CSC373

Week 3

Huffman Code

- Variable
  - Length encoding
  - Better approach because every letter has a different frequency of occurrence

Goal: Minimize $ABL(\gamma) = \sum_{x \in S} f(x) \cdot |\gamma(x)|$

S $-$ alphabet

$f(x) =$ frequency of occurence of x

$\gamma(x) =$ encoding of x

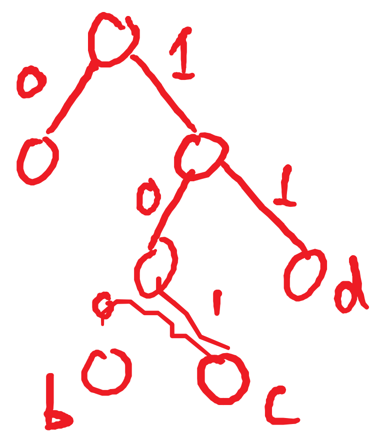
Biggest problem with variable length encoding is ambiguity

Solution: Prefix - code

How to generate prefix codes?

- Prefix-Code ↔ Full binary tree

a → 0
b → 100
c → 101
d → 11



Notes

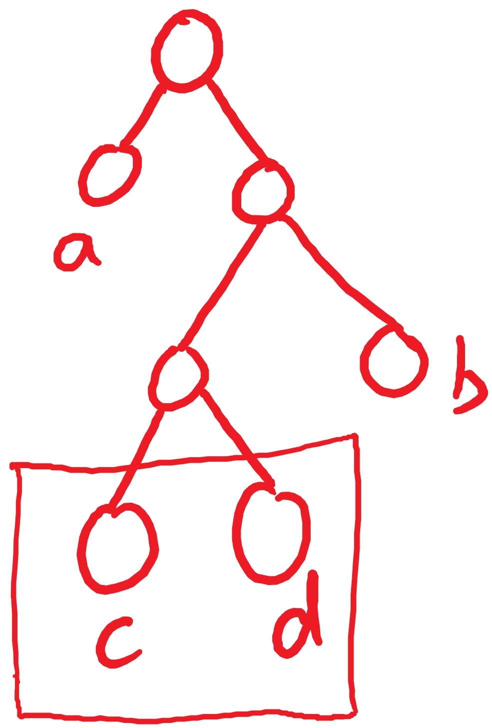
$S = \{a, b, c, d\}$

$f(a) = 0.4$

$f(b) = 0.3$

$f(c) = 0.2$

$f(d) = 0.1$

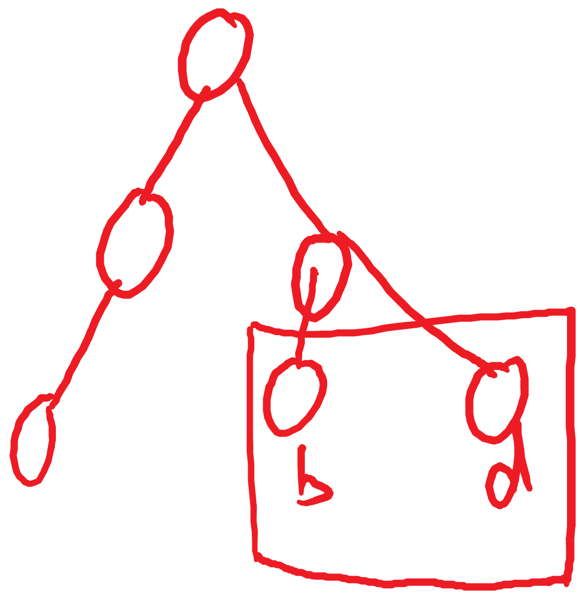


Notes

$$ABL(\gamma) = f(a)\cdot 1 + f(b) \quad f(c)\cdot 3 + f(d)\cdot 3$$

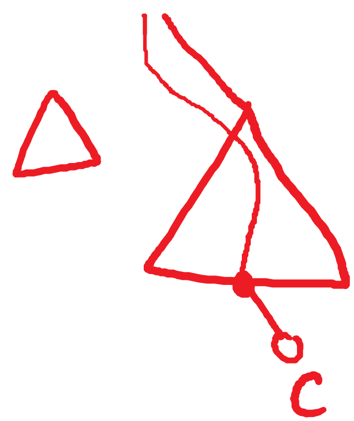
$$ABL(\gamma') = f(a)\cdot 3 + f(b) + f(c)\cdot 1 + f(d)\cdot 3$$

$$\begin{aligned} ABL(\gamma') - ABL(\gamma) &= f(a)(3-1) + f(c)\cdot(1-3) \\ &= 0.4 \times 2 + 0.2 \times (-2) \\ &= 0.8 - 0.4 \\ &= 0.4 > 0 \end{aligned}$$



$f(c) = 0.5$

$f(b) = 0.6$



Claim: Let x and y be the two least frequent letters. Then there is same optimal tree where x and y are siblings.

- Claim: Let x and y be two least frequent letters. Then there is an optimal tree Θ where x and y are siblings

- Proof: Let T be an optimal tree for (s, f)

  Let $\gamma = \text{prefix}-\text{code corresponding to } T$

  Let x be the least, and y be the second least

  If x and y are siblings, then done.

  so x and y are not siblings

  Assume, $|\gamma(x)| \geq |\gamma(y)|$

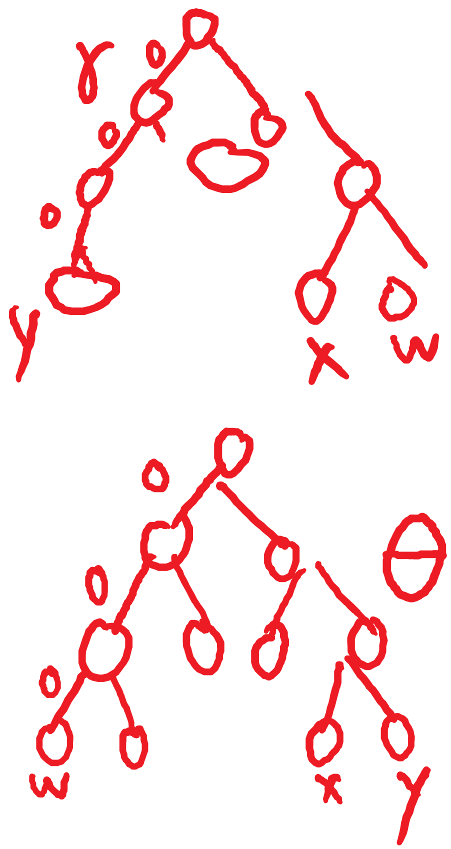
  Let ω be the sibling of x

  Interchange ω and y and get a new tree Θ

$$ABL(\Theta) - ABL(T) = f(w) \cdot |\gamma(y)| + f(y) \cdot |\gamma(w)| - f(w) \cdot |\gamma(w)| - f(y) \cdot |\gamma(y)|$$

$$= \underbrace{\big(f(w) - f(y)\big)}_{\geq 0}(|\gamma(y)| - |\gamma(w)|)$$

Notes



Proof (continued)

So, $|\gamma(y)| - |\gamma(w)| = |\gamma(y)| - |\gamma(x)| \leq 0$

So, $ABL(\Theta) - ABL(T) \leq 0$

Since T is optimal, $ABL(\Theta) - ABL(T) \nless 0$

$$\Rightarrow ABL(\Theta) - ABL(T) = 0$$

$$\Rightarrow ABL(\Theta) = ABL(T)$$

$$\Rightarrow \Theta \text{ is optimal}$$

Huffman Coding

If S has two letters then:

Encode one with 0 and the other with 1

Else:

Let y^* and z^* be the two least frequent letters

Form a new alphabet

$S' = S \cup \{w\}/\{y^*, z^*\}$

with $f(w) = f(\mathrm{y}^*) + \mathrm{f}(\mathrm{z}^*)$

"$Recursively\ construct\ a\ prefiz\ code\ \gamma'\ for\ S'\ with\ tree\ T'$

Define a prefix code for S as follows:

Start with T

Take the leaf labeled w in T' and add 2 children below if labeled y^* and z^*

End If

Notes

$S = \{a, b, c, d\}$

$S' = \{a, b, w\}$

$f(a) = 0.4$

$f(b) = 0.3$

$f(c) = 0.2$

$f(d) = 0.1$

$f(w) = f(c) + f(d) = 0$

Notes

$S = \{a, b, c, d, e\}$

$f(a) = f(b) = f(c) = f(d) = f(e) = 0.2$

$S' = \{a, b, c, w\}$

$f(w) = 0.4$

$S'' = \{z, c, w)$

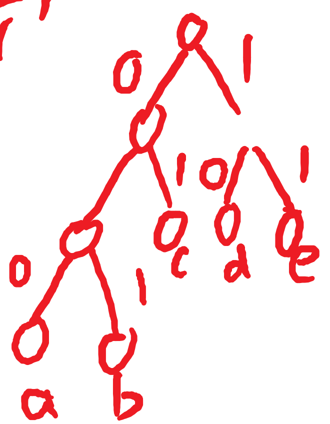
$f(z = 0.4)$

Notes

$S'' = \{y, w\}$

$f(y) = 0.6$

T'



$a = 000$

$b = 001$

$c = 01$

$d = 10$

$e = 11$

Proof why Huffman Coding works

- Proof by induction on n

Base Case:

$n = 2$: $S = \{a, b\}$

$\gamma(a) = 0 \quad \gamma(b) = 1$

Inductive Hypothesis: Assume Huffman coding gives an optimal tree for any alphabeted of size n

Inductive Step: Show it works for alphabet of size $n + 1$

$|S| = n + 1$

Let T be Huffman tree and let Θ be an optimal tree for S

Let y^* and z^* be the two least frequent letters

Replace y^* and z^* by w

$S' = S \cup \{w\}/\{y^*, z^*\}$

$f(w) = f(y^*) + f(z^*)$

$|S'| = n$

by inductive hypothesis, Huffman coding gives an optimal tree T' for S'

$$ABL(T) = \sum_{x \in S} f(x) \cdot |\gamma(x)|$$

$$= \sum_{x \in S \setminus \{y^*, z^*\}} f(x)|\gamma(x)| + f(y^*)|\gamma(y^*)| + f(z^*)|\gamma(z^*)|$$

$$= \sum f(x)|\gamma(x)| + \underbrace{(f(y^*) + f(z^*))}_{= f(w)}(|\gamma(w)| + 1)$$

$$= \sum_{x \in S \setminus \{y^*, z^*\}} f(x) \cdot |\gamma(x)| + f(w)(|\gamma(w)| + 1)$$

$$= \sum_{x \in S \setminus \{y^*, z^*\}} f(x) \cdot |\gamma(x)| + f(w)$$

$$= ABL(T') + f(w)$$

$$= ABL(T') + f(y^*) + f(z^*)$$

Proof (continued)

Assume for a contradiction that T is not optimal

Let O be an optimal tree for S such that y^* and z^* are sibling in O (by claim proved before)

If you remove y^* and z^* from O and replace them by w with $f(w) = f(y^*) + f(z^*)$

Then this new tree O′ must be optimal for S'

So $ABL(\mathrm{O}') = ABL(T')$

$\Rightarrow ABL(\mathrm{O}') + f(y^*) + f(z^*) = ABL(T') + f(y^*) + f(z^*) = ABL(T)$

ABL(O)

Weighted Interval Scheduling

- Single Resource
- Input

$$R = \{ 1 \quad 2 \quad n \}$$

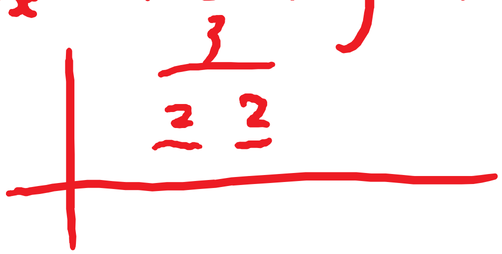
$$1 \to (\quad) \qquad 2 \to (s_2, f_2) \to v_2 \qquad n \to (s_n, f_n) \to v_n$$

- Goal: Maximize the total value of jobs scheduled

$$\max \sum_{i \in S} v_i$$

Use Greedy

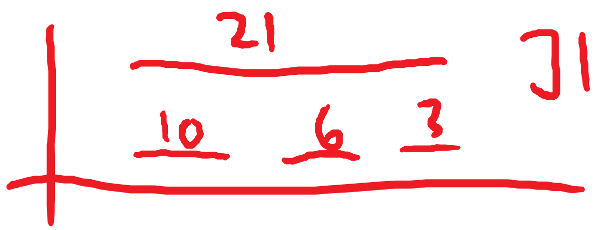
Choose the higest value



II. Choose by the maximum $\frac{\text{value}}{\text{length}}$ ratio

Use Greedy (continued)

III choose the earliest ending job



$J1 = \frac{21}{3} = 7$

IV $\frac{\text{Value}}{\text{\# conflict}}$

III. Dynamic Programming

- Decompose a problem into several subproblems and combine solutions

Goal: Schedule S

$S \subseteq R$

Question: Does $r_n \in S$?

Answer: It may or may not

Case I: $r_n \in S$

Case II: $r_n \notin S$

I. Sort all the jobs by their finish time

II. Define function P

$P(i) =$ the largest $j < i$ such that j does not overlap with i

In other words, all jobs between $P(\mathrm{i})$ and i overlap with i

Case I: $r_n \in S$

Any job between $r_{P(n)}$ and r_n is not in S

Next job we can possibly schedule is $r_{P(n)}$

Case II: $r_n \notin S$

Next job we can possibly schedule is r_{n-1}

- Subproblems look like solve the same problem on $\{r_1, \ldots, r_j\}$

$1 \leq j \leq n$

O_j: optimal solution to subproblems $\{r_1, \ldots, r_n\}$

OPT_j: value of that optimal solution

$= \sum_{i \in S_{\theta_j}} v_i$

If $r_j \in S_{\mathrm{O}_j}$: $OP \quad_j = OP \quad_{P(i)} + v_j$

If $r_j \notin S_{O_j}$: $OPT_j = OPT_{j-1}$

$OP \quad_j = \max\{OP \quad_{j-1}, OPT_{P(i)} + v_j\}$

If $OPT_{j-1} > OPT_{P(j)} + v_j$

then $r_j \notin S_{\theta_j}$

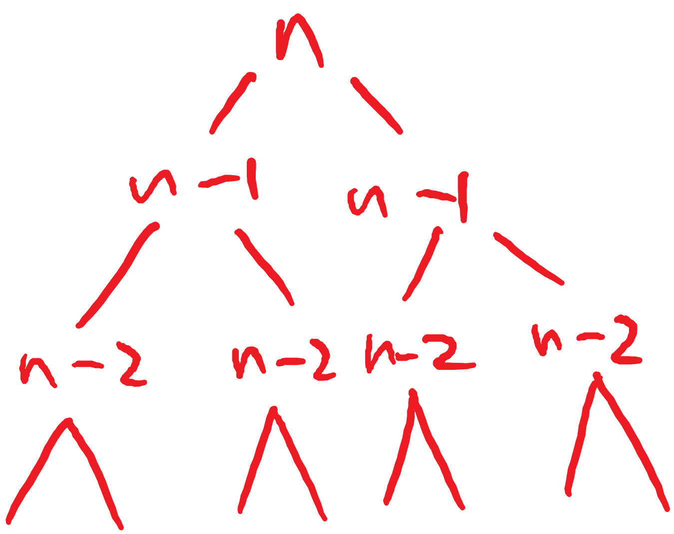
Else

$r_j \in S_{\mathrm{O}_j}$

```
Compute_WIS(i, P)
if j == 0:
        return 0
Else
        s1 = Compute_WIS((P(i), P) + v_j
        s2 = Compute_WIS(j-1, P)
        return max(S_1, S_2)
```

$P(i) = i - 1$

Complexity: $O(2^n)$



Solution: Save the results of any computation of use that as a cache

```
compute_WIS(j, P, M)
  if M[j] is defined
        return M[j]
  else:
        if j == 0 then
              q = 0
        else
              q = max(S_1, S_2)
              (S_1, S_2 as before)
        M[j] = q
        return q
```

Complexity: $O(n)$ ✓

each pass writes one entry of M and there are only n entries to write

OPT_n= compute_WIS(n, p, M)

```
Compute_WIS_Schedule(j, P, M)
        if M[j] == M[P[j]] + v_j
                print v_j
                q = P[i]
        else
                q = j – 1
        return Compute_WIS_Schedule(q, P, M)
```

Key concepts

1) Optimal substructure
2) Recursion relation between the subproblem and the main problem
3) Implementing recursion directly will most likely have an exponential growth

Memorization

- Save the results. Complexity drops downs from exponential to (pseudo) polynomial

Divide-and-conquer

$O(n^2)$ -> $O(n \ln n)$
Greedy: $O(n^2)$ -> $O(\log_2 7)$

```
Fib(n):
        if n == 1 or n == 2
                return 1
        return Fib(n-1) + Fib(n-2)
```

```
Fib(10)
         10
        /   \
      9       8
     / \     / \
    8   7   7   6
   / \ / \
  7  6 6  5
```

```
Fib(n, M):
    if n == 1 or n == 2
        return 1
    if M[n] is defined
        return M[n]
    M[n] = Fib(n-1, M) + Fib(n-2, M)
    return M[n]
```

O(n)