Lecture 8 - Greedy

AIAA 5037 Advanced Algorithms and Data Structures

Ying Sun, AI Thrust

Outline

- Activity Selection
- Huffman Code

Activity Selection

Activity Selection

Problem: Given a set $A = < a_1, a_2, \ldots, a_n >$ of n activities. Activity a_i takes place during $[s_i, f_i)$, where $0 \leq s_i < f_i < \infty$. Two activities a_i and a_j are compatible if $s_i \geq f_j$ or $s_j \geq f_i$. Assume f_i are in increasing order: $f_1 \leq f_2 \leq \ldots \leq f_n$. Find a largest subset of mutually compatible activities

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

- Example: 1, 4, 8, 11 & 3, 9, 11 are compatible

Brute Force: enumerate all subsequences of A

- Enumeration: 2^n 0-1 strings denoting whether to select each activity in A
- Check the order: at most $O(n)$
- $O(2^n n)$

Dynamic Programming? (Hint: each activity ends before the next begin, similar to LIS)

Dynamic Programming for Activity Selection

A relevant problem that solves the problem: given the set $A = < a_1, a_2, \ldots, a_n >$, find the maximized compatible set (MCS) that ends at position k

- MCS of A can be found among MCS ending at each position: $\max_{k=1}^{n} MCS_k(A).length$

Optimal Substructure: Let $C_i = < a_{c_1}, a_{c_2}, \ldots, a_{c_{k-1}}, a_i >$ be a MCS that ends at position i, $< a_{c_1}, a_{c_2}, \ldots, a_{c_{k-1}} >$ is a MCS that ends at position c_{k-1}, proof:

- $< a_{c_1}, a_{c_2}, \ldots, a_{c_{k-1}} >$ must be a compatible set that ends at position c_{k-1}
- If there is another larger CS that ends at c_{k-1}: $< a_{c_1'}, a_{c_2'}, \ldots, a_{c_{k-1}} >$, then $< a_{c_1'}, a_{c_2'}, \ldots, a_{c_{k-1}}, a_i >$ is a larger compatible set that ends at position i, contradicting the fact that C_i is the MCS

Dynamic Programming for Activity Selection

Optimal Substructure: Let $C_i = < a_{c_1}, a_{c_2}, \dots, a_{c_{k-1}}, a_i >$ be a MCS that ends at position i, $< a_{c_1}, a_{c_2}, \dots, a_{c_{k-1}} >$ is a compatible set that ends at position c_{k-1}

- Traversing all the MCS ending before i and try appending a_i produce the MCS ending at i
- All the MCS ending at c_{k-1} are the same for i, they form an compatible set ending at i of fixed length as long as $f_{k-1} < s_i \Rightarrow$ any one MCS for each position is okay

Dynamic Programming for Activity Selection

Optimal Substructure: Let $C_i = < a_{c_1}, a_{c_2}, \dots, a_{c_{k-1}}, a_i >$ be a MCS that ends at position i, $< a_{c_1}, a_{c_2}, \dots, a_{c_{k-1}} >$ is a compatible set that ends at position c_{k-1}

- Use $g(i)$ to represent the size of the MCS ending at a_i

$$g(i) = \max(1, \max_{j<i, f_j \le s_i} (g(j) + 1))$$

Complexity?

- In the worst case solve all the n subproblems
- The i-th subproblem depends on $i - 1$ subproblems
- In total: $\sum_{i=1}^{n-1} i = O(n^2)$

Time complexity: $O(n^2)$

Space complexity: $O(n)$

- Any better solution?

An Intuitive Strategy

Intuition: traverse from left to right, take part in the first activity if you can

(a sequential and greedy decision-making procedure, take best action for my current stage)

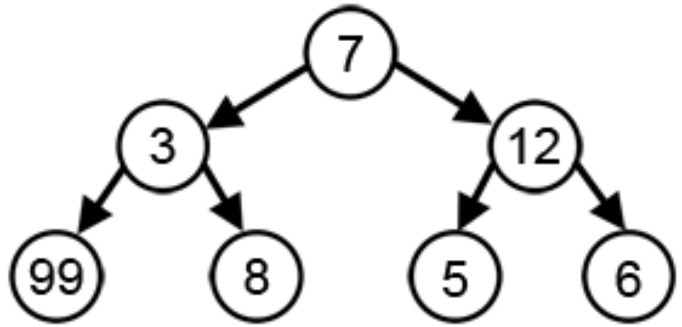
Is it optimal?

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

Greedy Algorithms

Solving a problem by always selecting the best option available now

- Advantages: Simplicity, efficiency, flexibility (at least generate a solution)
- Disadvantages: each choice influence the next choices you can take, may produce sub-optimal solution in some problems
  - Example: find the path from root to leaf with largest node weight



Simple but difficult

Activity Selection

1. Does making decision in the order of ending time lead to suboptimal?
   - No: Space of solution unchanged (we can produce all 0-1 sequence)
2. Does taking part in an activity change your subsequent choices? (suboptimal problem)
   - Yes – no longer choose activities that are incompatible with your choice

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

Activity Selection

Am I making a good choice?

Direction of thinking: given a state when you have selected set $C =< c_1, c_2, \dots, c_k >$, is there a dominant choice? (Dominant Choice: at least as good as any other choices)

- Supposing $f(A^*)$ represents the size of MCS from a set A^*
  - If $A' \subset A''$ (more choices), then $f(A') \leq f(A'')$
- Having chosen C, the rest activities you can choose: $g(A, C) = \{a_i | i > c_k \; and \; f_{c_k} \leq s_i\}$
- If choosing activity a_j then the maximum size you can get is $|C| + f\big(g(A, C \cup \{j\})\big) + 1$
- Observation: if $j_1 < j_2$, then $g(A, C \cup \{j_2\}) \subset g(A, C \cup \{j_1\})$
  - $j_1 < j_2 \Rightarrow f_{j_1} < f_{j_2}$. So, $i > j_2 \; and \; f_{j_2} \leq s_i$ induces $i > j_1 \; and \; f_{j_1} \leq s_i$.
- So $f\big(g(A, C \cup \{j_2\})\big) \leq f(g(A, C \cup \{j_1\}))$

Greedy Choice in a Dynamic Programming View

Another definition for subproblem: Given the set $A = < a_1, a_2, \ldots, a_n >$, find the maximized compatible set (MCS) for a subset $A_{i:j} = \{a_k \in A | \, s_k \geq f_i \; and \; f_k \leq s_j\}$ (activities starting after a_i ends and ending before a_j starts)

Optimal Substructure

- Let $C = < a_{c_1}, a_{c_2}, \ldots, a_{c_{m-1}}, a_{c_m} >$ be an MCS of $A_{i:j}$, then $C_1 = < a_{c_1}, \ldots, a_{c_{k-1}} >$ and $C_2 = < a_{c_{k+1}}, \ldots, a_{c_{m-1}}, a_{c_m} >$ are MCS for $A_{i:c_k}$ and $A_{c_k:j}$
  - Proof: if you can find better one for $A_{i:c_k}$ or $A_{c_k:j}$, you construct a better one for $A_{i:j}$

Traverse all the first choices: $f(A_{i:j}) = \max_{a_u \in A_{i:j}} \{f(A_{i:u}) + f(A_{u:j}) + 1\}$

Greedy choice: a_u is the first-ended activity in A, how to prove?

- We need to traverse u because we do not know whether u is in MCS —— prove a_1 is in one MCS of any A!

Proof for Greedy Choice

Theorem: For any $A\ != \emptyset$, given $a_m = \arg\min_{a_u \in A} f_u$, we have a_m is in one MCS of A

Proof (existence by construction): Let C be any MCS of A that does not contain a_m, $a_j = \arg\min_{a_u \in C} f_u$, we can construct another MCS containing a_m

- We construct $C' = (C - \{a_j\}) \cup \{a_m\}$, then $|C| = |C'|$.
- Obviously, $f_m \leq f_j < s_u$ for any $j != u \in C$
- Then C' is also compatible and has the same size as C
- C' is MCS containing a_m

Implementation

Recursive

```
ACTIVITY-SELECTOR(s,f,k,n)
1. m = k + 1
2. while m ≤ n and s[m] < f[k]
3.     # find the first activity to finish
4.     m = m + 1
5. if m ≤ n
6.     return {a_m} ∪ ACTIVITY-SELECTOR(s, f, m, n)
7. else return ∅

ACTIVITY-SELECTOR(s,f,0,n)
```

Iterative

```
GREEDY-ACTIVITY-SELECTOR(s,f,n)
1. n = s.length
2. A = {a_1}
3. k = 1
4. for m = 2 to n
5.     if s[m] ≥ f[k]
6.         A = A ∪ {a_m}
7.         k = m
8. return A
```

- Time complexity: $O(n)$
- Auxiliary Space complexity: $O(n)$

Greedy and Dynamic Programming

Both construct an optimal solution based on optimal solution of subproblems

- Dynamic Programming: try all the subproblems
- Greedy: directly find one best subproblem

Characteristics for using Greedy algorithms

Optimal Substructure Property: the optimal solution of a problem contains the optimal solution of its subproblems

Greedy Choice Property: There is a choice which must be contained in an optimal solution

Huffman Code

Background: Character Encoding

Data is encoded as binary digits (bits) in a computer

Question: what is the best way to encode characters to minimize storage

Fixed-length Encoding: represent n characters with $\log n$ bits

a	b	c	d	e	f
000	001	010	011	100	101

- Example: abbcd -> 000001001010011
- Space to store a file of length m composed of n distinct characters: $O(m\log n)$

Can storage be further optimized?

- **Key Observation**: Characters in typical datasets do not appear with equal frequency—some are used more than others

Background: Variable-Length Encoding

Characters in typical datasets do not appear with equal frequency

Example File:

aaaaaaaacbaaaaaaaaaaaaaaaacaaaeaaaaeaaaaaaaaabaaaaacaacaaaaaaaabaabaaacaaaaaebaaaaaaaaaaaaaaaabaaaaaaaacaaabaeaabaaaaabbaaabaaaacababaaaaabaaabaaaaaa
aaaaaaaaaacaaaaaeabeaaaaaabbaaadaaaaabfaaaaaeaaaaaaaaaaaabaaaaaabbaaaaaaaaaaaaaaaaadbaadaaaaaaaaaaaaaaaadaaaabaaaaaaaaaaaaaaabaaaacaebaaaaaaaadbaadaaaaba
baaaaaabaaaabaabaaaaaaabababaaaaaafaaaaaaaaabababaaaaabaadcaabaabebaaaafaaaaaaaaaabaaaaabaabbaafaaaaaaaaaaaaaaaaaaababaaaabaacaaaababaabaaaaaaaaaaaaaaaaa
aaaaaacabaaaafaaaaaaaabaaaaaaaaabaaaaaaaaabaaaaabaaaaaeaaabbaaabaaaabaabaeaaaaacdaaaacaaabbbabcaaaaaaaaabaabbaaaabcaaaaaaaaaaaaaaaabaaaaacaaaaaaabaaaaaa
aaaaaabeabaabcaacaacabaaaaaacaaaaaaabaaadaaaaaaeacbaadaaaacaaaaaaaaaaabbabaabaaaaaaaabbaaacbaaacaaaaaaaaaaaaafaaabaaaaaaaaaaaaaafeaabaaaaaacaaaaaaaaaaaa
aaafaaaaaaaaaaaabaaaabaaaaaaaabbaacaaaaaaaabaaaaaaaaaaacaaaeaaaaaaaeaabaaaaaaaaaaaaaaaabaaaaaaabbaacaaaaaaaaaacaaaaaaaaaaaaaaaaeafaaaaaaaaaaaaaaaaaa
aaaaaaaaeababbaaaaaaaaaaaaaafbaeaaaaaaaaabaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaebaaaaaaaaa

	a	b	c	d	e	f
frequency	830	100	30	10	20	10

Naïve Encoding:

a	b	c	d	e	f
000	001	010	011	100	101

Total bits: 3 * 1000 = 3000

Make ‘a’ shorter at the cost of the others longer? Example:

a	b	c	d	e	f
0	1000	1001	1010	1011	1100

1 * 830 + 4 * 170 = 1510

Background: Decoding

Match the codeword in the dictionary

- e.g., 0010001010001100 -> aabdaaf

a	b	c	d	e	f
0	1000	1001	1010	1011	1100

Question: Is this a good encoding system?

a	b	c	d	e	f
0	1	00	01	10	11

Encoding: acd->00001

Ambiguous Decoding: 00001 -> ?

Unique Decoding: Ensures that each encoded string is decodable in only one way

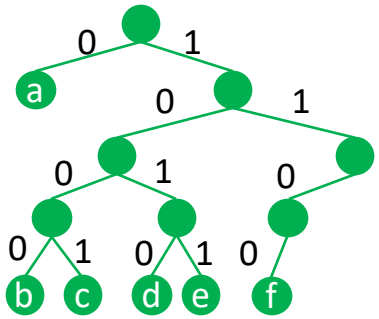
- **Prefix Code:** A type of code system where no codeword is a prefix of any other codeword

How to realize the decoding algorithm?

Background: Decoding

Organize the code dictionary in a binary tree structure (*prefix tree*)

- Nodes: characters or empty node
- Links: labeled with 0 (left child) or 1 (right child)
- **Decoding Process**:
  - Start at the root
  - Go left when encountering 0, right when encountering 1
  - Reach character node (leaves): append character to output, return to root

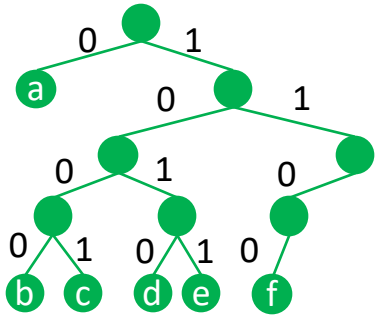


a	b	c	d	e	f
0	1000	1001	1010	1011	1100

0010001010001100 -> aabdaaf

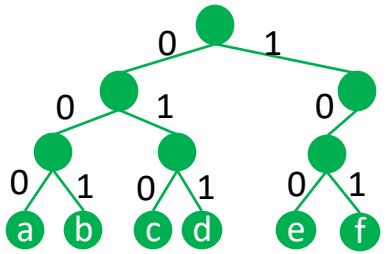
Background: Decoding

Prefix Tree for Prefix Code: Only leaf nodes have characters, no ambiguous matching



Prefix Code

a	b	c	d	e	f
0	1000	1001	1010	1011	1100

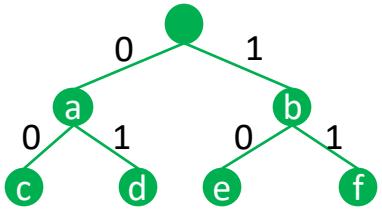


Prefix Code

a	b	c	d	e	f
000	001	010	011	100	101

None-Prefix Code

a	b	c	d	e	f
0	1	00	01	10	11



Best Codeword Problem

Problem: Given n characters represented as $A = \{a_1, a_2, \ldots a_n\}$, a_i represents the frequency of the i-th character, find the best prefix code that minimizes the total encoding length

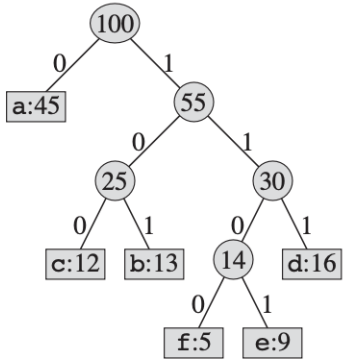
Storage given a prefix tree T

- $d_T(i)$: the depth of character i
- $B(T) = \sum_{i=1}^{n} a_i d_T(i)$

Problem transformed: find the best T that minimizes $B(T)$

Another way to calculate the storage

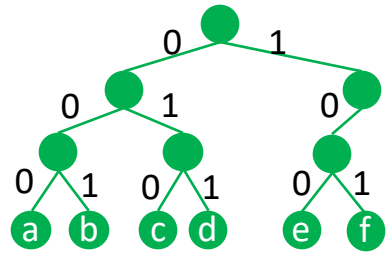
- $c_T(u)$: the total frequency of characters stored in subtree root on u
- $B(T) = \sum_{u \in T.nodes - \{T.root\}} c_T(u)$



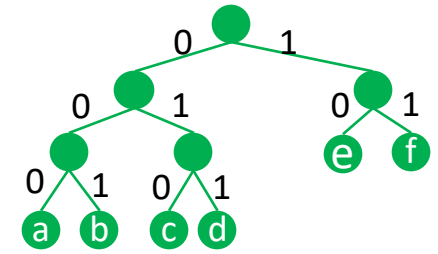
	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

What makes a Better Prefix Code Tree

Can we make this prefix code better?



a	b	c	d	e	f
000	001	010	011	100	101



a	b	c	d	e	f
000	001	010	011	10	11

We can raise the level of node e (100 -> 10) and f (101->11)

Generalize: nodes without siblings can be merged with its parent to decrease 1 bit

Important Property: the optimal solution is a full binary tree

- Each intrinsic node has two children

Build Prefix Code Tree

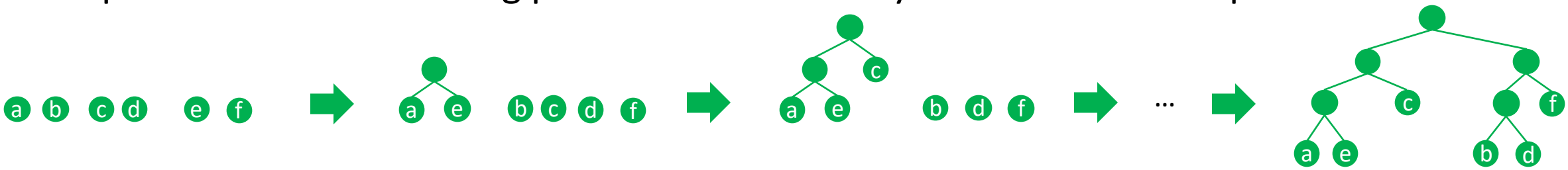
Properties we have so far:

1. Characters are leaf nodes, and no other leaf nodes
2. Full binary tree

A procedure to build ***any tree structure*** satisfying these two properties:

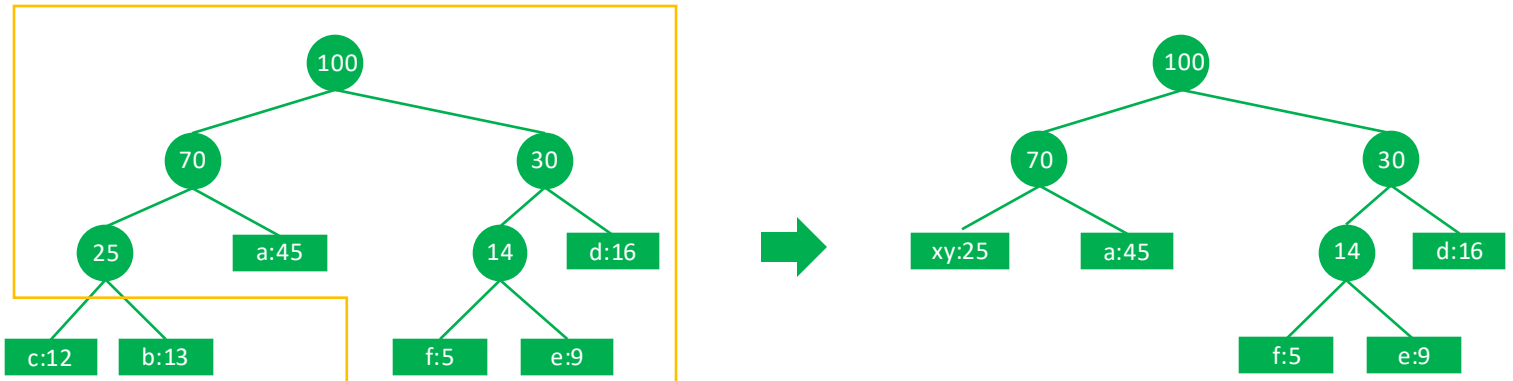
1. Start with n leaf nodes, each regarded as a tree (property 1)
2. (Repeatedly) merge two trees with a new root (property 2)

A sequential decision-making procedure like activity selection to find optimal solution



Optimal Substructure

Given an optimal tree T for n characters with frequency $A = \{a_1, a_2, \ldots, a_n\}$, supposing two characters x and y are siblings in T, T contains a tree T' which is an optimal tree for $n - 1$ characters with frequency set $A' = \left(A - \{a_x, a_y\}\right) \cup \{a_x + a_y\}$



Proof:

- T' is a prefix code tree for the new set A'
- If T' is not optimal and there is a better solution T'', you can construct a better prefix code tree for A (just add the two nodes as children for the merged node xy)

Recurrence for Optimized Value

Given an optimal tree T for n characters with frequency $A = \{a_1, a_2, \dots, a_n\}$, supposing two characters x and y are siblings in T, T contains a tree T' which is an optimal tree for $n - 1$ characters with frequency set $A' = \left(A - \{a_x, a_y\}\right) \cup \{a_x + a_y\}$

- If first merging x and y, the optimized value is $f(\left(A - \{a_x, a_y\}\right) \cup \{a_x + a_y\}) + a_x + a_y$
- Traverse all the first merges: $f(A) = \min_{x,y}\{f(\left(A - \{a_x, a_y\}\right) \cup \{a_x + a_y\}) + a_x + a_y\}$

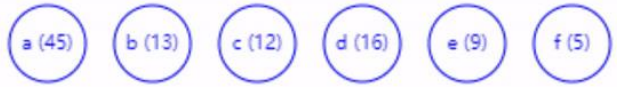
Complexity: imagine the number of subproblems, extremely high!

Intuition:

- Merging two subtree increases depth by 1 for the two subtrees in the resulting tree
- $B(T) = \sum_{i=1}^{n} a_i d_T(i)$ means larger weights should have lower depth
- Maybe we should merge (increase depth for) nodes with smaller weight

Huffman Code

Greedily merge two trees with the least sum of leaves weights




```
HUFFMAN(C)
1  n = |C|
2  Q = C
3  for i = 1 to n − 1
4       allocate a new node z
5       z.left = x = EXTRACT-MIN(Q)
6       z.right = y = EXTRACT-MIN(Q)
7       z.freq = x.freq + y.freq
8       INSERT(Q, z)
9  return EXTRACT-MIN(Q)      // return the root of the tree
```

Is it optimal (Greedy choice property)?

—— prove there is an optimal tree where the smallest weighted characters x, y are siblings

Proof

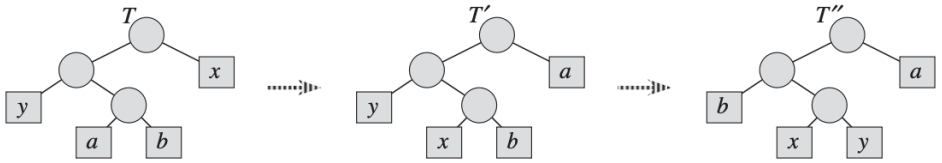
Given n characters represented as $A = \{a_1, a_2, \ldots a_n\}$, there is an optimal prefix code tree where the smallest weighted characters x and y are siblings

Main idea (prove existence by construction): we can construct an optimal solution T', where x and y are siblings, from an arbitrary optimal solution T

- Let u and v be sibling leaves of maximum depth in T, supposing $a_u \leq a_v$ and $a_x \leq a_y$
- x and y has lowest-2 frequencies: $a_u \geq a_x$ and $a_v \geq a_y$; u and v has largest depth: $d(x) \leq d(u)$ and $d(y) \leq d(v)$
- swap(u, x) and swap(v, y):

$$B(T') - B(T) = \big(d(u)a_x + d(x)a_u + d(v)a_y + d(y)a_v\big) - \big(d(u)a_u + d(x)a_x + d(v)a_v + d(y)a_y\big)$$
$$= \big(d(x) - d(u)\big)(a_u - a_x) + \big(d(y) - d(v)\big)\big(a_v - a_y\big) \leq 0$$

- So $B(T') \leq B(T)$. Since $B(T) = \min_{T^*} B(T^*)$, we have $B(T') = B(T)$, so T' is also optimal



Complexity

Space Complexity: Full binary tree - $O(n)$

Time complexity

- Key point: find the two trees with the smallest weight
  - Brute force: $O(n)$ for each step, $O(n^2)$ in total
  - Min-heap: $O(\log n)$ for each step, $O(n \log n)$ in total
  - Van Emde Boas tree: $O(\log \log n)$ for each step, $O(n \log \log n)$ in total

```
HUFFMAN(C)
1  n = |C|
2  Q = C
3  for i = 1 to n − 1                          O(n) iterations
4      allocate a new node z
5      z.left = x = EXTRACT-MIN(Q)             O(g(n)) ?
6      z.right = y = EXTRACT-MIN(Q)            O(g(n)) ?
7      z.freq = x.freq + y.freq
8      INSERT(Q, z)                            O(g(n)) ?
9  return EXTRACT-MIN(Q)      // return the root of the tree
```

Assignment

You can choose one of the following questions to answer (provide the algorithm and prove the correctness)

- Some people want to cross a river by boat. Each person has a weight, and each boat can carry an equal maximum weight limit. Each boat carries at most 2 people at the same time, provided the sum of the weight of those people is at most boat's weight limit. Return the minimum number of boats to carry every given person. Note that it is guaranteed each person can be carried by a boat.

- Bob has n toy buildings in a line, the i-th from left of which has weight ai. He wants to make these building nondecreasing in height from left to right. In one operation, he can take any contiguous subsegment of them and add 1 to each of their heights. Help Bob find the minimum number of operations he needs to perform to make his toy buildings nondecreasing.

Thank you!

AIAA 5037 Advanced Algorithms and Data Structures