Design and Analysis of Algorithms CS218M

Greedy Algorithms

Paritosh Pandya

Indian Institute of Technology, Bombay

Autumn, 2022

- Build the solution by selecting elements (or making choices) one by one.
- A simple rule allows choice of element at each stage. Local optimality.
- Greedy choice property: The current selection cannot be removed (no backtracking/exploring alternative choices).
- The final solution must be optimal.

Sequence of locally optimalchoices gives globally optimal solution.

Examples: Picking 10 coins, Finding shortest path, Minimum Spanning Tree.

Interval Scheduling

KT chap. 4

Problem

Given Set of requests $1, \ldots, n$ with $s(i), f(i)$ giving start and end time (interval) of the request. To find maximal sized $A \subseteq \{1, \ldots, n\}$ which is compatible.

A pair of intervals is compatible if they do not overlap. A is compatible if every pair of distinct intervals in A is compatible.

Interval Scheduling

Problem

Given Set of requests $1, \ldots, n$ with $s(i), f(i)$ giving start and end time (interval) of the request. To find maximal sized set $A \subseteq \{1, \ldots, n\}$ which is compatible.

A pair of intervals is compatible if they do not overlap. A is compatible if every pair of distinct intervals in A is compatible.

Strategy

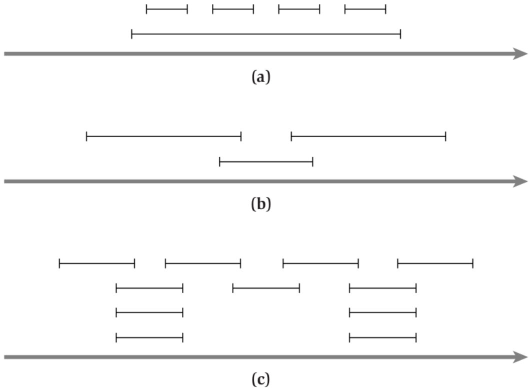
Choose sequence of intervals $i_1, \ldots, i_k$ using a greedy rule. After each choice remove incompatible intervals with the last choice.

Algorithm Template

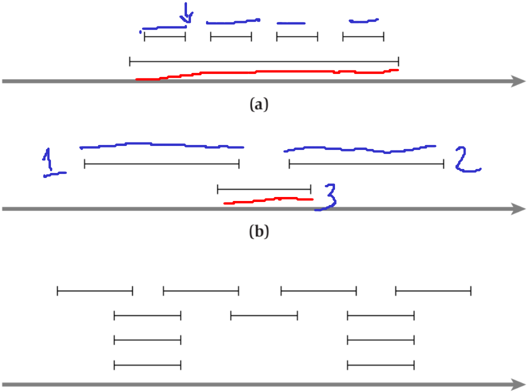
```
Initially let R be the set of all requests, and let A be empty
While R is not yet empty
  Choose a request i ∈ R that has the smallest finishing time
  Add request i to A
  Delete all requests from R that are not compatible with request i
EndWhile
Return the set A as the set of accepted requests
```

Let the sequence of intervals added to A be $i_1, \ldots, i_k$.

Some Examples



Some Examples



(a)

(b)

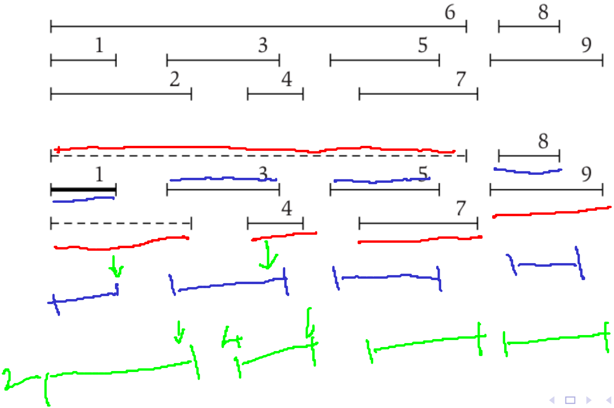
(c)

Some Greedy Rules

- Earliest start time.
- Shortest interval first
- Interval which overlaps least number of intervals.

Solution

Correct Greedy rule Choose and add to A interval i from R with smallest finishing time.



Claim: A is Compatible

Correctness

Claim: A is Compatible

Claim: $A = i_1, \ldots, i_k$ is optimal.

Claim: A is Compatible

Claim: $A = i_1, \ldots, i_k$ is optimal.

Let $O = j_1, \ldots, j_m$ be optimal. Then, $k \leq m$.
Claim: $\forall 1 \leq r \leq k : \quad f(i_r) \leq f(j_r)$

Correctness

Claim: A is Compatible

Claim: $A = i_1, \ldots, i_k$ is optimal.

Let $O = j_1, \ldots, j_m$ be optimal. Then, $k \leq m$.

Claim: $\forall 1 \leq r \leq k : \quad f(i_r) \leq f(j_r)$

- (Base step) $r = 1$. We choose i_1 with shortest $f(i)$. Hence, $f(i_1) \leq f(j_1)$.

Correctness

Claim: A is Compatible

Claim: $A = i_1, \ldots, i_k$ is optimal.

Let $O = j_1, \ldots, j_m$ be optimal. Then, $k \leq m$.
Claim: $\forall 1 \leq r \leq k : \quad f(i_r) \leq f(j_r)$

- (Base step) $r = 1$. We choose i_1 with shortest $f(i)$. Hence, $f(i_1) \leq f(j_1)$.
- (Induction) By hyp, $f(i_{r-1}) \leq f(j_{r-1})$.
  Also, $f(j_{r-1}) \leq s(j_r)$ (why?). Hence, $f(i_{r-1}) \leq s(j_r)$.

Correctness

Claim: A is Compatible

Claim: $A = i_1, \ldots, i_k$ is optimal.

Let $O = j_1, \ldots, j_m$ be optimal. Then, $k \leq m$.
Claim: $\forall 1 \leq r \leq k : \quad f(i_r) \leq f(j_r)$

- (Base step) $r = 1$. We choose i_1 with shortest $f(i)$. Hence, $f(i_1) \leq f(j_1)$.
- (Induction) By hyp, $f(i_{r-1}) \leq f(j_{r-1})$.
  Also, $f(j_{r-1}) \leq s(j_r)$ (why?). Hence, $f(i_{r-1}) \leq s(j_r)$.
- Hence, while selecting i_r, interval $j_r \in R$.
  Hence, $f(i_r) \leq f(j_r)$.

Correctness

Claim: A is Compatible

Claim: $A = i_1, \ldots, i_k$ is optimal.

Let $O = j_1, \ldots, j_m$ be optimal. Then, $k \leq m$.

Claim: $\forall 1 \leq r \leq k: \quad f(i_r) \leq f(j_r)$

- (Base step) $r = 1$. We choose i_1 with shortest $f(i)$. Hence, $f(i_1) \leq f(j_1)$.
- (Induction) By hyp, $f(i_{r-1}) \leq f(j_{r-1})$.
  Also, $f(j_{r-1}) \leq s(j_r)$ (why?). Hence, $f(i_{r-1}) \leq s(j_r)$.
- Hence, while selecting i_r, interval $j_r \in R$.
  Hence, $f(i_r) \leq f(j_r)$.

If $k < m$ then $R(k+1) \neq \emptyset$ and more intervals will be added to A.

We do not explicitly maintain R.

We do not explicitely maintain R.

- Sort given intervals by $f(i)$ to give interval list indexed $1, 2, \ldots, n$. Time $O(n \cdot lg(n))$.

We do not explicitely maintain R.

- Sort given intervals by $f(i)$ to give interval list indexed $1, 2, \ldots, n$. Time $O(n \cdot lg(n))$.
- Construct $S[1 \ldots n]$ s.t. $S[i] = s(i)$ for all i. Time $O(n)$.

Implementation and Complexity

We do not explicitely maintain R.

- Sort given intervals by $f(i)$ to give interval list indexed $1, 2, \ldots, n$. Time $O(n \cdot lg(n))$.
- Construct $S[1 \ldots n]$ s.t. $S[i] = s(i)$ for all i. Time $O(n)$.
- Initialize: $i = 1$. Include i in A.
- loop

  Find first $j > i$ s.t. $S[j] \geq f(i)$.
  If found then Include j in A
  else exit loop.

Implementation and Complexity

We do not explicitely maintain R.

- Sort given intervals by $f(i)$ to give interval list indexed $1, 2, \ldots, n$. Time $O(n \cdot lg(n))$.
- Construct $S[1 \ldots n]$ s.t. $S[i] = s(i)$ for all i. Time $O(n)$.
- Initialize: $i = 1$. Include i in A.
- loop

    Find first $j > i$ s.t. $S[j] \geq f(i)$.
    If found then Include j in A
    else exit loop.
- Overall complexity $O(n \cdot lg(n))$

Problem

Given jobs $1, \ldots, n$ where job i has load t_i and deadline d_i, assign interval $I(i) = [S(i), F(i)]$ to job i s.t.

Minimum Lateness Scheduling

Problem

Given jobs $1, \ldots, n$ where job i has load t_i and deadline d_i, assign interval $I(i) = [S(i), F(i)]$ to job i s.t.

- $F(i) = S(i) + t_i$ (i.e. non-preemptive scheduling)
- $I(i)$ and $I(j)$ are non-overlapping if $i \neq j$

Minimum Lateness Scheduling

Problem

Given jobs $1, \ldots, n$ where job i has load t_i and deadline d_i, assign interval $I(i) = [S(i), F(i)]$ to job i s.t.

– $F(i) = S(i) + t_i$ (i.e. non-preemptive scheduling)

– $I(i)$ and $I(j)$ are non-overlapping if $i \neq j$

and the assignment has minimum lateness.

Lateness $l_i = if \ \ (F(i) > d_i) \ \ then \ \ F(i) - d_i \ \ else \ \ 0$.

Let $L \ = max_i \ \ l_i$.

Minimum Lateness Scheduling

Problem

Given jobs $1, \ldots, n$ where job i has load t_i and deadline d_i, assign interval $I(i) = [S(i), F(i)]$ to job i s.t.

– $F(i) = S(i) + t_i$ (i.e. non-preemptive scheduling)

– $I(i)$ and $I(j)$ are non-overlapping if $i \neq j$

and the assignment has minimum lateness.

Lateness $l_i = if \ \ (F(i) > d_i) \ \ then \ \ F(i) - d_i \ \ else \ \ 0$.

Let $L \ = max_i \ \ l_i$.

Claim

There exists an optimal assignment with no idle time.

Problem

Given jobs $1, \ldots, n$ where job i has load t_i and deadline d_i, assign interval $I(i) = [S(i), F(i)]$ to job i s.t.

– $F(i) = S(i) + t_i$ (i.e. non-preemptive scheduling)

– $I(i)$ and $I(j)$ are non-overlapping if $i \neq j$

and the assignment has minimum lateness.

Lateness $l_i = if \;\; (F(i) > d_i) \;\; then \;\; F(i) - d_i \;\; else \;\; 0$.

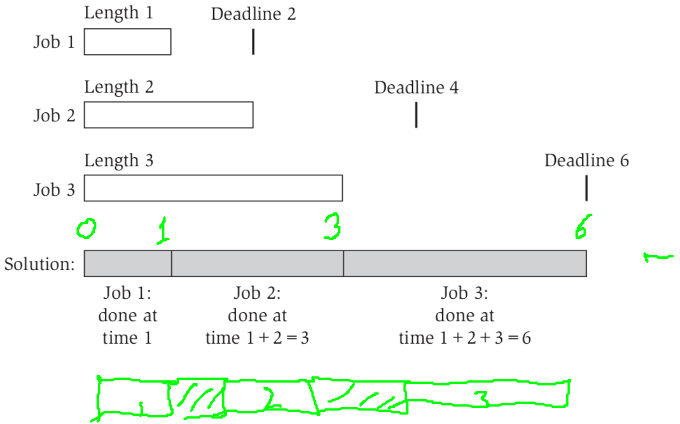
Let $L \; = max_i \;\; l_i$.

Claim

There exists an optimal assignment with no idle time.

Hence, we only need to order the jobs. A schedule $A = (A(1), \ldots, A(n))$ is a permutation of jobs $(1, \ldots, n)$.

Example


Length 1
Deadline 2
Job 1
Length 2
Deadline 4
Job 2
Length 3
Deadline 6
Job 3
0
1
3
6
Solution:
Job 1:
done at
time 1
Job 2:
done at
time 1 + 2 = 3
Job 3:
done at
time 1 + 2 + 3 = 6
1
2
3

(3 . 1, 2)

Inversions by Example

Index i	1	2	3	4	5	6	7
deadline d_i	1	6	7	2	4	8	3

2 7

- $(2,5)$ is an inversion.
- $(3,4)$ is an immediate inversion.
- If we exchange d_3, d_4 of immediate inversion $(3,4)$ the total number of inversions decreases by 1.

Given a schedule A, a pair (i, j) is an inversion if $(i < j) \wedge (d_i > d_j)$.

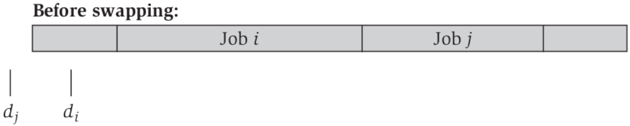
- Claim: If (i, j) is an inversion, there exists a k s.t. $i \leq k < j$ and $(k, k+1)$ is an inversion. Call it immediate inversion.
- claim: In any schedule A, if $(k, k+1)$ is an immediate inversion and we swap $A(k), A(k+1)$ by $A(k+1), A(k)$ to get A', the total number of inversions decreases by 1.

Exchange preserves optimality

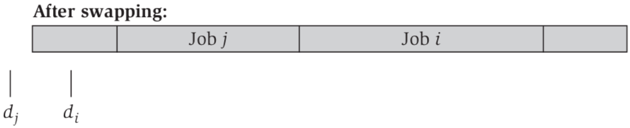
$j = i+1$

Theorem

If A is a schedule with lateness L and (i, j) is an immediate inversion, then A' obtained by exchanging jobs $A(i), A(j)$ has lateness L' with $L' \leq L$

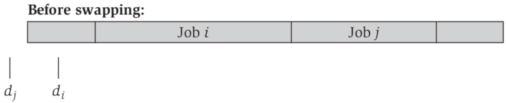


(a)

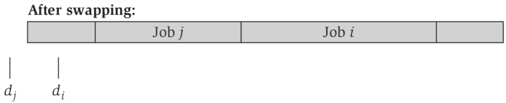


(b)

Exchange preserves optimality (2)



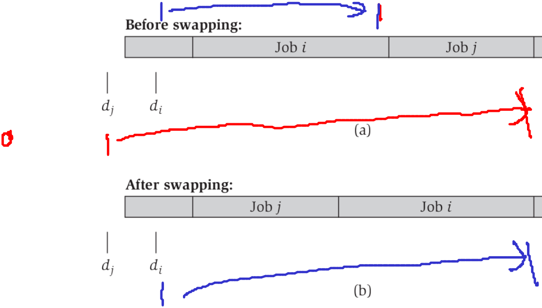
(a)



(b)

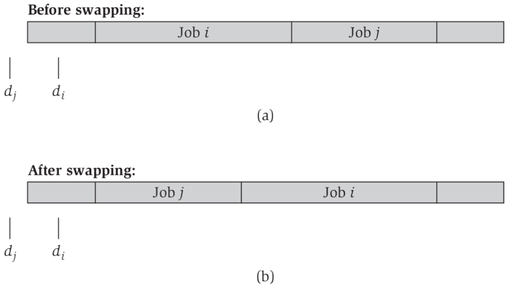
L_i, L_j

L'_i, L'_j



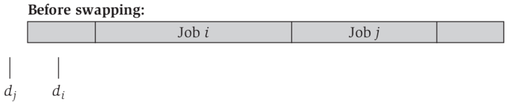
- Finishing time $F(r)$ for $r \neq i, j$ is unchanged. Hence $l'_r = l_r$.

Exchange preserves optimality (2)

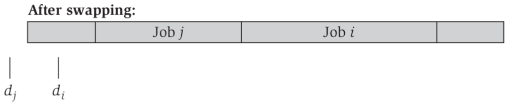


- Finishing time $F(r)$ for $r \neq i, j$ is unchanged. Hence $l'_r = l_r$.
- Finishing time $F(j)$ decreases. Hence $l'_j \leq l_j$.

Exchange preserves optimality (2)



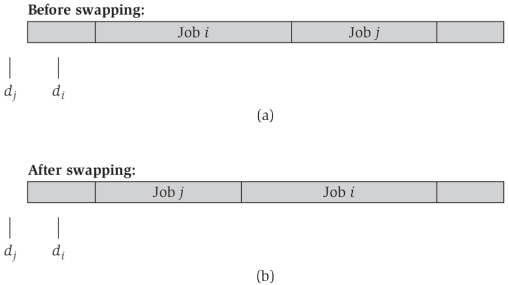
(a)



(b)

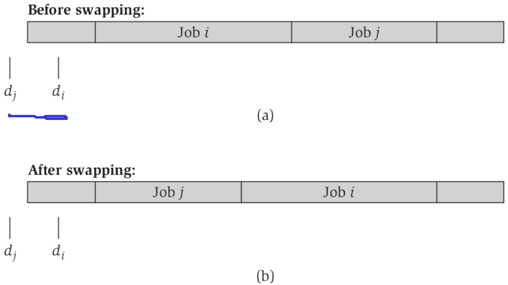
- Finishing time $F(r)$ for $r \neq i, j$ is unchanged. Hence $l'_r = l_r$.
- Finishing time $F(j)$ decreases. Hence $l'_j \leq l_j$.
- Finishing time of $F(i)$ increases. Lateness l'_i increases. However, $F'(i) = F(j)$.

Exchange preserves optimality (2)



- Finishing time $F(r)$ for $r \neq i, j$ is unchanged. Hence $l'_r = l_r$.
- Finishing time $F(j)$ decreases. Hence $l'_j \leq l_j$.
- Finishing time of $F(i)$ increases. Lateness l'_i increases. However, $F'(i) = F(j)$.
- We show that $l'_i \leq l_j$.

Exchange preserves optimality (2)



- Finishing time $F(r)$ for $r \neq i, j$ is unchanged. Hence $l'_r = l_r$.
- Finishing time $F(j)$ decreases. Hence $l'_j \leq l_j$.
- Finishing time of $F(i)$ increases. Lateness l'_i increases. However, $F'(i) = F(j)$.
- We show that $l'_i \leq l_j$.

$$l_j = F(j) - d_j = (d_i - d_j) + (F(j) - d_i)$$
$$\geq (F(j) - d_i)$$
$$= (F'(i) - d_i) = l'_i$$

Earliest Deadline First Schedule

Claim: There exists an inversion-free optimal schedule

- Let A be an arbitrary optimal schedule.
- Repeatedly remove one inversion at a time by exchanging any immediate inversion pair.
  This does not increase lateness. Hence, exchange preserves optimality.
- In the end we obtain optimal inversion free schedule.

Earliest Deadline First Schedule

Claim: There exists an inversion-free optimal schedule

- Let A be an arbitrary optimal schedule.
- Repeatedly remove one inversion at a time by exchanging any immediate inversion pair.
  This does not increase lateness. Hence, exchange preserves optimality.
- In the end we obtain optimal inversion free schedule.

Claim: All inversion-free schedules have same lateness

Proof: Homework!

Earliest Deadline First Schedule

Claim: There exists an inversion-free optimal schedule

- Let A be an arbitrary optimal schedule.
- Repeatedly remove one inversion at a time by exchanging any immediate inversion pair.
  This does not increase lateness. Hence, exchange preserves optimality.
- In the end we obtain optimal inversion free schedule.

Claim: All inversion-free schedules have same lateness

Proof: Homework!

EDF Schedulig Algorithm

Given a set of jobs, schedule them in the order of deadlines with earliest deadline first.
If two jobs have the same deadline, order them arbitrarily.

Claim: The resulting EDF schedule is optimal.

- Finite alphabet $S = \{a_1, \ldots, a_n\}$. Message $x_1 x_2 \ldots x_r$ is a finite sequence of symbols.
- Digital communication: Message encoded and communicated as a sequence of bits. It is decoded back on receipt.
- We encode each symbol x_i as bit sequence $\gamma(x_i)$.
- Fixed length encoding: $\lceil lg(n) \rceil$ bits are needed to encode a symbol from alphabet S of size n.
- E.g. ASCII characters into 7-bit encoding. Unicode characters into 16-bits.

Variable Length Code

Variable Length Code

Prefix Property

For any $x, y \in S,\ x \neq y$ we have
$\neg\ pref(\gamma(x), \gamma(y)) \quad \wedge \quad \neg\ pref(\gamma(y), \gamma(x))$

Prefix Property

For any $x, y \in S,\ x \neq y$ we have
$\neg\ pref(\gamma(x), \gamma(y)) \quad \wedge \quad \neg\ pref(\gamma(y), \gamma(x))$

Prefix property allows us to uniquely decode the encoded bit-string.

Prefix Code

$S = \{a, b, c, d, e\}$. The encoding γ_1 specified by

$$\gamma_1(a) = 11$$
$$\gamma_1(b) = 01$$
$$\gamma_1(c) = 001$$
$$\gamma_1(d) = 10$$
$$\gamma_1(e) = 000$$

Average Bits Per Letter

Let probability of letter x in message be f_x. Let γ be a prefix code. Then,

$$ABL(\gamma) \quad = \quad \Sigma_{x \in S} \quad f_x \cdot |\gamma(x)|$$

Prefix Code

$S = \{a, b, c, d, e\}$. The encoding γ_1 specified by

$$\gamma_1(a) = 11$$
$$\gamma_1(b) = 01$$
$$\gamma_1(c) = 001$$
$$\gamma_1(d) = 10$$
$$\gamma_1(e) = 000$$

Average Bits Per Letter

Let probability of letter x in message be f_x. Let γ be a prefix code. Then,

$$ABL(\gamma) = \Sigma_{x \in S} \; f_x \cdot |\gamma(x)|$$

$$f_a = .32, \quad f_b = .25, \quad f_c = .20, \quad f_d = .18, \quad f_e = .05.$$

$$.32 \cdot 2 + .25 \cdot 2 + .20 \cdot 3 + .18 \cdot 2 + .05 \cdot 3 = 2.25.$$

Optimal Prefix Code

$$\gamma_2(a) = 11$$
$$\gamma_2(b) = 10$$
$$\gamma_2(c) = 01$$
$$\gamma_2(d) = 001$$
$$\gamma_2(e) = 000$$

The average number of bits per letter using γ_2 is

$$.32 \cdot 2 + .25 \cdot 2 + .20 \cdot 2 + .18 \cdot 3 + .05 \cdot 3 = 2.23.$$

Optimal Prefix Code

$$\gamma_2(a) = 11$$
$$\gamma_2(b) = 10$$
$$\gamma_2(c) = 01$$
$$\gamma_2(d) = 001$$
$$\gamma_2(e) = 000$$

The average number of bits per letter using γ_2 is

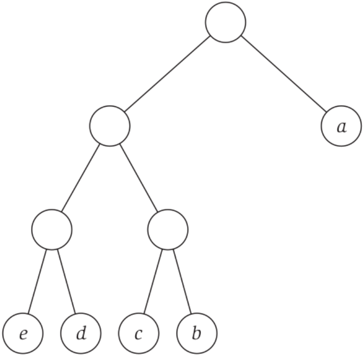
$$.32 \cdot 2 + .25 \cdot 2 + .20 \cdot 2 + .18 \cdot 3 + .05 \cdot 3 = 2.23.$$

Optimal Prefix Code

For a given alphabet S and probability distribution f, a prefix code γ over S is optimal if for all prefix codes γ' over S we have

$$ABL(\gamma) \leq ABL(\gamma')$$

Labelled Binary Trees



Code:
a→1
b→011
c→010
d→001
e→000

- LBT is a binary tree where leaves are labelled with letters from S.
- There is bijection between LBTs and prefix codes. They represent the same thing.
- Notation: Given LBT T and a leaf node u let $lb(u) \in S$ denote the label of u.

- Let $ABL(T) \quad = \quad \Sigma_{x \in S} \quad f_x \cdot depth(x)$

Properties of LBT

- Let $ABL(T) \quad = \quad \Sigma_{x \in S} \quad f_x \cdot depth(x)$
- A LBT is called full if every interior node has exactly two children.

Properties of LBT

- Let $ABL(T) \;=\; \Sigma_{x \in S} \;\; f_x \cdot depth(x)$
- A LBT is called full if every interior node has exactly two children.
- Every LBT can be tranformed to a more efficient full LBT. Hence, every optimal LBT is full.

Property of Optimal LBT (1)

Let T be an LBT. Let u, v be its leaf nodes with $lb(u) = y$ and $lb(v) = z$ and $depth(u) < depth(v)$. If T is optimal then $f_y \geq f_z$.

Proof (Exchange argument) We prove contrapositive.

Let T be an LBT. Let u, v be its leaf nodes with $lb(u) = y$ and $lb(v) = z$ and $depth(u) < depth(v)$. If T is optimal then $f_y \geq f_z$.

Proof (Exchange argument) We prove contrapositive.

- Let $f_y < f_z$. Let T' be T with $lb(u)$ and $lb(v)$ exchanged.

Let T be an LBT. Let u, v be its leaf nodes with $lb(u) = y$ and $lb(v) = z$ and $depth(u) < depth(v)$. If T is optimal then $f_y \geq f_z$.

Proof (Exchange argument) We prove contrapositive.

- Let $f_y < f_z$. Let T' be T with $lb(u)$ and $lb(v)$ exchanged.
- $ABL(T) - ABL(T')$
  $= \ (f_y \cdot depth(u) + f_z \cdot depth(v)) -$
  $(f_z \cdot depth(u) + f_y \cdot depth(v))$
  $= \ (f_y - f_z) \cdot (depth(u) - depth(v))$
  > 0 (as both factors are negative)

Property of Optimal LBT (1)

Let T be an LBT. Let u, v be its leaf nodes with $lb(u) = y$ and $lb(v) = z$ and $depth(u) < depth(v)$. If T is optimal then $f_y \geq f_z$.

Proof (Exchange argument) We prove contrapositive.

- Let $f_y < f_z$. Let T' be T with $lb(u)$ and $lb(v)$ exchanged.
- $ABL(T) - ABL(T')$
  $= (f_y \cdot depth(u) + f_z \cdot depth(v)) -$
  $(f_z \cdot depth(u) + f_y \cdot depth(v))$
  $= (f_y - f_z) \cdot (depth(u) - depth(v))$
  > 0 (as both factors are negative)
- Hence T is not optimal.

Property of Optimal LBT (1)

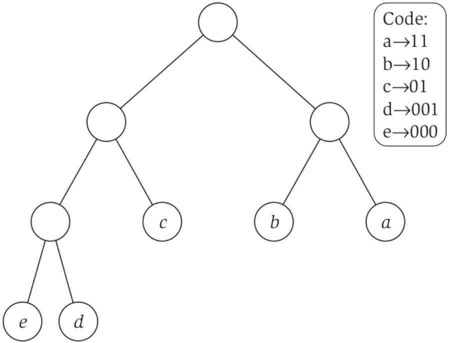
Let T be an LBT. Let u, v be its leaf nodes with $lb(u) = y$ and $lb(v) = z$ and $depth(u) < depth(v)$. If T is optimal then $f_y \geq f_z$.

Proof (Exchange argument) We prove contrapositive.

- Let $f_y < f_z$. Let T' be T with $lb(u)$ and $lb(v)$ exchanged.
- $ABL(T) - ABL(T')$
  $= \quad (f_y \cdot depth(u) + f_z \cdot depth(v)) -$
  $(f_z \cdot depth(u) + f_y \cdot depth(v))$
  $= \quad (f_y - f_z) \cdot (depth(u) - depth(v))$
  $> 0 \qquad$ (as both factors are negative)
- Hence T is not optimal.

Hence, in optimal LBT, as depth of leaves decreases the probability must increase or remain same.

Example: Optimal LBT



$$f_a = .32, \quad f_b = .25, \quad f_c = .20, \quad f_d = .18, \quad f_e = .05.$$

Property of Optimal LBT (2)

Greedy Choice

Let $x, y \in S$ s.t. $\forall z \in S..(f_z \leq f_x \wedge f_z \leq f_y)$. Such x, y are called least frequent pair. Then, there exists a LBT where x, y occur as siblings at maximum depth.

Huffman Algorithm for Optimal LBT (Huffman Code)

Given S and probability distribution f, constructl LBT H in bottom up fashion as follows.

1. Let x, y be a least frequent pair. Make a subtree with parent $u_{(x,y)}$ and two children labelled x and y.
2. Remove x, y from S and add new letter (x, y). Let $f_{(x,y)} = f_x + f_y$.
3. Repeat (1) if more than 1 letter. Replace leaf labelled (x, y) with subtree $u_{(x,y)}$.
4. Call the resulting LBT as H.

Claim: The LBT H constructed as above is optimal.

Example

$$f_a = .32, \quad f_b = .25, \quad f_c = .20, \quad f_d = .18, \quad f_e = .05.$$

Example

$$f_a = .32, \quad f_b = .25, \quad f_c = .20, \quad f_d = .18, \quad f_e = .05.$$

Example

$$f_a = .32, \quad f_b = .25, \quad f_c = .20, \quad f_d = .18, \quad f_e = .05.$$