Greedy algorithms

David Kauchak

cs140

Spring 2023

Administrative

Assignment 5 due today at 9pm

Assignment 6 out later today and due next Friday 3/10 at 8pm

LC meetings this week

LC meetings next week?

Greedy vs. divide and conquer

Greedy algorithms

Algorithm that makes a local decision with the goal of creating a globally optimal solution

Method for solving problems where optimal solutions can be defined in terms of optimal solutions to sub-problems

Greedy vs. divide and conquer

Divide and conquer

To solve the general problem:

Break into sum number of sub problems, solve:

then possibly do a little work
Greedy vs. divide and conquer

**Divide and conquer**

To solve the general problem:

The solution to the general problem is solved with respect to solutions to sub-problems!

---

**Greedy**

To solve the general problem:

Pick a locally optimal solution and repeat

The solution to the general problem is solved with respect to solutions to sub-problems!

Slightly different than divide and conquer

---

**Horn formula**

A horn formula is a set of implications and negative clauses:

\[ \Rightarrow x \quad x \land u \Rightarrow z \]
\[ \Rightarrow y \quad \overline{x} \lor \overline{y} \lor \overline{z} \]
Goal
Given a horn formula, determine if the formula is satisfiable, i.e. an assignment of true/false to the variables that is consistent with all of the implications/causes

\[ x \implies y \implies z \]
\[ u \land x \land y \land z \]

A greedy solution?

\[ w \land y \land z \implies x \]
\[ x \implies w \land \neg x \lor \neg y \]

u x y z
0 1 1 0
A greedy solution?

\[ x \Rightarrow x \land z \Rightarrow w \land y \land z \Rightarrow x \]

\[ x \Rightarrow y \]

\[ x \land y \Rightarrow w \land \overline{w} \lor \overline{x} \lor \overline{y} \]

\[
\begin{align*}
\text{w} & : 1 \\
\text{x} & : 1 \\
\text{y} & : 1 \\
\text{z} & : 0
\end{align*}
\]

not satisfiable

A greedy solution?

\[ x \Rightarrow x \land z \Rightarrow w \land y \land z \Rightarrow x \]

\[ x \Rightarrow y \]

\[ x \land y \Rightarrow w \land \overline{w} \lor \overline{x} \lor \overline{y} \]

\[
\begin{align*}
\text{w} & : 1 \\
\text{x} & : 1 \\
\text{y} & : 1 \\
\text{z} & : 0
\end{align*}
\]

not satisfiable

A greedy solution

Horn(B)

1. set all variables to false
2. if empty(LHS(i))
3. changed := true
4. while changed
5. changed := false
6. while changed
7. changed := false
8. for all implications i
9. if LHS(i) => true and RHS(i) := true
10. RHS(i) := true
11. changed := true
12. for all negative clauses c
13. if c = false
14. return false
15. return true

A greedy solution

Horn(B)

1. set all variables to false
2. if empty(LHS(i))
3. changed := true
4. while changed
5. changed := false
6. while changed
7. changed := false
8. for all implications i
9. if LHS(i) => true and RHS(i) := true
10. RHS(i) := true
11. changed := true
12. for all negative clauses c
13. if c = false
14. return false
15. return true
A greedy solution

Horn(IF)
1 set all variables to false
2 for all implications i
3 if EMPTY(LHS(i))
4 RHS(i) ← true
5 changed ← true
6 while changed
7 changed ← false
8 for all implications i
9 if LHS(i) = true and RHS(i) = true
10 RHS(i) ← true
11 changed ← true
12 for all negative clauses c
13 if c = false
14 return false
15 return true

How is this a greedy algorithm?

Horn(IF)
1 set all variables to false
2 for all implications i
3 if EMPTY(LHS(i))
4 RHS(i) ← true
5 changed ← true
6 while changed
7 changed ← false
8 for all implications i
9 if LHS(i) = true and RHS(i) = true
10 RHS(i) ← true
11 changed ← true
12 for all negative clauses c
13 if c = false
14 return false
15 return true

see if all of the negative clauses are satisfied

How is this a greedy algorithm?

Make a greedy decision about which variables to set and then moves on.
Correctness of greedy solution

Two parts:
- If our algorithm returns an assignment, is it a valid assignment?
- If our algorithm does not return an assignment, does an assignment exist?

```plaintext
Correctness of greedy solution

If our algorithm returns an assignment, is it a valid assignment?

```Hence()
1. set all variables to false
2. for all implications i
   3. if \( \text{Var}(LHS(i)) \)
      4. \( \text{RHS}(i) \rightarrow \text{true} \)
      5. changed = true
      6. while changed
         7. changed = false
         8. for all implications i
            9. if \( LHS(i) \) = true and \( \text{RHS}(i) \) = true
               10. \( \text{RHS}(i) \) = true
      11. changed = true
12. for all negative clauses c
13. if c = false
14. return true
15. return false
```
Correctness of greedy solution

If our algorithm does not return an assignment, does an assignment exist?

Our algorithm is "stingy". It only sets those variables that have to be true. All others remain false.

Running time?

\[ O(nm) \]

\( n = \text{number of variables} \)
\( m = \text{number of formulas} \)
Data compression

Given a file containing some data of a fixed alphabet $\Sigma$ (e.g. A, B, C, D), we would like to pick a binary character code that minimizes the number of bits required to represent the data.

A C A D A A D B …

0010100100100 …

minimize the size of the encoded file

Simplifying assumption: frequency only

Assume that we only have character frequency information for a file

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>70</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>20</td>
</tr>
<tr>
<td>D</td>
<td>37</td>
</tr>
</tbody>
</table>

Fixed length code

Use $\text{ceil}(\log_2|\Sigma|)$ bits for each character

A =
B =
C =
D =
### Fixed length code

Use $\lceil \log_2 |\Sigma| \rceil$ bits for each character

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>70</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>20</td>
</tr>
<tr>
<td>D</td>
<td>37</td>
</tr>
</tbody>
</table>

A = 00  
B = 01  
C = 10  
D = 11  

$2 \times 70 + 2 \times 3 + 2 \times 20 + 2 \times 37 = 260$ bits

How many bits to encode the file?

Can we do better?

### Variable length code

What about:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>70</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>20</td>
</tr>
<tr>
<td>D</td>
<td>37</td>
</tr>
</tbody>
</table>

A = 0  
B = 01  
C = 10  
D = 1  

$1 \times 70 + 2 \times 3 + 2 \times 20 + 1 \times 37 = 153$ bits

How many bits to encode the file?

### Decoding a file

A = 0  
B = 01  
C = 10  
D = 1  

010100011010

What characters does this sequence represent?
Decoding a file

A = 0
B = 01
C = 10
D = 1

What characters does this sequence represent?

Variable length code

What about:

A = 0
B = 100
C = 101
D = 11

Variable length code

What about:

A = 0
B = 100
C = 101
D = 11

Symbol | Frequency
-------|--------
A       | 70     
B       | 3      
C       | 20     
D       | 37     

Is it decodeable?

Variable length code

What about:

A = 0
B = 100
C = 101
D = 11

Symbol | Frequency
-------|--------
A       | 70     
B       | 3      
C       | 20     
D       | 37     

How many bits to encode the file?

\[ 1 \times 70 + 3 \times 3 + 3 \times 20 + 2 \times 37 = 213 \text{ bits} \]

(18% reduction)

Prefix codes

A prefix code is a set of codes where no codeword is a prefix of any other codeword

A = 0
B = 01
C = 10
D = 1

A = 0
B = 100
C = 101
D = 11
Prefix tree
We can encode a prefix code using a full binary tree where each leaf represents an encoding of a symbol.

A = 0
B = 100
C = 101
D = 11

Decoding using a prefix tree
To decode, we traverse the graph until a leaf node is reached and output the symbol.

A = 0
B = 100
C = 101
D = 11

Decoding using a prefix tree
Traverse the graph until a leaf node is reached and output the symbol.

100111010100

Decoding using a prefix tree
Traverse the graph until a leaf node is reached and output the symbol.

100111010100
B
Decoding using a prefix tree
Traverse the graph until a leaf node is reached and output the symbol

1000111010100
B A D C

45

46

Decoding using a prefix tree
Traverse the graph until a leaf node is reached and output the symbol

1000111010100
B A D

47

48
Decoding using a prefix tree
Traverse the graph until a leaf node is reached and output the symbol

Determining the cost of a file

\[
\text{cost}(T) = \sum_{i=1}^{n} f_i \cdot \text{depth}(i)
\]

Determining the cost of a file

What if we label the internal nodes with the sum of the children?
Determining the cost of a file

### Symbol Frequency

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>70</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>20</td>
</tr>
<tr>
<td>D</td>
<td>37</td>
</tr>
</tbody>
</table>

Cost is equal to the sum of the internal nodes (excluding the root) and the leaf nodes.

As we move down the tree, one bit gets read for every nonroot node:

- 70 times we see a 0 by itself
- 60 times we see a prefix that starts with a 1 of those, 37 times we see an additional 1
- the remaining 23 times we see an additional 0 of these, 20 times we see a last 1 and 3 times a last 0

A greedy algorithm?

Given file frequencies, can we come up with a prefix-free encoding (i.e. build a prefix tree) that minimizes the number of bits?

```
HUFFMAN(F)
1  Q = MAKEHEAP(F)
2  for i = 1 to |Q| - 1
3    allocate a new node z
4    left[z] = x = EXTRACTMIN(Q)
5    right[z] = y = EXTRACTMIN(Q)
6    f[z] = f[x] + f[y]
7    INSERT(z, Q)
8  return EXTRACTMIN(Q)
```
Huffman Tree

1. \( Q \leftarrow \text{MakeHeap}(F) \)
2. for \( i \leftarrow 1 \) to \(|Q| - 1\)
   3. allocate a new node \( z \)
   4. let \( f[z] \leftarrow x \leftarrow \text{ExtractMin}(Q) \)
   5. \( f[z] \leftarrow f[x] + f[y] \)
   6. \( \text{HeapInsert}(Q, z) \)
   7. return \( \text{ExtractMin}(Q) \)

## Symbol Frequency

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>70</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>20</td>
</tr>
<tr>
<td>D</td>
<td>37</td>
</tr>
</tbody>
</table>

Heap

merge node

---


Huffman Tree

1. \( Q \leftarrow \text{MakeHeap}(F) \)
2. for \( i \leftarrow 1 \) to \(|Q| - 1\)
   3. allocate a new node \( z \)
   4. let \( f[z] \leftarrow x \leftarrow \text{ExtractMin}(Q) \)
   5. \( f[z] \leftarrow f[x] + f[y] \)
   6. \( \text{HeapInsert}(Q, z) \)
   7. return \( \text{ExtractMin}(Q) \)

## Symbol Frequency

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>70</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>20</td>
</tr>
<tr>
<td>D</td>
<td>37</td>
</tr>
</tbody>
</table>

Heap

merge node

---


Huffman Tree

1. \( Q \leftarrow \text{MakeHeap}(F) \)
2. for \( i \leftarrow 1 \) to \(|Q| - 1\)
   3. allocate a new node \( z \)
   4. let \( f[z] \leftarrow x \leftarrow \text{ExtractMin}(Q) \)
   5. \( f[z] \leftarrow f[x] + f[y] \)
   6. \( \text{HeapInsert}(Q, z) \)
   7. return \( \text{ExtractMin}(Q) \)

## Symbol Frequency

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>70</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>20</td>
</tr>
<tr>
<td>D</td>
<td>37</td>
</tr>
</tbody>
</table>

Heap
The algorithm selects the symbols with the two smallest frequencies first (call them $f_1$ and $f_2$).
Is it correct?
The algorithm selects the symbols with the two smallest frequencies first (call them $f_1$ and $f_2$).

Consider a tree that did not do this (proof by contradiction):

Is it optimal?

Consider a tree that did not do this:

- Frequencies don’t change
- Cost will decrease since $f_1 < f_i$

contradiction

Is it correct?
The algorithm selects the symbols with the two smallest frequencies first (call them $f_1$ and $f_2$).

Runtime?

```
HUFFMAN(F)
1  Q ← MakeHeap(F)
2  for i = 1 to |Q| - 1
3    allocate a new node z
4    left[z] ← ExtractMin(Q)
5    right[z] ← ExtractMin(Q)
6    f[z] ← f[z] + f[i]
7    Insert(Q, z)
8  return ExtractMin(Q)
```

1 call to MakeHeap
2(n-1) calls ExtractMin
n-1 calls Insert

$O(n \log n)$

Symbol | Frequency
---|---
A | 5
B | 20
C | 10
D | 13
E | 9

What is the tree?
What is the encoding?
How many bits to encode the file?
Non-optimal greedy algorithms

All the greedy algorithms we’ve looked at so far give the optimal answer.

Some of the most common greedy algorithms generate good, but non-optimal solutions:
- set cover
- clustering
- hill-climbing
- relaxation

Knapsack problems: Greedy or not?

0-1 Knapsack – A thief robbing a store finds n items worth $v_1, v_2, \ldots, v_n$ dollars and weight $w_1, w_2, \ldots, w_n$ pounds, where $v_i$ and $w_i$ are integers. The thief can carry at most $W$ pounds in the knapsack. Which items should the thief take if he wants to maximize value.

Fractional knapsack problem – Same as above, but the thief happens to be at the bulk section of the store and can carry fractional portions of the items. For example, the thief could take 20% of item i for a weight of 0.2$w_i$ and a value of 0.2$v_i$.
I love proving things and looking at the Math behind the concepts from CS62.

the group assignments

Honestly I just really like the little comics at the start of every homework.

lectures are wayyy too fast, barely enough time to process things so it feels pointless to take notes; current course content is comprehensive and makes sense but it feels disorganized, like different content stitched together sort of so…

Having more examples, or going through the slides a bit slower.

The homeworks are a lot of work and the mentors are super helpful but someone's even they don't have the solutions and that wastes hours of our time. I think homeworks can have more straight forward problems that show we understand things rather than problems that we always have to scavenge the internet and bug mentors for understandings.
Course feedback

During Class, could we have some more exercises along with the lecture contents?

Checkpoint 1

Mean/Median: 17.5 (83%)