Greedy Algorithms - II

Huffman Codes & Data Compression

Huffman Codes & Data Compression

Variable length encoding schemes

Huffman Codes & Data Compression

Prefix Code :

Prefix Code for a set S of letters is a function $\gamma : S \rightarrow \{0,1\}^n$ such that for all $x, y \in S$, $x \neq y$, $\gamma(x)$ is not a prefix of $\gamma(y)$.

Huffman Codes & Data Compression

Optimal Prefix Code :

For a letter $x \in S$, let f_x represent its frequency, the fraction of the letters in the text that is equal to x.

$$\sum_{x \in S} f_x = 1$$

Huffman Codes & Data Compression

Optimal Prefix Code :

For a letter $x \in S$, let f_x represent its frequency, the fraction of the letters in the text that is equal to x.

$$\sum_{x \in S} f_x = 1$$

encoding length = $\sum_{x \in S} n f_x \, |\gamma(x)|$

Average number of bits per letter

$$\text{ABL}(\gamma) = \sum_{x \in S} f(x) \, |\gamma(x)|$$

An optimal Prefix Code is one that minimises ABL

Huffman Codes & Data Compression

Optimal Prefix Code and binary trees :

Huffman Codes & Data Compression

Optimal Prefix Code and binary trees :

Optimal Prefix Code ==>

binary tree that minimises $\sum_{x \in S} f(x) depth(x)$

Huffman Codes & Data Compression

The binary tree corresponding to the optimal prefix code is full.

Huffman Codes & Data Compression

Let $u, v \in S$.

Let depth(u) < depth(v). Then $f_u \geq f_v$.

Huffman Codes & Data Compression

There is an optimal prefix code, with corresponding tree T^*, in which the two lowest frequency letters are assigned to leaves that are siblings in T^*.

Huffman_Code(S)

```
If |S|=2
   encode one using 0 and other using 1
else
   Let y*, z* - lowest frequency letters
   S' = S \{y*,z*} U {w}
```

$$f_w = f_{y*} + f_{z*}$$

```
   T' = Huffman_Code(S')
   In T', take the leaf labeled with w and add two children
   labelled y*, z*
```

proof of optimality:

$$\mathrm{ABL}(T') = \mathrm{ABL}(T) + f_w$$

proof of optimality:

Running Time :

Scheduling to minimise lateness

- A single resource and a set of n requests to use the resource
- Each request has a deadline d_i and needs contiguous time interval of time t_i
- Assign non-overlapping intervals for requests
- A request i is assigned the interval $[s_i, f_i]$
- i is said to be late if $f_i > d$
- lateness = $f_i - d$
- Schedule all intervals such that the maximum lateness is minimised

Scheduling to minimise lateness

- Greedy choices

Scheduling to minimise lateness

- Earliest deadline first

Scheduling to minimise lateness

- Earliest deadline first

```
Order the jobs in order of their deadlines

f=s

For i=1 to n
```

$$s_i = f$$

$$f_i = f + t_i$$

$$f = f_i$$

Scheduling to minimise lateness

- Earliest deadline first

```
Order the jobs in order of their deadlines

f=s

For i=1 to n
```

$$s_i = f$$

$$f_i = f + t_i$$

$$f = f_i$$

Running Time?

Scheduling to minimise lateness

There is an optimal schedule with no idle time.

Scheduling to minimise lateness

Inversion in a schedule A′ : If a job i is scheduled before a job j with $d_j < d_i$

Scheduling to minimise lateness

Inversion in a schedule A′ : If a job i is scheduled before a job j with $d_j < d_i$

The greedy schedule has no inversions.

Scheduling to minimise lateness

Inversion in a schedule A′ : If a job i is scheduled before a job j with $d_j < d_i$

The greedy schedule has no inversions.

Proof by Exchange : We'll convert an optimal schedule into a schedule without inversions and not increasing the maximum lateness.