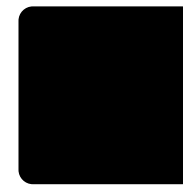
CS3383 Unit 2: Greedy. Lecture 0. Huffman Trees

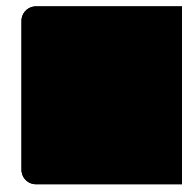
David Bremner

January 28, 2024



Greedy

- Huffman Trees
- Greedy Huffman Algorithm



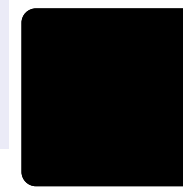
Lecture prerequisites

From Data Structures

- heaps
- priority queues

Background reading

- DPV 5.2
- `http://jeffe.cs.illinois.edu/teaching/algorithms/book/04-greedy.pdf`
- Huffman Coding is covered in §4.4



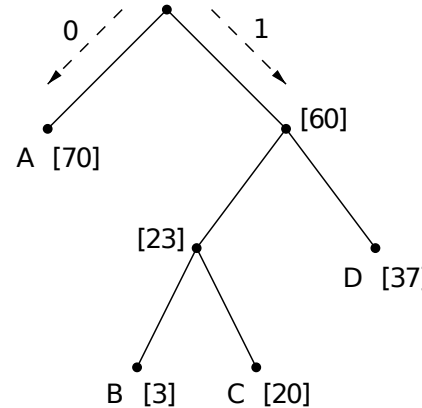
Prefix codes

Symbol	Freq	Codeword (bad)	Codeword
A	70	0	0
B	3	001	100
C	20	01	101
D	37	11	11

- variable length symbols
- avoiding ambiguous bitstreams: what is 001?
- no symbol should be a prefix of another.

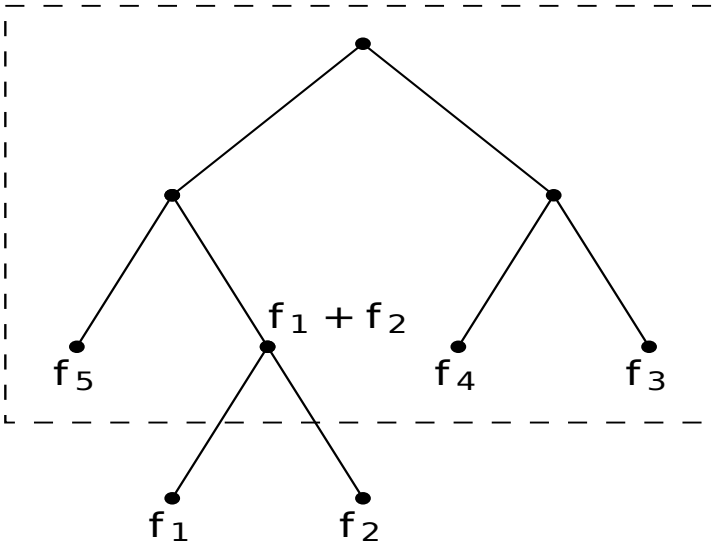
Huffman coding

Symbol	Codeword
A	0
B	100
C	101
D	11



(Avg cost) $$\text{cost}(T) = \sum_{i=1}^{n} f_i \text{depth}_i$$

Lightest leaves are deepest



- proof by swapping

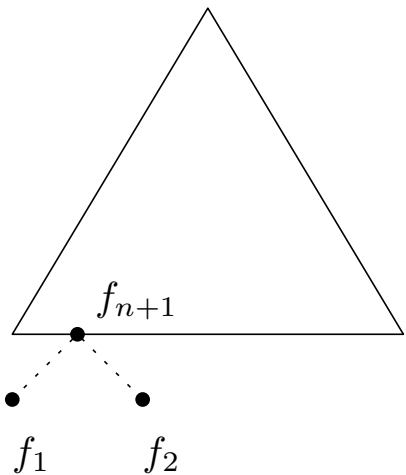
Huffman Algorithm

```
def huffman(f):
  tree=[]; H=[]; n = len(f)
  for i in range(0,n):
      heappush(H,(f[i],i))
      tree.append((f[i],None,None))
  for k in range(n,2*n-1):
      (f1,index1) = heappop(H)
      (f2,index2) = heappop(H)
      f3 = f1 + f2
      heappush(H,(f3,k))
      tree.append((f3,index1,index2))
  return tree
```

Huffman Algorithm

```
def huffman(f):
  tree=[]; H=[]; n = len(f)
  for i in range(0,n):
      heappush(H,(f[i],i))
      tree.append((f[i],None,None))
  for k in range(n,2*n-1):
      (f1,index1) = heappop(H)
      (f2,index2) = heappop(H)
      f3 = f1 + f2
      heappush(H,(f3,k))
      tree.append((f3,index1,index2))
  return tree
```

Correctness of Huffman



- By induction on the number of letters, if we remove f_1, and f_2, we have an optimal tree
- If the result of putting them back is not optimal, contradiction.