Greedy Algorithms

Murat Osmanoglu

Interval Scheduling

- given a set of intervals (I_1, I_2, ... , I_n)
- each interval I_i has a starting time s_i, a finishing time f_i
- your task is to find the largest subset of mutually non-overlapping intervals

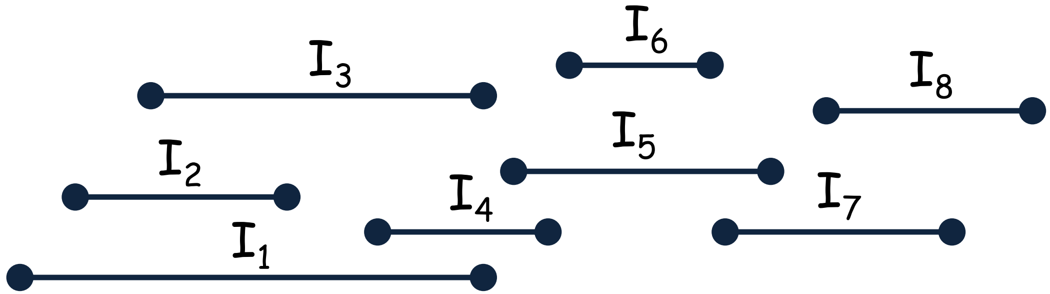
Interval Scheduling

- given a set of intervals (I_1, I_2, ... , I_n)
- each interval I_i has a starting time s_i, a finishing time f_i
- your task is to find the largest subset of mutually non-overlapping intervals

- Suppose there are n meetings requests for a meeting room.
- Each meeting i has a starting time s_i and an ending time t_i.
- We have a constraint : no two meetings can not be scheduled at same time.
- Our goal is to schedule as many meetings as possible

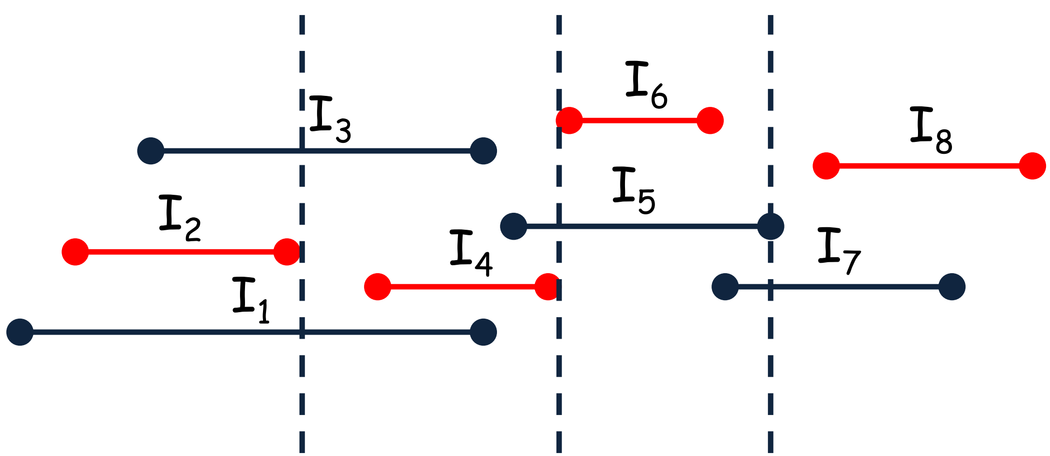
Interval Scheduling

- given a set of intervals $(I_1, I_2, \ldots, I_n)$
- each interval I_i has a starting time s_i, a finishing time f_i
- your task is to find the largest subset of mutually non-overlapping intervals



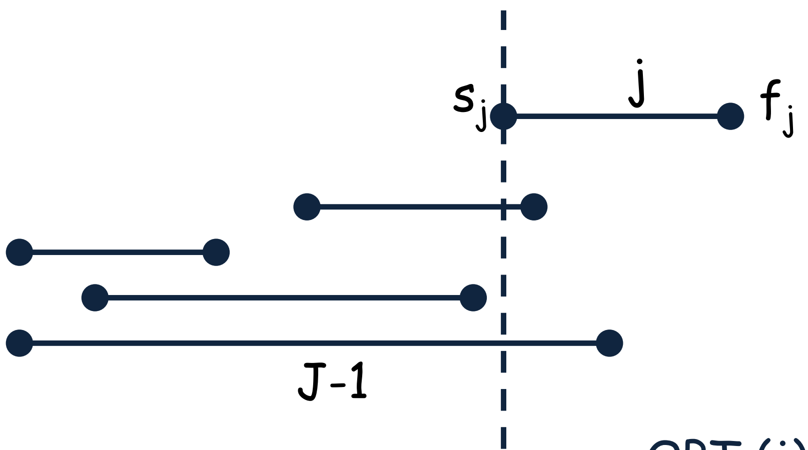
Interval Scheduling

- given a set of intervals ($I_1, I_2, \ldots, I_n$)
- each interval I_i has a starting time s_i, a finishing time f_i
- your task is to find the largest subset of mutually non-overlapping intervals



Interval Scheduling

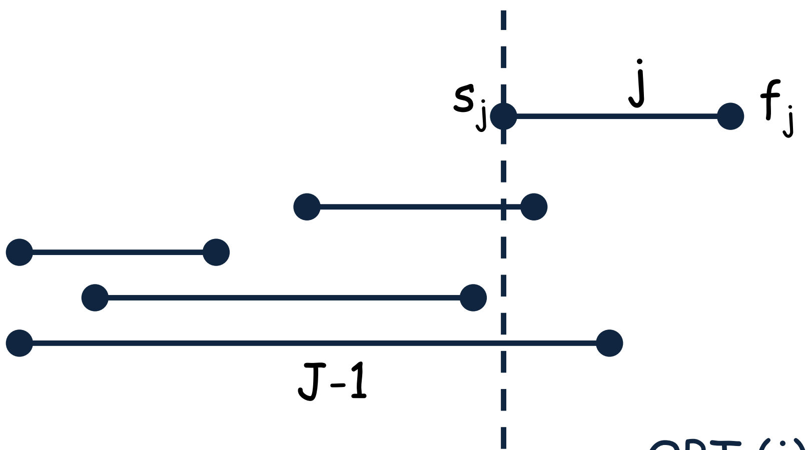
Dynamic Programming Solution



OPT (j) = max { OPT (j-1), 1 + OPT (p(j)) }

Interval Scheduling

Dynamic Programming Solution



OPT (j) = max { OPT (j-1), 1 + OPT (p(j)) }

Can we get a simpler solution?

Interval Scheduling

- solve the problem in myopic fashion

  (don't pay attention the global situaton - don't consider all possible solutions)

- make desicion at each step based on improving local state

  (use greedy approach – pick the one available to you at the moment based on some fixed and simple priority rules)

Interval Scheduling

- What is the best option?
  set the priority rules!

Interval Scheduling

- What is the best option?
  set the priority rules!

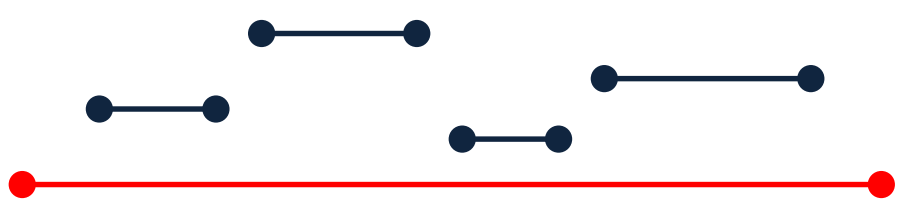
- choose the first interval as the one having the earliest start time
- remove all intervals not compatible with the chosen one

Interval Scheduling

- What is the best option?

  set the priority rules!

- choose the first interval as the one having the earliest start time
- remove all intervals not compatible with the chosen one



Interval Scheduling

- What is the best option?

  set the priority rules!

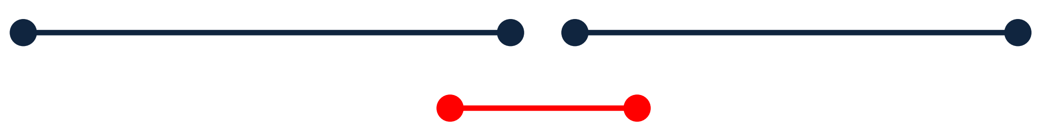
- choose the first interval as the shortest one
- remove all intervals not compatible with the chosen one

Interval Scheduling

- What is the best option?

  set the priority rules!

- choose the first interval as the shortest one
- remove all intervals not compatible with the chosen one



Interval Scheduling

- What is the best option?

  set the priority rules!

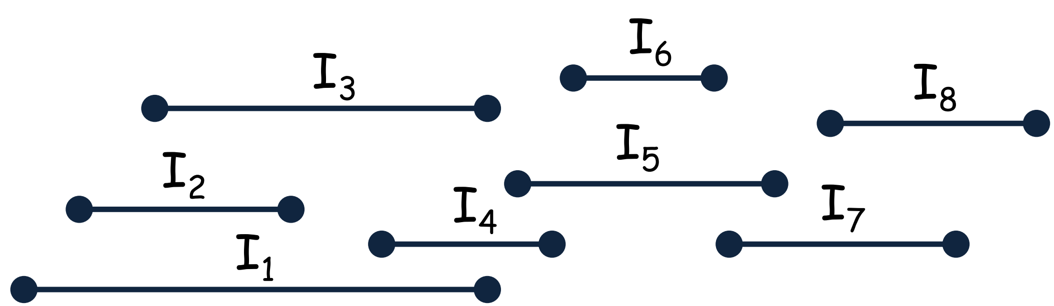
- choose the first interval as the one having the earliest finish time
- remove all intervals not compatible with the chosen one

Interval Scheduling

- What is the best option?

  set the priority rules!

- choose the first interval as the one having the earliest finish time
- remove all intervals not compatible with the chosen one

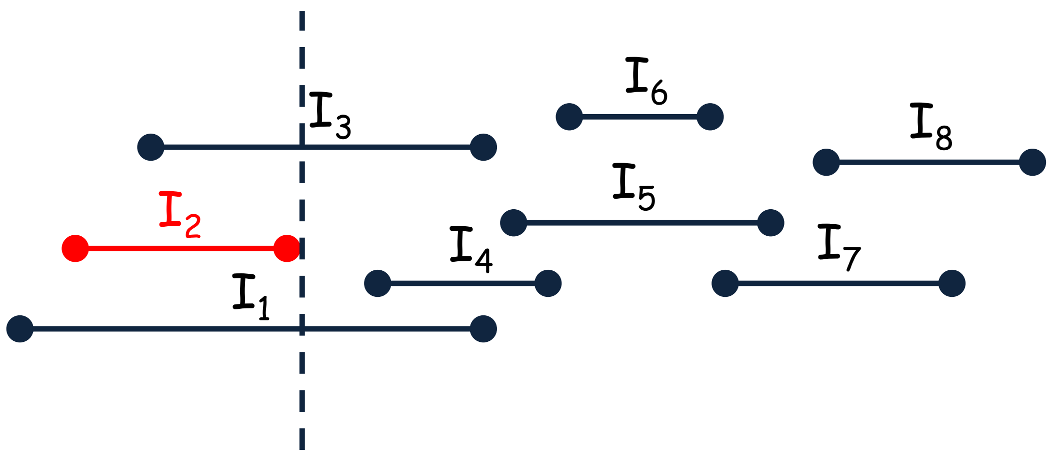


Interval Scheduling

- What is the best option?

  set the priority rules!

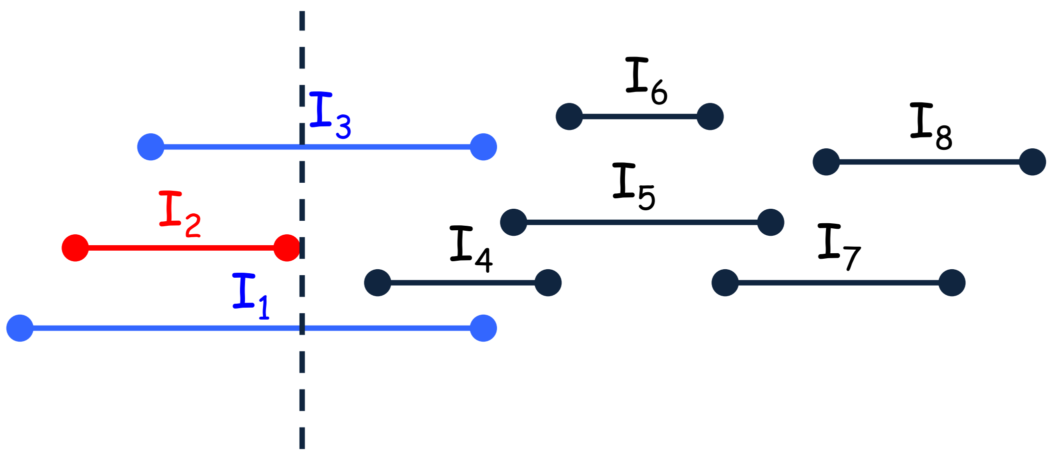
- choose the first interval as the one having the earliest finish time
- remove all intervals not compatible with the chosen one



Interval Scheduling

- What is the best option?
  set the priority rules!

- choose the first interval as the one having the earliest finish time
- remove all intervals not compatible with the chosen one

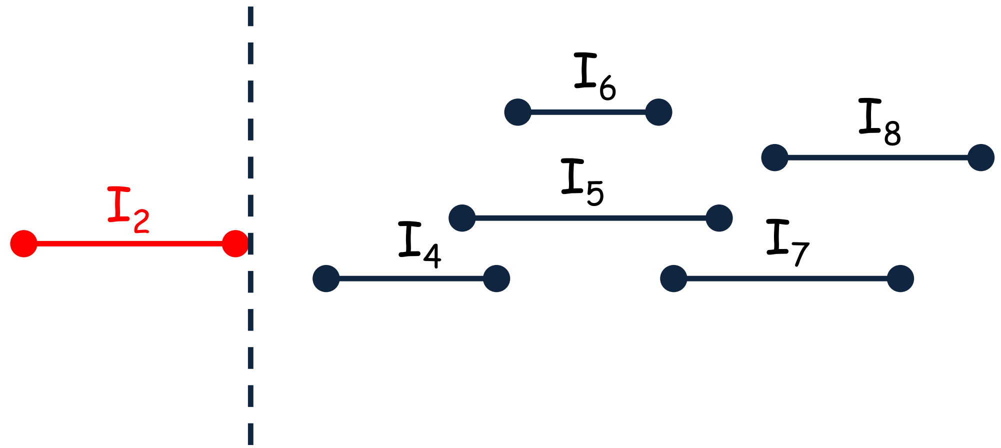


Interval Scheduling

- What is the best option?

  set the priority rules!

- choose the first interval as the one having the earliest finish time
- remove all intervals not compatible with the chosen one

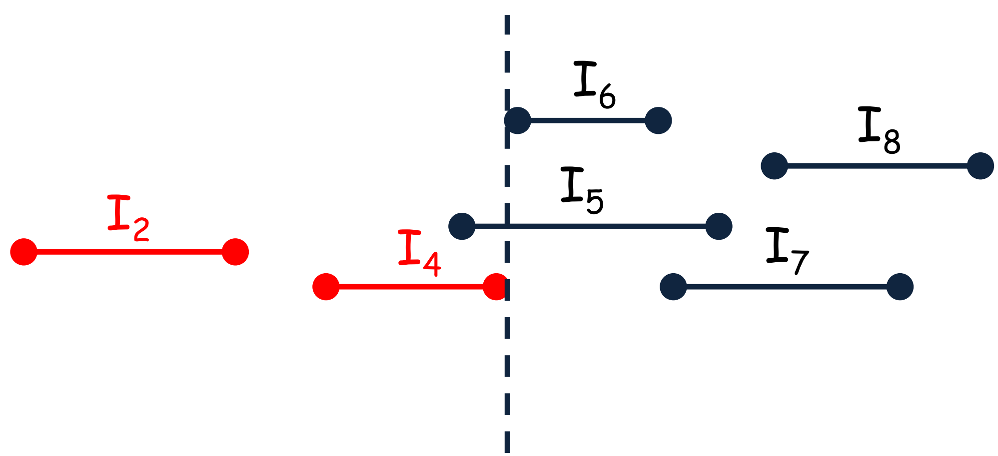


Interval Scheduling

- What is the best option?

    set the priority rules!

- choose the first interval as the one having the earliest finish time
- remove all intervals not compatible with the chosen one

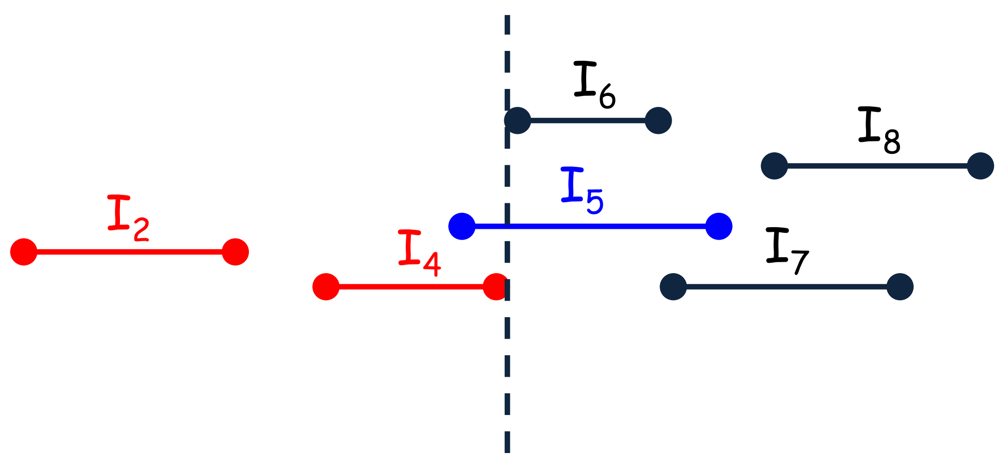


Interval Scheduling

- What is the best option?

  set the priority rules!

- choose the first interval as the one having the earliest finish time
- remove all intervals not compatible with the chosen one

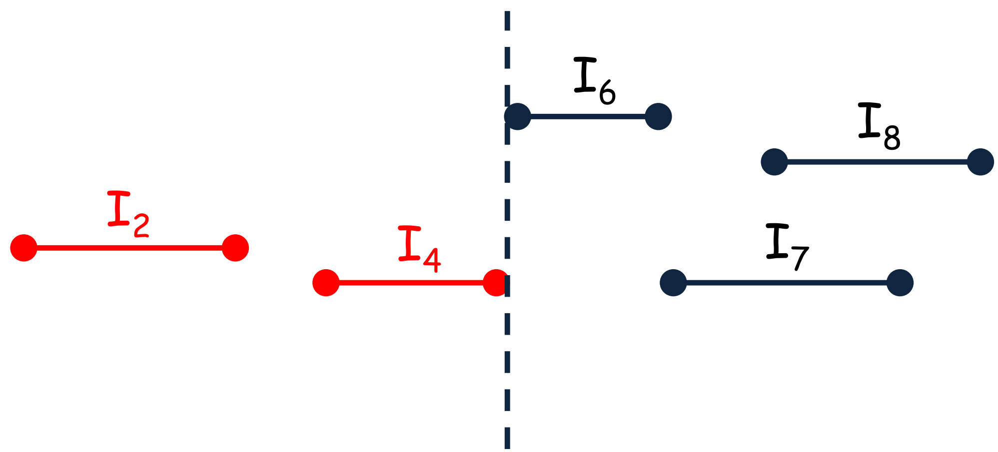


Interval Scheduling

- What is the best option?

  set the priority rules!

- choose the first interval as the one having the earliest finish time
- remove all intervals not compatible with the chosen one

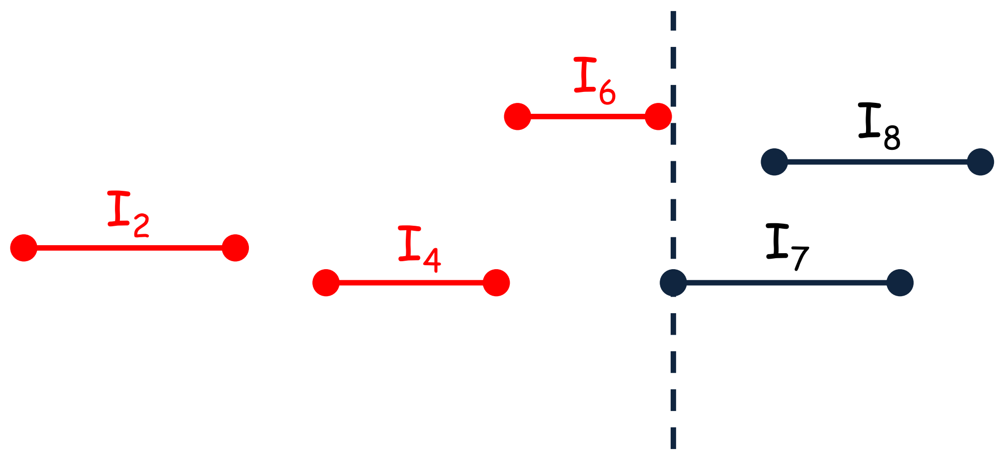


Interval Scheduling

- What is the best option?

  set the priority rules!

- choose the first interval as the one having the earliest finish time
- remove all intervals not compatible with the chosen one

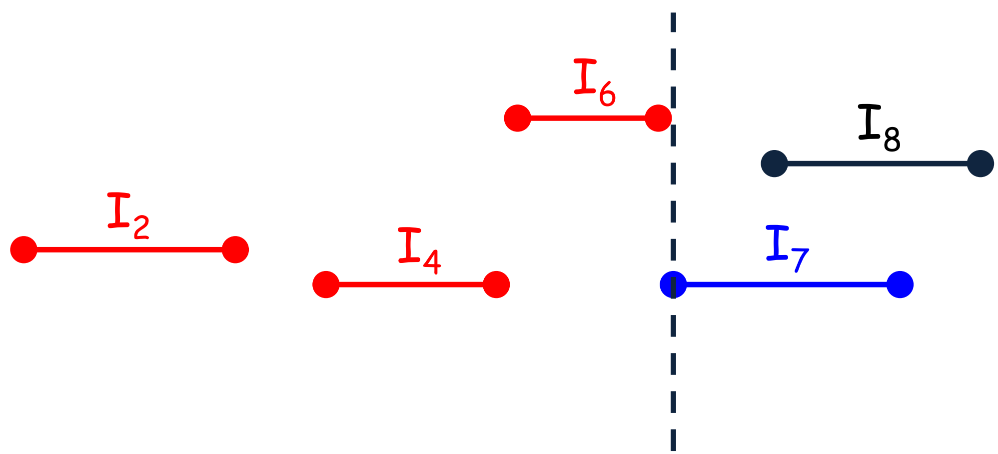


Interval Scheduling

- What is the best option?

  set the priority rules!

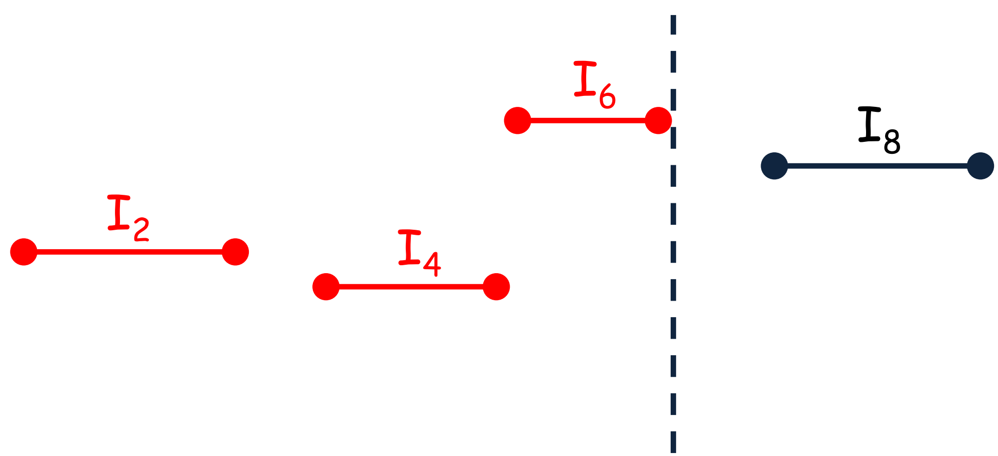
- choose the first interval as the one having the earliest finish time
- remove all intervals not compatible with the chosen one



Interval Scheduling

- What is the best option?
  set the priority rules!

- choose the first interval as the one having the earliest finish time
- remove all intervals not compatible with the chosen one

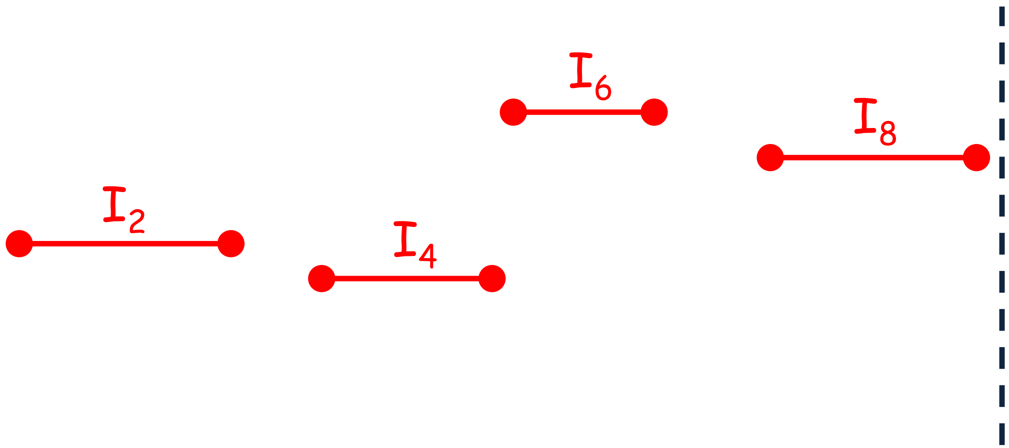


Interval Scheduling

- What is the best option?

  set the priority rules!

- choose the first interval as the one having the earliest finish time
- remove all intervals not compatible with the chosen one



Interval Scheduling

input : n interval (I_1, … , I_n) together with their start time and finish time

--sort intervals according to their finish time ($f_1 \leq f_2 \leq \ldots \leq f_n$)
--initialize an empty set S

```
for (i=1 to n)
    if interval I_i is compatible with S
        S = S ∪ { I_i }
return S
```

Interval Scheduling

Theorem (Greedy-choice property): The interval having earliest finish time (first interval) will be part of some optimal solution set. (Our greedy approach yields us an optimal solution)

Proof

Interval Scheduling

Theorem (Greedy-choice property): The interval having earliest finish time (first interval) will be part of some optimal solution set. (Our greedy approach yields us an optimal solution)

Proof

Assume S is an optimal solution set for problem and S does not contain the first interval I_1.

Interval Scheduling

Theorem (Greedy-choice property): The interval having earliest finish time (first interval) will be part of some optimal solution set. (Our greedy approach yields us an optimal solution)

Proof

Assume S is an optimal solution set for problem and S does not contain the first interval I_1.

Let I_i^* be the interval in S having earliest finish time.

Interval Scheduling

Theorem (Greedy-choice property): The interval having earliest finish time (first interval) will be part of some optimal solution set. (Our greedy approach yields us an optimal solution)

Proof

Assume S is an optimal solution set for problem and S does not contain the first interval I_1.

Let I_i^* be the interval in S having earliest finish time.

Since I_1 has the earliest finish time for all, $f_1 \leq f_i$.

Interval Scheduling

Theorem (Greedy-choice property): The interval having earliest finish time (first interval) will be part of some optimal solution set. (Our greedy approach yields us an optimal solution)

Proof

Assume S is an optimal solution set for problem and S does not contain the first interval I_1.

Let I_i^* be the interval in S having earliest finish time.

Since I_1 has the earliest finish time for all, $f_1 \leq f_i$.

$$S^* = S - \{ I_i^* \} \cup \{ I_1 \} \quad \text{such that } |S^*| = |S|$$

Interval Scheduling

Theorem (Greedy-choice property): The interval having earliest finish time (first interval) will be part of some optimal solution set. (Our greedy approach yields us an optimal solution)

Proof

Assume S is an optimal solution set for problem and S does not contain the first interval I_1.

Let I_i^* be the interval in S having earliest finish time.

Since I_1 has the earliest finish time for all, $f_1 \le f_i$.

$$S^* = S - \{ I_i^* \} \cup \{ I_1 \} \quad \text{such that } |S^*| = |S|$$

This is a contradiction!

Greedy Algorithms

- solve the problem by breaking it a sequence of subproblems
- make the best local choice among all feasible one available on that moment (one choice at a time)
  - your choice does not depend on any future choices or any past choices you have made
- prove that the Greedy Choice Property satisfies. A sequence of locally optimal choices yields a global optimal solution

Cashier's Problem

- given a certain amount of money, M cents, and a set of denominations of coins c_1 , ... , c_m
- make change for M cents using a minimum total number of coins

  (each denomination is available in unlimited quantity)

Cashier's Problem

- given a certain amount of money, M cents, and a set of denominations of coins $c_1, \ldots, c_m$
- make change for M cents using a minimum total number of coins

  (each denomination is available in unlimited quantity)

147 cents

(25,10,5,1)

Cashier's Problem

- given a certain amount of money, M cents, and a set of denominations of coins c_1 , … , c_m
- make change for M cents using a minimum total number of coins

  (each denomination is available in unlimited quantity)

2 + 4 + 5 + 32 = 43

<u>147 cents</u>

(25,10,5,1)

Cashier's Problem

- given a certain amount of money, M cents, and a set of denominations of coins c_1 , … , c_m
- make change for M cents using a minimum total number of coins

  (each denomination is available in unlimited quantity)

2 + 4 + 5 + 32 = 43

147 cents

(25,10,5,1)

4 + 3 + 2 + 7 = 16

Cashier's Problem

- given a certain amount of money, M cents, and a set of denominations of coins c_1 , … , c_m
- make change for M cents using a minimum total number of coins

  (each denomination is available in unlimited quantity)

147 cents

(25,10,5,1)

2 + 4 + 5 + 32 = 43

4 + 3 + 2 + 7 = 16

Cashier's Problem

- given a certain amount of money, M cents, and a set of denominations of coins c_1 , … , c_m
- make change for M cents using a minimum total number of coins

  (each denomination is available in unlimited quantity)

2 + 4 + 5 + 32 = 43

147 cents

(25,10,5,1)

4 + 3 + 2 + 7 = 16

5 +

Cashier's Problem

- given a certain amount of money, M cents, and a set of denominations of coins c_1 , ... , c_m
- make change for M cents using a minimum total number of coins

  (each denomination is available in unlimited quantity)

2 + 4 + 5 + 32 = 43

147 cents

(25,10,5,1)

4 + 3 + 2 + 7 = 16

5 + 2 +

Cashier's Problem

- given a certain amount of money, M cents, and a set of denominations of coins c_1 , … , c_m
- make change for M cents using a minimum total number of coins

(each denomination is available in unlimited quantity)

2 + 4 + 5 + 32 = 43

147 cents

(25,10,5,1)

4 + 3 + 2 + 7 = 16

5 + 2 + 0 +

Cashier's Problem

- given a certain amount of money, M cents, and a set of denominations of coins $c_1, \ldots, c_m$
- make change for M cents using a minimum total number of coins

  (each denomination is available in unlimited quantity)

2 + 4 + 5 + 32 = 43

147 cents

(25,10,5,1)

4 + 3 + 2 + 7 = 16

5 + 2 + 0 + 2 = 9

Cashier's Problem

input : an amount of money M
a set of denominations $(c_1 , \ldots , c_n)$

```
sort denominations
c_1 ≥ … ≥ c_n
totalw = M
j=1
k=0
while (j ≤ n)
    if ( c_j ≤ totalw )
        totalw = totalw - c_j
        k = k +1
    else
        j = j +1
return k
```

Cashier's Problem

Theorem (Greedy-choice property): Let (10, 5, 1) be the denomination set. For the amount M, there exists an optimal solution set that contains the largest denomination $c_j \leq M$.

(Our greedy approach yields us an optimal solution)

Proof

Cashier's Problem

Theorem (Greedy-choice property): Let (10, 5, 1) be the denomination set. For the amount M, there exists an optimal solution set that contains the largest denomination $c_j \le M$.

(Our greedy approach yields us an optimal solution)

Proof

Assume S is an optimal solution for M, and $10 \le M$. But S does not contain any 10.

Cashier's Problem

Theorem (Greedy-choice property): Let (10, 5, 1) be the denomination set. For the amount M, there exists an optimal solution set that contains the largest denomination $c_j \leq M$.

(Our greedy approach yields us an optimal solution)

Proof

Assume S is an optimal solution for M, and $10 \leq M$. But S does not contain any 10.

$$M = a.5 + b.1 \quad \text{(total } a + b \text{ coins)}$$

Cashier's Problem

Theorem (Greedy-choice property): Let (10, 5, 1) be the denomination set. For the amount M, there exists an optimal solution set that contains the largest denomination $c_j \leq M$.

(Our greedy approach yields us an optimal solution)

Proof

Assume S is an optimal solution for M, and 10 ≤ M. But S does not contain any 10.

$$M = a.5 + b.1 \quad \text{(total } a + b \text{ coins)}$$

M can be written as

$$M = a.5 + b.1 = 10 - 10 + a.5 + b.1$$

Cashier's Problem

Theorem (Greedy-choice property): Let (10, 5, 1) be the denomination set. For the amount M, there exists an optimal solution set that contains the largest denomination $c_j \leq M$.

(Our greedy approach yields us an optimal solution)

Proof

Assume S is an optimal solution for M, and $10 \leq M$. But S does not contain any 10.

$$M = a.5 + b.1 \quad \text{(total } a + b \text{ coins)}$$

M can be written as

$$M = a.5 + b.1 = 10 - 10 + a.5 + b.1$$
$$= 1.10 + (a - 2).5 + b.1$$

Cashier's Problem

Theorem (Greedy-choice property): Let (10, 5, 1) be the denomination set. For the amount M, there exists an optimal solution set that contains the largest denomination $c_j \le M$.

(Our greedy approach yields us an optimal solution)

Proof

Assume S is an optimal solution for M, and $10 \le M$. But S does not contain any 10.

$$M = a.5 + b.1 \quad \text{(total } a + b \text{ coins)}$$

M can be written as

$$M = a.5 + b.1 = 10 - 10 + a.5 + b.1$$
$$= 1.10 + (a - 2).5 + b.1 \quad \text{(total } a + b - 1 \text{ coins)}$$

Cashier's Problem

Theorem (Greedy-choice property): Let (10, 5, 1) be the denomination set. For the amount M, there exists an optimal solution set that contains the largest denomination $c_j \leq M$.

(Our greedy approach yields us an optimal solution)

Proof

Assume S is an optimal solution for M, and $10 \leq M$. But S does not contain any 10.

$$M = a.5 + b.1 \quad \text{(total } a + b \text{ coins)}$$

M can be written as

$$M = a.5 + b.1 = 10 - 10 + a.5 + b.1$$
$$= 1.10 + (a - 2).5 + b.1 \quad \text{(total } a + b - 1 \text{ coins)}$$

This is contradiction.

Cashier's Problem

Will the Greedy Technique give an optimal solution for all denomination set?

Cashier's Problem

Will the Greedy Technique give an optimal solution for all denomination set?

10

6

1

M = 24

Cashier's Problem

Will the Greedy Technique give an optimal solution for all denomination set?

10

6

1

M = 24 2 +

Cashier's Problem

Will the Greedy Technique give an optimal solution for all denomination set?

10

6

1

M = 24 2 + 0 +

Cashier's Problem

Will the Greedy Technique give an optimal solution for all denomination set?

10	6	1
		

M = 24   2 + 0 + 4 = 6

Cashier's Problem

Will the Greedy Technique give an optimal solution for all denomination set?

10 6 1

M = 24 2 + 0 + 4 = 6

4.6 = 24

Cashier's Problem

Will the Greedy Technique give an optimal solution for all denomination set?

10  6  1 

M = 24 2 + 0 + 4 = 6

use dynamic programming

4.6 = 24

Fractional Knapsack

- given n items and a knapsack with the capacity M
- each item i has a weight w_i, and a value p_i
- you are allowed to get a fraction x_i of an item i that yields a profit $x_i.p_i$ where $0 \leq x_i \leq 1$
- your goal is to get a filling that maximizes the profit under the weight constraint M

Fractional Knapsack

- given n items and a knapsack with the capacity M
- each item i has a weight w_i, and a value p_i
- you are allowed to get a fraction x_i of an item i that yields a profit $x_i.p_i$ where $0 \leq x_i \leq 1$
- your goal is to get a filling that maximizes the profit under the weight constraint M

w_1=20 P_1=10	w_2=5 p_2=5	w_1=15 p_1=5	w_1=5 p_1=15
pearl	gold	silver	diamonds

M = 25

Fractional Knapsack

- given n items and a knapsack with the capacity M
- each item i has a weight w_i, and a value p_i
- you are allowed to get a fraction x_i of an item i that yields a profit $x_i.p_i$ where $0 \leq x_i \leq 1$
- your goal is to get a filling that maximizes the profit under the weight constraint M

$w_1=20$ $P_1=10$	$w_2=5$ $p_2=5$	$w_1=15$ $p_1=5$	$w_1=5$ $p_1=15$	<u>M = 25</u>
pearl	gold	silver	diamonds	
$p_1/w_1=1/2$	$p_2/w_2=1$	$p_3/w_3=1/3$	$p_4/w_4=3$	

Fractional Knapsack

- given n items and a knapsack with the capacity M
- each item i has a weight w_i, and a value p_i
- you are allowed to get a fraction x_i of an item i that yields a profit $x_i.p_i$ where $0 \le x_i \le 1$
- your goal is to get a filling that maximizes the profit under the weight constraint M

$w_1=20$ $P_1=10$	$w_2=5$ $p_2=5$	$w_1=15$ $p_1=5$	$w_1=5$ $p_1=15$
pearl	gold	silver	diamonds
$p_1/w_1=1/2$	$p_2/w_2=1$	$p_3/w_3=1/3$	$p_4/w_4=3$

M = 25

M = 25

Fractional Knapsack

- given n items and a knapsack with the capacity M
- each item i has a weight w_i, and a value p_i
- you are allowed to get a fraction x_i of an item i that yields a profit $x_i.p_i$ where $0 \le x_i \le 1$
- your goal is to get a filling that maximizes the profit under the weight constraint M

pearl	gold	silver	diamonds
w_1=20 P_1=10	w_2=5 p_2=5	w_1=15 p_1=5	w_1=5 p_1=15
p_1/w_1 =1/2	p_2/w_2 =1	p_3/w_3 =1/3	p_4/w_4 =3

M = 25

M = 25

3 . 5

Fractional Knapsack

- given n items and a knapsack with the capacity M
- each item i has a weight w_i, and a value p_i
- you are allowed to get a fraction x_i of an item i that yields a profit $x_i.p_i$ where $0 \leq x_i \leq 1$
- your goal is to get a filling that maximizes the profit under the weight constraint M

pearl	gold	silver	diamonds
$w_1=20$ $P_1=10$	$w_2=5$ $p_2=5$	$w_1=15$ $p_1=5$	$w_1=5$ $p_1=15$
$p_1/w_1=1/2$	$p_2/w_2=1$	$p_3/w_3=1/3$	$p_4/w_4=3$

M = 25

M = 20

3 . 5 + 1 . 5

Fractional Knapsack

- given n items and a knapsack with the capacity M
- each item i has a weight w_i, and a value p_i
- you are allowed to get a fraction x_i of an item i that yields a profit $x_i.p_i$ where $0 \leq x_i \leq 1$
- your goal is to get a filling that maximizes the profit under the weight constraint M

w_1=20 P_1=10	w_2=5 p_2=5	w_1=15 p_1=5	w_1=5 p_1=15
pearl	gold	silver	diamonds
p_1/w_1 =1/2	p_2/w_2 =1	p_3/w_3 =1/3	p_4/w_4 =3

M = 25

M = 15

$3 \,.\, 5 + 1 \,.\, 5 + (1/2) \,.\, 15$

Fractional Knapsack

- given n items and a knapsack with the capacity M
- each item i has a weight w_i, and a value p_i
- you are allowed to get a fraction x_i of an item i that yields a profit $x_i.p_i$ where $0 \leq x_i \leq 1$
- your goal is to get a filling that maximizes the profit under the weight constraint M

pearl	gold	silver	diamonds
w_1=20 P_1=10	w_2=5 p_2=5	w_1=15 p_1=5	w_1=5 p_1=15
p_1/w_1 =1/2	p_2/w_2 =1	p_3/w_3 =1/3	p_4/w_4 =3

M = 25

M = 0

3 . 5 + 1 . 5 + (1/2) . 15 = 27.5

Fractional Knapsack

<u>input</u> : n items together with their prices p_i
and weight w_i, and a knapsack with the capacity M

```
sort items according to the ratio (p_i/w_i)
(p_1/w_1) ≤ … ≤ (p_n/w_n)
totalw = M
j=1
while (totalw > 0)
      if ( w_j > totalw )
            add totalw fraction of item j to the knapsack
            totalw = 0
      else
            add item j to the knapsack
            totalw = totalw - w_j
            j = j + 1
return knapsack
```

Fractional Knapsack

Theorem (Greedy-choice property): Let j be the item with the maximum ratio p_i/w_i. There exists an optimal solution that contains item j as much as possible.

(Our greedy approach yields us an optimal solution)

Proof

Fractional Knapsack

Theorem (Greedy-choice property): Let j be the item with the maximum ratio p_i/w_i. There exists an optimal solution that contains item j as much as possible.

(Our greedy approach yields us an optimal solution)

Proof

Assume S is an optimal solution with the full knapsack of capacity M and total profit U.

Fractional Knapsack

Theorem (Greedy-choice property): Let j be the item with the maximum ratio p_i/w_i. There exists an optimal solution that contains item j as much as possible.

(Our greedy approach yields us an optimal solution)

Proof

Assume S is an optimal solution with the full knapsack of capacity M and total profit U.

Assume S does not contain the item j as much as possible.

There must exist some item k such that $k \neq j$ and $(p_k / w_k) < (p_j / w_j)$

Fractional Knapsack

Theorem (Greedy-choice property): Let j be the item with the maximum ratio p_i/w_i. There exists an optimal solution that contains item j as much as possible.

(Our greedy approach yields us an optimal solution)

Proof

Assume S is an optimal solution with the full knapsack of capacity M and total profit U.

Assume S does not contain the item j as much as possible.

There must exist some item k such that $k \neq j$ and $(p_k / w_k) < (p_j / w_j)$

We take out some amount of item k, (suppose α) and put same amount of item j.

$$S^* = S - \{\alpha \text{ of item } k\} \cup \{\alpha \text{ of item } j\}$$

Fractional Knapsack

Theorem (Greedy-choice property): Let j be the item with the maximum ratio p_i/w_i. There exists an optimal solution that contains item j as much as possible.

(Our greedy approach yields us an optimal solution)

Proof

Assume S is an optimal solution with the full knapsack of capacity M and total profit U.

Assume S does not contain the item j as much as possible.

There must exist some item k such that $k \neq j$ and $(p_k / w_k) < (p_j / w_j)$

We take out some amount of item k, (suppose α) and put same amount of item j.

$$S^* = S - \{\alpha \text{ of item } k\} \cup \{\alpha \text{ of item } j\}$$

Let U^* be the profit of S^*. Then,

$$U^* = U - \alpha.(p_k / w_k) + \alpha.(p_j / w_j)$$

Fractional Knapsack

Theorem (Greedy-choice property): Let j be the item with the maximum ratio p_i/w_i. There exists an optimal solution that contains item j as much as possible.

(Our greedy approach yields us an optimal solution)

Proof

Assume S is an optimal solution with the full knapsack of capacity M and total profit U.

Assume S does not contain the item j as much as possible.

There must exist some item k such that $k \neq j$ and $(p_k / w_k) < (p_j / w_j)$

We take out some amount of item k, (suppose α) and put same amount of item j.

$$S^* = S - \{\alpha \text{ of item } k\} \cup \{\alpha \text{ of item } j\}$$

Let U^* be the profit of S^*. Then,

$$U^* = U - \alpha.(p_k / w_k) + \alpha.(p_j / w_j)$$

Since $(p_k / w_k) < (p_j / w_j)$, $U^* > U$.

Fractional Knapsack

Theorem (Greedy-choice property): Let j be the item with the maximum ratio p_i/w_i. There exists an optimal solution that contains item j as much as possible.

(Our greedy approach yields us an optimal solution)

Proof

Assume S is an optimal solution with the full knapsack of capacity M and total profit U.

Assume S does not contain the item j as much as possible.

There must exist some item k such that $k \neq j$ and $(p_k / w_k) < (p_j / w_j)$

We take out some amount of item k, (suppose α) and put same amount of item j.

$$S^* = S - \{\alpha \text{ of item } k\} \cup \{\alpha \text{ of item } j\}$$

Let U^* be the profit of S^*. Then,

$$U^* = U - \alpha.(p_k / w_k) + \alpha.(p_j / w_j)$$

Since $(p_k / w_k) < (p_j / w_j)$, $U^* > U$.

This is contradiction!

0/1 Knapsack Problem

- given n items and a knapsack with the capacity M
- each item i has a weight w_i, and a value p_i
- your goal is to get a filling that maximizes the profit under the weight constraint M
  (You cannot take fraction of an item, you take the item or not)

0/1 Knapsack Problem

- given n items and a knapsack with the capacity M
- each item i has a weight w_i, and a value p_i
- your goal is to get a filling that maximizes the profit under the weight constraint M
  (You cannot take fraction of an item, you take the item or not)

Can we use Greedy Technique to solve this problem?

0/1 Knapsack Problem

- given n items and a knapsack with the capacity M
- each item i has a weight w_i, and a value p_i
- your goal is to get a filling that maximizes the profit under the weight constraint M
  (You cannot take fraction of an item, you take the item or not)

Can we use Greedy Technique to solve this problem?

$w_1=10$ $p_1=60$	$w_2=20$ $p_2=100$	$w_3=30$ $p_3=120$	M = 50
$p_1/w_1=6$	$p_2/w_2=5$	$p_3/w_3=4$	

0/1 Knapsack Problem

- given n items and a knapsack with the capacity M
- each item i has a weight w_i, and a value p_i
- your goal is to get a filling that maximizes the profit under the weight constraint M
  (You cannot take fraction of an item, you take the item or not)

Can we use Greedy Technique to solve this problem?

$w_1=10$ $p_1=60$	$w_2=20$ $p_2=100$	$w_3=30$ $p_3=120$	M = 50
$p_1/w_1=6$	$p_2/w_2=5$	$p_3/w_3=4$	

M = 50 $\longleftrightarrow$ 60

0/1 Knapsack Problem

- given n items and a knapsack with the capacity M
- each item i has a weight w_i, and a value p_i
- your goal is to get a filling that maximizes the profit under the weight constraint M
  (You cannot take fraction of an item, you take the item or not)

Can we use Greedy Technique to solve this problem?

$w_1=10$ $p_1=60$	$w_2=20$ $p_2=100$	$w_3=30$ $p_3=120$	M = 50
$p_1/w_1=6$	$p_2/w_2=5$	$p_3/w_3=4$	

M = 40 $\longleftrightarrow$ 60 + 100

0/1 Knapsack Problem

- given n items and a knapsack with the capacity M
- each item i has a weight w_i, and a value p_i
- your goal is to get a filling that maximizes the profit under the weight constraint M
  (You cannot take fraction of an item, you take the item or not)

Can we use Greedy Technique to solve this problem?

$w_1=10$ $p_1=60$	$w_2=20$ $p_2=100$	$w_3=30$ $p_3=120$	<u>M = 50</u>
$p_1/w_1=6$	$p_2/w_2=5$	$p_3/w_3=4$	

$M = 20 \longleftrightarrow 60 + 100 = 160$

0/1 Knapsack Problem

- given n items and a knapsack with the capacity M
- each item i has a weight w_i, and a value p_i
- your goal is to get a filling that maximizes the profit under the weight constraint M
  (You cannot take fraction of an item, you take the item or not)

Can we use Greedy Technique to solve this problem?

w_1=10, p_1=60	w_2=20, p_2=100	w_3=30, p_3=120	M = 50
p_1/w_1 =6	p_2/w_2 =5	p_3/w_3 =4	

M = 20 ⟷ 60 + 100 = 160

100 + 120 = 220

0/1 Knapsack Problem

- given n items and a knapsack with the capacity M
- each item i has a weight w_i, and a value p_i
- your goal is to get a filling that maximizes the profit under the weight constraint M
  (You cannot take fraction of an item, you take the item or not)

Can we use Greedy Technique to solve this problem?

$w_1=10$, $p_1=60$	$w_2=20$, $p_2=100$	$w_3=30$, $p_3=120$	M = 50
$p_1/w_1=6$	$p_2/w_2=5$	$p_3/w_3=4$	

M = 20 $\longleftrightarrow$ 60 + 100 = 160

100 + 120 = 220

use dynamic programming

Process Scheduling

- given a computer and n processes $p_1, \ldots, p_n$ such that each of them has a completion time t_i
- find an optimal order of processes that has the minimum average finishing time

Process Scheduling

- given a computer and n processes $p_1, \ldots, p_n$ such that each of them has a completion time t_i
- find an optimal order of processes that has the minimum average finishing time

If we define the finishing time C_i of the process i as $C_i = \Sigma_{j=1..i} t_j$, then the average finishing time will be $(\Sigma_{i=1..n} C_i)/n$.

Process Scheduling

- given a computer and n processes $p_1, \ldots, p_n$ such that each of them has a completion time t_i
- find an optimal order of processes that has the minimum average finishing time

If we define the finishing time C_i of the process i as $C_i = \Sigma_{j=1..i} t_j$, then the average finishing time will be $(\Sigma_{i=1..n} C_i)/n$.

Our goal is to minimize $(\Sigma_{i=1..n} C_i)/n$

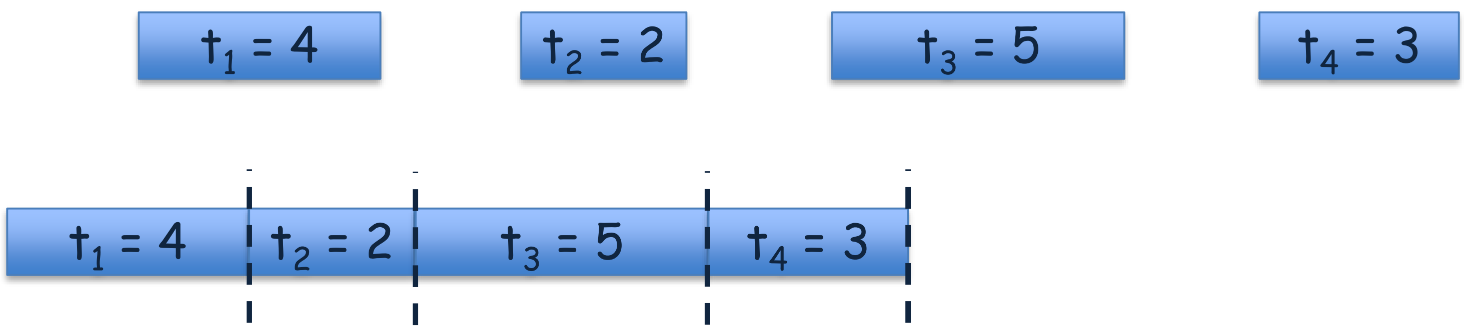
Process Scheduling

- Given t_1 , … , t_n
- If we define the finishing time C_i of the process i as $C_i = \Sigma_{j=1..i} t_j$, then the average finishing time will be $(\Sigma_{i=1..n} C_i)/n$.
- Our goal is to minimize $(\Sigma_{i=1..n} C_i)/n$

$t_1 = 4$ $t_2 = 2$ $t_3 = 5$ $t_4 = 3$

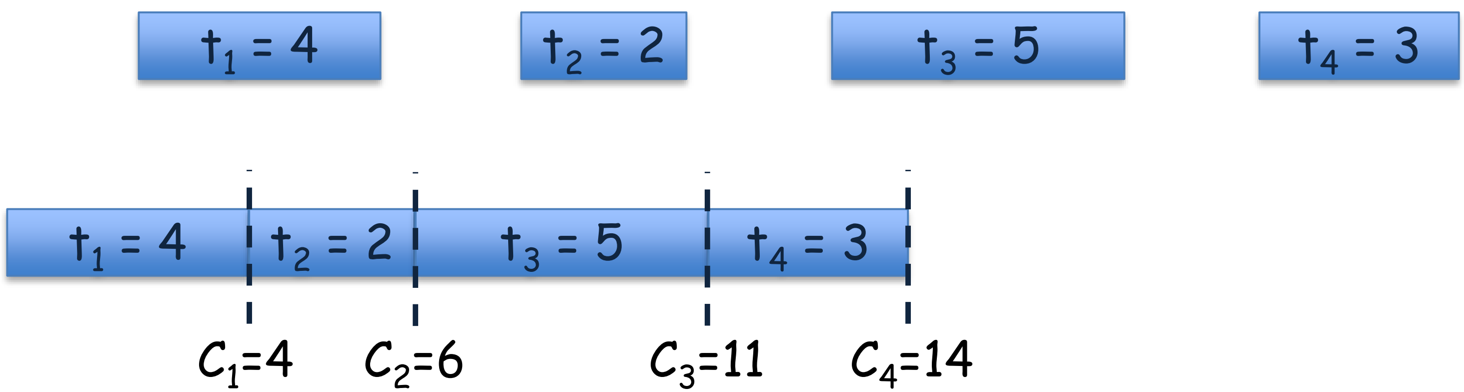
Process Scheduling

- Given t_1 , … , t_n
- If we define the finishing time C_i of the process i as $C_i = \Sigma_{j=1..i} t_j$, then the average finishing time will be $(\Sigma_{i=1..n} C_i)/n$.
- Our goal is to minimize $(\Sigma_{i=1..n} C_i)/n$



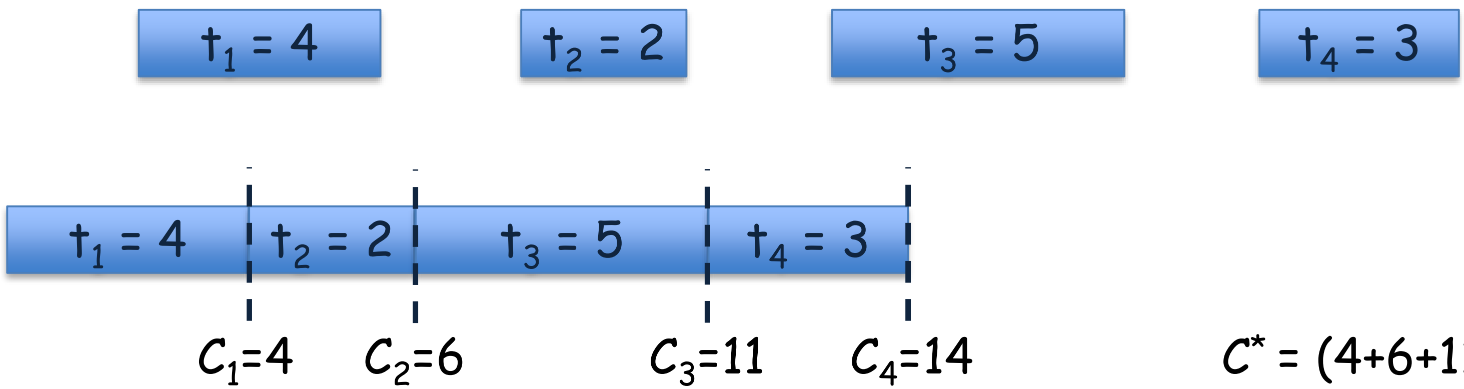
Process Scheduling

- Given t_1 , … , t_n
- If we define the finishing time C_i of the process i as $C_i = \Sigma_{j=1..i} t_j$, then the average finishing time will be $(\Sigma_{i=1..n} C_i)/n$.
- Our goal is to minimize $(\Sigma_{i=1..n} C_i)/n$



Process Scheduling

- Given t_1 , … , t_n
- If we define the finishing time C_i of the process i as $C_i = \Sigma_{j=1..i} t_j$, then the average finishing time will be $(\Sigma_{i=1..n} C_i)/n$.
- Our goal is to minimize $(\Sigma_{i=1..n} C_i)/n$

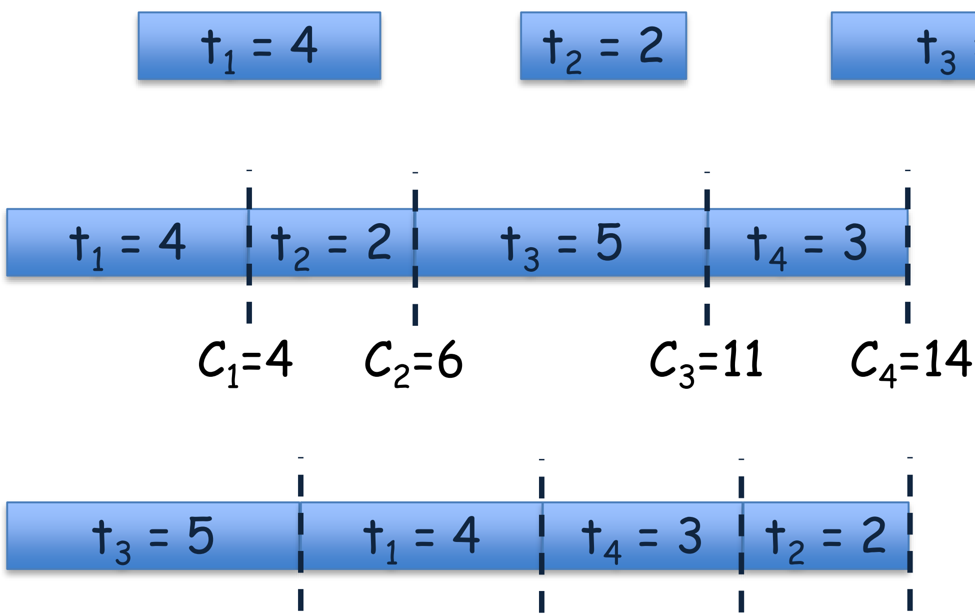


$C^* = (4+6+11+14)/4$

$C^* = 8.75$

Process Scheduling

- Given $t_1, \ldots, t_n$
- If we define the finishing time C_i of the process i as $C_i = \Sigma_{j=1..i} t_j$, then the average finishing time will be $(\Sigma_{i=1..n} C_i)/n$.
- Our goal is to minimize $(\Sigma_{i=1..n} C_i)/n$

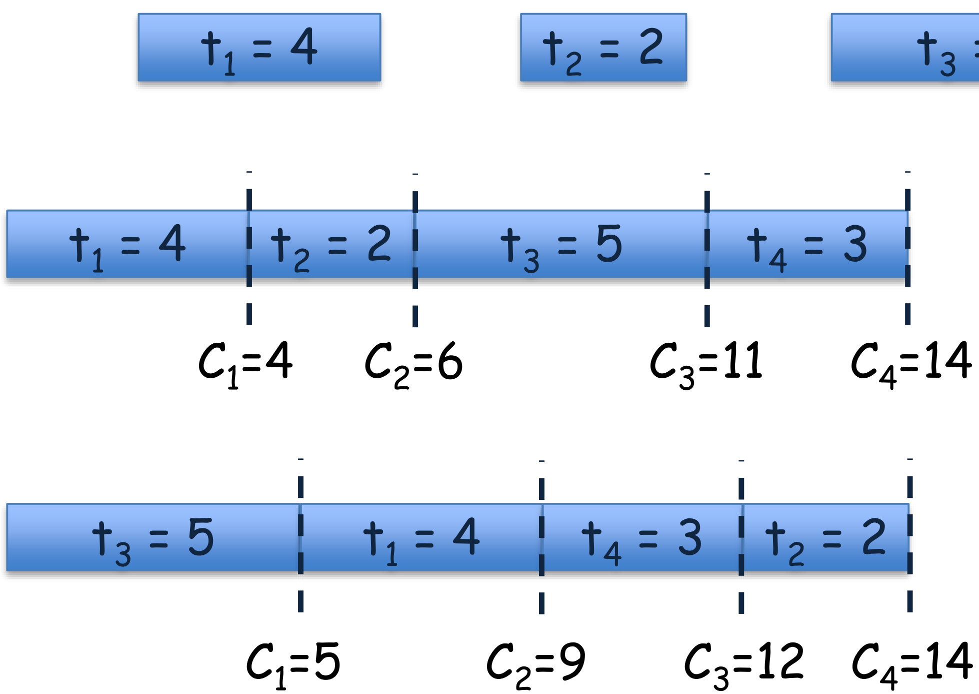


$C^* = (4+6+11+14)/4$

$C^* = 8.75$

Process Scheduling

- Given t_1 , … , t_n
- If we define the finishing time C_i of the process i as $C_i = \Sigma_{j=1..i} t_j$, then the average finishing time will be $(\Sigma_{i=1..n} C_i)/n$.
- Our goal is to minimize $(\Sigma_{i=1..n} C_i)/n$

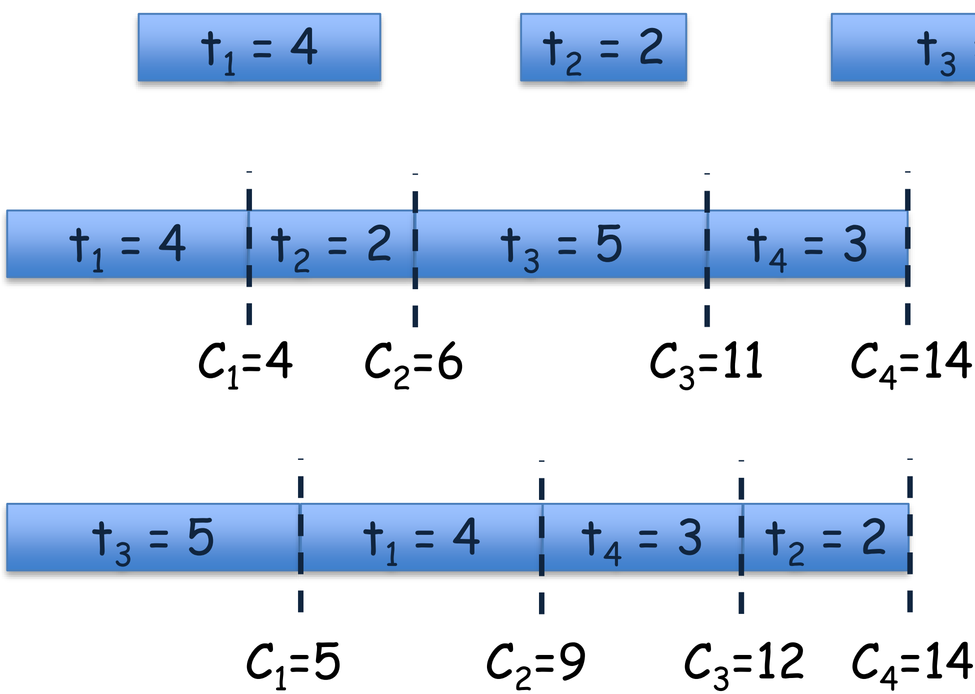


$C^* = (4+6+11+14)/4$

$C^* = 8.75$

Process Scheduling

- Given t_1 , … , t_n
- If we define the finishing time C_i of the process i as $C_i = \Sigma_{j=1..i} t_j$, then the average finishing time will be $(\Sigma_{i=1..n} C_i)/n$.
- Our goal is to minimize $(\Sigma_{i=1..n} C_i)/n$



$C^* = (4+6+11+14)/4$
$C^* = 8.75$

$C^* = (5+9+12+14)/4$
$C^* = 10$

Process Scheduling

- Given t_1 , … , t_n
- If we define the finishing time C_i of the process i as $C_i = \Sigma_{j=1..i} t_j$, then the average finishing time will be $(\Sigma_{i=1..n} C_i)/n$.
- Our goal is to minimize $(\Sigma_{i=1..n} C_i)/n$

this part is constant

Process Scheduling

- Given $t_1 , \dots , t_n$
- If we define the finishing time C_i of the process i as $C_i = \Sigma_{j=1..i} t_j$, then the average finishing time will be $(\Sigma_{i=1..n} C_i)/n$.
- Our goal is to minimize $(\Sigma_{i=1..n} C_i)/n$

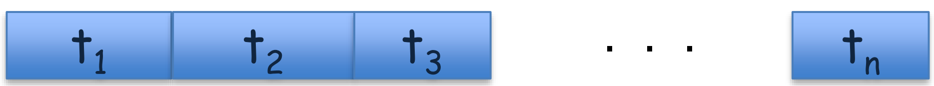
Process Scheduling

- Given t_1 , … , t_n
- If we define the finishing time C_i of the process i as $C_i = \Sigma_{j=1..i} t_j$, then the average finishing time will be $(\Sigma_{i=1..n} C_i)/n$.
- Our goal is to minimize $(\Sigma_{i=1..n} C_i)/n$

$C_1 = t_1$
$C_2 = t_1 + t_2$

Process Scheduling

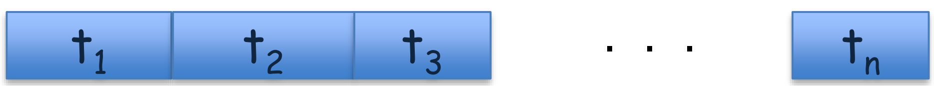
- Given t_1 , … , t_n
- If we define the finishing time C_i of the process i as $C_i = \Sigma_{j=1..i} t_j$, then the average finishing time will be $(\Sigma_{i=1..n} C_i)/n$.
- Our goal is to minimize $(\Sigma_{i=1..n} C_i)/n$



$C_1 = t_1$
$C_2 = t_1 + t_2$
$C_3 = t_1 + t_2 + t_3$
.
.
.
$C_n = t_1 + t_2 + t_3 + . . . + t_n$

Process Scheduling

- Given $t_1, \ldots, t_n$
- If we define the finishing time C_i of the process i as $C_i = \Sigma_{j=1..i} t_j$, then the average finishing time will be $(\Sigma_{i=1..n} C_i)/n$.
- Our goal is to minimize $(\Sigma_{i=1..n} C_i)/n$



$$C_1 = t_1$$
$$C_2 = t_1 + t_2$$
$$C_3 = t_1 + t_2 + t_3$$
$$\vdots$$
$$C_n = t_1 + t_2 + t_3 + \ldots + t_n$$

$$\Sigma_{i=1..n} C_i = n.t_1 + (n-1).t_2 + \ldots + t_n$$

Process Scheduling

- Given t_1 , … , t_n
- If we define the finishing time C_i of the process i as $C_i = \Sigma_{j=1..i} t_j$, then the average finishing time will be $(\Sigma_{i=1..n} C_i)/n$.
- Our goal is to minimize $(\Sigma_{i=1..n} C_i)/n$

t_1	t_2	t_3	. . .	t_n

$$
\begin{aligned}
C_1 &= t_1 \\
C_2 &= t_1 + t_2 \\
C_3 &= t_1 + t_2 + t_3 \\
&\;\;\vdots \\
C_n &= t_1 + t_2 + t_3 + \ldots + t_n
\end{aligned}
$$

$$\Sigma_{i=1..n} C_i = n.t_1 + (n-1).t_2 + \ldots + t_n$$

small t_1 makes the sum smaller

Process Scheduling

- Given t_1 , … , t_n
- If we define the finishing time C_i of the process i as $C_i = \Sigma_{j=1..i} t_j$, then the average finishing time will be $(\Sigma_{i=1..n} C_i)/n$.
- Our goal is to minimize $(\Sigma_{i=1..n} C_i)/n$

t_1	t_2	t_3	. . .	t_n

$$C_1 = t_1$$
$$C_2 = t_1 + t_2$$
$$C_3 = t_1 + t_2 + t_3$$
$$\vdots$$
$$C_n = t_1 + t_2 + t_3 + \ldots + t_n$$

$$\Sigma_{i=1..n} C_i = n.t_1 + (n-1).t_2 + \ldots + t_n$$

Greedy Approach : sort the processes according to the completion time in increasing order

small t_1 makes the sum smaller

Process Scheduling

Theorem (Greedy-choice property): Let S be a sequence of processes ordered according to the completion time. Then S is an optimal sequence. (Our greedy approach yields us an optimal solution)

Proof

Process Scheduling

Theorem (Greedy-choice property): Let S be a sequence of processes ordered according to the completion time. Then S is an optimal sequence. (Our greedy approach yields us an optimal solution)

Proof

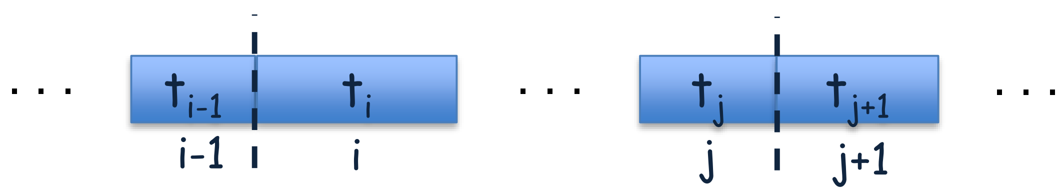
Assume there is an optimal sequence S^* in which the processes have not been sorted.

Process Scheduling

Theorem (Greedy-choice property): Let S be a sequence of processes ordered according to the completion time. Then S is an optimal sequence. (Our greedy approach yields us an optimal solution)

Proof

Assume there is an optimal sequence S^* in which the processes have not been sorted. Then there should be indices i and j such that i < j and $t_j < t_i$



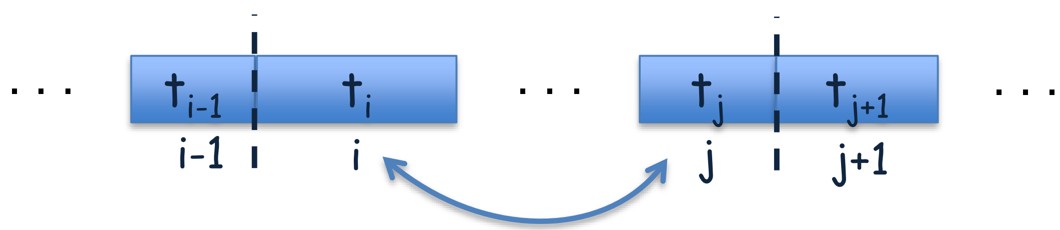
Process Scheduling

Theorem (Greedy-choice property): Let S be a sequence of processes ordered according to the completion time. Then S is an optimal sequence. (Our greedy approach yields us an optimal solution)

Proof

Assume there is an optimal sequence S^* in which the processes have not been sorted. Then there should be indices i and j such that i < j and $t_j < t_i$

What happens if we swap i and j?



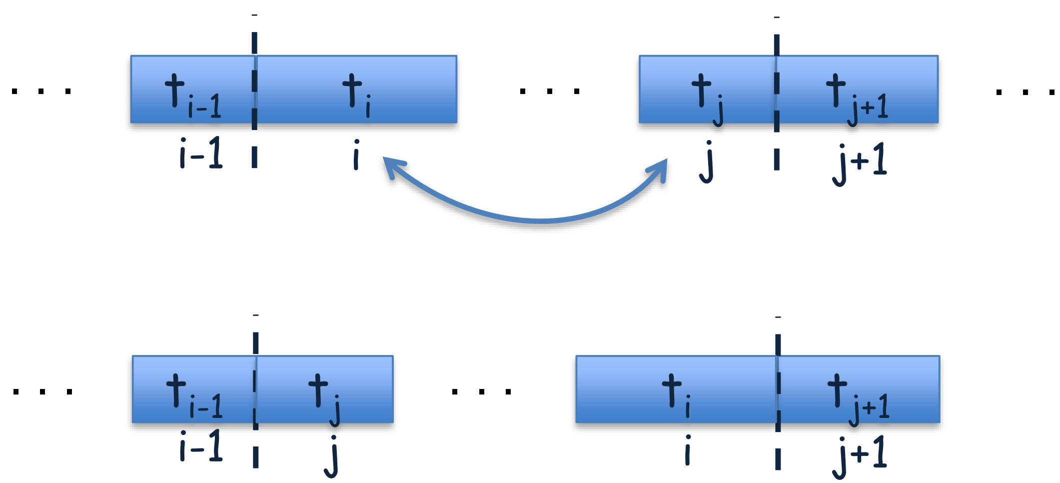
Process Scheduling

Theorem (Greedy-choice property): Let S be a sequence of processes ordered according to the completion time. Then S is an optimal sequence. (Our greedy approach yields us an optimal solution)

Proof

Assume there is an optimal sequence S^* in which the processes have not been sorted. Then there should be indices i and j such that $i < j$ and $t_j < t_i$

What happens if we swap i and j?

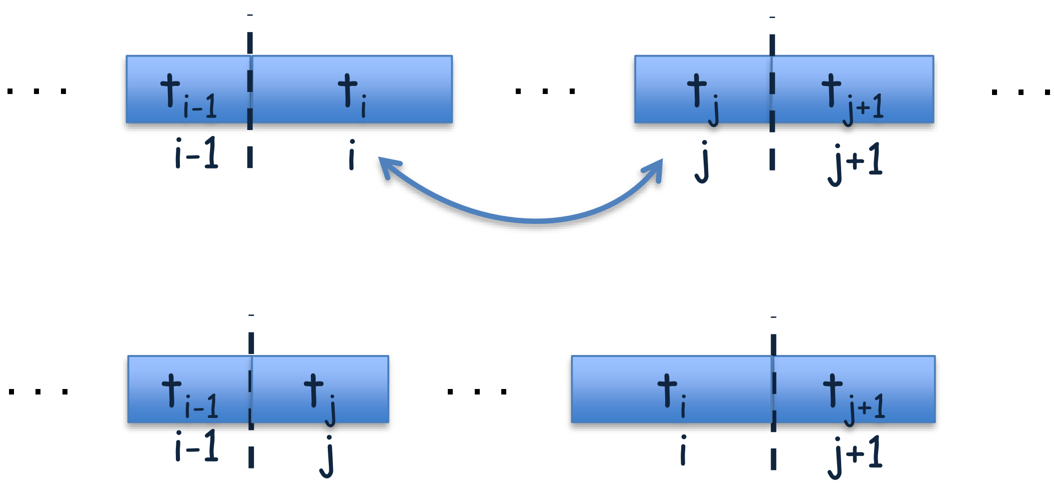


Process Scheduling

Theorem (Greedy-choice property): Let S be a sequence of processes ordered according to the completion time. Then S is an optimal sequence. (Our greedy approach yields us an optimal solution)

Proof

Assume there is an optimal sequence S^* in which the processes have not been sorted. Then there should be indices i and j such that $i < j$ and $t_j < t_i$



What happens if we swap i and j?

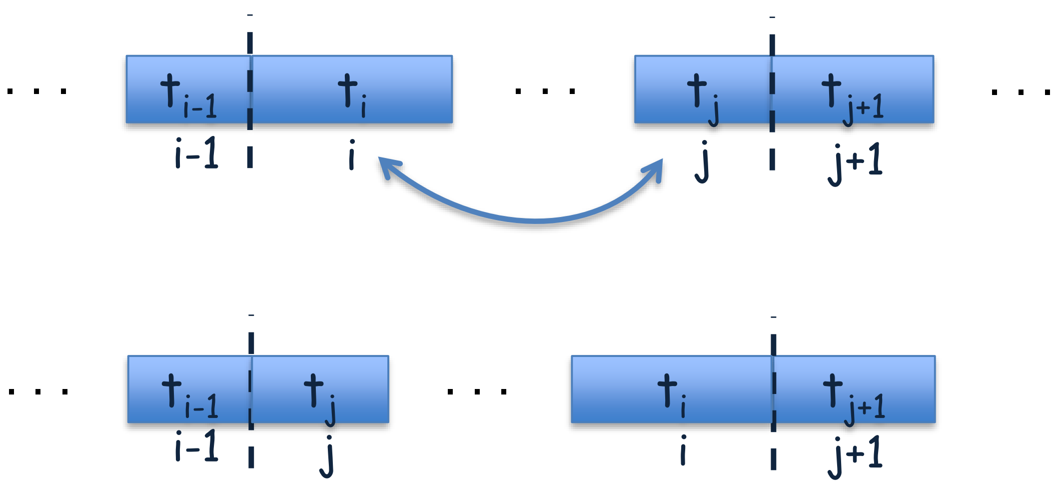
- finishing time of the processes up to i-1 not changing (C_1 , …, C_{i-1} remain same)
- finishing time of the processes after j not changing (C_j , …, C_n remain same)

Process Scheduling

Theorem (Greedy-choice property): Let S be a sequence of processes ordered according to the completion time. Then S is an optimal sequence. (Our greedy approach yields us an optimal solution)

Proof

Assume there is an optimal sequence S^* in which the processes have not been sorted. Then there should be indices i and j such that i < j and $t_j < t_i$



What happens if we swap i and j?

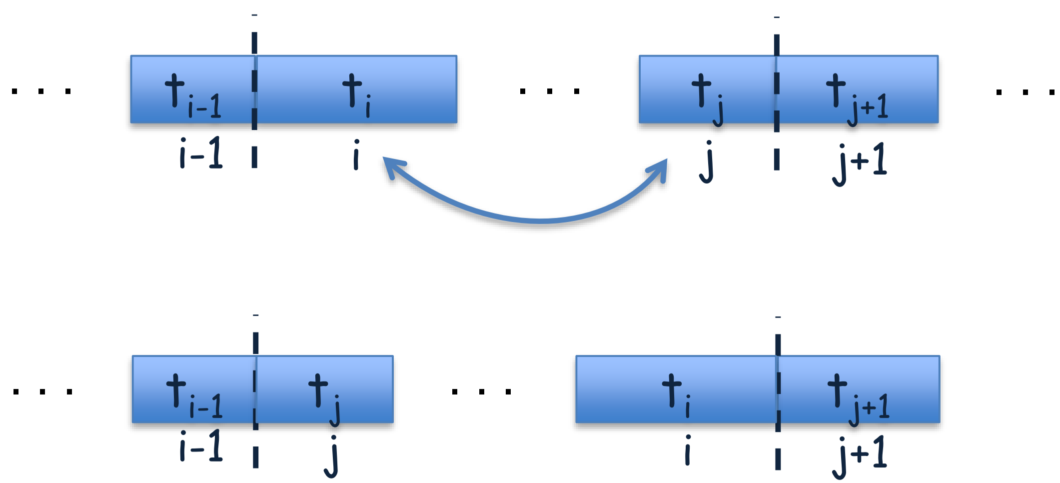
- finishing time of the processes up to i-1 not changing (C_1 , ..., C_{i-1} remain same)
- finishing time of the processes after j not changing (C_j , ..., C_n remain same)
- Let $\Delta = t_i - t_j$.

Process Scheduling

Theorem (Greedy-choice property): Let S be a sequence of processes ordered according to the completion time. Then S is an optimal sequence. (Our greedy approach yields us an optimal solution)

Proof

Assume there is an optimal sequence S^* in which the processes have not been sorted. Then there should be indices i and j such that i < j and $t_j < t_i$



What happens if we swap i and j?

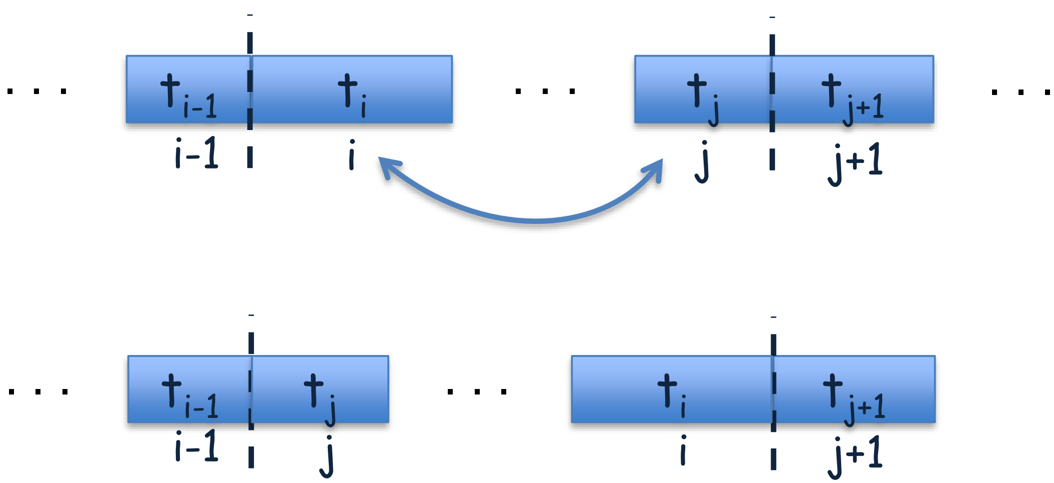
- finishing time of the processes up to i-1 not changing (C_1 , …, C_{i-1} remain same)
- finishing time of the processes after j not changing (C_j , …, C_n remain same)
- Let $\Delta = t_i - t_j$. Then $C_i^* = C_i - \Delta$

Process Scheduling

Theorem (Greedy-choice property): Let S be a sequence of processes ordered according to the completion time. Then S is an optimal sequence. (Our greedy approach yields us an optimal solution)

Proof

Assume there is an optimal sequence S^* in which the processes have not been sorted. Then there should be indices i and j such that i < j and $t_j < t_i$



What happens if we swap i and j?

- finishing time of the processes up to i-1 not changing (C_1 , …, C_{i-1} remain same)
- finishing time of the processes after j not changing (C_j , …, C_n remain same)
- Let $\Delta = t_i - t_j$. Then

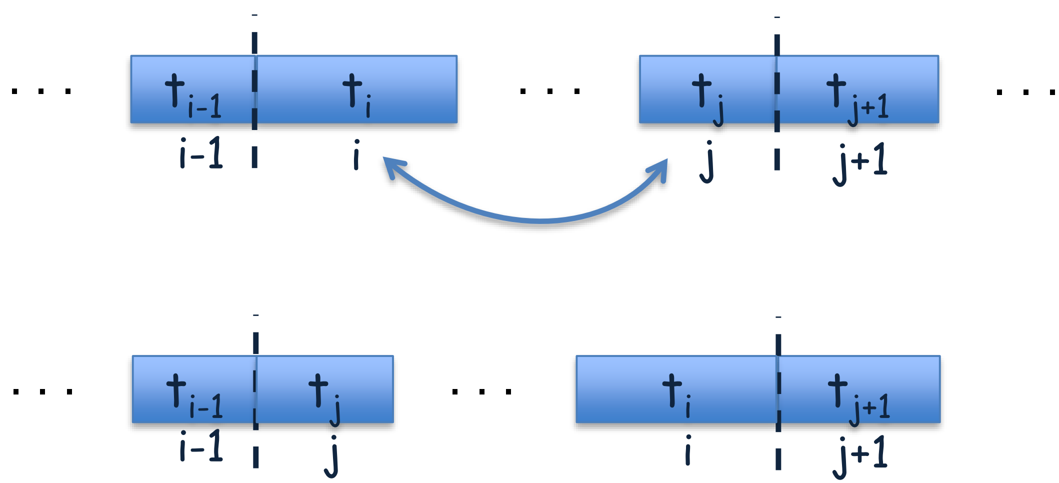
$$C_i^* = C_i - \Delta$$
$$C_{i+1}^* = C_{i+1} - \Delta$$
$$\vdots$$
$$C_{j-1}^* = C_{j-1} - \Delta$$

Process Scheduling

Theorem (Greedy-choice property): Let S be a sequence of processes ordered according to the completion time. Then S is an optimal sequence. (Our greedy approach yields us an optimal solution)

Proof

Assume there is an optimal sequence S^* in which the processes have not been sorted. Then there should be indices i and j such that i < j and $t_j < t_i$



What happens if we swap i and j?

- finishing time of the processes up to i-1 not changing (C_1 , …, C_{i-1} remain same)
- finishing time of the processes after j not changing (C_j , …, C_n remain same)
- Let $\Delta = t_i - t_j$. Then

$$C_i^* = C_i - \Delta$$
$$C_{i+1}^* = C_{i+1} - \Delta$$
$$\vdots$$
$$C_{j-1}^* = C_{j-1} - \Delta$$

- $\Sigma_{i=1..n}\, C_i$ decreasing

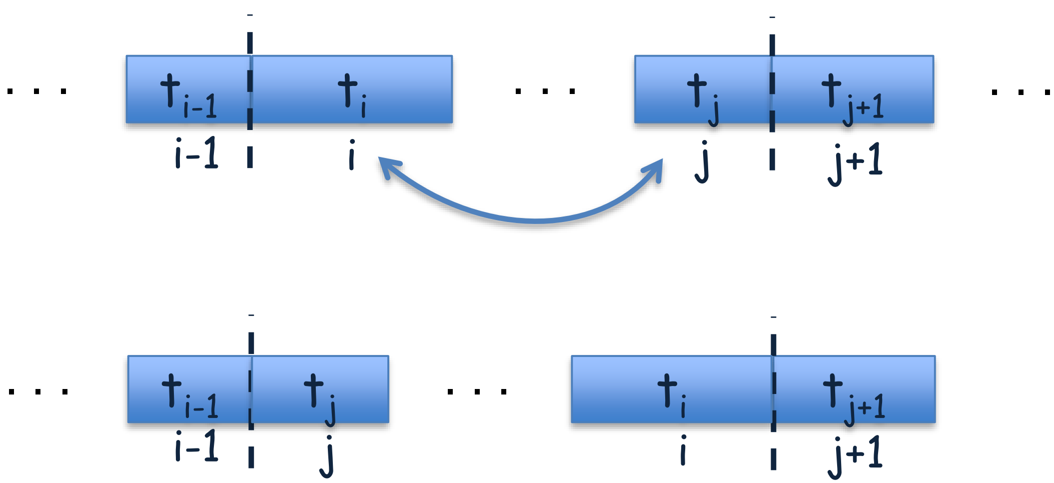
Process Scheduling

Theorem (Greedy-choice property): Let S be a sequence of processes ordered according to the completion time. Then S is an optimal sequence. (Our greedy approach yields us an optimal solution)

Proof

Assume there is an optimal sequence S^* in which the processes have not been sorted. Then there should be indices i and j such that i < j and $t_j < t_i$

What happens if we swap i and j?



- finishing time of the processes up to i-1 not changing (C_1 , …, C_{i-1} remain same)
- finishing time of the processes after j not changing (C_j , …, C_n remain same)
- Let $\Delta = t_i - t_j$. Then

$$C_i^* = C_i - \Delta$$
$$C_{i+1}^* = C_{i+1} - \Delta$$
$$\vdots$$
$$C_{j-1}^* = C_{j-1} - \Delta$$

this is a contradiction ← • $\Sigma_{i=1..n} C_i$ decreasing

Minimizing Lateness

- given a computer and n processes $p_1, \ldots, p_n$ such that each of them has a processing time t_i and a deadline d_i
- find an optimal order of processes that minimizes the maximum lateness

Minimizing Lateness

- given a computer and n processes $p_1, \dots, p_n$ such that each of them has a processing time t_i and a deadline d_i
- find an optimal order of processes that minimizes the maximum lateness

If we define the finishing time f_i of the process i as $f_i = \Sigma_{j=1..i} t_j$, then the lateness of the process i will be $l_i = \max\{0, f_i - d_i\}$

Minimizing Lateness

- given a computer and n processes $p_1, \ldots, p_n$ such that each of them has a processing time t_i and a deadline d_i
- find an optimal order of processes that minimizes the maximum lateness

If we define the finishing time f_i of the process i as $f_i = \Sigma_{j=1..i} t_j$, then the lateness of the process i will be $l_i = \max\{0, f_i - d_i\}$

Our goal is to minimize $L = \max_i l_i$

Minimizing Lateness

- Given t_1 , … , t_n
- If we define the finishing time f_i of the process i as $f_i = \Sigma_{j=1..i} t_j$, then the lateness of the process i will be $l_i = \max\{0, f_i - d_i\}$
- Our goal is to minimize $L = \max_i l_i$

Minimizing Lateness

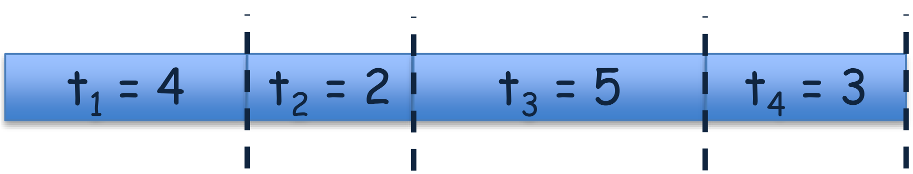
- Given t_1 , … , t_n
- If we define the finishing time f_i of the process i as $f_i = \Sigma_{j=1..i} t_j$, then the lateness of the process i will be $l_i = \max \{0, f_i - d_i\}$
- Our goal is to minimize $L = \max_i l_i$

	1	**2**	**3**	**4**
t_i	4	2	5	3
d_i	6	4	10	8

Minimizing Lateness

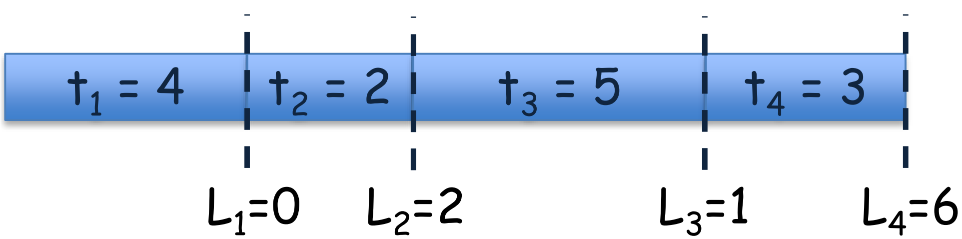
- Given t_1 , … , t_n
- If we define the finishing time f_i of the process i as $f_i = \Sigma_{j=1..i} t_j$, then the lateness of the process i will be $l_i = \max\{0, f_i - d_i\}$
- Our goal is to minimize $L = \max_i l_i$

	1	**2**	**3**	**4**
t_i	4	2	5	3
d_i	6	4	10	8



Minimizing Lateness

- Given t_1 , … , t_n
- If we define the finishing time f_i of the process i as $f_i = \Sigma_{j=1..i} t_j$, then the lateness of the process i will be $l_i = \max\{0, f_i - d_i\}$
- Our goal is to minimize $L = \max_i l_i$



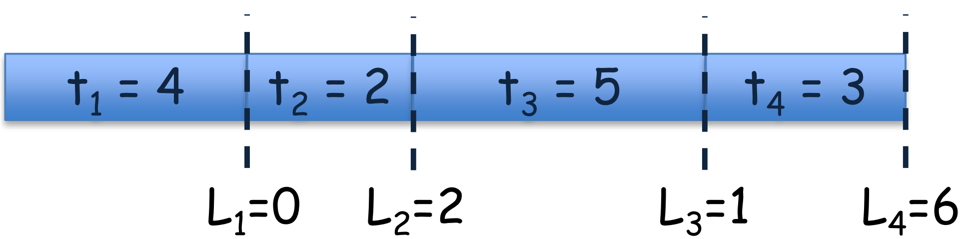
	1	**2**	**3**	**4**
t_i	4	2	5	3
d_i	6	4	10	8

$L = 6$

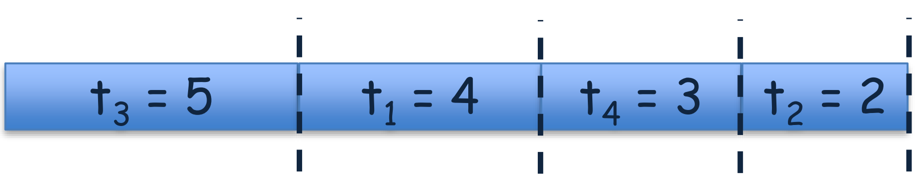
Minimizing Lateness

- Given $t_1, \ldots, t_n$
- If we define the finishing time f_i of the process i as $f_i = \Sigma_{j=1..i} t_j$, then the lateness of the process i will be $l_i = \max\{0, f_i - d_i\}$
- Our goal is to minimize $L = \max_i l_i$

	1	**2**	**3**	**4**
t_i	4	2	5	3
d_i	6	4	10	8



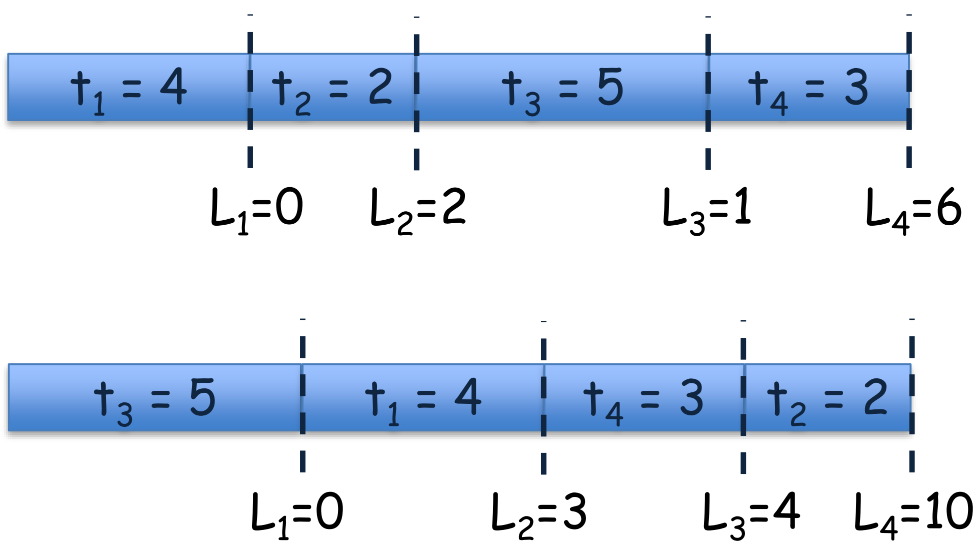
$L = 6$



Minimizing Lateness

- Given $t_1, \dots, t_n$
- If we define the finishing time f_i of the process i as $f_i = \Sigma_{j=1..i} t_j$, then the lateness of the process i will be $l_i = \max\{0, f_i - d_i\}$
- Our goal is to minimize $L = \max_i l_i$

	1	2	3	4
t_i	4	2	5	3
d_i	6	4	10	8



$L = 6$

$L = 10$

Minimizing Lateness

How do we sort the processes ?

- according to their process time t_i

Minimizing Lateness

How do we sort the processes ?

- according to their process time t_i

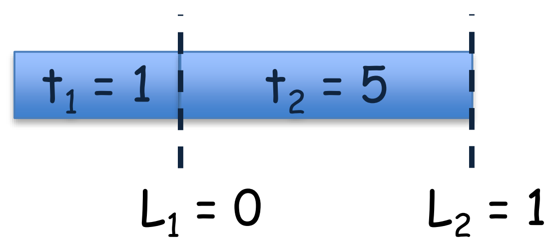
	1	**2**
t_i	1	5
d_i	12	5

Minimizing Lateness

How do we sort the processes ?

- according to their process time t_i

	1	**2**
t_i	1	5
d_i	12	5



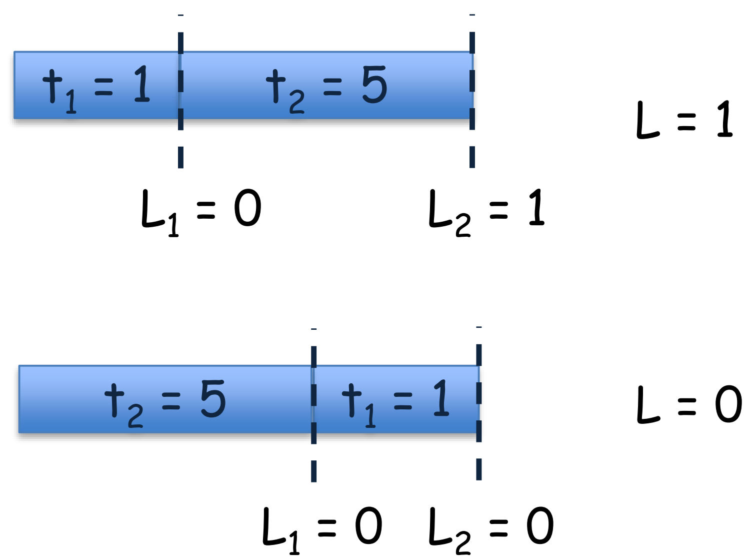
$L = 1$

Minimizing Lateness

How do we sort the processes ?

- according to their process time t_i

	1	**2**
$\mathbf{t_i}$	1	5
$\mathbf{d_i}$	12	5



Minimizing Lateness

How do we sort the processes ?

- according to their slack time $d_i - t_i$

Minimizing Lateness

How do we sort the processes ?

- according to their slack time $d_i - t_i$

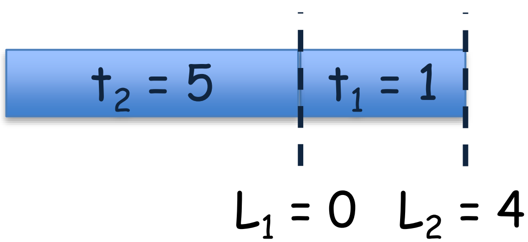
	1	2
t_i	1	5
d_i	2	5

Minimizing Lateness

How do we sort the processes ?

- according to their slack time d_i - t_i

	1	**2**
t_i	1	5
d_i	2	5



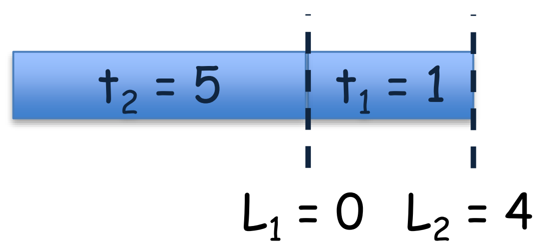
L = 4

Minimizing Lateness

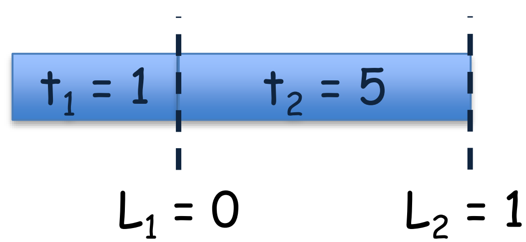
How do we sort the processes ?

- according to their slack time $d_i - t_i$

	1	2
t_i	1	5
d_i	2	5



$L = 4$



$L = 1$

Minimizing Lateness

How do we sort the processes ?

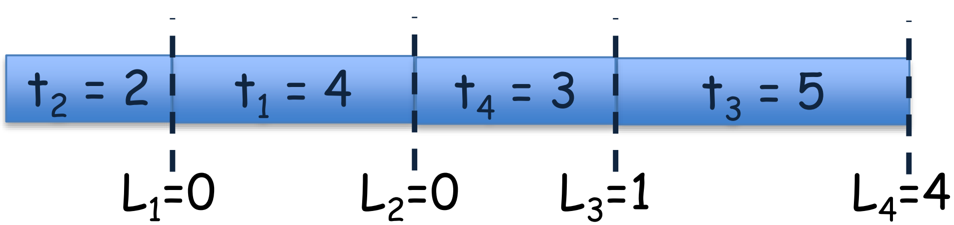
- according to their deadlines d_i

Minimizing Lateness

How do we sort the processes ?

- according to their deadlines d_i

	1	2	3	4
t_i	4	2	5	3
d_i	6	4	10	8



$L = 4$

Minimizing Lateness

Theorem (Greedy-choice property): Let S be a sequence of processes ordered according to the deadline. Then S is an optimal sequence. (Our greedy approach yields us an optimal solution)

Minimizing Lateness

Theorem (Greedy-choice property): Let S be a sequence of processes ordered according to the deadline. Then S is an optimal sequence. (Our greedy approach yields us an optimal solution)

Proof

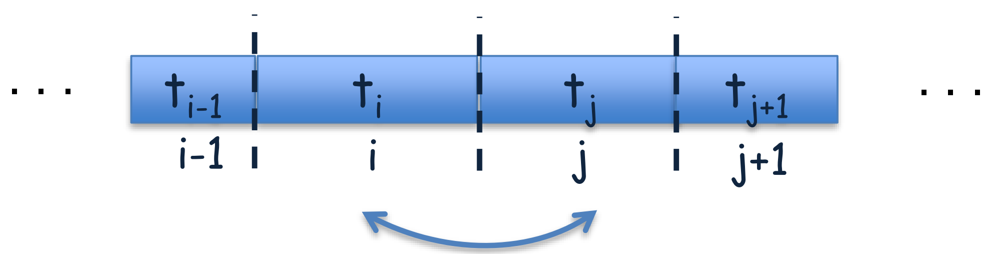
Assume there is an optimal sequence S^* in which the processes have not been sorted. Then there should be indices i and j such that $i < j$ and $d_j < d_i$

Minimizing Lateness

Theorem (Greedy-choice property): Let S be a sequence of processes ordered according to the deadline. Then S is an optimal sequence. (Our greedy approach yields us an optimal solution)

Proof

Assume there is an optimal sequence S^* in which the processes have not been sorted. Then there should be indices i and j such that i < j and $d_j < d_i$



What happens if we swap i and j?

Huffman Coding

Encoding

ABRACADABRA

A B R A C A D A B R A

Huffman Coding

Encoding

ABRACADABRA

A	B	R	A	C	A	D	A	B	R	A
000	001	010	000	011	000	100	000	001	010	000

- Encode them using 3-bit strings

Huffman Coding

Encoding

ABRACADABRA

A	B	R	A	C	A	D	A	B	R	A
000	001	010	000	011	000	100	000	001	010	000

freq

A
B
R
C
D

number of times
the letter appears

- Encode them using 3-bit strings

Huffman Coding

Encoding

ABRACADABRA

A	B	R	A	C	A	D	A	B	R	A
000	001	010	000	011	000	100	000	001	010	000

freq

A	5
B	2
R	2
C	1
D	1

- Encode them using 3-bit strings

Huffman Coding

Encoding

ABRACADABRA

A	B	R	A	C	A	D	A	B	R	A
000	001	010	000	011	000	100	000	001	010	000

	freq	
A	5	000
B	2	001
R	2	010
C	1	011
D	1	100

- Encode them using 3-bit strings

Huffman Coding

Encoding

ABRACADABRA

A	B	R	A	C	A	D	A	B	R	A
000	001	010	000	011	000	100	000	001	010	000

	freq		cost
A	5	000	15
B	2	001	6
R	2	010	6
C	1	011	3
D	1	100	3

- Encode them using 3-bit strings

Huffman Coding

Encoding

ABRACADABRA

A	B	R	A	C	A	D	A	B	R	A
000	001	010	000	011	000	100	000	001	010	000

	freq		cost
A	5	000	15
B	2	001	6
R	2	010	6
C	1	011	3
D	1	100	3
			33

- Encode them using 3-bit strings
- Total 33 bits required to encode

Huffman Coding

Encoding

ABRACADABRA

A	B	R	A	C	A	D	A	B	R	A
000	001	010	000	011	000	100	000	001	010	000

	freq		cost
A	5	000	15
B	2	001	6
R	2	010	6
C	1	011	3
D	1	100	3
			33

- Encode them using 3-bit strings
- Total 33 bits required to encode

Can we get better encoding?

Huffman Coding

Encoding

ABRACADABRA

A	B	R	A	C	A	D	A	B	R	A
000	001	010	000	011	000	100	000	001	010	000

	freq		cost
A	5	000	15
B	2	001	6
R	2	010	6
C	1	011	3
D	1	100	3
			33

- Encode them using 3-bit strings
- Total 33 bits required to encode

Can we get better encoding?

1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0

Huffman Coding

Encoding

ABRACADABRA

A	B	R	A	C	A	D	A	B	R	A
000	001	010	000	011	000	100	000	001	010	000

	freq		cost
A	5	000	15
B	2	001	6
R	2	010	6
C	1	011	3
D	1	100	3
			33

- Encode them using 3-bit strings
- Total 33 bits required to encode

Can we get better encoding?

1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0

Huffman Coding

Encoding

ABRACADABRA

A	B	R	A	C	A	D	A	B	R	A
000	001	010	000	011	000	100	000	001	010	000

	freq		cost
A	5	000	15
B	2	001	6
R	2	010	6
C	1	011	3
D	1	100	3
			33

- Encode them using 3-bit strings
- Total 33 bits required to encode

Can we get better encoding?

1 0 0	0 0 0	0 0 1	0 0 0	0 1 0	0 0 0
D	A	B	A	R	A

Huffman Coding

Encoding

ABRACADABRA

A	B	R	A	C	A	D	A	B	R	A
000	001	010	000	011	000	100	000	001	010	000

	freq		cost
A	5	000	15
B	2	001	6
R	2	010	6
C	1	011	3
D	1	100	3
			33

- Encode them using 3-bit strings
- Total 33 bits required to encode

Can we get better encoding?

1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0

D A B A R A

a unique decoding

Huffman Coding

	freq
A	5
B	2
R	2
C	1
D	1

Huffman Coding

	freq
A	5
B	2
R	2
C	1
D	1

- $B(T, \{ f_c \}) = \Sigma f_c . l_c$

Huffman Coding

	freq
A	5
B	2
R	2
C	1
D	1

- $B(T, \{ f_c \}) = \Sigma f_c . l_c$
- try to minimize the function B

Huffman Coding

	freq
A	5
B	2
R	2
C	1
D	1

- $B(T, \{ f_c \}) = \Sigma f_c . l_c$
- try to minimize the function B
- use smaller length encoding for the character having larger frequency

Huffman Coding

	freq	
A	5	0
B	2	1
R	2	01
C	1	10
D	1	11

- $B(T, \{ f_c \}) = \Sigma f_c.l_c$
- try to minimize the function B
- use smaller length encoding for the character having larger frequency

Huffman Coding

	freq		cost
A	5	0	5
B	2	1	2
R	2	01	4
C	1	10	2
D	1	11	2
			15

- $B(T, \{ f_c \}) = \Sigma f_c.l_c$
- try to minimize the function B
- use smaller length encoding for the character having larger frequency

Huffman Coding

	freq		cost
A	5	0	5
B	2	1	2
R	2	01	4
C	1	10	2
D	1	11	2
			15

- $B(T, \{ f_c \}) = \Sigma f_c . l_c$
- try to minimize the function B
- use smaller length encoding for the character having larger frequency

Is there any problem for this encoding?

Huffman Coding

	freq		cost
A	5	0	5
B	2	1	2
R	2	01	4
C	1	10	2
D	1	11	2
			15

- $B(T, \{ f_c \}) = \Sigma f_c . l_c$
- try to minimize the function B
- use smaller length encoding for the character having larger frequency

Is there any problem for this encoding?

0 0 1 1 0 1 1 0

Huffman Coding

	freq		cost
A	5	0	5
B	2	1	2
R	2	01	4
C	1	10	2
D	1	11	2
			15

- $B(T, \{ f_c \}) = \Sigma f_c . l_c$
- try to minimize the function B
- use smaller length encoding for the character having larger frequency

Is there any problem for this encoding?

0 0 1 1 0 1 1 0

Huffman Coding

	freq		cost
A	5	0	5
B	2	1	2
R	2	01	4
C	1	10	2
D	1	11	2
			15

- $B(T, \{ f_c \}) = \Sigma f_c.l_c$
- try to minimize the function B
- use smaller length encoding for the character having larger frequency

Is there any problem for this encoding?

0 | 0 | 1 1 | 0 1 | 1 0

A A D R C

Huffman Coding

	freq		cost
A	5	0	5
B	2	1	2
R	2	01	4
C	1	10	2
D	1	11	2
			15

- $B(T, \{ f_c \}) = \Sigma f_c . l_c$
- try to minimize the function B
- use smaller length encoding for the character having larger frequency

Is there any problem for this encoding?

0 | 0 1 | 1 | 0 1 | 1 0

A R B R C

AADRC

Huffman Coding

	freq		cost
A	5	0	5
B	2	1	2
R	2	01	4
C	1	10	2
D	1	11	2
			15

- $B(T, \{ f_c \}) = \Sigma f_c . l_c$
- try to minimize the function B
- use smaller length encoding for the character having larger frequency

Is there any problem for this encoding?

0 0 1 1 0 1 1 0

A A B B A D A

AADRC

ARBRC

Huffman Coding

	freq		cost
A	5	0	5
B	2	1	2
R	2	01	4
C	1	10	2
D	1	11	2
			15

- $B(T, \{ f_c \}) = \Sigma f_c . l_c$
- try to minimize the function B
- use smaller length encoding for the character having larger frequency

Is there any problem for this encoding?

0 0 1 1 0 1 1 0

A A BBA D A

AADRC
ARBRC
AABBADA

Huffman Coding

	freq		cost
A	5	0	5
B	2	1	2
R	2	01	4
C	1	10	2
D	1	11	2
			15

- $B(T, \{ f_c \}) = \Sigma f_c . l_c$
- try to minimize the function B
- use smaller length encoding for the character having larger frequency

Is there any problem for this encoding?

0 0 1 1 0 1 1 0

A A BBA D A

AADRC
ARBRC
AABBADA
.
.
.

Huffman Coding

	freq		cost
A	5	0	5
B	2	1	2
R	2	01	4
C	1	10	2
D	1	11	2
			15

- $B(T, \{ f_c \}) = \Sigma f_c . l_c$
- try to minimize the function B
- use smaller length encoding for the character having larger frequency

Is there any problem for this encoding?

0 0 1 1 0 1 1 0

A A BBA D A

decoding is not unique

AADRC

ARBRC

AABBADA

.

.

.

Huffman Coding

	freq		cost
A	5	0	5
B	2	1	2
R	2	01	4
C	1	10	2
D	1	11	2
			15

- $B(T, \{ f_c \}) = \Sigma f_c . l_c$
- try to minimize the function B
- use smaller length encoding for the character having larger frequency

Is there any problem for this encoding?

0 | 0 | 1 | 1 | 0 | 1 1 | 0

A A BBA D A

AADRC
ARBRC
AABBADA
.
.
.

decoding is not unique

to get unique decoding, coding should be 'prefix-free'

Huffman Coding

	freq		cost
A	5	0	5
B	2	1	2
R	2	01	4
C	1	10	2
D	1	11	2
			15

- $B(T, \{ f_c \}) = \Sigma f_c.l_c$
- try to minimize the function B
- use smaller length encoding for the character having larger frequency

to get unique decoding, coding should be 'prefix-free'

Huffman Coding

	freq		cost
A	5	0	5
B	2	1	2
R	2	01	4
C	1	10	2
D	1	11	2
			15

- $B(T, \{ f_c \}) = \Sigma\, f_c.l_c$
- try to minimize the function B
- use smaller length encoding for the character having larger frequency

to get unique decoding, coding should be 'prefix-free'

Coding is called 'prefix free' if for any i, j; encoding c_i is not prefix of encoding c_j

Huffman Coding

	freq		cost
A	5	0	5
B	2	1	2
R	2	01	4
C	1	10	2
D	1	11	2
			15

- $B(T, \{ f_c \}) = \Sigma f_c . l_c$
- try to minimize the function B
- use smaller length encoding for the character having larger frequency

to get unique decoding, coding should be 'prefix-free'

Coding is called 'prefix free' if for any i, j; encoding c_i is not prefix of encoding c_j

encoding of B -1- is prefix of encoding of C -10-

1 0

Huffman Coding

	freq		cost
A	5	0	5
B	2	1	2
R	2	01	4
C	1	10	2
D	1	11	2
			15

- $B(T, \{ f_c \}) = \Sigma f_c . l_c$
- try to minimize the function B
- use smaller length encoding for the character having larger frequency

to get unique decoding, coding should be 'prefix-free'

Coding is called 'prefix free' if for any i, j; encoding c_i is not prefix of encoding c_j

encoding of B -1- is prefix of encoding of C -10-

1 0

1 0 = C

Huffman Coding

	freq		cost
A	5	0	5
B	2	1	2
R	2	01	4
C	1	10	2
D	1	11	2
			15

- $B(T, \{ f_c \}) = \Sigma f_c . l_c$
- try to minimize the function B
- use smaller length encoding for the character having larger frequency

to get unique decoding, coding should be 'prefix-free'

Coding is called 'prefix free' if for any i, j; encoding c_i is not prefix of encoding c_j

encoding of B -1- is prefix of encoding of C -10-

1 0

1 0 = C

1 | 0 = BA

Huffman Coding

	freq
A	25
E	18
M	13
B	10
K	7
T	5
U	2
L	1

- If you have a prefix-free code, you can uniquely decode it

Huffman Coding

	freq
A	25
E	18
M	13
B	10
K	7
T	5
U	2
L	1

- If you have a prefix-free code, you can uniquely decode it
- encoding for each char ends with '0'
- use different length encoding for each char

Huffman Coding

	freq	
A	25	0
E	18	10
M	13	110
B	10	1110
K	7	11110
T	5	111110
U	2	1111110
L	1	11111110

- If you have a prefix-free code, you can uniquely decode it
- encoding for each char ends with '0'
- use different length encoding for each char

Huffman Coding

	freq		cost
A	25	0	25
E	18	10	36
M	13	110	39
B	10	1110	40
K	7	11110	35
T	5	111110	30
U	2	1111110	14
L	1	11111110	8
			227

- If you have a prefix-free code, you can uniquely decode it
- encoding for each char ends with '0'
- use different length encoding for each char

Huffman Coding

	freq		cost
A	25	0	25
E	18	10	36
M	13	110	39
B	10	1110	40
K	7	11110	35
T	5	111110	30
U	2	1111110	14
L	1	11111110	8
			227

- If you have a prefix-free code, you can uniquely decode it
- encoding for each char ends with '0'
- use different length encoding for each char

1 1 0 0 1 1 1 1 0 0 1 1 1 1 1 1 1 0 1 0

Huffman Coding

	freq		cost
A	25	0	25
E	18	10	36
M	13	110	39
B	10	1110	40
K	7	11110	35
T	5	111110	30
U	2	1111110	14
L	1	11111110	8
			227

- If you have a prefix-free code, you can uniquely decode it
- encoding for each char ends with '0'
- use different length encoding for each char

1 1 0 | 0 | 1 1 1 1 0 | 0 | 1 1 1 1 1 1 1 0 | 1 0

M A K A L E

Huffman Coding

	freq		cost
A	25	0	25
E	18	10	36
M	13	110	39
B	10	1110	40
K	7	11110	35
T	5	111110	30
U	2	1111110	14
L	1	11111110	8
			227

- If you have a prefix-free code, you can uniquely decode it
- encoding for each char ends with '0'
- use different length encoding for each char

1 1 0 0 1 1 1 1 0 0 1 1 1 1 1 1 1 0 1 0

M A K A L E

transform this encoding to a binary tree

Huffman Coding

	freq		cost
A	25	0	25
E	18	10	36
M	13	110	39
B	10	1110	40
K	7	11110	35
T	5	111110	30
U	2	1111110	14
L	1	11111110	8
			227

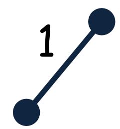
- for '1', create a left child
  for '0', create a right child

transform this encoding to a binary tree

Huffman Coding

	freq		cost
A	25	0	25
E	18	10	36
M	13	110	39
B	10	1110	40
K	7	11110	35
T	5	111110	30
U	2	1111110	14
L	1	11111110	8
			227

- for '1', create a left child
  for '0', create a right child

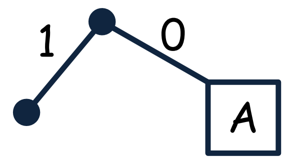


transform this encoding to a binary tree

Huffman Coding

	freq		cost
A	25	0	25
E	18	10	36
M	13	110	39
B	10	1110	40
K	7	11110	35
T	5	111110	30
U	2	1111110	14
L	1	11111110	8
			227

- for '1', create a left child
  for '0', create a right child

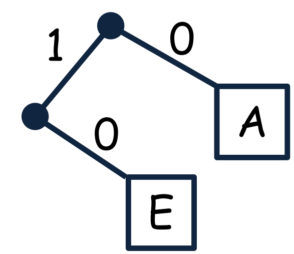


transform this encoding to a binary tree

Huffman Coding

	freq		cost
A	25	0	25
E	18	10	36
M	13	110	39
B	10	1110	40
K	7	11110	35
T	5	111110	30
U	2	1111110	14
L	1	11111110	8
			227

- for '1', create a left child
  for '0', create a right child

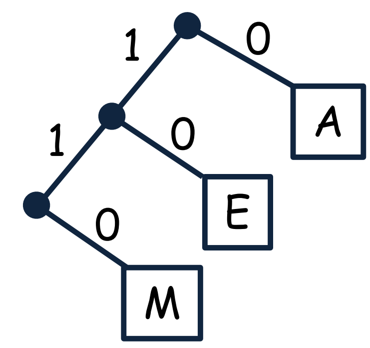


transform this encoding to a binary tree

Huffman Coding

	freq		cost
A	25	0	25
E	18	10	36
M	13	110	39
B	10	1110	40
K	7	11110	35
T	5	111110	30
U	2	1111110	14
L	1	11111110	8
			227

- for '1', create a left child
  for '0', create a right child

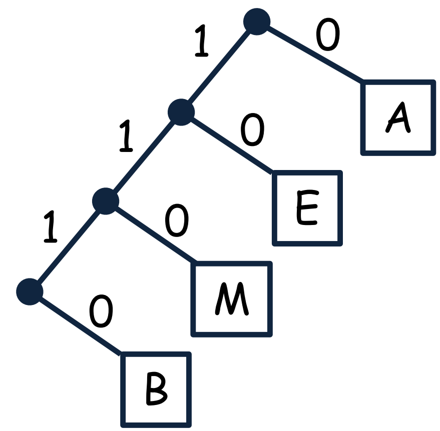


transform this encoding to a binary tree

Huffman Coding

	freq		cost
A	25	0	25
E	18	10	36
M	13	110	39
B	10	1110	40
K	7	11110	35
T	5	111110	30
U	2	1111110	14
L	1	11111110	8
			227

- for '1', create a left child
  for '0', create a right child

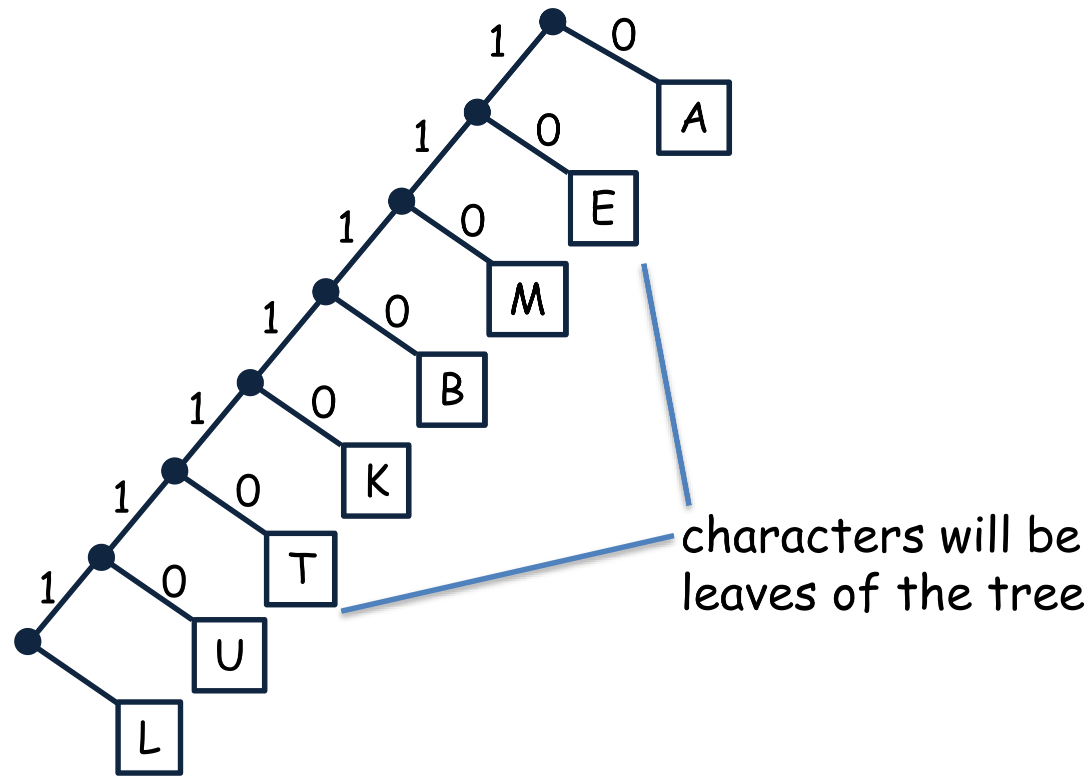


transform this encoding to a binary tree

Huffman Coding

	freq		cost
A	25	0	25
E	18	10	36
M	13	110	39
B	10	1110	40
K	7	11110	35
T	5	111110	30
U	2	1111110	14
L	1	11111110	8
			227

- for '1', create a left child
  for '0', create a right child

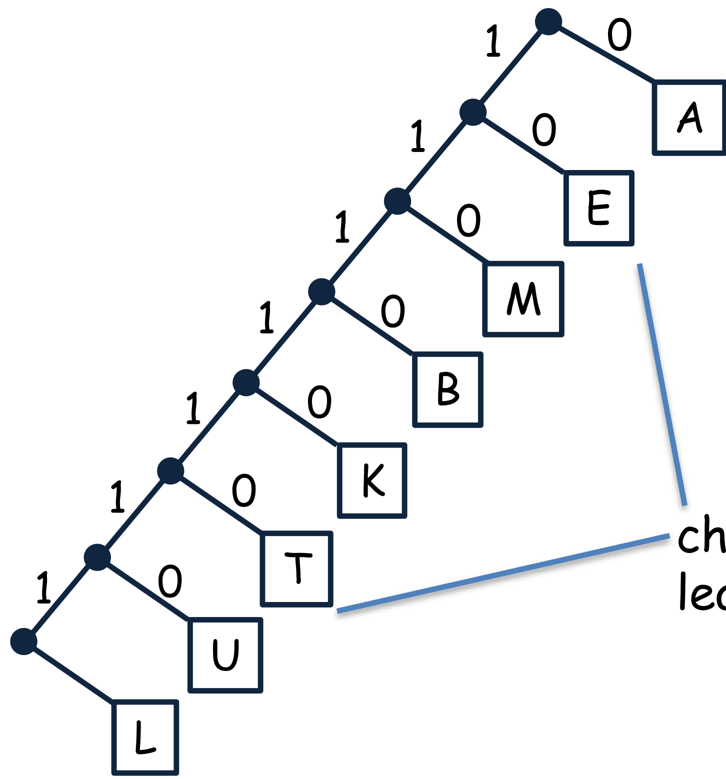


transform this encoding to a binary tree

Huffman Coding

	freq		cost
A	25	0	25
E	18	10	36
M	13	110	39
B	10	1110	40
K	7	11110	35
T	5	111110	30
U	2	1111110	14
L	1	11111110	8
			227

- for '1', create a left child
  for '0', create a right child



characters will be leaves of the tree

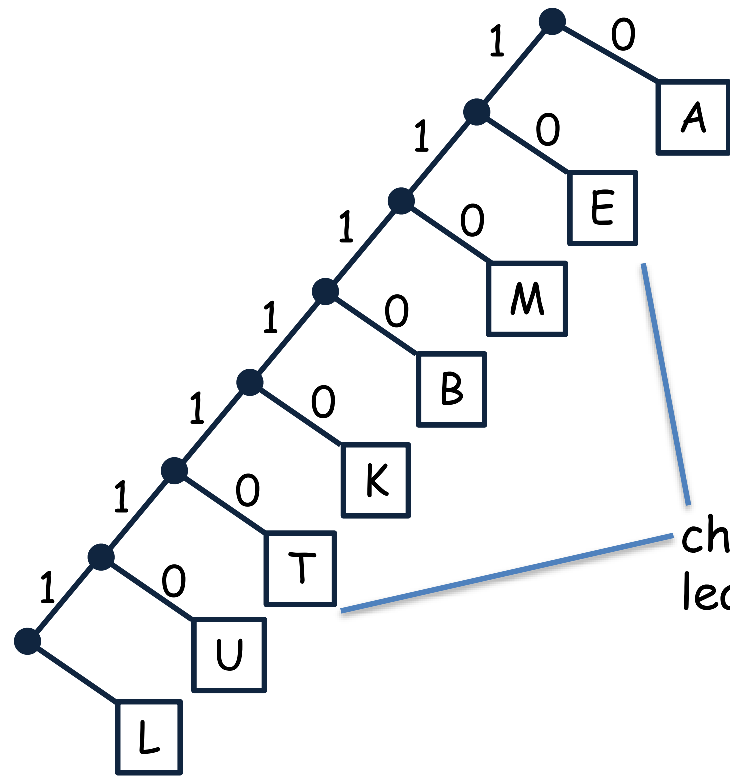
to get an optimal encoding, create an optimal tree

transform this encoding to a binary tree

Huffman Coding

	freq		cost
A	25	0	25
E	18	10	36
M	13	110	39
B	10	1110	40
K	7	11110	35
T	5	111110	30
U	2	1111110	14
L	1	11111110	8
			227

- for '1', create a left child
  for '0', create a right child



characters will be leaves of the tree

to get an optimal encoding, create an optimal tree

optimal encoding = optimal tree

transform this encoding to a binary tree

Huffman Coding

Lemma : Optimal tree is full.
(every node has either two children or no child)

Proof :

Huffman Coding

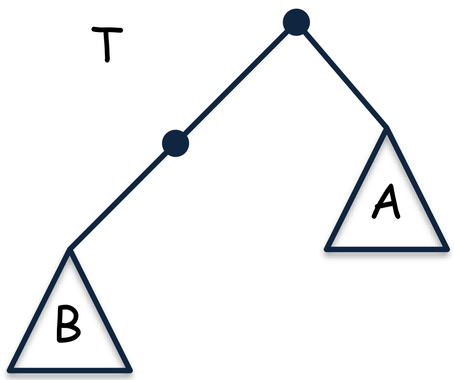
Lemma : Optimal tree is full.
(every node has either two children or no child)

Proof : Suppose there is an optimal tree T having one node with one child.

Huffman Coding

Lemma : Optimal tree is full.
(every node has either two children or no child)

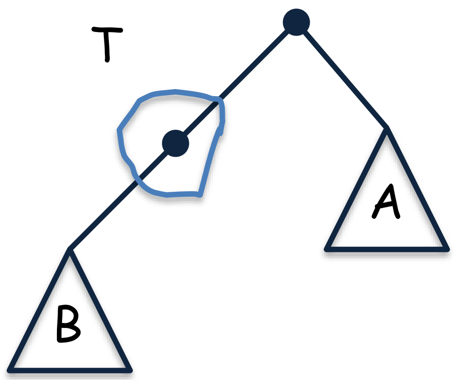
Proof : Suppose there is an optimal tree T having one node with one child.



Huffman Coding

Lemma : Optimal tree is full.
(every node has either two children or no child)

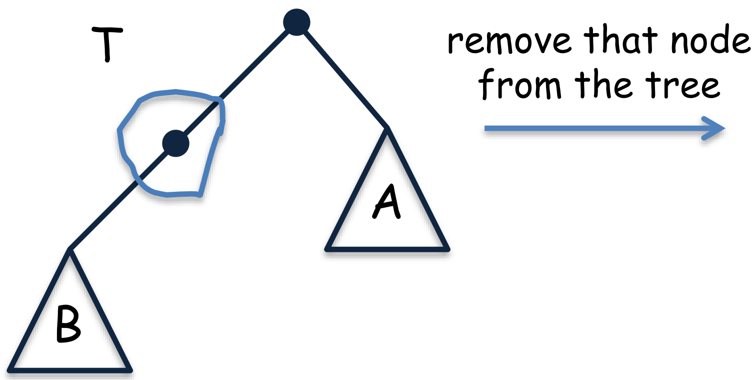
Proof : Suppose there is an optimal tree T having one node with one child.



Huffman Coding

Lemma : Optimal tree is full.
(every node has either two children or no child)

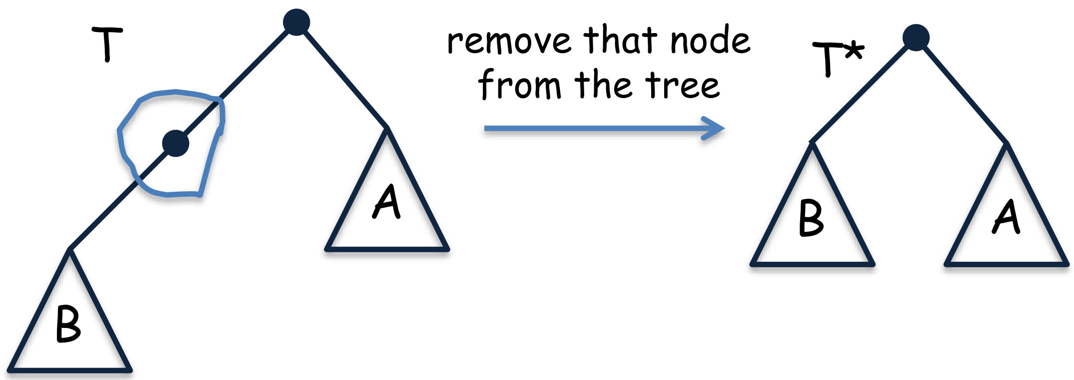
Proof : Suppose there is an optimal tree T having one node with one child.



Huffman Coding

Lemma : Optimal tree is full.
(every node has either two children or no child)

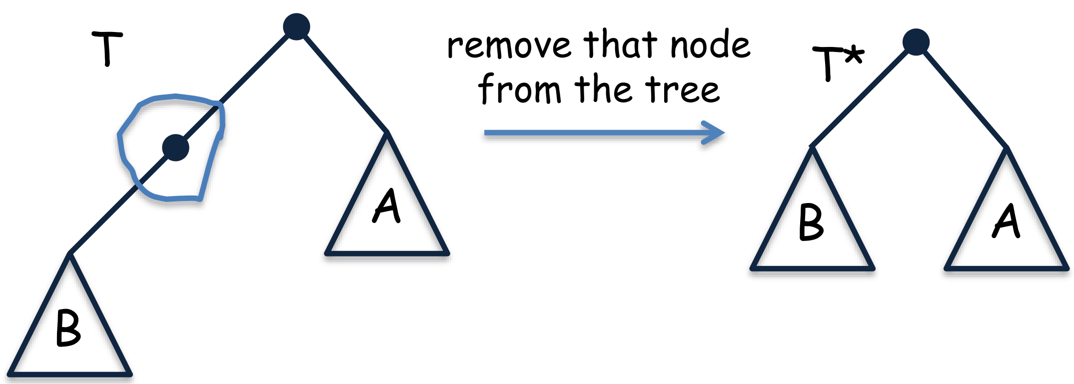
Proof : Suppose there is an optimal tree T having one node with one child.



Huffman Coding

Lemma : Optimal tree is full.
(every node has either two children or no child)

Proof : Suppose there is an optimal tree T having one node with one child.



Because all characters in subtree B of T* have encodings 1 bit shorter than encodings in subtree B of T,

$$B(T) > B(T^*)$$

Huffman Coding

- sort all frequencies in decreasing order

Huffman Coding

- sort all frequencies in decreasing order

25 18 13 10 7 5 2 1

Huffman Coding

- sort all frequencies in decreasing order
- start with lowest two frequencies

25	18	13	10	7	5	2	1

Huffman Coding

- sort all frequencies in decreasing order
- start with lowest two frequencies

  combine them in one one, rearrange to preserve the order

Huffman Coding

- sort all frequencies in decreasing order
- start with lowest two frequencies

  combine them in one one, rearrange to preserve the order

Huffman Coding

- sort all frequencies in decreasing order
- start with lowest two frequencies

  combine them in one one, rearrange to preserve the order

Huffman Coding

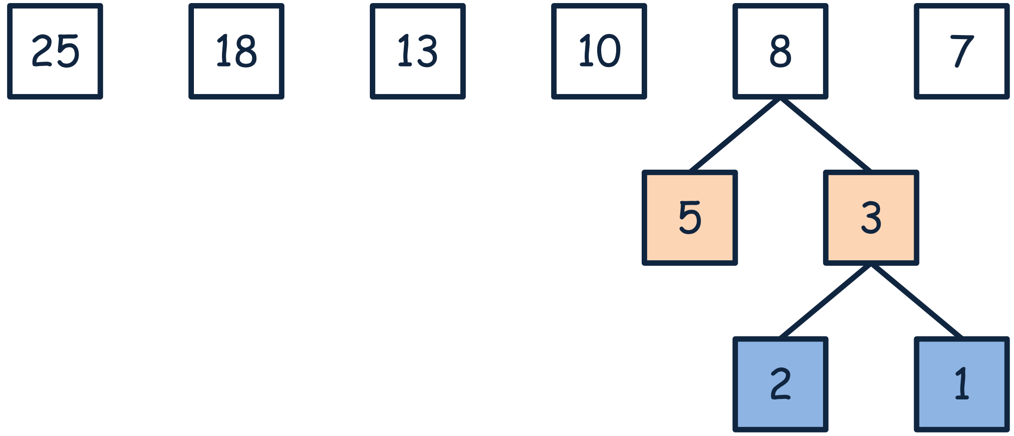
- sort all frequencies in decreasing order
- start with lowest two frequencies

    combine them in one one, rearrange to preserve the order

Huffman Coding

- sort all frequencies in decreasing order
- start with lowest two frequencies

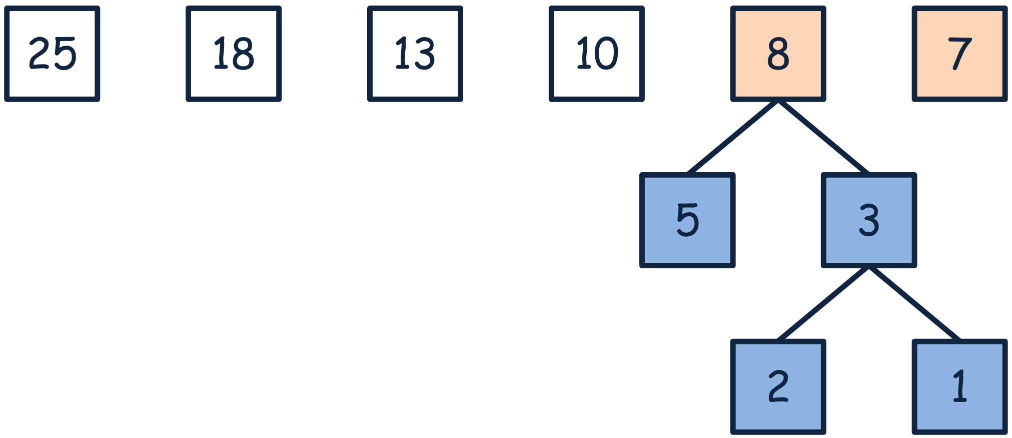
  combine them in one one, rearrange to preserve the order



Huffman Coding

- sort all frequencies in decreasing order
- start with lowest two frequencies

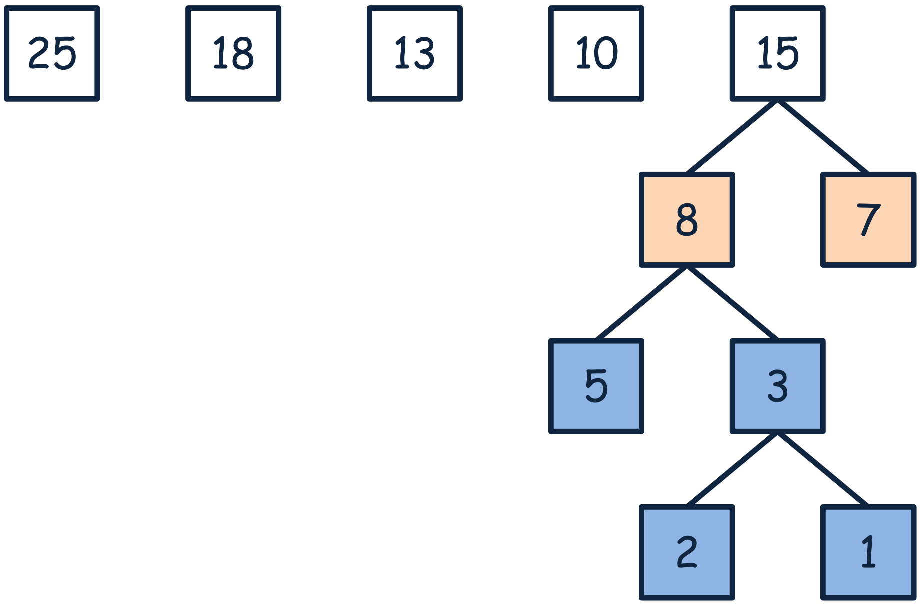
  combine them in one one, rearrange to preserve the order



Huffman Coding

- sort all frequencies in decreasing order
- start with lowest two frequencies

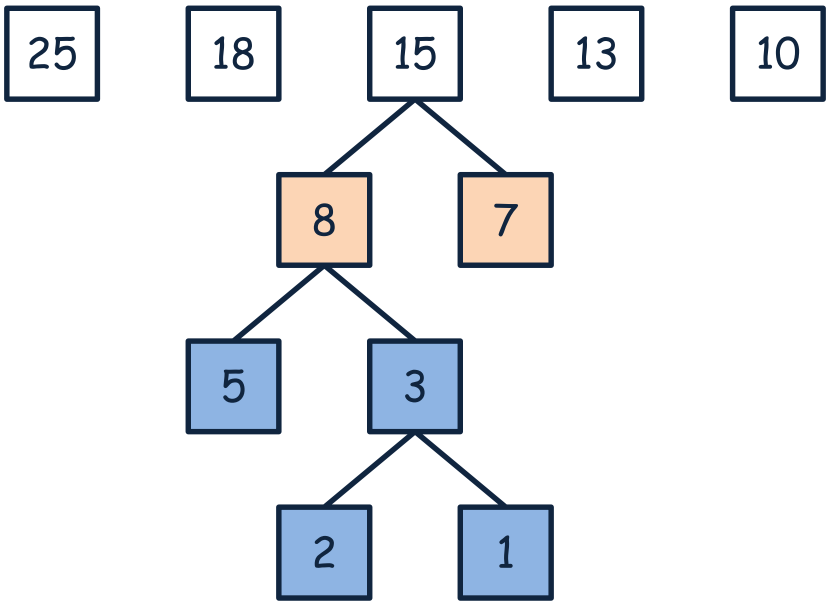
  combine them in one one, rearrange to preserve the order



Huffman Coding

- sort all frequencies in decreasing order
- start with lowest two frequencies

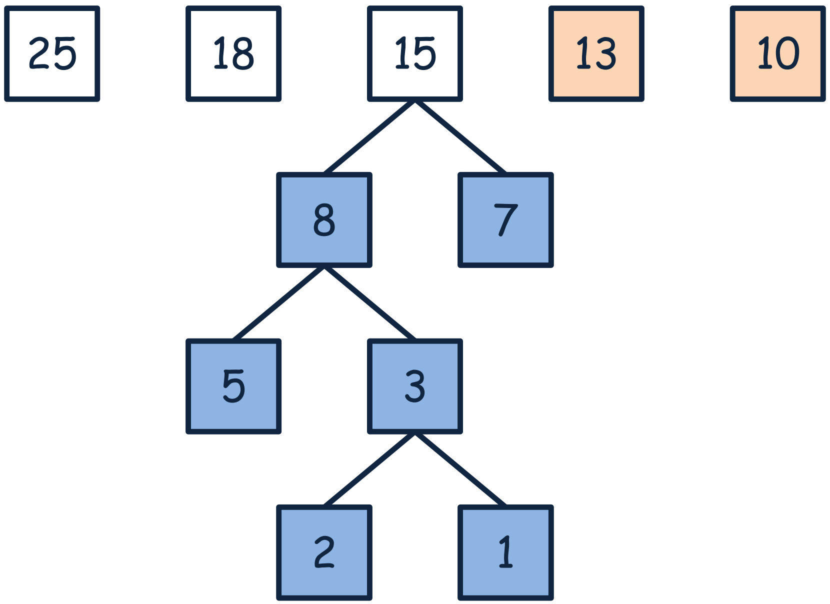
    combine them in one one, rearrange to preserve the order



Huffman Coding

- sort all frequencies in decreasing order
- start with lowest two frequencies

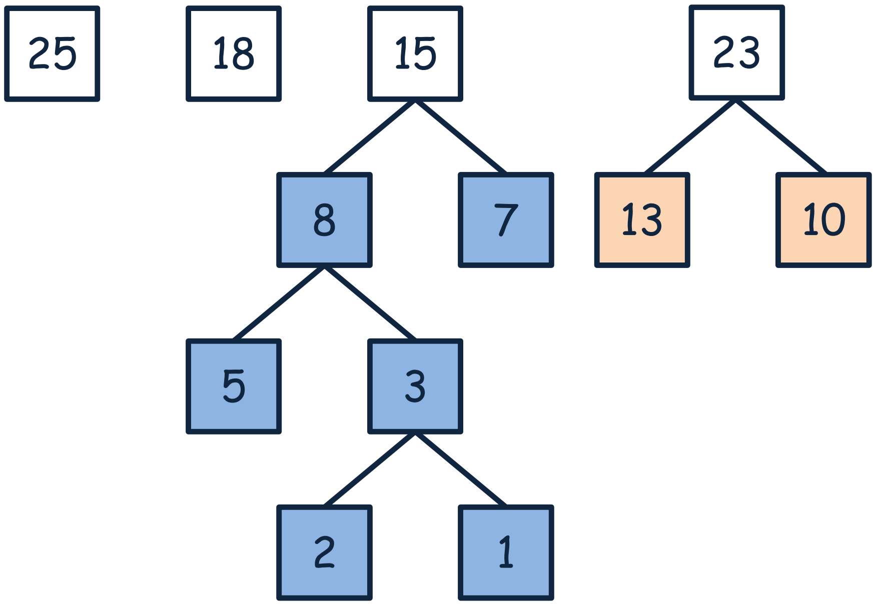
  combine them in one one, rearrange to preserve the order



Huffman Coding

- sort all frequencies in decreasing order
- start with lowest two frequencies

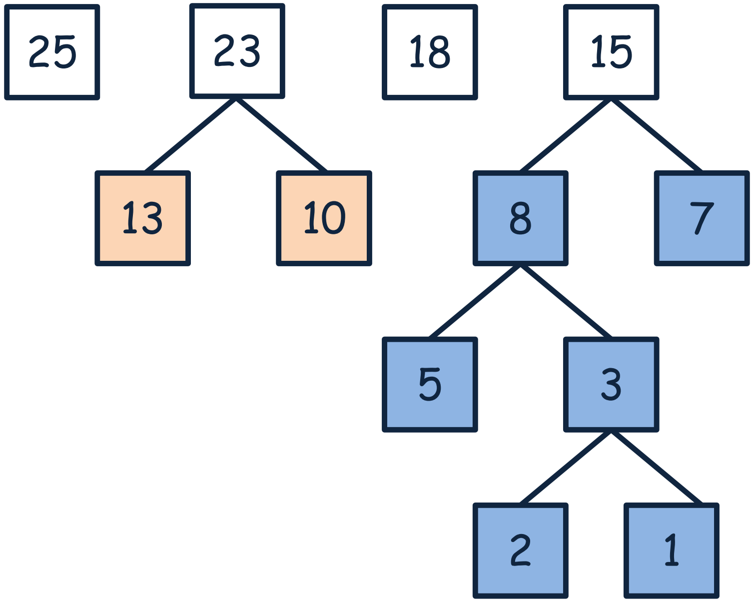
  combine them in one one, rearrange to preserve the order



Huffman Coding

- sort all frequencies in decreasing order
- start with lowest two frequencies

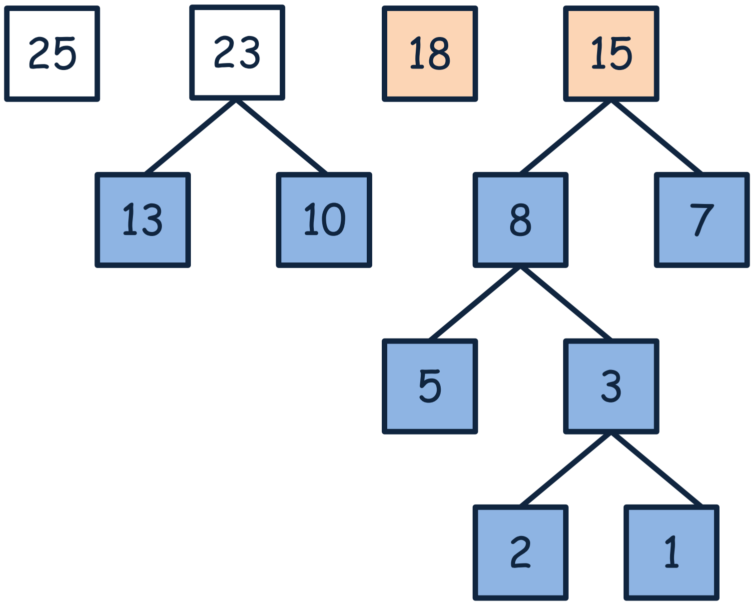
  combine them in one one, rearrange to preserve the order



Huffman Coding

- sort all frequencies in decreasing order
- start with lowest two frequencies

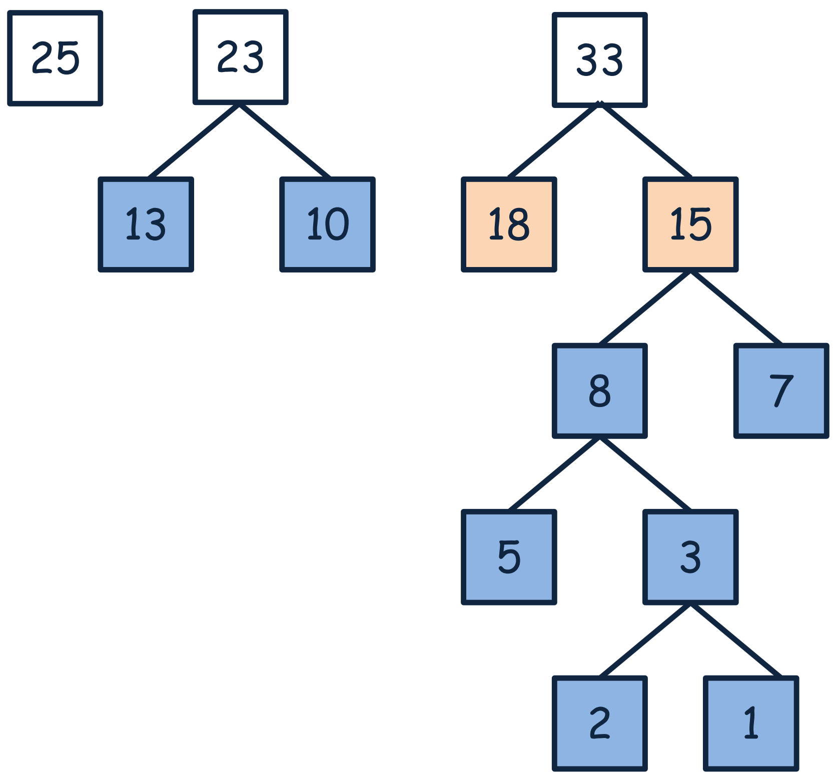
    combine them in one one, rearrange to preserve the order



Huffman Coding

- sort all frequencies in decreasing order
- start with lowest two frequencies

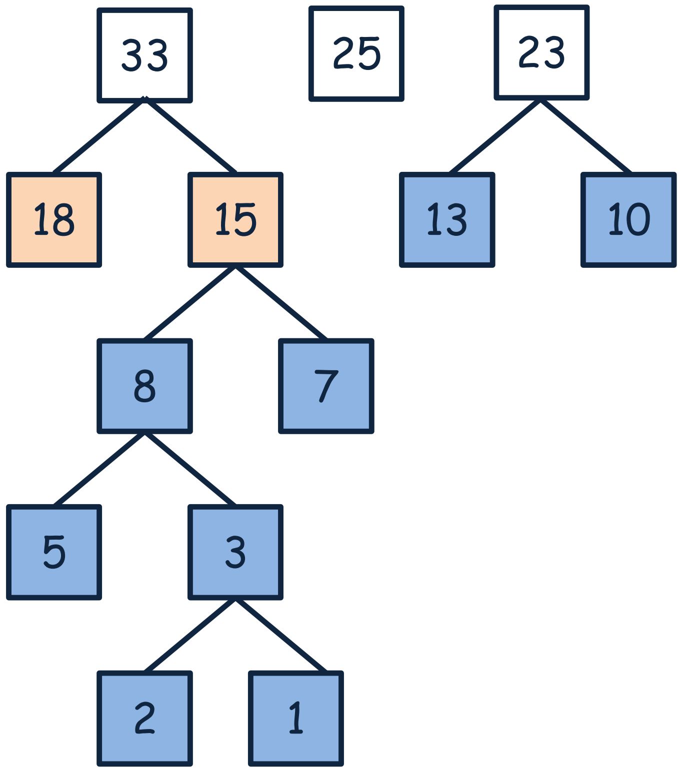
  combine them in one one, rearrange to preserve the order



Huffman Coding

- sort all frequencies in decreasing order
- start with lowest two frequencies

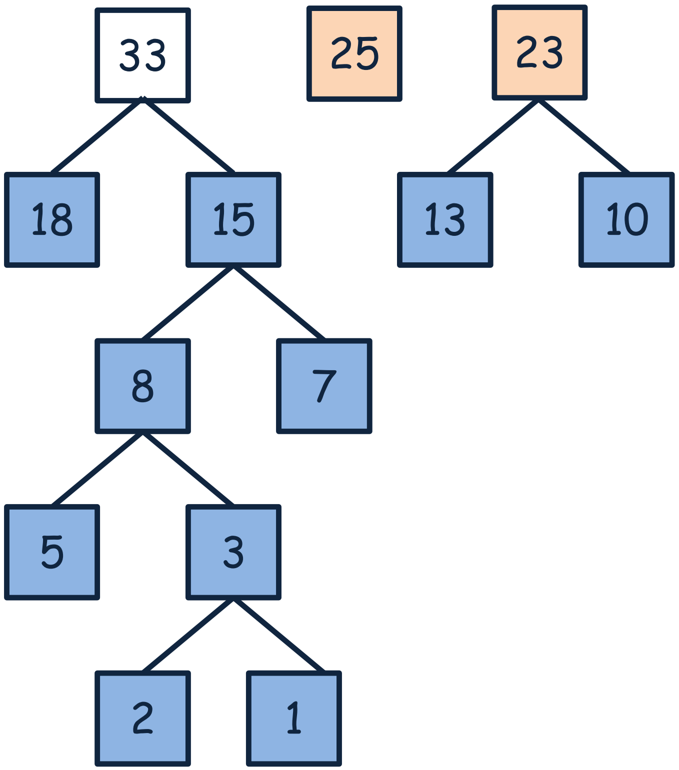
  combine them in one one, rearrange to preserve the order



Huffman Coding

- sort all frequencies in decreasing order
- start with lowest two frequencies

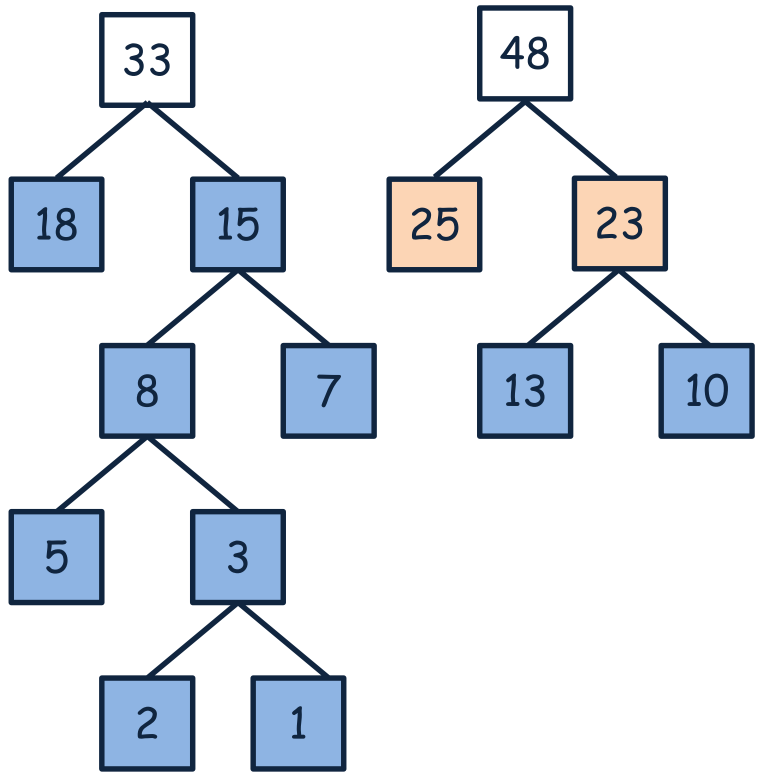
  combine them in one one, rearrange to preserve the order



Huffman Coding

- sort all frequencies in decreasing order
- start with lowest two frequencies

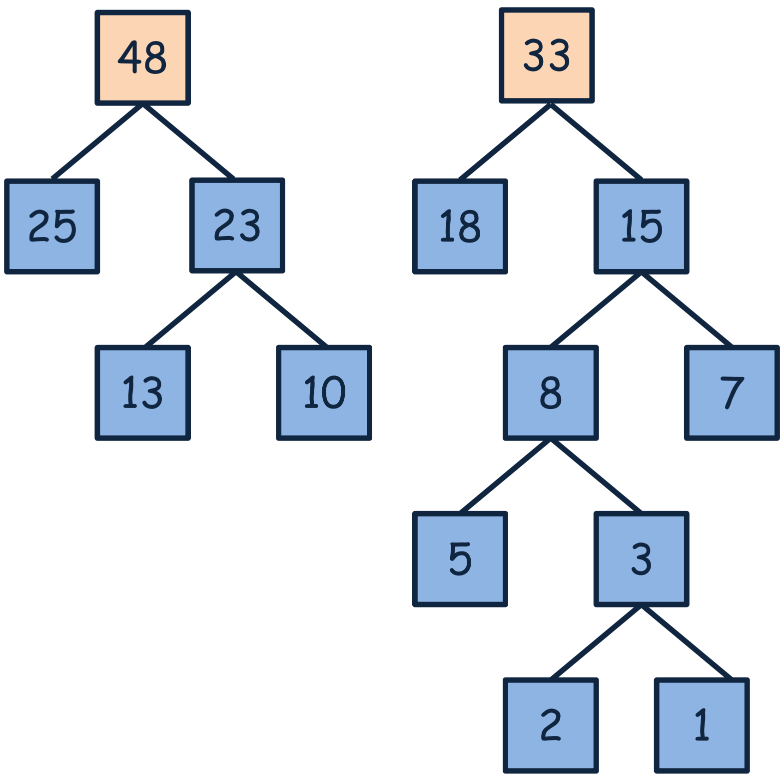
  combine them in one one, rearrange to preserve the order



Huffman Coding

- sort all frequencies in decreasing order
- start with lowest two frequencies

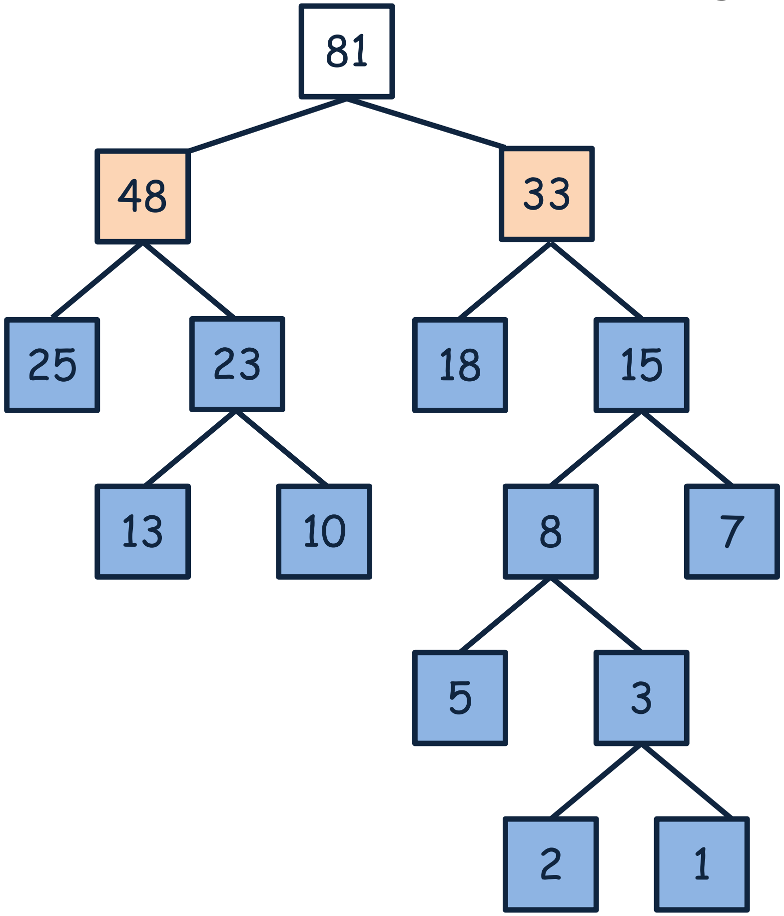
    combine them in one one, rearrange to preserve the order



Huffman Coding

- sort all frequencies in decreasing order
- start with lowest two frequencies

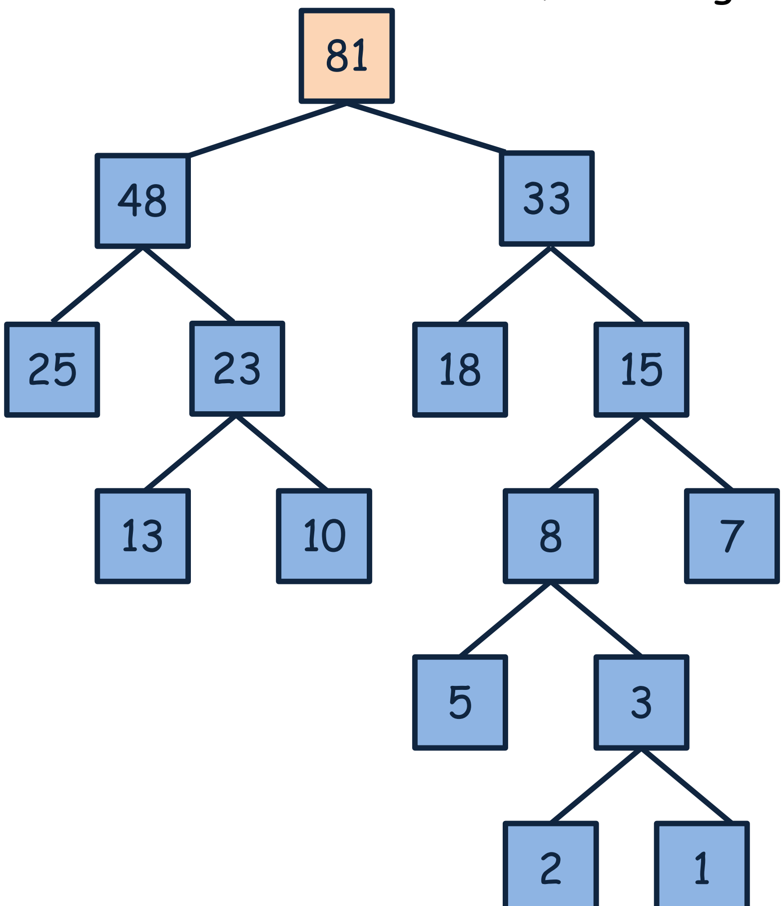
combine them in one one, rearrange to preserve the order



Huffman Coding

- sort all frequencies in decreasing order
- start with lowest two frequencies

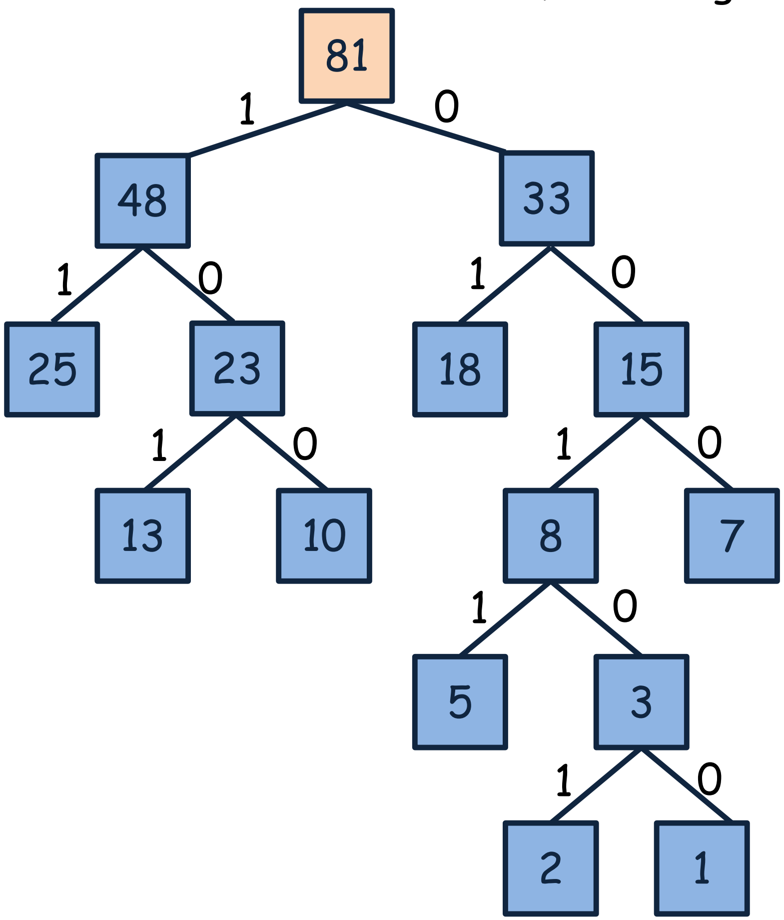
  combine them in one one, rearrange to preserve the order



Huffman Coding

- sort all frequencies in decreasing order
- start with lowest two frequencies

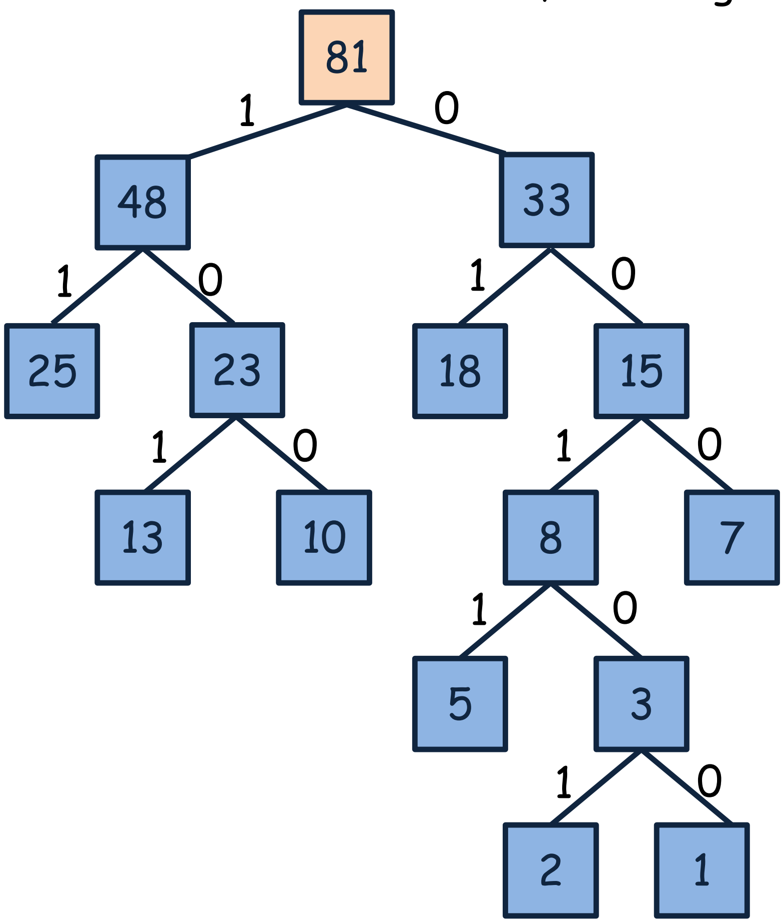
  combine them in one one, rearrange to preserve the order



Huffman Coding

- sort all frequencies in decreasing order
- start with lowest two frequencies

  combine them in one one, rearrange to preserve the order

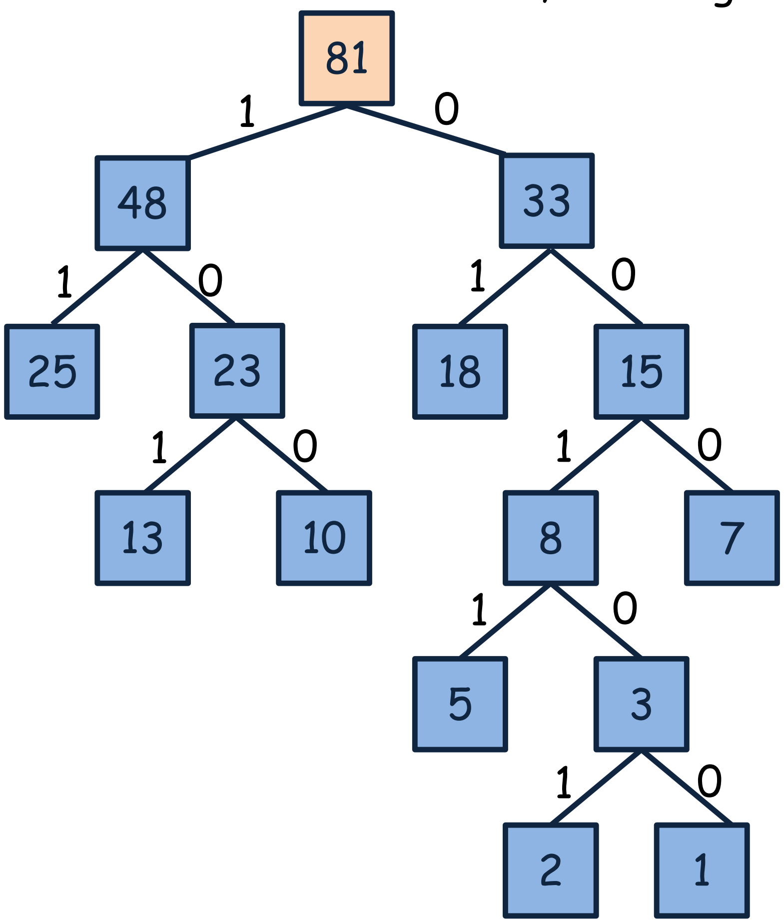


	freq
A	25
E	18
M	13
B	10
K	7
T	5
U	2
L	1

Huffman Coding

- sort all frequencies in decreasing order
- start with lowest two frequencies

  combine them in one one, rearrange to preserve the order

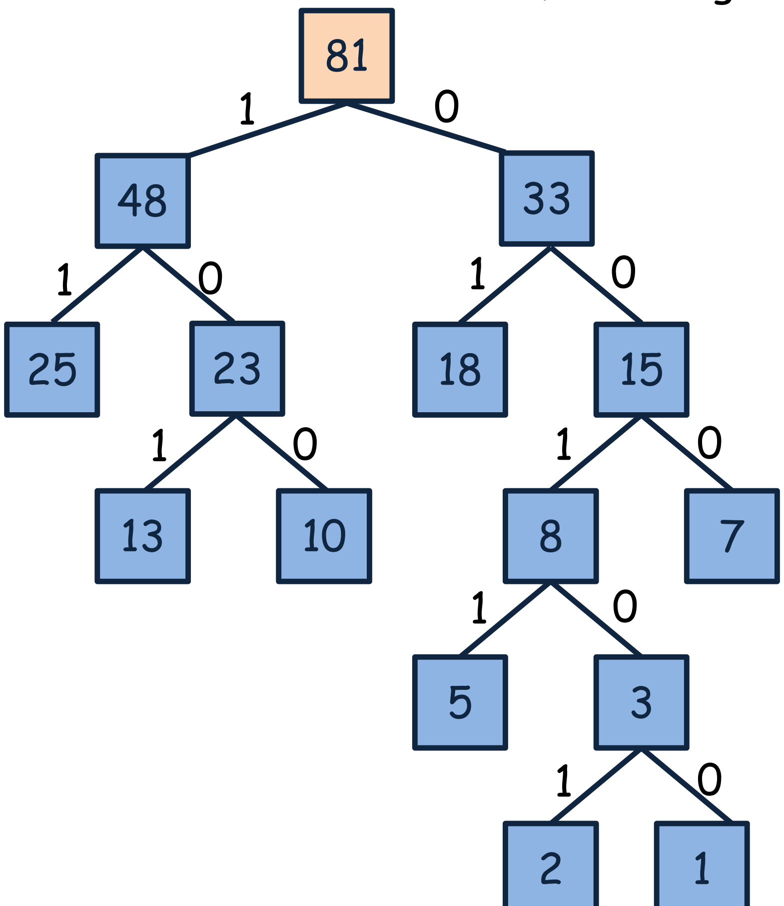


	freq	
A	25	11
E	18	01
M	13	101
B	10	100
K	7	000
T	5	0011
U	2	00101
L	1	00100

Huffman Coding

- sort all frequencies in decreasing order
- start with lowest two frequencies

combine them in one one, rearrange to preserve the order

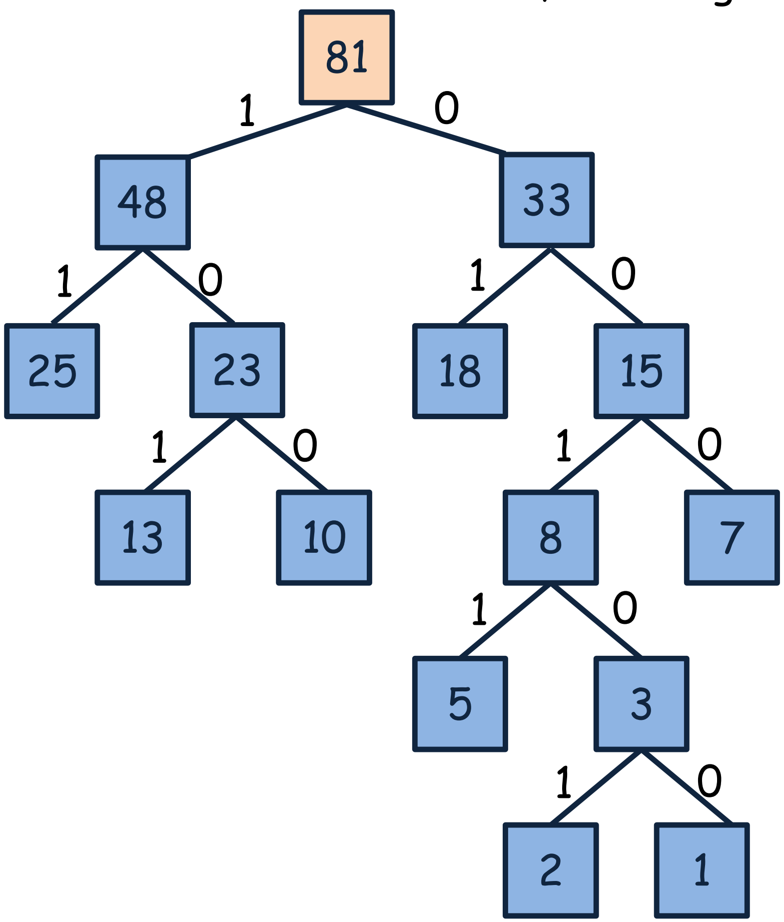


	freq		cost
A	25	11	50
E	18	01	36
M	13	101	39
B	10	100	30
K	7	000	21
T	5	0011	20
U	2	00101	10
L	1	00100	5

Huffman Coding

- sort all frequencies in decreasing order
- start with lowest two frequencies

combine them in one one, rearrange to preserve the order

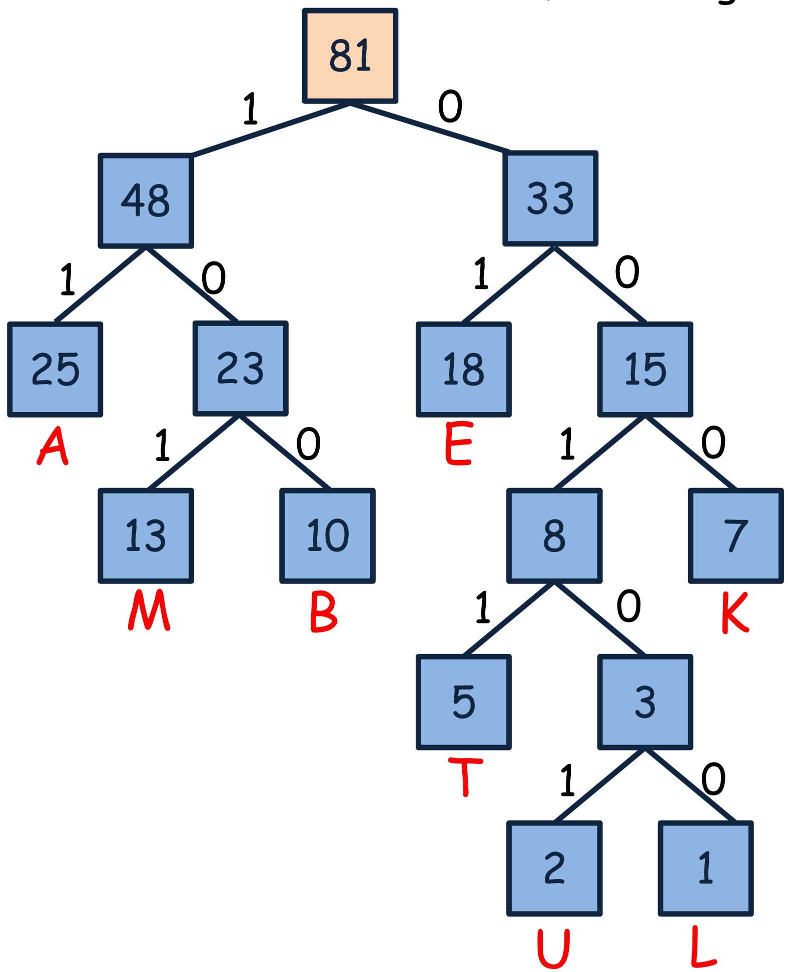


	freq		cost
A	25	11	50
E	18	01	36
M	13	101	39
B	10	100	30
K	7	000	21
T	5	0011	20
U	2	00101	10
L	1	00100	5
			211

Huffman Coding

- sort all frequencies in decreasing order
- start with lowest two frequencies

combine them in one one, rearrange to preserve the order

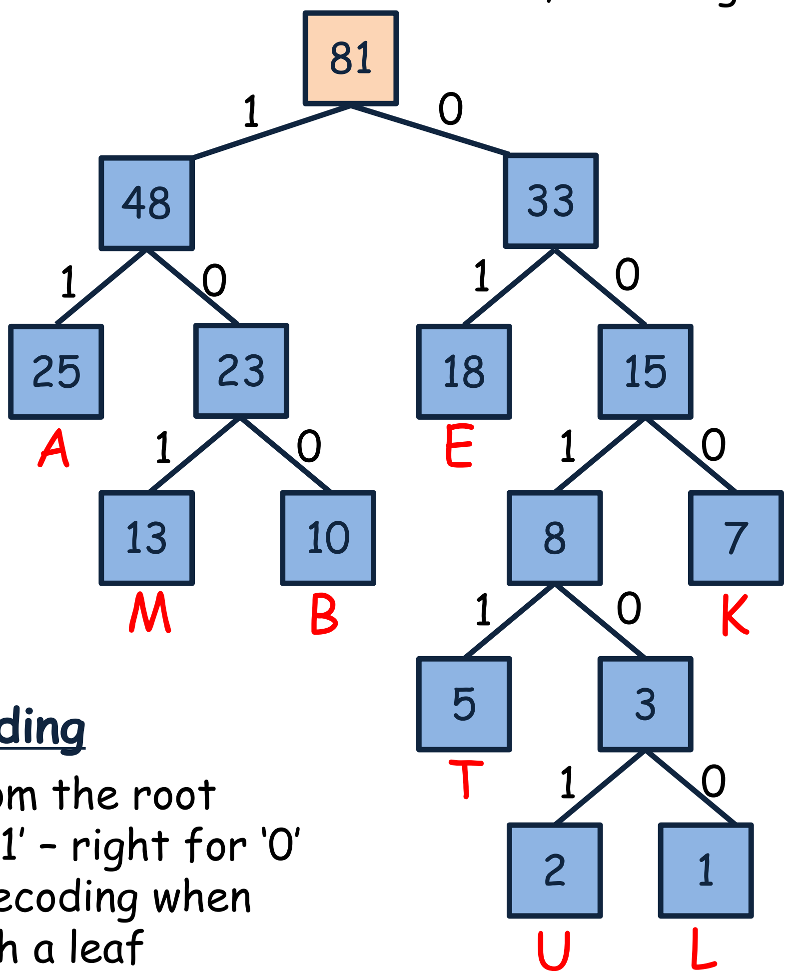


	freq		cost
A	25	11	50
E	18	01	36
M	13	101	39
B	10	100	30
K	7	000	21
T	5	0011	20
U	2	00101	10
L	1	00100	5
			211

Huffman Coding

- sort all frequencies in decreasing order
- start with lowest two frequencies

combine them in one one, rearrange to preserve the order



	freq		cost
A	25	11	50
E	18	01	36
M	13	101	39
B	10	100	30
K	7	000	21
T	5	0011	20
U	2	00101	10
L	1	00100	5
			211

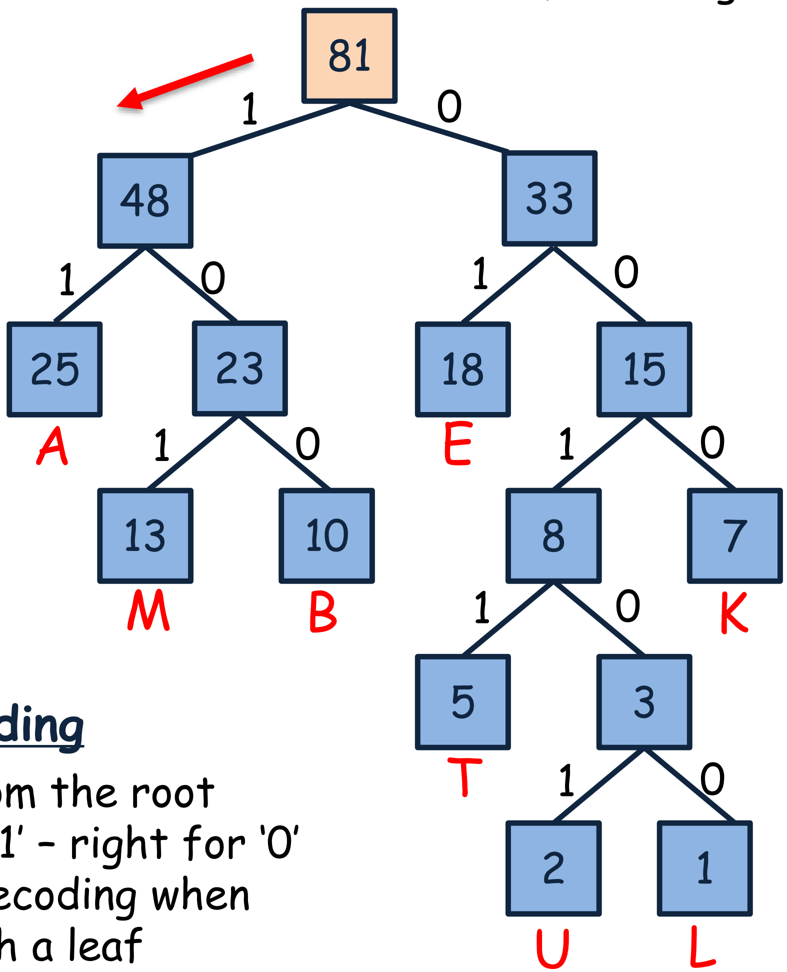
decoding

- start from the root
- left for '1' – right for '0'
- end of decoding when you reach a leaf

Huffman Coding

- sort all frequencies in decreasing order
- start with lowest two frequencies

combine them in one one, rearrange to preserve the order



	freq		cost
A	25	11	50
E	18	01	36
M	13	101	39
B	10	100	30
K	7	000	21
T	5	0011	20
U	2	00101	10
L	1	00100	5
			211

decoding

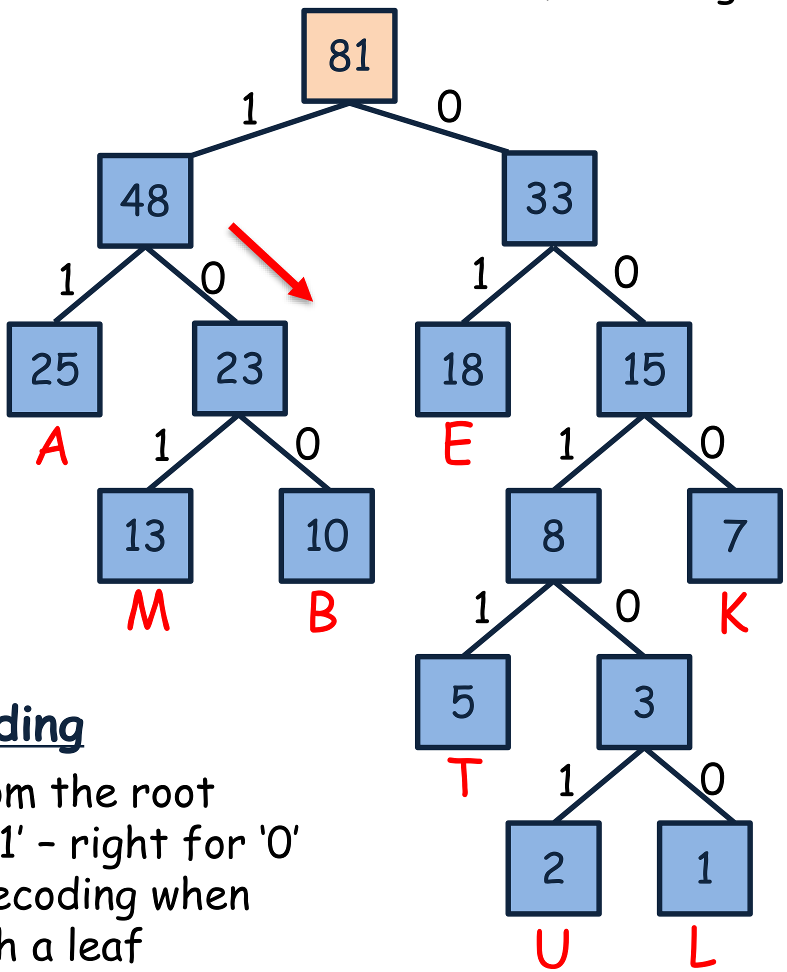
- start from the root
- left for '1' – right for '0'
- end of decoding when you reach a leaf

1 0 0 0 1 0 0 0

Huffman Coding

- sort all frequencies in decreasing order
- start with lowest two frequencies

combine them in one one, rearrange to preserve the order



	freq		cost
A	25	11	50
E	18	01	36
M	13	101	39
B	10	100	30
K	7	000	21
T	5	0011	20
U	2	00101	10
L	1	00100	5
			211

1 0 0 0 1 0 0 0

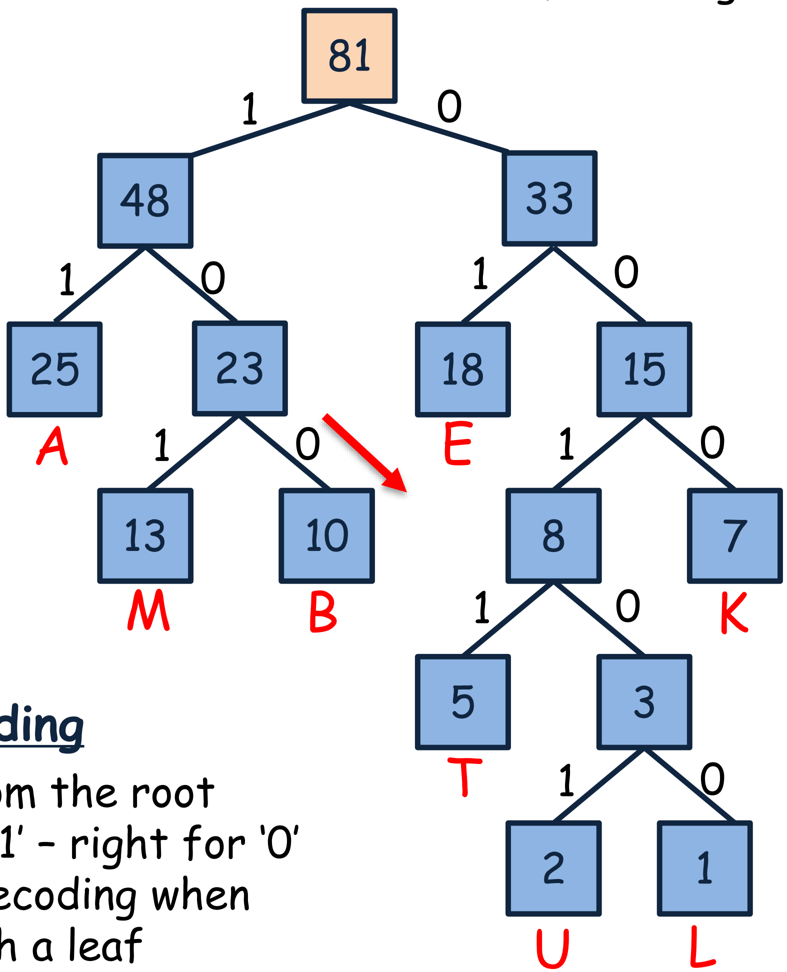
decoding

- start from the root
- left for '1' – right for '0'
- end of decoding when you reach a leaf

Huffman Coding

- sort all frequencies in decreasing order
- start with lowest two frequencies

combine them in one one, rearrange to preserve the order



	freq		cost
A	25	11	50
E	18	01	36
M	13	101	39
B	10	100	30
K	7	000	21
T	5	0011	20
U	2	00101	10
L	1	00100	5
			211

decoding

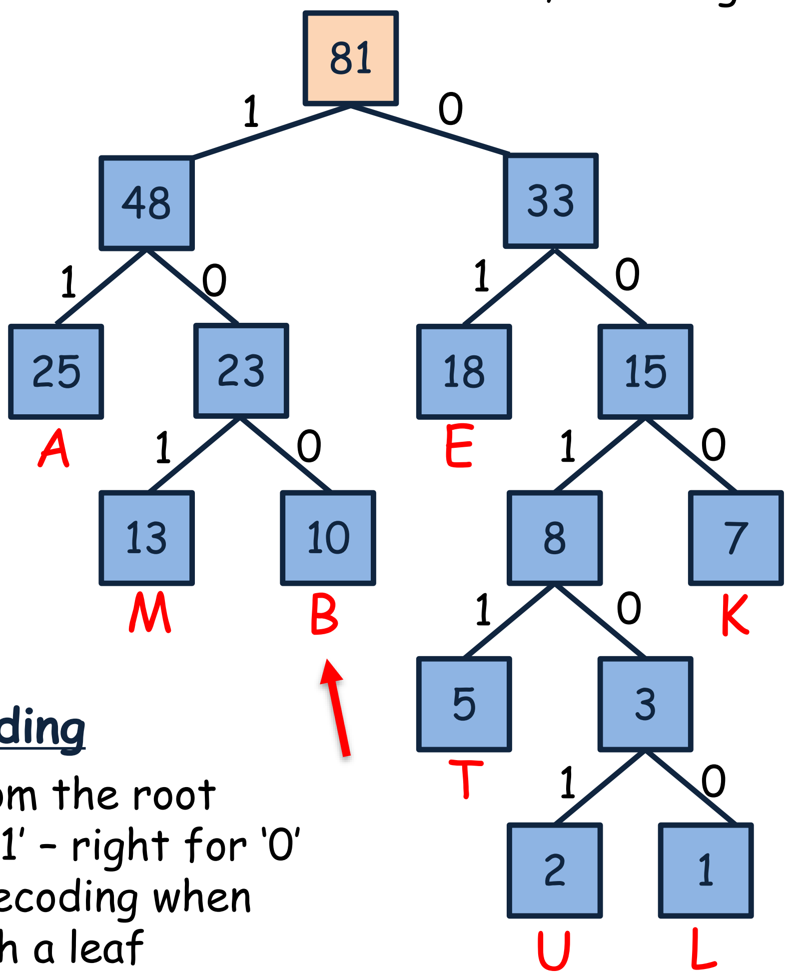
- start from the root
- left for '1' – right for '0'
- end of decoding when you reach a leaf

1 0 0 0 1 0 0 0

Huffman Coding

- sort all frequencies in decreasing order
- start with lowest two frequencies

  combine them in one one, rearrange to preserve the order



	freq		cost
A	25	11	50
E	18	01	36
M	13	101	39
B	10	100	30
K	7	000	21
T	5	0011	20
U	2	00101	10
L	1	00100	5
			211

decoding

- start from the root
- left for '1' – right for '0'
- end of decoding when you reach a leaf

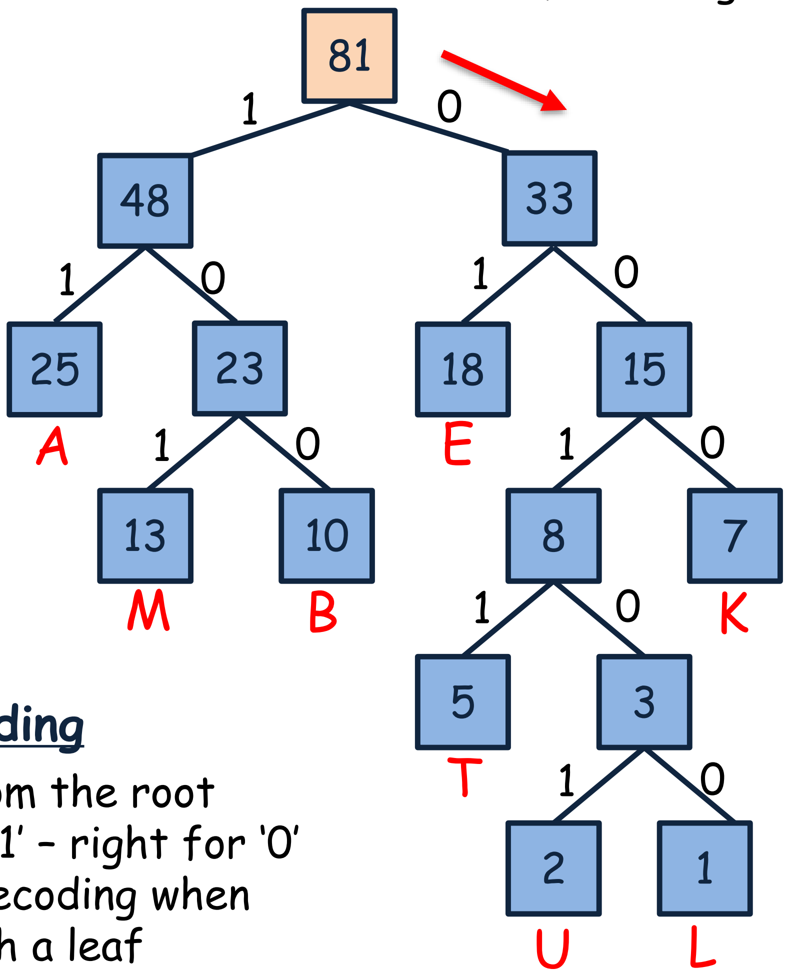
1 0 0 0 1 0 0 0

B

Huffman Coding

- sort all frequencies in decreasing order
- start with lowest two frequencies

combine them in one one, rearrange to preserve the order



	freq		cost
A	25	11	50
E	18	01	36
M	13	101	39
B	10	100	30
K	7	000	21
T	5	0011	20
U	2	00101	10
L	1	00100	5
			211

decoding

- start from the root
- left for '1' – right for '0'
- end of decoding when you reach a leaf

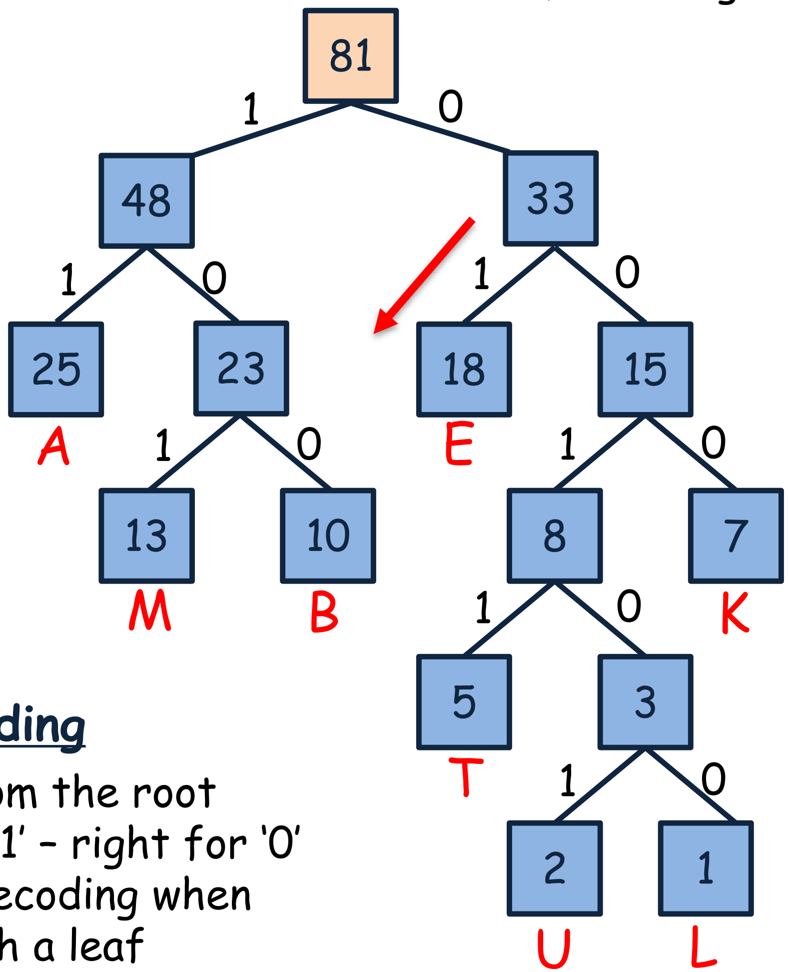
1 0 0 0 1 0 0 0

B

Huffman Coding

- sort all frequencies in decreasing order
- start with lowest two frequencies

combine them in one one, rearrange to preserve the order



	freq		cost
A	25	11	50
E	18	01	36
M	13	101	39
B	10	100	30
K	7	000	21
T	5	0011	20
U	2	00101	10
L	1	00100	5
			211

decoding

- start from the root
- left for '1' – right for '0'
- end of decoding when you reach a leaf

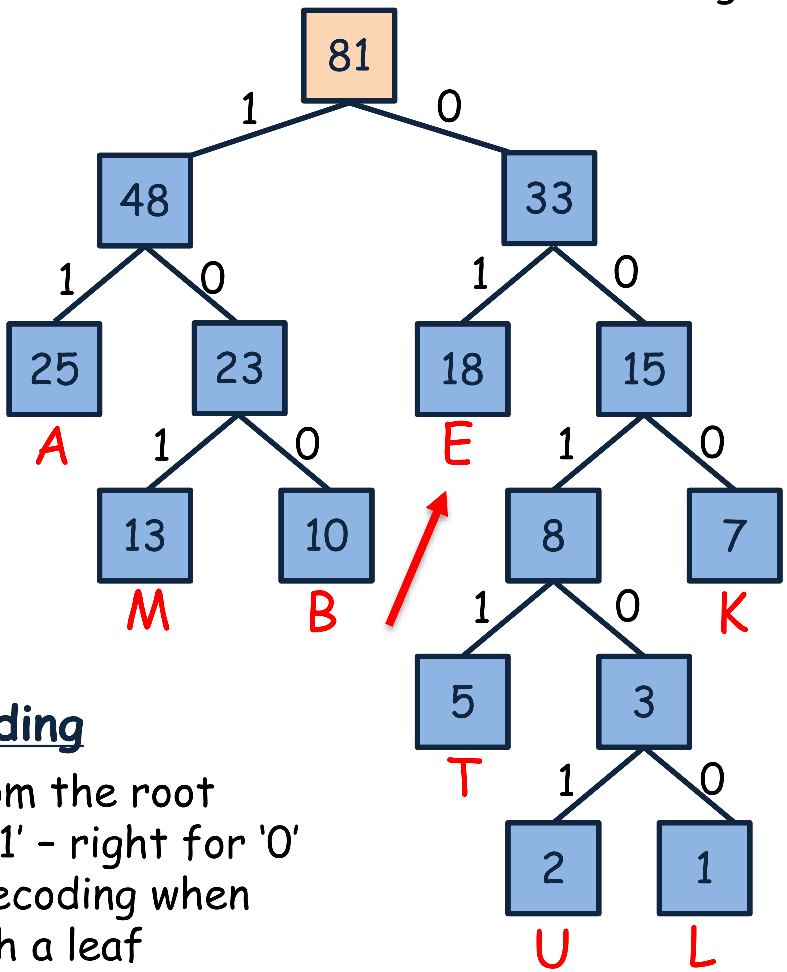
1 0 0 0 1 0 0 0

B

Huffman Coding

- sort all frequencies in decreasing order
- start with lowest two frequencies

combine them in one one, rearrange to preserve the order



	freq		cost
A	25	11	50
E	18	01	36
M	13	101	39
B	10	100	30
K	7	000	21
T	5	0011	20
U	2	00101	10
L	1	00100	5
			211

decoding

- start from the root
- left for '1' – right for '0'
- end of decoding when you reach a leaf

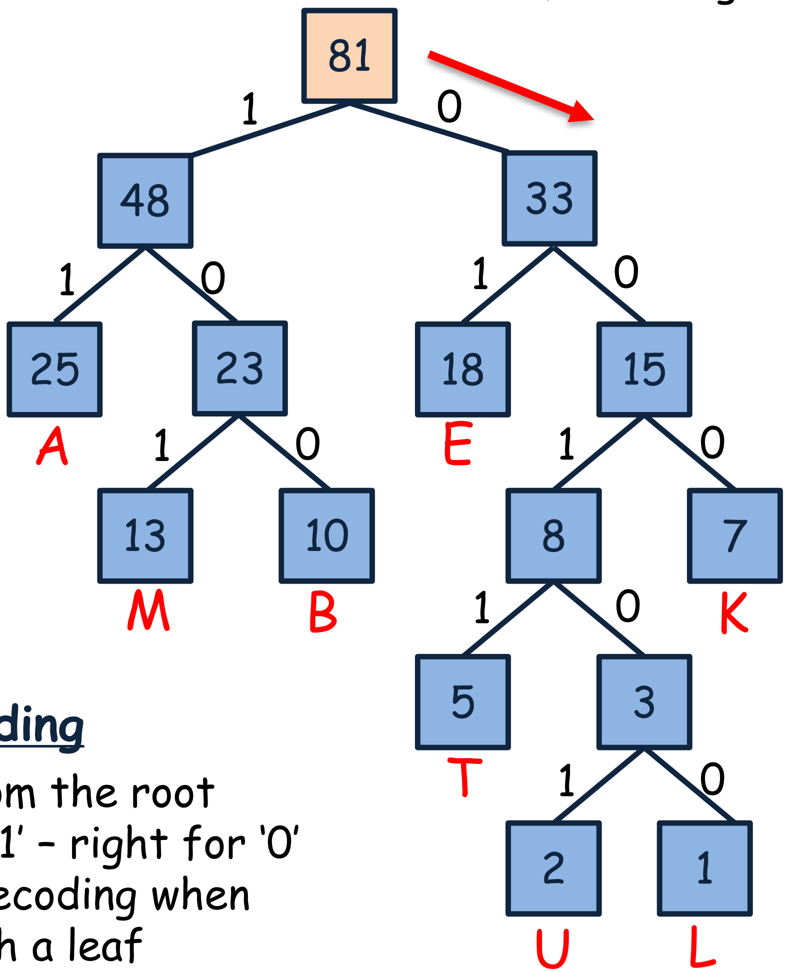
1 0 0 0 1 0 0 0

B E

Huffman Coding

- sort all frequencies in decreasing order
- start with lowest two frequencies

combine them in one one, rearrange to preserve the order



	freq		cost
A	25	11	50
E	18	01	36
M	13	101	39
B	10	100	30
K	7	000	21
T	5	0011	20
U	2	00101	10
L	1	00100	5
			211

decoding

- start from the root
- left for '1' – right for '0'
- end of decoding when you reach a leaf

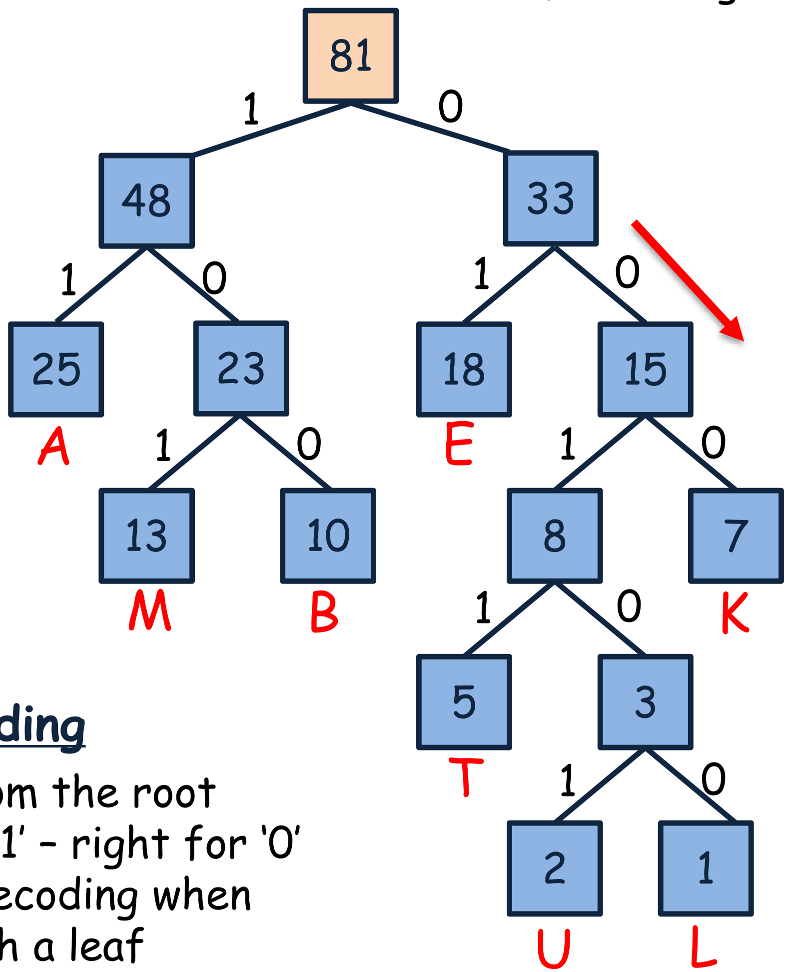
1 0 0 0 1 0 0 0

B E

Huffman Coding

- sort all frequencies in decreasing order
- start with lowest two frequencies

combine them in one one, rearrange to preserve the order



	freq		cost
A	25	11	50
E	18	01	36
M	13	101	39
B	10	100	30
K	7	000	21
T	5	0011	20
U	2	00101	10
L	1	00100	5
			211

decoding

- start from the root
- left for '1' – right for '0'
- end of decoding when you reach a leaf

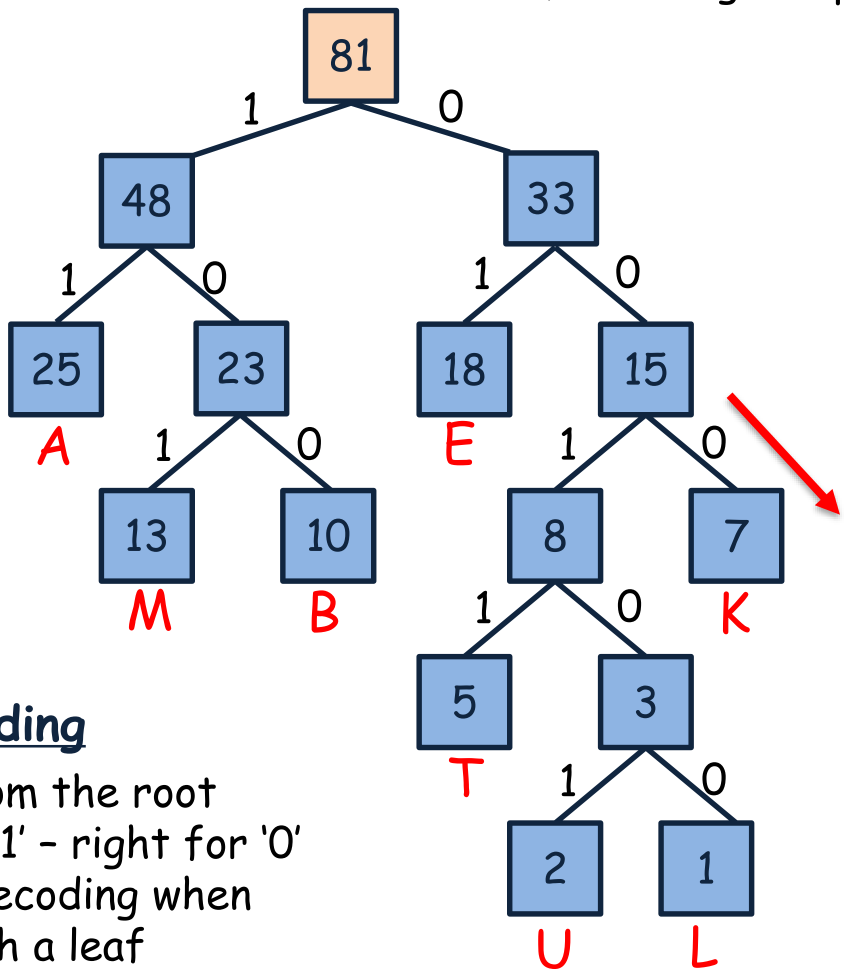
1 0 0 0 1 0 0 0

B E

Huffman Coding

- sort all frequencies in decreasing order
- start with lowest two frequencies

combine them in one one, rearrange to preserve the order



	freq		cost
A	25	11	50
E	18	01	36
M	13	101	39
B	10	100	30
K	7	000	21
T	5	0011	20
U	2	00101	10
L	1	00100	5
			211

decoding

- start from the root
- left for '1' – right for '0'
- end of decoding when you reach a leaf

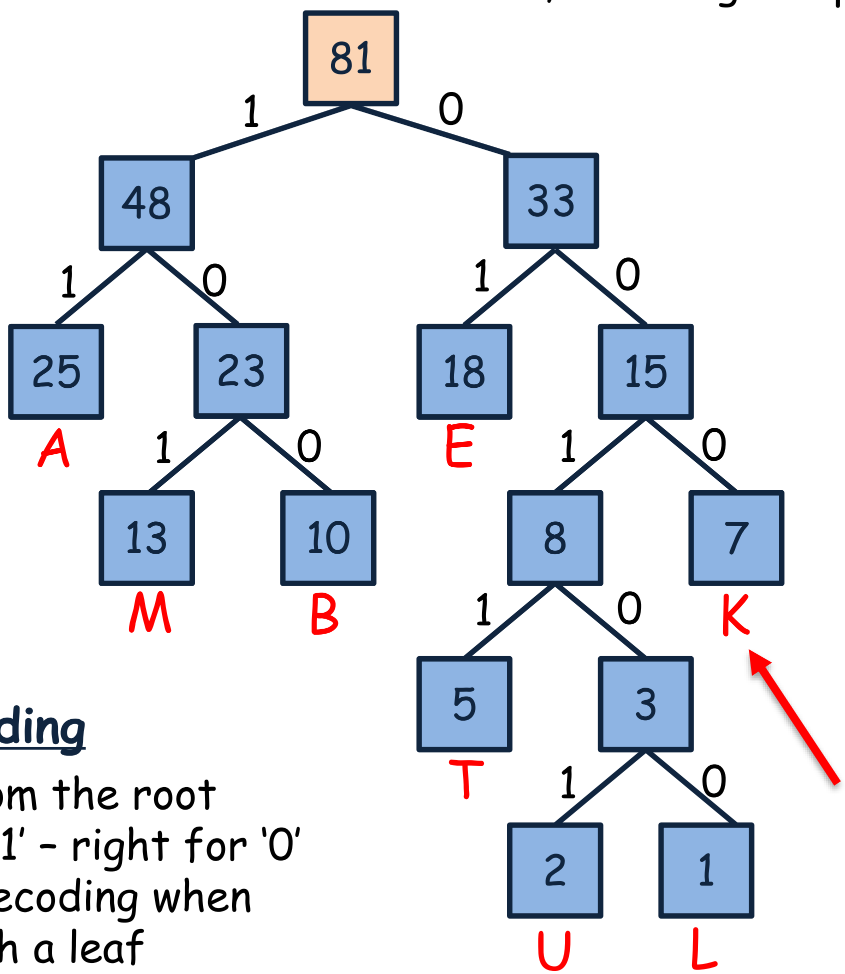
1 0 0 0 1 0 0 0

B E

Huffman Coding

- sort all frequencies in decreasing order
- start with lowest two frequencies

combine them in one one, rearrange to preserve the order



	freq		cost
A	25	11	50
E	18	01	36
M	13	101	39
B	10	100	30
K	7	000	21
T	5	0011	20
U	2	00101	10
L	1	00100	5
			211

decoding

- start from the root
- left for '1' – right for '0'
- end of decoding when you reach a leaf

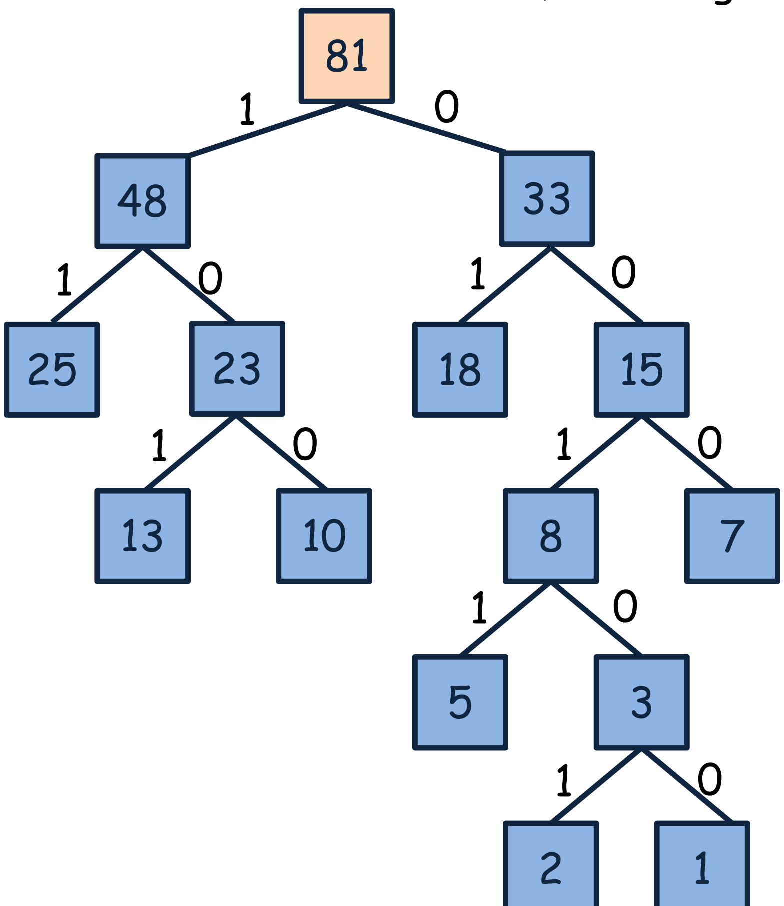
1 0 0 0 1 0 0 0

B E K

Huffman Coding

- sort all frequencies in decreasing order
- start with lowest two frequencies

  combine them in one one, rearrange to preserve the order

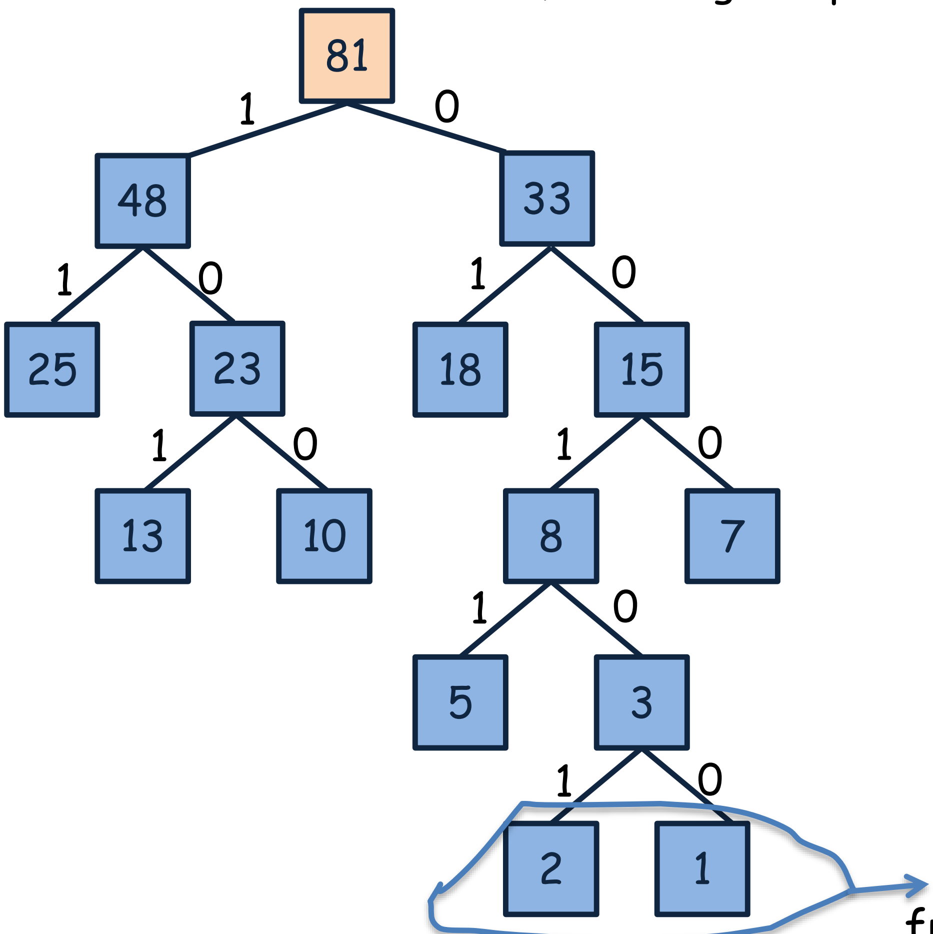


	freq		cost
A	25	11	50
E	18	01	36
M	13	101	39
B	10	100	30
K	7	000	21
T	5	0011	20
U	2	00101	10
L	1	00100	5
			211

Huffman Coding

- sort all frequencies in decreasing order
- start with lowest two frequencies

combine them in one one, rearrange to preserve the order



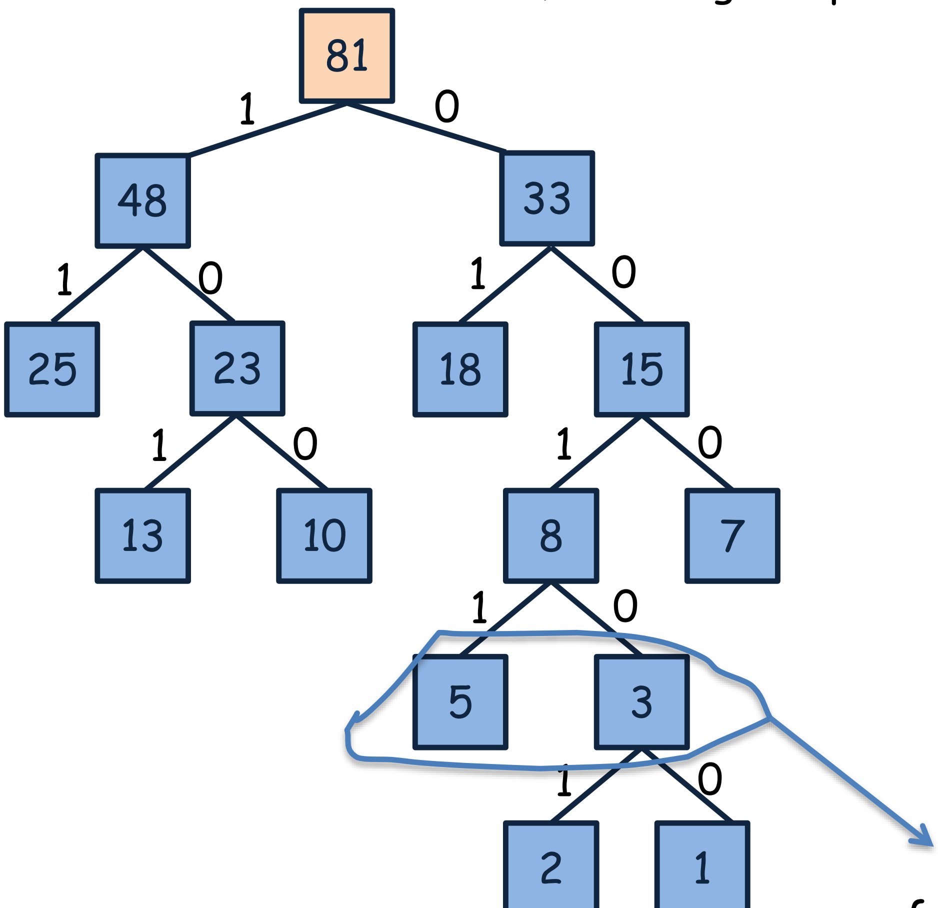
	freq		cost
A	25	11	50
E	18	01	36
M	13	101	39
B	10	100	30
K	7	000	21
T	5	0011	20
U	2	00101	10
L	1	00100	5
			211

two symbols with lowest frequencies will be siblings placed at lowest level in the tree

Huffman Coding

- sort all frequencies in decreasing order
- start with lowest two frequencies

combine them in one one, rearrange to preserve the order



	freq		cost
A	25	11	50
E	18	01	36
M	13	101	39
B	10	100	30
K	7	000	21
T	5	0011	20
U	2	00101	10
L	1	00100	5
			211

two symbols with lowest frequencies will be siblings placed at lowest level in the tree

Huffman Coding

Theorem (Greedy-choice property): Let x and y be twe symbols with the smallest frequencies f_x and f_y. There exists an optimal tree where x and y are siblings with the highest depth. (Our greedy approach yields us an optimal solution)

Proof

Huffman Coding

Theorem (Greedy-choice property): Let x and y be twe symbols with the smallest frequencies f_x and f_y. There exists an optimal tree where x and y are siblings with the highest depth. (Our greedy approach yields us an optimal solution)

Proof

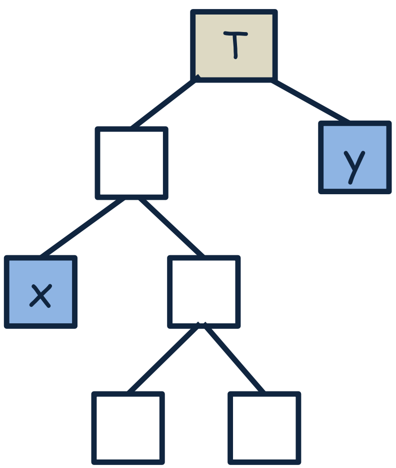
Assume there is an optimal tree T where x and y are not siblings.

Huffman Coding

Theorem (Greedy-choice property): Let x and y be two symbols with the smallest frequencies f_x and f_y. There exists an optimal tree where x and y are siblings with the highest depth. (Our greedy approach yields us an optimal solution)

Proof

Assume there is an optimal tree T where x and y are not siblings.

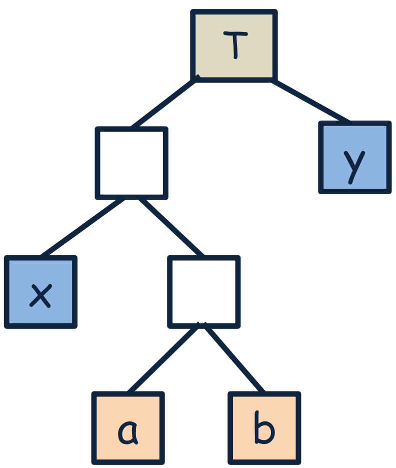


Huffman Coding

Theorem (Greedy-choice property): Let x and y be twe symbols with the smallest frequencies f_x and f_y. There exists an optimal tree where x and y are siblings with the highest depth. (Our greedy approach yields us an optimal solution)

Proof

Assume there is an optimal tree T where x and y are not siblings.



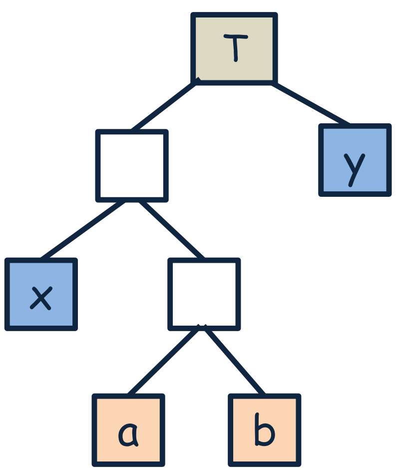
- Because T is a full tree, there should be two symbols a and b that are siblings placed at the lowest level in T

Huffman Coding

Theorem (Greedy-choice property): Let x and y be twe symbols with the smallest frequencies f_x and f_y. There exists an optimal tree where x and y are siblings with the highest depth. (Our greedy approach yields us an optimal solution)

Proof

Assume there is an optimal tree T where x and y are not siblings.



- Because T is a full tree, there should be two symbols a and b that are siblings placed at the lowest level in T
- Since f_x and f_y are the smallest frequencies,

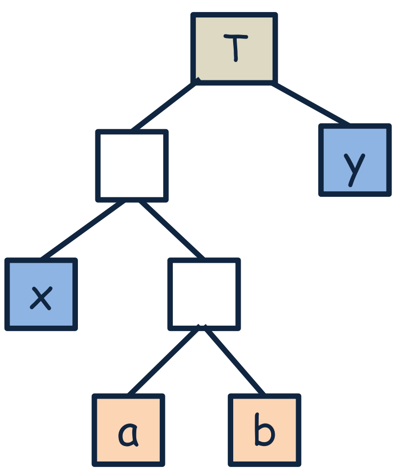
$$f_x\ ,\ f_y \le f_a\ ,\ f_b$$

Huffman Coding

Theorem (Greedy-choice property): Let x and y be twe symbols with the smallest frequencies f_x and f_y. There exists an optimal tree where x and y are siblings with the highest depth. (Our greedy approach yields us an optimal solution)

Proof

Assume there is an optimal tree T where x and y are not siblings.



- Because T is a full tree, there should be two symbols a and b that are siblings placed at the lowest level in T
- Since f_x and f_y are the smallest frequencies,

$$f_x , f_y \leq f_a , f_b$$

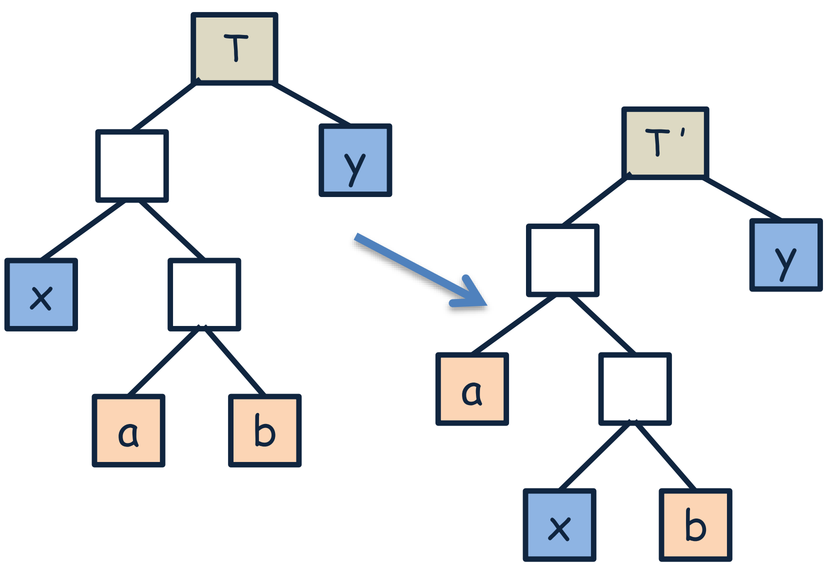
swap x and a

Huffman Coding

Theorem (Greedy-choice property): Let x and y be twe symbols with the smallest frequencies f_x and f_y. There exists an optimal tree where x and y are siblings with the highest depth. (Our greedy approach yields us an optimal solution)

Proof

Assume there is an optimal tree T where x and y are not siblings.



- Because T is a full tree, there should be two symbols a and b that are siblings placed at the lowest level in T
- Since f_x and f_y are the smallest frequencies,

$$f_x , f_y \leq f_a , f_b$$

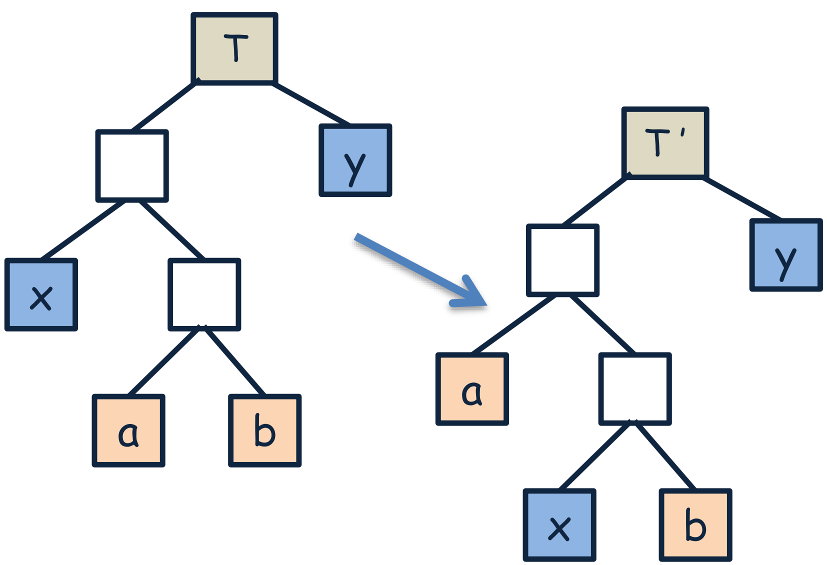
swap x and a

Huffman Coding

Theorem (Greedy-choice property): Let x and y be twe symbols with the smallest frequencies f_x and f_y. There exists an optimal tree where x and y are siblings with the highest depth. (Our greedy approach yields us an optimal solution)

Proof

Assume there is an optimal tree T where x and y are not siblings.



- Because T is a full tree, there should be two symbols a and b that are siblings placed at the lowest level in T
- Since f_x and f_y are the smallest frequencies,

$$f_x , f_y \leq f_a , f_b$$

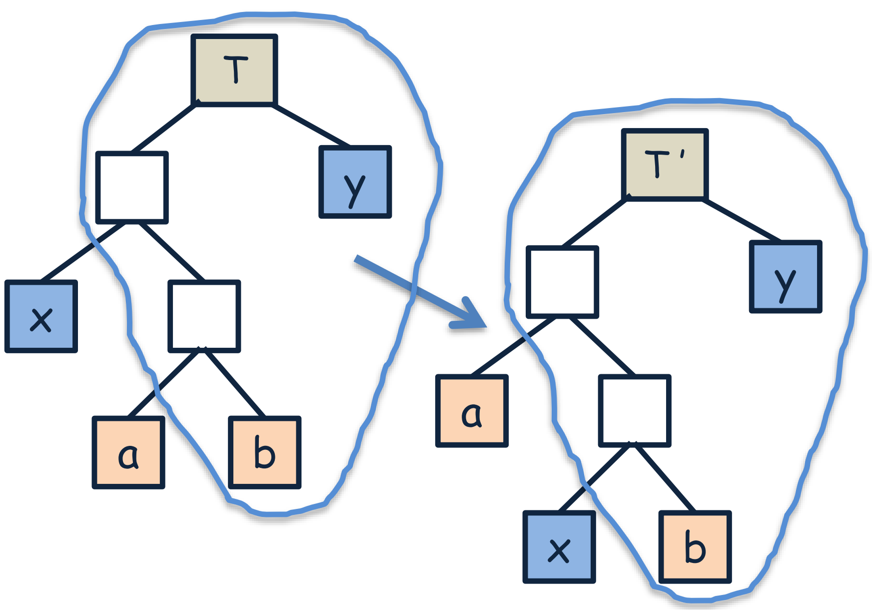
- B(T) =
-

Huffman Coding

Theorem (Greedy-choice property): Let x and y be twe symbols with the smallest frequencies f_x and f_y. There exists an optimal tree where x and y are siblings with the highest depth. (Our greedy approach yields us an optimal solution)

Proof

Assume there is an optimal tree T where x and y are not siblings.



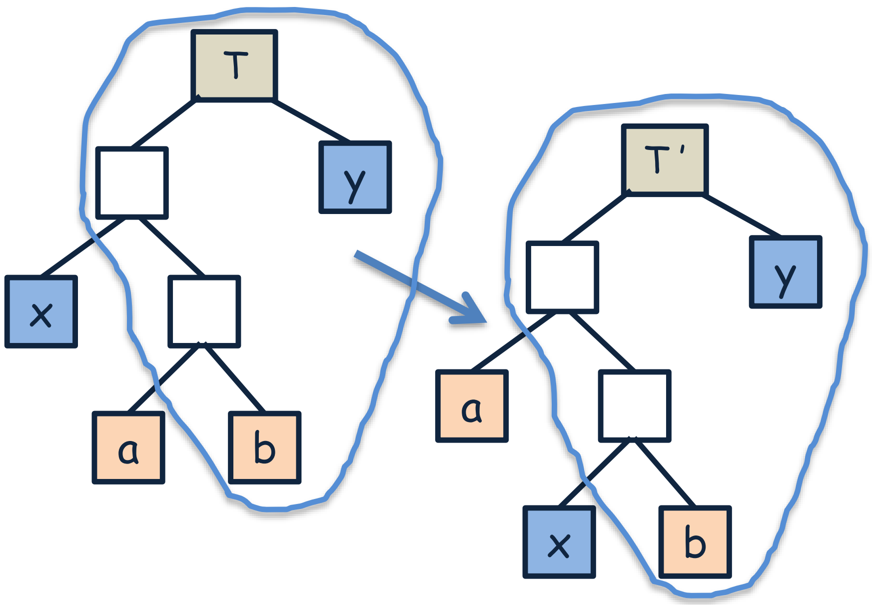
- Because T is a full tree, there should be two symbols a and b that are siblings placed at the lowest level in T
- Since f_x and f_y are the smallest frequencies,
  $f_x, f_y \le f_a, f_b$
- $B(T) = C +$
-

Huffman Coding

Theorem (Greedy-choice property): Let x and y be twe symbols with the smallest frequencies f_x and f_y. There exists an optimal tree where x and y are siblings with the highest depth. (Our greedy approach yields us an optimal solution)

Proof

Assume there is an optimal tree T where x and y are not siblings.



- Because T is a full tree, there should be two symbols a and b that are siblings placed at the lowest level in T
- Since f_x and f_y are the smallest frequencies,

  $f_x , f_y \leq f_a , f_b$

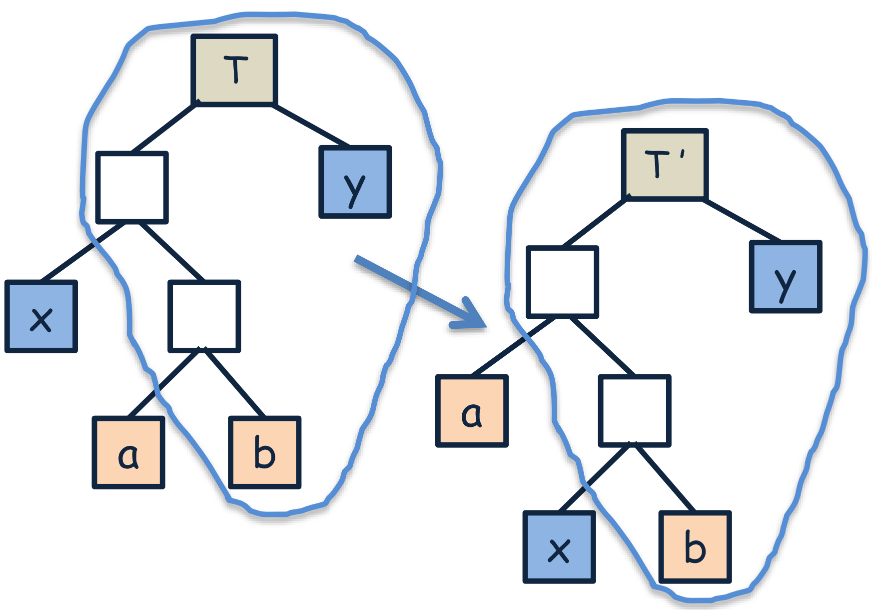
- $B(T) = C + f_x.l_x + f_a.l_a$
-

Huffman Coding

Theorem (Greedy-choice property): Let x and y be twe symbols with the smallest frequencies f_x and f_y. There exists an optimal tree where x and y are siblings with the highest depth. (Our greedy approach yields us an optimal solution)

Proof

Assume there is an optimal tree T where x and y are not siblings.



- Because T is a full tree, there should be two symbols a and b that are siblings placed at the lowest level in T
- Since f_x and f_y are the smallest frequencies,

$$f_x , f_y \leq f_a , f_b$$

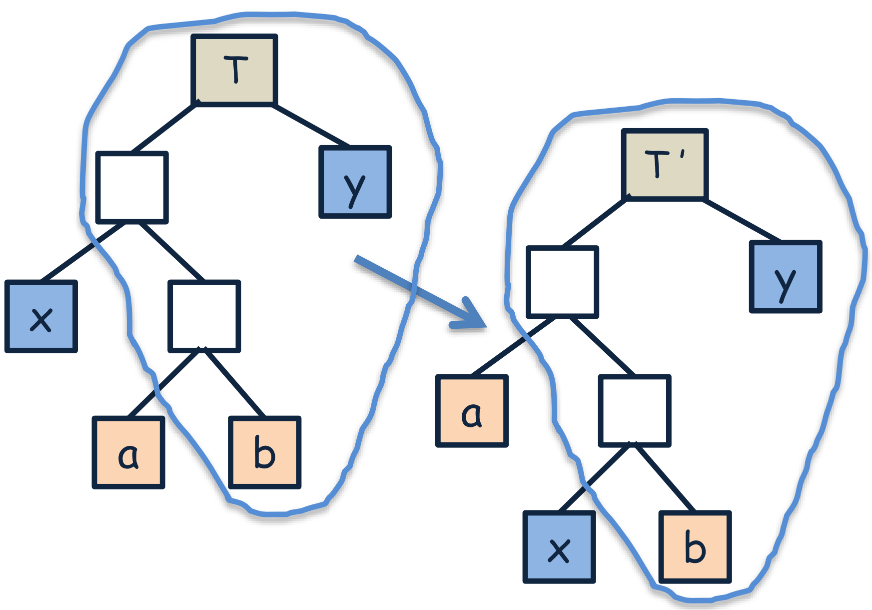
- $B(T) = C + f_x.l_x + f_a.l_a$
- $B(T') = C + f_x.l_a + f_a.l_x$

Huffman Coding

Theorem (Greedy-choice property): Let x and y be twe symbols with the smallest frequencies f_x and f_y. There exists an optimal tree where x and y are siblings with the highest depth. (Our greedy approach yields us an optimal solution)

Proof

Assume there is an optimal tree T where x and y are not siblings.



- Because T is a full tree, there should be two symbols a and b that are siblings placed at the lowest level in T
- Since f_x and f_y are the smallest frequencies,

  $$f_x , f_y \leq f_a , f_b$$

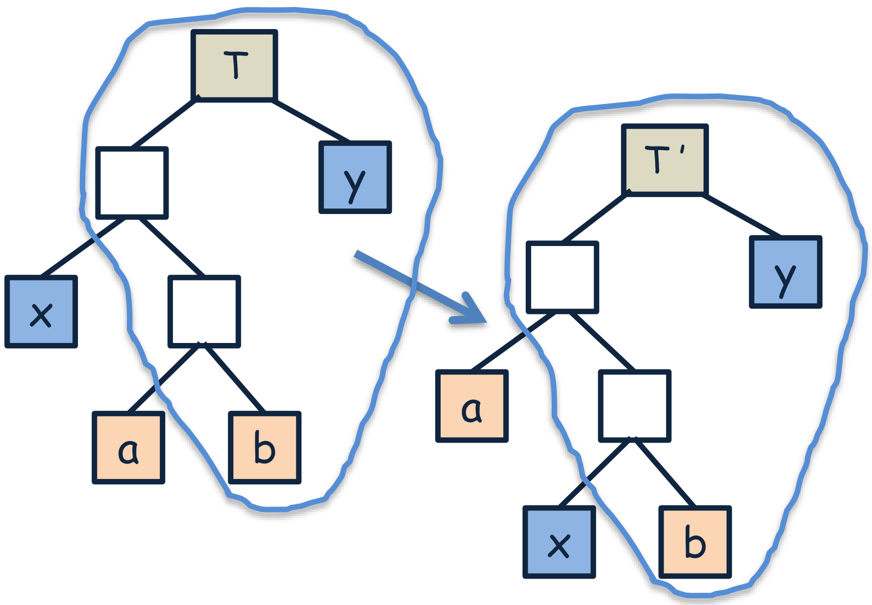
- $B(T) = C + f_x.l_x + f_a.l_a$
- $B(T') = C + f_x.l_a + f_a.l_x$
- $B(T) - B(T') =$

Huffman Coding

Theorem (Greedy-choice property): Let x and y be twe symbols with the smallest frequencies f_x and f_y. There exists an optimal tree where x and y are siblings with the highest depth. (Our greedy approach yields us an optimal solution)

Proof

Assume there is an optimal tree T where x and y are not siblings.



- Because T is a full tree, there should be two symbols a and b that are siblings placed at the lowest level in T
- Since f_x and f_y are the smallest frequencies,

$$f_x , f_y \leq f_a , f_b$$

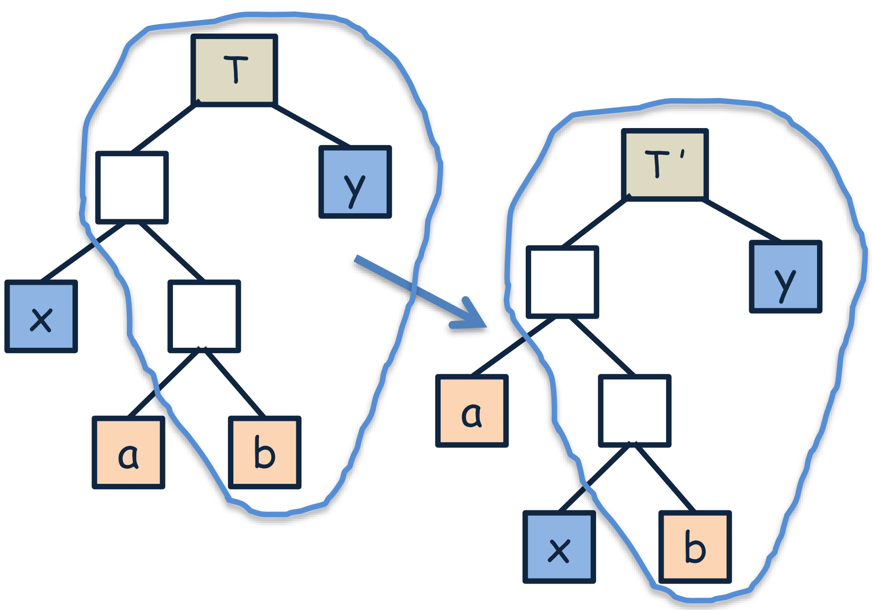
- $B(T) = C + f_x.l_x + f_a.l_a$
- $B(T') = C + f_x.l_a + f_a.l_x$
- $B(T) - B(T') = f_x (l_x - l_a) + f_a (l_a - l_x)$

Huffman Coding

Theorem (Greedy-choice property): Let x and y be twe symbols with the smallest frequencies f_x and f_y. There exists an optimal tree where x and y are siblings with the highest depth. (Our greedy approach yields us an optimal solution)

Proof

Assume there is an optimal tree T where x and y are not siblings.



- Because T is a full tree, there should be two symbols a and b that are siblings placed at the lowest level in T
- Since f_x and f_y are the smallest frequencies,

$$f_x\ ,\ f_y \leq f_a\ ,\ f_b$$

- $B(T) = C + f_x.l_x + f_a.l_a$
- $B(T') = C + f_x.l_a + f_a.l_x$
- $B(T) - B(T') = f_x\,(l_x - l_a) + f_a\,(l_a - l_x)$

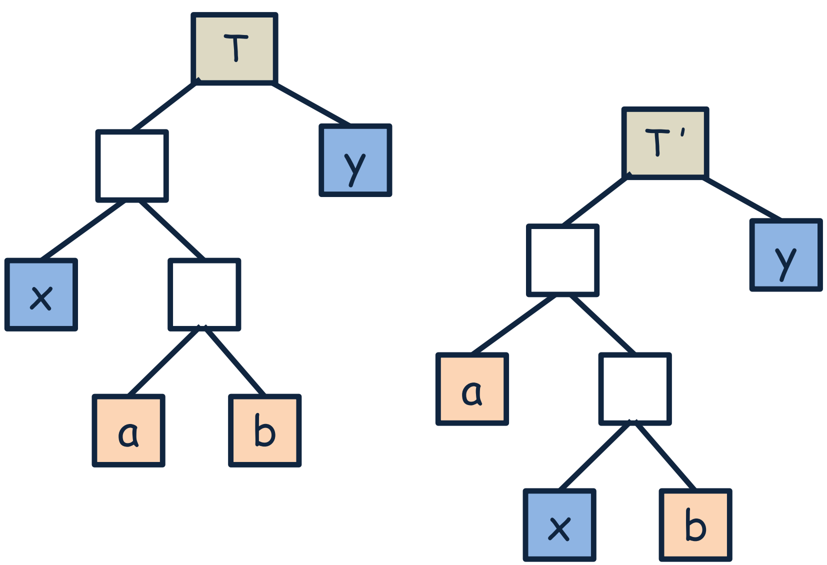
$$= (l_a - l_x)\,(f_a - f_x) \geq 0$$

Huffman Coding

Theorem (Greedy-choice property): Let x and y be twe symbols with the smallest frequencies f_x and f_y. There exists an optimal tree where x and y are siblings with the highest depth. (Our greedy approach yields us an optimal solution)

Proof

Assume there is an optimal tree T where x and y are not siblings.



swap y and b

- Because T is a full tree, there should be two symbols a and b that are siblings placed at the lowest level in T
- Since f_x and f_y are the smallest frequencies,

$$f_x , f_y \leq f_a , f_b$$

- $B(T) = C + f_x.l_x + f_a.l_a$
- $B(T') = C + f_x.l_a + f_a.l_x$
- $B(T) - B(T') = f_x (l_x - l_a) + f_a (l_a - l_x)$

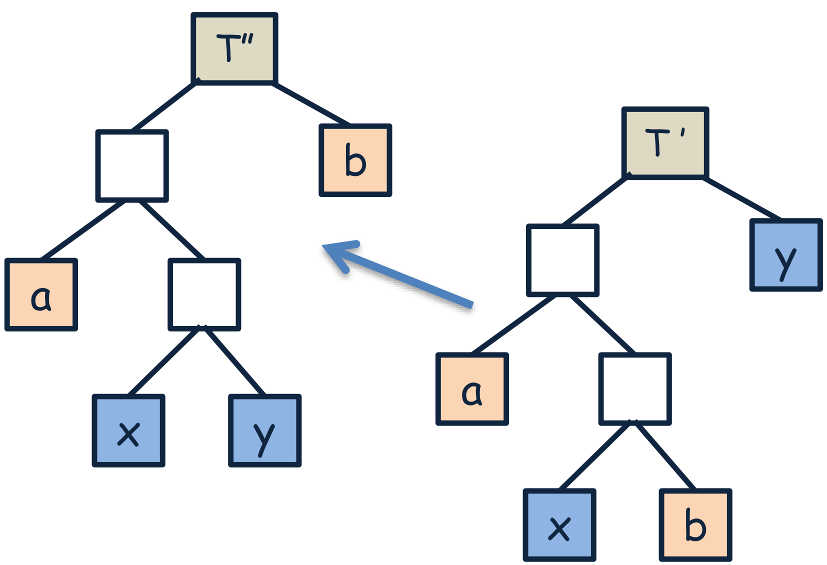
$$= (l_a - l_x)(f_a - f_x) \geq 0$$

Huffman Coding

Theorem (Greedy-choice property): Let x and y be twe symbols with the smallest frequencies f_x and f_y. There exists an optimal tree where x and y are siblings with the highest depth. (Our greedy approach yields us an optimal solution)

Proof

Assume there is an optimal tree T where x and y are not siblings.



swap y and b

- Because T is a full tree, there should be two symbols a and b that are siblings placed at the lowest level in T
- Since f_x and f_y are the smallest frequencies,

$$f_x , f_y \leq f_a , f_b$$

- $B(T) = C + f_x.l_x + f_a.l_a$
- $B(T') = C + f_x.l_a + f_a.l_x$
- $B(T) - B(T') = f_x (l_x - l_a) + f_a (l_a - l_x)$

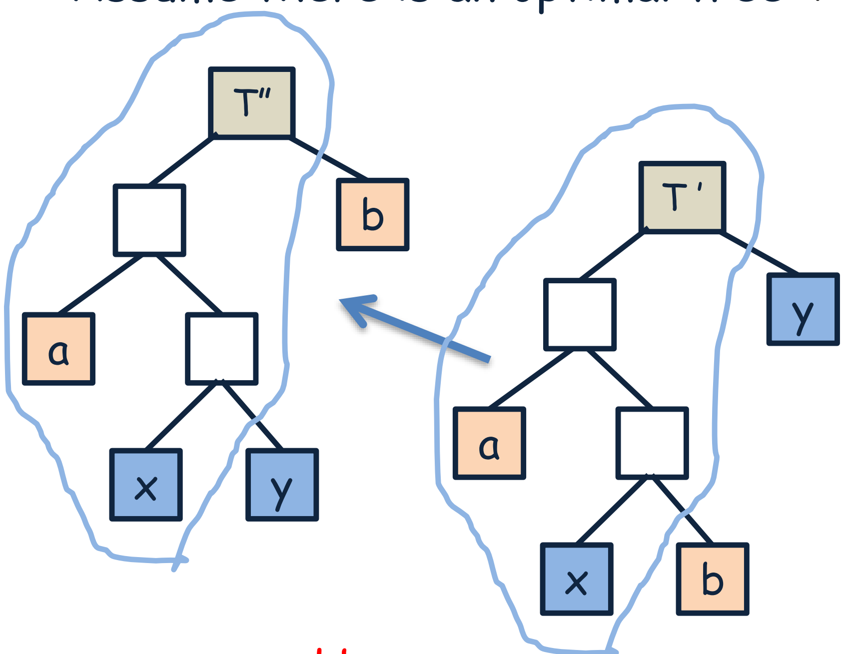
$$= (l_a - l_x)(f_a - f_x) \geq 0$$

Huffman Coding

Theorem (Greedy-choice property): Let x and y be twe symbols with the smallest frequencies f_x and f_y. There exists an optimal tree where x and y are siblings with the highest depth. (Our greedy approach yields us an optimal solution)

Proof

Assume there is an optimal tree T where x and y are not siblings.



swap y and b

- Because T is a full tree, there should be two symbols a and b that are siblings placed at the lowest level in T
- Since f_x and f_y are the smallest frequencies,

$$f_x , f_y \leq f_a , f_b$$

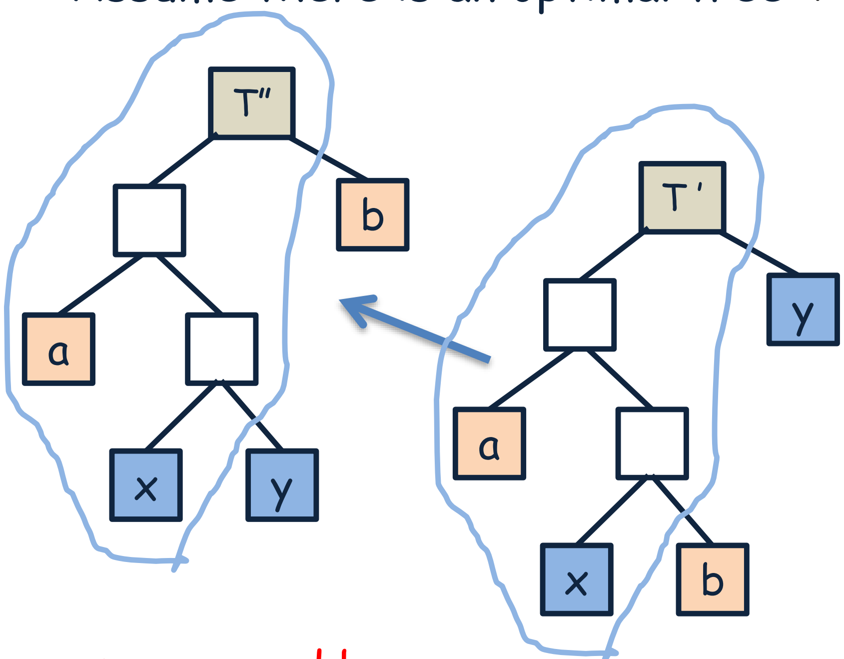
- $B(T) = C + f_x.l_x + f_a.l_a$
- $B(T') = C + f_x.l_a + f_a.l_x$
- $B(T) - B(T') = f_x (l_x - l_a) + f_a (l_a - l_x)$
  $= (l_a - l_x) (f_a - f_x) \geq 0$
- $B(T') - B(T'') \geq 0$

Huffman Coding

Theorem (Greedy-choice property): Let x and y be twe symbols with the smallest frequencies f_x and f_y. There exists an optimal tree where x and y are siblings with the highest depth. (Our greedy approach yields us an optimal solution)

Proof

Assume there is an optimal tree T where x and y are not siblings.



swap y and b

- Because T is a full tree, there should be two symbols a and b that are siblings placed at the lowest level in T
- Since f_x and f_y are the smallest frequencies,

$$f_x \, , f_y \leq f_a \, , f_b$$

- $B(T) = C + f_x.l_x + f_a.l_a$
- $B(T') = C + f_x.l_a + f_a.l_x$
- $B(T) - B(T') = f_x\,(l_x - l_a) + f_a\,(l_a - l_x)$

$$= (l_a - l_x)\,(f_a - f_x) \geq 0$$

- $B(T') - B(T'') \geq 0$

this is a contradiction

Greedy Algorithms

- solve the problem by breaking it a sequence of subproblems

- make the best local choice among all feasible one available on that moment (one choice at a time)
    - your choice does not depend on any future choices or any past choices you have made

- prove that the Greedy Choice Property satisfies. A sequence of locally optimal choices yields a global optimal solution