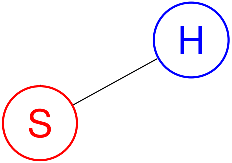
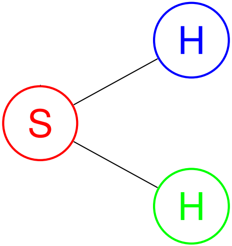
CS 170: Algorithms

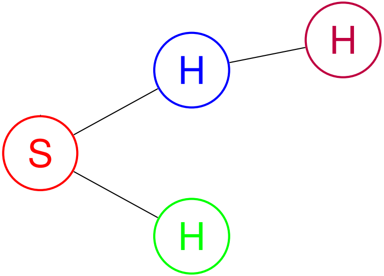
CS 170: Algorithms

CS 170: Algorithms

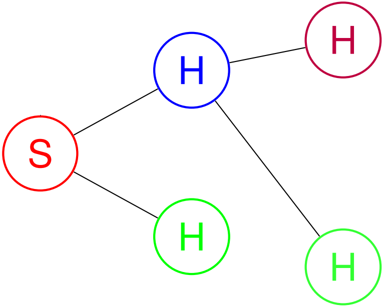


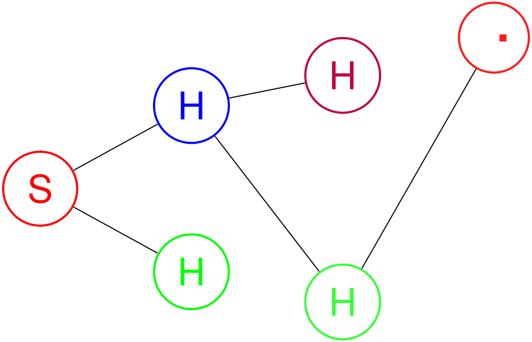
CS 170: Algorithms



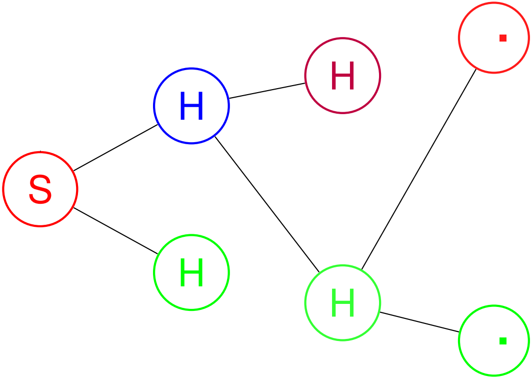

S
H
H
H

CS 170: Algorithms

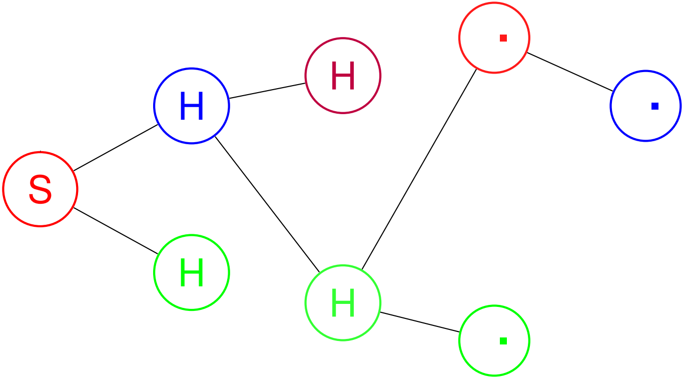


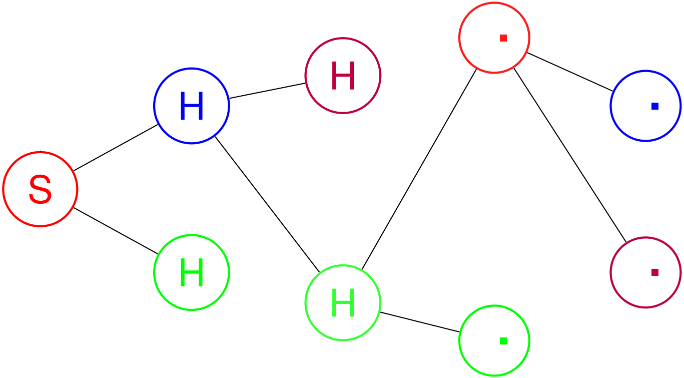

S
H
H
H
H

CS 170: Algorithms

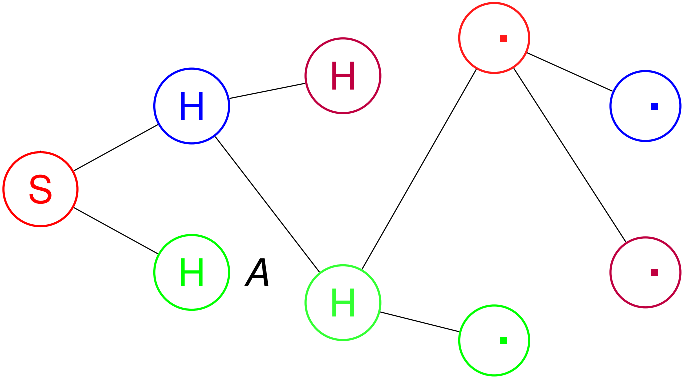


CS 170: Algorithms

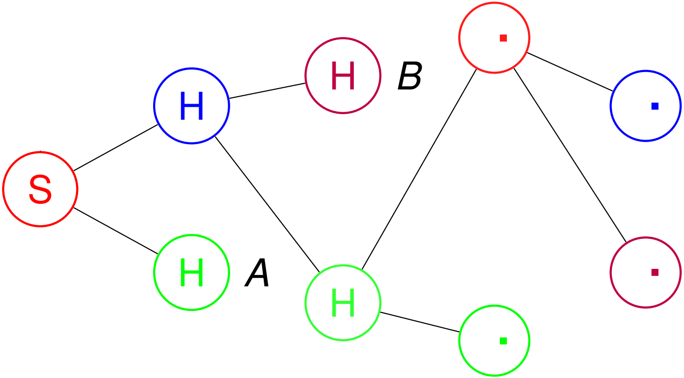



S
H
H
H
H

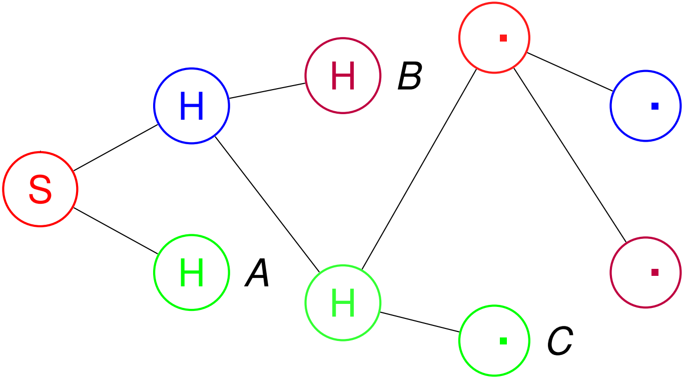
CS 170: Algorithms



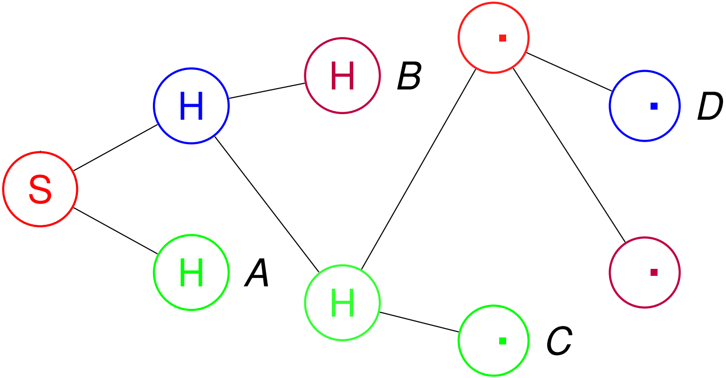
CS 170: Algorithms

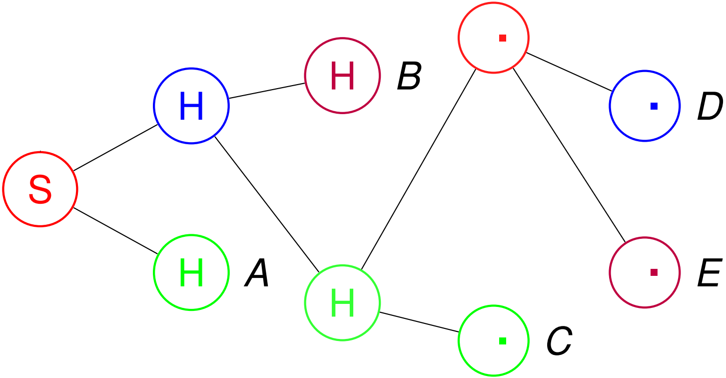


CS 170: Algorithms

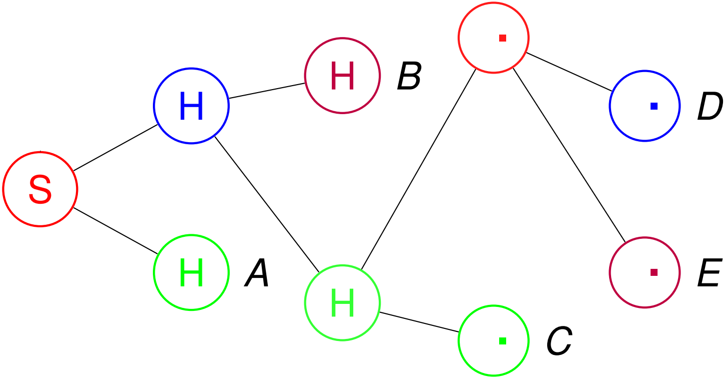


CS 170: Algorithms




S
H
H
B
H
A
H
C
D
E

CS 170: Algorithms



Prims and Huffman Coding.

Lecture in a minute.

Cut Property: MST.

Exists MST that uses minimum weight edge across cut.

Exchange argument.   Prim: $S = \{s\}$

Add cheapest edge (u, v) across $(S, V - S)$

$S = S + v$.

Repeat.

Use priority queue: $O((|V| + |E|) \log |V|)$.

Lecture in a minute.

Cut Property: MST.

Exists MST that uses minimum weight edge across cut.

Exchange argument. Prim: $S = \{s\}$

Add cheapest edge (u,v) across $(S, V-S)$

$S = S + v$.

Repeat.

Use priority queue: $O((|V|+|E|)\log|V|)$.

Huffman Coding.

Symbols, s, with frequencies.

Lecture in a minute.

Cut Property: MST.
Exists MST that uses minimum weight edge across cut.
Exchange argument. Prim: $S = \{s\}$
Add cheapest edge (u,v) across $(S, V-S)$
$S = S + v$.
Repeat.
Use priority queue: $O((|V|+|E|)\log|V|)$.

Huffman Coding.
Symbols, s, with frequencies.
Prefix-Free code.
$\equiv$ binary tree with symbols at leaves.
Cost: sum of depth(s)$\times$$freq(s)$.
Cost2: sum of frequencies of internal nodes.

Lecture in a minute.

Cut Property: MST.
  Exists MST that uses minimum weight edge across cut.
    Exchange argument.   Prim: $S = \{s\}$
   Add cheapest edge (u, v) across $(S, V - S)$
   $S = S + v$.
  Repeat.
  Use priority queue: $O((|V| + |E|) \log |V|)$.

Huffman Coding.
  Symbols, s, with frequencies.
  Prefix-Free code.
   $\equiv$ binary tree with symbols at leaves.
  Cost: sum of depth(s)$\times$*freq*(s).
  Cost2: sum of frequencies of internal nodes.
Algorithm: merge lowest fequency symbols, recurse.
  Exchange Argument $\implies$
    exists optimal tree with this structure.

What is a Cut?

What is a Cut?

What is a cut in an undirected graph, $G = (V, E)$?

(A) A set of edges whose removal disconnects a graph.

(B) For partition of V, $(S, V - S)$, set of edges across it; $E \cap (S \times V - S)$.

What is a Cut?

What is a cut in an undirected graph, $G = (V, E)$?

(A) A set of edges whose removal disconnects a graph.

(B) For partition of V, $(S, V - S)$, set of edges across it; $E \cap (S \times V - S)$.

(A)

What is a Cut?

What is a cut in an undirected graph, $G = (V, E)$?

(A) A set of edges whose removal disconnects a graph.

(B) For partition of V, $(S, V - S)$, set of edges across it; $E \cap (S \times V - S)$.

(A) and (B).

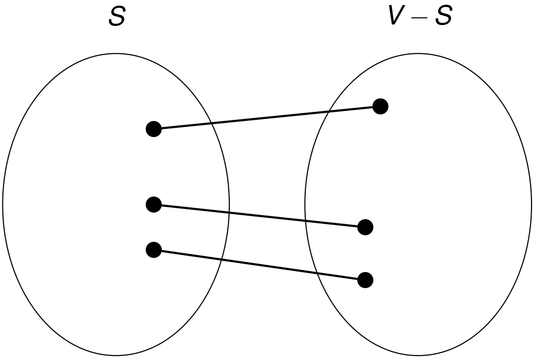
What is a Cut?

What is a cut in an undirected graph, $G = (V, E)$?

(A) A set of edges whose removal disconnects a graph.

(B) For partition of V, $(S, V - S)$, set of edges across it; $E \cap (S \times V - S)$.

(A) and (B).



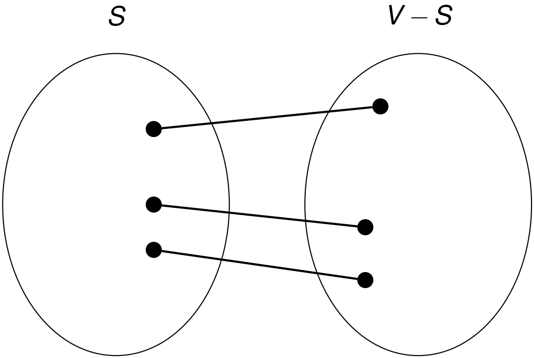
What is a Cut?

What is a cut in an undirected graph, $G = (V, E)$?

(A) A set of edges whose removal disconnects a graph.

(B) For partition of V, $(S, V - S)$, set of edges across it; $E \cap (S \times V - S)$.

(A) and (B).



Note:

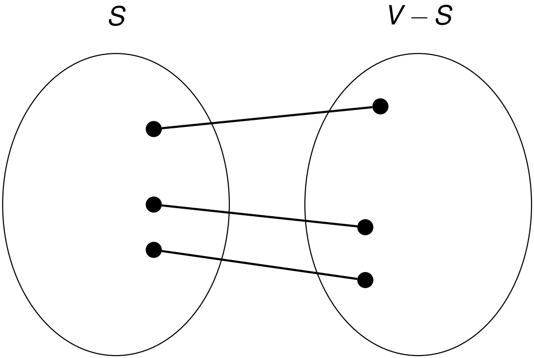
What is a Cut?

What is a cut in an undirected graph, $G = (V, E)$?

(A) A set of edges whose removal disconnects a graph.

(B) For partition of V, $(S, V - S)$, set of edges across it; $E \cap (S \times V - S)$.

(A) and (B).



Note:sometimes specified as $(S, V - S)$

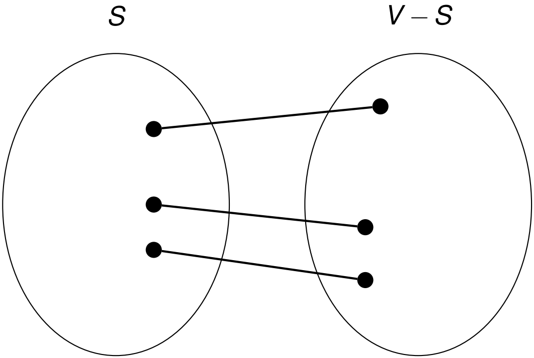
What is a Cut?

What is a cut in an undirected graph, $G = (V, E)$?

(A) A set of edges whose removal disconnects a graph.

(B) For partition of V, $(S, V - S)$, set of edges across it; $E \cap (S \times V - S)$.

(A) and (B).



Note:sometimes specified as $(S, V - S)$
..... sometimes explicitly as subset of edges E'.

What is the cut property for MSTs?

(A) Any edges in a cut is in some mst.

(B) The smallest edge in a cut is in some mst.

(C) The largest edge in a cut cannot be in an mst.

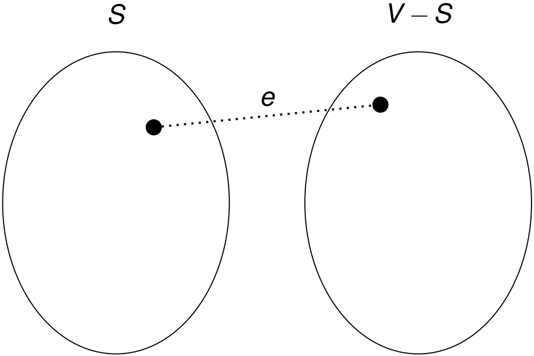
What is the cut property for MSTs?

(A) Any edges in a cut is in some mst.

(B) The smallest edge in a cut is in some mst.

(C) The largest edge in a cut cannot be in an mst.

B.



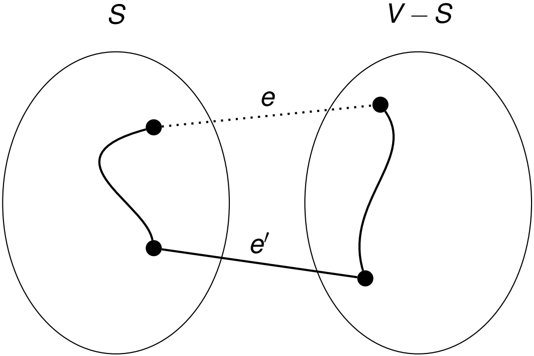
What is the cut property for MSTs?

(A) Any edges in a cut is in some mst.

(B) The smallest edge in a cut is in some mst.

(C) The largest edge in a cut cannot be in an mst.

B.



Cut Property: Any edge of minimal weight in a cut is in some MST.
(If unique, it must be in MST.)

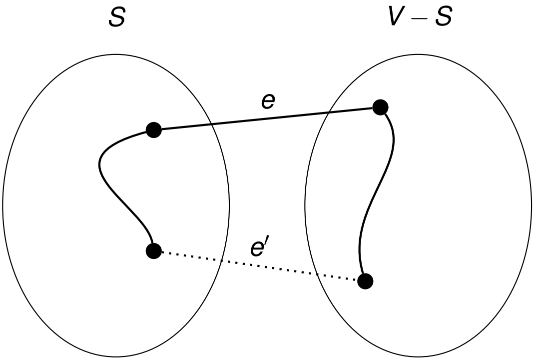
What is the cut property for MSTs?

(A) Any edges in a cut is in some mst.

(B) The smallest edge in a cut is in some mst.

(C) The largest edge in a cut cannot be in an mst.

B.



Cut Property: Any edge of minimal weight in a cut is in some MST.
(If unique, it must be in MST.)
Proof: replace,

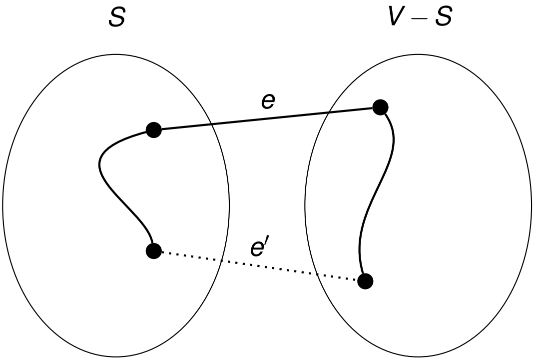
What is the cut property for MSTs?

(A) Any edges in a cut is in some mst.

(B) The smallest edge in a cut is in some mst.

(C) The largest edge in a cut cannot be in an mst.

B.



Cut Property: Any edge of minimal weight in a cut is in some MST.
(If unique, it must be in MST.)
Proof: replace, $n-1$ edges

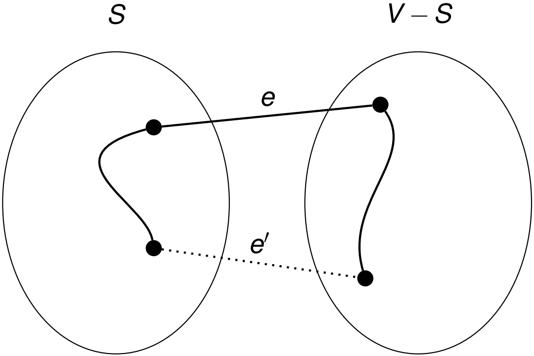
What is the cut property for MSTs?

(A) Any edges in a cut is in some mst.

(B) The smallest edge in a cut is in some mst.

(C) The largest edge in a cut cannot be in an mst.

B.



Cut Property: Any edge of minimal weight in a cut is in some MST.
(If unique, it must be in MST.)
Proof: replace, $n - 1$ edges still

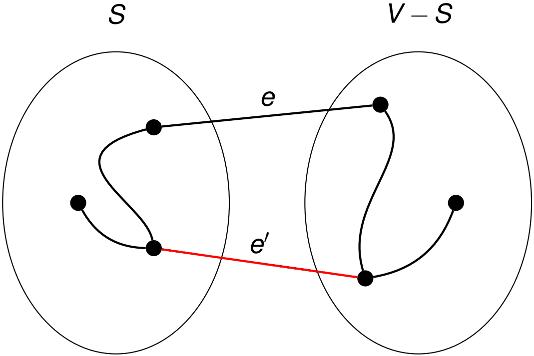
What is the cut property for MSTs?

(A) Any edges in a cut is in some mst.

(B) The smallest edge in a cut is in some mst.

(C) The largest edge in a cut cannot be in an mst.

B.



Cut Property: Any edge of minimal weight in a cut is in some MST.
(If unique, it must be in MST.)
Proof: replace, $n-1$ edges still

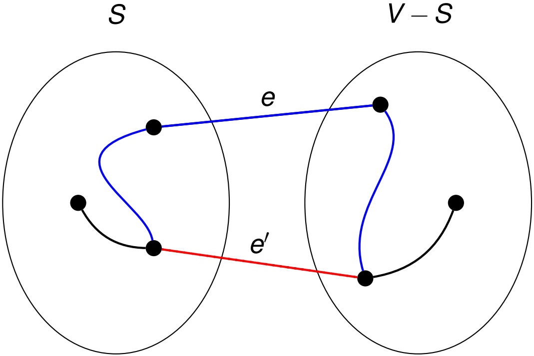
What is the cut property for MSTs?

(A) Any edges in a cut is in some mst.

(B) The smallest edge in a cut is in some mst.

(C) The largest edge in a cut cannot be in an mst.

B.



Cut Property: Any edge of minimal weight in a cut is in some MST.
(If unique, it must be in MST.)
Proof: replace, $n-1$ edges still connected

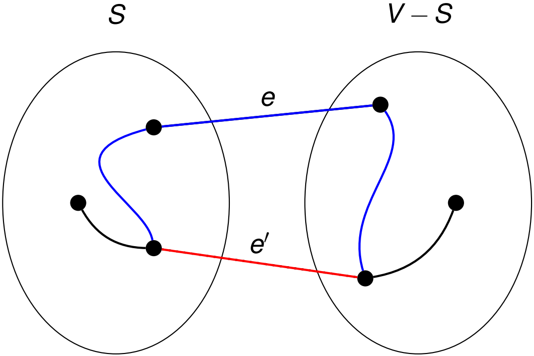
What is the cut property for MSTs?

(A) Any edges in a cut is in some mst.

(B) The smallest edge in a cut is in some mst.

(C) The largest edge in a cut cannot be in an mst.

B.



Cut Property: Any edge of minimal weight in a cut is in some MST.
(If unique, it must be in MST.)
Proof: replace, $n-1$ edges still connected and cheaper.

Prim's algorithm

Cut Property: Any edge of minimal weight in a cut is in some MST.
(If unique, it must be in MST.)

Prim's algorithm

Cut Property: Any edge of minimal weight in a cut is in some MST. (If unique, it must be in MST.)

Generic MST:

Prim's algorithm

Cut Property: Any edge of minimal weight in a cut is in some MST.
(If unique, it must be in MST.)

Generic MST:
While not connected.

Prim's algorithm

Cut Property: Any edge of minimal weight in a cut is in some MST.
(If unique, it must be in MST.)

Generic MST:
While not connected.
  Add smallest edge across a cut.

Prim's algorithm

Cut Property: Any edge of minimal weight in a cut is in some MST.
(If unique, it must be in MST.)

Generic MST:
While not connected.
  Add smallest edge across a cut.

Correctness:

Prim's algorithm

Cut Property: Any edge of minimal weight in a cut is in some MST.
(If unique, it must be in MST.)

Generic MST:
While not connected.
  Add smallest edge across a cut.

Correctness: cut property

Prim's algorithm

Cut Property: Any edge of minimal weight in a cut is in some MST.
(If unique, it must be in MST.)

Generic MST:
While not connected.
  Add smallest edge across a cut.

Correctness: cut property with unique weight edges.

Prim's algorithm

Cut Property: Any edge of minimal weight in a cut is in some MST.
(If unique, it must be in MST.)

Generic MST:
While not connected.
  Add smallest edge across a cut.

Correctness: cut property with unique weight edges.

Break ties for smallest edge according to lowest neighbors.

Prim's Algorithm.

Generic MST:

Prim's Algorithm.

Generic MST:
Start with $S = v$:

Prim's Algorithm.

Generic MST:

Start with $S = v$:

Find smallest edge (x, y) in crossing $(S, V - S)$

let $S = S \cup \{y\}$

Prim's Algorithm.

Generic MST:
Start with $S = v$:
  Find smallest edge (x, y) in crossing $(S, V - S)$
    let $S = S \cup \{y\}$

Implementation?

Prim's Algorithm.

Generic MST:
Start with $S = v$:
  Find smallest edge (x, y) in crossing $(S, V - S)$
    let $S = S \cup \{y\}$

Implementation?

Find smallest edge (x, y)?

Prim's Algorithm.

Generic MST:
Start with $S = v$:
  Find smallest edge (x, y) in crossing $(S, V - S)$
    let $S = S \cup \{y\}$

Implementation?

Find smallest edge (x, y)? $O(m)$ time.

Prim's Algorithm.

Generic MST:
Start with $S = v$:
  Find smallest edge (x, y) in crossing $(S, V - S)$
    let $S = S \cup \{y\}$

Implementation?

Find smallest edge (x, y)? $O(m)$ time.

$O(nm)$ time.

Prim's Algorithm.

Generic MST:
Start with $S = v$:
  Find smallest edge (x, y) in crossing $(S, V - S)$
    let $S = S \cup \{y\}$

Implementation?

Find smallest edge (x, y)? $O(m)$ time.

$O(nm)$ time.

Look at same edges over and over again!

Prim's Algorithm.

Generic MST:

Start with $S = v$:

Find smallest edge (x, y) in crossing $(S, V - S)$

let $S = S \cup \{y\}$

Implementation?

Find smallest edge (x, y)? $O(m)$ time.

$O(nm)$ time.

Look at same edges over and over again!

Use a priority queue to keep edges!

Prim's Algorithm.

Generic MST:
Start with $S = v$:
  Find smallest edge (x, y) in crossing $(S, V - S)$
    let $S = S \cup \{y\}$

Implementation?

Find smallest edge (x, y)? $O(m)$ time.

$O(nm)$ time.

Look at same edges over and over again!

Use a priority queue to keep edges!

Actually...

Prim's Algorithm.

Generic MST:
Start with $S = v$:
  Find smallest edge (x, y) in crossing $(S, V - S)$
    let $S = S \cup \{y\}$

Implementation?

Find smallest edge (x, y)? $O(m)$ time.

$O(nm)$ time.

Look at same edges over and over again!

Use a priority queue to keep edges!

Actually... use a priority queue to keep "closest" node.

Prim's Algorithm

Prim(G,s)

foreach $v \in V$: $c(v) = \infty, prev(v) = nil$

c(s) = 0, prev(s) = s

Prim's Algorithm

Prim(G,s)

foreach $v \in V$: $c(v) = \infty, prev(v) = nil$

c(s) = 0, prev(s) = s

H = make_pqueue(V,c)

Prim's Algorithm

```
Prim(G,s)
  foreach v ∈ V: c(v) = ∞, prev(v) = nil
  c(s) = 0, prev(s) = s

  H = make_pqueue(V,c)
  while (v = deletemin(H)):
```

Prim's Algorithm

Prim(G,s)
foreach $v \in V$: $c(v) = \infty, prev(v) = nil$
c(s) = 0, prev(s) = s

H = make_pqueue(V,c)
while (v = deletemin(H)):
foreach (v,w):

Prim's Algorithm

```
Prim(G,s)
  foreach v ∈ V: c(v) = ∞, prev(v) = nil
  c(s) = 0, prev(s) = s

  H = make_pqueue(V,c)
  while  (v = deletemin(H)):
     foreach (v,w):
        if (w(v,w) < c(w)):
             c(w) = w(v,w), prev(w) = v
```

Prim's Algorithm

```
Prim(G,s)
  foreach v ∈ V: c(v) = ∞, prev(v) = nil
  c(s) = 0, prev(s) = s

  H = make_pqueue(V,c)
  while (v = deletemin(H)):
     foreach (v,w):
        if (w(v,w) < c(w)):
             c(w) = w(v,w), prev(w) = v
             decreaseKey(H,w)
```

Prim's Algorithm

Prim(G,s)
  foreach $v \in V$: $c(v) = \infty, prev(v) = nil$
  c(s) = 0, prev(s) = s

  H = make_pqueue(V,c)
  while (v = deletemin(H)):
    foreach (v,w):
      if $(w(v, w) < c(w))$:
        $c(w) = w(v, w), prev(w) = v$
        decreaseKey(H,w)

Dijkstras:

Prim's Algorithm

Prim(G,s)
  foreach $v \in V$: $c(v) = \infty, prev(v) = nil$
  c(s) = 0, prev(s) = s

  H = make_pqueue(V,c)
  while (v = deletemin(H)):
    foreach (v,w):
      if $(w(v,w) < c(w))$:
        $c(w) = w(v,w), prev(w) = v$
        decreaseKey(H,w)

Dijkstras:

      if $c(v)+w(v,w) < c(w))$:
        $c(w) = c(v) + w(v,w), prev(w) = v$

Prim's Algorithm

Prim(G,s)
  foreach $v \in V$: $c(v) = \infty, prev(v) = nil$
  c(s) = 0, prev(s) = s

  H = make_pqueue(V,c)
  while (v = deletemin(H)):
    foreach (v,w):
      if $(w(v,w) < c(w))$:
        $c(w) = w(v,w), prev(w) = v$
        decreaseKey(H,w)

Dijkstras:

      if $c(v) + w(v,w) < c(w))$:
        $c(w) = c(v) + w(v,w), prev(w) = v$

Runtime?

Prim's Algorithm

Prim(G,s)
foreach $v \in V$: $c(v) = \infty, prev(v) = nil$
c(s) = 0, prev(s) = s

H = make_pqueue(V,c)
while (v = deletemin(H)):
foreach (v,w):
if $(w(v,w) < c(w))$:
$c(w) = w(v,w), prev(w) = v$
decreaseKey(H,w)

Dijkstras:

if $c(v) + w(v,w) < c(w)$):
$c(w) = c(v) + w(v,w), prev(w) = v$

Runtime? $\Theta(mn)$?

Prim's Algorithm

Prim(G,s)
  foreach $v \in V$: $c(v) = \infty, prev(v) = nil$
  c(s) = 0, prev(s) = s

  H = make_pqueue(V,c)
  while (v = deletemin(H)):
    foreach (v,w):
      if $(w(v,w) < c(w))$:
        $c(w) = w(v,w), prev(w) = v$
        decreaseKey(H,w)

Dijkstras:

  if $c(v) + w(v,w) < c(w))$:
    $c(w) = c(v) + w(v,w), prev(w) = v$

Runtime? $\Theta(mn)$? $\Theta((m+n)\log n)$?

Prim's Algorithm

Prim(G,s)
  foreach $v \in V$: $c(v) = \infty, prev(v) = nil$
  c(s) = 0, prev(s) = s

  H = make_pqueue(V,c)
  while (v = deletemin(H)):
    foreach (v,w):
      if $(w(v,w) < c(w))$:
        $c(w) = w(v,w), prev(w) = v$
        decreaseKey(H,w)

Dijkstras:

      if $c(v) + w(v,w) < c(w))$:
        $c(w) = c(v) + w(v,w), prev(w) = v$

Runtime? $\Theta(mn)$? $\Theta((m+n)\log n)$? $\Theta(m+n\log n)$?

Prim's Algorithm

Prim(G,s)
foreach $v \in V$: $c(v) = \infty, prev(v) = nil$
c(s) = 0, prev(s) = s

H = make_pqueue(V,c)
while (v = deletemin(H)):
foreach (v,w):
if $(w(v,w) < c(w))$:
$c(w) = w(v,w), prev(w) = v$
decreaseKey(H,w)

Dijkstras:

if $c(v) + w(v,w) < c(w))$:
$c(w) = c(v) + w(v,w), prev(w) = v$

Runtime? $\Theta(mn)$? $\Theta((m+n)\log n)$? $\Theta(m+n\log n)$? $O((m+n)\log n)$

Prim's Algorithm

Prim(G,s)
foreach $v \in V$: $c(v) = \infty, prev(v) = nil$
c(s) = 0, prev(s) = s

H = make_pqueue(V,c)
while (v = deletemin(H)):
foreach (v,w):
if $(w(v,w) < c(w))$:
$c(w) = w(v,w), prev(w) = v$
decreaseKey(H,w)

Dijkstras:

if $c(v)+w(v,w) < c(w))$:
$c(w) = c(v) + w(v,w), prev(w) = v$

Runtime? $\Theta(mn)$? $\Theta((m+n)\log n)$? $\Theta(m+n\log n)$? $O((m+n)\log n)$

With Fibonacci Heaps: $O(m+n\log n)$.

Compression.

Given a long file, make it shorter.

Compression.

Given a long file, make it shorter.

16 characters alphabet, four bits/character.

Compression.

Given a long file, make it shorter.

16 characters alphabet, four bits/character.
One Idea: shorter representation of frequent characters.

Compression.

Given a long file, make it shorter.

16 characters alphabet, four bits/character.
One Idea: shorter representation of frequent characters.

Morse code:

Compression.

Given a long file, make it shorter.

16 characters alphabet, four bits/character.
One Idea: shorter representation of frequent characters.

Morse code:

E . "dot"
T - "dash"
I .. "dot dot"
A .- "dot dash"
N -. "dash dot"
S ... "dot dot dot"
…

Compression.

Given a long file, make it shorter.

16 characters alphabet, four bits/character.
One Idea: shorter representation of frequent characters.

Morse code:

E . “dot”
T - “dash”
I .. “dot dot”
A .- “dot dash”
N -. “dash dot”
S ... “dot dot dot”
...

Common Characters are shorter...

Compression.

Given a long file, make it shorter.

16 characters alphabet, four bits/character.
One Idea: shorter representation of frequent characters.

Morse code:

E . “dot”
T - “dash”
I .. “dot dot”
A .- “dot dash”
N -. “dash dot”
S ... “dot dot dot”
...

Common Characters are shorter...
Cool!

Compression.

Given a long file, make it shorter.

16 characters alphabet, four bits/character.
One Idea: shorter representation of frequent characters.

Morse code:

E . “dot”
T - “dash”
I .. “dot dot”
A .- “dot dash”
N -. “dash dot”
S ... “dot dot dot”
...

Common Characters are shorter...
Cool! Translate to 0 and 1?

Compression.

Given a long file, make it shorter.

16 characters alphabet, four bits/character.
One Idea: shorter representation of frequent characters.

Morse code:

E . “dot”
T - “dash”
I .. “dot dot”
A .- “dot dash”
N -. “dash dot”
S ... “dot dot dot”
...

Common Characters are shorter...
Cool! Translate to 0 and 1? Sure.

Compression.

Given a long file, make it shorter.

16 characters alphabet, four bits/character.
One Idea: shorter representation of frequent characters.

Morse code:

E . “dot”
T - “dash”
I .. “dot dot”
A .- “dot dash”
N -. “dash dot”
S ... “dot dot dot”
...

Common Characters are shorter...
Cool! Translate to 0 and 1? Sure.

What is .. or “dot dot”?
“I”

Compression.

Given a long file, make it shorter.

16 characters alphabet, four bits/character.
One Idea: shorter representation of frequent characters.

Morse code:

E . "dot"
T - "dash"
I .. "dot dot"
A .- "dot dash"
N -. "dash dot"
S ... "dot dot dot"
...

Common Characters are shorter...
Cool! Translate to 0 and 1? Sure.

What is **..** or "dot dot"?
"I" or "EE" ??

Compression.

Given a long file, make it shorter.

16 characters alphabet, four bits/character.
One Idea: shorter representation of frequent characters.

Morse code:

E . “dot”
T - “dash”
I .. “dot dot”
A .- “dot dash”
N -. “dash dot”
S ... “dot dot dot”
...

Common Characters are shorter...
Cool! Translate to 0 and 1? Sure.

What is .. or “dot dot”?
“I” or “EE” ??

Separate using pauses in morse code..

Compression.

Given a long file, make it shorter.

16 characters alphabet, four bits/character.
One Idea: shorter representation of frequent characters.

Morse code:

E . “dot”
T - “dash”
I .. “dot dot”
A .- “dot dash”
N -. “dash dot”
S ... “dot dot dot”
...

Common Characters are shorter...
Cool! Translate to 0 and 1? Sure.

What is **..** or “dot dot”?
“I” or “EE” ??

Separate using pauses in morse code.. for binary?

Prefix free codes.

No code for a letter is a prefix of another.

Prefix free codes.

No code for a letter is a prefix of another.

Letters: A,B, C,D.

Prefix free codes.

No code for a letter is a prefix of another.

Letters: A,B, C,D.

Codewords: strings in $\{0, 1\}^*$.

Prefix free codes.

No code for a letter is a prefix of another.

Letters: A,B, C,D.

Codewords: strings in $\{0,1\}^*$.

Example:

Prefix free codes.

No code for a letter is a prefix of another.

Letters: A,B, C,D.

Codewords: strings in $\{0,1\}^*$.

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

Prefix free codes.

No code for a letter is a prefix of another.

Letters: A,B, C,D.

Codewords: strings in $\{0, 1\}^*$.

Example:
A: 00
B: 01
C: 10
D: 11
Fixed length codewords.

Prefix free codes.

No code for a letter is a prefix of another.

Letters: A,B, C,D.

Codewords: strings in $\{0,1\}^*$.

Example:
A: 00
B: 01
C: 10
D: 11
Fixed length codewords.
No codeword is prefix of another.

Prefix free codes.

No code for a letter is a prefix of another.

Letters: A,B, C,D.

Codewords: strings in $\{0,1\}^*$.

Example:
A: 00
B: 01
C: 10
D: 11
Fixed length codewords.
No codeword is prefix of another.

What is 100011?

Prefix free codes.

No code for a letter is a prefix of another.

Letters: A,B, C,D.

Codewords: strings in $\{0,1\}^*$.

Example:
A: 00
B: 01
C: 10
D: 11
Fixed length codewords.
No codeword is prefix of another.

What is 100011?
First two: “C”

Prefix free codes.

No code for a letter is a prefix of another.

Letters: A,B, C,D.

Codewords: strings in $\{0,1\}^*$.

Example:
A: 00
B: 01
C: 10
D: 11
Fixed length codewords.
No codeword is prefix of another.

What is 100011?
First two: “C”
Next two: “A”

Prefix free codes.

No code for a letter is a prefix of another.

Letters: A,B, C,D.

Codewords: strings in $\{0,1\}^*$.

Example:
A: 00
B: 01
C: 10
D: 11
Fixed length codewords.
No codeword is prefix of another.

What is 100011?
First two: “C”
Next two: “A”
Third two: “D”

Prefix free codes.

No code for a letter is a prefix of another.

Letters: A,B, C,D.

Codewords: strings in $\{0,1\}^*$.

Example:
A: 00
B: 01
C: 10
D: 11
Fixed length codewords.
No codeword is prefix of another.

What is 100011?
First two: "C"
Next two: "A"
Third two: "D"

Can decode!

Prefix freeness.

Another prefix free code for A,B,C,D.

Prefix freeness.

Another prefix free code for A,B,C,D.

$(A:0)$,

Prefix freeness.

Another prefix free code for A,B,C,D.

$(A:0),(B:10),$

Prefix freeness.

Another prefix free code for A,B,C,D.

$(A:0),(B:10),(C:110),$

Prefix freeness.

Another prefix free code for A,B,C,D.

$(A : 0), (B : 10), (C : 110), (D : 111)$

Prefix freeness.

Another prefix free code for A,B,C,D.

$(A : 0), (B : 10), (C : 110), (D : 111)$

“110010” ???

Prefix freeness.

Another prefix free code for A,B,C,D.

$(A:0),(B:10),(C:110),(D:111)$

"110010" ???

C

Prefix freeness.

Another prefix free code for A,B,C,D.

$(A:0),(B:10),(C:110),(D:111)$

"110010" ???

CA

Prefix freeness.

Another prefix free code for A,B,C,D.

$(A:0),(B:10),(C:110),(D:111)$

"110010" ???

CA B

DNA example.

Consists of letters A, C, T, G with varying frequencies.

DNA example.

Consists of letters A, C, T, G with varying frequencies.

A: .4
C: .1
T: .2
G: .3

DNA example.

Consists of letters A, C, T, G with varying frequencies.

A: .4
C: .1
T: .2
G: .3

Expected length of fixed length encoding for N chars:

DNA example.

Consists of letters A, C, T, G with varying frequencies.

A: .4
C: .1
T: .2
G: .3

Expected length of fixed length encoding for N chars: $2N$.

DNA example.

Consists of letters A, C, T, G with varying frequencies.

A: .4
C: .1
T: .2
G: .3

Expected length of fixed length encoding for N chars: $2N$.

A: .4

DNA example.

Consists of letters A, C, T, G with varying frequencies.

A: .4
C: .1
T: .2
G: .3

Expected length of fixed length encoding for N chars: $2N$.

A: .4 0
C: .1

DNA example.

Consists of letters A, C, T, G with varying frequencies.

A: .4
C: .1
T: .2
G: .3

Expected length of fixed length encoding for N chars: $2N$.

A: .4	0
C: .1	100
T: .2	

DNA example.

Consists of letters A, C, T, G with varying frequencies.

A: .4
C: .1
T: .2
G: .3

Expected length of fixed length encoding for N chars: $2N$.

A: .4	0
C: .1	100
T: .2	101
G: .3	

DNA example.

Consists of letters A, C, T, G with varying frequencies.

A: .4
C: .1
T: .2
G: .3

Expected length of fixed length encoding for N chars: $2N$.

A: .4	0
C: .1	100
T: .2	101
G: .3	11

DNA example.

Consists of letters A, C, T, G with varying frequencies.

A: .4
C: .1
T: .2
G: .3

Expected length of fixed length encoding for N chars: $2N$.

A: .4	0
C: .1	100
T: .2	101
G: .3	11

Prefix Free?

DNA example.

Consists of letters A, C, T, G with varying frequencies.

A: .4
C: .1
T: .2
G: .3

Expected length of fixed length encoding for N chars: $2N$.

A: .4	0
C: .1	100
T: .2	101
G: .3	11

Prefix Free?

0 not prefix of 100, 101, or 11.

DNA example.

Consists of letters A, C, T, G with varying frequencies.

A: .4
C: .1
T: .2
G: .3

Expected length of fixed length encoding for N chars: $2N$.

A: .4	0
C: .1	100
T: .2	101
G: .3	11

Prefix Free?

- 0 not prefix of 100, 101, or 11.
- 11 not prefix of 0, 100 or 101

DNA example.

Consists of letters A, C, T, G with varying frequencies.

A: .4
C: .1
T: .2
G: .3

Expected length of fixed length encoding for N chars: $2N$.

A: .4	0
C: .1	100
T: .2	101
G: .3	11

Prefix Free?

0 not prefix of 100, 101, or 11.
11 not prefix of 0, 100 or 101
...

DNA example.

Consists of letters A, C, T, G with varying frequencies.

A: .4
C: .1
T: .2
G: .3

Expected length of fixed length encoding for N chars: $2N$.

A: .4	0
C: .1	100
T: .2	101
G: .3	11

Prefix Free?

0 not prefix of 100, 101, or 11.
11 not prefix of 0, 100 or 101
...

DNA example.

Consists of letters A, C, T, G with varying frequencies.

A: .4
C: .1
T: .2
G: .3

Expected length of fixed length encoding for N chars: $2N$.

A: .4 0
C: .1 100
T: .2 101
G: .3 11

Prefix Free?

0 not prefix of 100, 101, or 11.
11 not prefix of 0, 100 or 101
...

Yes!

DNA example.

Consists of letters A, C, T, G with varying frequencies.

A: .4
C: .1
T: .2
G: .3

Expected length of fixed length encoding for N chars: $2N$.

A: .4	0
C: .1	100
T: .2	101
G: .3	11

Prefix Free?

0 not prefix of 100, 101, or 11.
11 not prefix of 0, 100 or 101
...

Yes!

Expected length:

DNA example.

Consists of letters A, C, T, G with varying frequencies.

A: .4
C: .1
T: .2
G: .3

Expected length of fixed length encoding for N chars: $2N$.

A: .4	0
C: .1	100
T: .2	101
G: .3	11

Prefix Free?

0 not prefix of 100, 101, or 11.
11 not prefix of 0, 100 or 101
...

Yes!

Expected length: N(.4*1

DNA example.

Consists of letters A, C, T, G with varying frequencies.

A: .4
C: .1
T: .2
G: .3

Expected length of fixed length encoding for N chars: $2N$.

A:	.4	0
C:	.1	100
T:	.2	101
G:	.3	11

Prefix Free?

0 not prefix of 100, 101, or 11.
11 not prefix of 0, 100 or 101
...

Yes!

Expected length: N(.4*1 + .1*3 +

DNA example.

Consists of letters A, C, T, G with varying frequencies.

A: .4
C: .1
T: .2
G: .3

Expected length of fixed length encoding for N chars: $2N$.

A: .4	0
C: .1	100
T: .2	101
G: .3	11

Prefix Free?

0 not prefix of 100, 101, or 11.
11 not prefix of 0, 100 or 101
...

Yes!

Expected length: N(.4*1 + .1*3 + .2 *3

DNA example.

Consists of letters A, C, T, G with varying frequencies.

A: .4
C: .1
T: .2
G: .3

Expected length of fixed length encoding for N chars: $2N$.

A: .4	0
C: .1	100
T: .2	101
G: .3	11

Prefix Free?

0 not prefix of 100, 101, or 11.
11 not prefix of 0, 100 or 101
...

Yes!

Expected length: N(.4*1 + .1*3 + .2 *3 + .3 * 2)

DNA example.

Consists of letters A, C, T, G with varying frequencies.

A: .4
C: .1
T: .2
G: .3

Expected length of fixed length encoding for N chars: $2N$.

A: .4	0
C: .1	100
T: .2	101
G: .3	11

Prefix Free?

0 not prefix of 100, 101, or 11.
11 not prefix of 0, 100 or 101
...

Yes!

Expected length: N(.4*1 + .1*3 + .2 *3 + .3 * 2) = 1.9N

DNA example.

Consists of letters A, C, T, G with varying frequencies.

A: .4
C: .1
T: .2
G: .3

Expected length of fixed length encoding for N chars: $2N$.

A: .4	0
C: .1	100
T: .2	101
G: .3	11

Prefix Free?

0 not prefix of 100, 101, or 11.
11 not prefix of 0, 100 or 101
...

Yes!

Expected length: N(.4*1 + .1*3 + .2 *3 + .3 * 2) = 1.9N

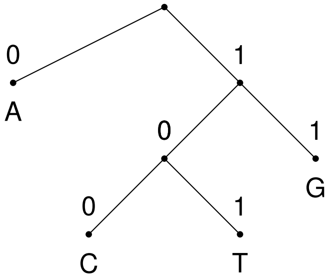
Yessss!!!!

Prefix codes and trees.

Any prefix-free code corresponds to a full binary tree:
each internal node has two children.

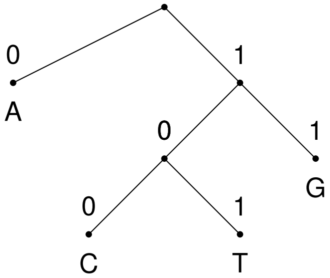
Prefix codes and trees.

Any prefix-free code corresponds to a full binary tree:
each internal node has two children.



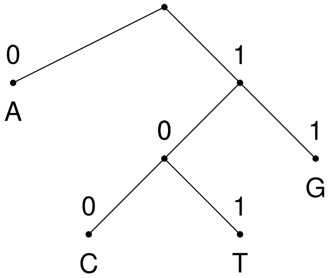
Prefix codes and trees.

Any prefix-free code corresponds to a full binary tree:
each internal node has two children.



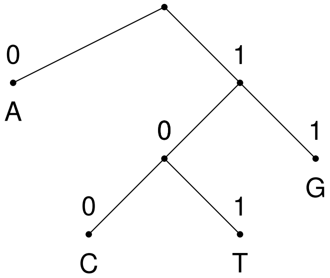
Prefix codes and trees.

Any prefix-free code corresponds to a full binary tree:
each internal node has two children.



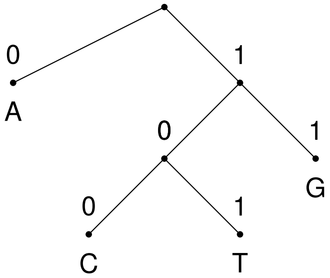
Prefix codes and trees.

Any prefix-free code corresponds to a full binary tree:
each internal node has two children.



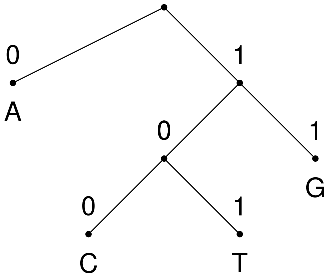
Prefix codes and trees.

Any prefix-free code corresponds to a full binary tree:
each internal node has two children.



Prefix codes and trees.

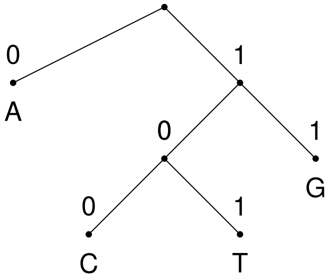
Any prefix-free code corresponds to a full binary tree:
each internal node has two children.



011101100

Prefix codes and trees.

Any prefix-free code corresponds to a full binary tree:
each internal node has two children.

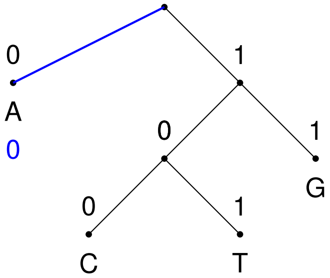


011101100

0 11 101 100

Prefix codes and trees.

Any prefix-free code corresponds to a full binary tree:
each internal node has two children.



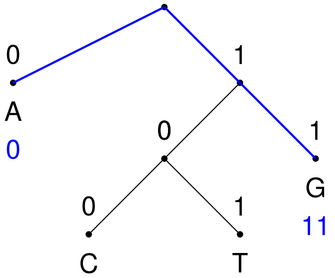
011101100

0 11 101 100

A

Prefix codes and trees.

Any prefix-free code corresponds to a full binary tree:
each internal node has two children.



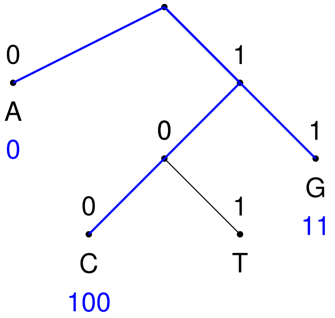
011101100

0 11 101 100

A G

Prefix codes and trees.

Any prefix-free code corresponds to a full binary tree:
each internal node has two children.



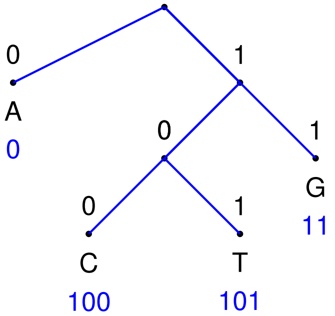
011101100

0 11 101 100

A G C

Prefix codes and trees.

Any prefix-free code corresponds to a full binary tree:
each internal node has two children.



011101100

0 11 101 100

A G C T

Tree from Prefix-Free Code.

Given prefix free code:
$S = \{s_1, s_2, \ldots, s_n\}$ for symbols $\{c_1, \ldots, c_n\}$.

Tree from Prefix-Free Code.

Given prefix free code:
$S = \{s_1, s_2, \ldots, s_n\}$ for symbols $\{c_1, \ldots, c_n\}$.

Tree from Prefix-Free Code.

Given prefix free code:
$S = \{s_1, s_2, \ldots, s_n\}$ for symbols $\{c_1, \ldots, c_n\}$.

If $\emptyset \in S$, end of a codeword.

Tree from Prefix-Free Code.

Given prefix free code:
$S = \{s_1, s_2, \ldots, s_n\}$ for symbols $\{c_1, \ldots, c_n\}$.

If $\emptyset \in S$, end of a codeword.

Else make root node for S:
Recurse.

Tree from Prefix-Free Code.

Given prefix free code:
$S = \{s_1, s_2, \ldots, s_n\}$ for symbols $\{c_1, \ldots, c_n\}$.

If $\emptyset \in S$, end of a codeword.

Else make root node for S:
Recurse.
Left: $S'_0 = \{s | 0s \in S\}$

Tree from Prefix-Free Code.

Given prefix free code:
$S = \{s_1, s_2, \ldots, s_n\}$ for symbols $\{c_1, \ldots, c_n\}$.

If $\emptyset \in S$, end of a codeword.

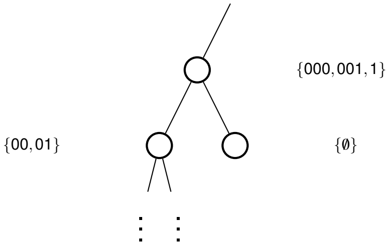
Else make root node for S:
Recurse.
Left: $S'_0 = \{s | 0s \in S\}$
Right: $S'_1 = \{s | 1s \in S\}$

Tree from Prefix-Free Code.

Given prefix free code:
$S = \{s_1, s_2, \ldots, s_n\}$ for symbols $\{c_1, \ldots, c_n\}$.

If $\emptyset \in S$, end of a codeword.

Else make root node for S:
Recurse.
Left: $S'_0 = \{s | 0s \in S\}$
Right: $S'_1 = \{s | 1s \in S\}$

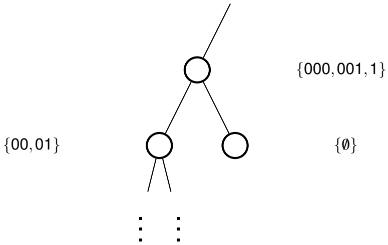


Tree from Prefix-Free Code.

Given prefix free code:
$S = \{s_1, s_2, \ldots, s_n\}$ for symbols $\{c_1, \ldots, c_n\}$.

If $\emptyset \in S$, end of a codeword.

Else make root node for S:
Recurse.
Left: $S'_0 = \{s | 0s \in S\}$
Right: $S'_1 = \{s | 1s \in S\}$



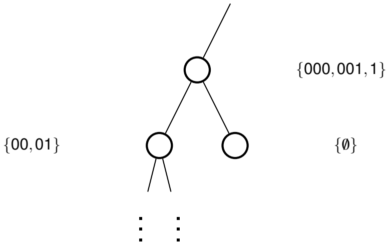
Correctness: Every codeword/symbol corresponds to leaf.

Tree from Prefix-Free Code.

Given prefix free code:
$S = \{s_1, s_2, \ldots, s_n\}$ for symbols $\{c_1, \ldots, c_n\}$.

If $\emptyset \in S$, end of a codeword.

Else make root node for S:
Recurse.
Left: $S'_0 = \{s | 0s \in S\}$
Right: $S'_1 = \{s | 1s \in S\}$



Correctness: Every codeword/symbol corresponds to leaf.

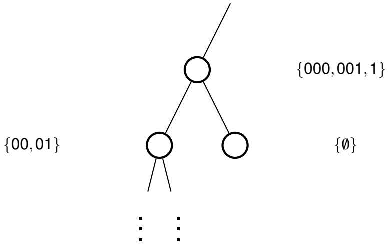
Let S_p be subset corresponding to node at “path” p in tree.

Tree from Prefix-Free Code.

Given prefix free code:
$S = \{s_1, s_2, \ldots, s_n\}$ for symbols $\{c_1, \ldots, c_n\}$.

If $\emptyset \in S$, end of a codeword.

Else make root node for S:
Recurse.
Left: $S'_0 = \{s | 0s \in S\}$
Right: $S'_1 = \{s | 1s \in S\}$



Correctness: Every codeword/symbol corresponds to leaf.

Let S_p be subset corresponding to node at "path" p in tree.

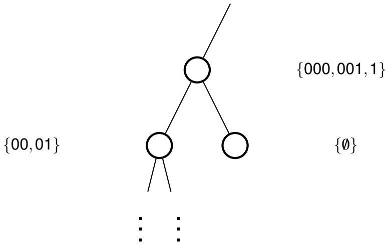
Corresponds to strings where p is prefix.

Tree from Prefix-Free Code.

Given prefix free code:
$S = \{s_1, s_2, \ldots, s_n\}$ for symbols $\{c_1, \ldots, c_n\}$.

If $\emptyset \in S$, end of a codeword.

Else make root node for S:
Recurse.
Left: $S'_0 = \{s | 0s \in S\}$
Right: $S'_1 = \{s | 1s \in S\}$



Correctness: Every codeword/symbol corresponds to leaf.

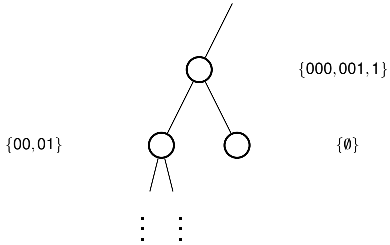
Let S_p be subset corresponding to node at "path" p in tree.

Corresponds to strings where p is prefix.

If there is internal node S_p with $p \in S$.

Tree from Prefix-Free Code.

Given prefix free code:
$S = \{s_1, s_2, \ldots, s_n\}$ for symbols $\{c_1, \ldots, c_n\}$.



If $\emptyset \in S$, end of a codeword.

Else make root node for S:
Recurse.
Left: $S'_0 = \{s | 0s \in S\}$
Right: $S'_1 = \{s | 1s \in S\}$

Correctness: Every codeword/symbol corresponds to leaf.

Let S_p be subset corresponding to node at "path" p in tree.

Corresponds to strings where p is prefix.

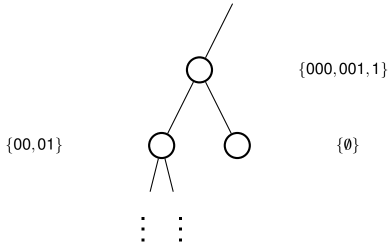
If there is internal node S_p with $p \in S$.
p is prefix of another codeword.

Tree from Prefix-Free Code.

Given prefix free code:
$S = \{s_1, s_2, \ldots, s_n\}$ for symbols $\{c_1, \ldots, c_n\}$.

If $\emptyset \in S$, end of a codeword.

Else make root node for S:
Recurse.
Left: $S'_0 = \{s | 0s \in S\}$
Right: $S'_1 = \{s | 1s \in S\}$



Correctness: Every codeword/symbol corresponds to leaf.

Let S_p be subset corresponding to node at "path" p in tree.

Corresponds to strings where p is prefix.

If there is internal node S_p with $p \in S$.
p is prefix of another codeword.
Contradiction.

Huffman Coding.

Given symbol frequencies $f_1, \ldots, f_n$, find "best" prefix code.

Huffman Coding.

Given symbol frequencies $f_1, \ldots, f_n$, find "best" prefix code.

Smallest average length.

Huffman Coding.

Given symbol frequencies $f_1, \ldots, f_n$, find "best" prefix code.

Smallest average length.

Cost of prefix tree with symbol leaves:

$$\sum_i f_i(\text{depth of symbol } i \text{ in tree.})$$

Huffman Coding.

Given symbol frequencies $f_1, \ldots, f_n$, find "best" prefix code.

Smallest average length.

Cost of prefix tree with symbol leaves:

$$\sum_i f_i (\text{depth of symbol } i \text{ in tree.})$$

Example: (A,.4), (C,.1),(T,.2),(G,.3)

Huffman Coding.

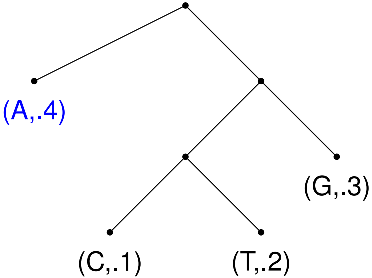
Given symbol frequencies $f_1, \ldots, f_n$, find "best" prefix code.

Smallest average length.

Cost of prefix tree with symbol leaves:

$$\sum_i f_i(\text{depth of symbol } i \text{ in tree.})$$

Example: (A,.4), (C,.1),(T,.2),(G,.3)



Cost: $.4 * 1$

Huffman Coding.

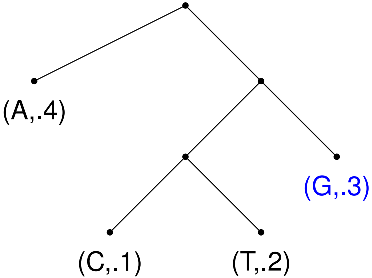
Given symbol frequencies $f_1, \ldots, f_n$, find "best" prefix code.

Smallest average length.

Cost of prefix tree with symbol leaves:

$$\sum_i f_i(\text{depth of symbol } i \text{ in tree.})$$

Example: (A,.4), (C,.1),(T,.2),(G,.3)



Cost: $.4 * 1 + .3 * 2$

Huffman Coding.

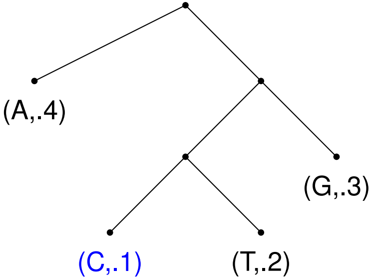
Given symbol frequencies $f_1, \ldots, f_n$, find "best" prefix code.

Smallest average length.

Cost of prefix tree with symbol leaves:

$$\sum_i f_i(\text{depth of symbol } i \text{ in tree.})$$

Example: (A,.4), (C,.1),(T,.2),(G,.3)



Cost: $.4 * 1 + .3 * 2 + .1 * 3$

Huffman Coding.

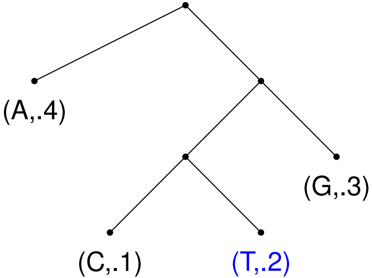
Given symbol frequencies $f_1, \ldots, f_n$, find "best" prefix code.

Smallest average length.

Cost of prefix tree with symbol leaves:

$$\sum_i f_i(\text{depth of symbol } i \text{ in tree.})$$

Example: (A,.4), (C,.1),(T,.2),(G,.3)



Cost: $.4 * 1 + .3 * 2 + .1 * 3 + .2 * 3$

Huffman Coding.

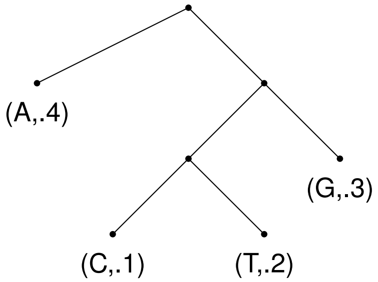
Given symbol frequencies $f_1, \ldots, f_n$, find "best" prefix code.

Smallest average length.

Cost of prefix tree with symbol leaves:

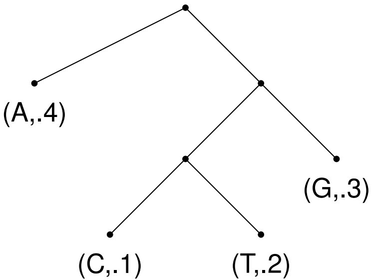
$$\sum_i f_i(\text{depth of symbol } i \text{ in tree.})$$

Example: (A,.4), (C,.1),(T,.2),(G,.3)



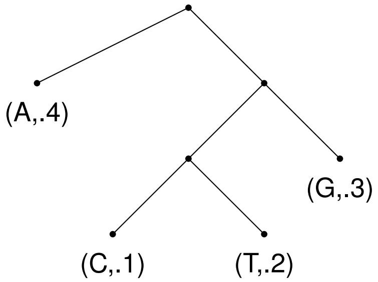
Cost: $.4*1+.3*2+.1*3+.2*3=1.9$

Another view of cost.



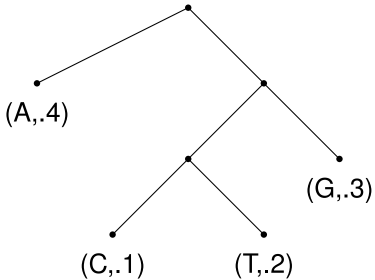
Sum over all nodes, except root, of their frequency.

Another view of cost.



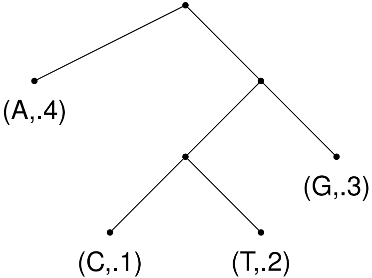
Sum over all nodes, except root, of their frequency.
.4

Another view of cost.



Sum over all nodes, except root, of their frequency.
$.4 + .1$

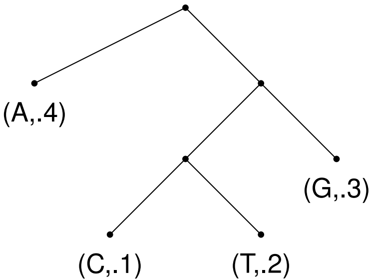
Another view of cost.



Sum over all nodes, except root, of their frequency.

$.4 + .1 + .2$

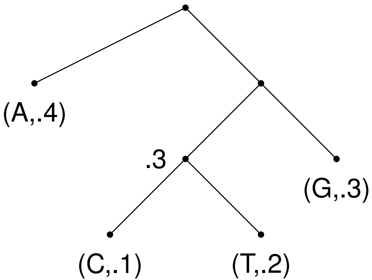
Another view of cost.



Sum over all nodes, except root, of their frequency.

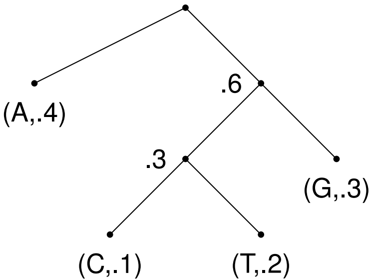
$.4 + .1 + .2 + .3$

Another view of cost.



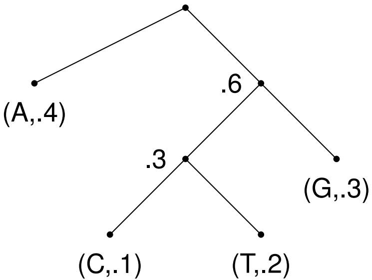
Sum over all nodes, except root, of their frequency.
$.4 + .1 + .2 + .3 + .3$

Another view of cost.



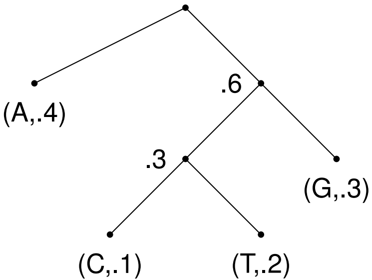
Sum over all nodes, except root, of their frequency.
$.4 + .1 + .2 + .3 + .3 + .6$

Another view of cost.



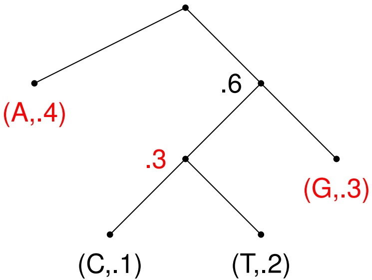
Sum over all nodes, except root, of their frequency.
$.4 + .1 + .2 + .3 + .3 + .6 = 1.9$

Another view of cost.



Sum over all nodes, except root, of their frequency.
$.4 + .1 + .2 + .3 + .3 + .6 = 1.9$
Optimal Tree should be optimal above any subtree.

Another view of cost.



Sum over all nodes, except root, of their frequency.
$.4 + .1 + .2 + .3 + .3 + .6 = 1.9$
Optimal Tree should be optimal above any subtree.
E.g., Optimal tree on $\{(.4, A), (.3, \{C, T\}), (.3, G)\}$.

Greedy Algorithm.

Greedy Algorithm.

Recursive View:

Greedy Algorithm.

Recursive View: internal node has frequency

Greedy Algorithm.

Recursive View: internal node has frequency
..."internal nodes" $\equiv$ "sort of symbols."

Greedy Algorithm.

Recursive View: internal node has frequency
..."internal nodes" $\equiv$ "sort of symbols."

Idea: Merge two symbols to make new internal node (symbol).

Greedy Algorithm.

Recursive View: internal node has frequency
..."internal nodes" $\equiv$ "sort of symbols."

Idea: Merge two symbols to make new internal node (symbol).

Which ones?

Greedy Algorithm.

Recursive View: internal node has frequency
..."internal nodes" $\equiv$ "sort of symbols."

Idea: Merge two symbols to make new internal node (symbol).

Which ones?

Cost of prefix tree with symbol leaves:

Greedy Algorithm.

Recursive View: internal node has frequency
..."internal nodes" $\equiv$ "sort of symbols."

Idea: Merge two symbols to make new internal node (symbol).

Which ones?

Cost of prefix tree with symbol leaves:

$$\sum_i f_i(\text{depth of symbol } i \text{ in tree.})$$

Greedy Algorithm.

Recursive View: internal node has frequency
..."internal nodes" $\equiv$ "sort of symbols."

Idea: Merge two symbols to make new internal node (symbol).

Which ones?

Cost of prefix tree with symbol leaves:

$$\sum_i f_i(\text{depth of symbol } i \text{ in tree.})$$

Might as well merge two lowest frequency symbols...

Greedy Algorithm.

Recursive View: internal node has frequency
..."internal nodes" $\equiv$ "sort of symbols."

Idea: Merge two symbols to make new internal node (symbol).

Which ones?

Cost of prefix tree with symbol leaves:

$$\sum_i f_i(\text{depth of symbol } i \text{ in tree.})$$

Might as well merge two lowest frequency symbols...
to make low freq internal symbol.

Greedy Algorithm.

Recursive View: internal node has frequency
..."internal nodes" $\equiv$ "sort of symbols."

Idea: Merge two symbols to make new internal node (symbol).

Which ones?

Cost of prefix tree with symbol leaves:

$$\sum_i f_i (\text{depth of symbol } i \text{ in tree.})$$

Might as well merge two lowest frequency symbols...
to make low freq internal symbol.

Let's see method!

Greedy Algorithm.

Recursive View: internal node has frequency
..."internal nodes" $\equiv$ "sort of symbols."

Idea: Merge two symbols to make new internal node (symbol).

Which ones?

Cost of prefix tree with symbol leaves:

$$\sum_i f_i(\text{depth of symbol } i \text{ in tree.})$$

Might as well merge two lowest frequency symbols...
to make low freq internal symbol.

Let's see method!

$(A, .4), (C, .1), (T, .2), (G, .3)$

Greedy Algorithm.

Recursive View: internal node has frequency
..."internal nodes" $\equiv$ "sort of symbols."

Idea: Merge two symbols to make new internal node (symbol).

Which ones?

Cost of prefix tree with symbol leaves:

$$\sum_i f_i(\text{depth of symbol } i \text{ in tree.})$$

Might as well merge two lowest frequency symbols...
to make low freq internal symbol.

Let's see method!

$(A, .4), (C, .1), (T, .2), (G, .3)$

$(A, .4), (\{C, T\}, .3), (G, .3)$

Greedy Algorithm.

Recursive View: internal node has frequency
..."internal nodes" $\equiv$ "sort of symbols."

Idea: Merge two symbols to make new internal node (symbol).

Which ones?

Cost of prefix tree with symbol leaves:

$$\sum_i f_i(\text{depth of symbol } i \text{ in tree.})$$

Might as well merge two lowest frequency symbols...
to make low freq internal symbol.

Let's see method!

$(A, .4), (C, .1), (T, .2), (G, .3)$

$(A, .4), (\{C, T\}, .3), (G, .3)$

$(A, .4), (\{\{C, T\}, G\}.6)$

Greedy Algorithm.

Recursive View: internal node has frequency
..."internal nodes" $\equiv$ "sort of symbols."

Idea: Merge two symbols to make new internal node (symbol).

Which ones?

Cost of prefix tree with symbol leaves:

$$\sum_i f_i(\text{depth of symbol } i \text{ in tree.})$$

Might as well merge two lowest frequency symbols...
to make low freq internal symbol.

Let's see method!

$(A, .4), (C, .1), (T, .2), (G, .3)$

$(A, .4), (\{C, T\}, .3), (G, .3)$

$(A, .4), (\{\{C, T\}, G\}.6)$

$(\{A, \{\{C, T\}, G\}\}, 1)$

Algorithm.

Cost2: Sum over all nodes, except root, of their “frequency”.

Algorithm.

Cost2: Sum over all nodes, except root, of their “frequency”.

Algorithm:

Algorithm.

Cost2: Sum over all nodes, except root, of their “frequency”.

Algorithm:
Make each symbol into single node tree.

Algorithm.

Cost2: Sum over all nodes, except root, of their “frequency”.

Algorithm:
Make each symbol into single node tree.
While more than one tree:

Algorithm.

Cost2: Sum over all nodes, except root, of their “frequency”.

Algorithm:
Make each symbol into single node tree.
While more than one tree:
    Merge two lowest frequency trees, into a new tree.

Algorithm.

Cost2: Sum over all nodes, except root, of their “frequency”.

Algorithm:
Make each symbol into single node tree.
While more than one tree:
    Merge two lowest frequency trees, into a new tree.

(A,.4)

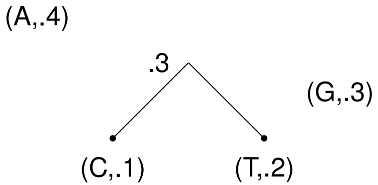
(G,.3)

(C,.1)   (T,.2)

Algorithm.

Cost2: Sum over all nodes, except root, of their "frequency".

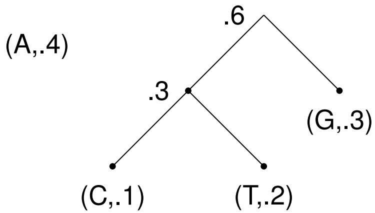
Algorithm:
Make each symbol into single node tree.
While more than one tree:
  Merge two lowest frequency trees, into a new tree.



Algorithm.

Cost2: Sum over all nodes, except root, of their “frequency”.

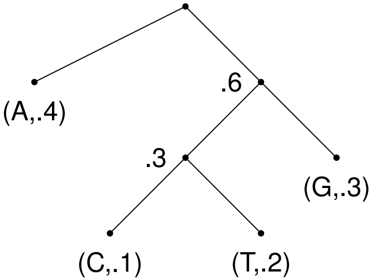
Algorithm:
Make each symbol into single node tree.
While more than one tree:
    Merge two lowest frequency trees, into a new tree.



Algorithm.

Cost2: Sum over all nodes, except root, of their "frequency".

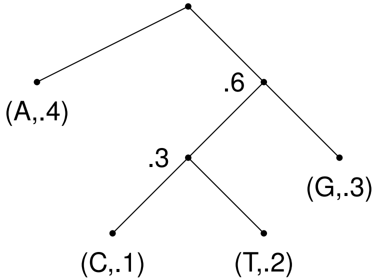
Algorithm:
Make each symbol into single node tree.
While more than one tree:
    Merge two lowest frequency trees, into a new tree.



Algorithm.

Cost2: Sum over all nodes, except root, of their “frequency”.

Algorithm:
Make each symbol into single node tree.
While more than one tree:
    Merge two lowest frequency trees, into a new tree.



Implementation: priority queue to get lowest frequency trees.

Correctness..

Recall MST: added a “could have” edge in tree every time.

Correctness..

Recall MST: added a “could have” edge in tree every time.

“must have” if no ties.

Correctness..

Recall MST: added a “could have” edge in tree every time.

“must have” if no ties.

Huffman algorithm: merge two lowest frequency symbols.

Correctness..

Recall MST: added a “could have” edge in tree every time.
   “must have” if no ties.

Huffman algorithm: merge two lowest frequency symbols.
   Make supersymbol.

Correctness..

Recall MST: added a “could have” edge in tree every time.
  “must have” if no ties.

Huffman algorithm: merge two lowest frequency symbols.
  Make supersymbol. Recurse.

Correctness..

Recall MST: added a “could have” edge in tree every time.
“must have” if no ties.

Huffman algorithm: merge two lowest frequency symbols.
Make supersymbol. Recurse.

Correctness:
Exists: optimal tree where two lowest frequency symbols are siblings.

Correctness..

Recall MST: added a “could have” edge in tree every time.
  “must have” if no ties.

Huffman algorithm: merge two lowest frequency symbols.
  Make supersymbol. Recurse.

Correctness:
 Exists: optimal tree where two lowest frequency symbols are siblings.

Correctness..

Recall MST: added a “could have” edge in tree every time.
“must have” if no ties.

Huffman algorithm: merge two lowest frequency symbols.
Make supersymbol. Recurse.

Correctness:
Exists: optimal tree where two lowest frequency symbols are siblings.

Consider optimal tree.

Correctness..

Recall MST: added a “could have” edge in tree every time.
  “must have” if no ties.

Huffman algorithm: merge two lowest frequency symbols.
  Make supersymbol. Recurse.

Correctness:
 Exists: optimal tree where two lowest frequency symbols are siblings.

Consider optimal tree.
Consider lowest frequency two symbols (assume no ties).

Correctness..

Recall MST: added a “could have” edge in tree every time.
 “must have” if no ties.

Huffman algorithm: merge two lowest frequency symbols.
 Make supersymbol. Recurse.

Correctness:
 Exists: optimal tree where two lowest frequency symbols are siblings.

Consider optimal tree.
Consider lowest frequency two symbols (assume no ties).
If siblings

Correctness..

Recall MST: added a “could have” edge in tree every time.
“must have” if no ties.

Huffman algorithm: merge two lowest frequency symbols.
Make supersymbol. Recurse.

Correctness:
Exists: optimal tree where two lowest frequency symbols are siblings.

Consider optimal tree.
Consider lowest frequency two symbols (assume no ties).
If siblings $\rightarrow$ done.

Correctness..

Recall MST: added a “could have” edge in tree every time.
“must have” if no ties.

Huffman algorithm: merge two lowest frequency symbols.
Make supersymbol. Recurse.

Correctness:
Exists: optimal tree where two lowest frequency symbols are siblings.

Consider optimal tree.
Consider lowest frequency two symbols (assume no ties).
If siblings $\rightarrow$ done.
Otherwise ... switch each with deepest pair of siblings improves tree.

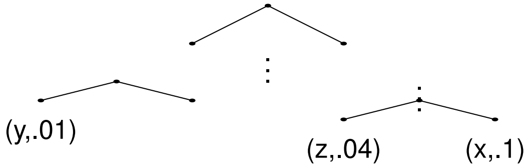
Correctness..

Recall MST: added a "could have" edge in tree every time.
"must have" if no ties.

Huffman algorithm: merge two lowest frequency symbols.
Make supersymbol. Recurse.

Correctness:
Exists: optimal tree where two lowest frequency symbols are siblings.

Consider optimal tree.
Consider lowest frequency two symbols (assume no ties).
If siblings $\rightarrow$ done.
Otherwise ... switch each with deepest pair of siblings improves tree.



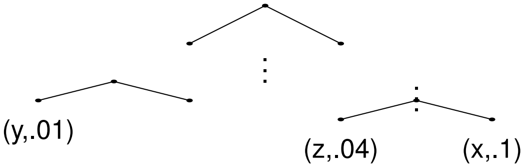
Correctness..

Recall MST: added a “could have” edge in tree every time.
“must have” if no ties.

Huffman algorithm: merge two lowest frequency symbols.
Make supersymbol. Recurse.

Correctness:
Exists: optimal tree where two lowest frequency symbols are siblings.

Consider optimal tree.
Consider lowest frequency two symbols (assume no ties).
If siblings $\rightarrow$ done.
Otherwise ... switch each with deepest pair of siblings improves tree.



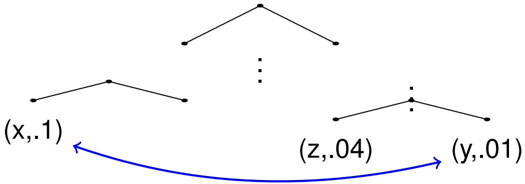
Correctness..

Recall MST: added a “could have” edge in tree every time.
“must have” if no ties.

Huffman algorithm: merge two lowest frequency symbols.
Make supersymbol. Recurse.

Correctness:
Exists: optimal tree where two lowest frequency symbols are siblings.

Consider optimal tree.
Consider lowest frequency two symbols (assume no ties).
If siblings $\rightarrow$ done.
Otherwise ... switch each with deepest pair of siblings improves tree.



Cost gets better with this switch.

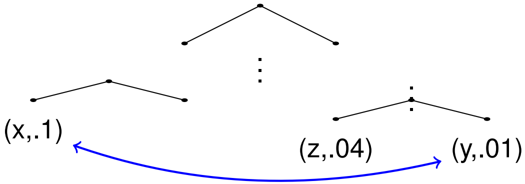
Correctness..

Recall MST: added a “could have” edge in tree every time.
“must have” if no ties.

Huffman algorithm: merge two lowest frequency symbols.
Make supersymbol. Recurse.

Correctness:
Exists: optimal tree where two lowest frequency symbols are siblings.

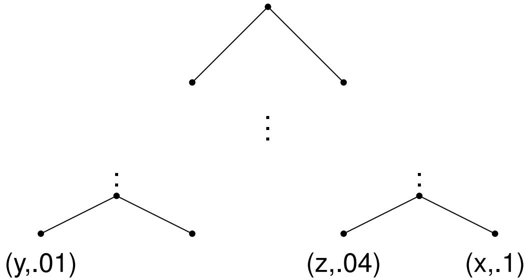
Consider optimal tree.
Consider lowest frequency two symbols (assume no ties).
If siblings $\rightarrow$ done.
Otherwise ... switch each with deepest pair of siblings improves tree.



Cost gets better with this switch.
Lowest frequency pair are now siblings!

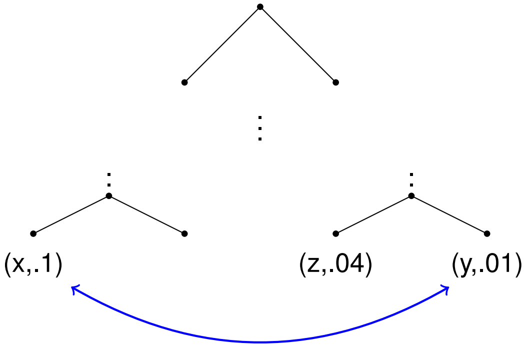
Lowest frequency at same depth.

What about same depth?



Lowest frequency at same depth.

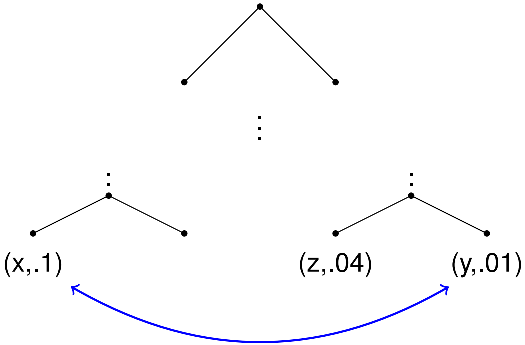
What about same depth?



Cost stays the same,

Lowest frequency at same depth.

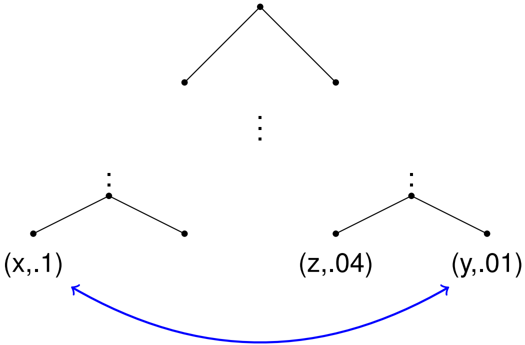
What about same depth?



Cost stays the same,
but lowest pair are siblings.

Lowest frequency at same depth.

What about same depth?

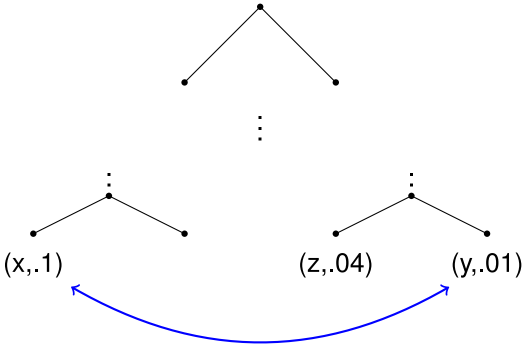


Cost stays the same,
but lowest pair are siblings.

$\Longrightarrow$ There is optimal tree where lowest frequency pair are siblings.

Lowest frequency at same depth.

What about same depth?



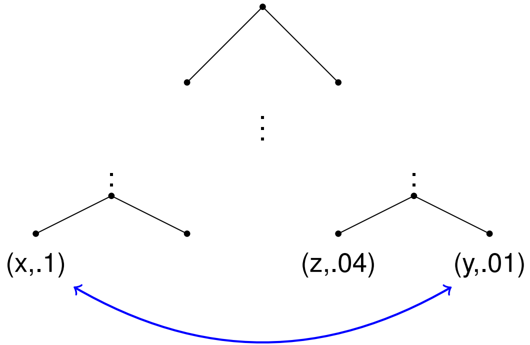
Cost stays the same,
but lowest pair are siblings.

$\Longrightarrow$ There is optimal tree where lowest frequency pair are siblings.

"Algorithm: merge lowest frequency pair, and recurse."

Lowest frequency at same depth.

What about same depth?



Cost stays the same,
but lowest pair are siblings.

$\Longrightarrow$ There is optimal tree where lowest frequency pair are siblings.

"Algorithm: merge lowest frequency pair, and recurse."

Produces optimal tree.

Lecture in a minute.

Cut Property: MST.

Exists MST that uses minimum weight edge across cut.

Exchange argument. Prim: $S = \{s\}$

Add cheapest edge (u, v) across $(S, V - S)$

$S = S + v$.

Repeat.

Use priority queue: $O((|V| + |E|)\log|V|)$.

Lecture in a minute.

Cut Property: MST.

Exists MST that uses minimum weight edge across cut.

Exchange argument. Prim: $S = \{s\}$

Add cheapest edge (u, v) across $(S, V - S)$

$S = S + v$.

Repeat.

Use priority queue: $O((|V| + |E|) \log |V|)$.

Huffman Coding.

Symbols, s, with frequencies.

Lecture in a minute.

Cut Property: MST.
- Exists MST that uses minimum weight edge across cut.
  - Exchange argument. Prim: $S = \{s\}$
 - Add cheapest edge (u, v) across $(S, V - S)$
 - $S = S + v$.
- Repeat.
- Use priority queue: $O((|V| + |E|) \log |V|)$.

Huffman Coding.
- Symbols, s, with frequencies.
- Prefix-Free code.
- $\equiv$ binary tree with symbols at leaves.
- Cost: sum of depth(s)$\times$*freq*(s).
- Cost2: sum of frequencies of internal nodes.

Lecture in a minute.

Cut Property: MST.
Exists MST that uses minimum weight edge across cut.
Exchange argument. Prim: $S = \{s\}$
Add cheapest edge (u,v) across $(S, V-S)$
$S = S + v$.
Repeat.
Use priority queue: $O((|V|+|E|)\log|V|)$.

Huffman Coding.
Symbols, s, with frequencies.
Prefix-Free code.
$\equiv$ binary tree with symbols at leaves.
Cost: sum of depth(s)$\times$*freq*(s).
Cost2: sum of frequencies of internal nodes.
Algorithm: merge lowest fequency symbols, recurse.
Exchange Argument $\implies$
exists optimal tree with this structure.