CS 170
Efficient Algorithms and Intractable Problems

Lecture 9
Greedy Algorithms II

Nika Haghtalab and John Wright

EECS, UC Berkeley

Announcements

Midterm on Oct 1.
→ Look for Ed post on logistics

Second half of lecture today: Quiz 1

Greedy Algorithms

Algorithms that build up a solution

piece by piece, always choosing the next piece

that offers the most obvious and immediate benefit!

We saw:

- Scheduling
- Optimal Coding

Today:

- Wrap up Optimal Coding
- Satisfiability

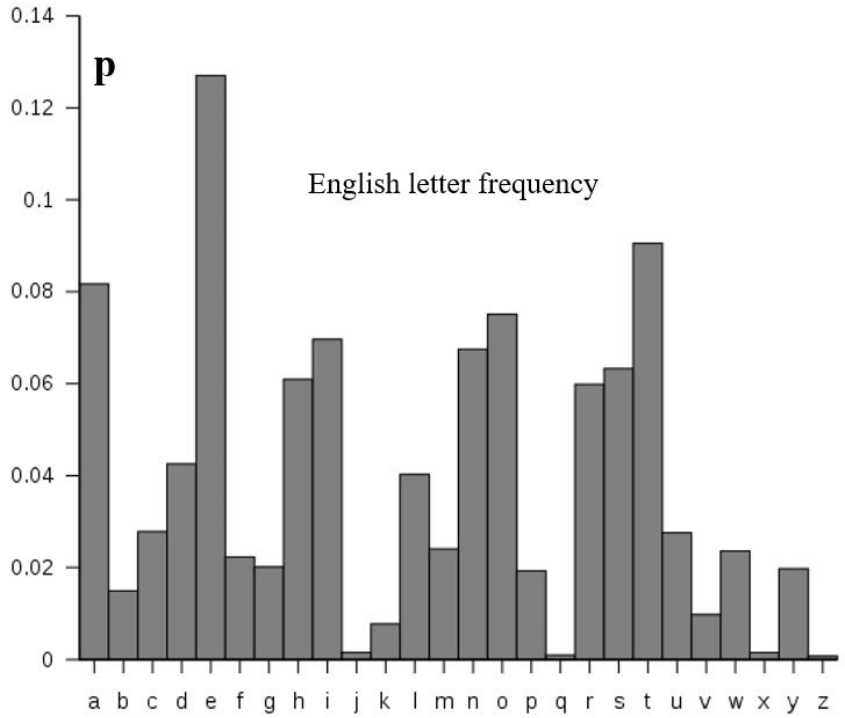
Data Compression and Encoding

Recap!

Common encodings of English characters use a fixed length of code per character.

If the goal is to save space, can we encode the alphabet better?

- If we know which letters are more common
- Use shorter codes for very common characters (like e, a, s, t).

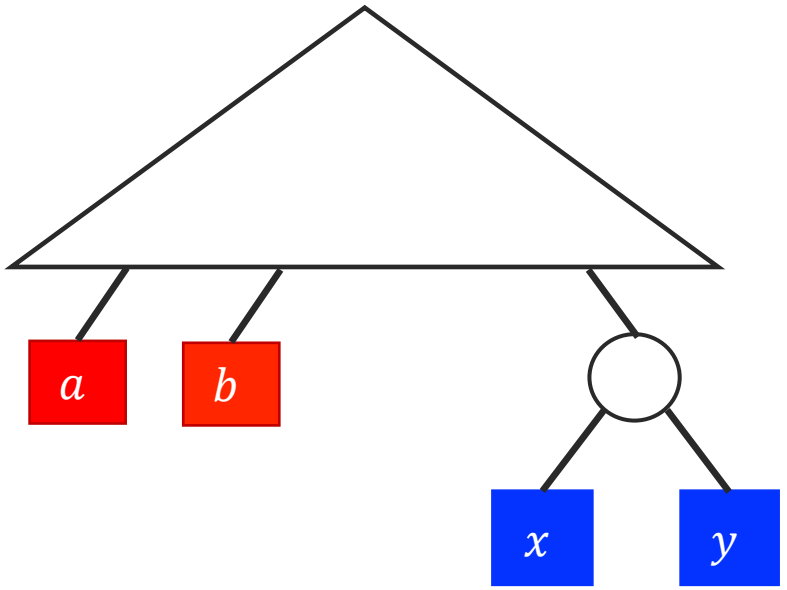


What do optimal subtrees look like?

Recap!

W proved that:

There is an optimal tree where the two lowest freq. symbols are sibling leaves.



Greedy algorithm

Recap!

Huffman coding: Greedily build subtrees from the lowest frequency letters.

Node a object with

a.freq = f_a

a.left = left child

a.right = right child

Huffman-code$(f_1, \dots, f_n)$

For all $a = 1, \dots, n$,

create node a with $a.\text{freq} = f_a$ and no children

Insert the node in a priority queue Q use key f_a

While $\text{len}(Q) > 1$

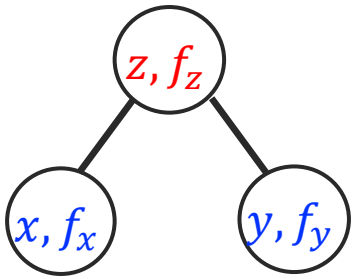
x and $y \leftarrow$ the nodes in Q with lowest keys

create a node z, with $z.\text{freq} = x.\text{freq} + y.\text{freq}$

Let $z.left = x$ and $z.right = y$.

Insert z with key f_z into Q and remove x, y.

Return the only node left in Q.

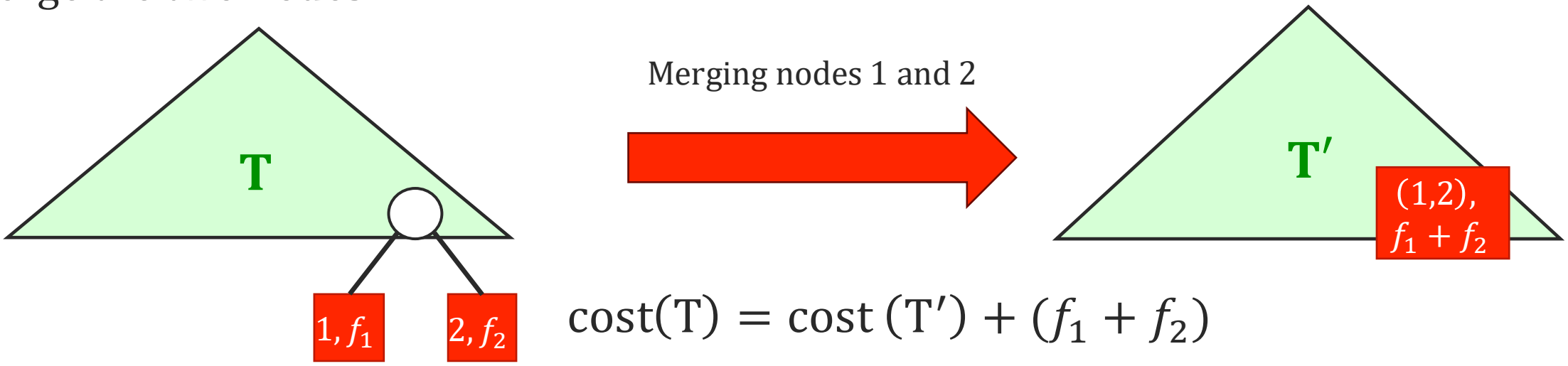


Optimality of Huffman Coding

Induction step: Assuming that Huffman tree produces an optimal tree for $n-1$ letters, show that it produces an optimal tree for n letters.

Let T below be the optimal prefix-free tree for frequencies $f_1, \ldots, f_n$ and WLOG $f_1 \leq f_2 \leq \cdots f_n$.

- WLOG, assume that the two lowest frequency nodes are siblings.
  → Because, we proved earlier that this claim is true!
- Merge the two nodes

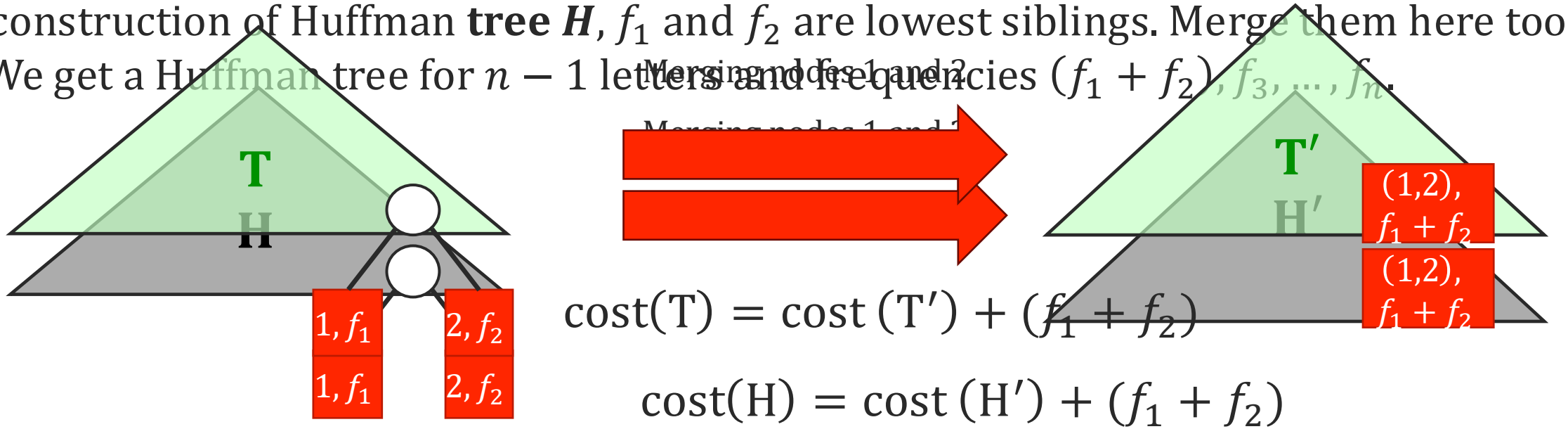


$$\text{cost}(\text{T}) = \text{cost}\,(\text{T}') + (f_1 + f_2)$$

Optimality of Huffman Coding

Claim: Huffman coding is an optimal prefix-free tree.

By construction of Huffman **tree *H***, f_1 and f_2 are lowest siblings. Merge them here too.
→ We get a Huffman tree for $n-1$ letters and frequencies $(f_1 + f_2), f_3, \dots, f_n$.



$$\text{cost}(\text{T}) = \text{cost}(\text{T}') + (f_1 + f_2)$$

$$\text{cost}(\text{H}) = \text{cost}(\text{H}') + (f_1 + f_2)$$

Optimality of Huffman Coding

Claim: Huffman coding is an optimal prefix-free tree.

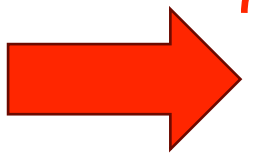
We showed that for tree T that is optimal for n letters, $\text{Cost}(\text{T}) = \text{cost}\,(\text{T}') + (f_1 + f_2)$.

And for Huffman coding tree H for n letters, $\text{Cost}(\text{H}) = \text{cost}\,(\text{H}') + (f_1 + f_2)$.

Putting everything together.

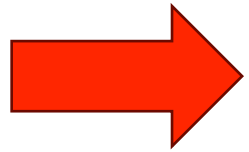
$n-1$ letters

$\text{cost}\,(\text{H}') \leq \text{cost}(T')$



n letters

$\text{cost}\,(H) \leq \text{cost}(T)$



Huffman is optimal for n letters

By induction hypothesis, Huffman coding for $n-1$ letters is optimal

Next up: Greedy for SAT

Horn Formula

Variables: $x_1, \ldots, x_n \in \{True, False\}$, a literal is x_i or $\overline{x_i}$.

Clauses:

1. "Implication clause" (with no negative variable)

$$\left(x_i \wedge x_j \wedge \cdots\right) \Rightarrow x_k \longleftrightarrow \text{Equivalent to } \overline{x_i} \vee \overline{x_j} \vee \cdots \vee x_k$$

2. "Pure negative clauses"

$$\left(\overline{x_i} \vee \overline{x_j} \vee \cdots\right)$$

Horn Formula: AND of m Horn clauses

Horn Clause's Significance

Why care about these clauses?

→Used in computational logic, theorem proving, etc.

→Prolog is based on Horn clauses.

Horn-SAT

Implication clause
$(x_i \wedge x_j \wedge \cdots) \Rightarrow x_k$

Pure negative clauses
$(\overline{x_i} \vee \overline{x_j} \vee \cdots)$

Input:

A Horn formula (AND of Horn clauses)

Output:

Find an assignment for the variables that makes the Horn formula True, if such and assignment exists.

Greedy Algorithm for Horn-SAT

Horn-Formula Clauses:

$(w \wedge y \wedge z) \Rightarrow x$

$(x \wedge z) \Rightarrow w$

$x \Rightarrow y$

$\Rightarrow x$

$(x \wedge y) \Rightarrow w$

$(\bar{w} \vee \bar{x} \vee \bar{y})$

Variable assignments:

x

y

z

w

For all i, set $x_i = False$

While there exists $\left((x_i \wedge \cdots \wedge x_j) \Rightarrow x_k\right) = False$

Set $x_k = True$

If every pure negative clause $\left(\bar{x_i} \vee \cdots \vee \bar{x_j}\right) = True$

Return $(x_1, \ldots, x_n)$

Else

Return “not satisfiable”

Why does Greedy Work for Horn-SAT?

What's the pattern in this case?

We want to establish that when Greedy sets a variable $x_i = True$, it does not ruin a satisfying assignment.

In fact, we will prove

→ The set of variables set to True by the Greedy algorithm, are also set to True in any satisfying assignment.

Proof of Claim

Claim: The variables set to True by Greedy, are also True in the satisfying solution.

Proof: By induction on the iteration of the While loop

Base case: In the 0^{th} iteration of the While loop, nothing is set to True.

Induction hypothesis: The first m variables set to True by Greedy are also True in every satisfying solution.

Inductive step:

- Let x_{m+1} be the $m+1^{st}$ variable set to True by Greedy.

→ Only happens if there was an unsatisfied implication $(x_i \wedge \cdots \wedge x_j) \Rightarrow x_{m+1}$ before the $m+1$ iteration of the while loop. i.e., $x_{m+1} = False$ and $(x_i \wedge \cdots \wedge x_j) = True$.

→ If $(x_i \wedge \cdots \wedge x_j) = True$ before the m+1 iteration of Greedy, then $(x_i \wedge \cdots \wedge x_j) = True$ also in the satisfying solution.

- The only way to satisfy this clause in SAT is to also have $x_{m+1} = True$.

Horn-SAT Proof completed

Claim: The greedy solution is correct.

1) If Greedy outputs a solution, then the solution is satisfiable.

This is true because the While loop and If condition check that all clauses are satisfied

2) If the Horn Formula is satisfiable, then Greedy outputs a satisfiable solution.

Assume to the contrary that this is not true. So, Greedy outputs "unsatisfiable" even though a satisfying assignment exists.

→In Greedy's assignment, there is a violated pure clause $(\bar{x}_i \vee \cdots \vee \bar{x}_j)$

→So, every variable in this clause is set to True

→By previous slide, these variables are also set to True in any satisfiable solution and this clause is also violated by the satisfying assignment

→Contradiction!

Wrap up

We saw one more example of greedy algorithms.

Next time, we'll see greedy algorithms on graphs.

Now … Quiz 1