Greedy Algorithms

Lecture 16
Backtracking

• We have seen Backtracking/DP so far
  — Make a simple choice
  — Recursively solve everything else

e.g. **Subset Sum**: is a certain element of the set in the subset or not? If only we could know…
Backtracking

• We have seen Backtracking/DP so far
  — Try all options for
  — Make a simple choice
  — Recursively solve everything else For each choice!

  e.g. **Subset Sum**: is a certain element of the set in the subset or not? If only we could know…

  **LIS**: Do I include an element in the sequence or not?

  **NFA accept**: should I transition to a certain state?

(see nondeterminism)
Backtracking

• We have seen Backtracking/DP so far
  Try all options for
  — Make a simple choice
  — Recursively solve everything else For each choice!

Greedy

Really tempting to

• Choose one option

• Recurse (e.g. Edit Distance: choose two characters that are equal to leave them as such)
Course Policy on Greedy

- When you use greedy algorithm, you need to ALWAYS prove correctness. Otherwise you get a zero, EVEN IF THE ALGORITHM IS CORRECT!

- Greedy is a loaded gun!
Greedy Algorithm Example

- Sorting files on magnetic tape (not RAM)
- Remember music cassettes?
- Blue Water Supercomputer.
Greedy Algorithm Example

- Sorting files on magnetic tape (not RAM)
- Remember music cassettes?
- Blue Water Supercomputer.

The Problem:

- Given an array of lengths of each file: \( L[1...n] \)
- I want to sort the files so that if someone asks me for a random file, the expected time it takes to wind the tape to the start of the file and rewind it back is small.
The Problem:

• Given an array of lengths of each file: \( L[1\ldots n] \)

• I want to sort the files so that if someone asks me for a random file, the expected time it takes to wind the tape to the start of the file and rewind it back is small.

• Formally, I want to find a permutation that minimizes

\[
\sum_{k=1}^{n} \sum_{i=1}^{k} L[\pi(i)]
\]

Where \( \pi(i) \) is the index of the file sorted in position \( i \) of the tape
What order should I sort them?

Claim:
Sort L, in increasing order of lengths is the best solution

\[ L[\pi(i)] \leq L[\pi(i + 1)] \quad \text{for all } i \]

Needs proof!!!
Sorting Files on Tape

Proof:
Assume in optimal ordering $\pi$

$L[\pi(i)] > L[\pi(i + 1)]$ for some $i$

what happens if we switch A and B?

\[
\begin{array}{c}
i & i+1 \\
\hline
A & B \\
\end{array}
\]

\[
\begin{array}{c}
i & i+1 \\
\hline
B & A \\
\end{array}
\]
Sorting Files on Tape

**Proof:**

Assume in optimal ordering $\pi$

$L[\pi(i)] > L[\pi(i + 1)]$ for some $i$

Cost(A) increases by $L[B]$
Cost(B) decreases by $L[A]$

Total cost increases by $L[B] - L[A] < 0$
Exchange Argument

• Consider any non-greedy solution

• Perform an exchange to make the solution look more greedy

• Argue that the new solution after doing the exchange is no worse.

• In our example, the new solution was strictly better, so greedy is the only way.
Sorting Files on Tape

• What if I also had frequencies?

• L[1…n] lengths of files and F[1…n] frequencies.

• Need to minimize: \[ \sum_{k=1}^{n} \sum_{i=1}^{k} (F[\pi(k)] \cdot L[\pi(i)]) \]

• If all the lengths the same and frequencies different?

Class Scheduling

- University decides to start a new major, CS+ climbing

- Degree requirements involve taking certain number of classes, certain hours and certain categories.

- Bulk of the degree is determined by taking a certain number of classes. None of these classes require actual work.

- Without the instructors permission, you cannot register for two classes whose times overlap.

- You only need to sign up! Goal: sign up for as many classes as possible, without overlapping classes.
Class Scheduling

• Given a collection of intervals with start and end time, want to choose a subset of those intervals such that no pair overlaps.

• Subset needs to be as large as possible.

• Model it as a graph problem (next time): Independent Set!
Class Scheduling

• Algorithm? DP? Greedy?

• e.g. find the earliest class, take it and recurse

• find the longest class, throw it away and recurse.

• find the shortest class, take it and recurse.
Class Scheduling

• None of those work!

• Instead: pick the class the ends earliest
Class Scheduling

• None of those work!

• Instead: pick the class the ends earliest
Class Scheduling

• None of those work!

• Instead: pick the class the ends earliest
Class Scheduling

• None of those work!

• Instead: pick the class the ends earliest
Class Scheduling

• None of those work!

• Instead: pick the class the ends earliest
Class Scheduling

- Sort classes according to finish time

Because of sorting, $O(n \log n)$, while DP in $O(n^2)$
Class Scheduling

• Why is it optimal? Proof!

• Not the only optimal schedule. There are many optimal schedules.
Class Scheduling

• Exchange argument.

• Think of it as a recursive algorithm. Pick the class what finishes first and then recurse.

• Proof by induction!

Lemma:

At least one maximal conflict free schedule includes the class that ends first.
Class Scheduling

Lemma:

At least one maximal conflict free schedule includes the class that ends first.

Proof:

Let \( f \) be the class that ends first. Consider any schedule \( X \) that excludes \( f \). Let \( g \) be the first class ending in \( X \). \( F[f] < F[g] \) implies that \( f \) does not overlap any class in \( X \setminus \{g\} \). Let \( Y = X - \{g\} + \{f\} \) is a valid schedule of same size! What if \( X \) is empty?
Huffman Codes

• Binary code assigns a string of 0s and 1s to each character in the alphabet.

• 7-bit ASCII code, Unicode, Morse

• We want the code to be prefix free (Morse code is not).

• Any prefix free code can be visualized as a binary code tree, where the characters are stored at the leafs.

• Codeword for each symbol is given by the path from the root to the corresponding leaf (e.g. 1 for right 0 for left).

• Length of codeword for a symbol is the depth of the corresponding leaf.
Huffman Codes

• Goal is to encode messages in an $n$-character alphabet so that the encoded message is as short as possible.

• Given array of frequencies: $f[1...n]$, we want to compute a prefix-free binary code that minimizes the total encoded length of message.

$$\sum_{i=1}^{n} f[i] \cdot \text{depth}(i)$$
Huffman Codes

10111
Huffman Codes

This sentence contains three a’s, three c’s, two d’s, twenty-six e’s, five f’s, three g’s, eight h’s, thirteen i’s, two l’s, sixteen n’s, nine o’s, six r’s, twenty-seven s’s, twenty-two t’s, two u’s, five v’s, eight w’s, four x’s, five y’s, and only one z.

| A | C | D | E | F | G | H | I | L | N | O | R | S | T | U | V | W | X | Y | Z |
| 3 | 3 | 2 | 26| 5 | 3 | 8 | 13| 2 | 16| 9 | 6 | 27| 22| 2 | 5 | 8 | 4 | 5 | 1 |

Huffman’s algorithm: merge two least frequent letters and recurse!
Huffman Codes

| A | C | D | E | F | G | H | I | L | N | O | R | S | T | U | V | W | X | Y | Z |
| 3 | 3 | 2 | 26| 5 | 3 | 8 | 13| 2 | 16| 9 | 6 | 27| 22| 2 | 5 | 8 | 4 | 5 | 1 |

| A | C | E | F | G | H | I | L | N | O | R | S | T | U | V | W | X | Y | Z |
| 3 | 3 | 26| 5 | 3 | 8 | 13| 2 | 16| 9 | 6 | 27| 22| 2 | 5 | 8 | 4 | 5 | 3 | 1 |
Lemma: Let x and y be the two least frequent characters. There is an optimal code tree in which x and y are siblings, and have the largest depth of any leaf.

Proof: Exchange argument!

Assume, for the optimal schedule that the deepest two leaves are not x and y.
**BUILDHUFFMAN**($f[1..n]$):

for $i \leftarrow 1$ to $n$

$L[i] \leftarrow 0$; $R[i] \leftarrow 0$

INSERT($i, f[i]$)

for $i \leftarrow n$ to $2n - 1$

$x \leftarrow $ EXTRACTMIN()

$y \leftarrow $ EXTRACTMIN()

$f[i] \leftarrow f[x] + f[y]$

$L[i] \leftarrow x$; $R[i] \leftarrow y$

$P[x] \leftarrow i$; $P[y] \leftarrow i$

INSERT($i, f[i]$)

$P[2n - 1] \leftarrow 0$

**HUFFMANENCODE**($A[1..k]$):

$m \leftarrow 1$

for $i \leftarrow 1$ to $k$

HUFFMANENCODEONE($A[i]$)

**HUFFMANENCODEONE**($x$):

if $x < 2n - 1$

HUFFMANENCODEONE($P[x]$)

if $x = L[P[x]]$

$B[m] \leftarrow 0$

else

$B[m] \leftarrow 1$

$m \leftarrow m + 1$

**HUFFMANDECODE**($B[1..m]$):

$k \leftarrow 1$

$v \leftarrow 2n - 1$

for $i \leftarrow 1$ to $m$

if $B[i] = 0$

$v \leftarrow L[v]$

else

$v \leftarrow R[v]$

if $L[v] = 0$

$A[k] \leftarrow v$

$k \leftarrow k + 1$

$v \leftarrow 2n - 1$