A Generic Top-Down Dynamic-Programming Approach to Prefix-Free Coding

Mordecai Golin

Hong Kong UST

Jiajin YU

Fudan Univ

Xiaoming XU

Fudan Univ

The Short Version

Prefix-Free coding is easy, isn't it?

Solvable by the $O(n)$ (greedy) Huffman algorithm.

The Short Version

Prefix-Free coding is easy, isn't it?
Solvable by the $O(n)$ (greedy) Huffman algorithm.

True for plain vanilla Huffman coding
But, add any restriction on the codes, e.g.,

- Length Limited, One-Ended, Mixed-Radix,
- Limit on # of distinct code lengths,
- Limit on # of 1's used *(Sound of Silence)*, etc.,

and Huffman algorithm fails.
Variants often approached using Dynamic Programming.

The Short Version

Prefix-Free coding is easy, isn't it?
Solvable by the $O(n)$ (greedy) Huffman algorithm.

True for plain vanilla Huffman coding
But, add any restriction on the codes, e.g.,

- Length Limited, One-Ended, Mixed-Radix,
- Limit on # of distinct code lengths,
- Limit on # of 1's used *(Sound of Silence)*, etc.,

and Huffman algorithm fails.
Variants often approached using Dynamic Programming.

This talk: a simple technique for speeding up the DP for many prefix-free coding variants.

- Introduction
  - A Quick Review of Prefix-Free Coding
  - New Results

- The Basic Top-Down Dynamic Programming Technique

- The Speedup

- Conclusion & Comments

A Quick Review

A Quick Review

Given alphabet Σ.

- *Code* $W = \{w_1, w_2, \ldots, w_n\}$ is a set of *codewords* in Σ^*.

A Quick Review

Given alphabet Σ.

- *Code* $W = \{w_1, w_2, \ldots, w_n\}$ is a set of *codewords* in Σ^*.

- Length of w is $|w|$; e.g. $|010| = 3$.

A Quick Review

Given alphabet Σ.

- *Code* $W = \{w_1, w_2, \ldots, w_n\}$ is a set of *codewords* in Σ^*.

- Length of w is $|w|$; e.g. $|010| = 3$.

- w is a prefix of w', if w is the start of w'
  e.g., 010 is a prefix of 01011001

A Quick Review

Given alphabet Σ.

- *Code* $W = \{w_1, w_2, \ldots, w_n\}$ is a set of *codewords* in Σ^*.

- Length of w is $|w|$; e.g. $|010| = 3$.

- w is a prefix of w', if w is the start of w'
  e.g., 010 is a prefix of 01011001

- W is prefix-free if $\forall w, w' \in W$, w is not a prefix of w'.

  E.g., $\begin{matrix} 10 \\ 11 \\ 01 \end{matrix}$ is prefix free; $\begin{matrix} 1 \\ 11 \\ 01 \end{matrix}$ is not prefix-free;

A Quick Review (II)

- W is prefix-free if $\forall w, w' \in W$, w is not a prefix of w'.

A Quick Review (II)

- W is prefix-free if $\forall w, w' \in W$, w is not a prefix of w'.

The Prefix Free Coding Problem

A Quick Review (II)

- W is prefix-free if $\forall w, w' \in W$, w is not a prefix of w'.

The Prefix Free Coding Problem

- Given weights $P = \{p_1, p_2, \ldots, p_n\}$.

A Quick Review (II)

- W is prefix-free if $\forall w, w' \in W$, w is not a prefix of w'.

The Prefix Free Coding Problem

- Given weights $P = \{p_1, p_2, \ldots, p_n\}$.

- Create prefix-free code $W = \{w_1, \ldots, w_n\}$ that minimizes $Cost(W, P) = \sum_{i=1}^{n} p_i |w_i|$.

A Quick Review (II)

- W is prefix-free if $\forall w, w' \in W$, w is not a prefix of w'.

The Prefix Free Coding Problem

- Given weights $P = \{p_1, p_2, \ldots, p_n\}$.

- Create prefix-free code $W = \{w_1, \ldots, w_n\}$ that minimizes $Cost(W, P) = \sum_{i=1}^{n} p_i |w_i|$.

  - Same problem as finding a tree with n leaves weighted by P that minimizes weighted external path length.

A Quick Review (III)

Correspondence between codes on $\Sigma = \{0, 1\}$ and binary trees.

(or codes on general Σ and $|\Sigma|$-ary trees)

Let 0 denote a left edge and 1 a right edge.

Codewords are leaves; Create paths to all codewords.

A Quick Review (III)

Correspondence between codes on $\Sigma = \{0, 1\}$ and binary trees.

(or codes on general Σ and $|\Sigma|$-ary trees)

Let 0 denote a left edge and 1 a right edge.

Codewords are leaves; Create paths to all codewords.

A Quick Review (III)

Correspondence between codes on $\Sigma = \{0, 1\}$ and binary trees.

(or codes on general Σ and $|\Sigma|$-ary trees)

Let 0 denote a left edge and 1 a right edge.

Codewords are leaves; Create paths to all codewords.

$w_1 = 00$
$w_2 = 10$
$w_3 = 11$
$w_4 = 010$
$w_5 = 011$

A Quick Review (III)

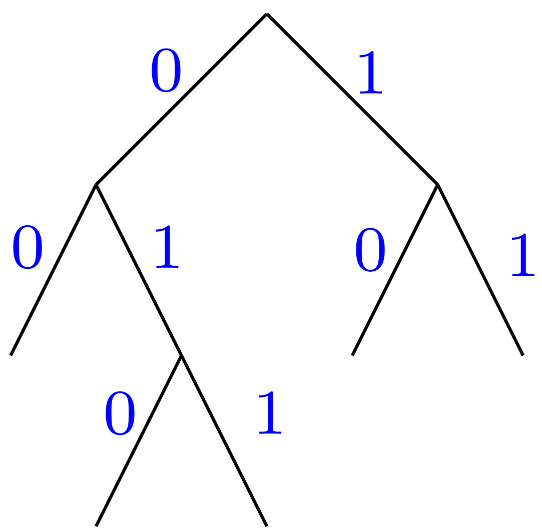
Correspondence between codes on $\Sigma = \{0, 1\}$ and binary trees.

(or codes on general Σ and $|\Sigma|$-ary trees)

Let 0 denote a left edge and 1 a right edge.

Codewords are leaves; Create paths to all codewords.

$w_1 = 00$
$w_2 = 10$
$w_3 = 11$
$w_4 = 010$
$w_5 = 011$



A Quick Review (III)

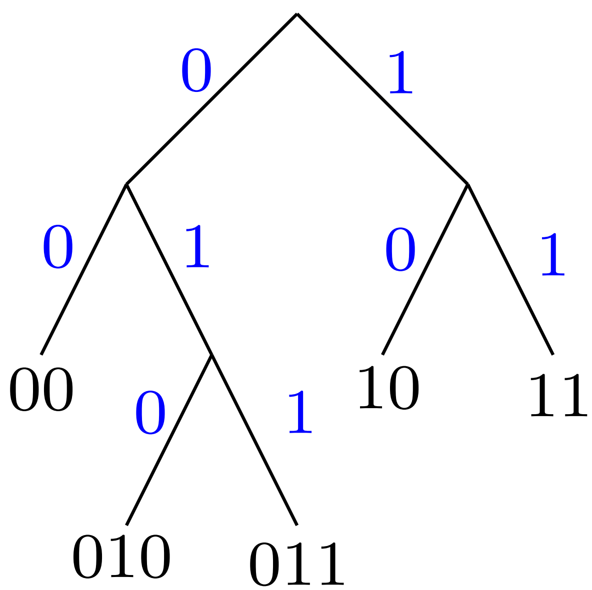
Correspondence between codes on $\Sigma = \{0, 1\}$ and binary trees.

(or codes on general Σ and $|\Sigma|$-ary trees)

Let 0 denote a left edge and 1 a right edge.

Codewords are leaves; Create paths to all codewords.

$w_1 = 00$
$w_2 = 10$
$w_3 = 11$
$w_4 = 010$
$w_5 = 011$



A Quick Review (III)

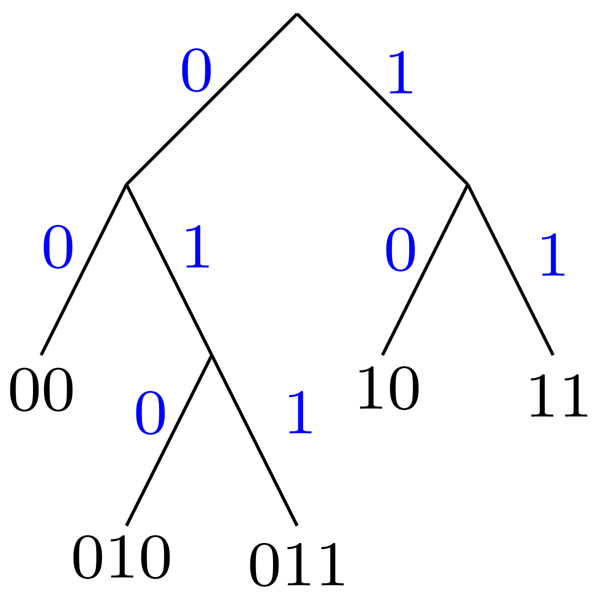
Correspondence between codes on $\Sigma = \{0, 1\}$ and binary trees.

(or codes on general Σ and $|\Sigma|$-ary trees)

Let 0 denote a left edge and 1 a right edge.

Codewords are leaves; Create paths to all codewords.

$w_1 = 00$
$w_2 = 10$
$w_3 = 11$
$w_4 = 010$
$w_5 = 011$



$|w_i|$ is depth of leaf i in tree.

A Quick Review (III)

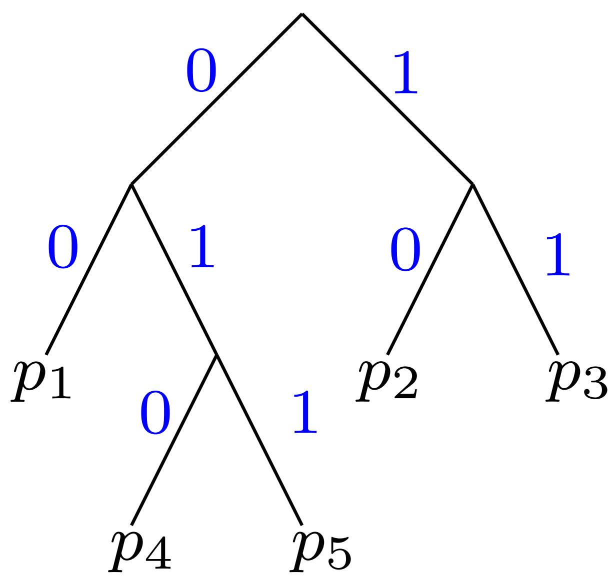
Correspondence between codes on $\Sigma = \{0, 1\}$ and binary trees.

(or codes on general Σ and $|\Sigma|$-ary trees)

Let 0 denote a left edge and 1 a right edge.

Codewords are leaves; Create paths to all codewords.

$w_1 = 00$
$w_2 = 10$
$w_3 = 11$
$w_4 = 010$
$w_5 = 011$



$|w_i|$ is depth of leaf i in tree.

Assign weight p_i to leaf i.

A Quick Review (III)

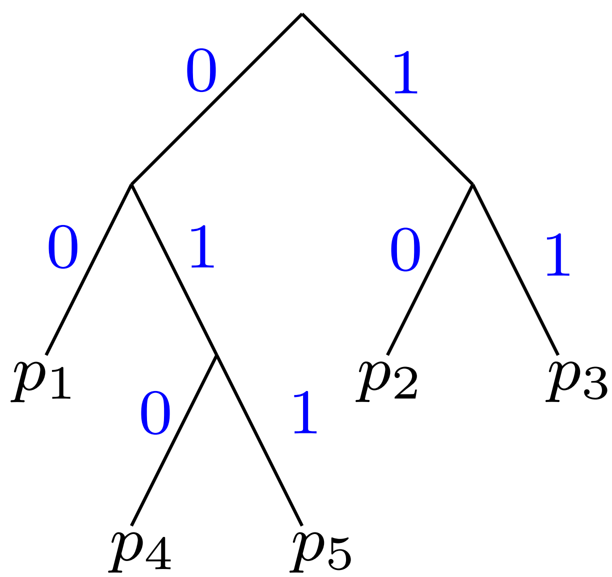
Correspondence between codes on $\Sigma = \{0, 1\}$ and binary trees.

(or codes on general Σ and $|\Sigma|$-ary trees)

Let 0 denote a left edge and 1 a right edge.

Codewords are leaves; Create paths to all codewords.

$w_1 = 00$
$w_2 = 10$
$w_3 = 11$
$w_4 = 010$
$w_5 = 011$



$|w_i|$ is depth of leaf i in tree.

Assign weight p_i to leaf i.

Weighted external path length is

$$\sum_{i=1}^{n} |w_i| p_i$$

which is $Cost(W, P)$.

A Quick Review (III)

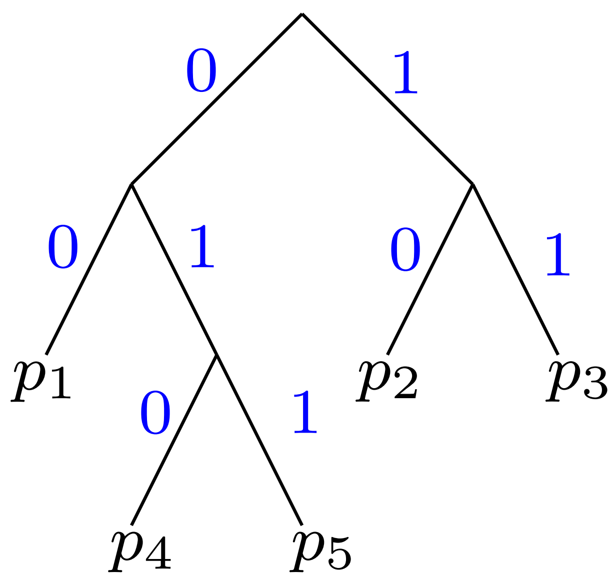
Correspondence between codes on $\Sigma = \{0,1\}$ and binary trees.

(or codes on general Σ and $|\Sigma|$-ary trees)

Let 0 denote a left edge and 1 a right edge.

Codewords are leaves; Create paths to all codewords.

$w_1 = 00$
$w_2 = 10$
$w_3 = 11$
$w_4 = 010$
$w_5 = 011$



$Cost = 2(p_1 + p_2 + p_3) + 3(p_4 + p_5)$

$|w_i|$ is depth of leaf i in tree.

Assign weight p_i to leaf i.

Weighted external path length is

$$\sum_{i=1}^{n} |w_i| p_i$$

which is $Cost(W, P)$.

A Quick Review (III)

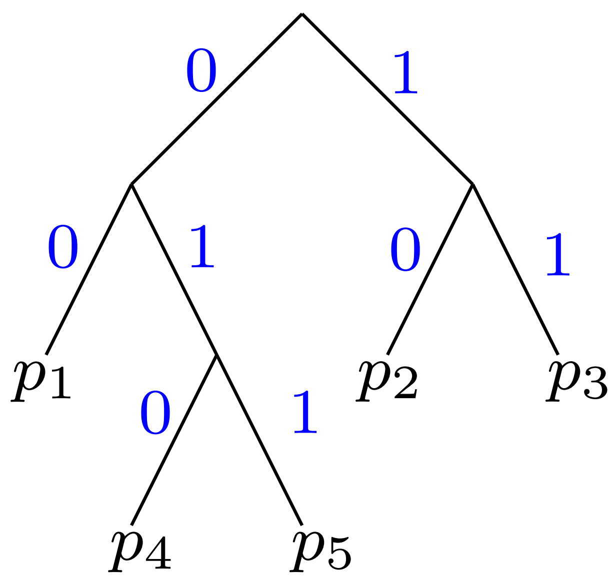
Correspondence between codes on $\Sigma = \{0, 1\}$ and binary trees.

(or codes on general Σ and $|\Sigma|$-ary trees)

Let 0 denote a left edge and 1 a right edge.

Codewords are leaves; Create paths to all codewords.

$w_1 = 00$
$w_2 = 10$
$w_3 = 11$
$w_4 = 010$
$w_5 = 011$



$Cost = 2(p_1 + p_2 + p_3) + 3(p_4 + p_5)$

$|w_i|$ is depth of leaf i in tree.

Assign weight p_i to leaf i.

Weighted external path length is

$$\sum_{i=1}^{n} |w_i| p_i$$

which is $Cost(W, P)$.

Change problem to

Given P,
Find Min-Cost Tree

- Introduction
    - A Quick Review of Prefix-Free Coding
    - New Results

- The Basic Top-Down Dynamic Programming Technique

- The Speedup

- Conclusion & Comments

Some Variants

Some Variants

- Length-Limited Coding:
  Find min-cost tree with height at most D.

Some Variants

- Length-Limited Coding:
  Find min-cost tree with height at most D.
- One-Ended Coding:
  Only use codewords that end with a 1,
  e.g., only count cost of *right-leaves*

Some Variants

- Length-Limited Coding:
  Find min-cost tree with height at most D.
- One-Ended Coding:
  Only use codewords that end with a 1,
  e.g., only count cost of *right-leaves*
- The Sound of Silence:
  Find min-cost code containing at most U 1's in each codeword,
  e.g., no tree path contains more than U right edges

Some Variants

- Length-Limited Coding:
  Find min-cost tree with height at most D.
- One-Ended Coding:
  Only use codewords that end with a 1,
  e.g., only count cost of *right-leaves*
- The Sound of Silence:
  Find min-cost code containing at most U 1's in each codeword,
  e.g., no tree path contains more than U right edges
- Reserved Length Coding:
  (i) leaves can only occur on g specified levels of the tree or
  (ii) leaves can only appear on g levels (you can choose the levels)

Some Variants

- Length-Limited Coding:
  Find min-cost tree with height at most D.
- One-Ended Coding:
  Only use codewords that end with a 1,
  e.g., only count cost of *right-leaves*
- The Sound of Silence:
  Find min-cost code containing at most U 1's in each codeword,
  e.g., no tree path contains more than U right edges
- Reserved Length Coding:
  (i) leaves can only occur on g specified levels of the tree or
  (ii) leaves can only appear on g levels (you can choose the levels)
- Mixed-Radix Coding:
  Size of alphabet depends upon position of character within codeword,
  e.g., arity of node depends upon level in the tree.

New Results

With exception of Length-Limited Coding (which takes advantage of Schieber's (1998) min-cost length-limited paths in Monge-graphs result) we improve the DP-based algorithms for all problems on previous page.

New Results

With exception of Length-Limited Coding (which takes advantage of Schieber's (1998) min-cost length-limited paths in Monge-graphs result) we improve the DP-based algorithms for all problems on previous page.

Problem	Previous Best Result	This paper
Mixed Radix Coding	$O(n^4 \log n)$ [1]	$O(n^3)$
Reserved Length Coding (i)	$O(gn^3)$ [2]	$O(gn^2)$
Reserved Length Coding (ii)	$O(g^3 n^3 \log^g n)$ [2]	$O(gn^2 \log n)$
One-ended Coding	$O(n^3)$ [3]	$O(n^2)$
The Sound of Silence	$O(n^{U+2})$ [4]	$O(n^{U+1})$

[1]: Chu and Gill (1992)
[2]: Baer (2008)
[3]: Chan and Golin (2000)
[4]: Dolev, Korach and Yukelson (1999)

New Results (II)

Problem	Previous Best Result	This paper
Mixed Radix Coding	$O(n^4 \log n)$ [1]	$O(n^3)$
Reserved Length Coding (i)	$O(gn^3)$ [2]	$O(gn^2)$
Reserved Length Coding (ii)	$O(g^3 n^3 \log^g n)$ [2]	$O(gn^2 \log n)$
One-ended Coding	$O(n^3)$ [3]	$O(n^2)$
The Sound of Silence	$O(n^{U+2})$ [4]	$O(n^{U+1})$

New Results (II)

Problem	Previous Best Result	This paper
Mixed Radix Coding	$O(n^4 \log n)$ [1]	$O(n^3)$
Reserved Length Coding (i)	$O(gn^3)$ [2]	$O(gn^2)$
Reserved Length Coding (ii)	$O(g^3 n^3 \log^g n)$ [2]	$O(gn^2 \log n)$
One-ended Coding	$O(n^3)$ [3]	$O(n^2)$
The Sound of Silence	$O(n^{U+2})$ [4]	$O(n^{U+1})$

Previous results were all based on Dynamic Programming.

DP creates a search space and calculates *optimal* cost
for every item in the search space.

Optimal cost of larger items is based on optimal cost of smaller items.

Running time of DP algorithm, is time required to calculate all costs.

New Results (II)

Problem	Previous Best Result	This paper
Mixed Radix Coding	$O(n^4 \log n)$ [1]	$O(n^3)$
Reserved Length Coding (i)	$O(gn^3)$ [2]	$O(gn^2)$
Reserved Length Coding (ii)	$O(g^3 n^3 \log^g n)$ [2]	$O(gn^2 \log n)$
One-ended Coding	$O(n^3)$ [3]	$O(n^2)$
The Sound of Silence	$O(n^{U+2})$ [4]	$O(n^{U+1})$

Previous results were all based on Dynamic Programming.

DP creates a search space and calculates *optimal* cost
for every item in the search space.

Optimal cost of larger items is based on optimal cost of smaller items.

Running time of DP algorithm, is time required to calculate all costs.

Our speedups come from *batching* cost calculations.

New Results (II)

Problem	Previous Best Result	This paper
Mixed Radix Coding	$O(n^4 \log n)$ [1]	$O(n^3)$
Reserved Length Coding (i)	$O(gn^3)$ [2]	$O(gn^2)$
Reserved Length Coding (ii)	$O(g^3 n^3 \log^g n)$ [2]	$O(gn^2 \log n)$
One-ended Coding	$O(n^3)$ [3]	$O(n^2)$
The Sound of Silence	$O(n^{U+2})$ [4]	$O(n^{U+1})$

Previous results were all based on Dynamic Programming.
DP creates a search space and calculates *optimal* cost
for every item in the search space.
Optimal cost of larger items is based on optimal cost of smaller items.
Running time of DP algorithm, is time required to calculate all costs.

Our speedups come from *batching* cost calculations.

Instead of calculating optimal-cost of each item individually,

New Results (II)

Problem	Previous Best Result	This paper
Mixed Radix Coding	$O(n^4 \log n)$ [1]	$O(n^3)$
Reserved Length Coding (i)	$O(gn^3)$ [2]	$O(gn^2)$
Reserved Length Coding (ii)	$O(g^3 n^3 \log^g n)$ [2]	$O(gn^2 \log n)$
One-ended Coding	$O(n^3)$ [3]	$O(n^2)$
The Sound of Silence	$O(n^{U+2})$ [4]	$O(n^{U+1})$

Previous results were all based on Dynamic Programming.
DP creates a search space and calculates *optimal* cost
for every item in the search space.
Optimal cost of larger items is based on optimal cost of smaller items.
Running time of DP algorithm, is time required to calculate all costs.

Our speedups come from *batching* cost calculations.

Instead of calculating optimal-cost of each item individually,
we group sets of items together and calculate *all* of their
optimal-costs together at the same time.

New Results (II)

Problem	Previous Best Result	This paper
Mixed Radix Coding	$O(n^4 \log n)$ [1]	$O(n^3)$
Reserved Length Coding (i)	$O(gn^3)$ [2]	$O(gn^2)$
Reserved Length Coding (ii)	$O(g^3 n^3 \log^g n)$ [2]	$O(gn^2 \log n)$
One-ended Coding	$O(n^3)$ [3]	$O(n^2)$
The Sound of Silence	$O(n^{U+2})$ [4]	$O(n^{U+1})$

Previous results were all based on Dynamic Programming.
DP creates a search space and calculates *optimal* cost
for every item in the search space.
Optimal cost of larger items is based on optimal cost of smaller items.
Running time of DP algorithm, is time required to calculate all costs.

Our speedups come from *batching* cost calculations.

Instead of calculating optimal-cost of each item individually,
we group sets of items together and calculate *all* of their
optimal-costs together at the same time.

This leads to lower amortized time per optimal-cost calculation.

- Introduction
  - A Quick Review of Prefix-Free Coding
  - New Results

- The Basic Top-Down Dynamic Programming Technique

- The Speedup

- Conclusion & Comments

The Technique

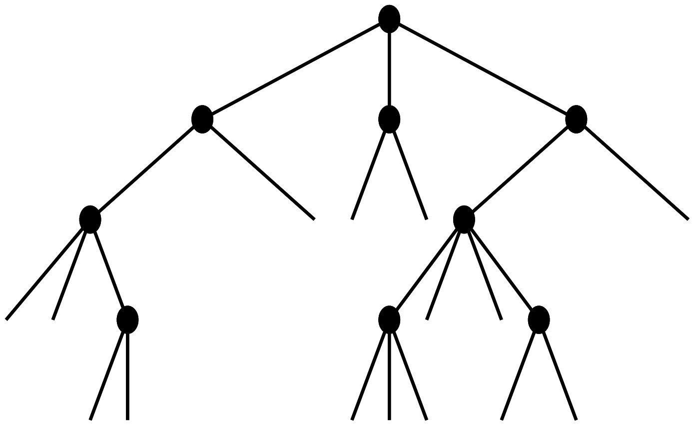
The Technique

We illustrate the technique by showing how to speed up *mixed-radix* coding from $O(n^4)$ down to $O(n^3)$. The same technique, with various bells and whistles added, speeds up all of the other problems.

The Technique

We illustrate the technique by showing how to speed up *mixed-radix* coding from $O(n^4)$ down to $O(n^3)$. The same technique, with various bells and whistles added, speeds up all of the other problems.

In mixed -radix coding, input is weight set $P = \{p_1, \ldots, p_n\}$ and *arity* list $R = \{r_1, r_2, r_3 \ldots\}$.

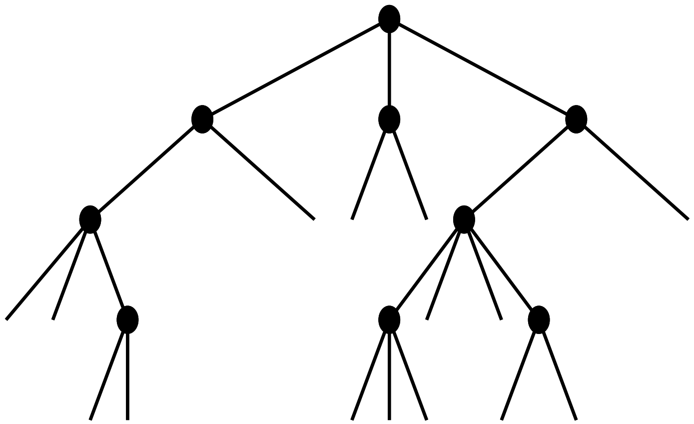


The Technique

We illustrate the technique by showing how to speed up *mixed-radix* coding from $O(n^4)$ down to $O(n^3)$. The same technique, with various bells and whistles added, speeds up all of the other problems.

In mixed -radix coding, input is weight set $P = \{p_1, \ldots, p_n\}$ and *arity* list $R = \{r_1, r_2, r_3 \ldots\}$.

Nodes on level $i-1$, have arity $\leq r_i$.



$r_1 = 3$
$r_2 = 2$
$r_3 = 4$
$r_4 = 3$

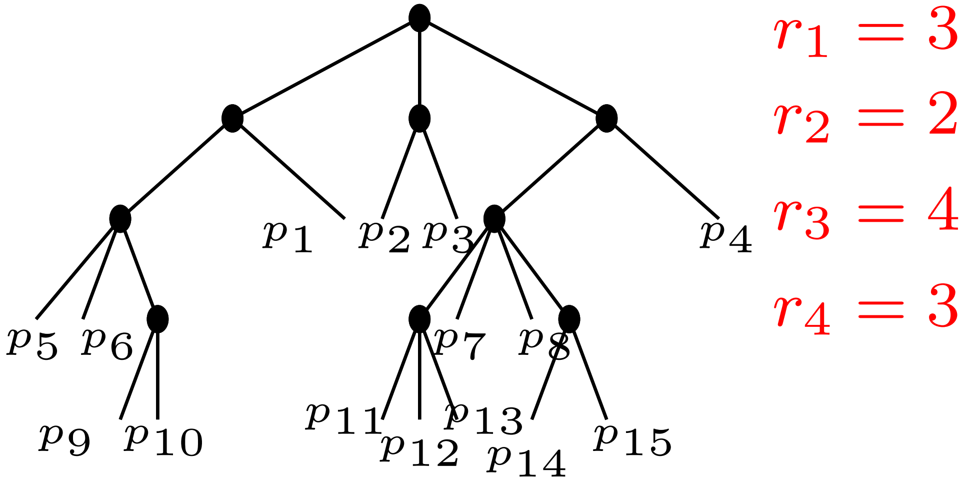
The Technique

We illustrate the technique by showing how to speed up *mixed-radix* coding from $O(n^4)$ down to $O(n^3)$. The same technique, with various bells and whistles added, speeds up all of the other problems.

In mixed -radix coding, input is weight set $P = \{p_1, \ldots, p_n\}$ and *arity* list $R = \{r_1, r_2, r_3 \ldots\}$.

Nodes on level $i-1$, have arity $\leq r_i$.

Want to find tree satisfying R with minimum cost $\sum_{i=1}^{n} p_i d(v_i)$.



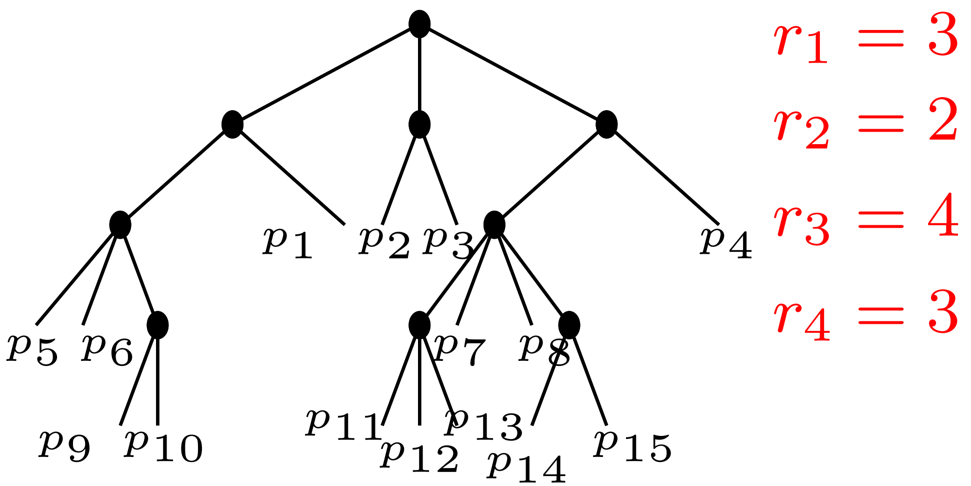
The Technique

We illustrate the technique by showing how to speed up *mixed-radix* coding from $O(n^4)$ down to $O(n^3)$. The same technique, with various bells and whistles added, speeds up all of the other problems.

In mixed -radix coding, input is weight set $P = \{p_1, \ldots, p_n\}$ and *arity* list $R = \{r_1, r_2, r_3 \ldots\}$.

Nodes on level $i-1$, have arity $\leq r_i$.

Want to find tree satisfying R with minimum cost $\sum_{i=1}^{n} p_i d(v_i)$.

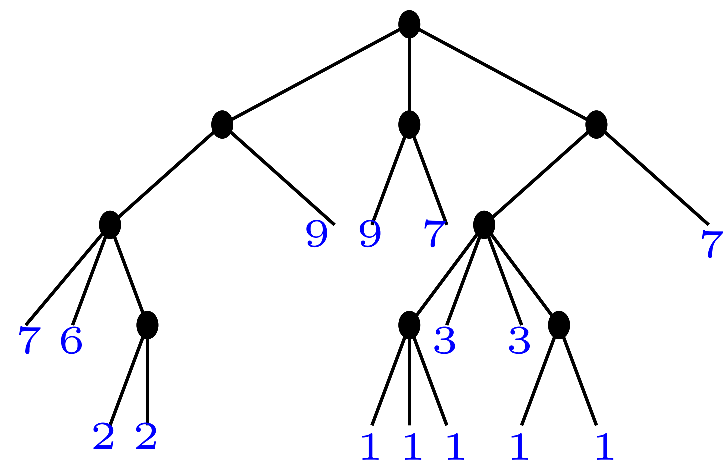


W.L.O.G. assume that the p_i are sorted in non-increasing order

The Technique

We illustrate the technique by showing how to speed up *mixed-radix* coding from $O(n^4)$ down to $O(n^3)$. The same technique, with various bells and whistles added, speeds up all of the other problems.

In mixed -radix coding, input is weight set $P = \{p_1, \ldots, p_n\}$ and *arity* list $R = \{r_1, r_2, r_3 \ldots\}$.



$r_1 = 3$

$r_2 = 2$

$r_3 = 4$

$r_4 = 3$

Nodes on level $i-1$, have arity $\leq r_i$.

Want to find tree satisfying R with minimum cost $\sum_{i=1}^{n} p_i d(v_i)$.

W.L.O.G. assume that the p_i are sorted in non-increasing order

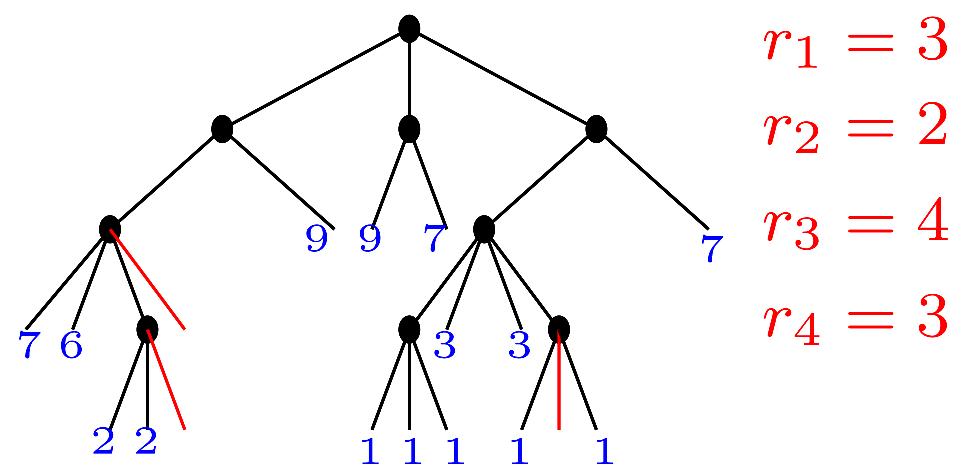
The Technique

We illustrate the technique by showing how to speed up *mixed-radix* coding from $O(n^4)$ down to $O(n^3)$. The same technique, with various bells and whistles added, speeds up all of the other problems.

In mixed -radix coding, input is weight set $P = \{p_1, \ldots, p_n\}$ and *arity* list $R = \{r_1, r_2, r_3 \ldots\}$.

Nodes on level $i-1$, have arity $\leq r_i$.

Want to find tree satisfying R with minimum cost $\sum_{i=1}^{n} p_i d(v_i)$.



W.L.O.G. assume that the p_i are sorted in non-increasing order

W.L.O.G. also assume that all internal nodes have exactly r_i children.

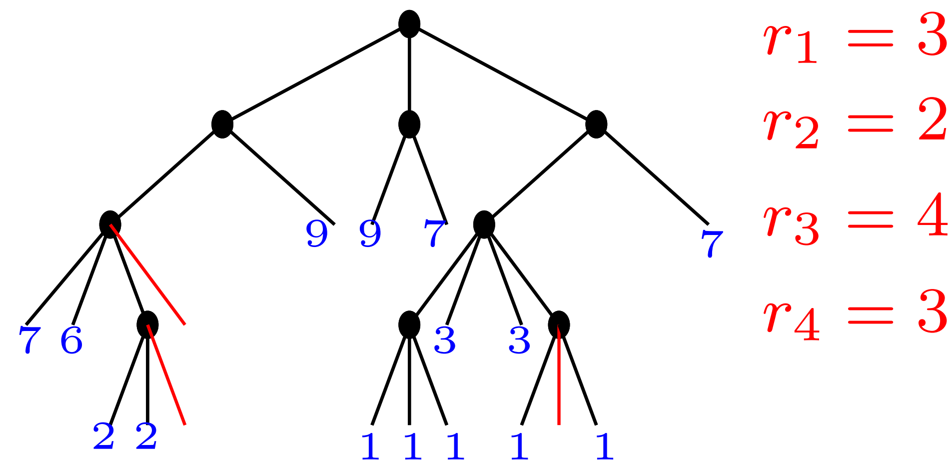
The Technique

We illustrate the technique by showing how to speed up *mixed-radix* coding from $O(n^4)$ down to $O(n^3)$. The same technique, with various bells and whistles added, speeds up all of the other problems.

In mixed -radix coding, input is weight set $P = \{p_1, \ldots, p_n\}$ and *arity* list $R = \{r_1, r_2, r_3 \ldots\}$.

Nodes on level $i-1$, have arity $\le r_i$.

Want to find tree satisfying R with minimum cost $\sum_{i=1}^{n} p_i d(v_i)$.



W.L.O.G. assume that the p_i are sorted in non-increasing order

W.L.O.G. also assume that all internal nodes have exactly r_i children.

Can ensure this by padding P with arbitrarily many 0s.

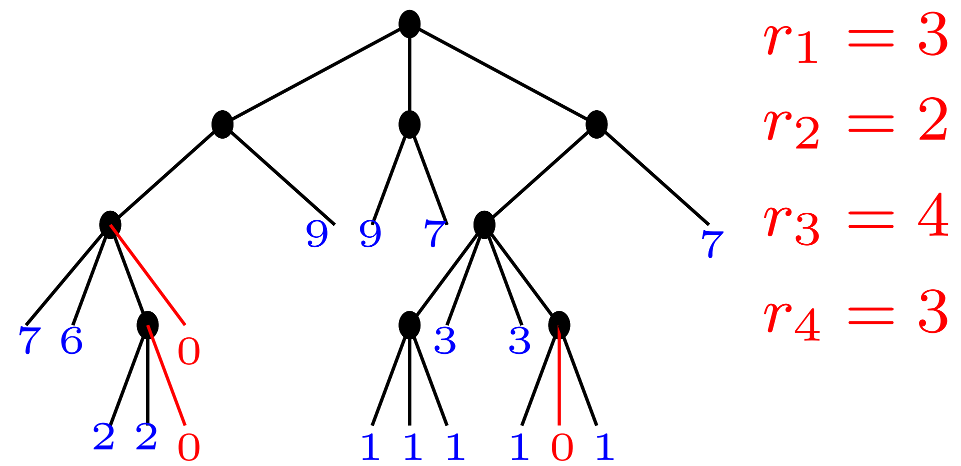
The Technique

We illustrate the technique by showing how to speed up *mixed-radix* coding from $O(n^4)$ down to $O(n^3)$. The same technique, with various bells and whistles added, speeds up all of the other problems.

In mixed -radix coding, input is weight set $P = \{p_1, \ldots, p_n\}$ and *arity* list $R = \{r_1, r_2, r_3 \ldots\}$.

Nodes on level $i-1$, have arity $\leq r_i$.

Want to find tree satisfying R with minimum cost $\sum_{i=1}^{n} p_i d(v_i)$.



W.L.O.G. assume that the p_i are sorted in non-increasing order

W.L.O.G. also assume that all internal nodes have exactly r_i children.

Can ensure this by padding P with arbitrarily many 0s.

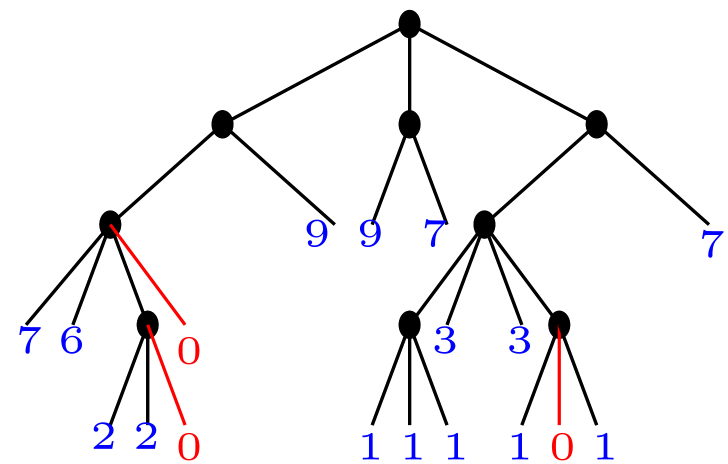
The Technique

We illustrate the technique by showing how to speed up *mixed-radix* coding from $O(n^4)$ down to $O(n^3)$. The same technique, with various bells and whistles added, speeds up all of the other problems.

In mixed -radix coding, input is weight set $P = \{p_1, \ldots, p_n\}$ and *arity* list $R = \{r_1, r_2, r_3 \ldots\}$.

Nodes on level $i-1$, have arity $\leq r_i$.

Want to find tree satisfying R with minimum cost $\sum_{i=1}^{n} p_i d(v_i)$.



$r_1 = 3$

$r_2 = 2$

$r_3 = 4$

$r_4 = 3$

W.L.O.G. assume that the p_i are sorted in non-increasing order

W.L.O.G. also assume that all internal nodes have exactly r_i children.

Can ensure this by padding P with arbitrarily many 0s.

(might require moving some p_i's up the tree)

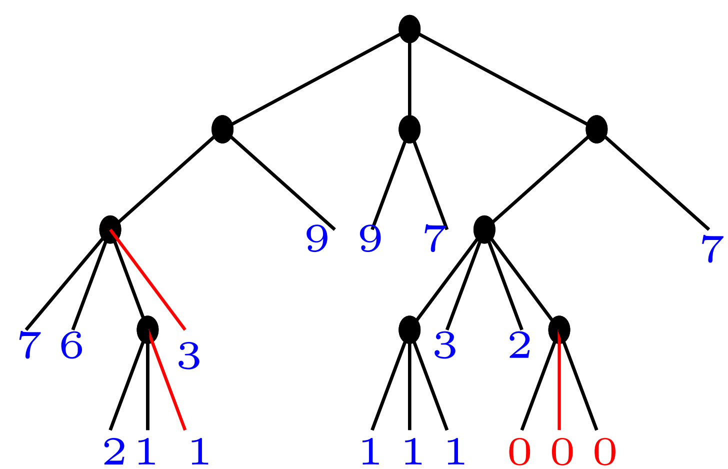
The Technique

We illustrate the technique by showing how to speed up *mixed-radix* coding from $O(n^4)$ down to $O(n^3)$. The same technique, with various bells and whistles added, speeds up all of the other problems.

In mixed -radix coding, input is weight set $P = \{p_1, \ldots, p_n\}$ and *arity* list $R = \{r_1, r_2, r_3 \ldots\}$.

Nodes on level $i-1$, have arity $\leq r_i$.

Want to find tree satisfying R with minimum cost $\sum_{i=1}^{n} p_i d(v_i)$.



$r_1 = 3$
$r_2 = 2$
$r_3 = 4$
$r_4 = 3$

W.L.O.G. assume that the p_i are sorted in non-increasing order

W.L.O.G. also assume that all internal nodes have exactly r_i children.

Can ensure this by padding P with arbitrarily many 0s.

(might require moving some p_i's up the tree)

The Technique

The Technique

Build the tree *top-down*, level-by-level, using DP.

Standard technique: e.g., Golin & Rote '98, Dolev, Korach & Yukelson '99, Chan & Golin '00, Baer '08

The Technique

Build the tree *top-down*, level-by-level, using DP.

Standard technique: e.g., Golin & Rote '98, Dolev, Korach & Yukelson '99, Chan & Golin '00, Baer '08

Idea is to keep track, at depth i, of

m : # of leaves so far

b: # of " internal" depth i nodes.
These are nodes that will be
"expanded" at next step

The Technique

Build the tree *top-down*, level-by-level, using DP.

Standard technique: e.g., Golin & Rote '98, Dolev, Korach & Yukelson '99, Chan & Golin '00, Baer '08

Idea is to keep track, at depth i, of

m : # of leaves so far

b: # of " internal" depth i nodes.
These are nodes that will be "expanded" at next step

	d	m	b
•	0	0	1

The Technique

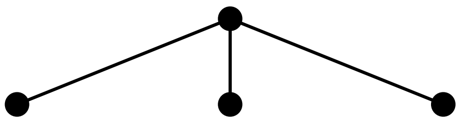
Build the tree *top-down*, level-by-level, using DP.

Standard technique: e.g., Golin & Rote '98, Dolev, Korach & Yukelson '99, Chan & Golin '00, Baer '08

Idea is to keep track, at depth i, of

m : # of leaves so far

b: # of " internal" depth i nodes.
These are nodes that will be
"expanded" at next step



d	m	b
0	0	1
1	0	3

The Technique

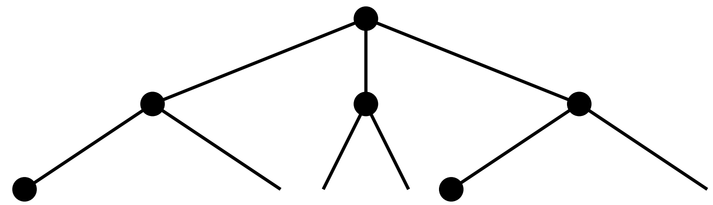
Build the tree *top-down*, level-by-level, using DP.

Standard technique: e.g., Golin & Rote '98, Dolev, Korach & Yukelson '99, Chan & Golin '00, Baer '08

Idea is to keep track, at depth i, of

m : # of leaves so far

b: # of " internal" depth i nodes.
These are nodes that will be "expanded" at next step



d	m	b
0	0	1
1	0	3
2	4	2

The Technique

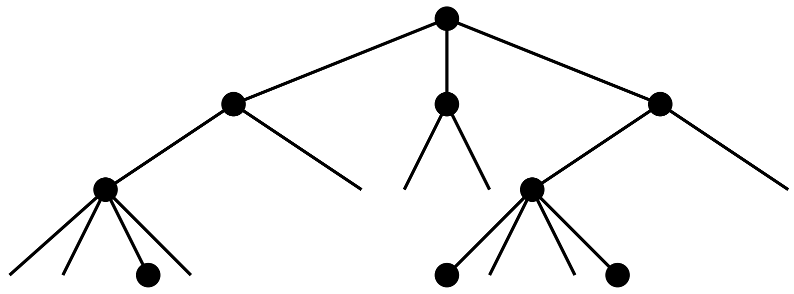
Build the tree *top-down*, level-by-level, using DP.

Standard technique: e.g., Golin & Rote '98, Dolev, Korach & Yukelson '99, Chan & Golin '00, Baer '08

Idea is to keep track, at depth i, of

m : # of leaves so far

b: # of " internal" depth i nodes.
These are nodes that will be "expanded" at next step



d	m	b
0	0	1
1	0	3
2	4	2
3	9	3

The Technique

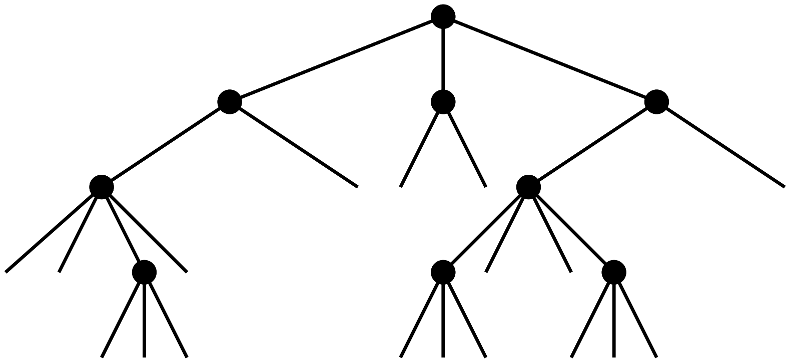
Build the tree *top-down*, level-by-level, using DP.

Standard technique: e.g., Golin & Rote '98, Dolev, Korach & Yukelson '99, Chan & Golin '00, Baer '08

Idea is to keep track, at depth i, of

m : # of leaves so far

b: # of " internal" depth i nodes.
These are nodes that will be "expanded" at next step



d	m	b
0	0	1
1	0	3
2	4	2
3	9	3
4	18	0

The Technique

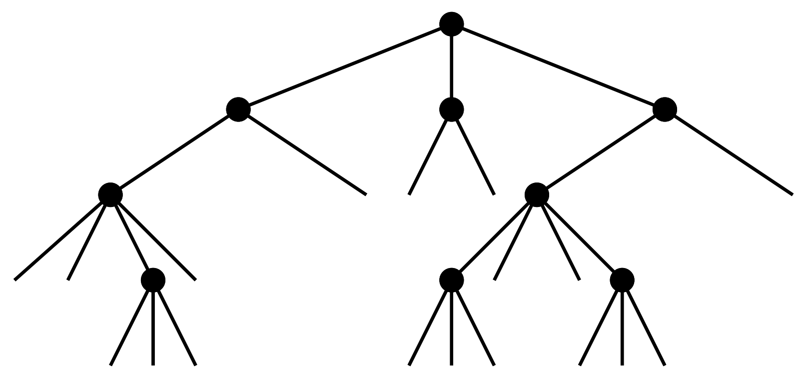
Build the tree *top-down*, level-by-level, using DP.

Standard technique: e.g., Golin & Rote '98, Dolev, Korach & Yukelson '99, Chan & Golin '00, Baer '08

Idea is to keep track, at depth i, of

m : # of leaves so far

b: # of " internal" depth i nodes.
These are nodes that will be "expanded" at next step



d	m	b
0	0	1
1	0	3
2	4	2
3	9	3
4	18	0

Will also keep track of cost "so far".

$\sum_{t=1}^{m} p_i d_i + i \sum_{t>m} p_t$

The Technique

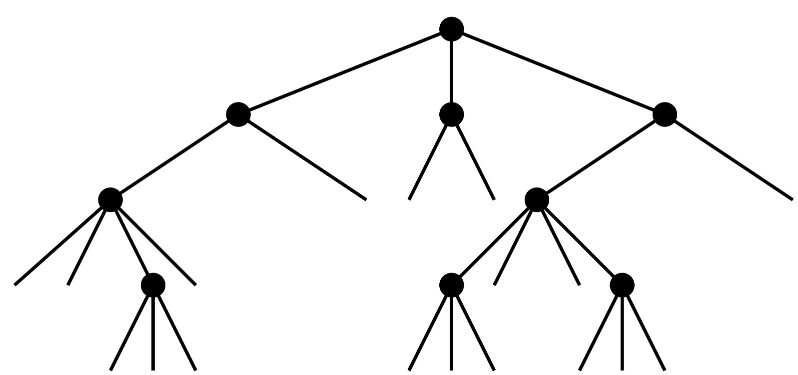
Build the tree *top-down*, level-by-level, using DP.

Standard technique: e.g., Golin & Rote '98, Dolev, Korach & Yukelson '99, Chan & Golin '00, Baer '08

Idea is to keep track, at depth i, of

m : # of leaves so far

b: # of " internal" depth i nodes.
These are nodes that will be
"expanded" at next step



d	m	b
0	0	1
1	0	3
2	4	2
3	9	3
4	18	0

Will also keep track of cost "so far".

$$\sum_{t=1}^{m} p_i d_i + i \sum_{t>m} p_t$$

Ex: $P = \{3,3,3,3,3,\ 2,2,2,2,2,\ 1,1,1,1,1,\ 0,0,\ldots\}$

The Technique

Build the tree *top-down*, level-by-level, using DP.

Standard technique: e.g., Golin & Rote '98, Dolev, Korach & Yukelson '99, Chan & Golin '00, Baer '08

Idea is to keep track, at depth i, of

m : # of leaves so far

b: # of " internal" depth i nodes.
These are nodes that will be "expanded" at next step

d	m	b	c
0	0	1	0
1	0	3	
2	4	2	
3	9	3	
4	18	0	

Will also keep track of cost "so far".

$$\sum_{t=1}^{m} p_i d_i + i \sum_{t>m} p_t$$

Ex: $P = \{3, 3, 3, 3, 3,\ 2, 2, 2, 2, 2,\ 1, 1, 1, 1, 1,\ 0, 0, \ldots\}$

The Technique

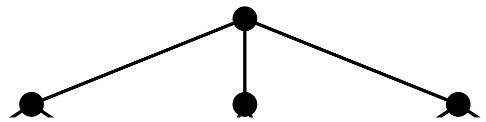
Build the tree *top-down*, level-by-level, using DP.

Standard technique: e.g., Golin & Rote '98, Dolev, Korach & Yukelson '99, Chan & Golin '00, Baer '08

Idea is to keep track, at depth i, of

m : # of leaves so far

b: # of " internal" depth i nodes.
These are nodes that will be "expanded" at next step



d	m	b	c
0	0	1	0
1	0	3	30
2	4	2	
3	9	3	
4	18	0	

Will also keep track of cost "so far".

$$\sum_{t=1}^{m} p_i d_i + i \sum_{t>m} p_t$$

Ex: $P = \{\underbrace{3,3,3,3,3,\ 2,2,2,2,2,\ 1,1,1,1,1,\ 0,0,\ldots}_{X1}\}$

The Technique

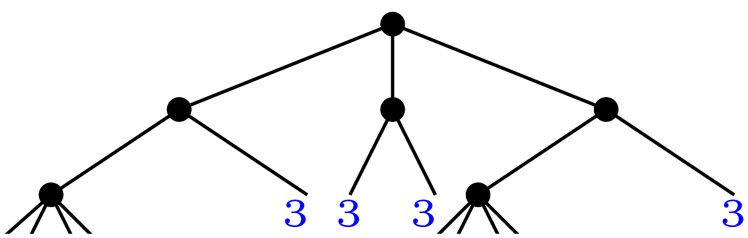
Build the tree *top-down*, level-by-level, using DP.

Standard technique: e.g., Golin & Rote '98, Dolev, Korach & Yukelson '99, Chan & Golin '00, Baer '08

Idea is to keep track, at depth i, of

m : # of leaves so far

b: # of " internal" depth i nodes.
These are nodes that will be "expanded" at next step



d	m	b	c
0	0	1	0
1	0	3	30
2	4	2	60
3	9	3	
4	18	0	

Will also keep track of cost "so far".

$$\sum_{t=1}^{m} p_i d_i + i \sum_{t>m} p_t$$

Ex: $P = \{3, 3, 3, 3, \underbrace{3,\ 2, 2, 2, 2, 2,\ 1, 1, 1, 1, 1,\ 0, 0, \ldots}_{X2}\}$

The Technique

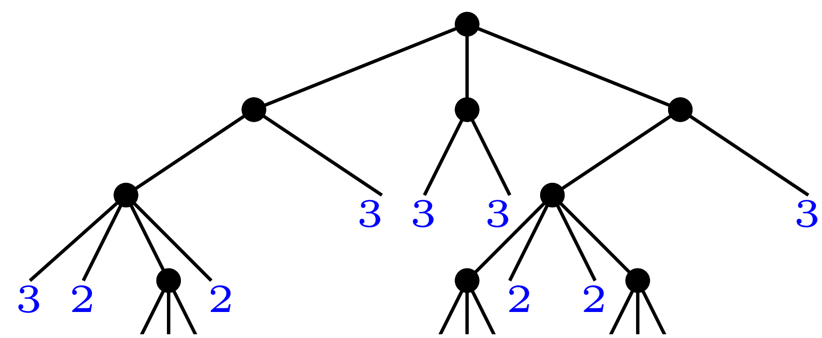
Build the tree *top-down*, level-by-level, using DP.

Standard technique: e.g., Golin & Rote '98, Dolev, Korach & Yukelson '99, Chan & Golin '00, Baer '08

Idea is to keep track, at depth i, of

m : # of leaves so far

b: # of " internal" depth i nodes.
These are nodes that will be "expanded" at next step



d	m	b	c
0	0	1	0
1	0	3	30
2	4	2	60
3	9	3	78
4	18	0	

Will also keep track of cost "so far".

$$\sum_{t=1}^{m} p_i d_i + i \sum_{t>m} p_t$$

Ex: $P = \{3,3,3,3,3,\ 2,2,2,2,\underbrace{2,\ 1,1,1,1,1,\ 0,0,\ldots}_{X3}.\}$

The Technique

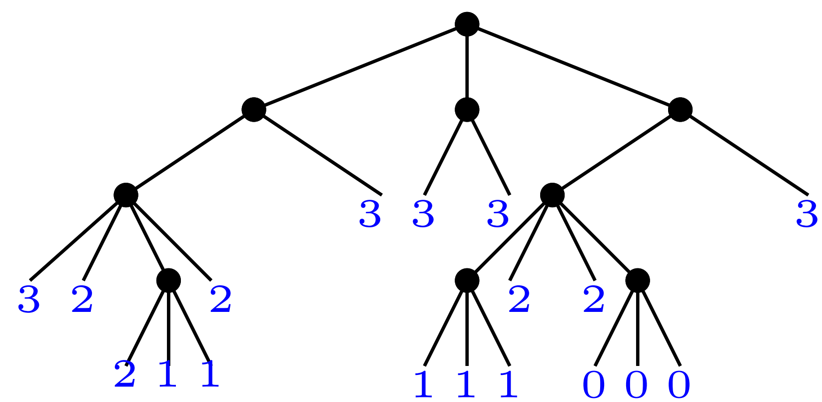
Build the tree *top-down*, level-by-level, using DP.

Standard technique: e.g., Golin & Rote '98, Dolev, Korach & Yukelson '99, Chan & Golin '00, Baer '08

Idea is to keep track, at depth i, of

m : # of leaves so far

b: # of " internal" depth i nodes.
These are nodes that will be "expanded" at next step



d	m	b	c
0	0	1	0
1	0	3	30
2	4	2	60
3	9	3	78
4	18	0	84

Will also keep track of cost "so far".

$\sum_{t=1}^{m} p_i d_i + i \sum_{t>m} p_t$

Ex: $P = \{3,3,3,3,3,\ 2,2,2,2,2,\ 1,1,1,1,1,\ 0,0,\ldots\}$

The Technique

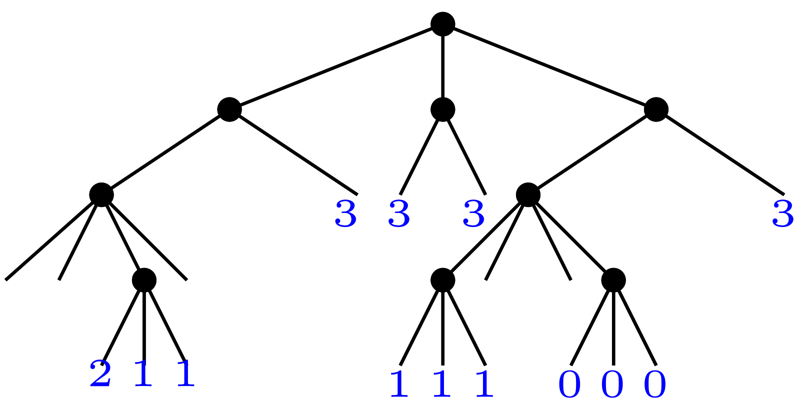
Build the tree *top-down*, level-by-level, using DP.

Standard technique: e.g., Golin & Rote '98, Dolev, Korach & Yukelson '99, Chan & Golin '00, Baer '08

Idea is to keep track, at depth i, of

m : # of leaves so far

b: # of " internal" depth i nodes.
These are nodes that will be "expanded" at next step



d	m	b	c
0	0	1	0
1	0	3	30
2	4	2	60
3	9	3	
4	18	0	84

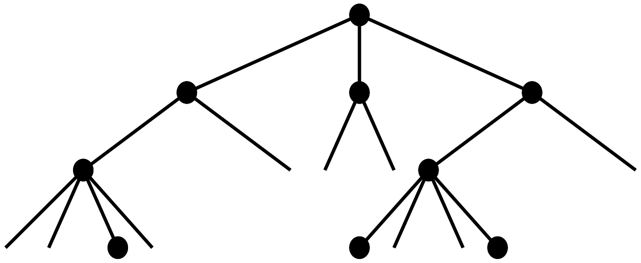
Will also keep track of cost "so far".

$$\sum_{t=1}^{m} p_i d_i + i \sum_{t>m} p_t$$

If $m \geq n$, then "cost so far" is real cost of tree

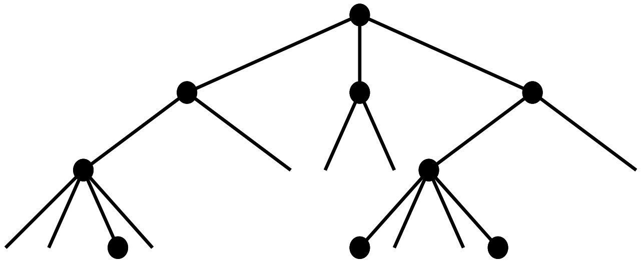
The Technique

T is an i-level tree if $d(T) \le i$.



The Technique

T is an i-level tree if $d(T) \leq i$.



$$sig_i(T) = (m, b)$$

$m = \#$ leaves at depth $\leq i$.

$b = \#$ internals at depth i.

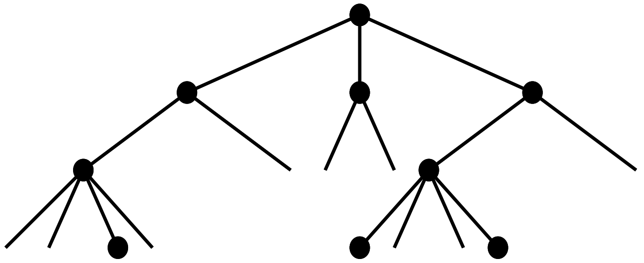
The Technique

T is an i-level tree if $d(T) \leq i$.

$$sig_i(T) = (m, b)$$

$m = \#$ leaves at depth $\leq i$.

$b = \#$ internals at depth i.



$sig_2(T) = (9, 3)$

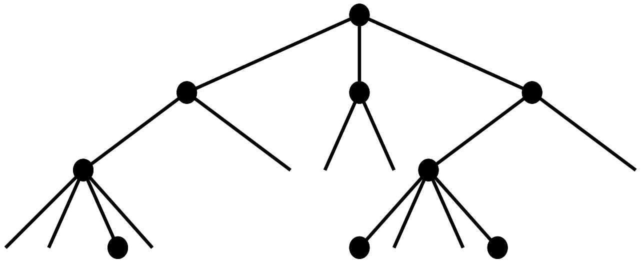
The Technique

T is an i-level tree if $d(T) \leq i$.

$$sig_i(T) = (m, b)$$

$m = \#$ leaves at depth $\leq i$.

$b = \#$ internals at depth i.



$sig_2(T) = (9, 3)$

$$OPT^i[m, b] = \min \left[cost_i(T) \mid sig_i(T) = (m, b) \right].$$

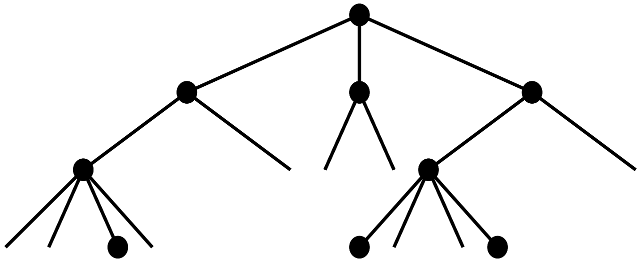
The Technique

T is an i-level tree if $d(T) \leq i$.

$$sig_i(T) = (m, b)$$

$m = \#$ leaves at depth $\leq i$.

$b = \#$ internals at depth i.



$$sig_2(T) = (9, 3)$$

$$OPT^i[m, b] = \min\left[cost_i(T) \mid sig_i(T) = (m, b)\right].$$

$$\min_{m \geq n}\left(OPT^i(m, 0)\right)$$

is cost of min-cost tree with at least n leaves and depth $\leq i$.

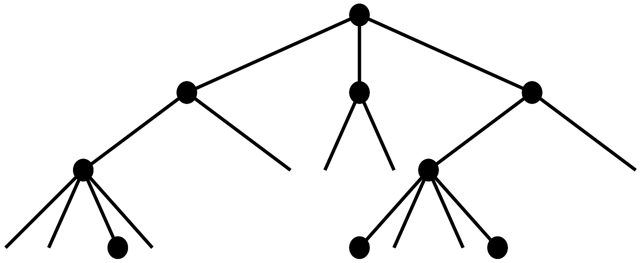
The Technique

T is an i-level tree if $d(T) \leq i$.

$$sig_i(T) = (m, b)$$

$m = \#$ leaves at depth $\leq i$.

$b = \#$ internals at depth i.



$sig_2(T) = (9, 3)$

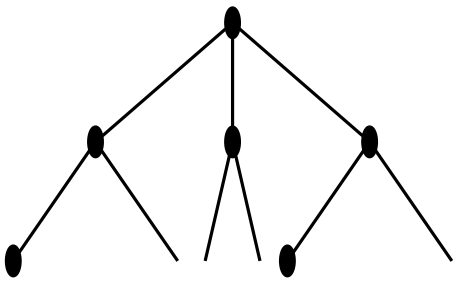
$$OPT^i[m, b] = \min\left[cost_i(T) \mid sig_i(T) = (m, b)\right].$$

$$\min_{m \geq n}\left(OPT^i(m, 0)\right)$$

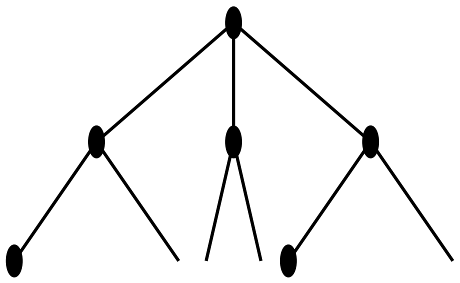
is cost of min-cost tree with at least n leaves and depth $\leq i$.

Goal: Find $\min_{m \geq n}\left(OPT^n(m, 0)\right)$ and tree that achieves it

Let T' be an $(i-1)$-level tree with $sig_{i-1}(T) = (m', b')$.

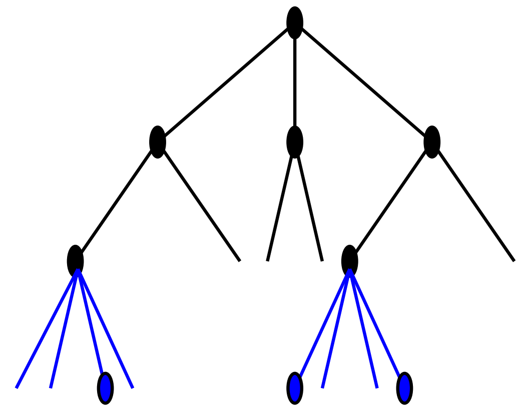


Let T' be an $(i-1)$-level tree with $sig_{i-1}(T) = (m', b')$.



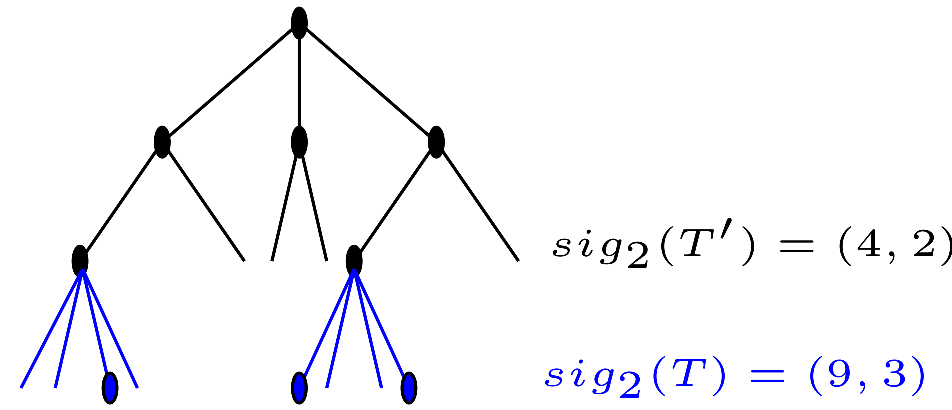
T' is *expanded* to an i level tree T by adding the $r_i b'$ children on level i and choosing b of them to be internal.

Let T' be an $(i-1)$-level tree with $sig_{i-1}(T) = (m', b')$.



T' is *expanded* to an i level tree T by adding the $r_i b'$ children on level i and choosing b of them to be internal.

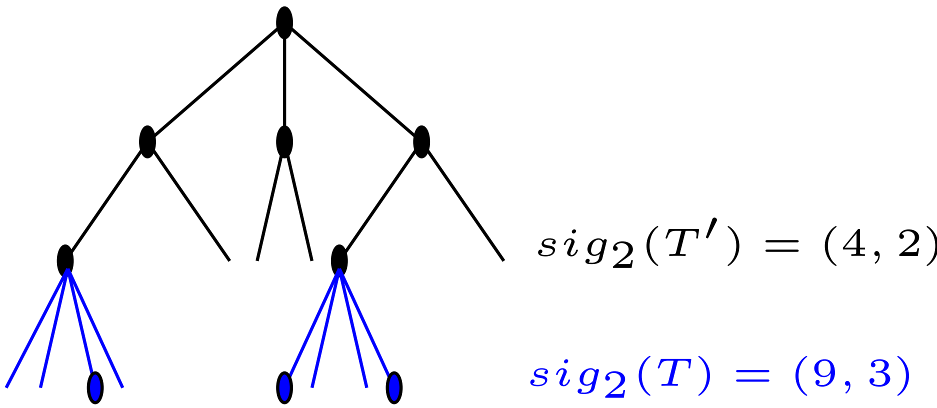
Let T' be an $(i-1)$-level tree with $sig_{i-1}(T) = (m', b')$.



T' is *expanded* to an i level tree T by adding the $r_i b'$ children on level i and choosing b of them to be internal.

Lemma: $m = m' + b'r_i - b$ and $cost_i(T) = cost_{i-1}(T') + \sum_{t>m'} p_t$.

Let T' be an $(i-1)$-level tree with $sig_{i-1}(T) = (m', b')$.

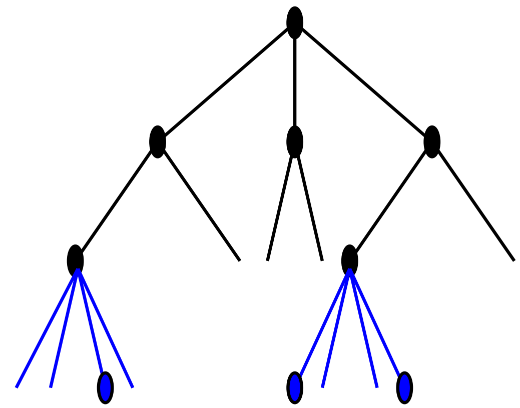


T' is *expanded* to an i level tree T by adding the $r_i b'$ children on level i and choosing b of them to be internal.

Lemma: $m = m' + b' r_i - b$ and $cost_i(T) = cost_{i-1}(T') + \sum_{t > m'} p_t$.

We say that $(m', b') \xrightarrow{i} (m, b)$ if $\exists T', T$ as above.

Let T' be an $(i-1)$-level tree with $sig_{i-1}(T) = (m', b')$.



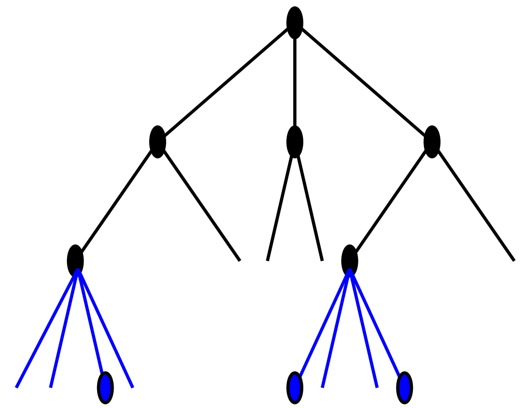
T' is *expanded* to an i level tree T by adding the $r_i b'$ children on level i and choosing b of them to be internal.

Lemma: $m = m' + b' r_i - b$ and $cost_i(T) = cost_{i-1}(T') + \sum_{t > m'} p_t$. $W_{m'}$

We say that $(m', b') \xrightarrow{i} (m, b)$ if $\exists T', T$ as above.

Let T' be an $(i-1)$-level tree with $sig_{i-1}(T) = (m', b')$.

T' is *expanded* to an i level tree T by adding the $r_i b'$ children on level i and choosing b of them to be internal.



$sig_2(T') = (4, 2)$

$sig_2(T) = (9, 3)$

Lemma: $m = m' + b' r_i - b$ and $cost_i(T) = cost_{i-1}(T') + \sum_{t > m'} p_t$.
$W_{m'}$

We say that $(m', b') \xrightarrow{i} (m, b)$ if $\exists T', T$ as above.

The DP recurrence is thus

$$OPT^i[m, b] = \min_{\{(m',b') \mid (m',b') \xrightarrow{i} (m,b)\}} \left\{ OPT^{i-1}[m', b'] + W_{m'} \right\}.$$

with initial condition $OPT^0[0, 1] = 0.$

- Introduction
  - A Quick Review of Prefix-Free Coding
  - New Results

- The Basic Top-Down Dynamic Programming Technique

- The Speedup

- Conclusion & Comments

$$OPT^i[m,b] = \min_{\{(m',b') \mid (m',b') \xrightarrow{i} (m,b)\}} \left\{ OPT^{i-1}[m',b'] + W_{m'} \right\}.$$

where $m = m' + b'r_i - b$ and $b \leq b'r_i$.

$$OPT^i[m,b] = \min_{\{(m',b') \mid (m',b') \xrightarrow{i} (m,b)\}} \left\{ OPT^{i-1}[m',b'] + W_{m'} \right\}.$$

where $m = m' + b'r_i - b$ and $b \le b'r_i$.

So, only need to check $O(m)$ entries to calculate given $OPT^i[m,b]$.

$$OPT^i[m,b] = \min_{\{(m',b') \mid (m',b') \xrightarrow{i} (m,b)\}} \left\{ OPT^{i-1}[m',b'] + W_{m'} \right\}.$$

where $m = m' + b'r_i - b$ and $b \le b'r_i$.

So, only need to check $O(m)$ entries to calculate given $OPT^i[m,b]$.

Not hard to prove that

if $b > 0$ then $m + b \le n$ and

if $b = 0$ then $m < n + r_i$.

So, only need to fill in $O\left(n^2\right)$ entries.

Note: paper shows how to make $O(n^2)$ independent of r_i

$\Rightarrow$ Total time to fill in $OPT^i[\,,\,]$ table is $O(n^3)$.

$$OPT^i[m,b] = \min_{\{(m',b') \mid (m',b') \xrightarrow{i} (m,b)\}} \left\{ OPT^{i-1}[m',b'] + W_{m'} \right\}.$$

where $m = m' + b'r_i - b$ and $b \leq b'r_i$.

So, only need to check $O(m)$ entries to calculate given $OPT^i[m,b]$.

Not hard to prove that

if $b > 0$ then $m + b \leq n$ and

if $b = 0$ then $m < n + r_i$.

So, only need to fill in $O\left(n^2\right)$ entries.

Note: paper shows how to make $O(n^2)$ independent of r_i

$\Rightarrow$ Total time to fill in $OPT^i[\,,\,]$ table is $O(n^3)$.

We now (finally) see how to reduce this down to $O(n^2)$.

$$OPT^i[m,b] = \min_{\{(m',b') \mid (m',b') \xrightarrow{i} (m,b)\}} \left\{ OPT^{i-1}[m',b'] + W_{m'} \right\}.$$

where $m = m' + b'r_i - b$ and $b \le b'r_i$.

So, only need to check $O(m)$ entries to calculate given $OPT^i[m,b]$.

Not hard to prove that
if $b > 0$ then $m + b \le n$ and
if $b = 0$ then $m < n + r_i$.
So, only need to fill in $O\left(n^2\right)$ entries.

Note: paper shows how to make $O(n^2)$ *independent of* r_i

$\Rightarrow$ Total time to fill in $OPT^i[\,,\,]$ table is $O(n^3)$.

We now (finally) see how to reduce this down to $O(n^2)$.
Filling in *all* of the tables and solving the entire problem in $O(n^3)$ time.

$$OPT^i[m,b] = \min_{\{(m',b') \mid (m',b') \xrightarrow{i} (m,b)\}} \left\{ OPT^{i-1}[m',b'] + W_{m'} \right\}.$$

where $m = m' + b'r_i - b$ and $b \leq b'r_i$.

$$OPT^i[m,b] = \min_{\{(m',b') \mid (m',b') \xrightarrow{i} (m,b)\}} \left\{ \overset{X[m',b']}{OPT^{i-1}[m',b'] + W_{m'}} \right\}.$$

where $m = m' + b'r_i - b$ and $b \leq b'r_i$.

$$OPT^i[m,b] = \min_{\{(m',b') \mid (m',b') \xrightarrow{i} (m,b)\}} \left\{ \underbrace{OPT^{i-1}[m',b'] + W_{m'}}_{X[m',b']} \right\}.$$

where $m = m' + b'r_i - b$ and $b \leq b'r_i$.

For fixed $d \geq 1$ set

$$\mathcal{I}(d) = \{(m,b) \mid m+b=d\}, \quad \mathcal{I}'_i(d) = \left\{(m',b') \mid m' + b'r_i = d\right\}.$$

$$OPT^i[m,b] = \min_{\{(m',b') \mid (m',b') \xrightarrow{i} (m,b)\}} \left\{ \overset{X[m',b']}{\cancel{OPT^{i-1}[m',b'] + W_{m'}}} \right\}.$$

where $m = m' + b'r_i - b$ and $b \leq b'r_i$.

For fixed $d \geq 1$ set

$$\mathcal{I}(d) = \{(m,b) \mid m+b=d\}, \quad \mathcal{I}'_i(d) = \left\{(m',b') \mid m' + b'r_i = d\right\}.$$

Then, $\forall (m,b) \in \mathcal{I}(d)$,

$$\text{“}(m',b') \xrightarrow{i} (m,b)\text{”} \Leftrightarrow \text{“}(m',b') \in \mathcal{I}'_i(d) \text{ with } b \leq b'r_i\text{”}.$$

$$OPT^i[m,b] = \min_{\{(m',b') \mid (m',b') \xrightarrow{i} (m,b)\}} \left\{ \overset{X[m',b']}{\cancel{OPT^{i-1}[m',b'] + W_{m'}}} \right\}.$$

where $m = m' + b'r_i - b$ and $b \leq b'r_i$.

For fixed $d \geq 1$ set

$$\mathcal{I}(d) = \{(m,b) \mid m + b = d\}, \quad \mathcal{I}'_i(d) = \left\{(m',b') \mid m' + b'r_i = d\right\}.$$

Then, $\forall (m,b) \in \mathcal{I}(d)$,

$$\text{“}(m',b') \xrightarrow{i} (m,b)\text{”} \Leftrightarrow \text{“}(m',b') \in \mathcal{I}'_i(d) \text{ with } b \leq b'r_i\text{”}.$$

In particular

$$OPT^i[m,b] = \min\{X[m',b'] : (m',b') \in \mathcal{I}'_i(d), \;\; b/r_i \leq b'\}$$

$$\mathcal{I}(d) = \{(m, b) \mid m + b = d\}, \quad \mathcal{I}'_i(d) = \Big\{(m', b') \mid m' + b' r_i = d\Big\}.$$

$$OPT^i[m, b] = \min\{X[m', b'] : (m', b') \in \mathcal{I}'_i(d), \; b/r_i \le b'\}$$

$$\mathcal{I}(d) = \{(m, b) \mid m + b = d\}, \quad \mathcal{I}'_i(d) = \left\{(m', b') \mid m' + b' r_i = d\right\}.$$

$$OPT^i[m, b] = \min\{X[m', b'] : (m', b') \in \mathcal{I}'_i(d),\ \ b/r_i \leq b'\}$$

EX: $r_i = 3$, $d = 12$

m	b	(m', b') to minimize over

$$\mathcal{I}(d) = \{(m,b) \mid m+b=d\}, \quad \mathcal{I}'_i(d) = \left\{(m',b') \mid m'+b'r_i = d\right\}.$$

$$OPT^i[m,b] = \min\{X[m',b'] \,:\, (m',b') \in \mathcal{I}'_i(d),\;\; b/r_i \le b'\}$$

EX: $r_i = 3$, $d = 12$

m	b	(m', b') to minimize over
0	12	$(0,4)$
1	11	$(0,4)$
2	10	$(0,4)$

$$\mathcal{I}(d) = \{(m, b) \mid m + b = d\}, \quad \mathcal{I}_i'(d) = \left\{(m', b') \mid m' + b' r_i = d\right\}.$$

$$OPT^i[m, b] = \min\{X[m', b'] : (m', b') \in \mathcal{I}_i'(d), \;\; b/r_i \leq b'\}$$

EX: $r_i = 3$, $d = 12$

m	b	(m', b') to minimize over
0	12	$(0, 4)$
1	11	$(0, 4)$
2	10	$(0, 4)$
3	9	$(0, 4)$, $(3, 3)$
4	8	$(0, 4)$, $(3, 3)$
5	7	$(0, 4)$, $(3, 3)$

$$\mathcal{I}(d) = \{(m,b) \mid m+b=d\}, \quad \mathcal{I}_i'(d) = \left\{(m',b') \mid m'+b'r_i = d\right\}.$$

$$OPT^i[m,b] = \min\{X[m',b'] : (m',b') \in \mathcal{I}_i'(d), \;\; b/r_i \le b'\}$$

EX: $r_i = 3$, $d = 12$

m	b	(m', b') to minimize over
0	12	$(0,4)$
1	11	$(0,4)$
2	10	$(0,4)$
3	9	$(0,4),\ (3,3)$
4	8	$(0,4),\ (3,3)$
5	7	$(0,4),\ (3,3)$
6	6	$(0,4),\ (3,3),\ (6,2)$
7	5	$(0,4),\ (3,3),\ (6,2)$
8	4	$(0,4),\ (3,3),\ (6,2)$

$$\mathcal{I}(d) = \{(m,b) \mid m+b=d\}, \quad \mathcal{I}_i'(d) = \left\{(m',b') \mid m'+b'r_i = d\right\}.$$

$$OPT^i[m,b] = \min\{X[m',b'] : (m',b') \in \mathcal{I}_i'(d), \; b/r_i \le b'\}$$

EX: $r_i = 3$, $d = 12$

m	b	(m',b') to minimize over
0	12	$(0,4)$
1	11	$(0,4)$
2	10	$(0,4)$
3	9	$(0,4)$, $(3,3)$
4	8	$(0,4)$, $(3,3)$
5	7	$(0,4)$, $(3,3)$
6	6	$(0,4)$, $(3,3)$, $(6,2)$
7	5	$(0,4)$, $(3,3)$, $(6,2)$
8	4	$(0,4)$, $(3,3)$, $(6,2)$
9	3	$(0,4)$, $(3,3)$, $(6,2)$, $(9,1)$
10	2	$(0,4)$, $(3,3)$, $(6,2)$, $(9,1)$
11	1	$(0,4)$, $(3,3)$, $(6,2)$, $(9,1)$

$$\mathcal{I}(d) = \{(m,b) \mid m+b=d\}, \quad \mathcal{I}'_i(d) = \left\{(m',b') \mid m' + b'r_i = d\right\}.$$

$$OPT^i[m,b] = \min\{X[m',b'] : (m',b') \in \mathcal{I}'_i(d),\ \ b/r_i \le b'\}$$

EX: $r_i = 3$, $d = 12$

m	b	(m',b') to minimize over
0	12	$(0,4)$
1	11	$(0,4)$
2	10	$(0,4)$
3	9	$(0,4)$, $(3,3)$
4	8	$(0,4)$, $(3,3)$
5	7	$(0,4)$, $(3,3)$
6	6	$(0,4)$, $(3,3)$, $(6,2)$
7	5	$(0,4)$, $(3,3)$, $(6,2)$
8	4	$(0,4)$, $(3,3)$, $(6,2)$
9	3	$(0,4)$, $(3,3)$, $(6,2)$, $(9,1)$
10	2	$(0,4)$, $(3,3)$, $(6,2)$, $(9,1)$
11	1	$(0,4)$, $(3,3)$, $(6,2)$, $(9,1)$
12	0	$(0,4)$, $(3,3)$, $(6,2)$, $(9,1)$, $(12,0)$

For fixed d,
time needed to calculate
all $OPT^i[m,b]$
with $(m,b) \in \mathcal{I}(d)$ is
$O(|\mathcal{I}(d)| + |\mathcal{I}'d|) = O(d)$

$$\mathcal{I}(d) = \{(m,b) \mid m+b=d\}, \quad \mathcal{I}'_i(d) = \left\{(m',b') \mid m'+b'r_i = d\right\}.$$

$$OPT^i[m,b] = \min\{X[m',b'] : (m',b') \in \mathcal{I}'_i(d),\ \ b/r_i \le b'\}$$

EX: $r_i = 3$, $d = 12$

m	b	(m',b') to minimize over
0	12	$(0,4)$
1	11	$(0,4)$
2	10	$(0,4)$
3	9	$(0,4)$, $(3,3)$
4	8	$(0,4)$, $(3,3)$
5	7	$(0,4)$, $(3,3)$
6	6	$(0,4)$, $(3,3)$, $(6,2)$
7	5	$(0,4)$, $(3,3)$, $(6,2)$
8	4	$(0,4)$, $(3,3)$, $(6,2)$
9	3	$(0,4)$, $(3,3)$, $(6,2)$, $(9,1)$
10	2	$(0,4)$, $(3,3)$, $(6,2)$, $(9,1)$
11	1	$(0,4)$, $(3,3)$, $(6,2)$, $(9,1)$
12	0	$(0,4)$, $(3,3)$, $(6,2)$, $(9,1)$, $(12,0)$

For fixed d,
time needed to calculate
all $OPT^i[m,b]$
with $(m,b) \in \mathcal{I}(d)$ is
$O(|\mathcal{I}(d)| + |\mathcal{I}'d|) = O(d)$

We just saw how to calculate $OPT^i[m, b]$ for all

$$(m, b) \in \mathcal{I}(d) = \{(m, b) \,:\, m + b = d\}$$

in $O(d)$ time.

We just saw how to calculate $OPT^i[m,b]$ for all

$$(m,b) \in \mathcal{I}(d) = \{(m,b) \,:\, m+b=d\}$$

in $O(d)$ time.

Since $m+b=O(n)$, the entire $OPT^i[m,b]$ table can be partitioned into the $\mathcal{I}(d)$ sets and filled in in time

$$O\left(\sum_d d\right) = O(n^2).$$

We just saw how to calculate $OPT^i[m,b]$ for all

$$(m,b) \in \mathcal{I}(d) = \{(m,b) \,:\, m+b=d\}$$

in $O(d)$ time.

Since $m+b = O(n)$, the entire $OPT^i[m,b]$ table can be partitioned into the $\mathcal{I}(d)$ sets and filled in in time

$$O\left(\sum_d d\right) = O(n^2).$$

To fully solve the problem, we must fill in,

$OPT^1[m,b], OPT^2[m,b], \ldots, OPT^n[m,b]$.

From above this takes only $O(n^3)$ time,

improving upon the old bound of $O(n^4 \log n)$.

- Introduction
  - A Quick Review of Prefix-Free Coding
  - New Results
- The Basic Top-Down Dynamic Programming Technique
- The Speedup
- Conclusion & Comments

Problem	Previous Best Result	This paper
Mixed Radix Coding	$O(n^4 \log n)$ [1]	$O(n^3)$
Reserved Length Coding (i)	$O(gn^3)$ [2]	$O(gn^2)$
Reserved Length Coding (ii)	$O(g^3 n^3 \log^g n)$ [2]	$O(gn^2 \log n)$
One-ended Coding	$O(n^3)$ [3]	$O(n^2)$
The Sound of Silence	$O(n^{U+2})$ [4]	$O(n^{U+1})$

Problem	Previous Best Result	This paper
Mixed Radix Coding	$O(n^4 \log n)$ [1]	$O(n^3)$
Reserved Length Coding (i)	$O(gn^3)$ [2]	$O(gn^2)$
Reserved Length Coding (ii)	$O(g^3 n^3 \log^g n)$ [2]	$O(gn^2 \log n)$
One-ended Coding	$O(n^3)$ [3]	$O(n^2)$
The Sound of Silence	$O(n^{U+2})$ [4]	$O(n^{U+1})$

We just showed how to use batching of dynamic program entries to reduce the running time of mixed-radix coding.

Problem	Previous Best Result	This paper
Mixed Radix Coding	$O(n^4 \log n)$ [1]	$O(n^3)$
Reserved Length Coding (i)	$O(gn^3)$ [2]	$O(gn^2)$
Reserved Length Coding (ii)	$O(g^3 n^3 \log^g n)$ [2]	$O(gn^2 \log n)$
One-ended Coding	$O(n^3)$ [3]	$O(n^2)$
The Sound of Silence	$O(n^{U+2})$ [4]	$O(n^{U+1})$

We just showed how to use batching of dynamic program entries to reduce the running time of mixed-radix coding.

Mixed-Radix coding seems like a very special case.
In reality, all of the above problems can be solved using a top-down DP very similar to the one for mixed-radix coding. The major problem-specific change is in the definition of signature.

Problem	Previous Best Result	This paper
Mixed Radix Coding	$O(n^4 \log n)$ [1]	$O(n^3)$
Reserved Length Coding (i)	$O(gn^3)$ [2]	$O(gn^2)$
Reserved Length Coding (ii)	$O(g^3 n^3 \log^g n)$ [2]	$O(gn^2 \log n)$
One-ended Coding	$O(n^3)$ [3]	$O(n^2)$
The Sound of Silence	$O(n^{U+2})$ [4]	$O(n^{U+1})$

We just showed how to use batching of dynamic program entries to reduce the running time of mixed-radix coding.

Mixed-Radix coding seems like a very special case.
In reality, all of the above problems can be solved using a top-down DP very similar to the one for mixed-radix coding. The major problem-specific change is in the definition of signature.

Furthermore, almost the same type of batching technique, e.g., defining similar $\mathcal{I}(d)$ and $\mathcal{I}'(d)$ and showing that $OPT[m, b]$ for $(m, b) \in \mathcal{I}(d)$ only depend upon values in $\mathcal{I}'(d)$, holds for all of these problems.

A Final Comment

A Final Comment

Literature contains two standard techniques for speeding up dynamic programs;

(i) The Knuth-Yao quadrangle inequality and

(ii) Monge property technques (SMAWK) .

A Final Comment

Literature contains two standard techniques for speeding up dynamic programs;
(i) The Knuth-Yao quadrangle inequality and
(ii) Monge property technques (SMAWK) .

Our original approach was to search for a Monge property in the DP. We found one in Mixed-Radix coding, immediately implying a speedup. The batching technique can be thought of as a simpler speedup.

A Final Comment

Literature contains two standard techniques for speeding up dynamic programs;

(i) The Knuth-Yao quadrangle inequality and

(ii) Monge property technques (SMAWK) .

Our original approach was to search for a Monge property in the DP. We found one in Mixed-Radix coding, immediately implying a speedup. The batching technique can be thought of as a simpler speedup.

The batching technique was later shown to be applicable to other coding problems, such as 1-ended coding, that do not (at least obviously) possess the Monge property.

What other problems can this type of batching speed up?