Announcements

- Homework 3 Solutions online
- No homework this week
- Exam 2 on Friday
- W 10:30-12:00 office hours (this week) will be held online at: https://ucsd.zoom.us/j/9296249412

Last Time

- Greedy Algorithms
- Exchange Arguments

Greedy Algorithms

General Algorithmic Technique:

1. Find decision criterion
2. Make best choice according to criterion
3. Repeat until done

Surprisingly, this sometimes works.

Exchange Argument

- Greedy algorithm makes a sequence of decisions D_1, D_2, D_3,…,D_n eventually reaching solution G.
- Need to show that for arbitrary solutions A that G ≥ A.
- Find sequence of solutions
  $A=A_0$, A_1, A_2,…,A_n = G
  so that:
  - $A_i \leq A_{i+1}$
  - A_i agrees with D_1,D_2,…,D_i

Exchange Argument

In particular, we need to show that given any A_i consistent with $D_1,\ldots,D_i$ we can find an A_{i+1} so that:

- A_{i+1} is consistent with $D_1,\ldots,D_{i+1}$
- $A_{i+1} \geq A_i$

Then we inductively construct sequence

$A=A_0 \leq A_1 \leq A_2 \leq \ldots \leq A_n = G$

Thus, $G \geq A$ for any A. So G is optimal.

Today

- Huffman Codes
- Minimum Spanning Trees

Huffman Codes

- Want to encode string of letters in binary.

  Ex: `ABCDACBDAD`

Huffman Codes

- Want to encode string of letters in binary.

  Ex: `ABCDACBDAD`

- `A = 00, B = 01, C = 10, D = 11`

Huffman Codes

- Want to encode string of letters in binary.

  Ex: `ABCDACBDAD`

- `A = 00, B = 01, C = 10, D = 11`

A	B	C	D	A	C	B	D	A	D
00	01	10	11	00	10	01	11	00	11

Huffman Codes

- Want to encode string of letters in binary.

Ex: ABCDACBDAD

- A = 00, B = 01, C = 10, D = 11

A	B	C	D	A	C	B	D	A	D
00	01	10	11	00	10	01	11	00	11

- Use two bits to encode each letter.

Question: Encoding Length

Using the coding scheme from the last slide, how many bits are needed to encode a string of n `A`s, `B`s, `C`s and `D`s?

1) 2
2) n
3) 2n
4) 4n
5) n^2

Question: Encoding Length

Using the coding scheme from the last slide, how many bits are needed to encode a string of n `A`s, `B`s, `C`s and `D`s?

1) 2
2) n
3) 2n
4) 4n
5) n^2

You need two bits per letter.

Non-Fix Length Encodings

- Suppose instead we had to decode:

`AAABAACBAABADAAA`

Non-Fix Length Encodings

- Suppose instead we had to decode:

  `AAABAACBAABADAAA`

- 16 Letters requires 32 bits.

Non-Fix Length Encodings

- Suppose instead we had to decode:

  `AAABAACBAABADAAA`

- 16 Letters requires 32 bits.
- Note that there are a lot of `A`s here. If we could find a way to encode them with fewer bits, we could save a lot.

Unique Decoding

Cannot do any encoding we like.

Unique Decoding

Cannot do any encoding we like.

Suppose we tried:

`A = 0, B = 1, C = 10, D = 01`

Unique Decoding

Cannot do any encoding we like.

Suppose we tried:

`A = 0, B = 1, C = 10, D = 01`

How do you decode `01`? Either `AB` or `D`.

Unique Decoding

Cannot do any encoding we like.

Suppose we tried:

`A = 0, B = 1, C = 10, D = 01`

How do you decode `01`? Either `AB` or `D`.

Problem: The encoding for `A` is a prefix of the encoding for `D`. When you see it, you don't know if it's an `A`, or the start of a `D`.

Prefix Free Encodings

Definition: An encoding is prefix-free if the encoding of no letter is a prefix of the encoding of any other.

Prefix Free Encodings

Definition: An encoding is prefix-free if the encoding of no letter is a prefix of the encoding of any other.

Example:

`A = 0, B = 10, C = 110, D = 111`

Prefix Free Encodings

Definition: An encoding is prefix-free if the encoding of no letter is a prefix of the encoding of any other.

Example:

`A = 0, B = 10, C = 110, D = 111`

Lemma: Any prefix-free encoding can be uniquely decoded.

Example

`A = 0, B = 10, C = 110, D = 111`

Decode:

`00010001101000100111000`

Example

A = 0, B = 10, C = 110, D = 111

Decode:

00010001101000100111000

A

Example

A = 0, B = 10, C = 110, D = 111

Decode:

00010001101000100111000

A A

Example

A = 0, B = 10, C = 110, D = 111

Decode:

00010001101000100111000

AAA

Example

A = 0, B = 10, C = 110, D = 111

Decode:

00010001101000100111000

A A A B

Example

A = 0, B = 10, C = 110, D = 111

Decode:

0001000110100010011100 0

A A A B A

Example

A = 0, B = 10, C = 110, D = 111

Decode:

00010001101000100111000

AAA B AA

Example

A = 0, B = 10, C = 110, D = 111

Decode:

00010001101000100111000

AAA B AA C

Example

A = 0, B = 10, C = 110, D = 111

Decode:

0001000110100010011000

A A A B A A C B

Example

`A = 0, B = 10, C = 110, D = 111`

Decode:

`00010001101000100111000`

`AAA B AA C B A`

Example

A = 0, B = 10, C = 110, D = 111

Decode:

0001000110100010011100 0

A A A B A A C B A A

Example

A = 0, B = 10, C = 110, D = 111

Decode:

0001000110100010011100 0

A A A B A A C B A A B

Example

A = 0, B = 10, C = 110, D = 111

Decode:

0001000110100010011100

A A A B A A C B A A B A

Example

A = 0, B = 10, C = 110, D = 111

Decode:

0001000110100010011 1000

A A A B A A C B A A B A D

Example

A = 0, B = 10, C = 110, D = 111

Decode:

00010001101000100111000

A A A B A A C B A A B A D A

Example

A = 0, B = 10, C = 110, D = 111

Decode:

00010001101000100111000

A A A B A A C B A A B A D A A

Example

A = 0, B = 10, C = 110, D = 111

Decode:

00010001101000100111000

A A A B A A C B A A B A D A A A

Example

A = 0, B = 10, C = 110, D = 111

Decode:

00010001101000100111000

A A A B A A C B A A B A D A A A

Only 23 bits instead of 32!

Optimal Encoding

<u>Problem:</u> Given a string, S, find a prefix-free encoding that encodes S using the fewest number of bits.

How Long is the Encoding?

If for each letter x in our string, x appears f(x) times and if we encode x as a string of length ℓ(x), the total encoding length is:

$$\Sigma f(x) \cdot \ell(x).$$

How Long is the Encoding?

If for each letter x in our string, x appears f(x) times and if we encode x as a string of length ℓ(x), the total encoding length is:

$$\sum f(x) \cdot \ell(x).$$

Our example has:

11 `A`s, 3 `B`s, 1 `C`, 1 `D`.

How Long is the Encoding?

If for each letter x in our string, x appears f(x) times and if we encode x as a string of length ℓ(x), the total encoding length is:

$$\sum f(x) \cdot \ell(x).$$

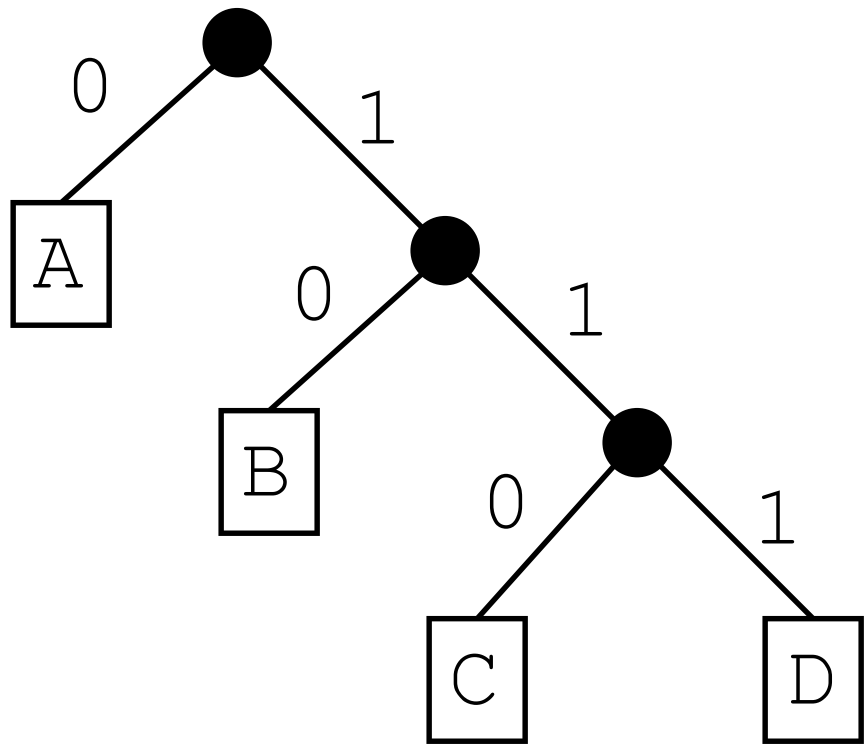
Our example has:

11 `A`s, 3 `B`s, 1 `C`, 1 `D`.

These are the frequencies. We need to find the best encoding.

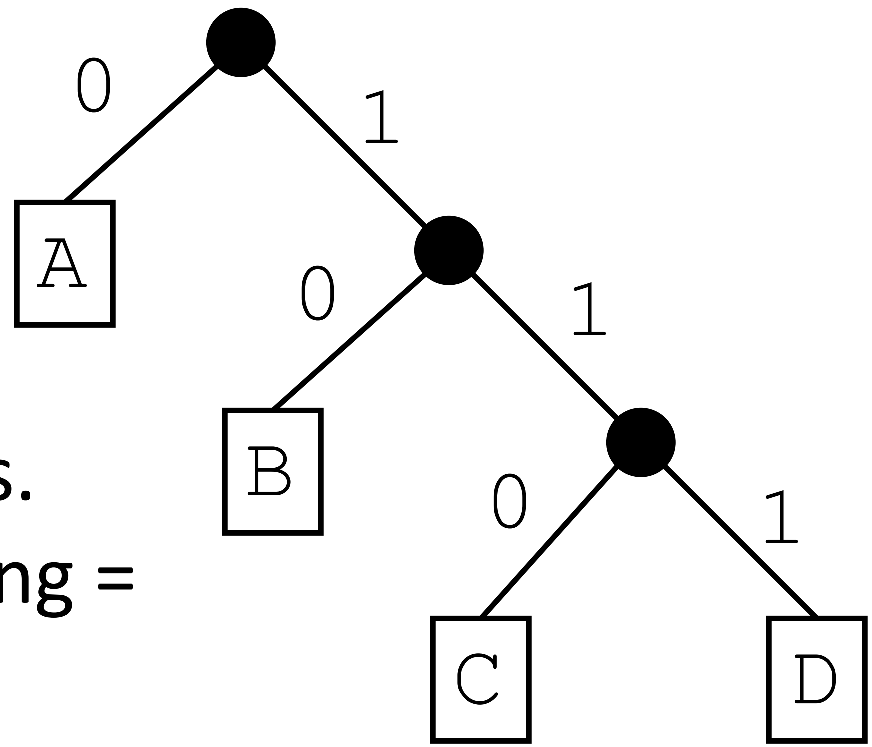
Tree Representation

Can represent prefix-free encoding as a tree.



Tree Representation

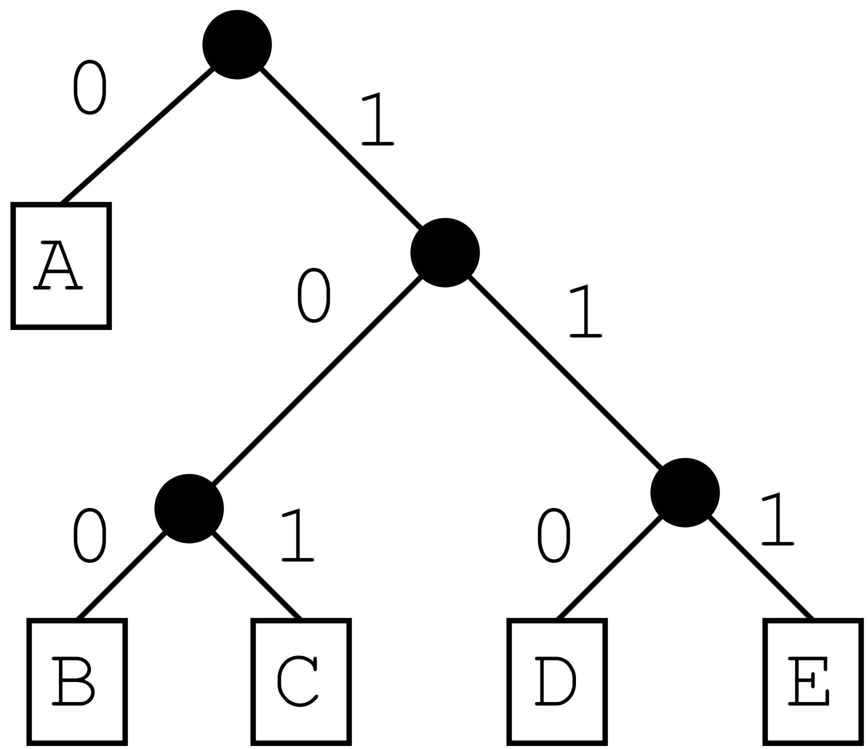
Can represent prefix-free encoding as a tree.



Letters are leaves.
Length of encoding =
Depth of leaf.

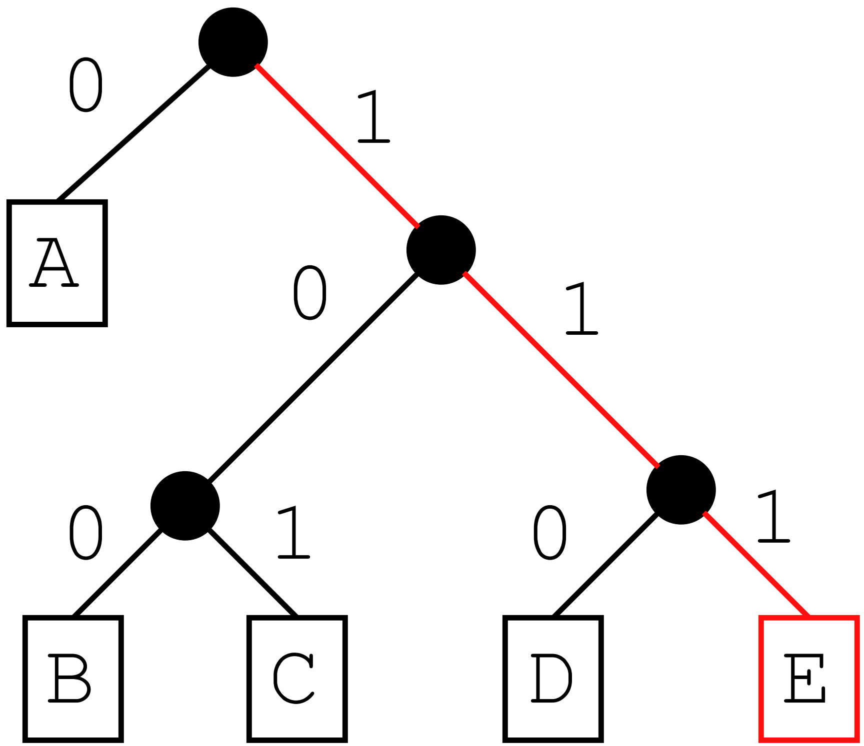
Question: Tree Decoding

What letter does the string `111` correspond to in this tree?



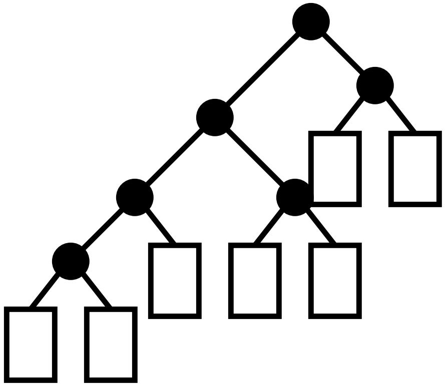
Question: Tree Decoding

What letter does the string `111` correspond to in this tree?

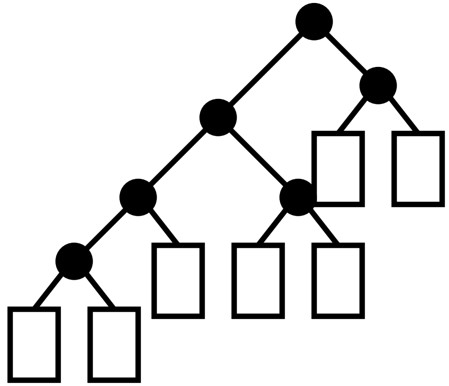


Placement of Leaves

Suppose we know the tree structure. Where do we put the letters?



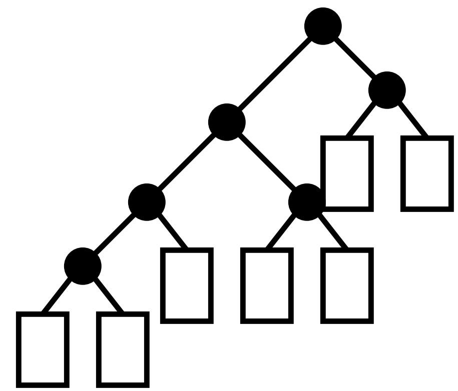
Placement of Leaves



Suppose we know the tree structure. Where do we put the letters?

Objective = Σ freq(x)depth(x)

Placement of Leaves



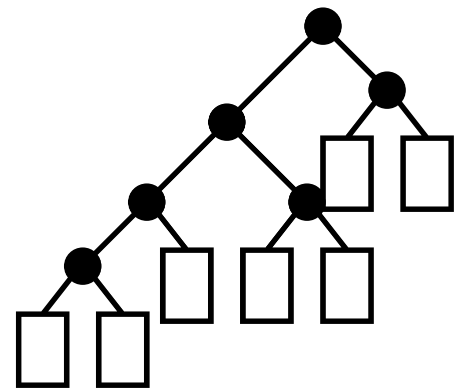
Suppose we know the tree structure. Where do we put the letters?

Objective = Σ freq(x)depth(x)

Want least frequent letters at lowest depth.

Placement of Leaves

Suppose we know the tree structure. Where do we put the letters?



Objective = Σ freq(x)depth(x)

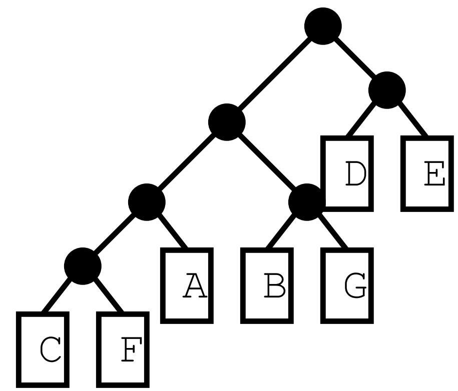
Want least frequent letters at lowest depth.

<u>Letter frequencies</u>

`Ax10, Bx15, Cx4, Dx22, Ex31, Fx5, Gx19`

Placement of Leaves

Suppose we know the tree structure. Where do we put the letters?



Objective = Σ freq(x)depth(x)

Want least frequent letters at lowest depth.

Letter frequencies

Ax10, Bx15, Cx4, Dx22, Ex31, Fx5, Gx19

Siblings

- No matter what the tree structure, two of the deepest leaves are siblings.

Siblings

- No matter what the tree structure, two of the deepest leaves are siblings.
- Can assume filled by two least frequent elements.

Siblings

- No matter what the tree structure, two of the deepest leaves are siblings.
- Can assume filled by two least frequent elements.
- Can assume that two least frequent elements are siblings!

Example

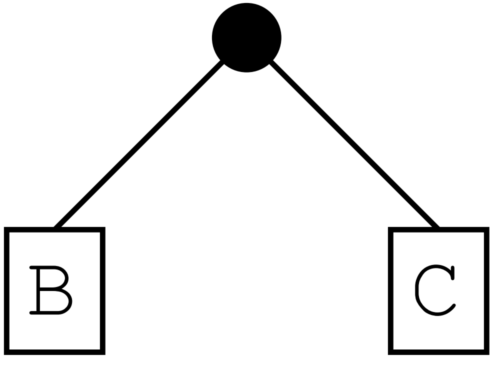
Frequencies:

`Ax30, Bx15, Cx25, Dx50, Ex65`

Example

Frequencies:

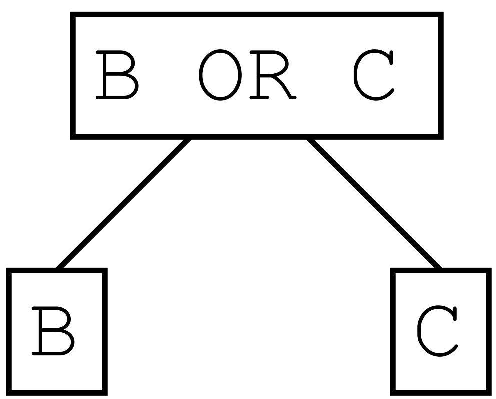
Ax30, Bx15, Cx25, Dx50, Ex65



Example

Frequencies:

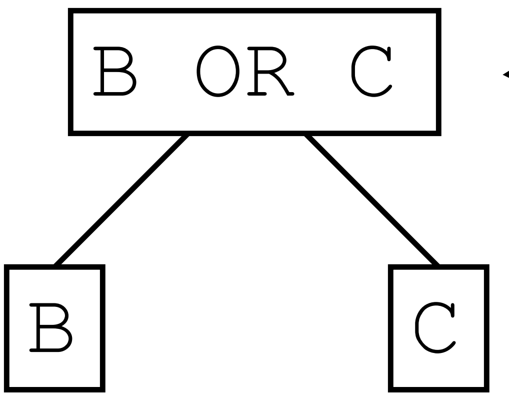
`Ax30, Bx15, Cx25, Dx50, Ex65`



Example

Frequencies:

`Ax30, Bx15, Cx25, Dx50, Ex65`

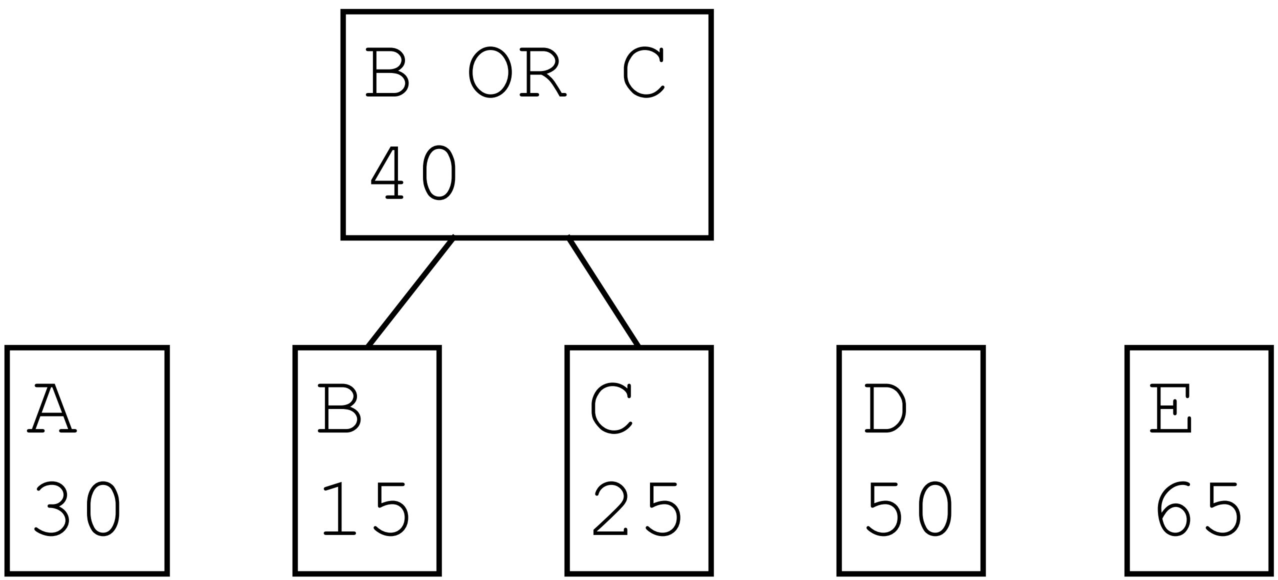


Think of as a new node of weight 15+25=40

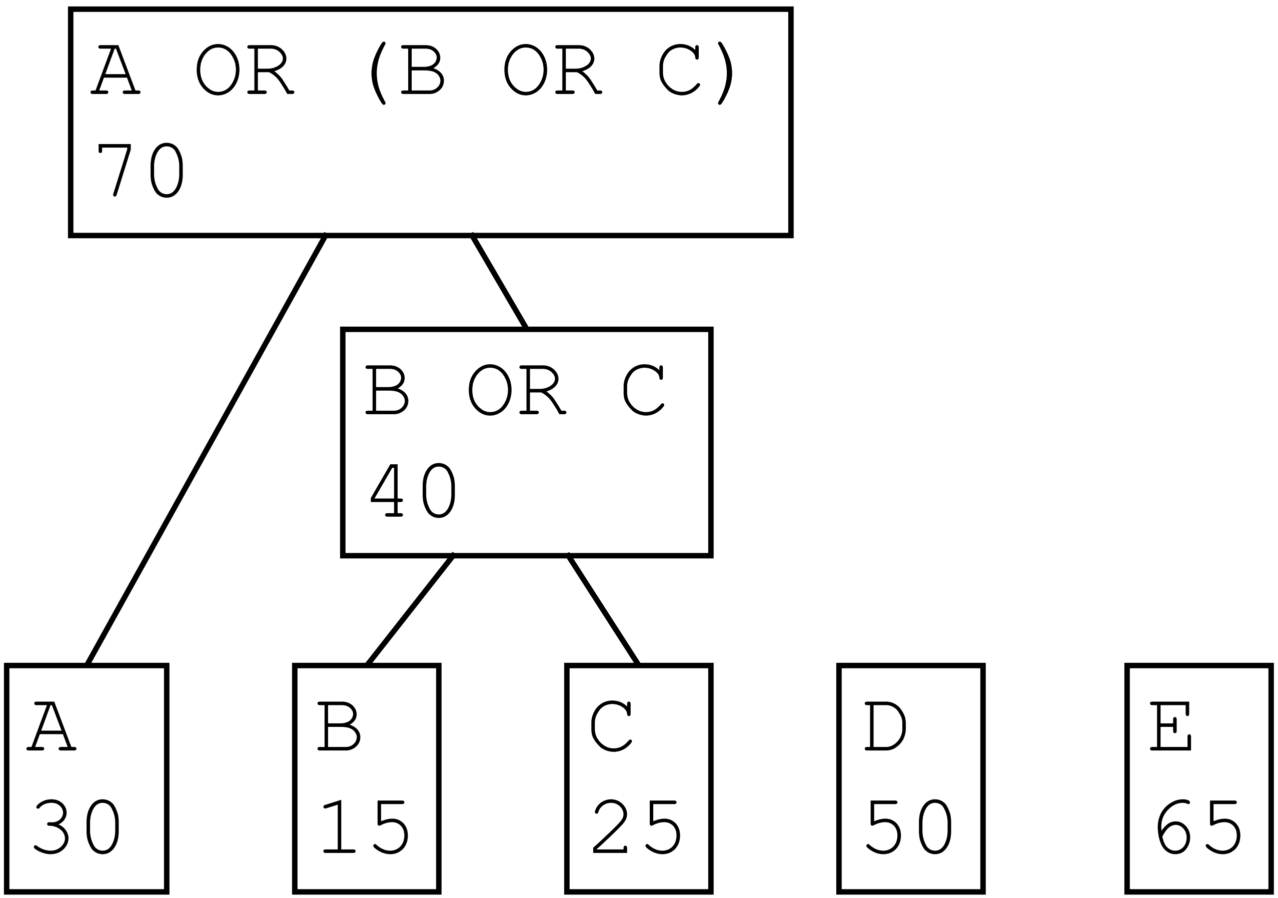
Example

A	B	C	D	E
30	15	25	50	65

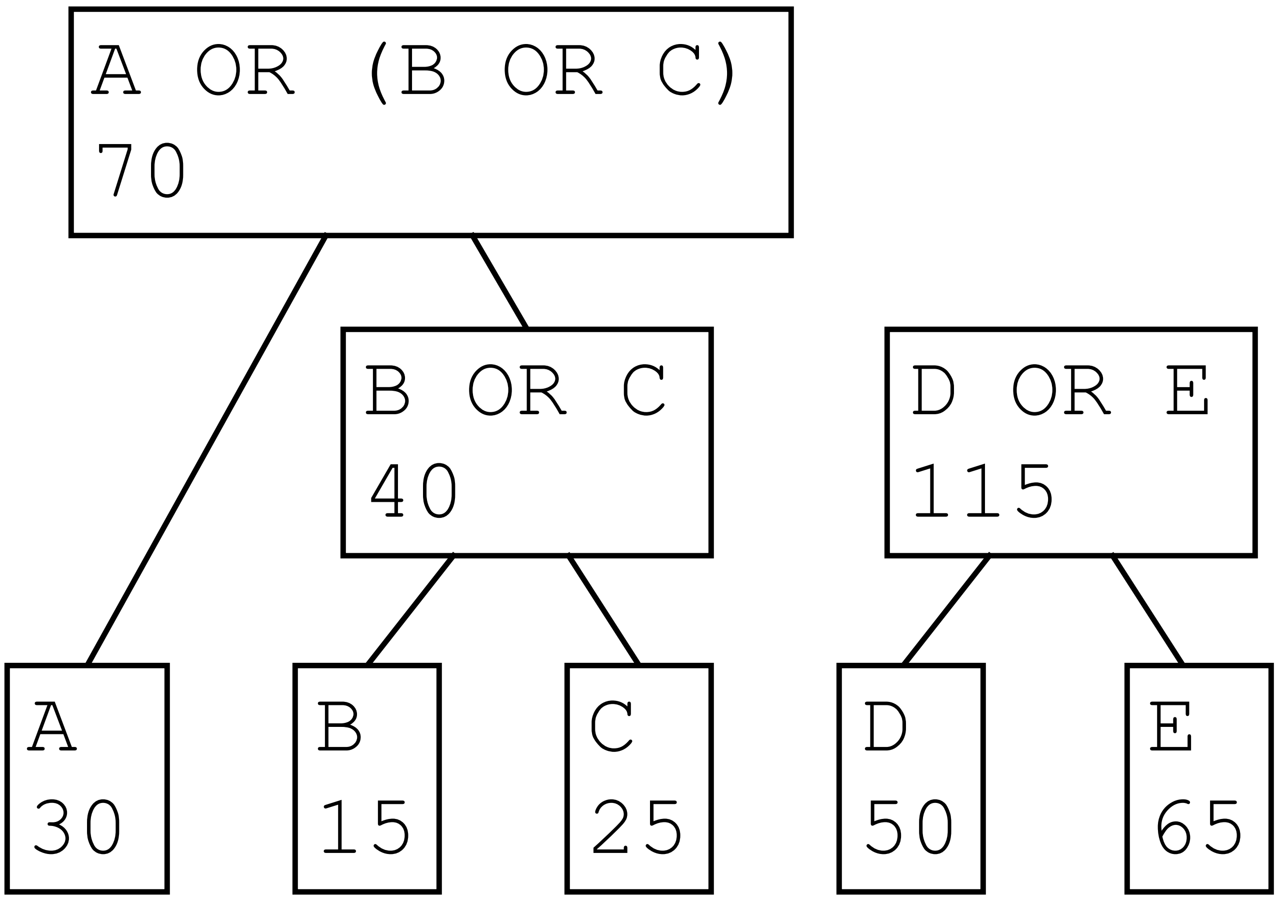
Example



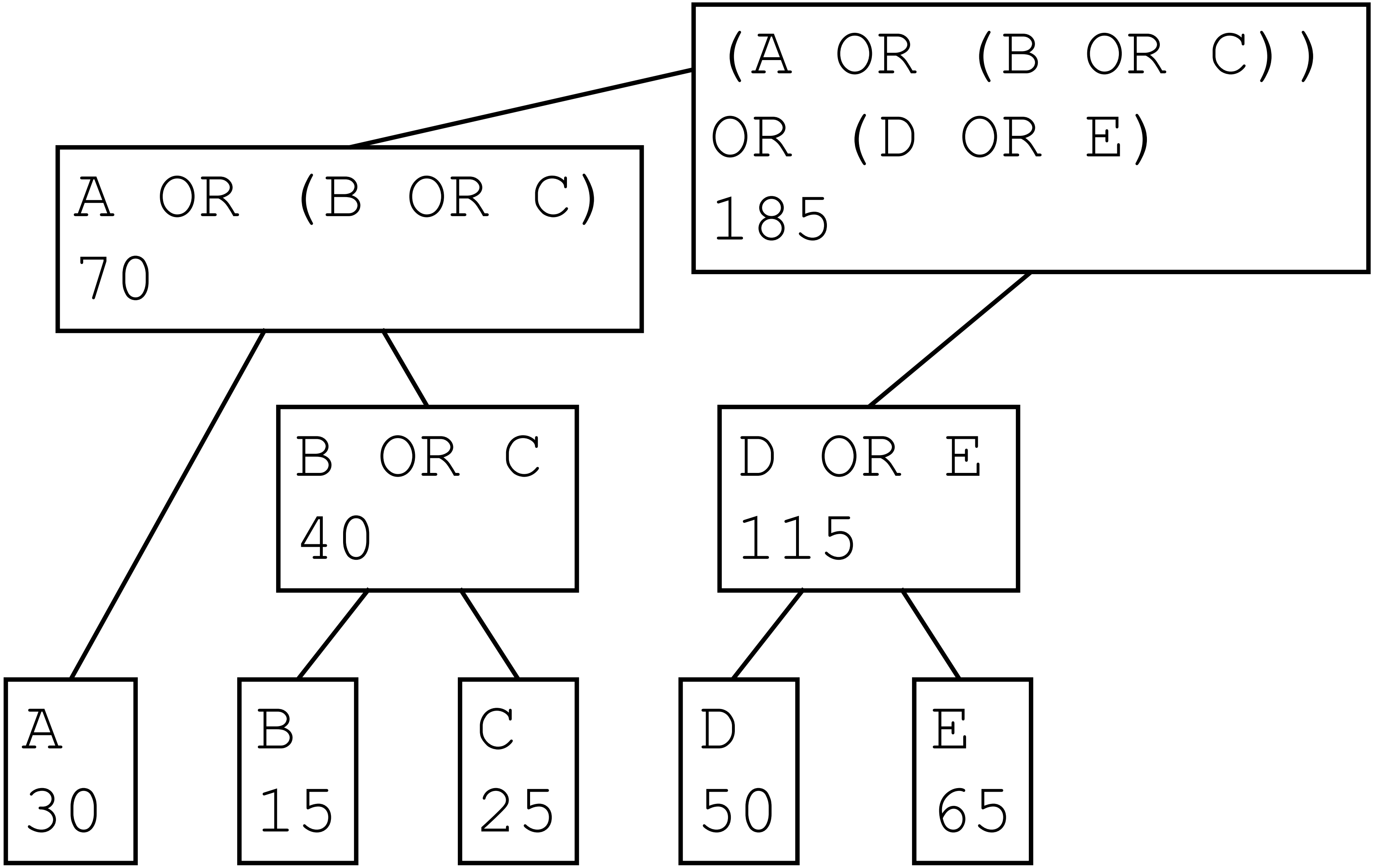
Example



Example



Example


(A OR (B OR C)) OR (D OR E)
185
A OR (B OR C)
70
B OR C
40
D OR E
115
A
30
B
15
C
25
D
50
E
65

Algorithm

```
HuffmanTree(L)
  While(at least two left)
    x, y ← Two least frequent
    z new node f(z) ← f(x)+f(y)
    x and y children of z
    Replace x and y with z in L
  Return remaining elt of L
```

Algorithm

```
HuffmanTree(L)
   While(at least two left)
      x, y ← Two least frequent
      z new node f(z) ← f(x)+f(y)
      x and y children of z
      Replace x and y with z in L
   Return remaining elt of L
```

Better with priority queue.

Optimized Algorithm

```
HuffmanTree(L)
  Priority queue Q
  Insert all elements of L to Q
  While(|Q| ≥ 2)
    x ← Q.DeleteMin()
    y ← Q.DeleteMin()
    Create z, f(z) = f(x) + f(y)
    x and y children of z
    Q.Insert(z)
  Return Q.DeleteMin()
```

Optimized Algorithm

```
HuffmanTree(L)
  Priority queue Q
  Insert all elements of L to Q
  While(|Q| ≥ 2)
    x ← Q.DeleteMin()
    y ← Q.DeleteMin()
    Create z, f(z) = f(x) + f(y)
    x and y children of z
    Q.Insert(z)
  Return Q.DeleteMin()
```

O(n log(n))

Optimized Algorithm

```
HuffmanTree(L)
  Priority queue Q
  Insert all elements of L to Q
  While(|Q| ≥ 2)
    x ← Q.DeleteMin()
    y ← Q.DeleteMin()
    Create z, f(z) = f(x) + f(y)
    x and y children of z
    Q.Insert(z)
  Return Q.DeleteMin()
```

O(n log(n)) (Priority queue Q / Insert all elements of L to Q)

O(n) Iterations (While(|Q| ≥ 2))

Optimized Algorithm

```
HuffmanTree(L)
  Priority queue Q
  Insert all elements of L to Q
  While(|Q| ≥ 2)
    x ← Q.DeleteMin()
    y ← Q.DeleteMin()
    Create z, f(z) = f(x) + f(y)
    x and y children of z
    Q.Insert(z)
  Return Q.DeleteMin()
```

O(n log(n))

O(n) Iterations

O(log(n))

Optimized Algorithm

```
HuffmanTree(L)
  Priority queue Q
  Insert all elements of L to Q
  While(|Q| ≥ 2)
    x ← Q.DeleteMin()
    y ← Q.DeleteMin()
    Create z, f(z) = f(x) + f(y)
    x and y children of z
    Q.Insert(z)
  Return Q.DeleteMin()
```

O(n log(n)) — Priority queue Q / Insert all elements of L to Q

O(n) Iterations — While(|Q| ≥ 2)

O(log(n)) — loop body

Runtime: O(n log(n))

Proof of Correctness

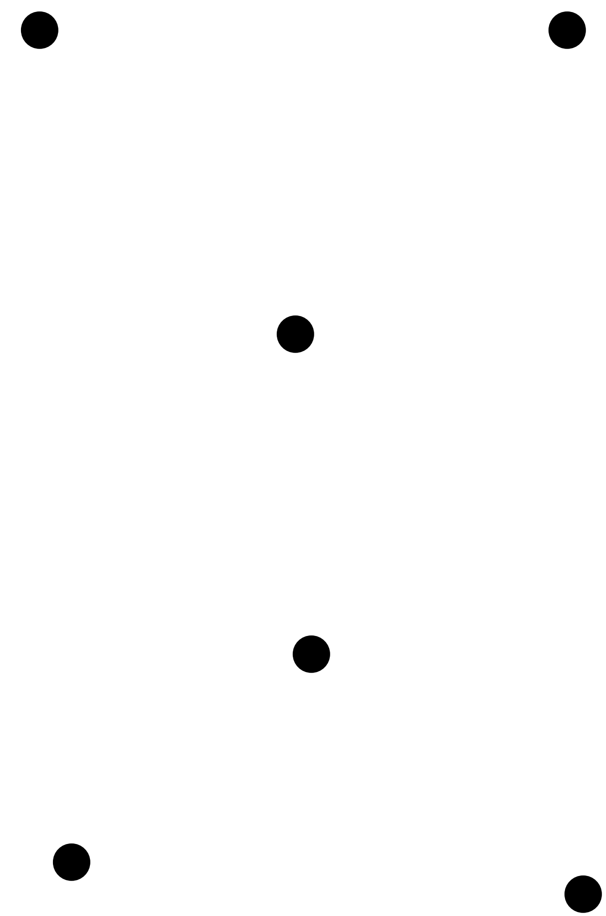
- Know that there is a correct solution with lightest elements as siblings
- *If* we require that lightest elements are siblings, problem is *equivalent* to smaller Huffman tree problem
- By induction, smaller problem is solved correctly

Minimum Spanning Trees

- Suppose that you have a collection of cities that you would like to connect by roads.

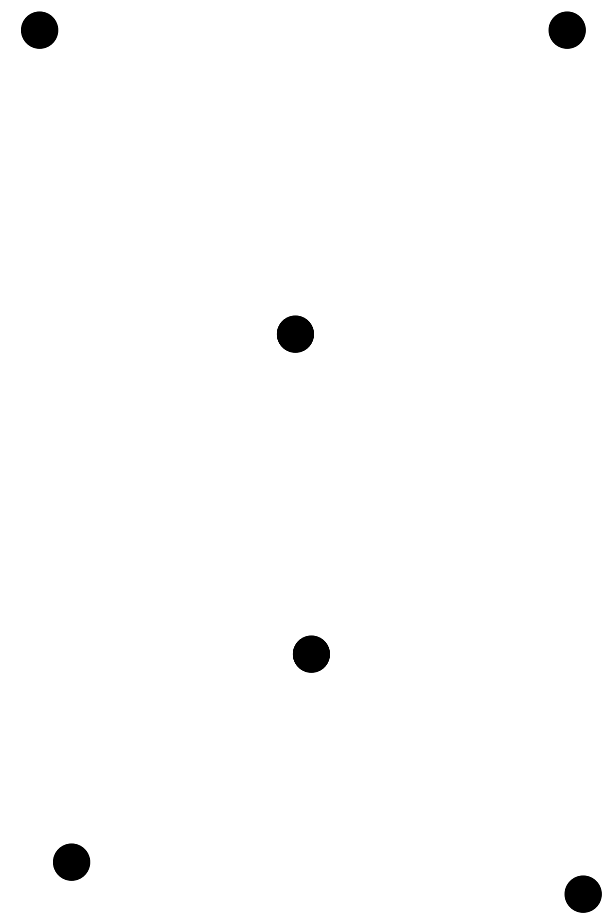
Minimum Spanning Trees

- Suppose that you have a collection of cities that you would like to connect by roads.



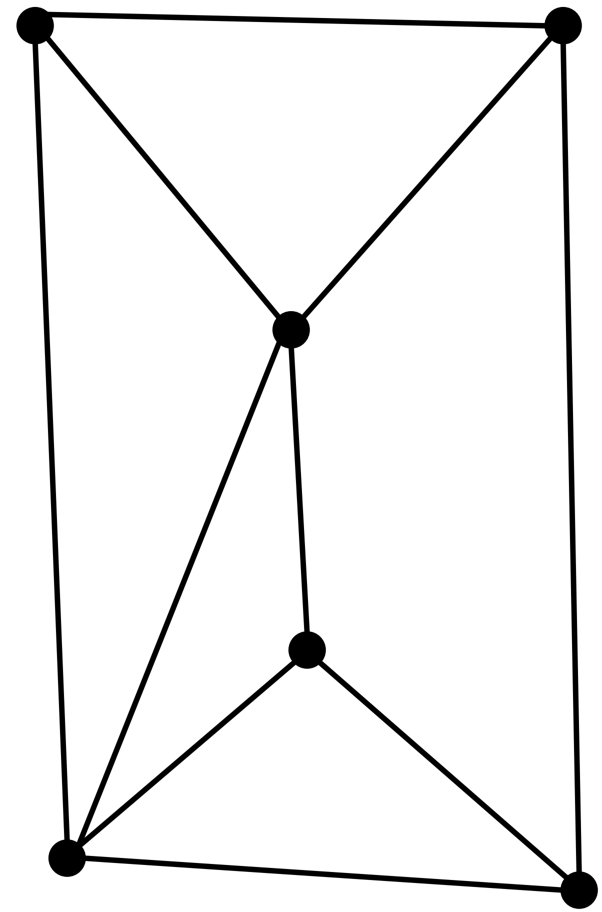
Minimum Spanning Trees

- Suppose that you have a collection of cities that you would like to connect by roads.
- There are several potential roads you could build.



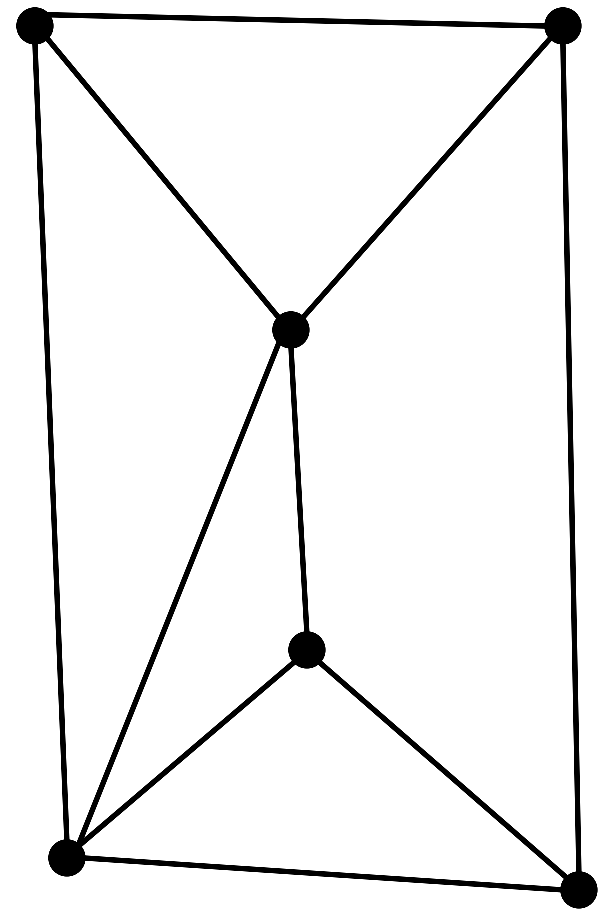
Minimum Spanning Trees

- Suppose that you have a collection of cities that you would like to connect by roads.
- There are several potential roads you could build.



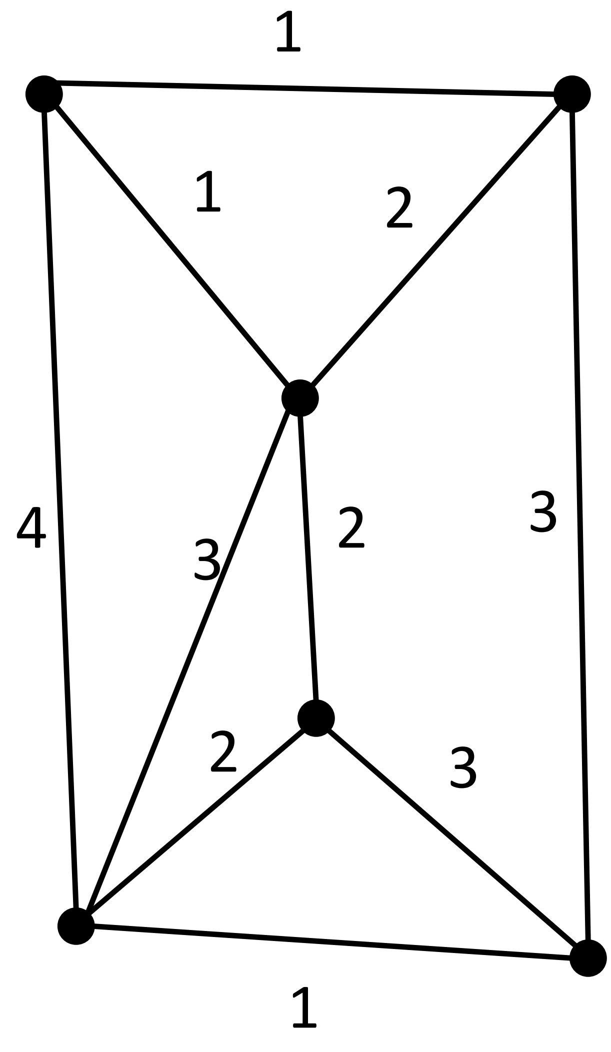
Minimum Spanning Trees

- Suppose that you have a collection of cities that you would like to connect by roads.
- There are several potential roads you could build.
- Each has a cost.



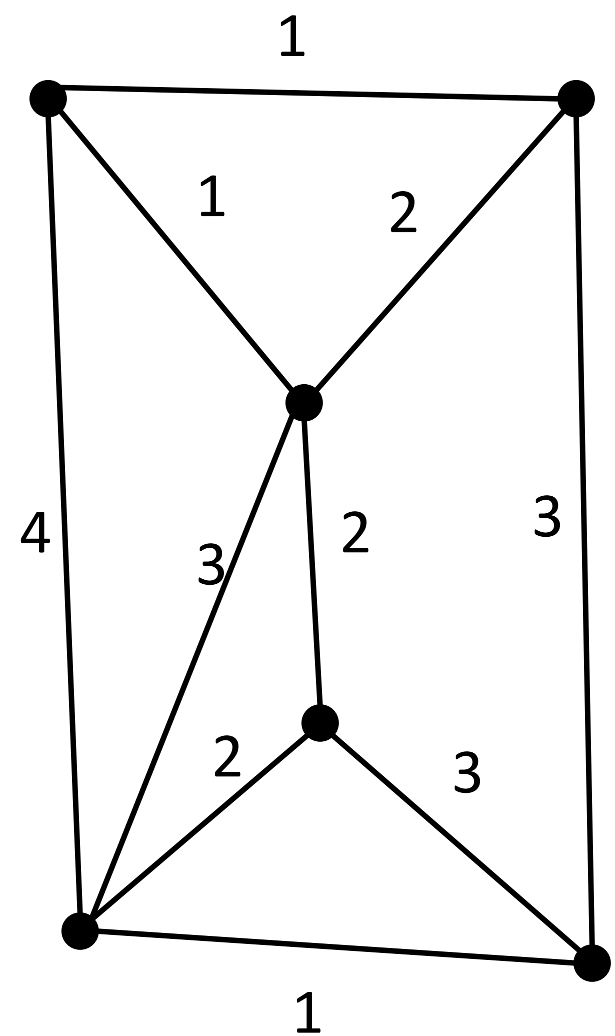
Minimum Spanning Trees

- Suppose that you have a collection of cities that you would like to connect by roads.
- There are several potential roads you could build.
- Each has a cost.



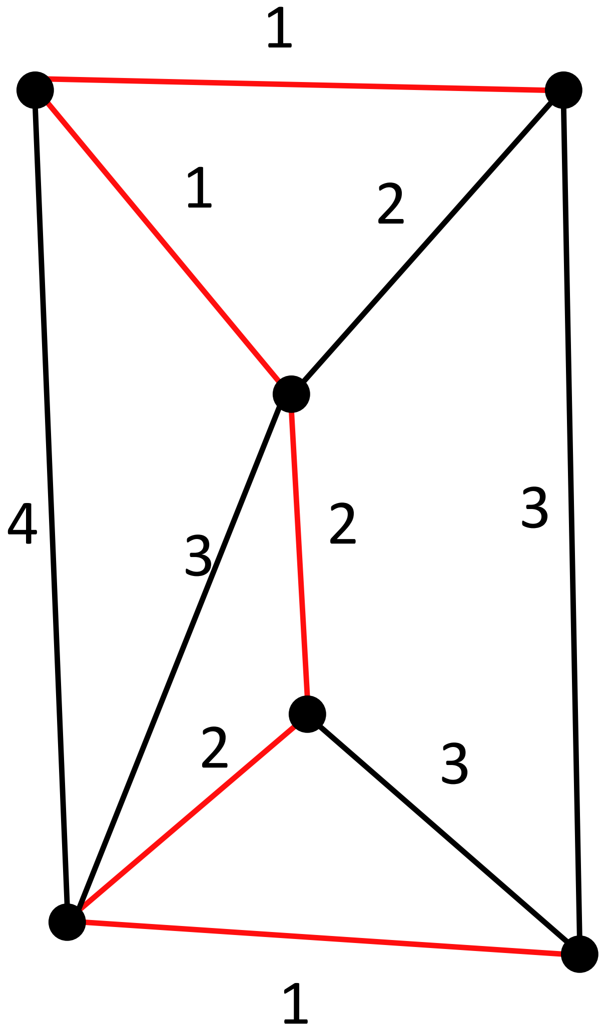
Minimum Spanning Trees

- Suppose that you have a collection of cities that you would like to connect by roads.
- There are several potential roads you could build.
- Each has a cost.
- What is the cheapest way to connect them?



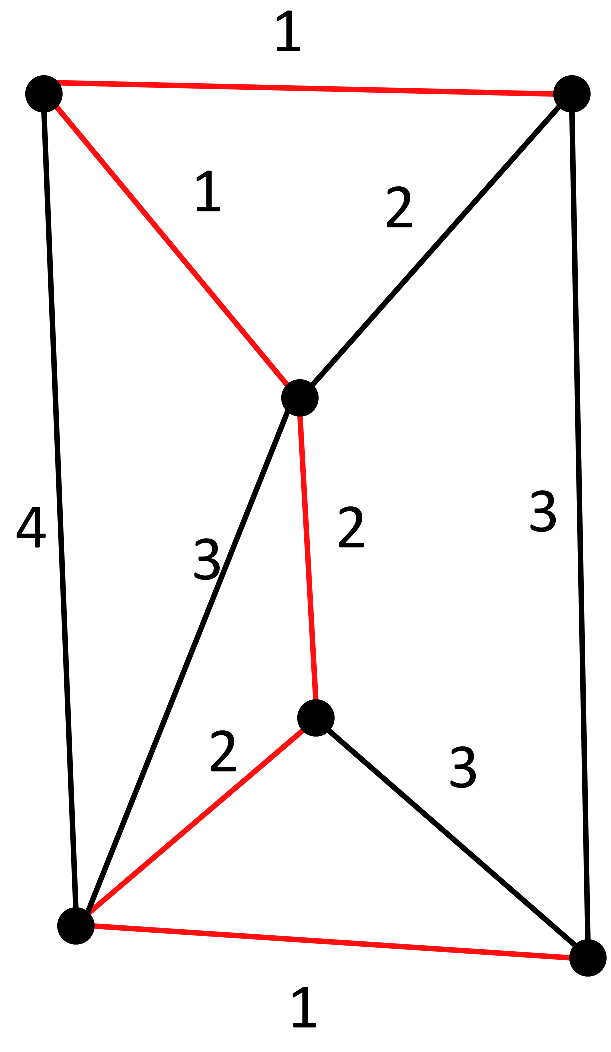
Minimum Spanning Trees

- Suppose that you have a collection of cities that you would like to connect by roads.
- There are several potential roads you could build.
- Each has a cost.
- What is the cheapest way to connect them?



Minimum Spanning Trees

- Suppose that you have a collection of cities that you would like to connect by roads.
- There are several potential roads you could build.
- Each has a cost.
- What is the cheapest way to connect them? 1+1+1+2+2=7



Trees

Note: In this problem, you will never want to build more roads than necessary. This means, you will never want to have a cycle.

Trees

Note: In this problem, you will never want to build more roads than necessary. This means, you will never want to have a cycle.

Definition: A tree is a connected graph, with no cycles.

Trees

<u>Note:</u> In this problem, you will never want to build more roads than necessary. This means, you will never want to have a cycle.

<u>Definition:</u> A <u>tree</u> is a connected graph, with no cycles.

A <u>spanning tree</u> in a graph G, is a subset of the edges of G that connect all vertices and have no cycles.

Trees

Note: In this problem, you will never want to build more roads than necessary. This means, you will never want to have a cycle.

Definition: A tree is a connected graph, with no cycles.

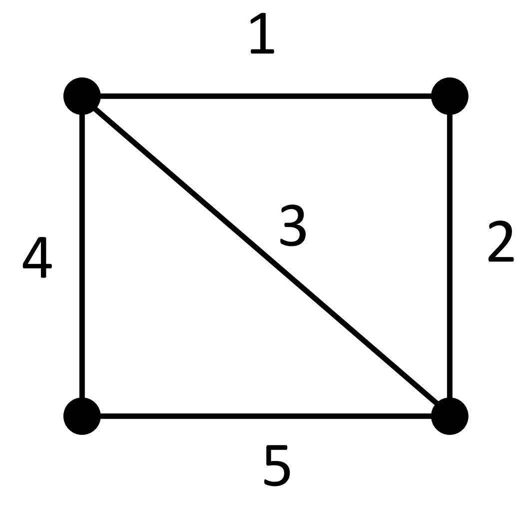
A spanning tree in a graph G, is a subset of the edges of G that connect all vertices and have no cycles.

If G has weights, a minimum spanning tree is a spanning tree whose total weight is as small as possible.

Question: MST

What is the weight of the minimum spanning tree of the graph below?

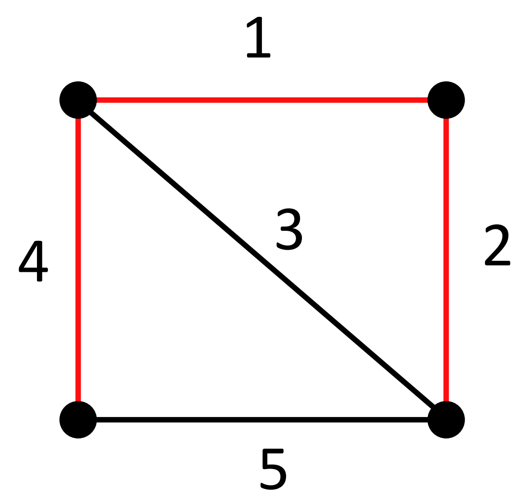
A) 5
B) 6
C) 7
D) 8
E) 9



Question: MST

What is the weight of the minimum spanning tree of the graph below?

A) 5
B) 6
C) 7
D) 8
E) 9



Basic Facts about Trees

<u>Lemma:</u> For an undirected graph G, any two of the below imply the third:

1. $|E| = |V|-1$
2. G is connected
3. G has no cycles

Basic Facts about Trees

Lemma: For an undirected graph G, any two of the below imply the third:

1. $|E| = |V|-1$
2. G is connected
3. G has no cycles

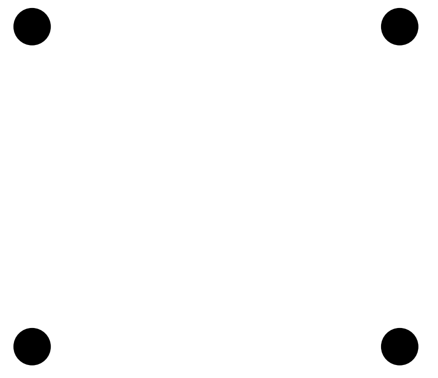
Corollary: If G is a tree, then $|E| = |V|-1$.

Proof Idea

- Start with a graph with no edges.

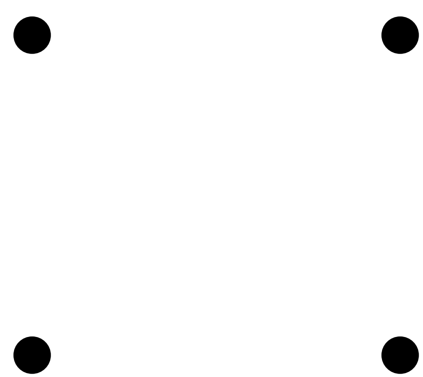
Proof Idea

- Start with a graph with no edges.



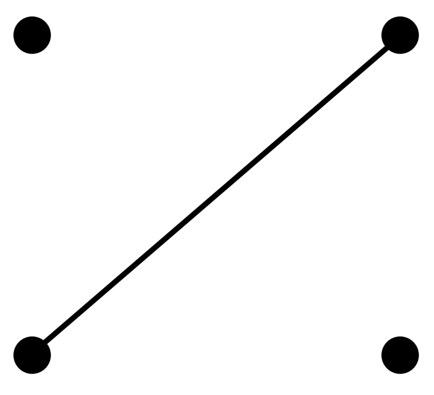
Proof Idea

- Start with a graph with no edges.
- Add edges one at a time.



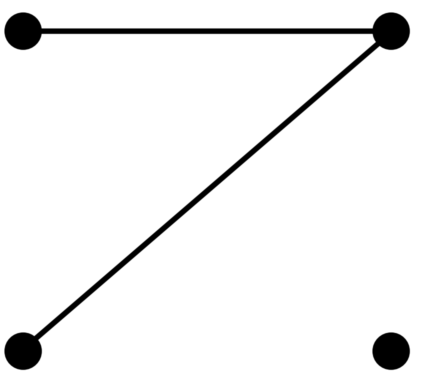
Proof Idea

- Start with a graph with no edges.
- Add edges one at a time.



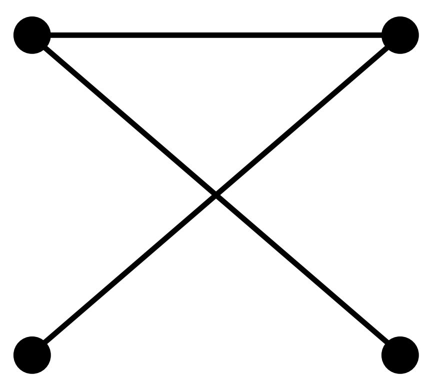
Proof Idea

- Start with a graph with no edges.
- Add edges one at a time.



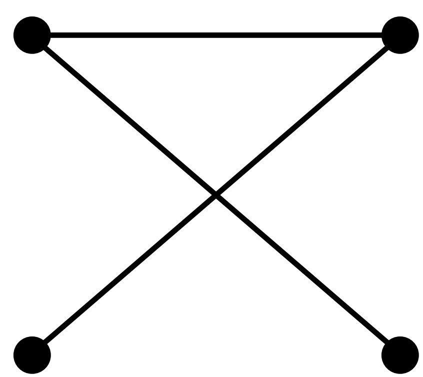
Proof Idea

- Start with a graph with no edges.
- Add edges one at a time.



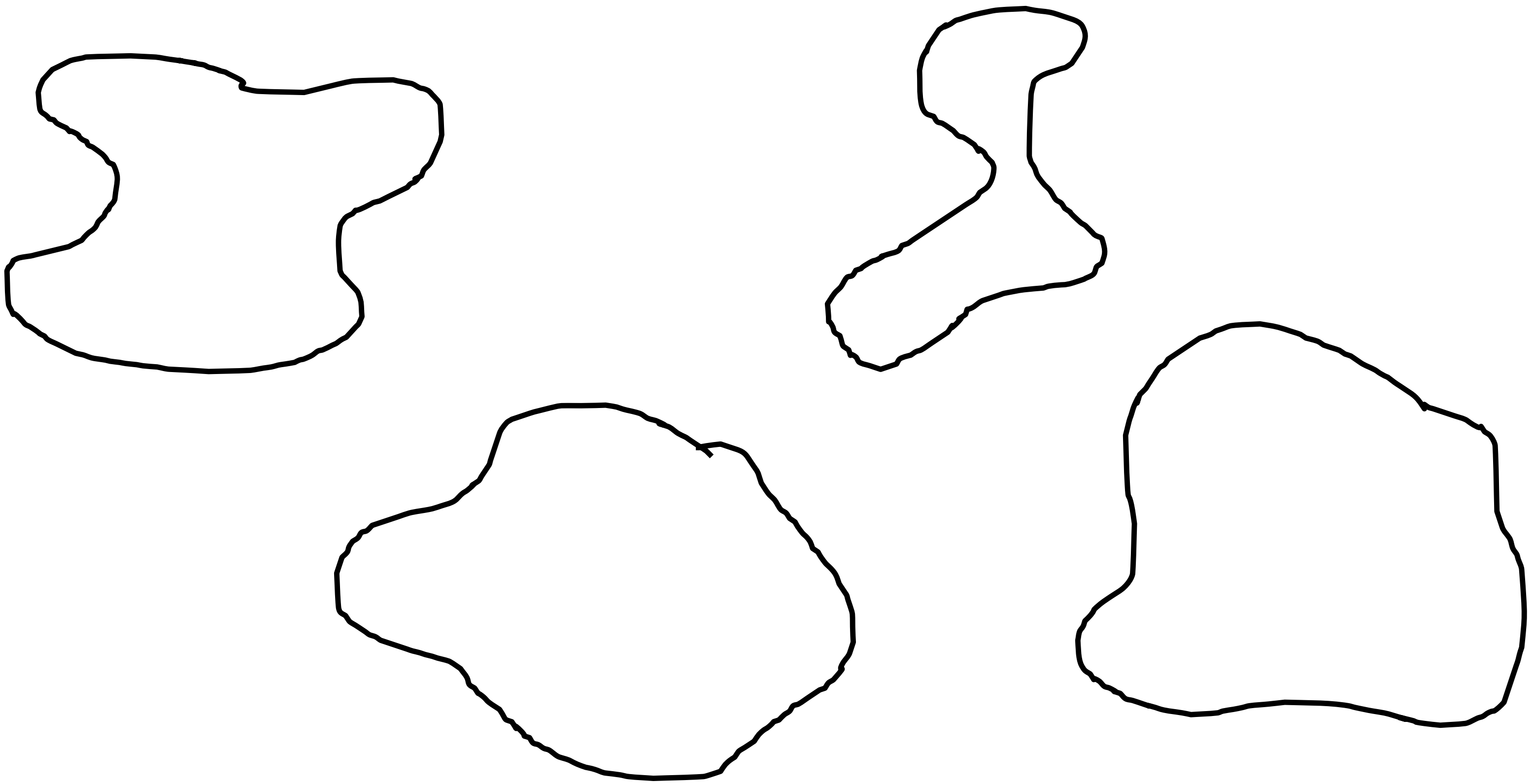
Proof Idea

- Start with a graph with no edges.
- Add edges one at a time.
- Count number of connected components.



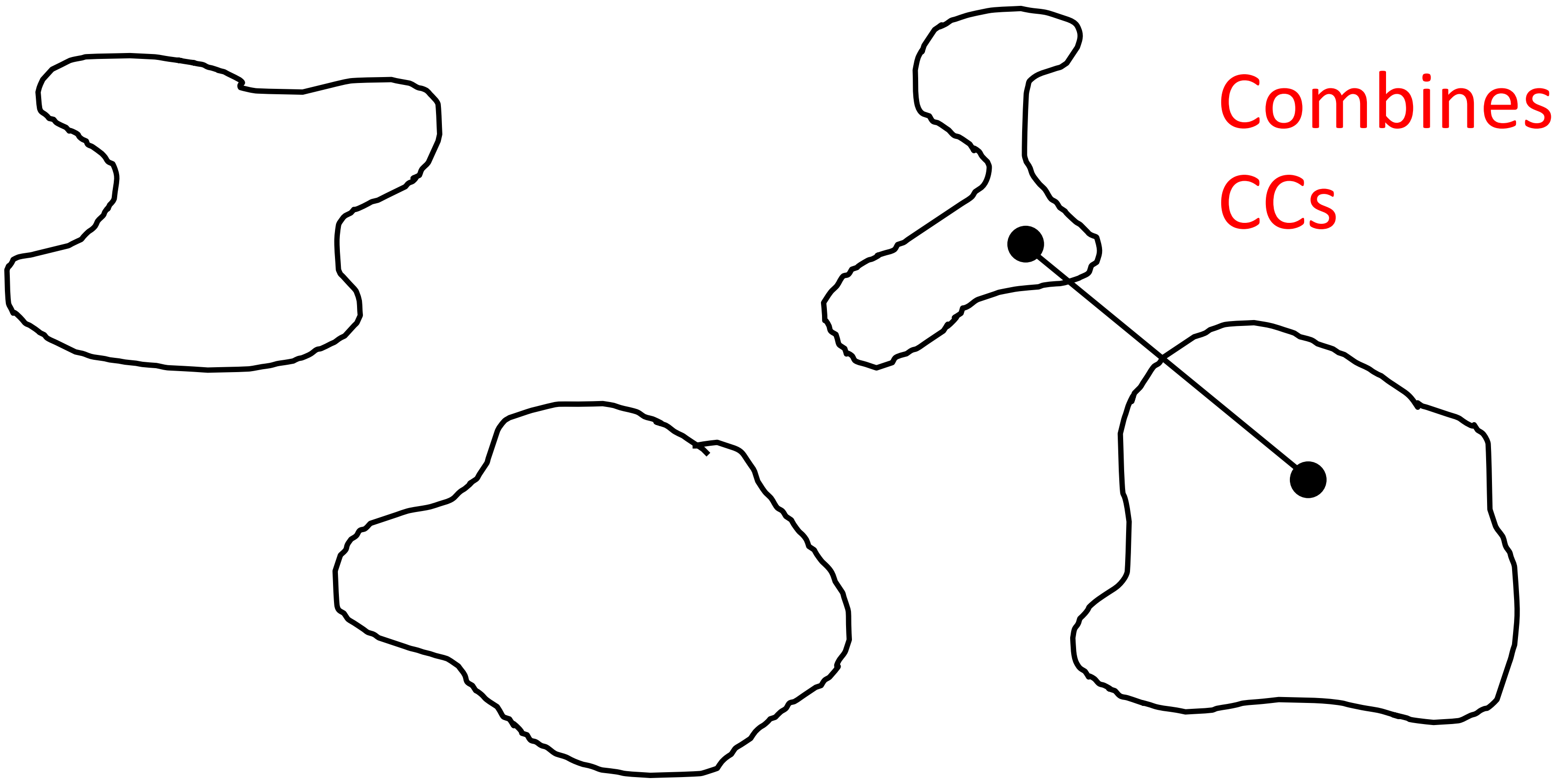
Extra Edge

An extra edge decreases the number of CCs by 1 *unless* it creates a cycle.



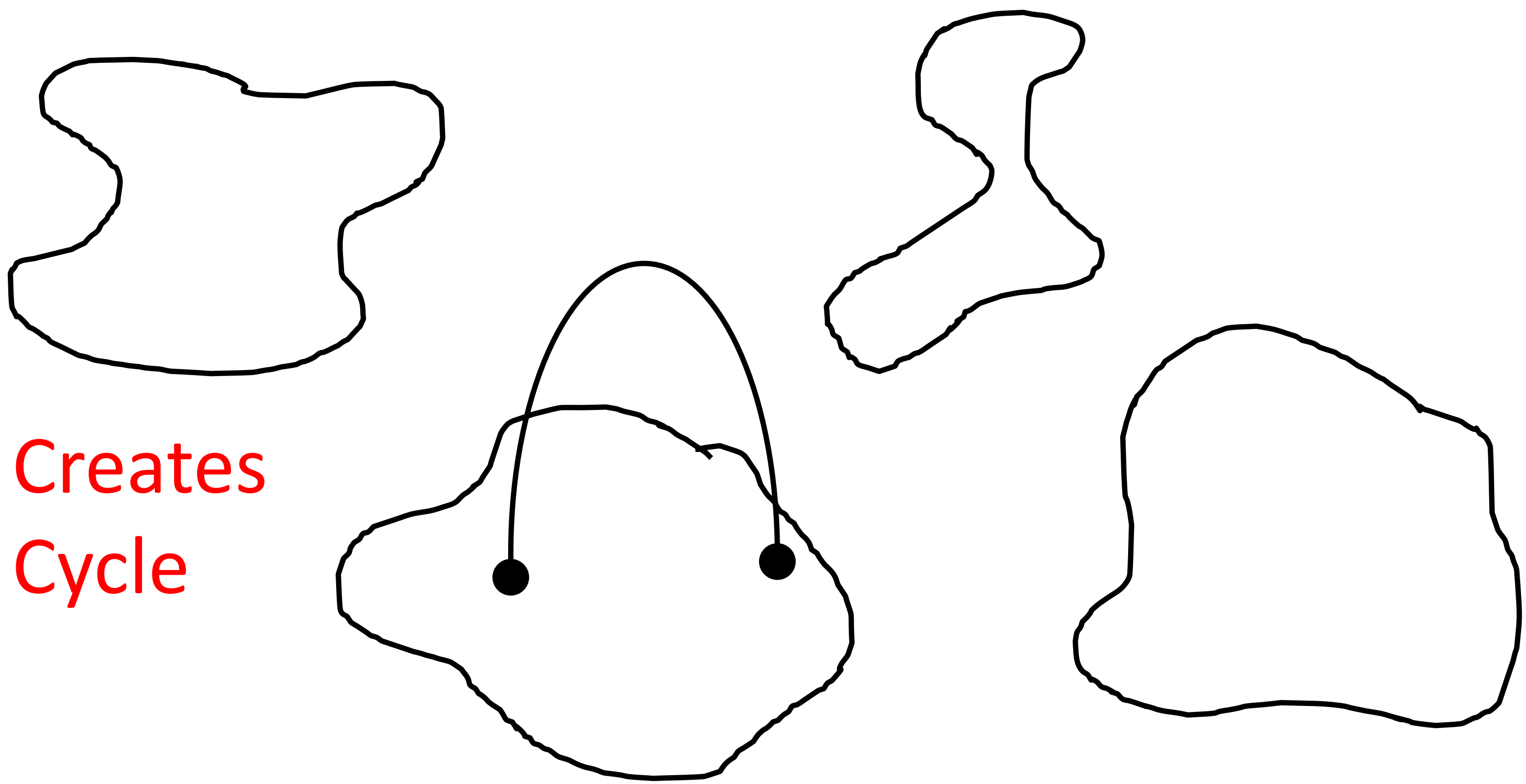
Extra Edge

An extra edge decreases the number of CCs by 1 *unless* it creates a cycle.



Extra Edge

An extra edge decreases the number of CCs by 1 *unless* it creates a cycle.



Number of Components

- Starts with |V|.
- Each edge decreases by 1 unless a cycle.
- Final graph is connected if it reduces to 1.

Number of Components

- Starts with |V|.
- Each edge decreases by 1 unless a cycle.
- Final graph is connected if it reduces to 1.

If |E|=|V|-1 and no cycle, then only 1 CC left.

Number of Components

- Starts with $|V|$.
- Each edge decreases by 1 unless a cycle.
- Final graph is connected if it reduces to 1.

If $|E|=|V|-1$ and no cycle, then only 1 CC left.

If $|E|=|V|-1$ and connected, each edge must decrease by 1, so no cycles.

Number of Components

- Starts with |V|.
- Each edge decreases by 1 unless a cycle.
- Final graph is connected if it reduces to 1.

If |E|=|V|-1 and no cycle, then only 1 CC left.

If |E|=|V|-1 and connected, each edge must decrease by 1, so no cycles.

If connected and no cycles, each edge decreases by 1, so must be |V|-1 edges.