day 11 2/16/98 "Shouldn't ask questions that you don't want to know the answer."

1 Dynamic programming

1.1 Optimal Binary Search Trees

1.1.1 given

- keys $\{a_1, ..., a_n\} = S$
- $a_i < a_{i+1}$
- $p_1, ..., p_n$ where $Prob(search(a_i, S)) = p_i$
- $q_i = Prob(Search(a, S) \wedge a_i < a < a_{i+1})$
- q_0 $=Prob(Search(a, S) \wedge a < a_0)$
- q_n $=Prob(Search(a, S) \wedge a_n < a)$

1.1.2 analysis

The cost of a search for a_i is equal to $Depth(a_i) + 1$ if the search is succesful, otherwise the cost is $Depth(newleaf)$.

Expected cost $= \sum_i p_i(D(a_i) + 1) + \sum_i q_i D(i)$

1.1.3 definitions

T_{ij} = min cost tree $\{a_{i+1}, ..., a_j\}$, $1 \leq i < j \leq n$

- $T_{ii+1} = a_i$
- $C_{ij} = cost(T_{ij})$
- $V_{ij} = root$
- $W_{ij} = \sum_{k=i}^{j} q_k + \sum_{k=i+1}^{j} p_k$

Suppose we want to compute T_{ij} and all subintervals have been calculated. Pick a key $a_k : i+1 \leq k \leq j$. Lookup the cost of T_{ik-1} and T_{kj}.

$$Cost(T) = Cost(T_{ik-1}) + w_{ik-1} + Cost(T_{kj}) + w_{kj} + p_k$$

$$= Cost(T_{ik-1}) + Cost(T_{kj}) + w_{ij}$$

Then,

$$c_{ij} = min_{i+1<k<j}\{C_{ik+1} + C_{kj}\} + w_{ij}$$

This is formula involves exponential computation without memoization. With memoization it is $O(n^3)$ because there are $O(n^2)$ c_{ij} and each c_{ij} costs $O(n)$ to calculate.

1.2 Huffman codes

1.2.1 givens

- alphabet $\sum = \{0, ..., \sigma - 1\} = \{0, 1\}$ for us
- code words $w = a_i ... a_l \in \sum^l$
- set of words = code = $\{w_1, ..., w_k\}$

1.2.2 Unique decipherability

The code $\{10, 0, 01\}$ is not uniquely decipherable. 010 can be parsed as: 0(10) and as (01)0. An algorithm which determines unique decipherability would be worthwhile.

Define: W codeword w=ps, p = prefix and s = suffix

Define: $c_1 ... c_k$ = code words

Define: non-empty word t is tail if $\exists$ messages $c_1 ... c_m$ and $c_1^{'} ... c_n^{'}$ s.t. $c_1 \neq c_1^{'}$, t is a suffix of $c_n^{'}$, $c_1 ... c_m t = c_1^{'} ... c_n^{'}$

Lemma: A code, C is uniquely decipherable iff no tail is a code word.

1.2.3 Algorithm

$tails = \emptyset$

Generate-tails

1. $\forall c_i \& c_j \;\; i \neq j$.

- if $c_i = c_j$ return "not UD"
- if $\exists s$ s.t. $c_i s = c_j$ or $c_i = c_j s$ then $tail = tails \cup \{s\}$

1. $\forall c, \forall t \in tails$

- if $t = c$ return "not UD"
- $\exists s \; ts = c$ or $cs = t$ then $tails = tails \cup \{s\}$

1. return "C is UD"

1.2.4 Analysis

- $l =$ number of code words
- $\#Tails \leq nl$
- $case1 : n^2l$
- $case2 : \#tails = n^2l * l = n^2l^2$

So $O(n^2l^2)$

1.2.5 Algorithm correctness

Lemma 1: Algorithm generates only tails. The tails in case 1 are obvious. Those generated in case 2 are of the form $ts = c$ or $cs = t$ where a new tail is $\{s\}$. s is indeed a tail here.

Lemma 2: Algorithm generates all tails Proof by induction.

base case: let t, a tail $M(t)$ =minimal n+m s.t. $c_1...c_mt = c_1^{'}...c_n^{'}$

when $M(t) = n + m = 2$, $c_1t = c_1^{'}$ which is covered by the first step in the algorithm.

Inductive case: $M(t) > 2$ with $c_1...c_mt = c_1^{'}...c_n^{'}$. Assume $pt = c_n^{'}$

- $p = c_m$ then not UD
- p suffix of c_m, then $c_1...c_{m-1} = c_1^{'}...c_{n-1}^{'}p$ because p is a tail.
- c_m is a suffix of p, then $t' = c_mt$ is a tail. so $c_1...c_{m-1}t' = c_1^{'}...c_n^{'}$