Week 7: Monday

EECS 281

MAXIM ALEKSA

maximal.io

Agenda

Project 3 postmortem

Graphs

Graph Traversals

Minimum Spanning Tree

Algorithm Design

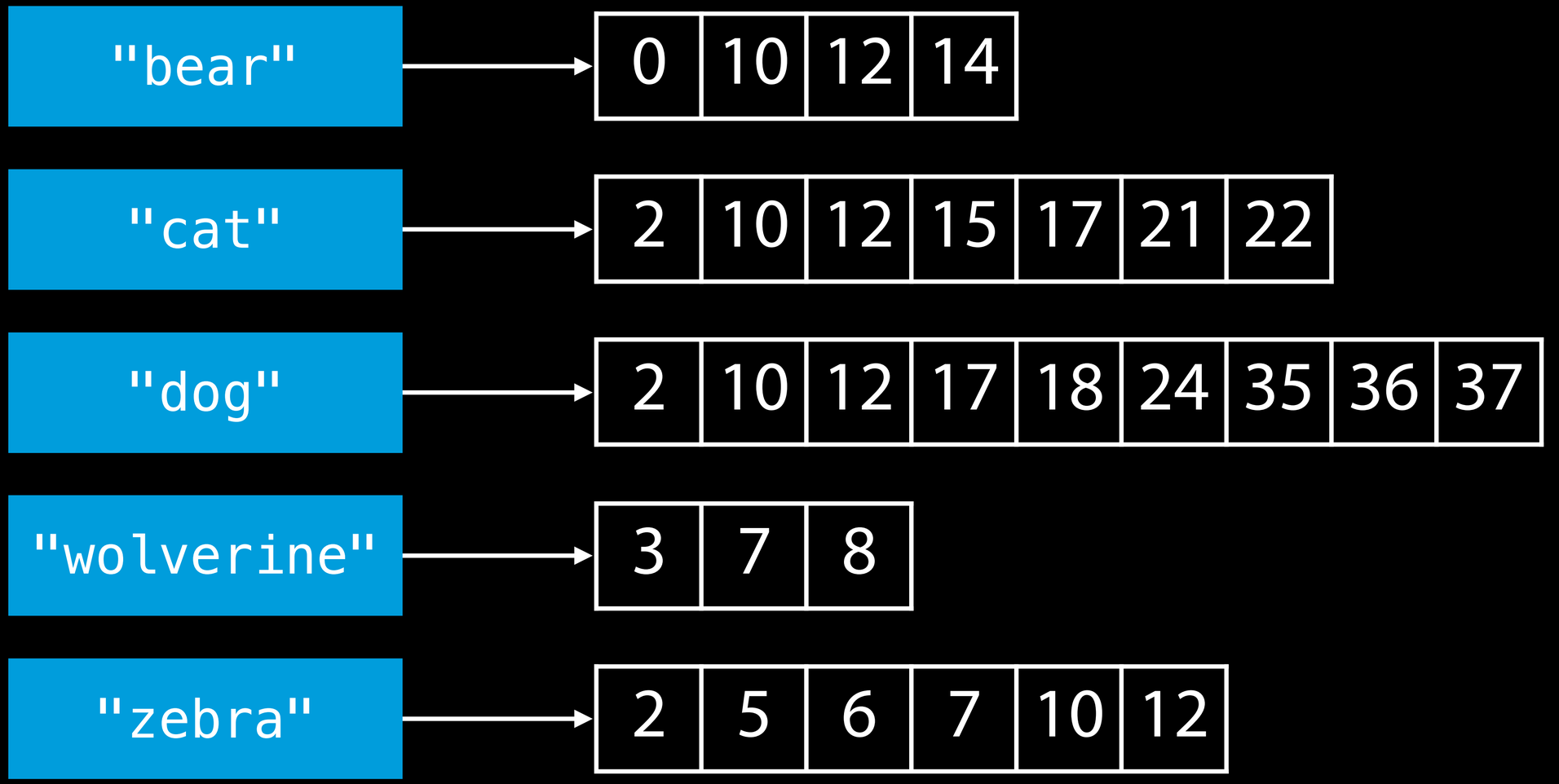
Brute force

Greedy

Swift

Project 3

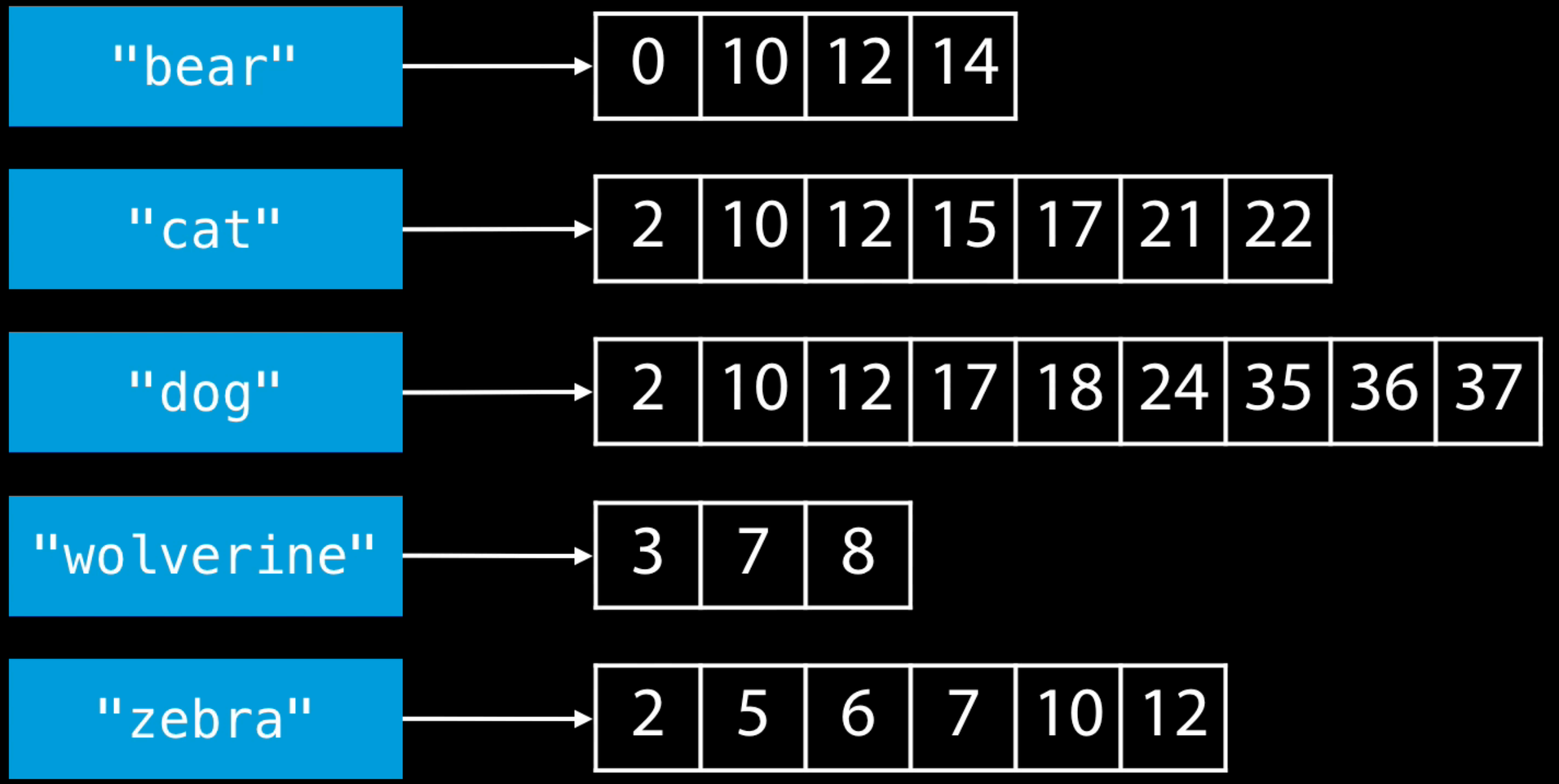
Project 3


"bear"
0 10 12 14
"cat"
2 10 12 15 17 21 22
"dog"
2 10 12 17 18 24 35 36 37
"wolverine"
3 7 8
"zebra"
2 5 6 7 10 12

Project 3

Comparison $\Theta(n \log n)$

Counting $\rightarrow \Theta(n)$



Project 3

`k cat dog zebra`

2	10	12

Excerpt list

2	10	12	0	1	6	10	36	37	28	29	31

Project 3

`k cat dog zebra`

2	10	12

search results

Excerpt list

sort ⇝ merge $\Theta(n)$

2	10	12	0	1	6	10	36	37	28	29	31	2	10	12

Project 3

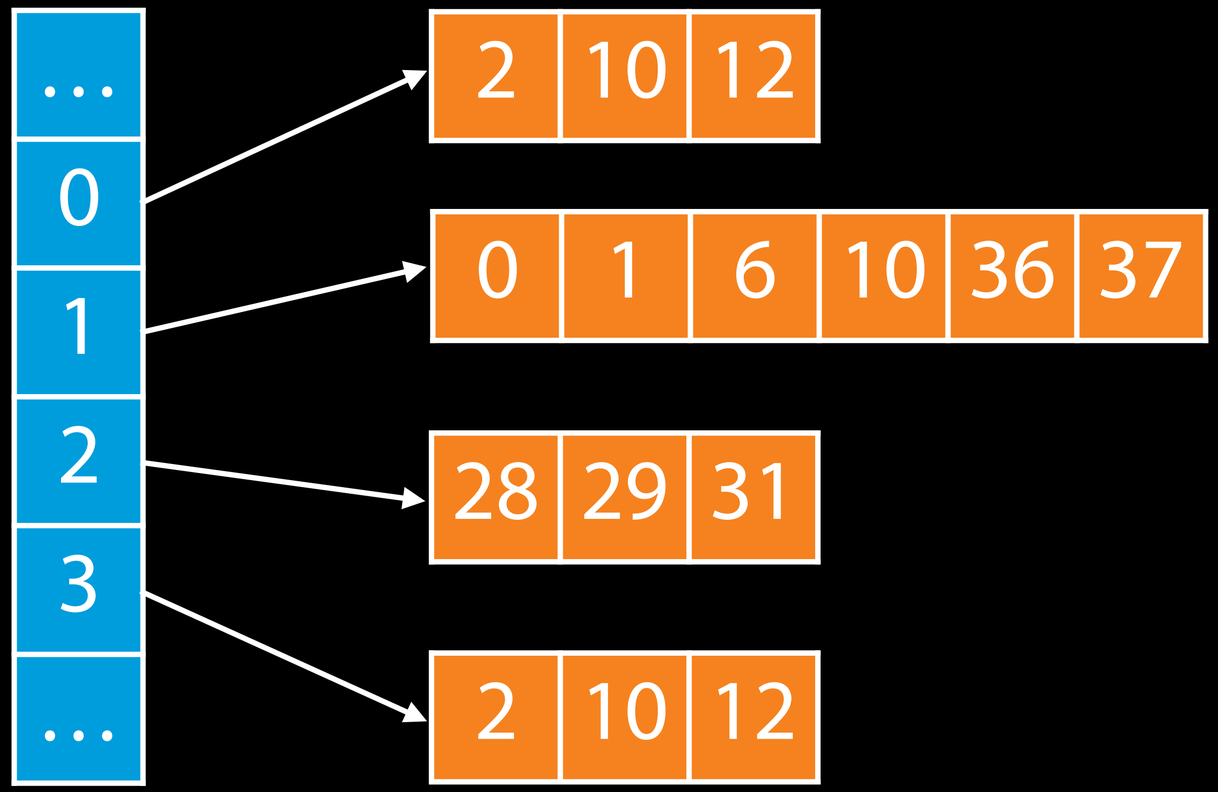
`k cat dog zebra`

2	10	12

Excerpt list

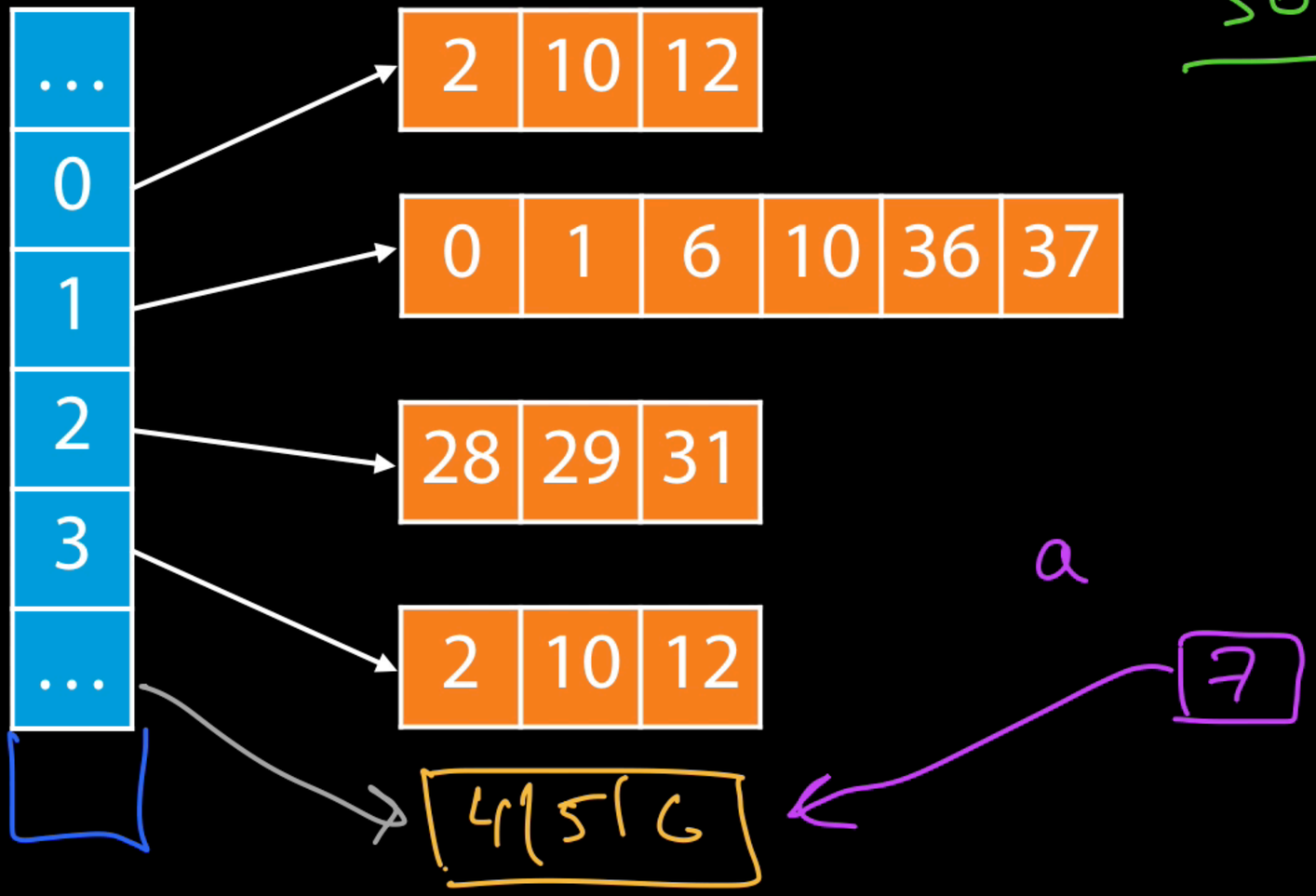
2	10	12	0	1	6	10	36	37	28	29	31	2	10	12

Project 3



Project 3

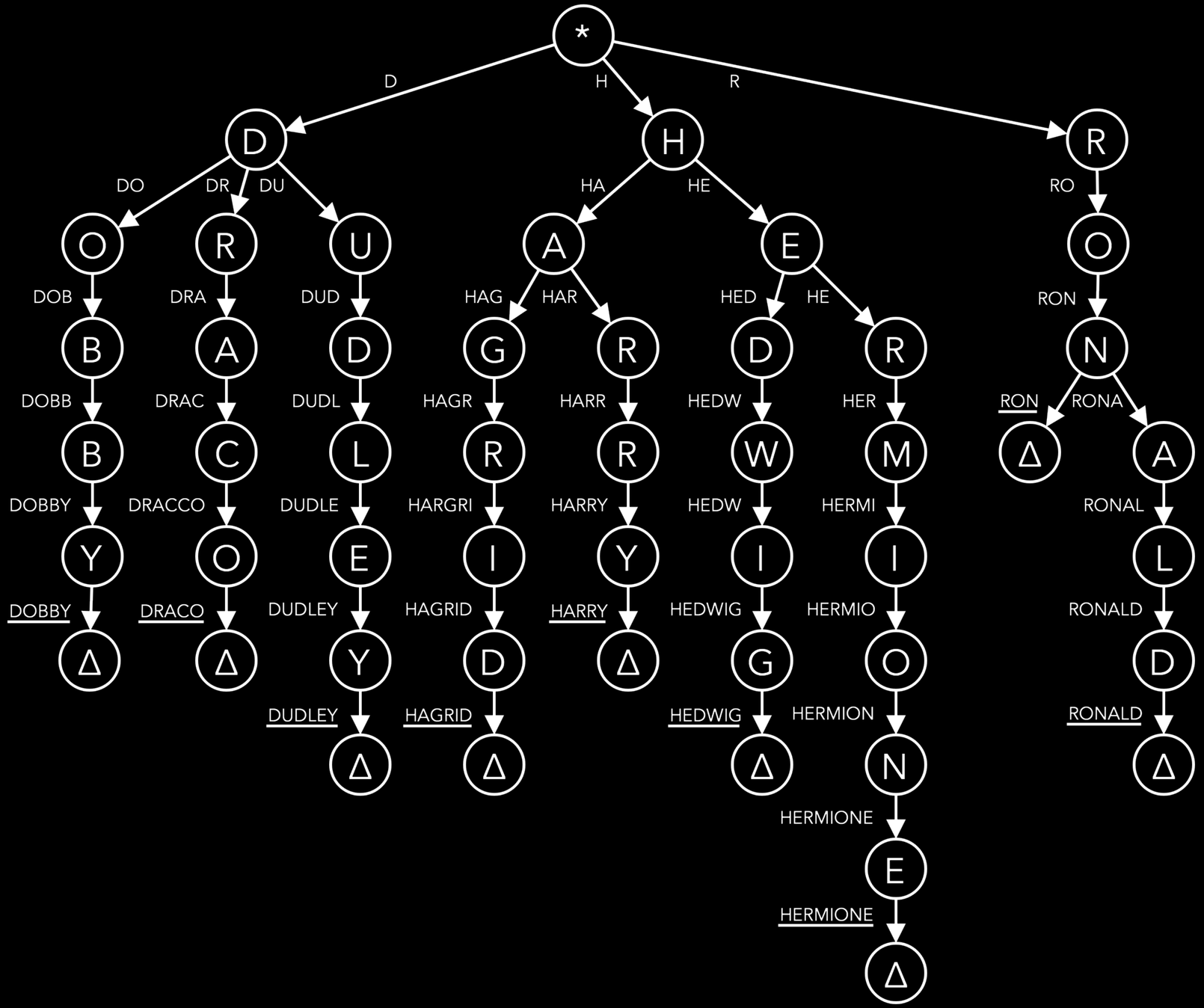
deque of sublists (vectors)

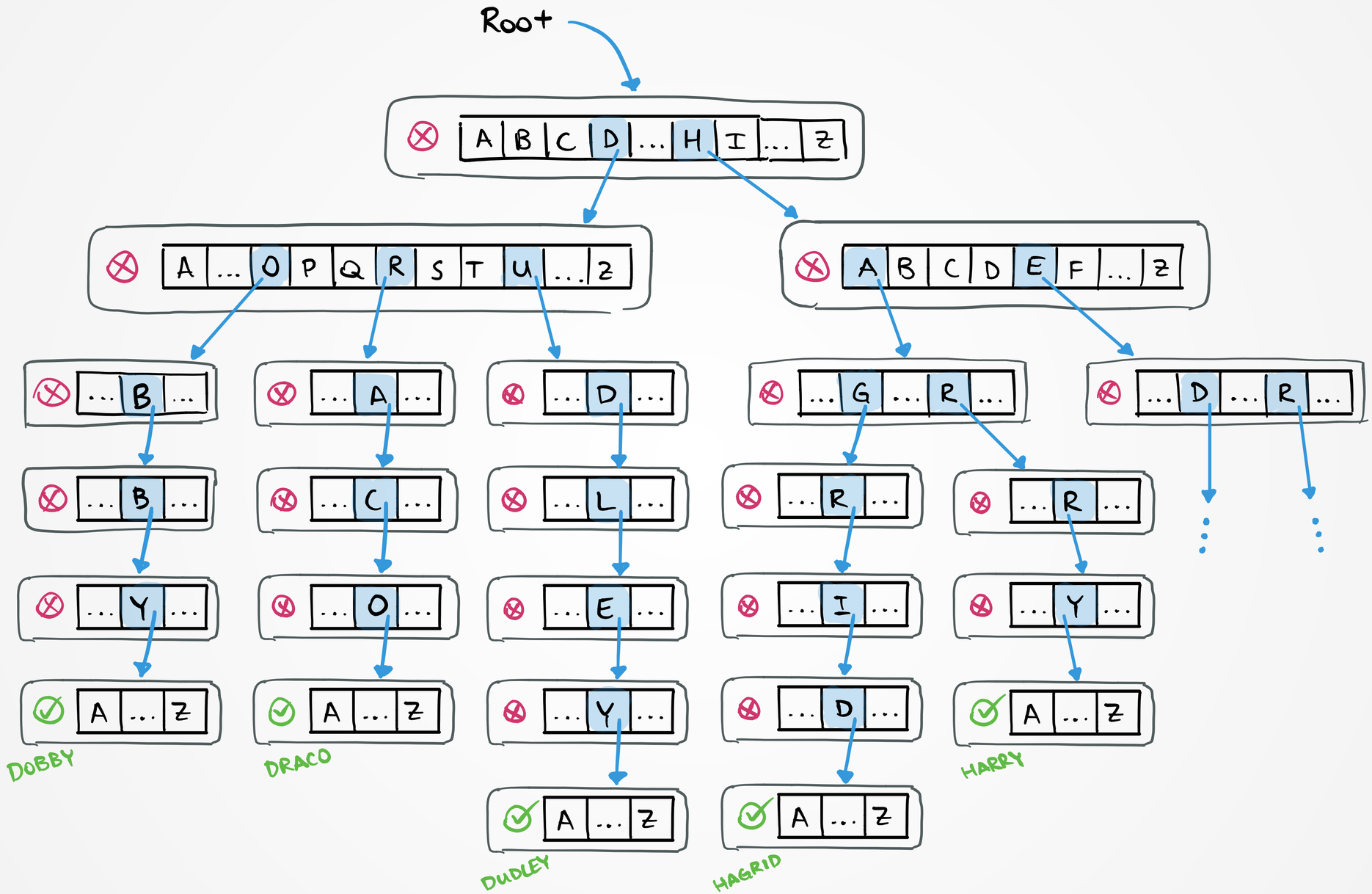


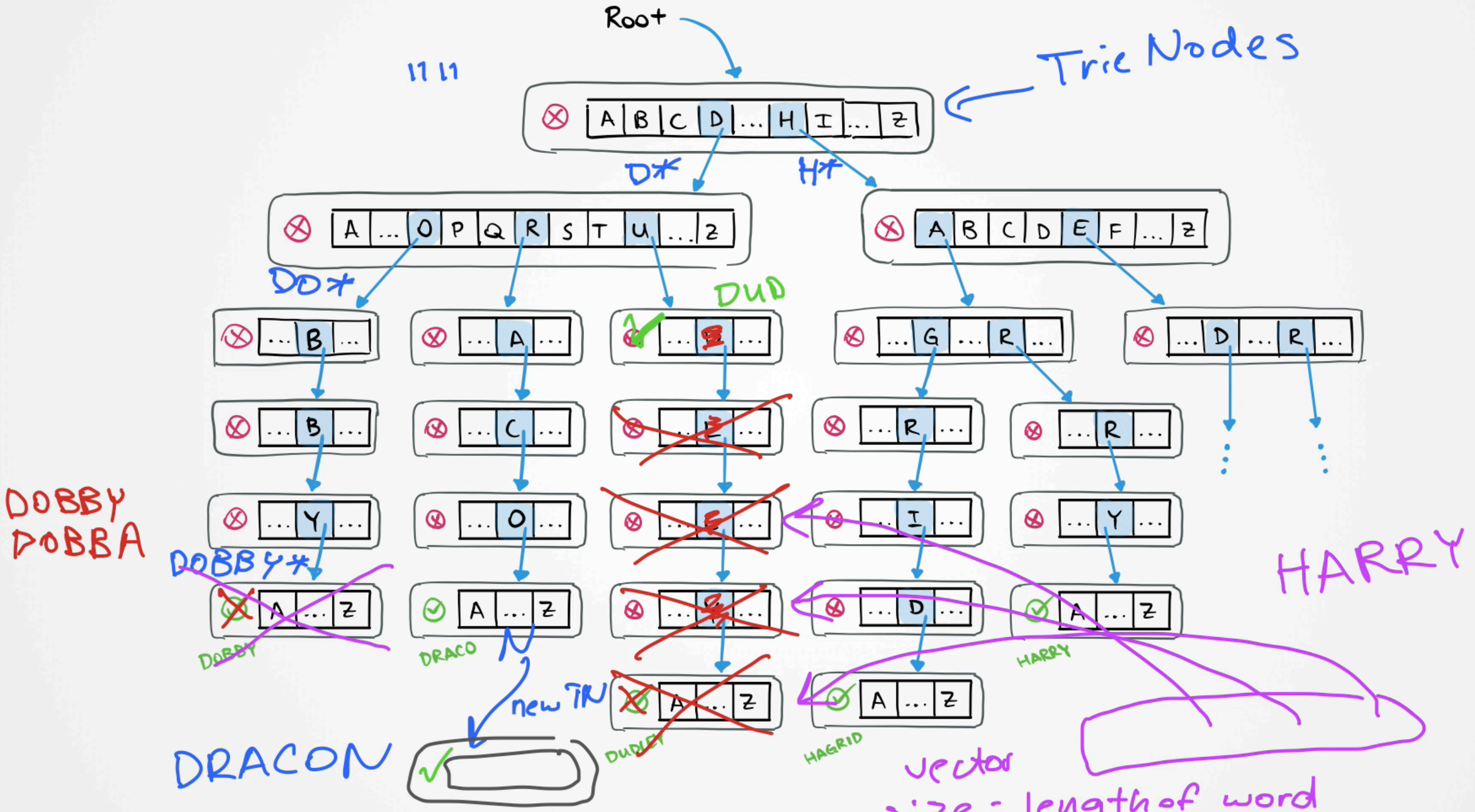
sort →

take 2 sublists
merge
put at end
of deque

Lab 8: Mispelings


*
D
H
R
D
H
R
DO
DR
DU
HA
HE
RO
O
R
U
A
E
O
DOB
DRA
DUD
HAG
HAR
HED
HE
RON
B
A
D
G
R
D
R
N
DOBB
DRAC
DUDL
HAGR
HARR
HEDW
HER
RON
RONA
B
C
L
R
R
W
M
Δ
A
DOBBY
DRACCO
DUDLE
HARGRI
HARRY
HEDW
HERMI
RONAL
Y
O
E
I
Y
I
I
L
DOBBY
DRACO
DUDLEY
HAGRID
HARRY
HEDWIG
HERMIO
RONALD
Δ
Δ
Y
D
Δ
G
O
D
DUDLEY
HAGRID
HEDWIG
HERMION
RONALD
Δ
Δ
Δ
N
Δ
HERMIONE
E
HERMIONE
Δ


Root
A B C D ... H I ... Z
A ... O P Q R S T U ... Z
A B C D E F ... Z
... B ...
... A ...
... D ...
... G ... R ...
... D ... R ...
... B ...
... C ...
... L ...
... R ...
... R ...
... Y ...
... O ...
... E ...
... I ...
... Y ...
A ... Z
A ... Z
... Y ...
... D ...
A ... Z
A ... Z
DOBBY
DRACO
DUDLEY
HAGRID
HARRY


Root
Trie Nodes
D*
H*
DO*
DUD
DOBBY
DOBBA
DOBBY*
HARRY
DRACON
new TN
vector
size = length of word
DOBBY
DRACO
DUDLEY
HAGRID
HARRY

Graphs

Graph traversal

Pre-order depth-first traversal

Post-order depth-first traversal

Breadth-first traversal

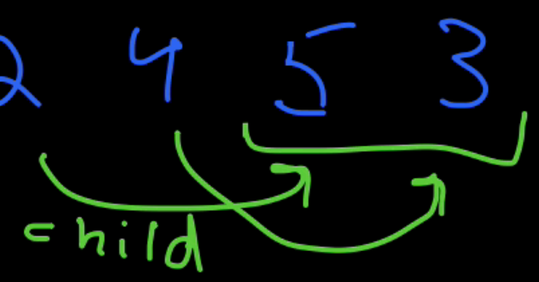
Graph traversal

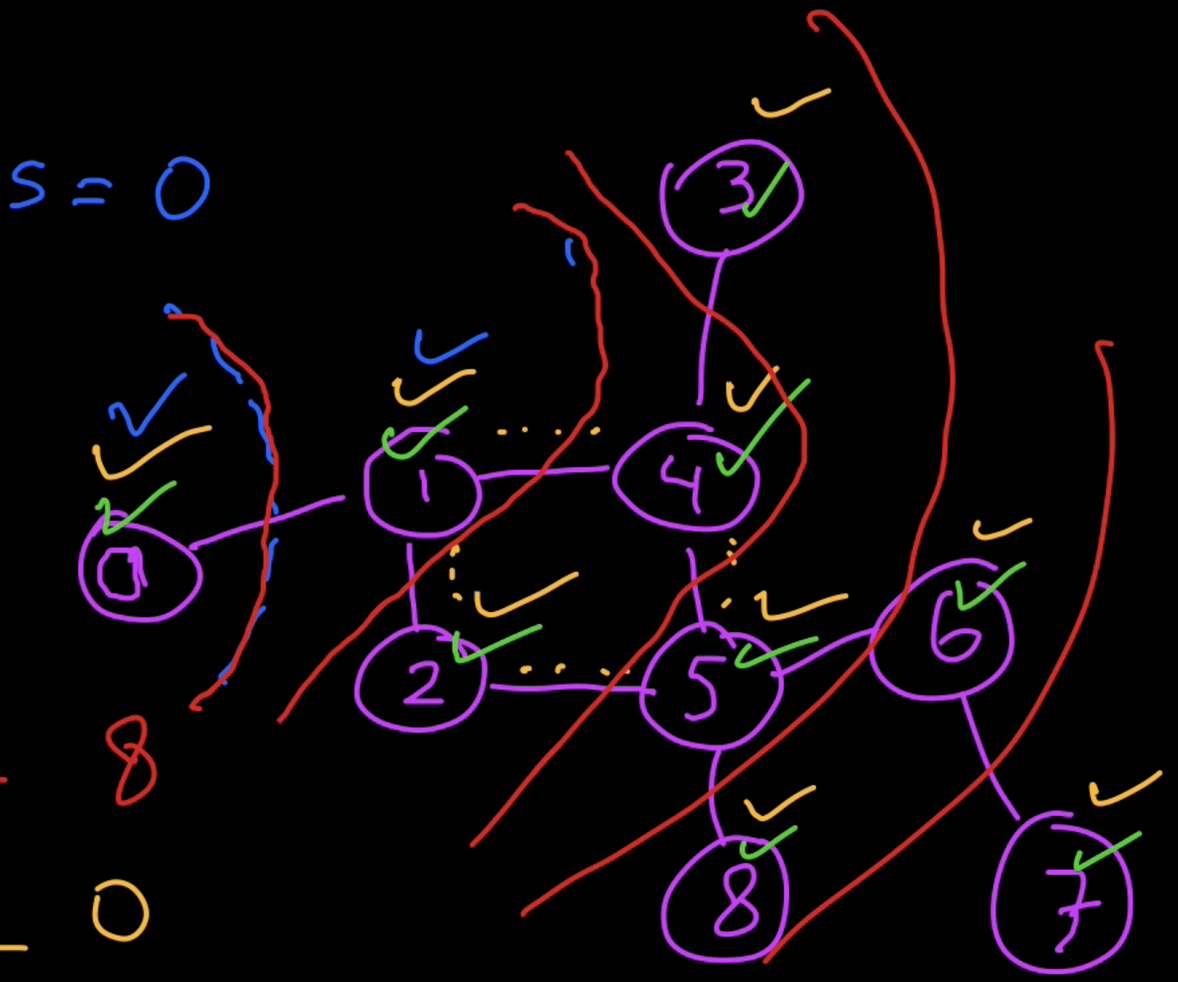
1. Pre-order depth-first traversal
2. Post-order depth-first traversal
3. Breadth-first traversal

1 0 1 2 5 4 3 6 7 8

2 3 4 7 6 8 5 2 1 0

3 0 1 2 4 5 3 6 8 7






```
n.visited = T
for c in n.children
    dfs(c)
visit(n)
```

Graph problems

***s*-*t* Path**. Is there a path between vertices s and t?

Shortest *s*-*t* Path. What is the shortest path between vertices *s* and *t*?

Cycle. Does the graph contain any cycles?

Euler Tour. Is there a cycle that uses every edge exactly once?

Hamilton Tour. Is there a cycle that uses every vertex exactly once?

Graph problems

Connectivity. Is the graph connected, i.e. is there a path between all vertex pairs?

Biconnectivity. Is there a vertex whose removal disconnects the graph?

Planarity. Can you draw the graph on a piece of paper with no crossing edges?

Isomorphism. Are two graphs isomorphic (the same graph in disguise).

Graph problems

Shortest path.

Minimum spanning tree.

Graph problems

Shortest path.

s to t

or through all nodes

Minimum spanning tree.

weighted graphs

↓

edges have weight

$-\infty \ldots 0 \ldots \infty$

(unweighted:
all weights are 1)

Minimum spanning tree

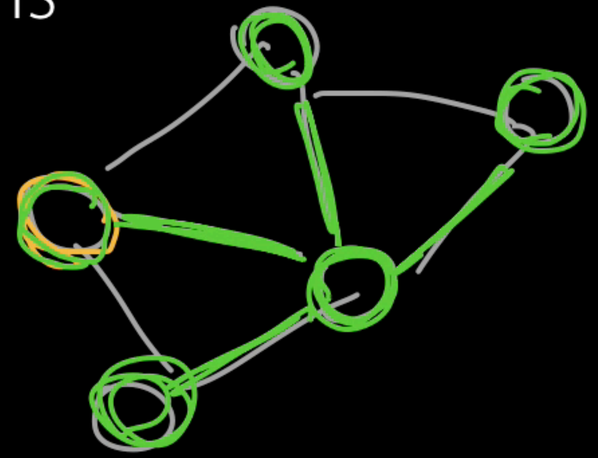
Given an undirected graph *G*, a spanning tree *T* is a subgraph of *G*, where T:

- Is connected.
- Is acyclic.
- Includes all of the vertices.

A minimum spanning tree is a spanning tree of minimum total weight.

Minimum spanning tree

Given an undirected graph *G*, a spanning tree *T* is a subgraph of *G*, where T:



- Is **connected**.
- Is **acyclic**.
- Includes **all of the vertices**.

A **minimum spanning tree** is a spanning tree of minimum total weight.

Minimum spanning tree

Prim’s algorithm

Kruskal’s algorithm

Prim's Algorithm

Start from some arbitrary start vertex.

Repeatedly add the vertex closest to the MST under construction.

Repeat until $V - 1$ edges.

Prim's Algorithm

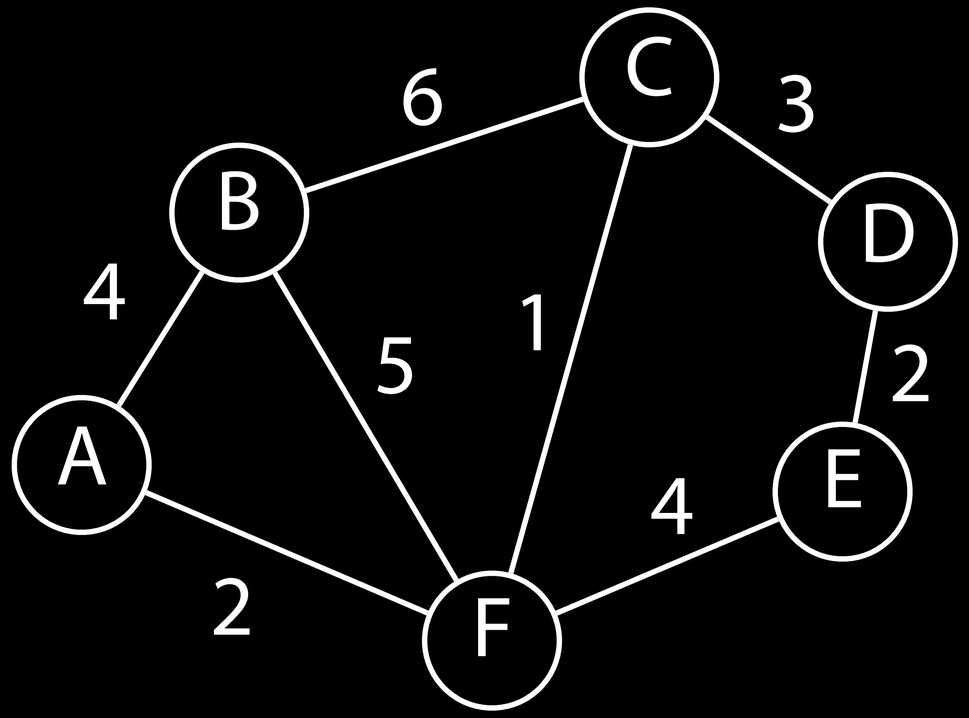
Start from some arbitrary start vertex.

Repeatedly add the vertex closest to the MST under construction.

Repeat until $V - 1$ edges.

Greedy

Prim's Algorithm



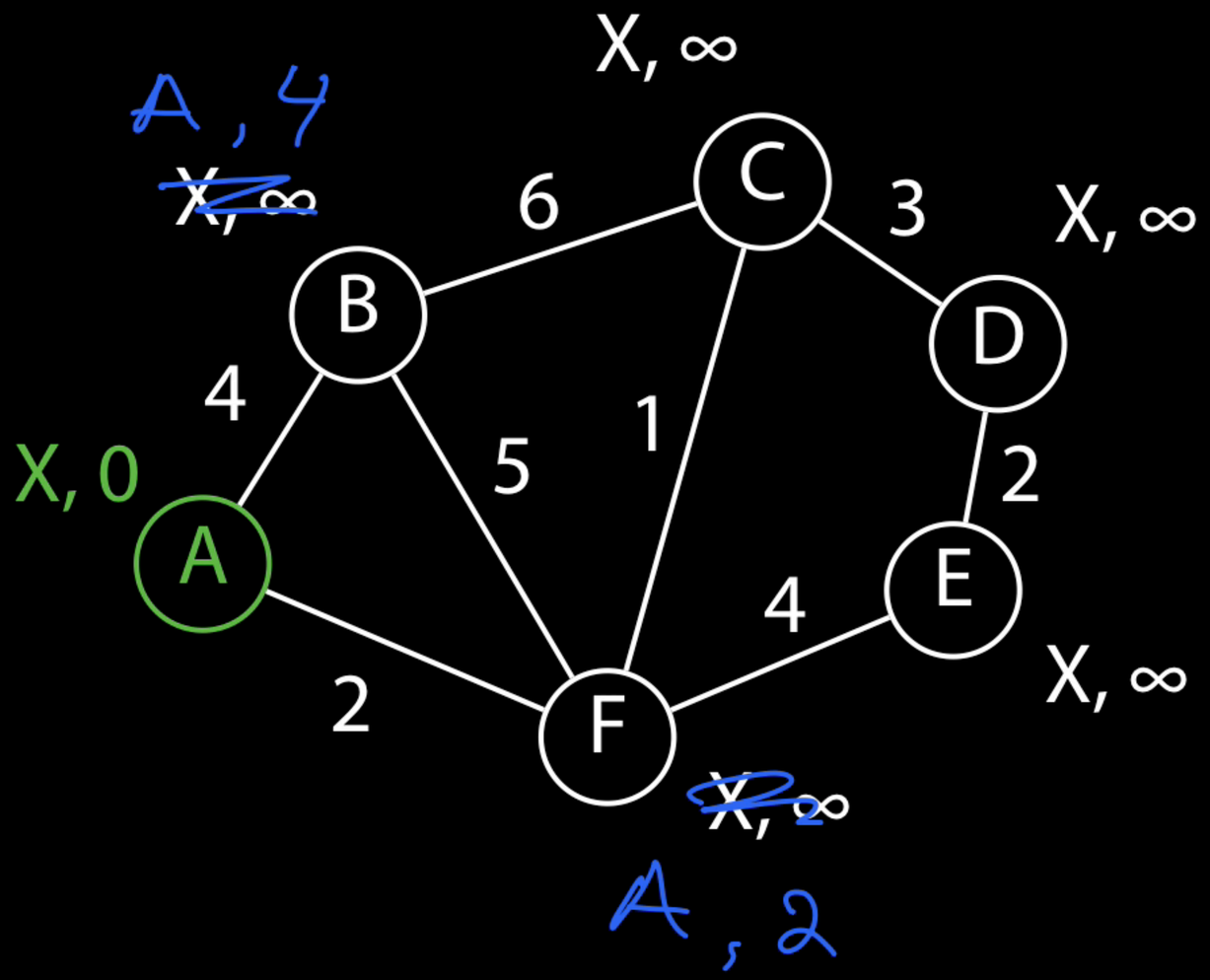
Repeatedly add the vertex closest to the MST under construction.

`distances`

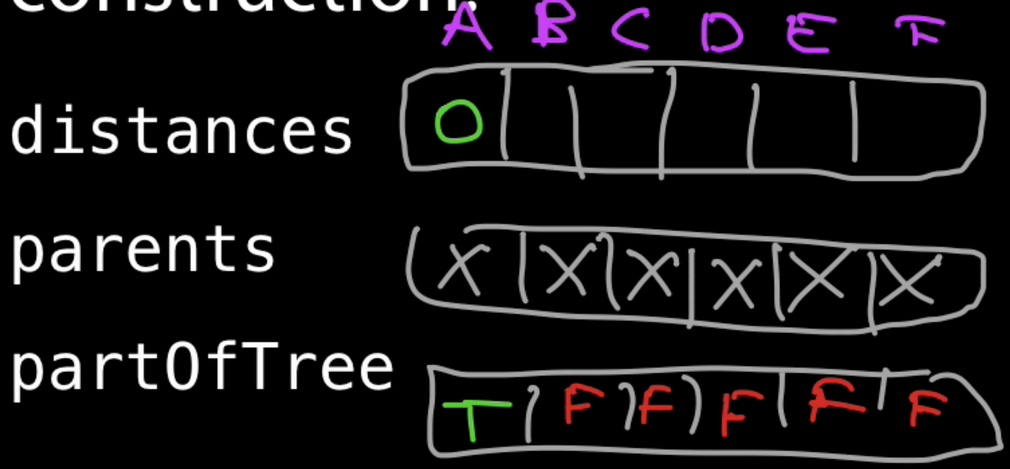
`parents`

`partOfTree`

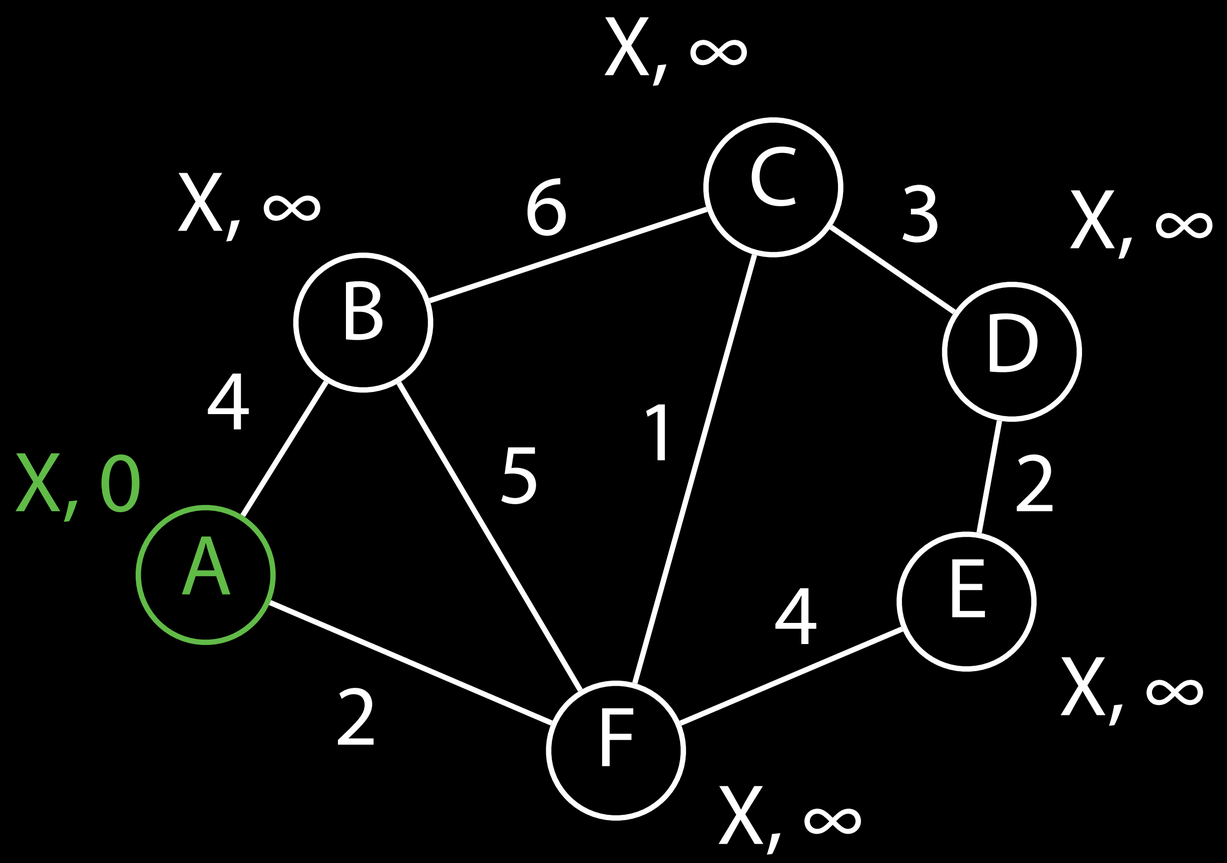
Prim's Algorithm



Repeatedly add the vertex closest to the MST under construction



Prim's Algorithm



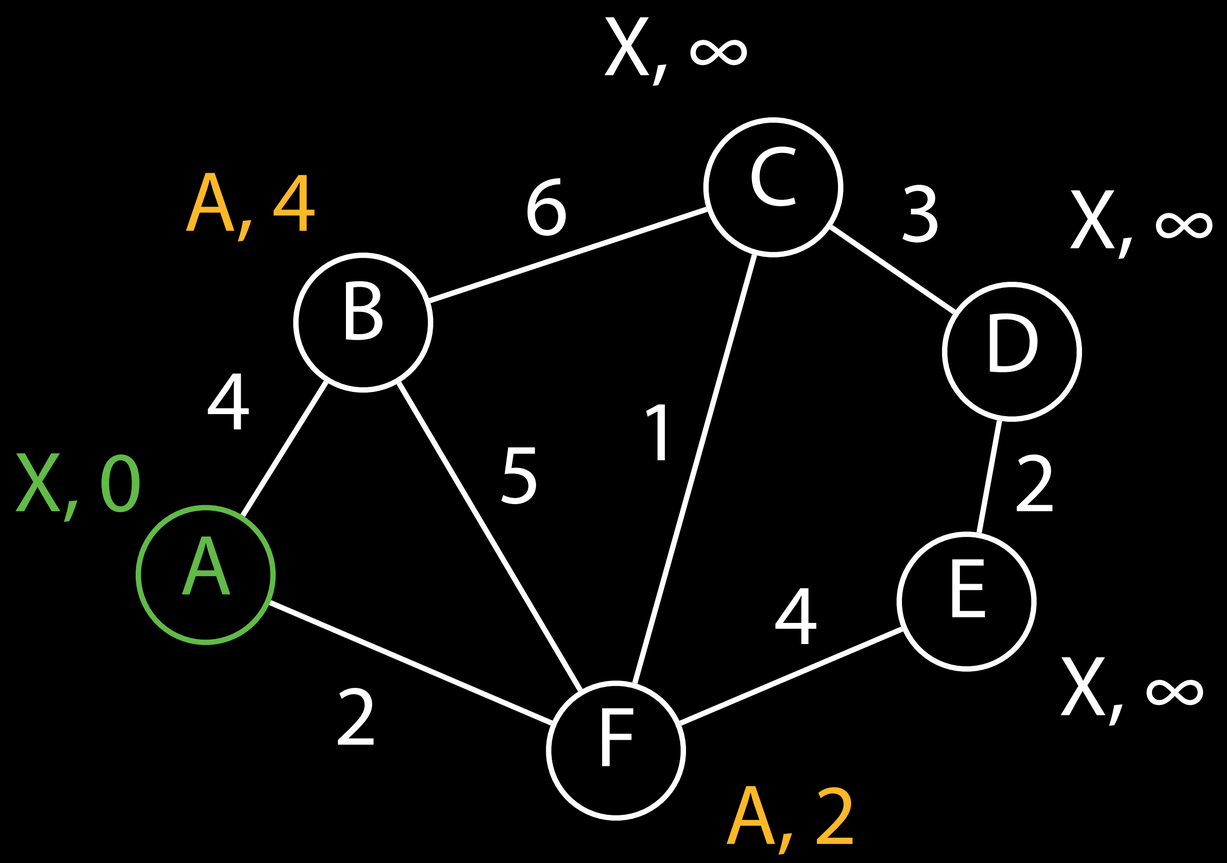
Repeatedly add the vertex closest to the MST under construction.

`distances`

`parents`

`partOfTree`

Prim's Algorithm



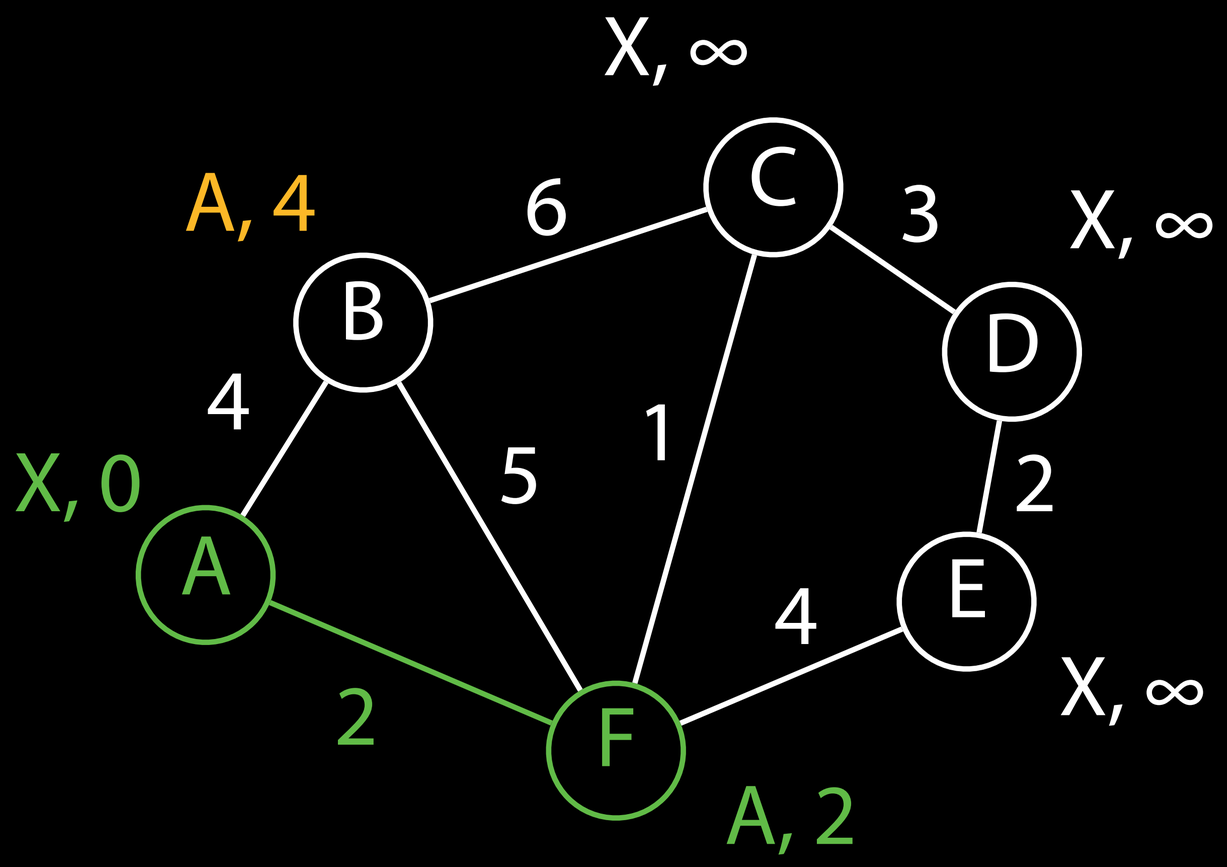
Repeatedly add the vertex closest to the MST under construction.

`distances`

`parents`

`partOfTree`

Prim's Algorithm



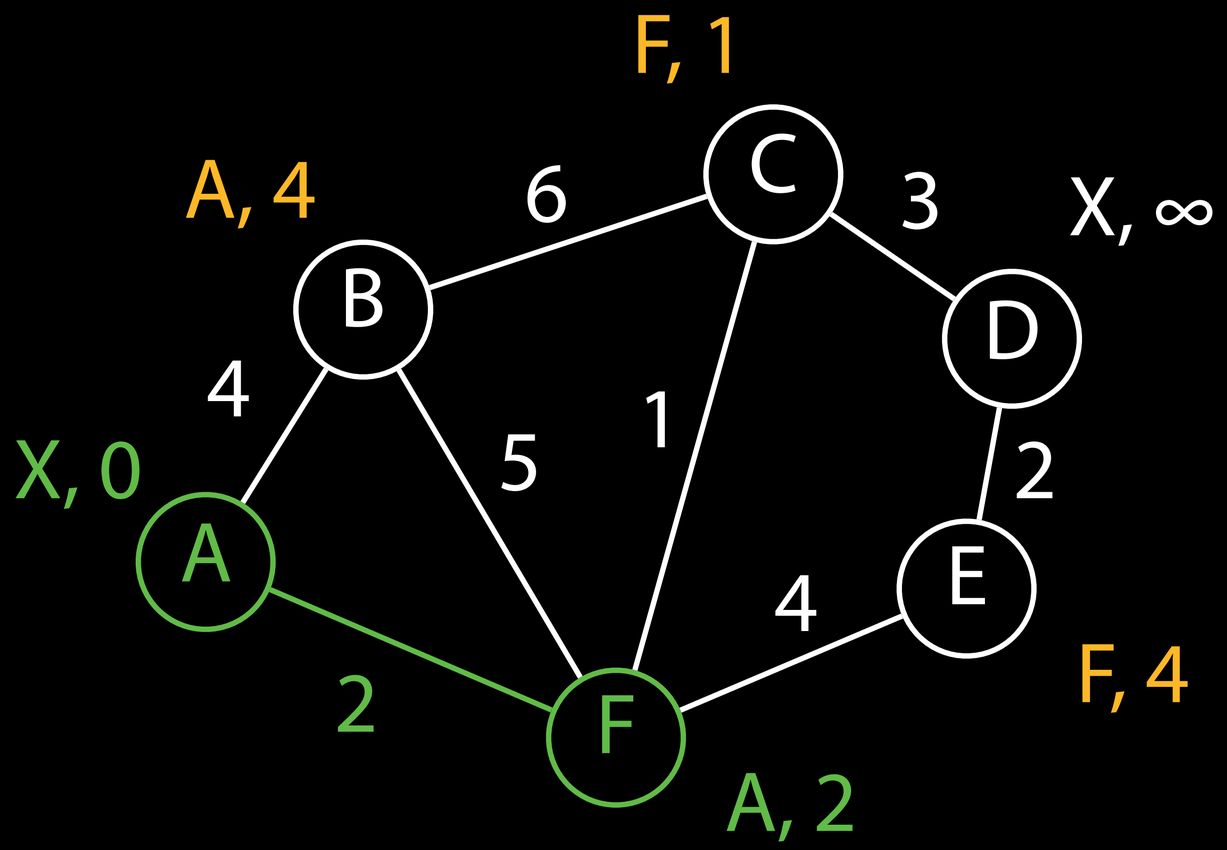
Repeatedly add the vertex closest to the MST under construction.

`distances`

`parents`

`partOfTree`

Prim's Algorithm



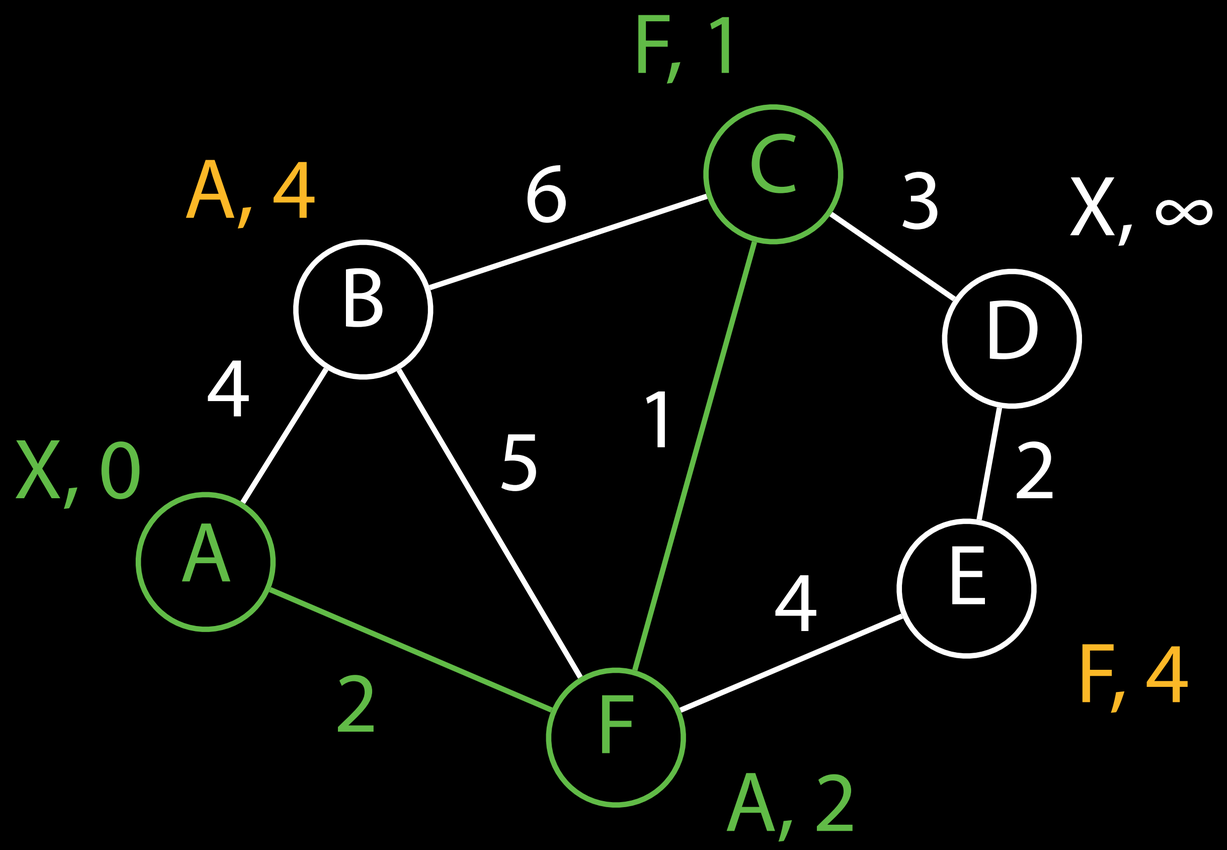
Repeatedly add the vertex closest to the MST under construction.

`distances`

`parents`

`partOfTree`

Prim's Algorithm



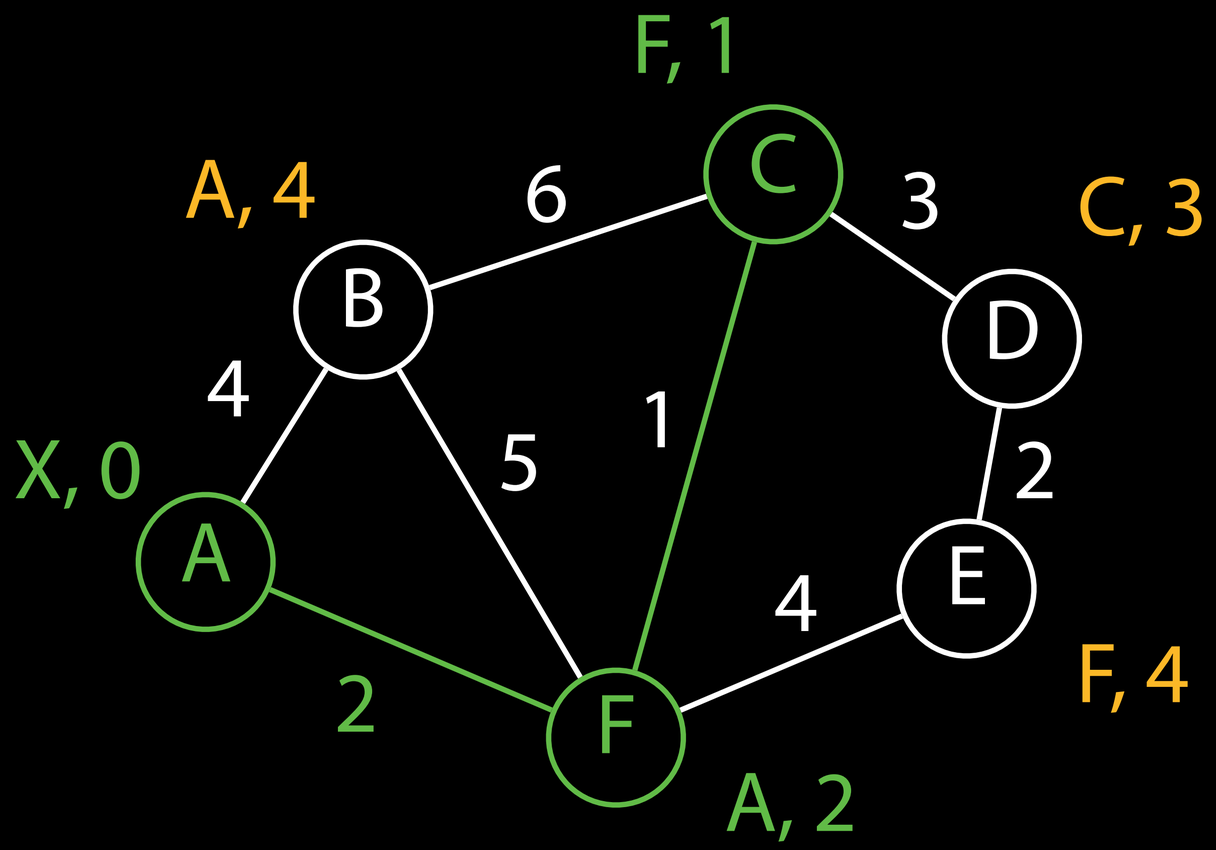
Repeatedly add the vertex closest to the MST under construction.

`distances`

`parents`

`partOfTree`

Prim's Algorithm



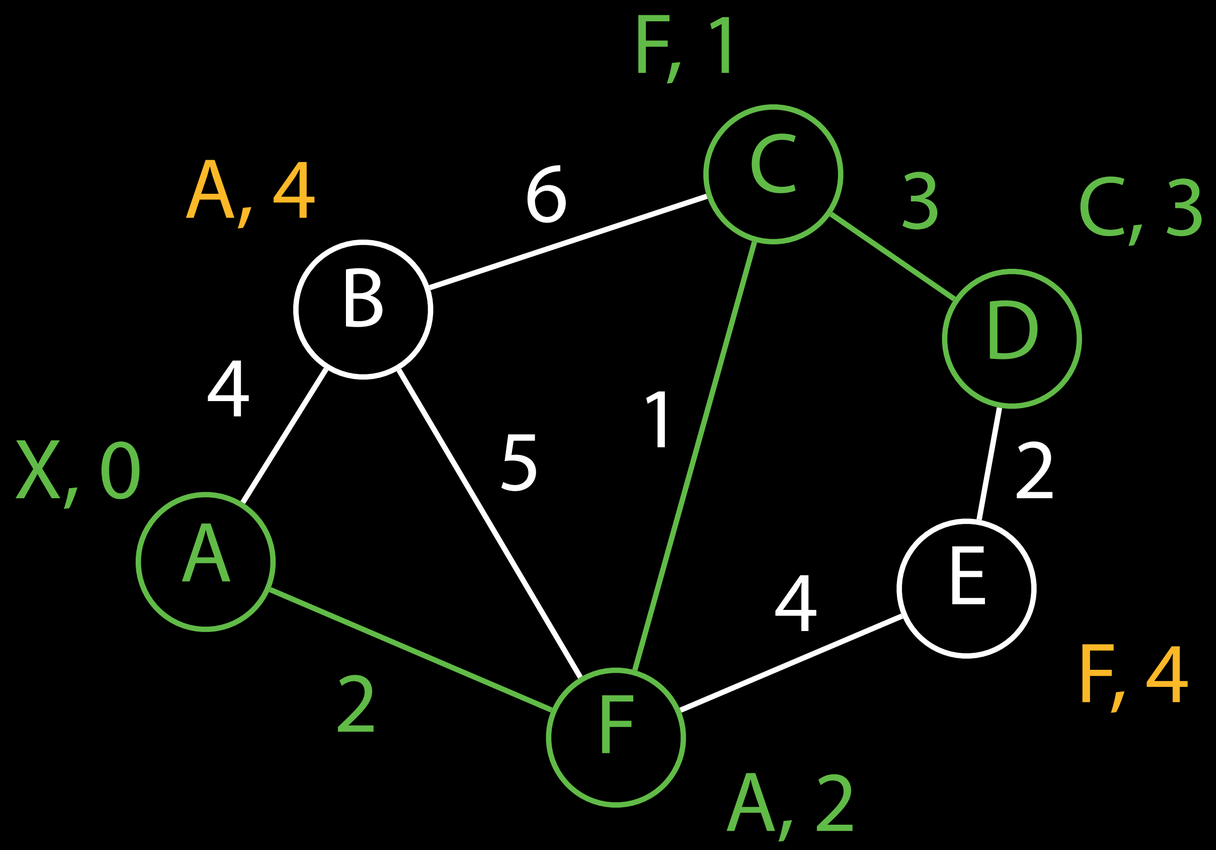
Repeatedly add the vertex closest to the MST under construction.

`distances`

`parents`

`partOfTree`

Prim's Algorithm



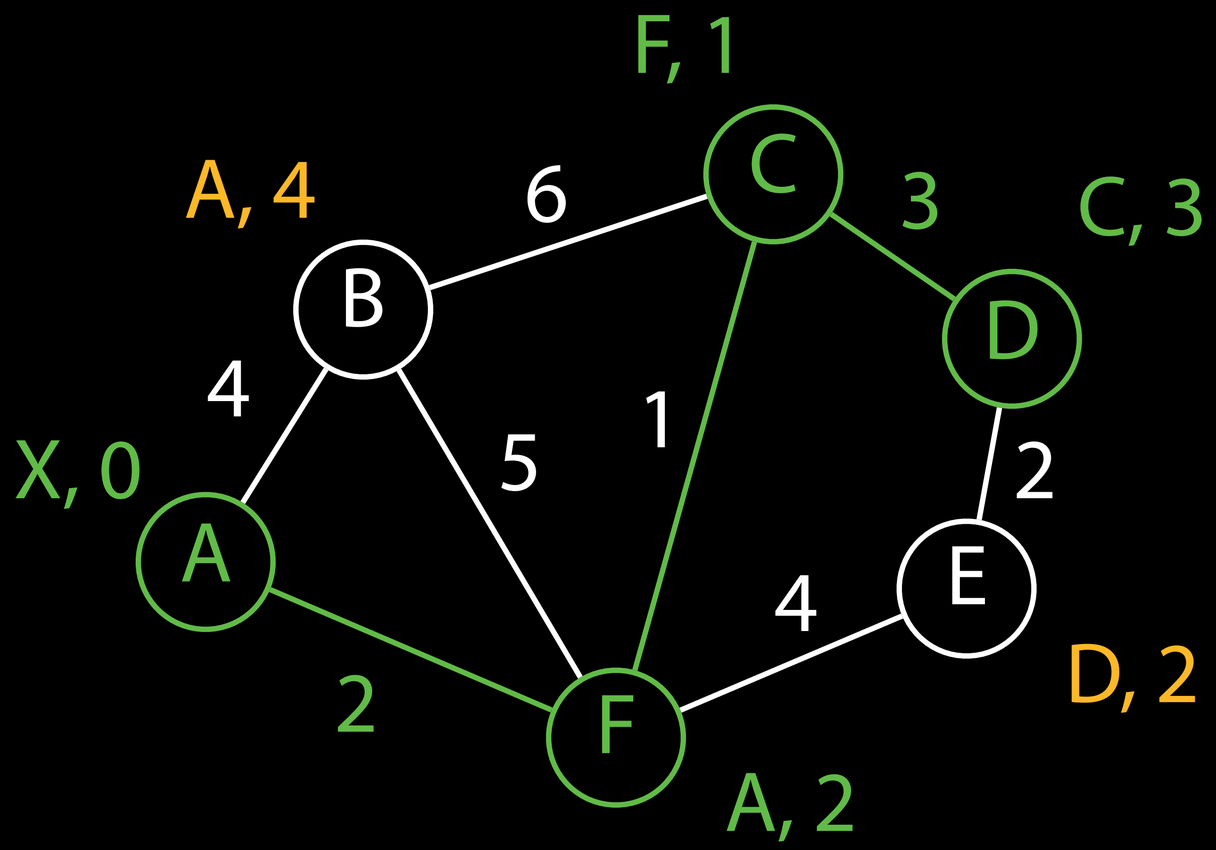
Repeatedly add the vertex closest to the MST under construction.

`distances`

`parents`

`partOfTree`

Prim's Algorithm



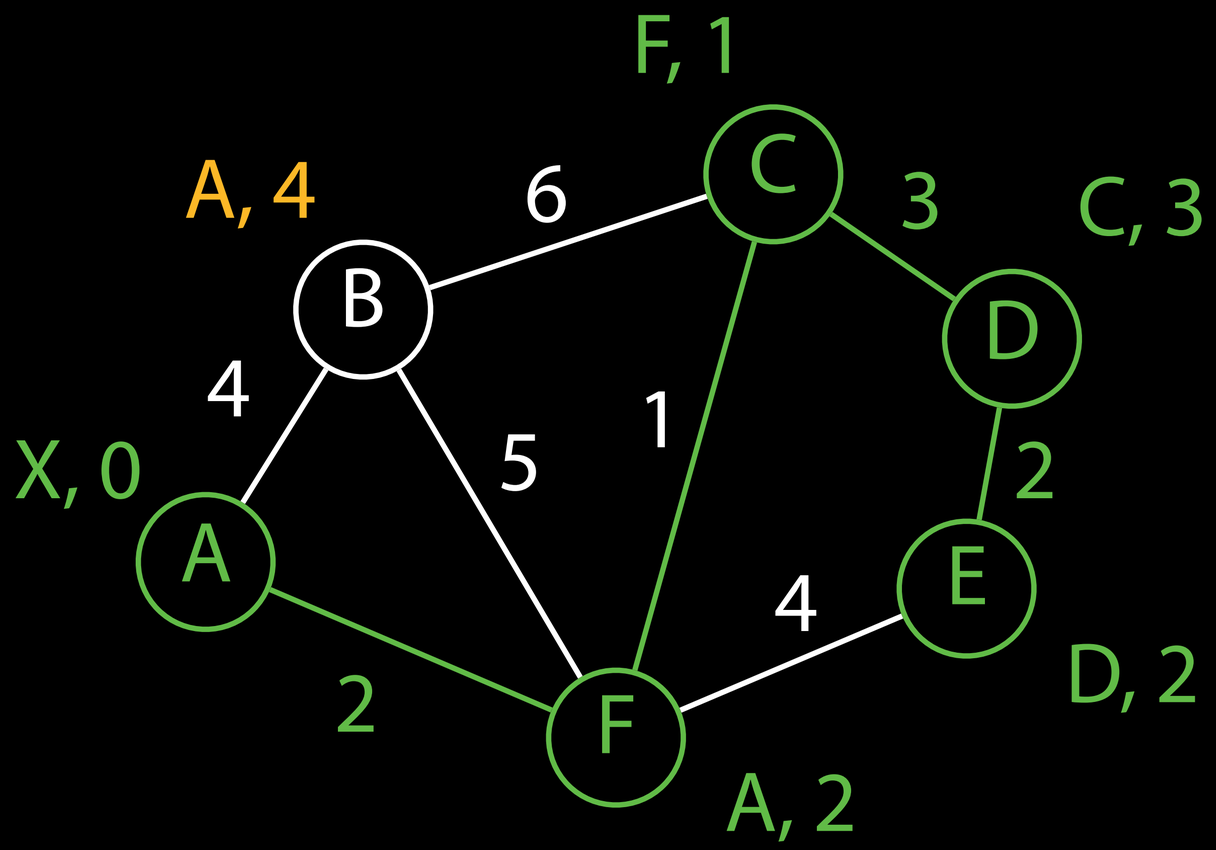
Repeatedly add the vertex closest to the MST under construction.

`distances`

`parents`

`partOfTree`

Prim's Algorithm



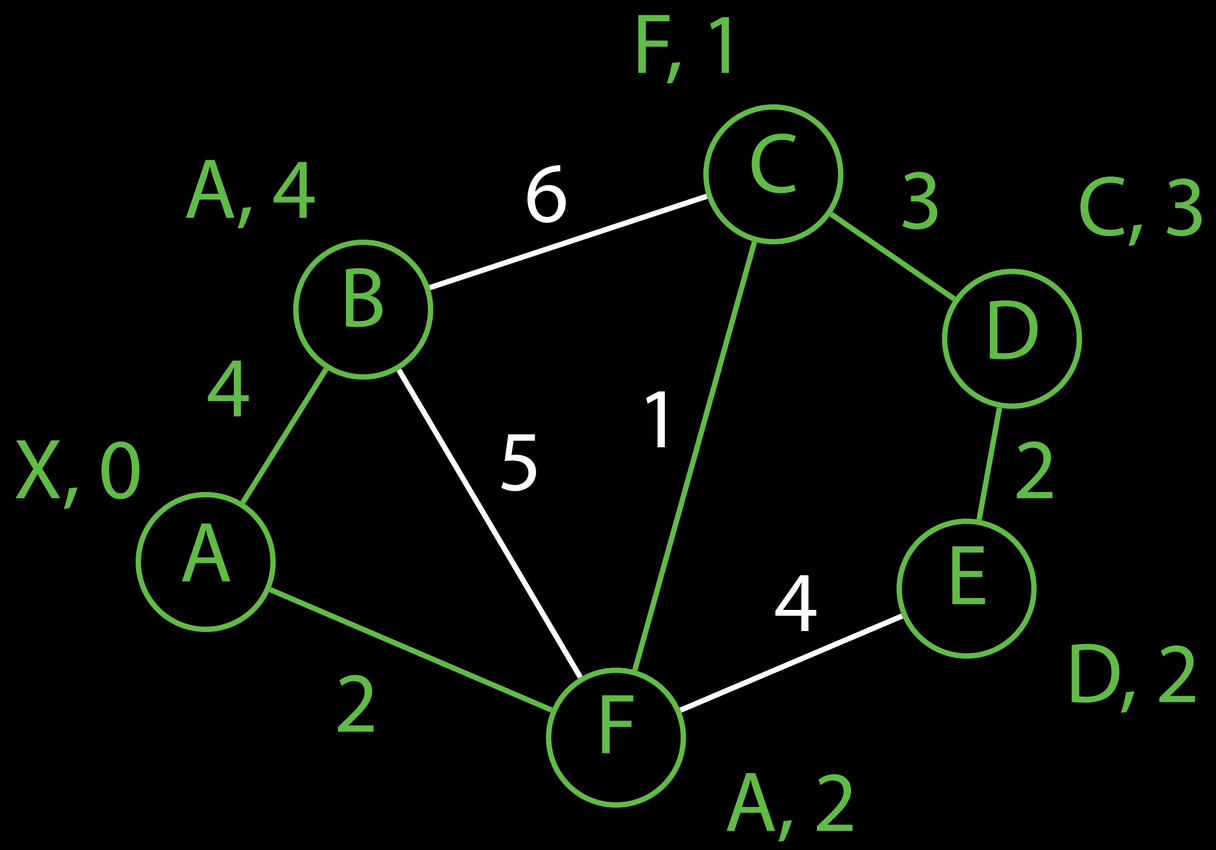
Repeatedly add the vertex closest to the MST under construction.

`distances`

`parents`

`partOfTree`

Prim's Algorithm



Repeatedly add the vertex closest to the MST under construction.

`distances`

`parents`

`partOfTree`

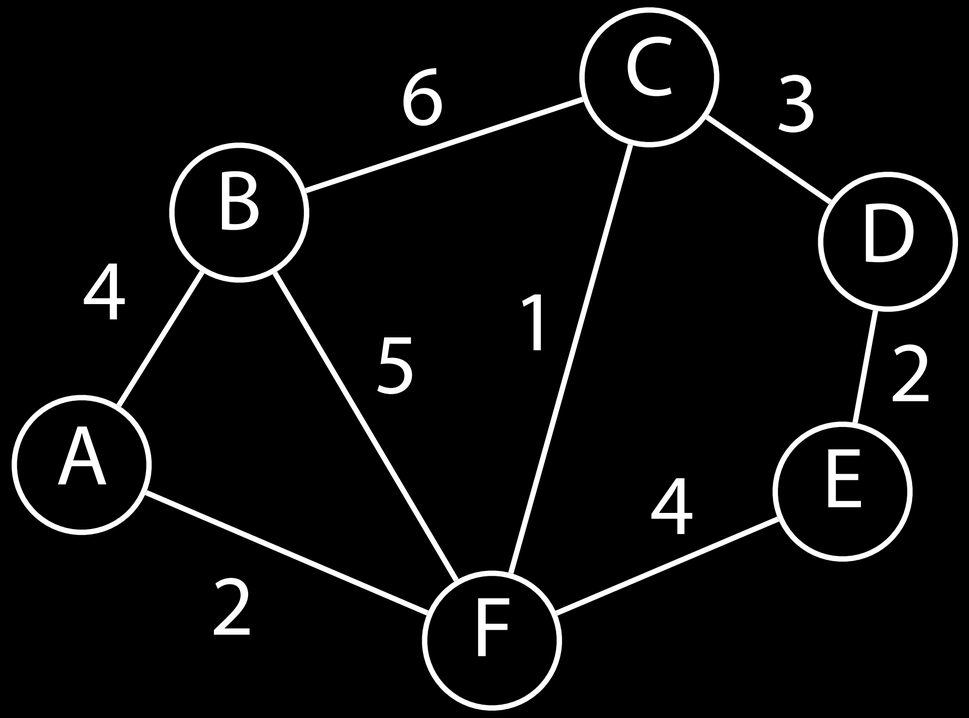
Kruskal's Algorithm

Consider edges in increasing order of weight. (Sort edges.)

For each edge, add edge to MST unless doing so creates a cycle.

Repeat until $V - 1$ edges.

Kruskal's Algorithm



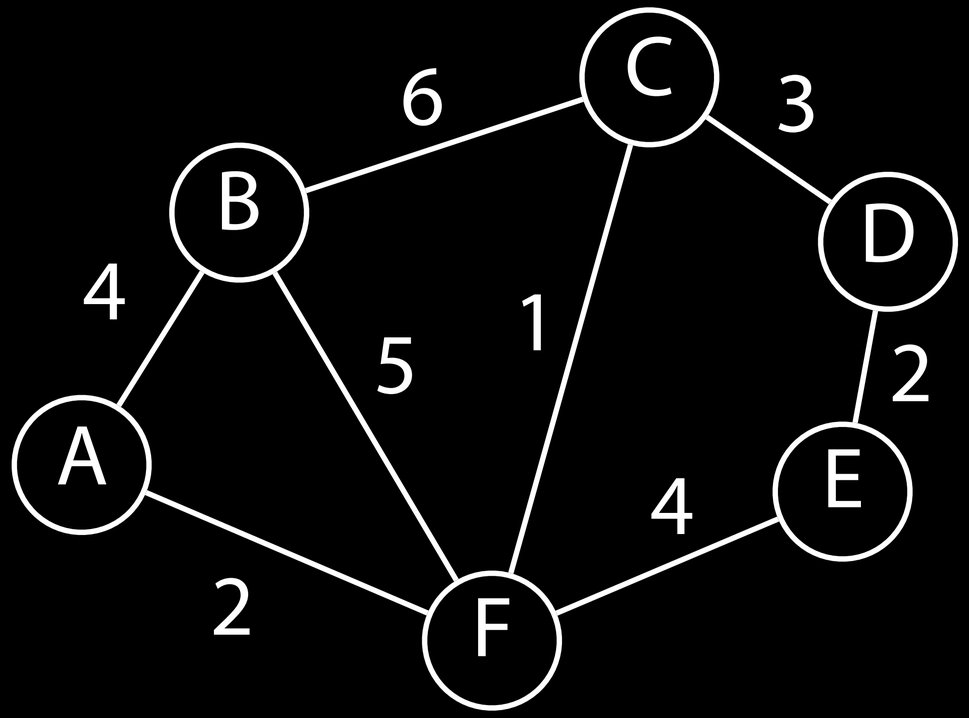
Sort edges by weight.

For each edge, add edge to MST unless doing so creates a cycle.

Edges

A-B, 4	C-D, 3
A-F, 2	C-F, 1
B-C, 6	D-E, 2
B-F, 5	E-F, 4

Kruskal's Algorithm



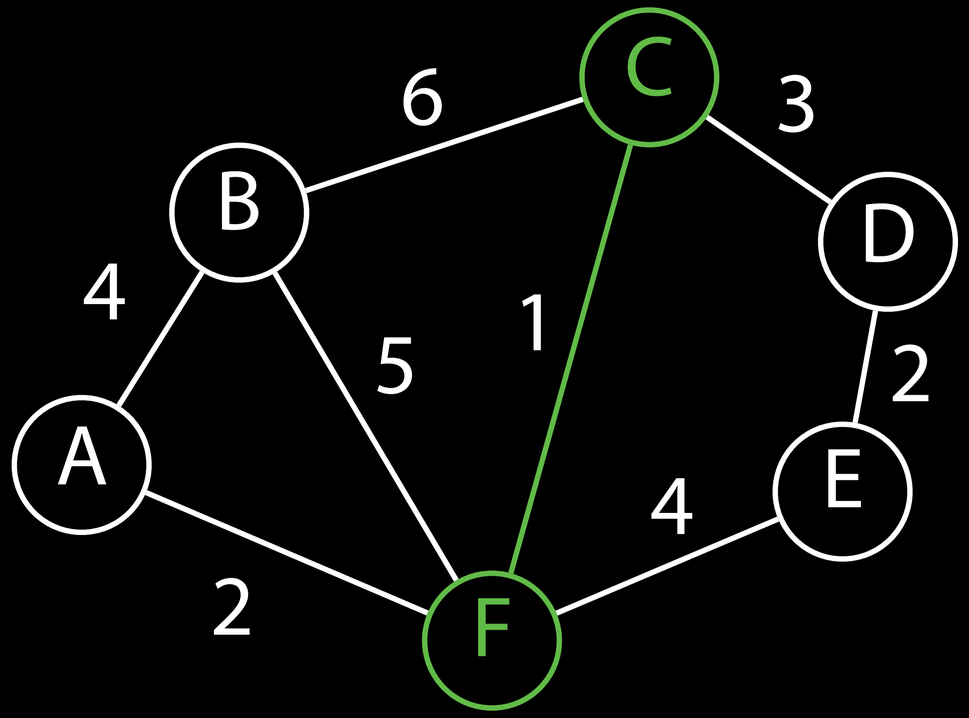
Sort edges by weight.

For each edge, add edge to MST unless doing so creates a cycle.

Edges

C-F, 1	A-B, 4
A-F, 2	E-F, 4
D-E, 2	B-F, 5
C-D, 3	B-C, 6

Kruskal's Algorithm



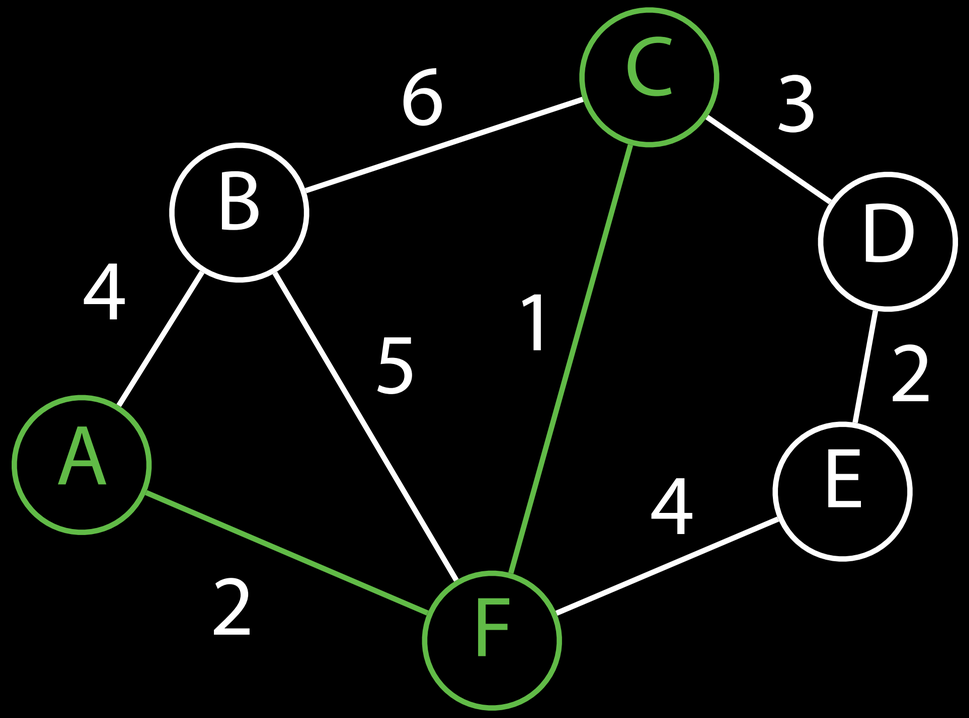
Sort edges by weight.

For each edge, add edge to MST unless doing so creates a cycle.

Edges

C-F, 1	A-B, 4
A-F, 2	E-F, 4
D-E, 2	B-F, 5
C-D, 3	B-C, 6

Kruskal's Algorithm



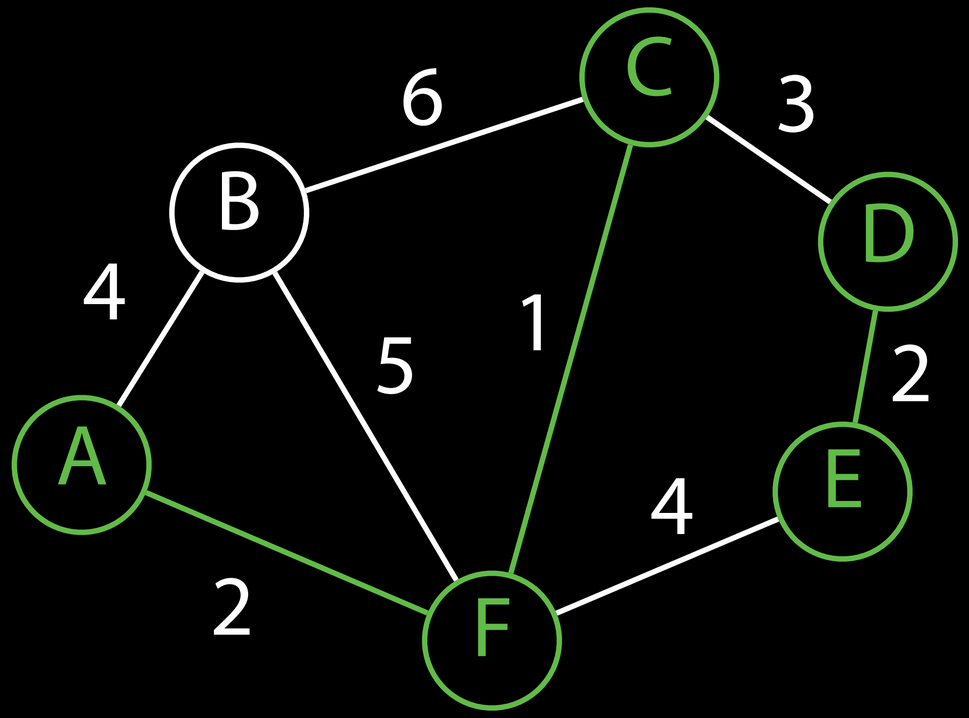
Sort edges by weight.

For each edge, add edge to MST unless doing so creates a cycle.

Edges

C-F, 1	A-B, 4
A-F, 2	E-F, 4
D-E, 2	B-F, 5
C-D, 3	B-C, 6

Kruskal's Algorithm



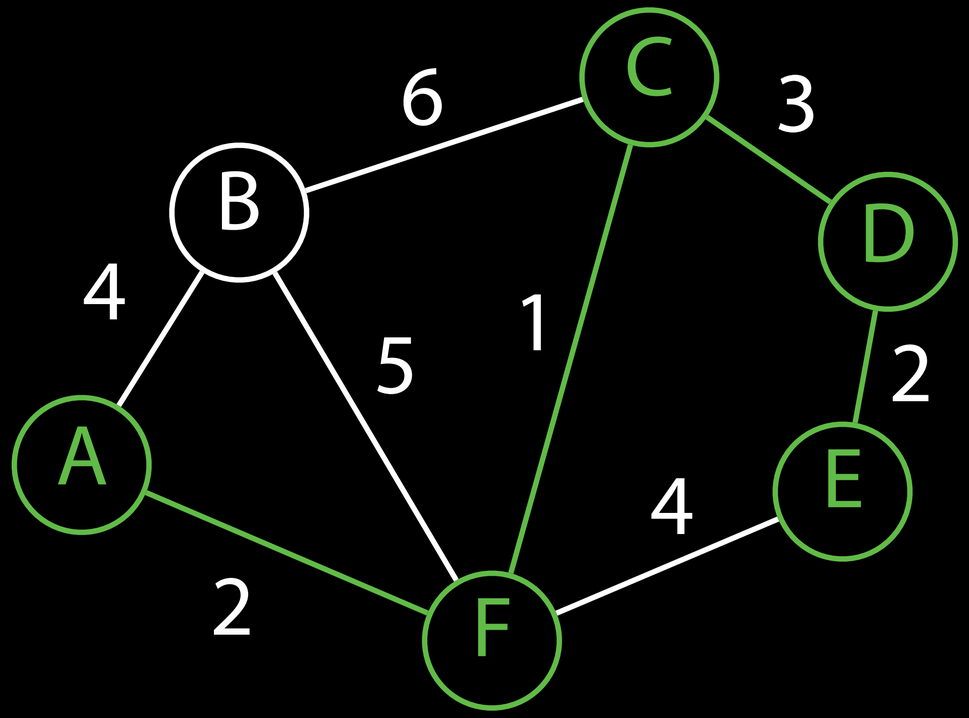
Sort edges by weight.

For each edge, add edge to MST unless doing so creates a cycle.

Edges

C-F, 1	A-B, 4
A-F, 2	E-F, 4
D-E, 2	B-F, 5
C-D, 3	B-C, 6

Kruskal's Algorithm



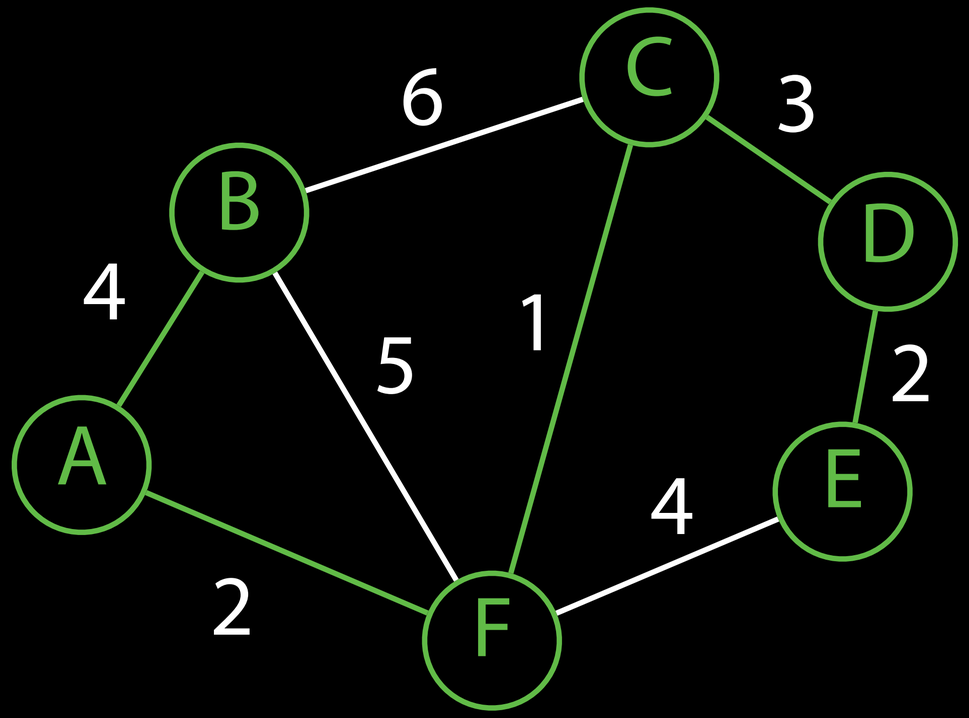
Sort edges by weight.

For each edge, add edge to MST unless doing so creates a cycle.

Edges

C-F, 1	A-B, 4
A-F, 2	E-F, 4
D-E, 2	B-F, 5
C-D, 3	B-C, 6

Kruskal's Algorithm



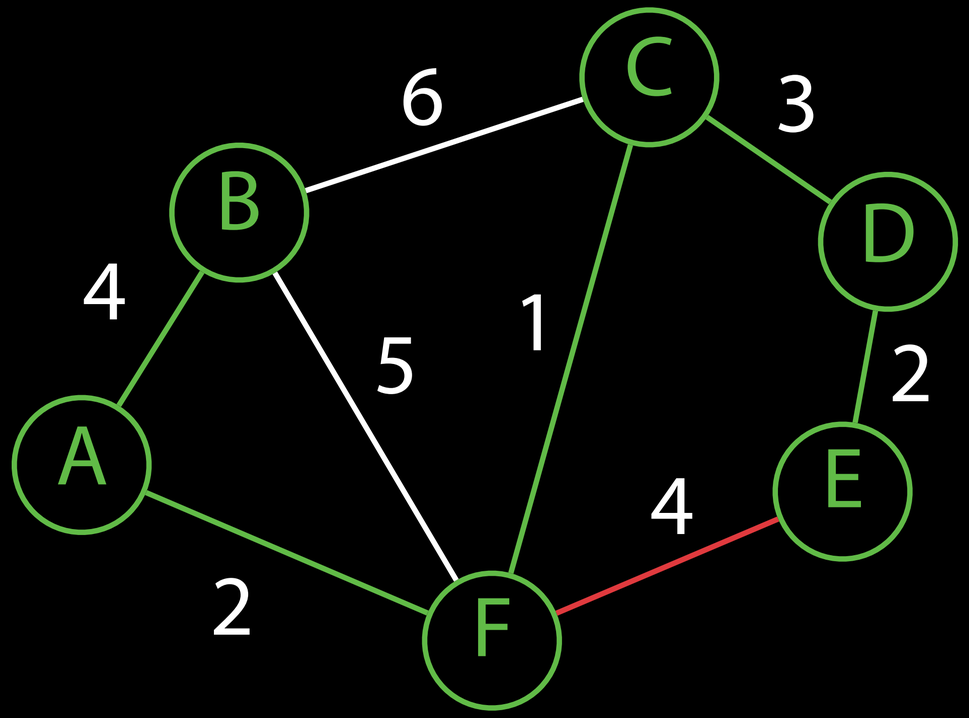
Sort edges by weight.

For each edge, add edge to MST unless doing so creates a cycle.

Edges

C-F, 1	A-B, 4
A-F, 2	E-F, 4
D-E, 2	B-F, 5
C-D, 3	B-C, 6

Kruskal's Algorithm



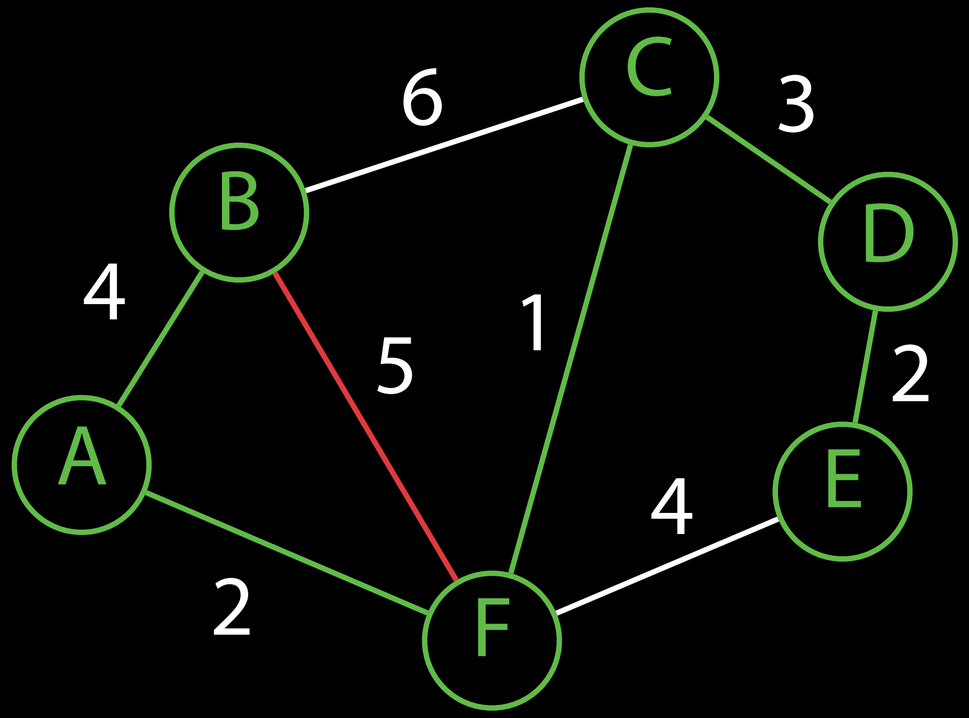
Sort edges by weight.

For each edge, add edge to MST unless doing so creates a cycle.

Edges

C-F, 1	A-B, 4
A-F, 2	E-F, 4
D-E, 2	B-F, 5
C-D, 3	B-C, 6

Kruskal's Algorithm



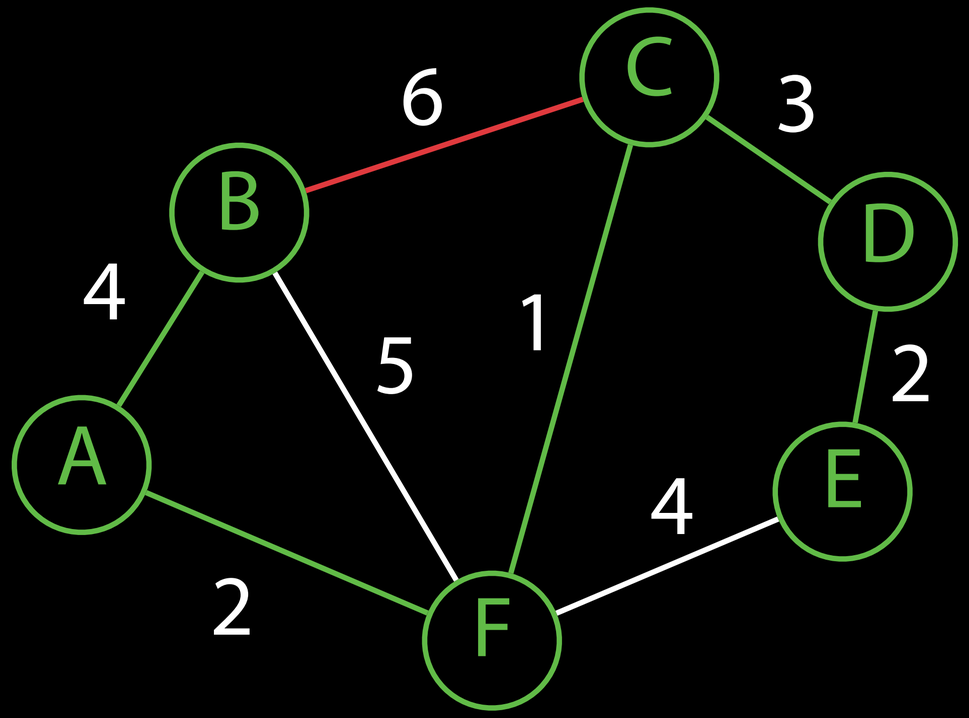
Sort edges by weight.

For each edge, add edge to MST unless doing so creates a cycle.

Edges

C-F, 1	A-B, 4
A-F, 2	E-F, 4
D-E, 2	B-F, 5
C-D, 3	B-C, 6

Kruskal's Algorithm



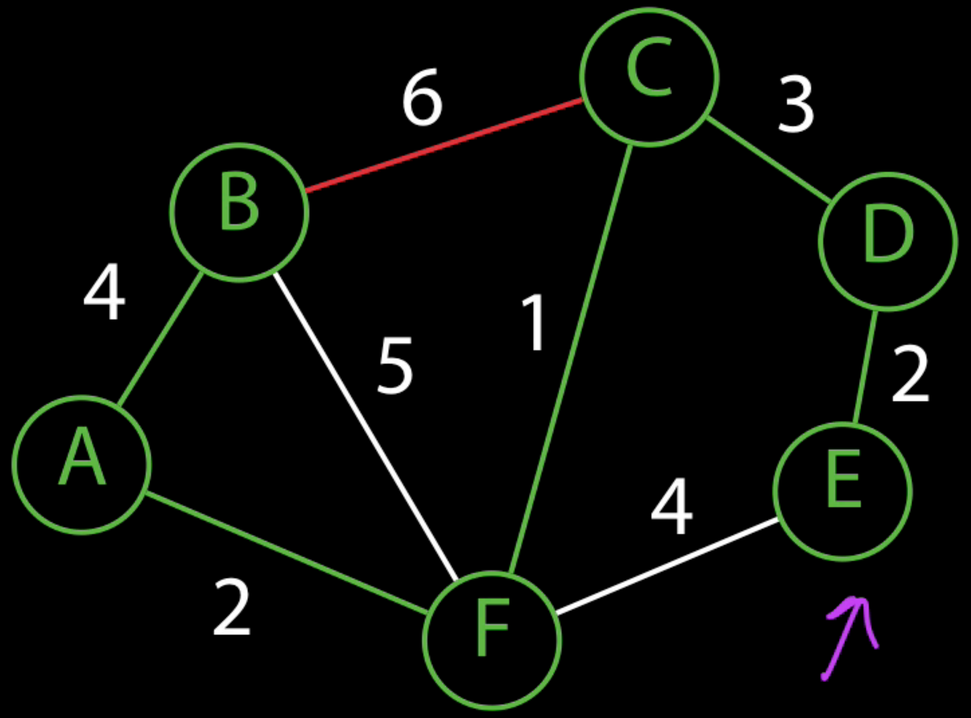
Sort edges by weight.

For each edge, add edge to MST unless doing so creates a cycle.

Edges

C-F, 1	A-B, 4
A-F, 2	E-F, 4
D-E, 2	B-F, 5
C-D, 3	B-C, 6

Kruskal's Algorithm

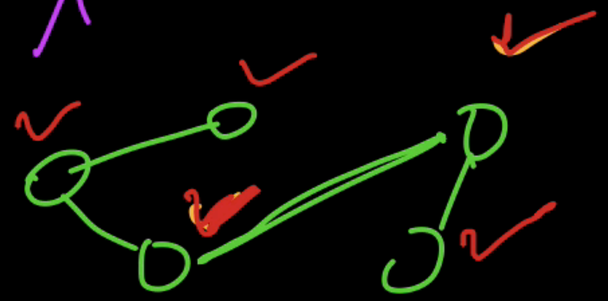


Sort edges by weight.

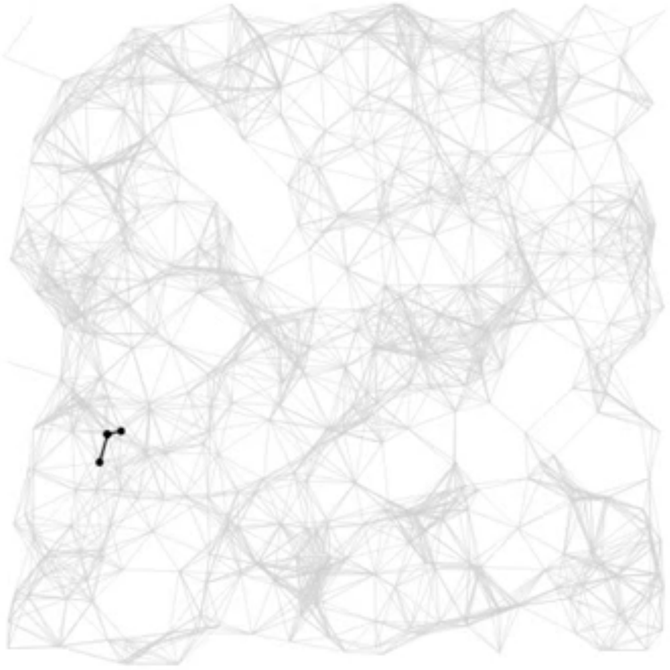
For each edge, add edge to MST unless doing so creates a cycle.

Edges

C-F, 1	A-B, 4
A-F, 2	E-F, 4
D-E, 2	B-F, 5
C-D, 3	B-C, 6



Prim’s vs. Kruskal’s



Prim’s

Kruskal’s

Algorithm Design

Algorithm Design

Brute force

Greedy

Divide and conquer

Backtracking

Branch and Bound

Dynamic programming

Brute force

Generates every possible answer and selects only the valid ones

Usually runs in exponential time

Requires a generation of each answer such as with a permutation-generation function or a tree-traversal function

Examples:
Pick all subsets of numbers and see which add up to a given sum
Pick all routes in the travelling salesman problem and choose the best.

Schedule
Assignments
Grades
People
Calendar
Files
Quizzes
U-M Course Manager
Piazza
Lecture Recordings
Autograder 1
Autograder 2
Gradescope
Teaching Evaluations
Engineering Honor Code
Alternate Exam Request

Tree Questions for Tree Points

3 points. For each question, select all correct answers.

Question 1 — 1 pts

Consider the *Tree Sort* algorithm in which sorting is achieved by placing all elements to be sorted into a binary search tree and then taking the in-order traversal of the tree. Which of the following statements are true about Tree Sort?

- [ ] Tree Sort performs well on input that is already in nearly sorted order.
- [ ] The worst case time complexity of Tree Sort is $\Theta(n \log n)$.
- [ ] Using an AVL tree (i.e. "AVL Sort") instead of a regular binary search tree improves the worst case complexity.
- [ ] Tree Sort is stable, as long as the right policy is used for equally valued elements.
- [ ] None of the above.

Power function

double pow(double, double)

pow(2, 3) → 8

Restrict to ints

x^y

Brute force:

$\underbrace{x \cdot x \cdot x \cdots x}_{y}$

$\Theta(y)$

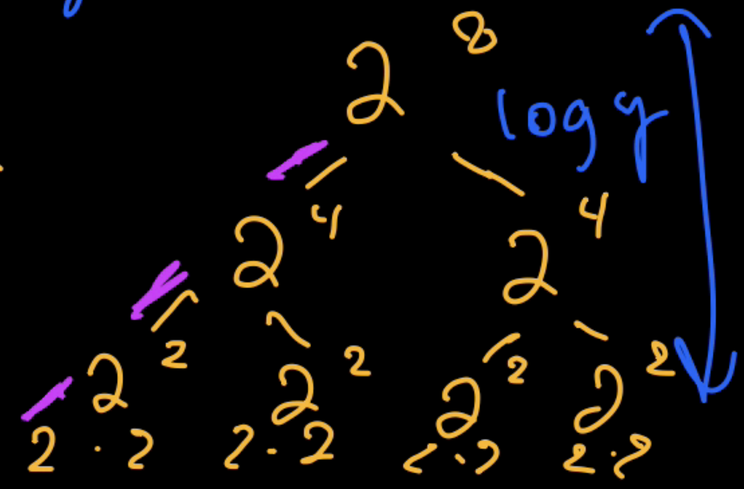
2^8

Better:

$x^{y/2} \cdot x^{y/2}$

$\Theta(\log y)$

$2^8 = 2^4 \cdot 2^4$



Greedy

Picks the “best” next partial solution from the current partial solution at every step, by some meaning of “best”.

Usually visits each node once (often this translates to linear complexity, but not always).

May not always produce an optimal solution.

Examples: MST algorithms like Prim’s and Kruskal’s, making change using U.S. coins

Greedy

Picks the "best" next partial solution from the current partial solution at every step, by some meaning of "best".

Never change our mind

Usually visits each node once (often this translates to linear complexity, but not always).

May not always produce an optimal solution.

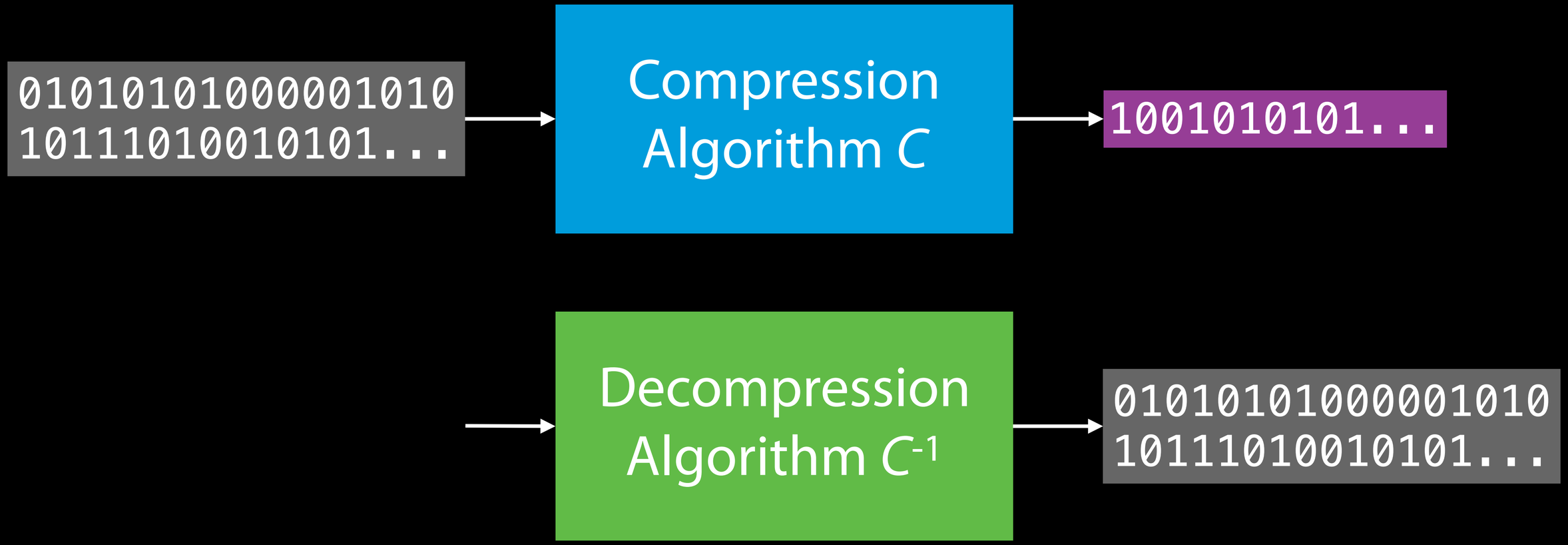
Examples: MST algorithms like Prim's and Kruskal's, making change using U.S. coins

Silicon Valley

http://www.hbo.com/silicon-valley

Compression

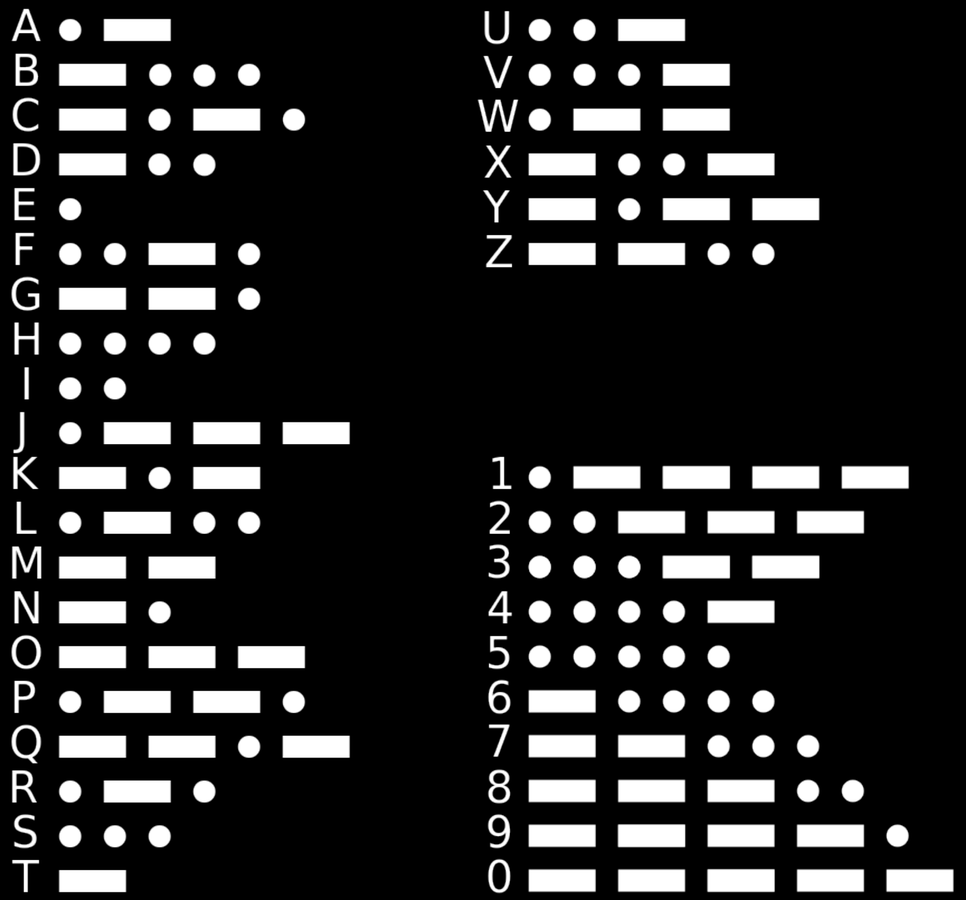
Lossless Compression



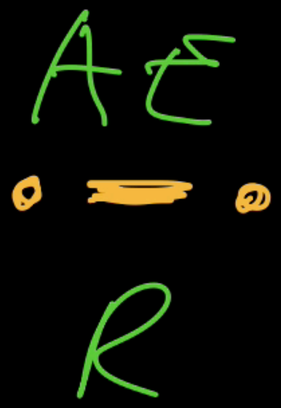
ASCII

Character	Decimal	Binary	Hexadecimal
A	65	1000001	41
B	66	1000010	42
C	67	1000011	43
D	68	1000100	44
E	69	1000101	45

Morse code



Morse code



Prefix Property

```
A   0
B   1
C   01
D   11
```

Prefix Property

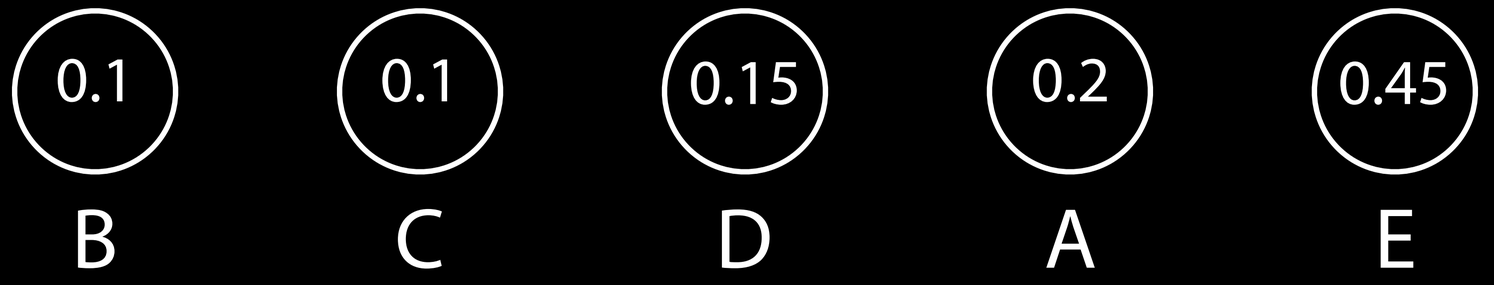
A	0
B	1
C	01
D	11

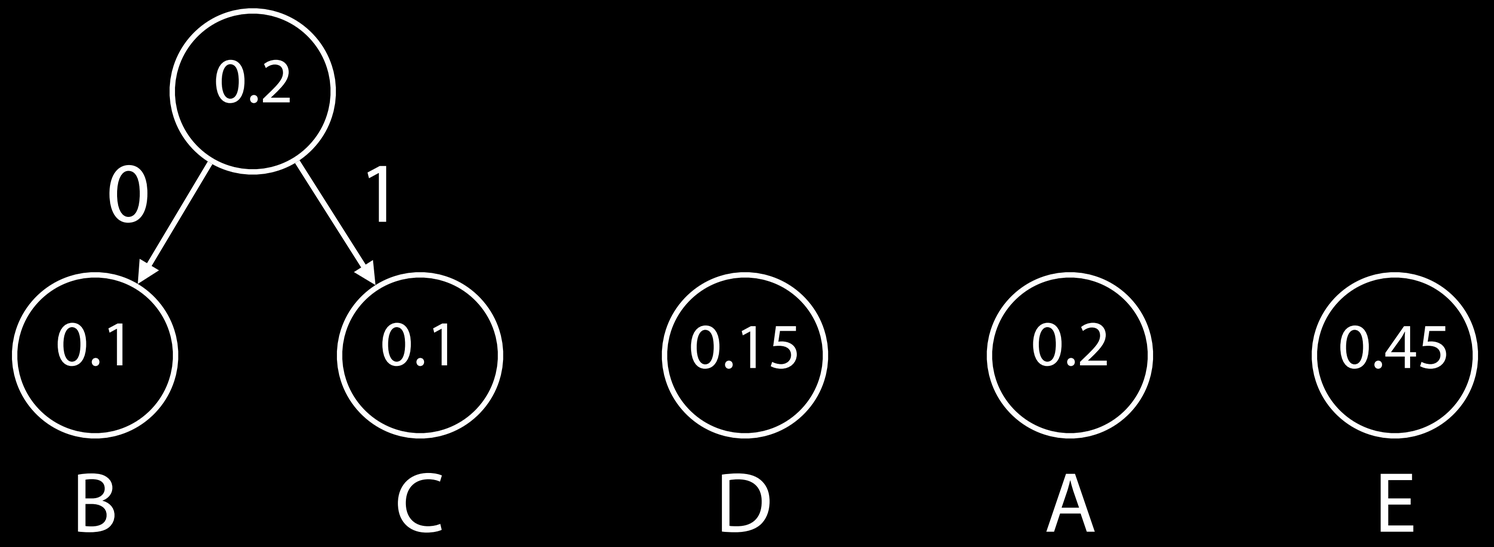
A	0
B	10
C	110
D	111

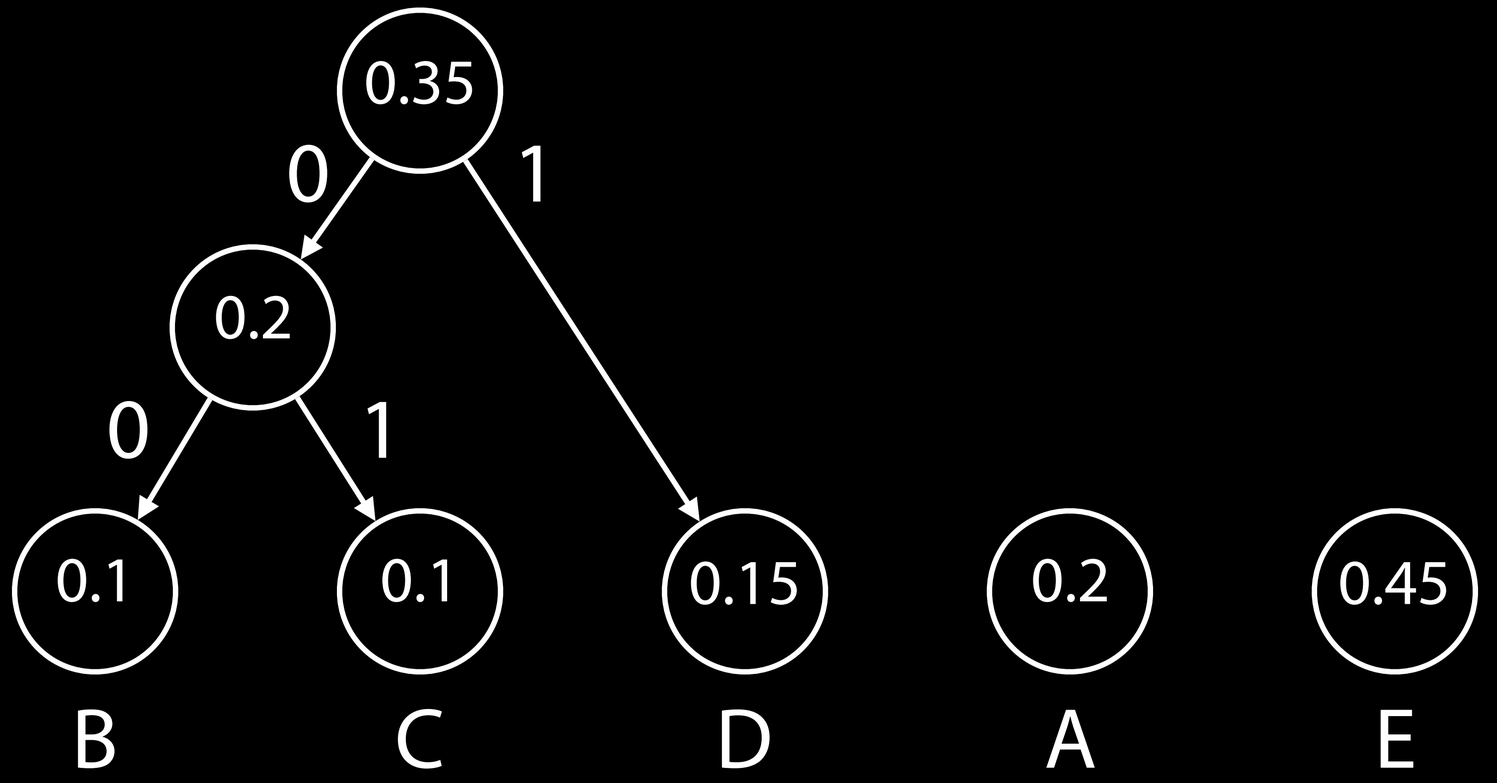
Huffman coding

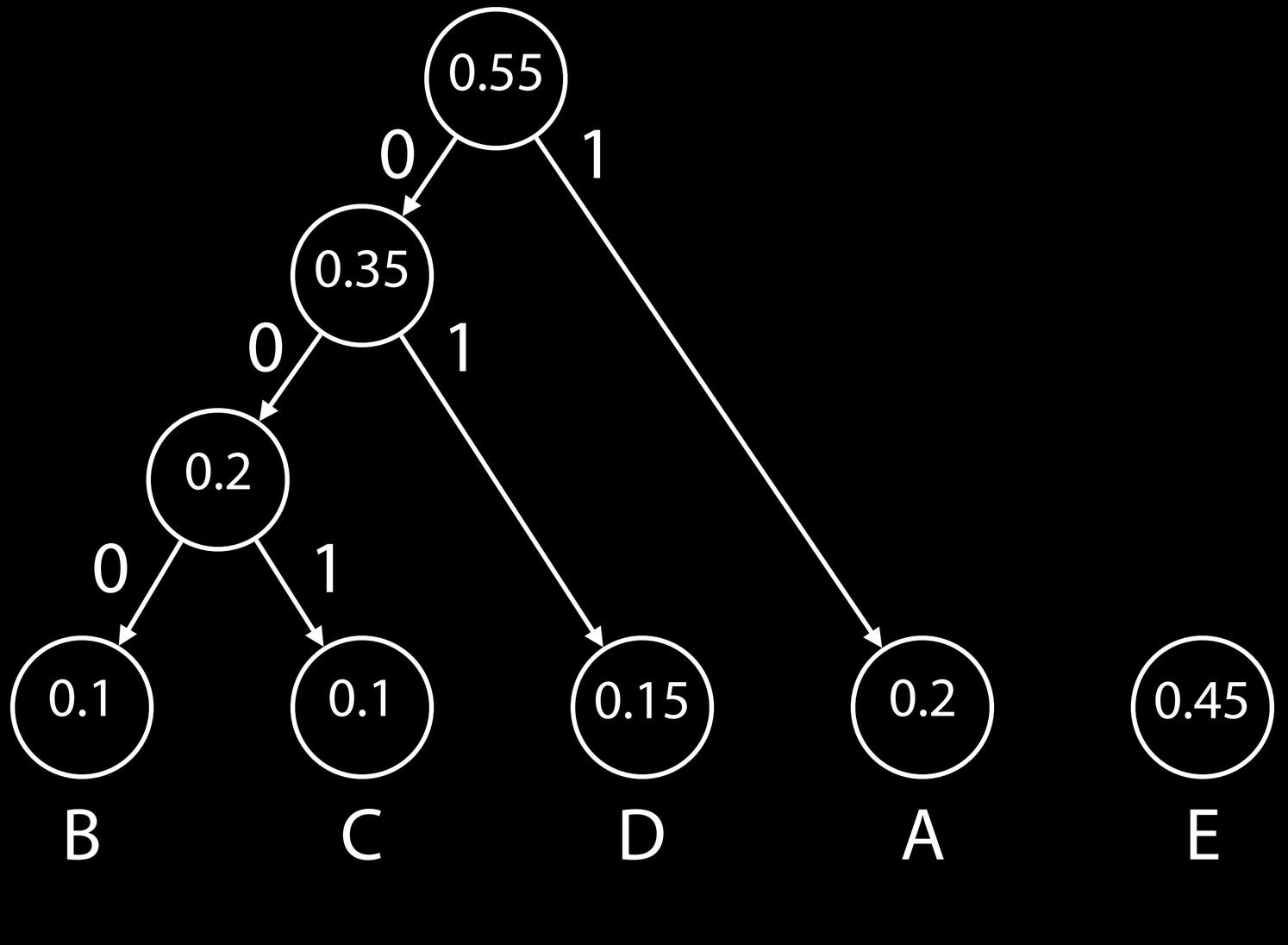
ECEABEADCAEFEEEECEADEEEEEDBAAEABDB
BAAEAAACDDCCEABEEDCDEEDEAEEEEEAEED
BCEBEEADEAEEDAEBCDEDEAEEDCEEAEEE

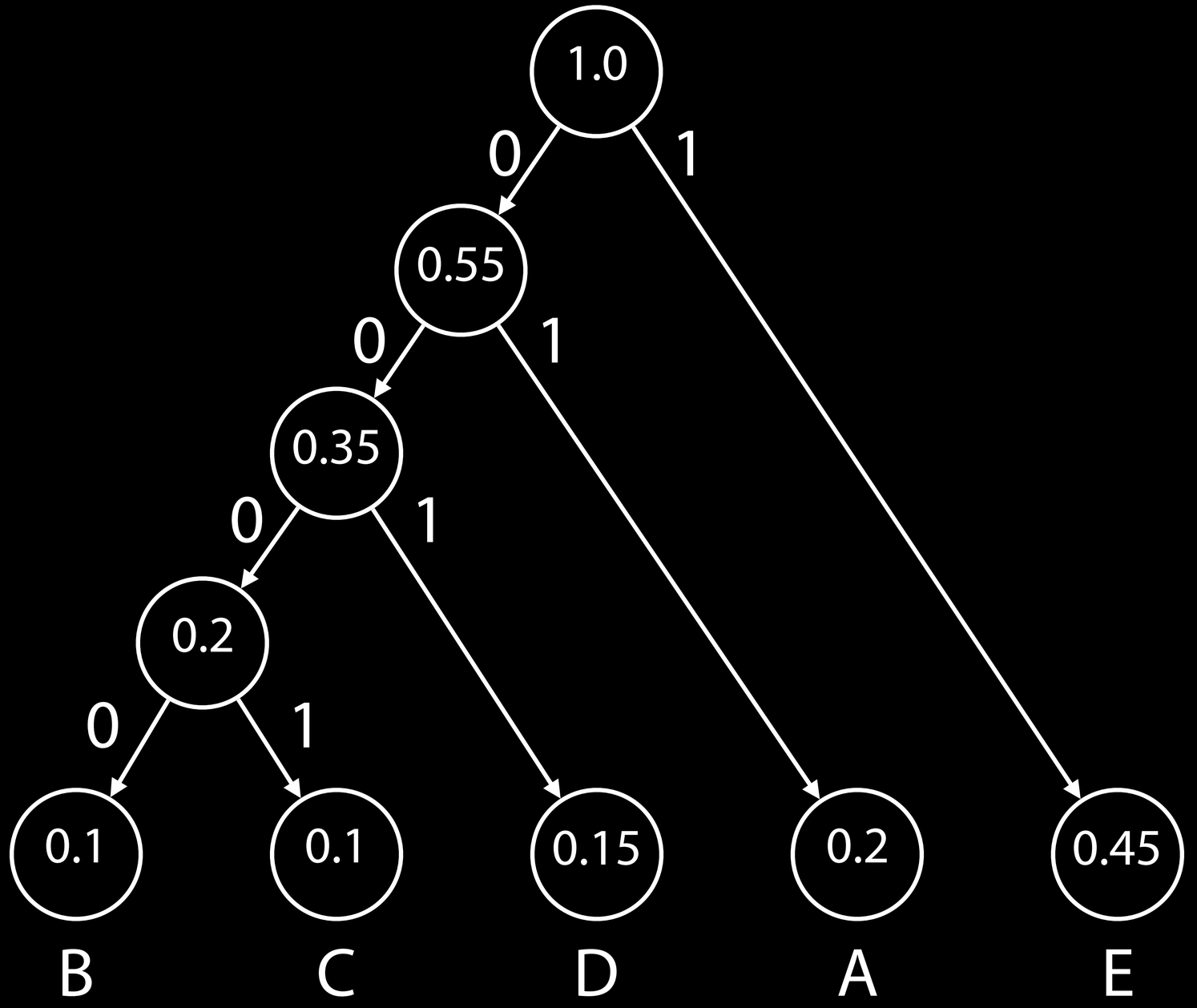
character	A	B	C	D	E
frequency	0.2	0.1	0.1	0.15	0.45

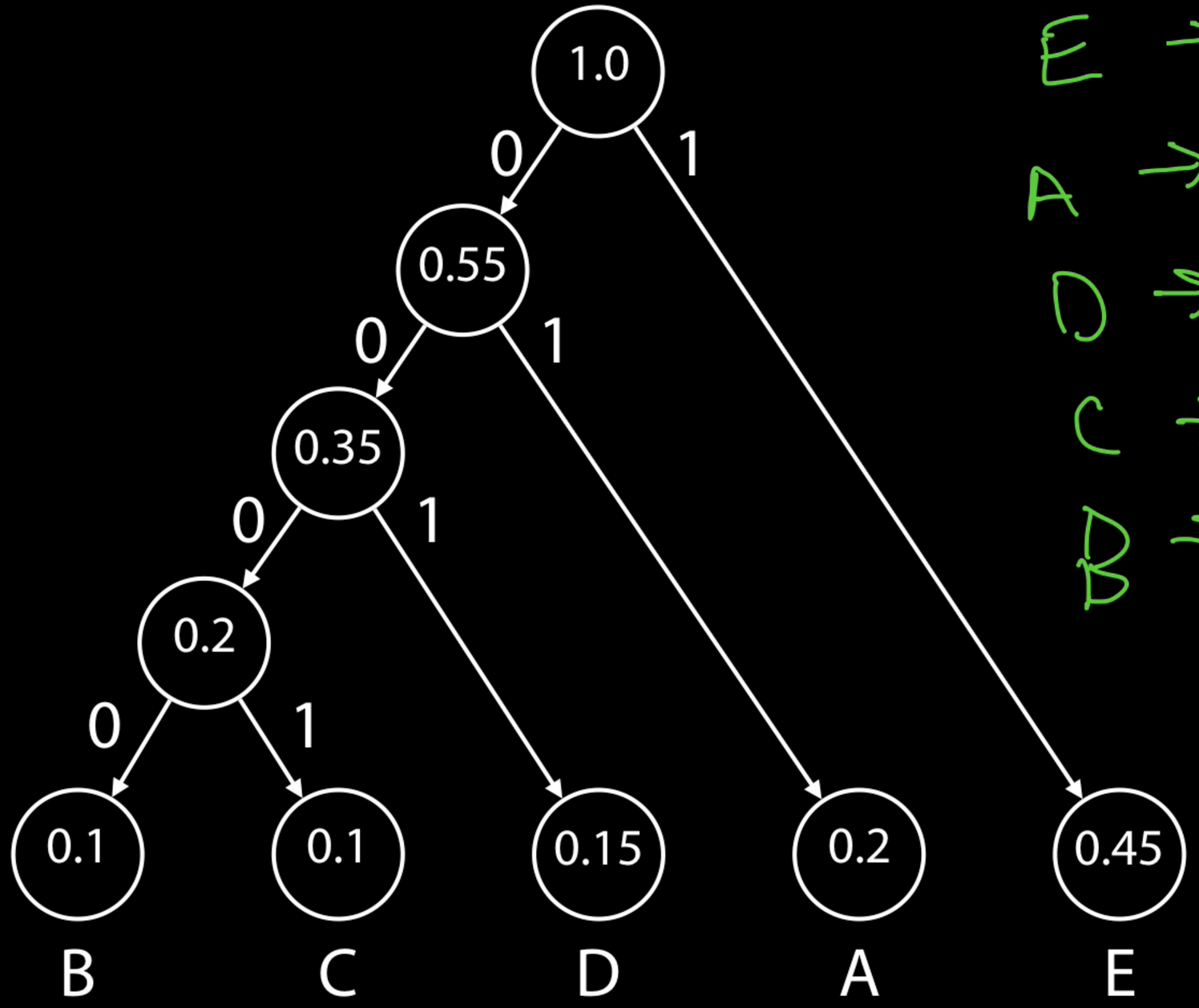

0.1
0.1
0.15
0.2
0.45
B
C
D
A
E


0.2
0
1
0.1
0.1
0.15
0.2
0.45
B
C
D
A
E


0.35
0
1
0.2
0
1
0.1
0.1
0.15
0.2
0.45
B
C
D
A
E


0.55
0
1
0.35
0
1
0.2
0
1
0.1
0.1
0.15
0.2
0.45
B
C
D
A
E


1.0
0
1
0.55
0
1
0.35
0
1
0.2
0
1
0.1
0.1
0.15
0.2
0.45
B
C
D
A
E



E → 1

A → 01

D → 001

C → 0001

B → 0000

Huffman coding

Exploit redundancy and existing order inside the sequence.

Sequences with no existing redundancy or order may get expanded.

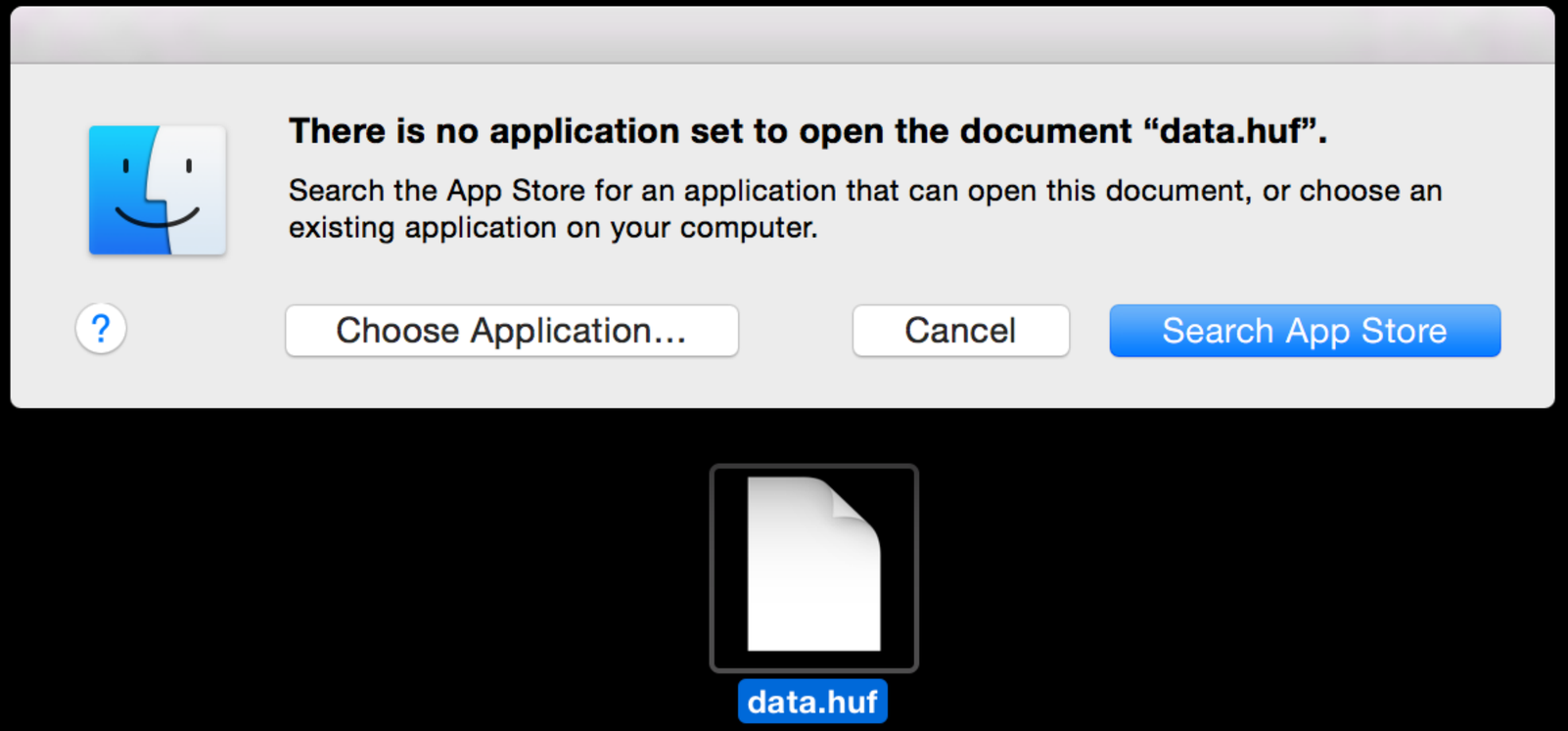
SuperZip

Suppose an algorithm designer says their algorithm SuperZip can compress any bitstream by 50%. Why is this impossible?

SuperZip

Suppose an algorithm designer says their algorithm SuperZip can compress any bitstream by 50%. Why is this impossible?

↳ eventually 1 bit


There is no application set to open the document “data.huf”.
Search the App Store for an application that can open this document, or choose an existing application on your computer.
Choose Application…
Cancel
Search App Store
data.huf

Things get weird

```
0587355292:Desktop Maxim$ zip dog.zip dog.txt
  adding: dog.txt (stored 0%)
0587355292:Desktop Maxim$ ls -l dog*
-rw-r--r--  1 Maxim  staff    7 Jun 18 12:57 dog.txt
-rw-r--r--  1 Maxim  staff  171 Jun 18 12:58 dog.zip
```

Q&A