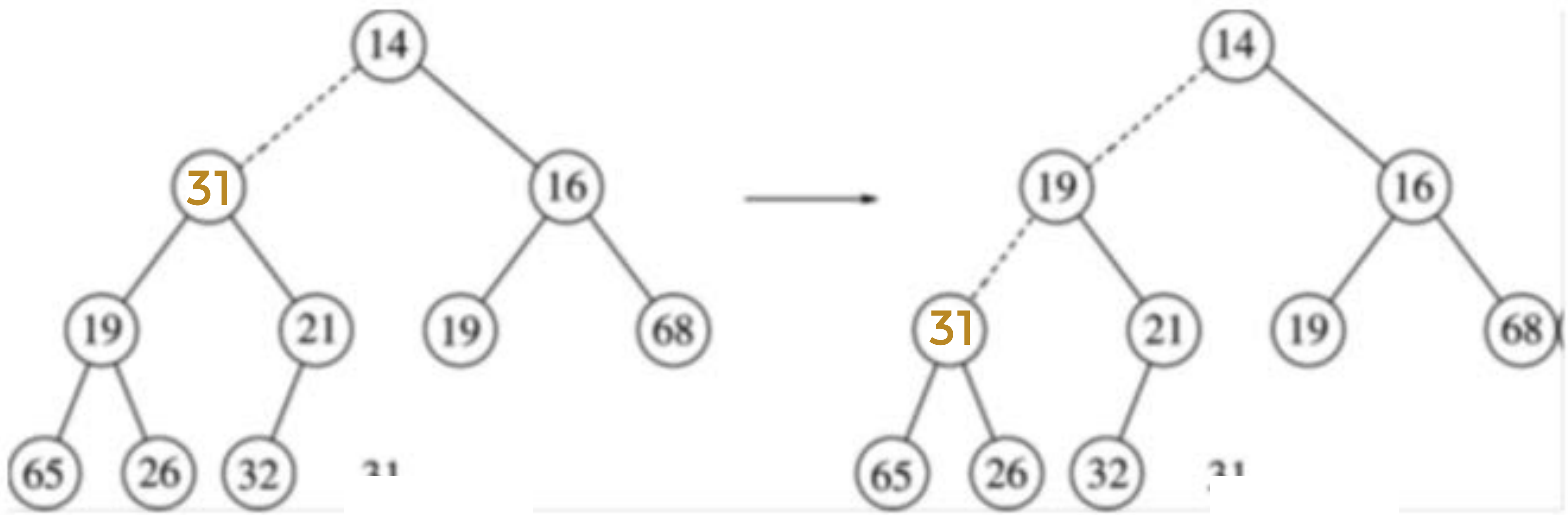
Is 31 > min(14, 16)?
If yes, swap 31 with min(14, 16)
If no, leave 31 in hole

Value in child is always greater than parent, and this is retained.

I assume the initial removal and movement is an atomic action.

Yellow 181 137 0 #b58900

Percolating down…

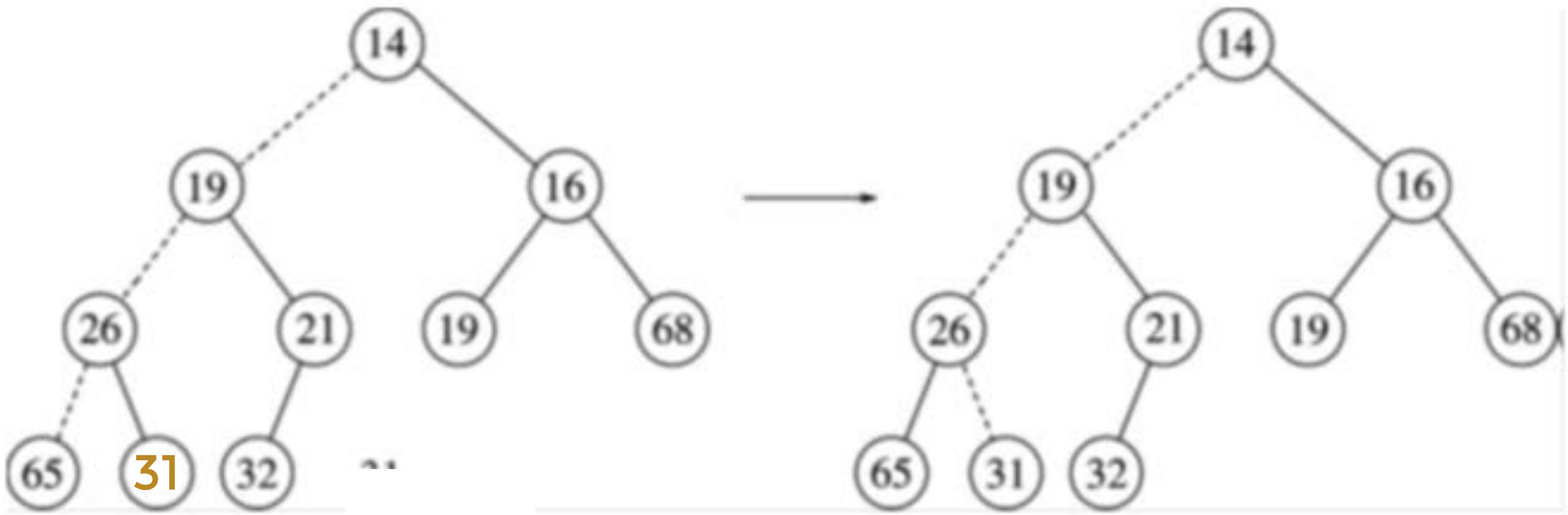


Is 31 > min(19, 21)? 1 swap
If yes, swap 31 with min(19, 21)
If no, leave 31 in hole

Is 31 > min(65, 26)? 2 swaps
If yes, swap 31 with min(65, 26)
If no, leave 31 in hole

181 137 0 #b58900

Percolating down...



3 swaps

Heap-order property okay;
Structure okay;
Done.

Magenta 211 54 130 #d33682

$V = 11$

Magenta 211 54 130 #d33682

V = 11

11 nodes = _ levels?

Magenta 211 54 130 #d33682

$V = 11$

11 nodes = _ levels?

$\text{LOG } V = \text{LOG}(2)\ 11 = 3.4$

Magenta 211 54 130 #d33682

V = 11

11 nodes = _ levels?

LOG V = LOG(2) 11 = 3.4

WORST CASE =
3 SWAPS

Green 133 153 0 #859900

$$(V \times \text{LOG } V) + E$$

Popping every node in the graph

Violet 108 113 196 #6c71c4

$$(V \times \mathrm{LOG}\ V) + E$$

Updating distance estimates for everyone’s neighbours

```
(* Dijkstra's Algorithm *)
let val q: queue = new_queue()
  val visited: vertexMap = create_vertexMap()
  fun expand(v: vertex) =
    let val neighbors: vertex list = Graph.outgoing(v)
        val dist: int = valOf(get(visited, v))
        fun handle_edge(v': vertex, weight: int) =
          case get(visited, v') of
            SOME(d') =>
              if dist+weight < d'
              then ( add(visited, v', dist+weight);
                     incr_priority(q, v', dist+weight) )
              else ()
          | NONE => ( add(visited, v', dist+weight);
```

```
// Update the estimate to the neighbour, based on going through this node
distanceEstimate.put(neighbour, getDistanceEstimate(node) + getDistance(node, neighbour));
```

```
    end
in
  add(visited, v0, 0);
  expand(v0);
  while (not (empty_queue(q)) do expand(pop(q))
end
```

220 50 47 #dc322f

BELLMAN-FORD

FLOYD–WARSHALL

A*

220 50 47 #dc322f

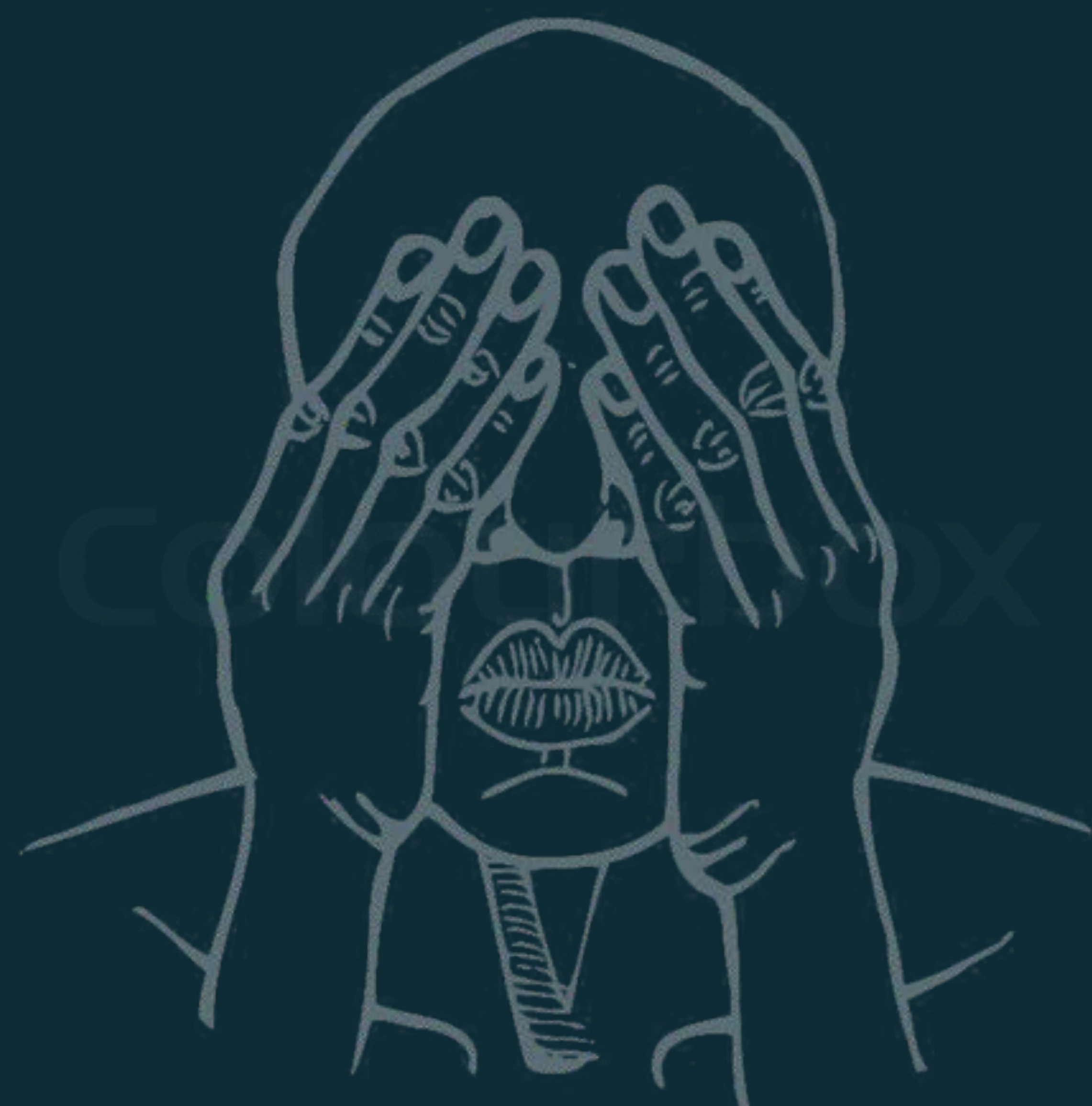
BELLMAN-FORD

FLOYD–WARSHALL

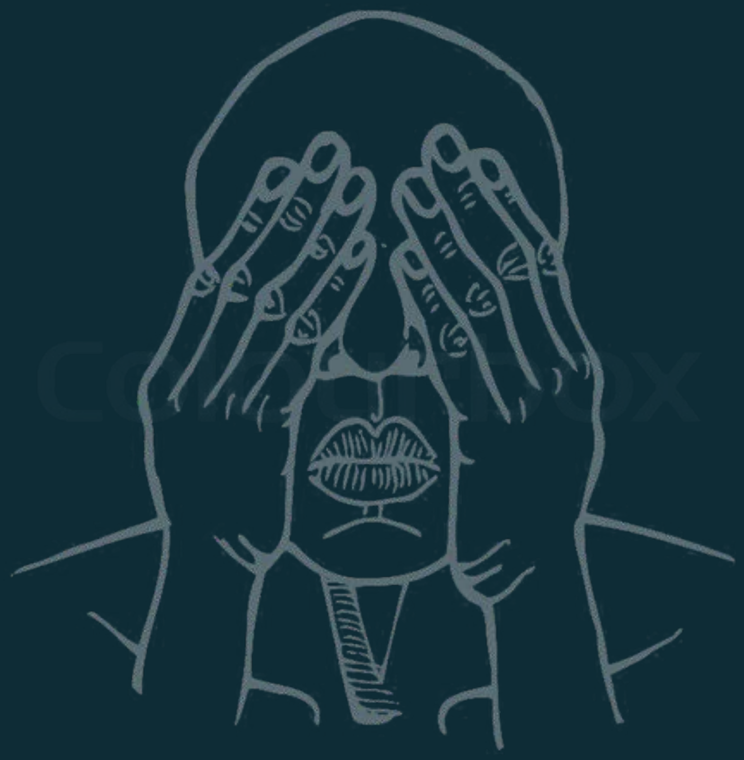
A*

Orange 203 75 22 #cb4b16

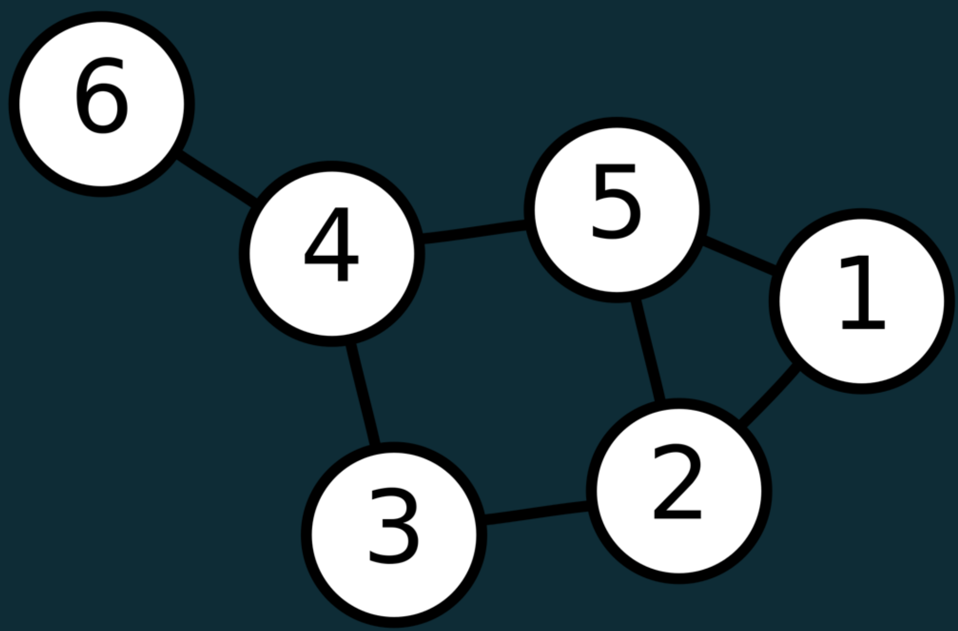
LOOKING FOR HIDDEN THINGS



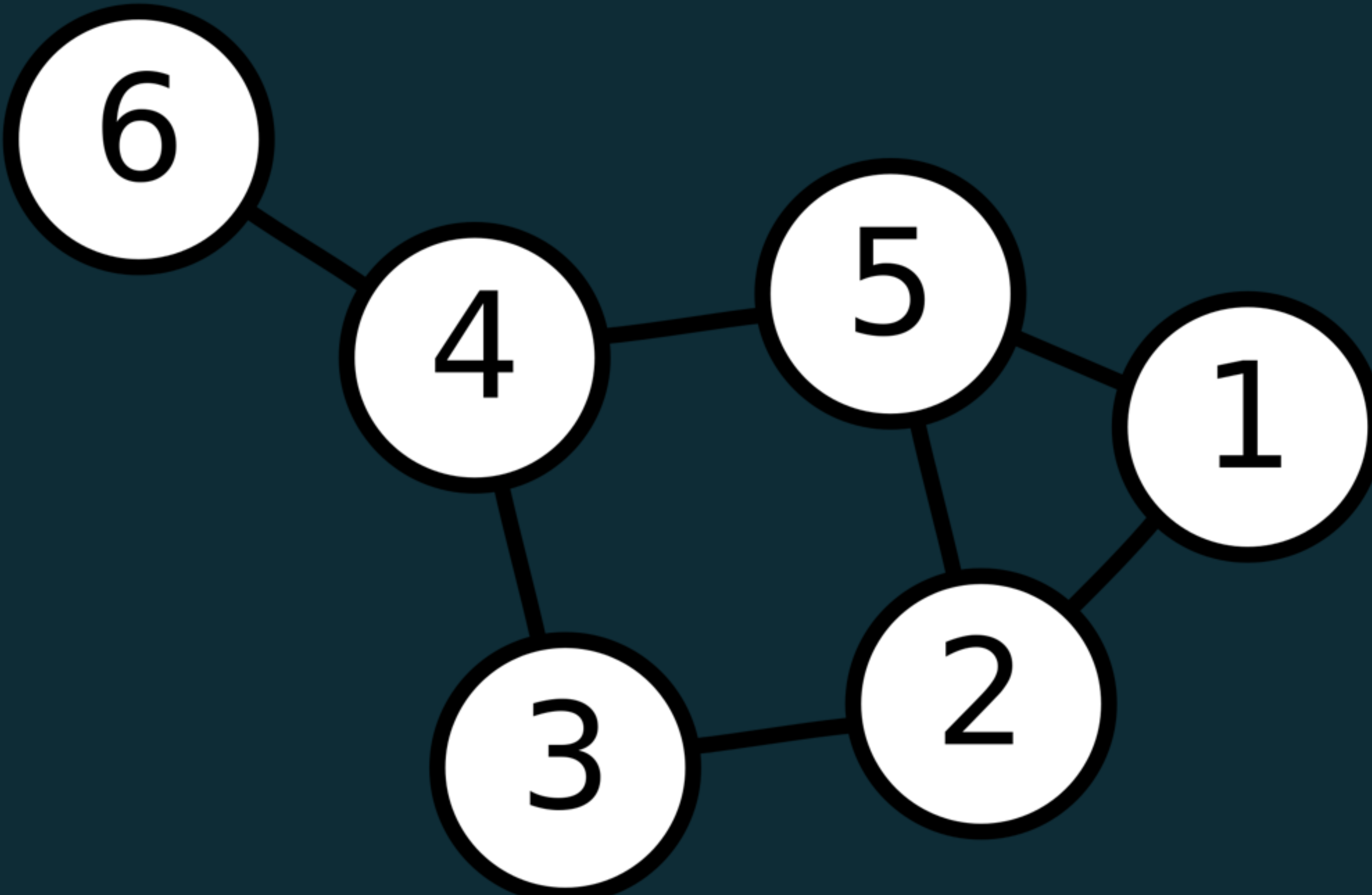
108 113 196 #6c71c4



+

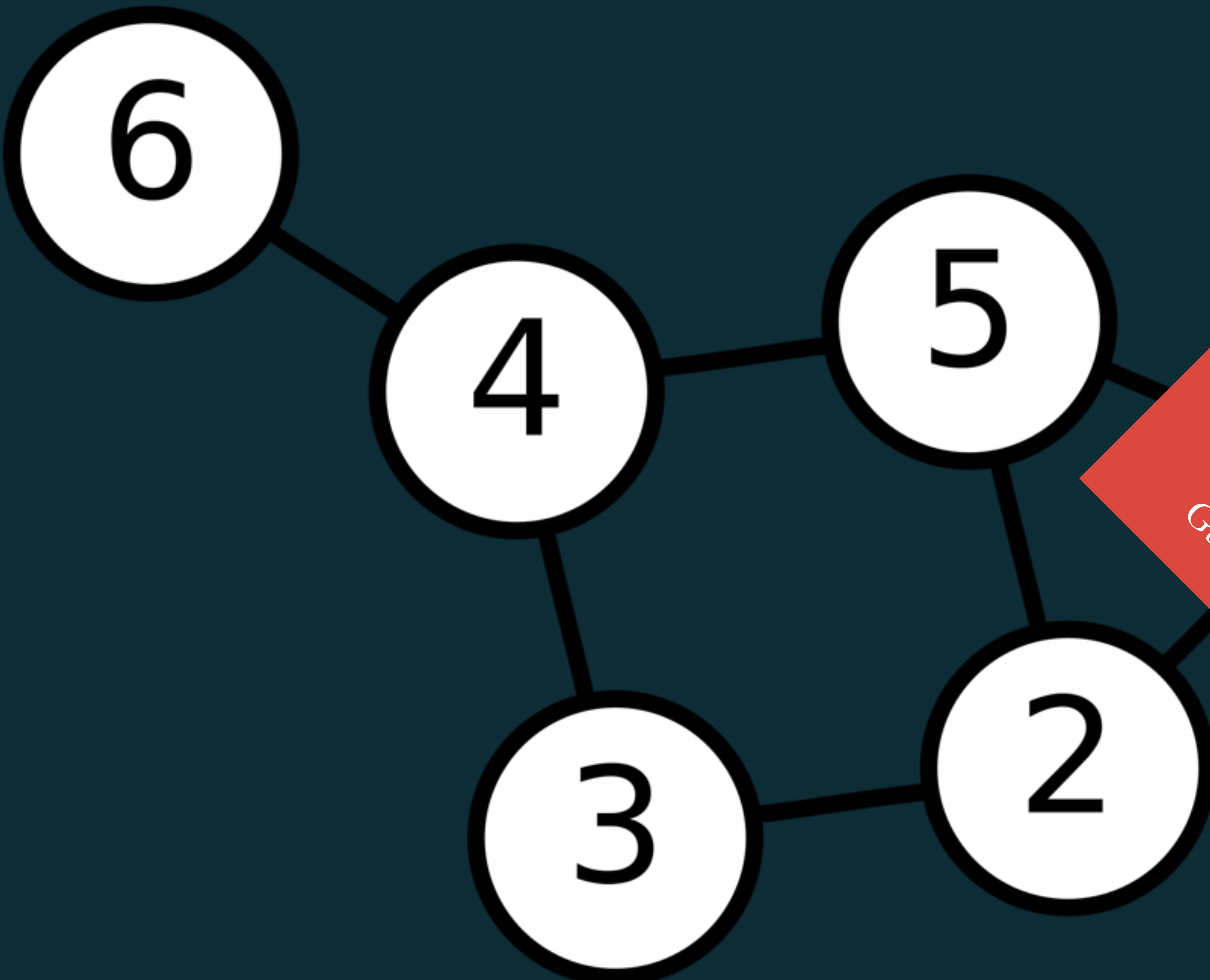


=

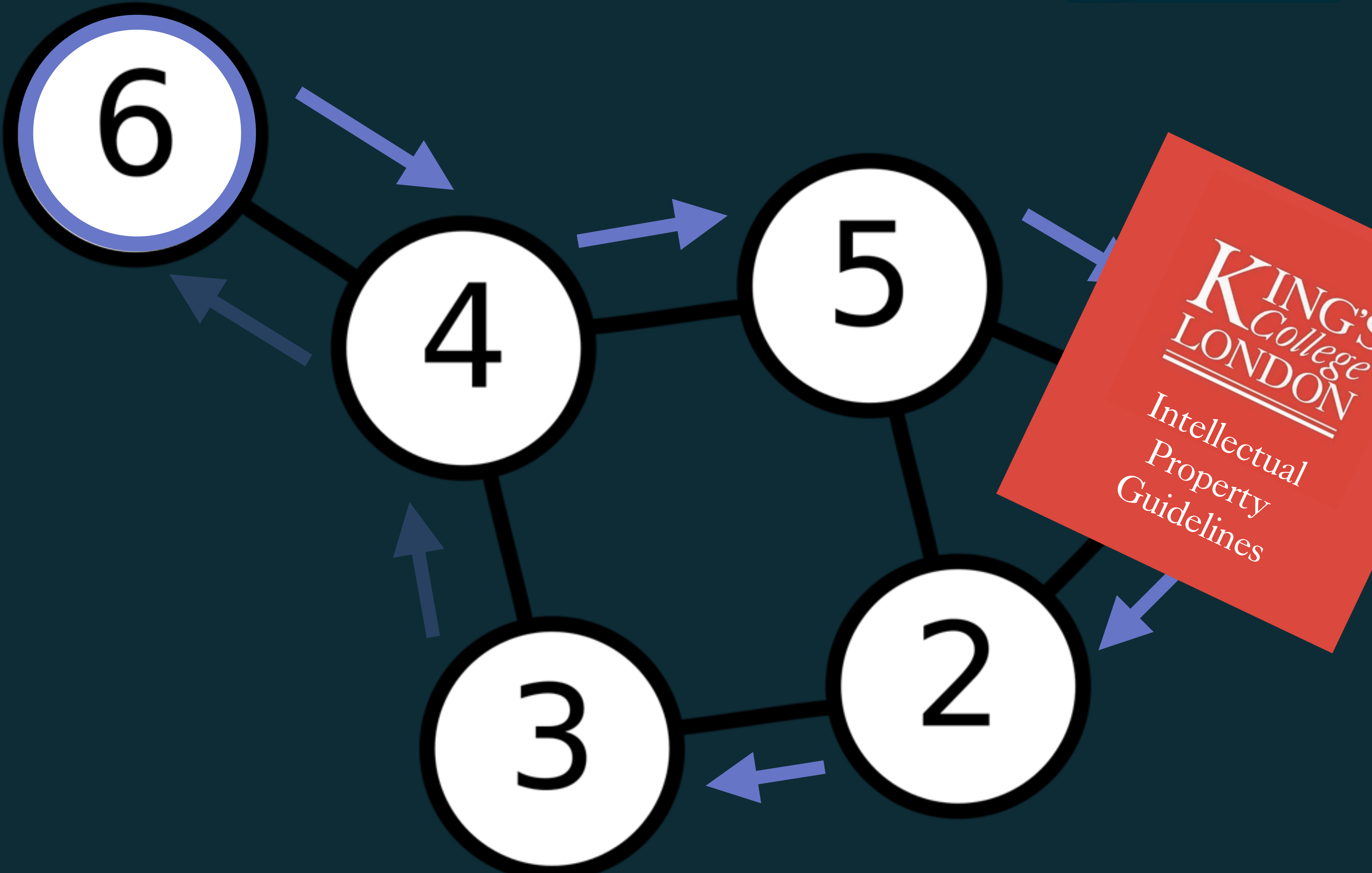

6
4
5
1
3
2

KING'S College LONDON

Intellectual Property Guidelines


6
4
5
3
2

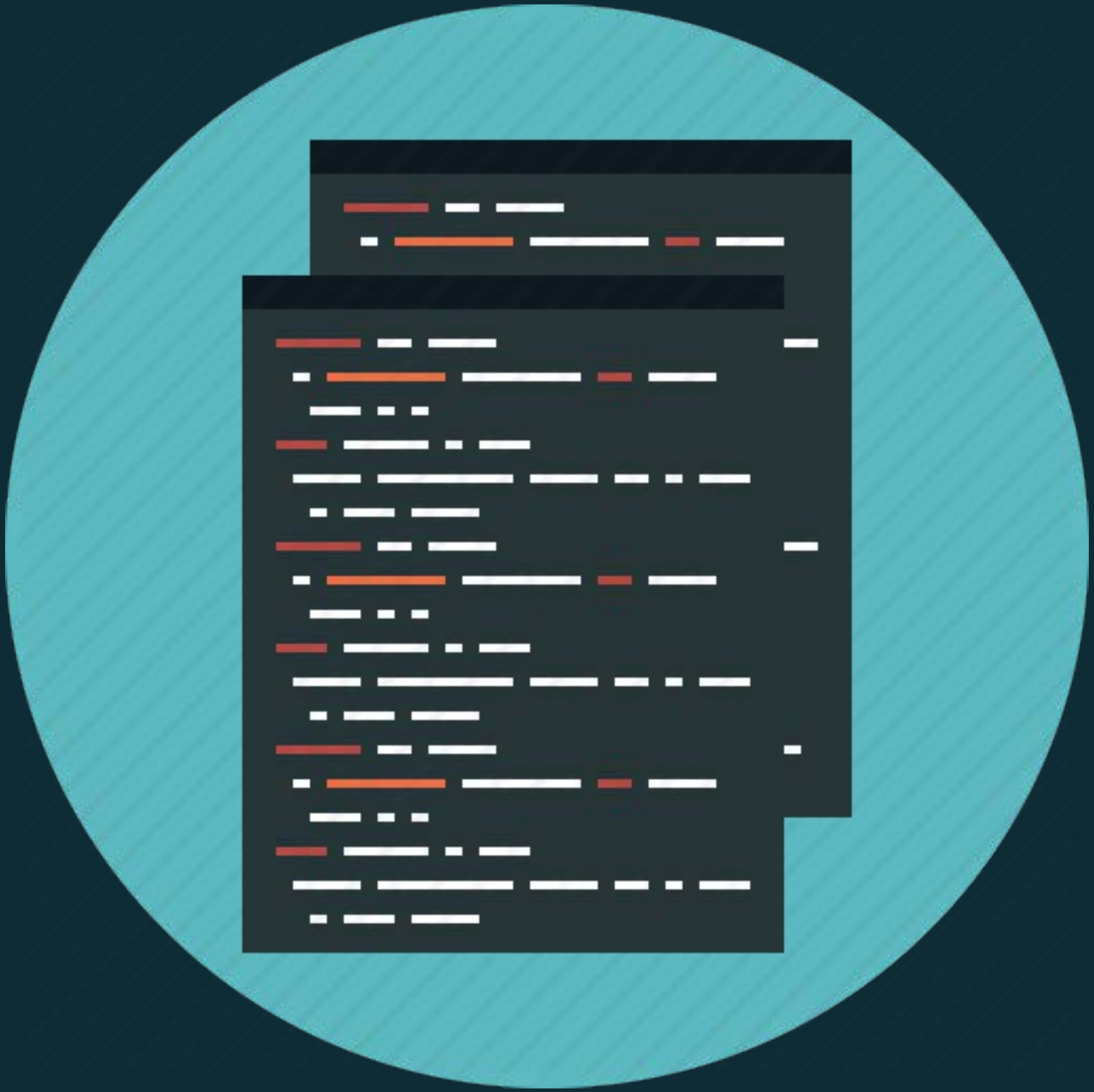
108 113 196 #6c71c4



Notice that we’ve dropped the weights

Blue 38 139 210 #268bd2

HAMILTONIAN PATH (CYCLE)



Violet 108 113 196 #6c71c4

COMPLEXITY

Green 133 153 0 #859900

QUADRATIC

N^2

Seems a bit optimistic. Many different permutations on this classification depending on the implementation.

Cyan 42 161 152 #2aa198

Notation	Name	Example
$O(1)$	constant	Determining if a binary number is even or odd; Calculating $(-1)^n$; Using a constant-size lookup table
$O(\log \log n)$	double logarithmic	Number of comparisons spent finding an item using interpolation search in a sorted array of uniformly distributed values
$O(\log n)$	logarithmic	Finding an item in a sorted array with a binary search or a balanced search tree as well as all operations in a Binomial heap
$O((\log n)^c),\ c > 1$	[illegible]	[illegible]
$O(n^c),\ 0 < c < 1$	fractional power	Searching in a kd-tree
$O(n)$	linear	Finding an item in an unsorted list or a malformed tree (worst case) or in an unsorted array; adding two n-bit integers by ripple carry
$O(n \log^* n)$	n log-star n	Performing triangulation of a simple polygon using Seidel's algorithm, or the union–find algorithm. Note that $\log^*(n) = \begin{cases} 0, & \text{if } n \leq 1 \\ 1 + \log^*(\log n), & \text{if } n > 1 \end{cases}$
$O(n \log n) = O(\log n!)$	linearithmic, loglinear, or [illegible]	Performing a fast Fourier transform; heapsort, quicksort (best and average case), or merge sort
$O(n^2)$	quadratic	Multiplying two n-digit numbers by a simple algorithm; bubble sort (worst case or naive implementation), Shell sort, quicksort (worst case), selection sort or insertion sort
$O(n^c),\ c > 1$	polynomial or algebraic	Tree-adjoining grammar parsing; maximum matching for bipartite graphs
$L_n[\alpha, c],\ 0 < \alpha < 1 = e^{(c+o(1))(\ln n)^\alpha (\ln \ln n)^{1-\alpha}}$	L-notation or sub-exponential	Factoring a number using the quadratic sieve or number field sieve
$O(c^n),\ c > 1$	exponential	Finding the (exact) solution to the travelling salesman problem using dynamic programming; determining if two logical statements are equivalent using brute-force search
$O(n!)$	factorial	Solving the traveling salesman problem via brute-force search; generating all unrestricted permutations of a poset; finding the determinant with expansion by minors; enumerating all partitions of a set

NP-Complete

Red 220 50 47 #dc322f

What is NP?

1 NP is the set of all decision problems (questions with a yes-or-no answer) for which the 'yes'-answers can be ***verified*** in polynomial time ($O(n^k)$ where *n* is the problem size, and *k* is a constant) by a deterministic Turing machine. Polynomial time is sometimes used as the definition of *fast* or *quickly*.

What is P?

P is the set of all decision problems which can be **solved** in *polynomial time* by a *deterministic Turing machine*. Since they can be solved in polynomial time, they can also be verified in polynomial time. Therefore P is a subset of NP.

What is NP-Complete?

2 A problem x that is in NP is also in NP-Complete *if and only if* every other problem in NP can be quickly (ie. in polynomial time) transformed into x.

In other words:

1. x is in NP, and
2. Every problem in NP is *reducible* to x

So, what makes *NP-Complete* so interesting is that if any one of the NP-Complete problems was to be solved quickly, then all *NP* problems can be solved quickly.

See also the post What's "P=NP?", and why is it such a famous question?

What is NP-Hard?

NP-Hard are problems that are at least as hard as the hardest problems in NP. Note that NP-Complete problems are also NP-hard. However not all NP-hard problems are NP (or even a decision problem), despite having `NP` as a prefix. That is the NP in NP-hard does not mean *non-deterministic polynomial time*. Yes, this is confusing, but its usage is entrenched and unlikely to change.

3 0 #859900

1

What is NP?

NP is the set of all decision problems (questions with a yes-or-no answer) for which the 'yes'-answers can be ***verified*** in polynomial time ($O(n^k)$ where *n* is the problem size, and *k* is a constant) by a deterministic Turing machine. Polynomial time is sometimes used as the definition of *fast* or *quickly*.

What is P?

P is the set of all decision problems which can be **solved** in *polynomial time* by a *deterministic Turing machine*. Since they can be solved in polynomial time, they can also be verified in polynomial time. Therefore P is a subset of NP.

2

What is NP-Complete?

A problem x that is in NP is also in NP-Complete *if and only if* every other problem in NP can be quickly (ie. in polynomial time) transformed into x.

In other words:

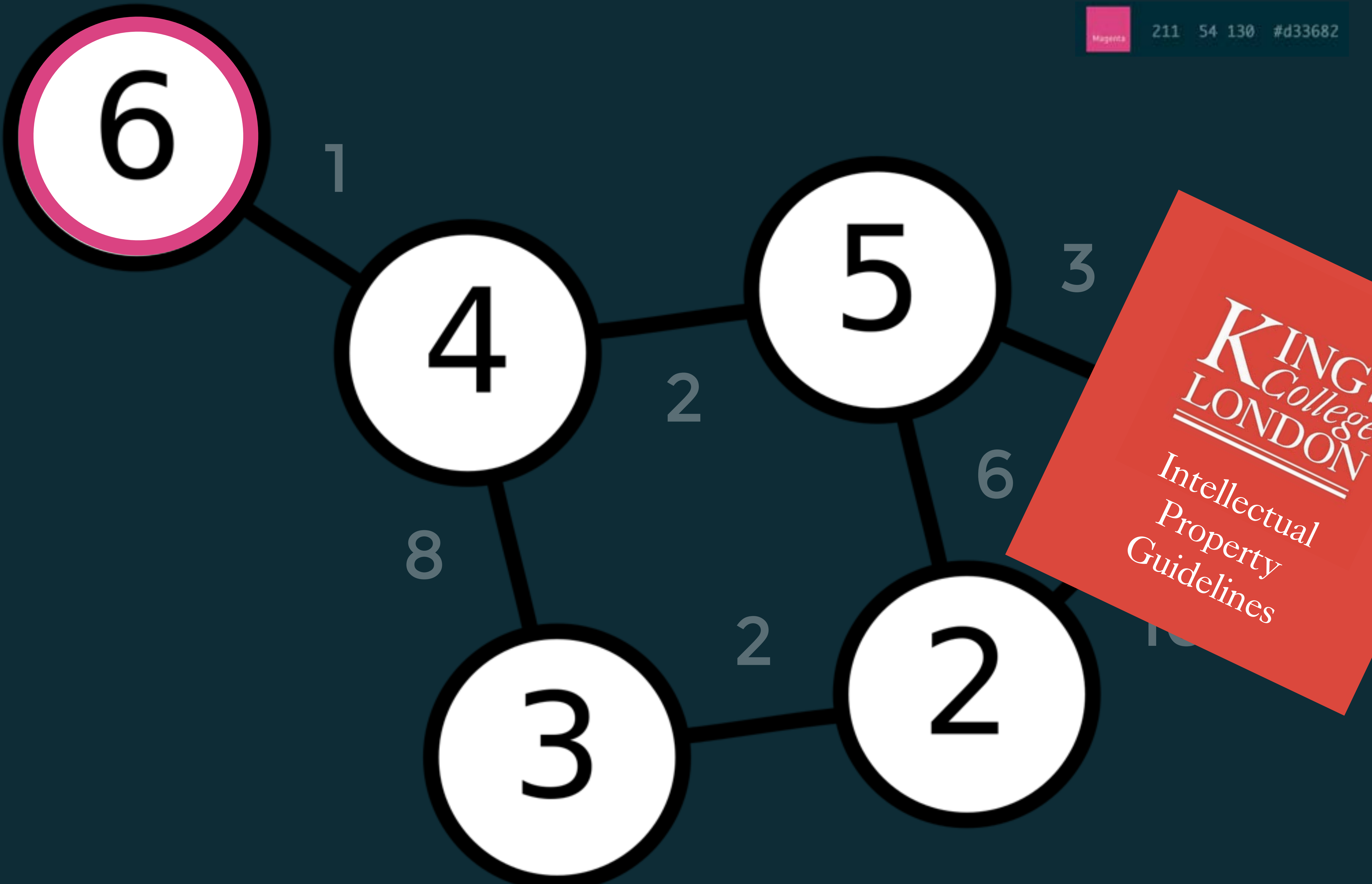
1. x is in NP, and
2. Every problem in NP is *reducible* to x

So, what makes *NP-Complete* so interesting is that if any one of the NP-Complete problems was to be solved quickly, then all *NP* problems can be solved quickly.

See also the post What's "P=NP?", and why is it such a famous question?

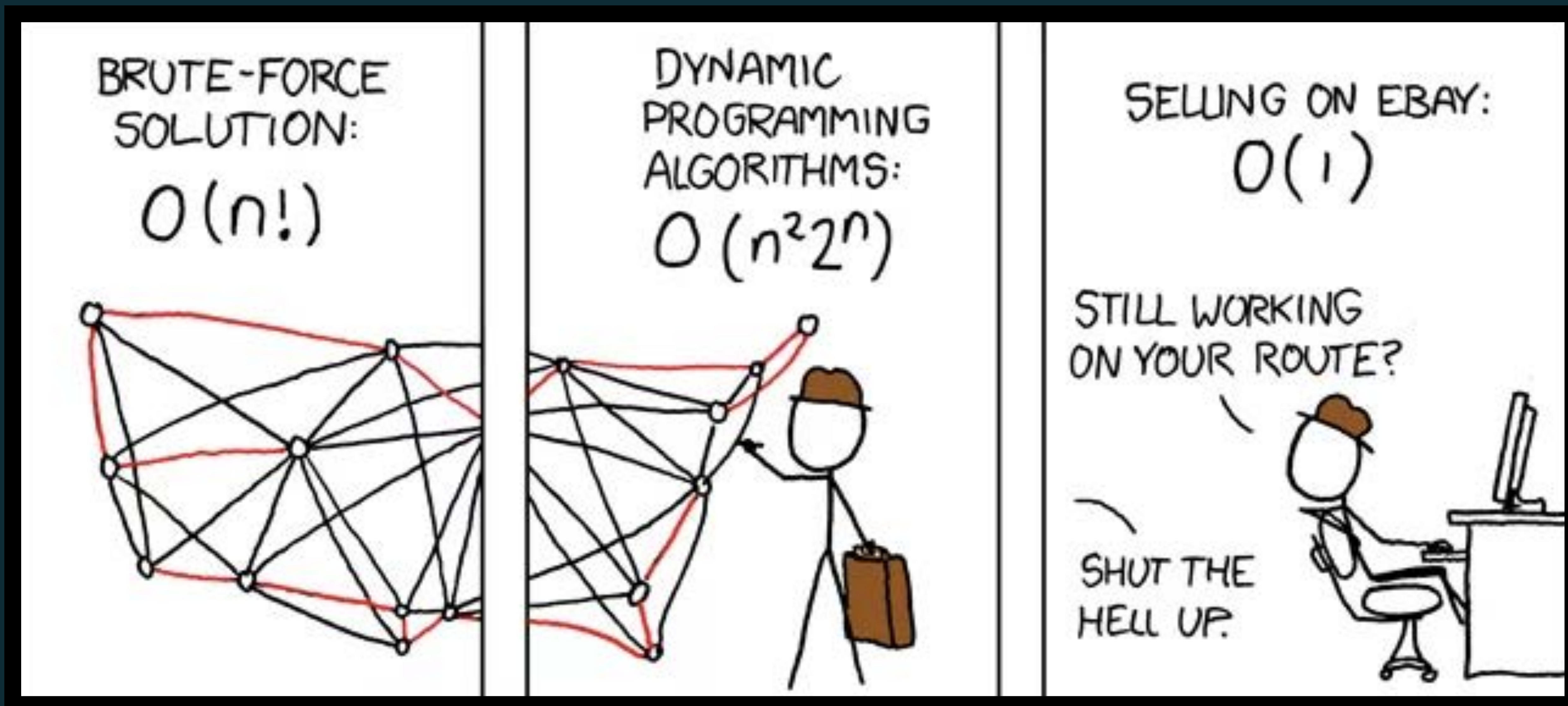
What is NP-Hard?

NP-Hard are problems that are at least as hard as the hardest problems in NP. Note that NP-Complete problems are also NP-hard. However not all NP-hard problems are NP (or even a decision problem), despite having `NP` as a prefix. That is the NP in NP-hard does not mean *non-deterministic polynomial time*. Yes, this is confusing, but its usage is entrenched and unlikely to change.



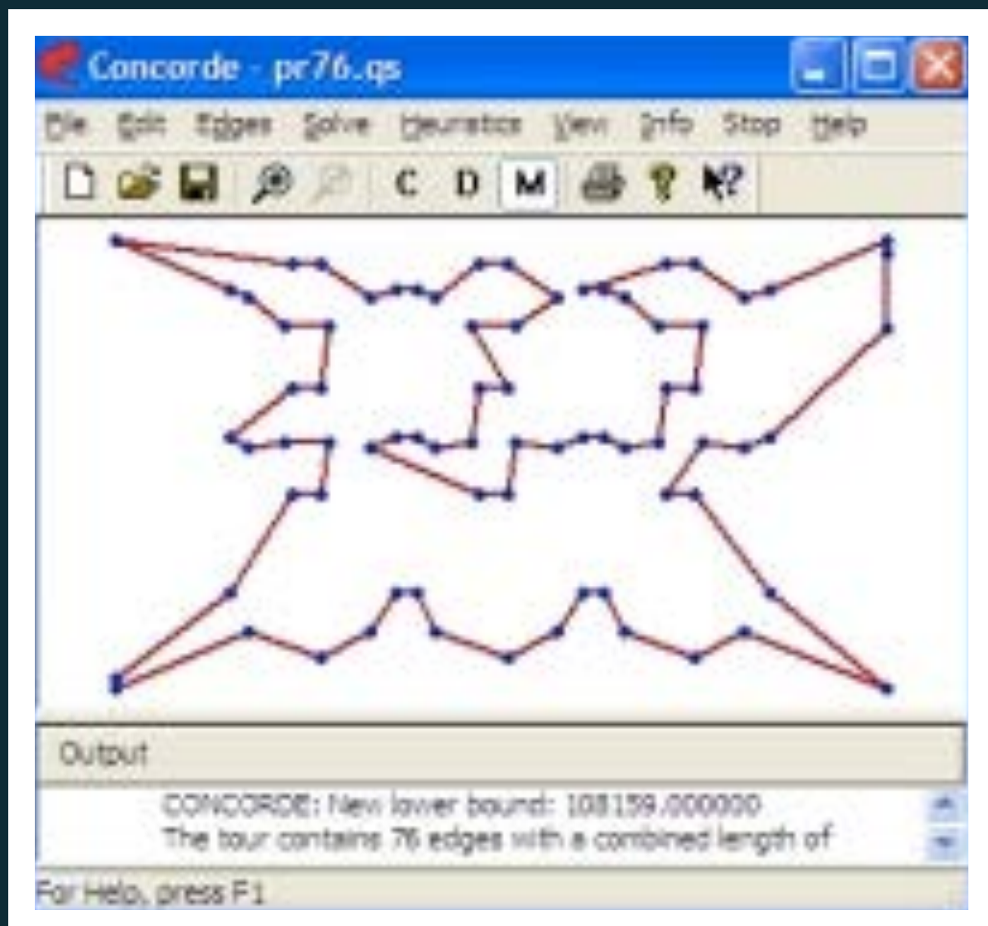
Weights are back, so need to think about efficiency: Exponential and NP-hard

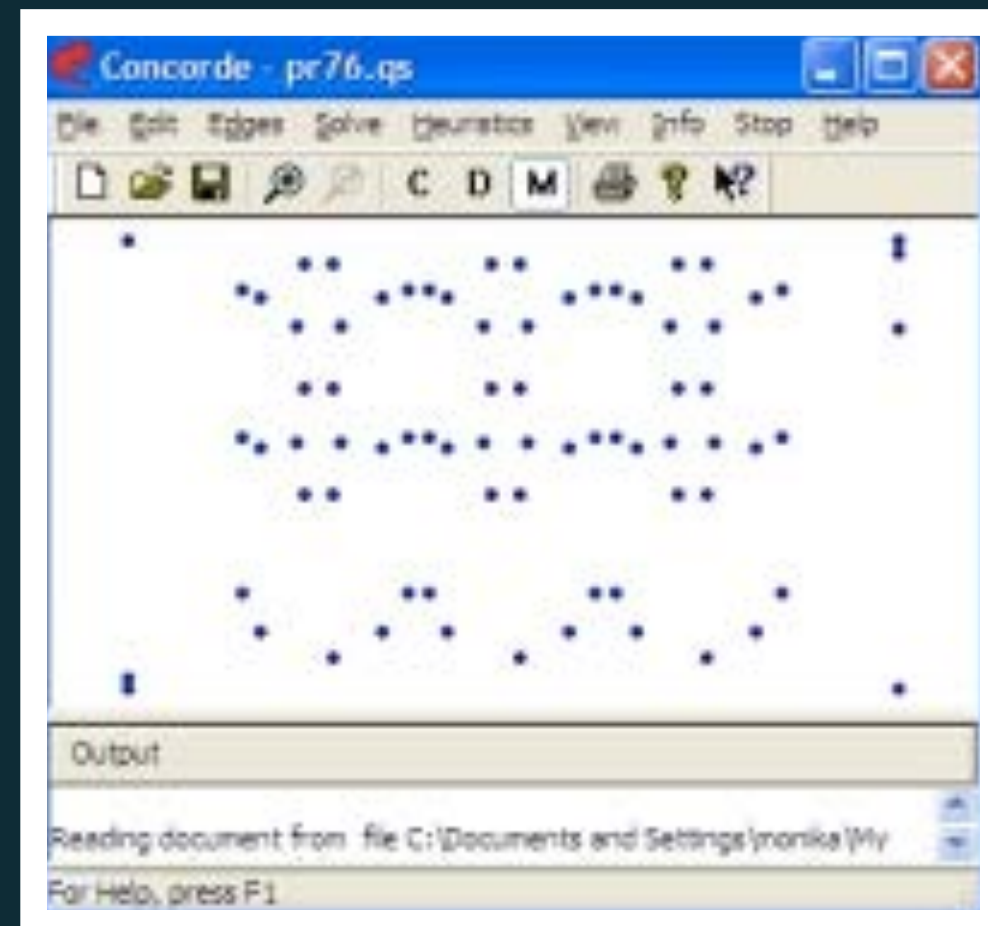
108 113 196 #6c71c4



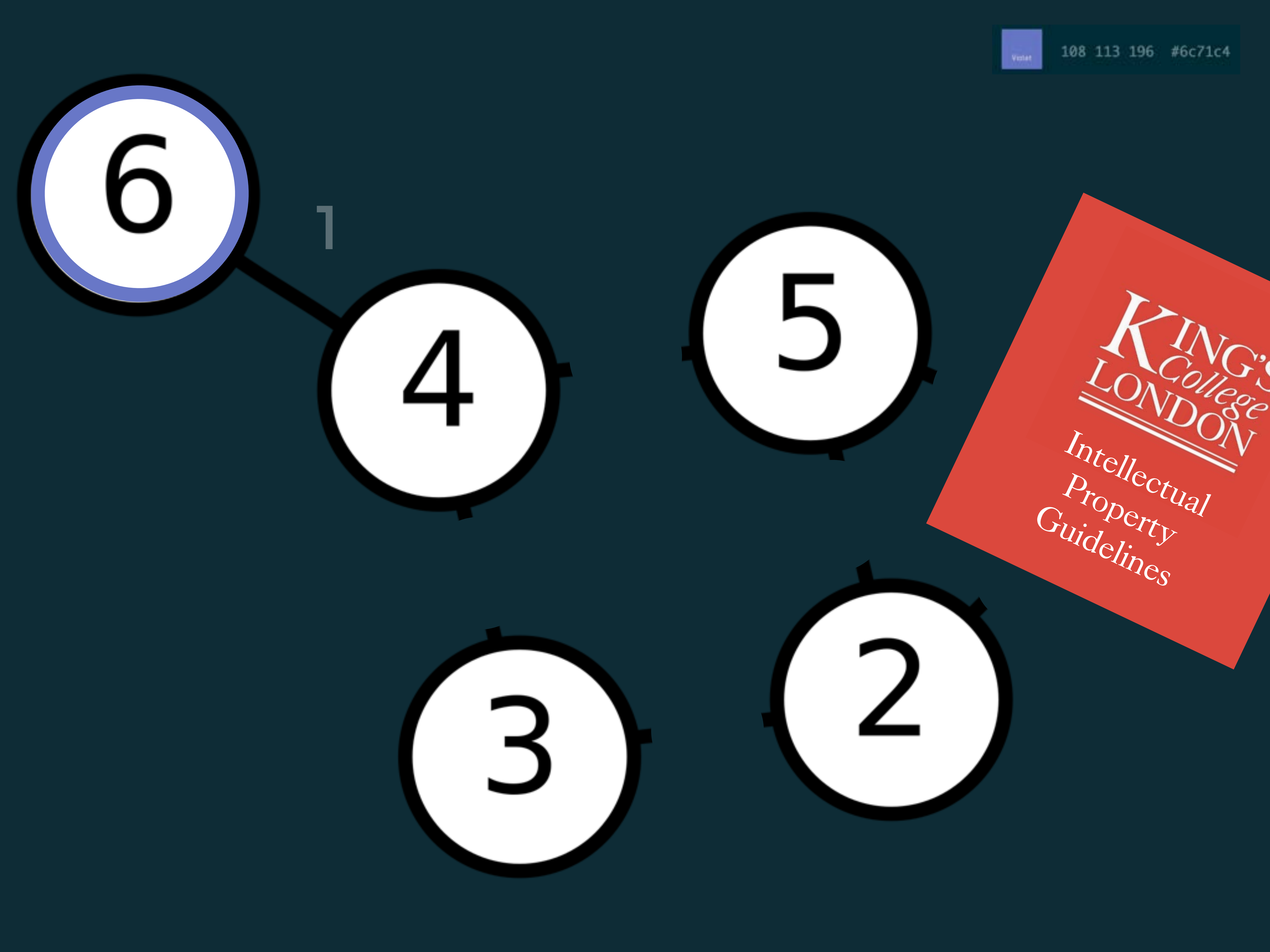
An intuitive analogy for the relationship between running times (complexity)

133 153 0 #859900





David Applegate, Ribert Bixby, Vasek Chvatal, and William Cook. Concorde TSP Solver. Available at http://www.math.uwaterloo.ca/tsp/concorde/, 2006.

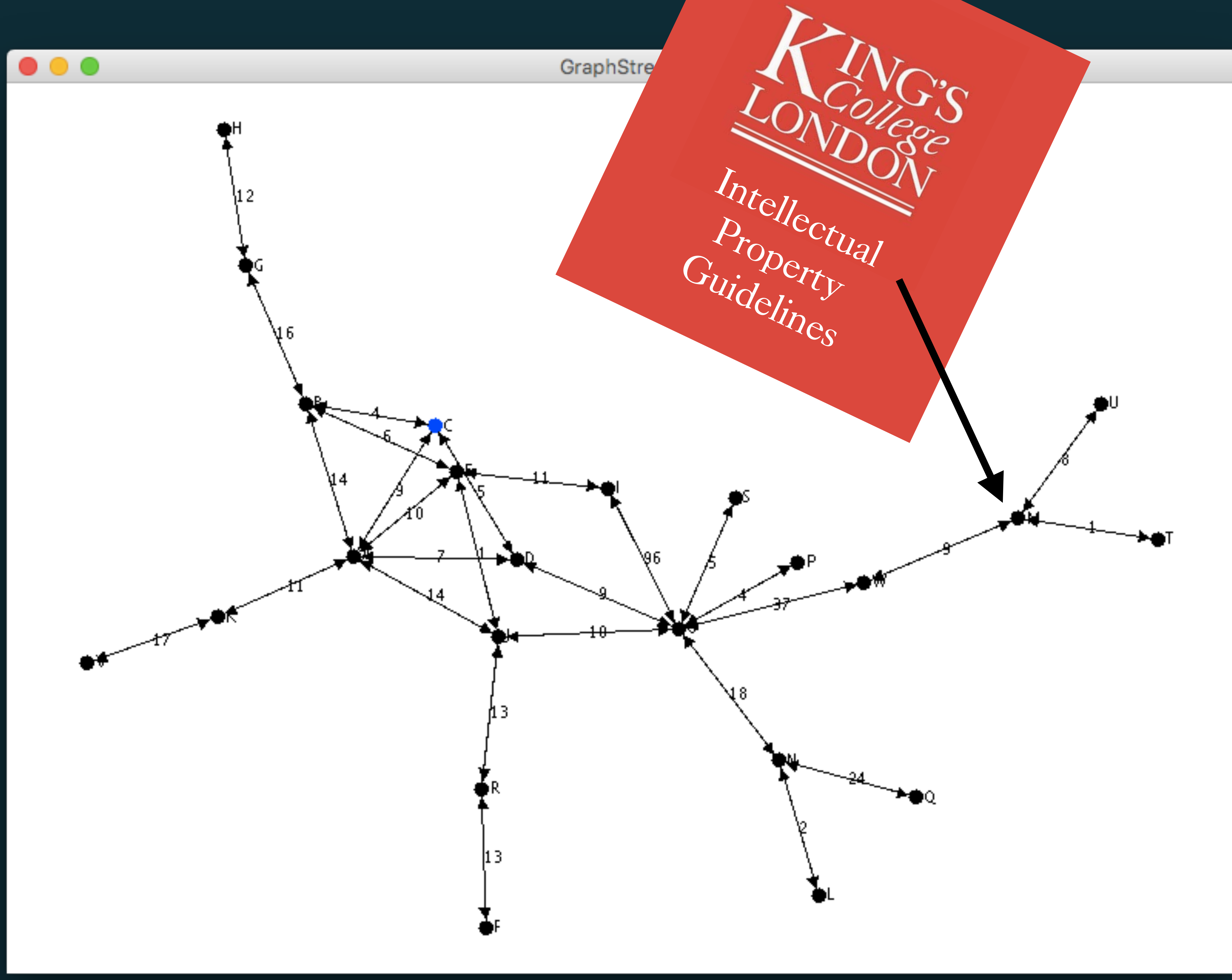

108 113 196 #6c71c4
6
1
4
5
3
2
KING'S College LONDON
Intellectual Property Guidelines

Orange 203 75 22 #cb4b16

MORE GENERAL EXPLORATION STRATEGIES ARE NEEDED

210 #268bd2

Experimentation is needed.


GraphStre
KING'S College LONDON
Intellectual Property Guidelines

Constructor Summary

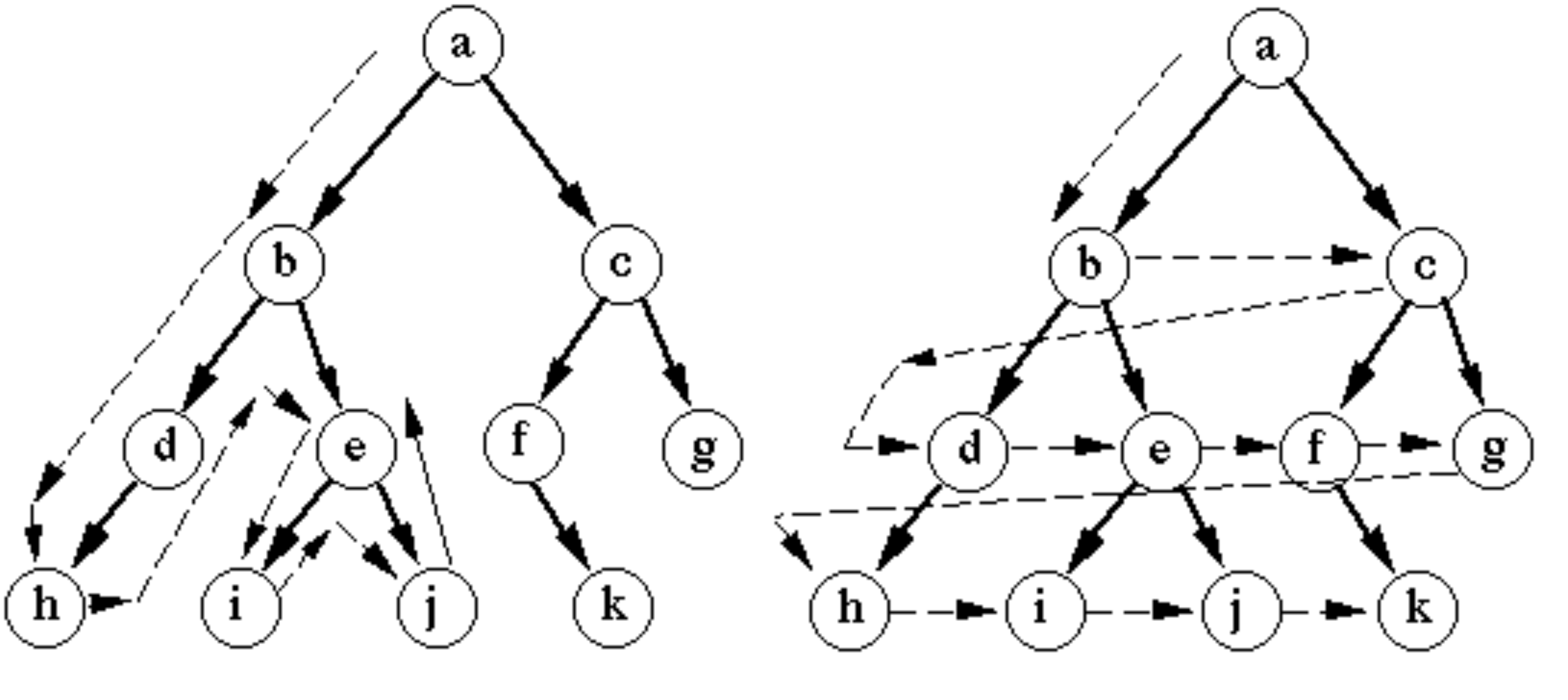
Constructors

Constructor and Description
`EncapsulatedGraph(java.lang.String yourName)` Create a new graph that will communicate with my server.

Method Summary

All Methods **Instance Methods** **Concrete Methods**

Modifier and Type	Method and Description
`java.util.ArrayList<org.graphstream.graph.Edge>`	`edgesOfCurrentNode()` Gets the edges associated with the current node
`boolean`	`found()` Whether the desired object, hidden within this graph, has been found.
`org.graphstream.graph.Node`	`getCurrentNode()` The most important piece of encapsulated graph state is the current node, upon which a `searcher' currently exists.
`java.util.ArrayList<org.graphstream.graph.Node>`	`getPath()` Get the path you have created so far with your search.
`boolean`	`moveToNewNode(org.graphstream.graph.Node node)` Move the `pointer' within the encapsulated graph to a new node.
`void`	`sendPath()` Upload your path to the server for scoring!



Depth-first search

Breadth-first search

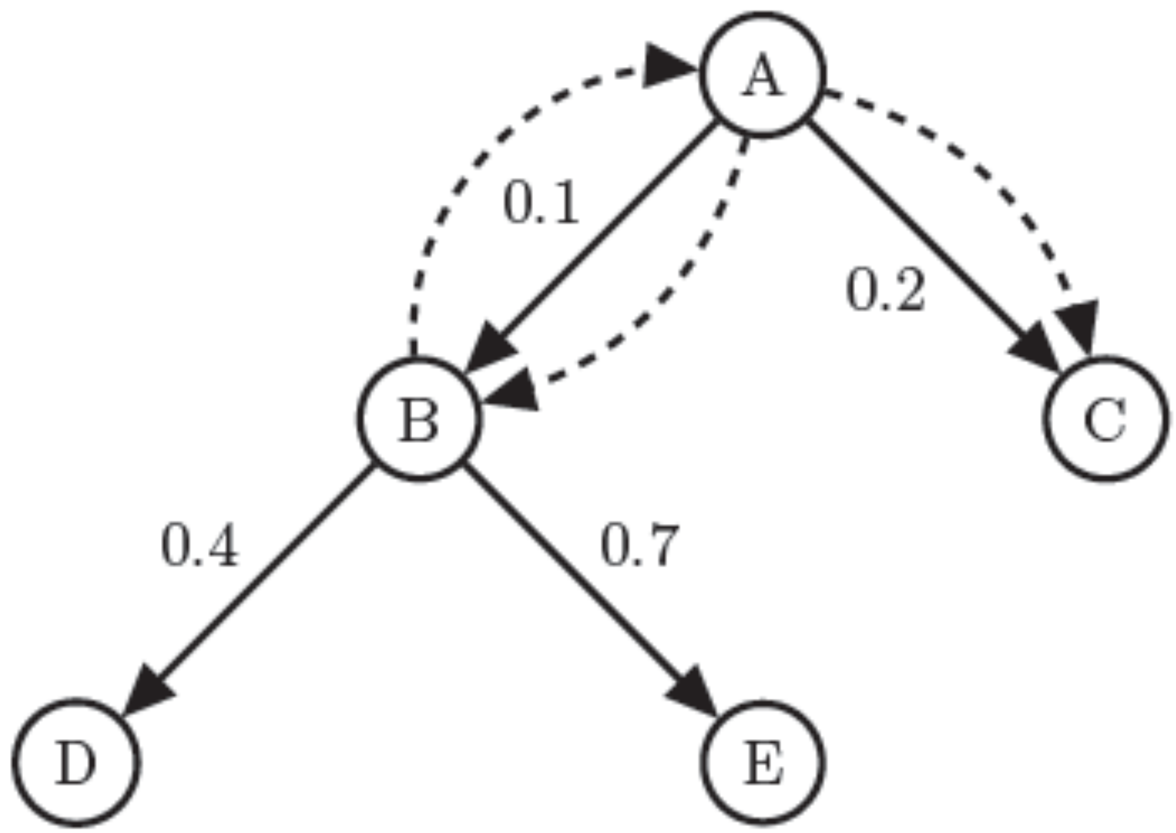


Figure 4.1: An example of how sBacktrackGreedy may backtrack to save cost: this strategy starts at node A, and then proceeds to node B, after recording the distance to node C. At vertex B, sBacktrackGreedy calculates that it will cost strictly less to move to node C *via* node A, where it has already been, than to take the single hop to nodes D or E, so it makes this move.

Base03	0	43	54	#002b36	Yellow	181	137	0	#b58900
Base02	7	54	66	#073642	Orange	203	75	22	#cb4b16
Base01	88	110	117	#586e75	Red	220	50	47	#dc322f
Base00	101	123	131	#657b83	Magenta	211	54	130	#d33682
Base0	131	148	150	#839496	Violet	108	113	196	#6c71c4
Base1	147	161	161	#93a1a1	Blue	38	139	210	#268bd2
Base2	238	232	213	#eee8d5	Cyan	42	161	152	#2aa198
Base3	253	246	227	#fdf6e3	Green	133	153	0	#859900