Greedy Algorithm

Construct a solution iteratively, via sequence of myopic decisions, and hope that everything works out in the end.

Prepared by Pawan K. Mishra

P1. Find maximum number of non overlapping intervals.

IP: Set of intervals (Lectures)

OP: Find the maximum number of intervals that does not overlap each other.

Analogy: Lectures (intervals) and one classroom (single resource)

IP: Set of intervals

OP: Find the maximum number of intervals that does not overlap each other.

Analogy: Lectures (intervals) and one classroom (single resource)

Greedy Choices

Choice 1- Pick the smallest interval, and delete all overlapping intervals, and repeat.

Choice 2- Pick interval which has the least number of overlaps, and delete all overlapping intervals, and repeat.

Choice 3- Pick job which start first (starting time), and delete all overlapping intervals, and repeat.

Construct counter examples for the above greedy choices.

Let’s think how to get design good greedy

Suppose this is an optimal solution, can we add one and remove one?

Let’s think how to get design good greedy

Suppose this is an optimal solution, can we add one and remove one?

I can add this interval (if exists one)

Let’s think how to get design good greedy

Suppose this is an optimal solution, can we add one and remove one?

I can add this interval (if exists one)

Can you think of a greedy choice with this observations?

Let’s think how to get design good greedy

Can you think of an interval which certainly satisfies the previous observation?

Let’s think how to get design good greedy

Can you think of an interval which certainly satisfies the previous observation?

Intervals that end first (if it is not there)… think why??

Algorithm

Sort intervals with Ending time

Add intervals that end first, delete all non compatible events,

Repeat.

Correctness

Proof by contradiction:

$S = \{I_1, I_2, \ldots, I_k\}$ -- returned by our Algorithm. Assume that it is not an optimal solution.

Correctness

Proof by contradiction:

$S = \{I_1, I_2, \ldots, I_k\}$ -- returned by our Algorithm. Assume that it is not an optimal solution.

Fact: We know that there can be multiple optimal solutions, and let OPT be one of the optimal solutions which has maximum number of intervals common with our solution, i.e., S.

Let $I_1, I_2, \ldots, I_m$ be common in both S and OPT. Here $m < k$.

Let $\mathbf{I_{m+1}}$ be (m+1)-th interval in S, and **Z** be the (m+1)-th interval in OPT.

Correctness

Note that any intervals which overlaps with $I_1, I_2, \ldots, I_m$ are not present in both S and OPT.

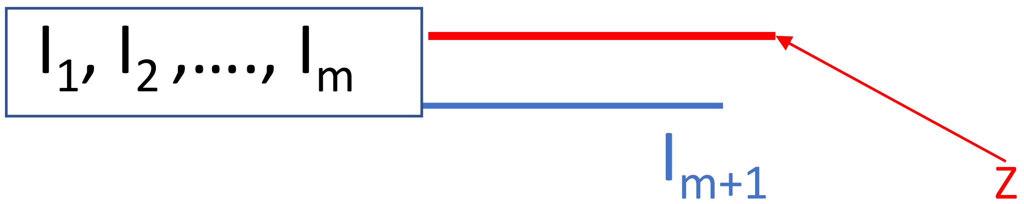
Correctness

Note that any intervals which overlaps with I_1, I_2 ,…., I_m are not present in both S and OPT.

I_{m+1} ends before **Z** ?? Else algorithm will select **Z**.

Correctness

Proof by contradiction:



Correctness

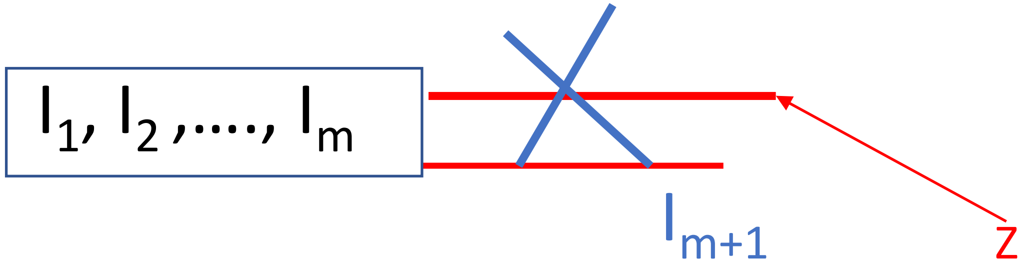
Note that any intervals which overlaps with I_1, I_2 ,…., I_m are not present in both S and OPT.

I_{m+1} ends before **Z** ?? Else algorithm will select **Z**.

Since Z does not overlaps with other any intervals of OPT which ends after interval Z, I_{m+1} will also not overlap with them too.

Correctness

Proof by contradiction:



Correctness

Now, if we replace $\mathbf{I_{m+1}}$ with **Z** in OPT, we will get another optimal solution.

In this optimal solution, I_1, I_2 ,…., I_m , $\mathbf{I_{m+1}}$ are common with our solution S.

This is the contradiction to the fact.

Template

Greedy Algorithm(A,n)

Candidates = rank (A)
solution = ø
for i = 1 to n
c= findbest(Candidates)
solution= solution U {c}
candidates= candidates \ {c}
candidates= revaluate(candidates)
return **solution**

Sorting / or do something to rank
Initialize solution
Some greedy choice
Check feasibility before adding
Remove the selected element
Update the set candidates (optional)

Exchange trick, to prove correctness (Worked often, but not always)

Let A be the greedy algorithm that we are trying to prove correct, and

A(I) the output of A on some input I.

Let O be an optimal solution on input I that is not equal to A(I).

The goal in exchange argument is to show how to modify O to create a new solution O’ with the following properties:

- 1. O’ is at least as good of solution as O (or equivalently O’ is also optimal), and
- 2. O’ is “more like” A(I) than O.

Exchange trick, to prove correctness (Worked often, but not always)

Let A be the greedy algorithm that we are trying to prove correct, and

A(I) the output of A on some input I.

Let O be an optimal solution on input I that is not equal to A(I).

The goal in exchange argument is to show how to modify O to create a new solution O’ with the following properties:

- 1. O’ is at least as good of solution as O (or equivalently O’ is also optimal), and
- 2. O’ is “more like” A(I) than O.

THIS IS THE CREATIVE PART - different for each algorithm/problem.

Exchange trick, to prove correctness (Worked often, but not always)

Let A be the greedy algorithm that we are trying to prove correct, and

A(I) the output of A on some input I.

Let O be an optimal solution on input I that is not equal to A(I).

The goal in exchange argument is to show how to modify O to create a new solution O’ with the following properties:

- 1. O’ is at least as good of solution as O (or equivalently O’ is also optimal), and
- 2. O’ is “more like” A(I) than O.

THIS IS THE CREATIVE PART - different for each algorithm/problem.

One good idea to think of A constructing A(I) over time, and then to look to made the modification at the first point where A makes a choice that is different than what is in O.

Two ways to proceed

First way.

Theorem: The algorithm A solves the problem.

Proof by contradiction: Algorithm A doesn't solve the problem.

- Hence, there must be some input I on which A does not produce an optimal solution. Let the output produced by A be A(I).

Fact: Let O be the optimal solution that is most like A(I).

- If we can show how to modify O to create a new solution O' with the following properties:

1. O' is at least as good of solution as O (and hence O' is also optimal), and
2. O' is more like A(I) than O.

Then we have a contradiction to the choice of O. Thus, the theorem.

Two ways to proceed

Second way.

Theorem: The algorithm A solves the problem.

Let I be an arbitrary instance. Let O be arbitrary optimal solution for I. Assume that we can show how to modify O to create a new solution O’ with the following properties

1. O’ is at least as good of solution as O (and hence O’ is also optimal), and
2. O’ is more like A(I) than O.

Then consider the sequence O’; O’’; O’’’; O’’’’; ….

Each element of this sequence is optimal, and more like A(I) than the proceeding element. Hence, ultimately this sequence must terminate with A(I).

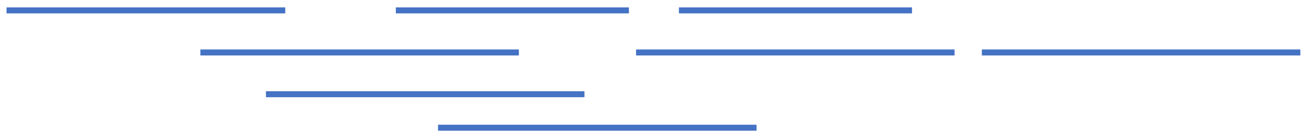
Hence, A(I) is optimal.

P2. Interval Partitioning

Analogy: Lectures (intervals) and classrooms (multiple resources)

IP: Set of intervals (Lectures)

OP: Find the minimum number of resources such that all intervals got served

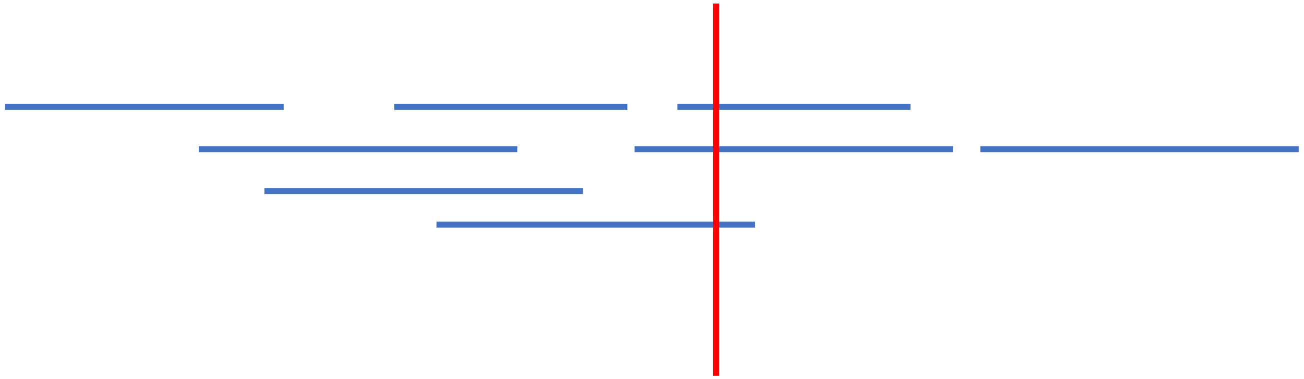


P2. Interval Partitioning

Analogy: Lectures (intervals) and classrooms (multiple resources)

IP: Set of intervals (Lectures)

OP: Find the minimum number of resources such that all intervals got served

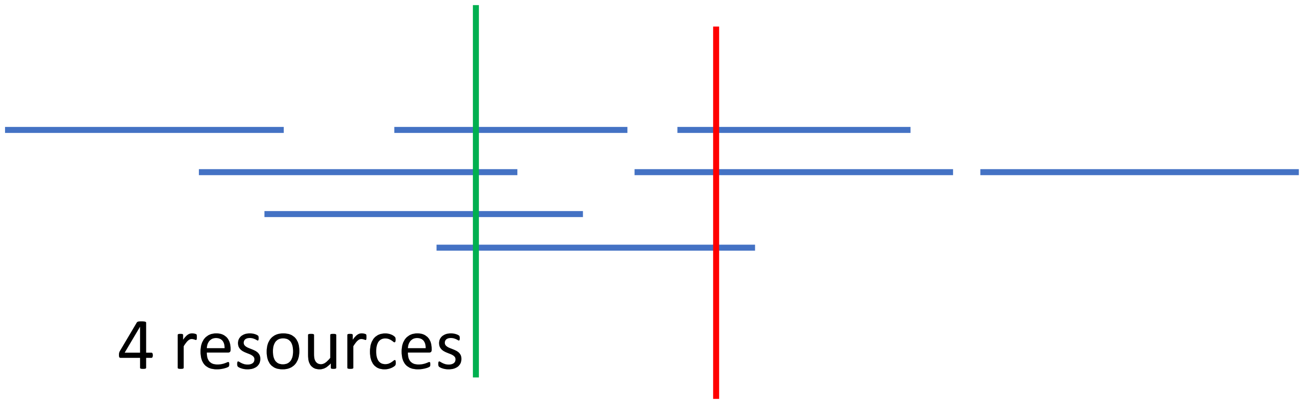


P2. Interval Partitioning

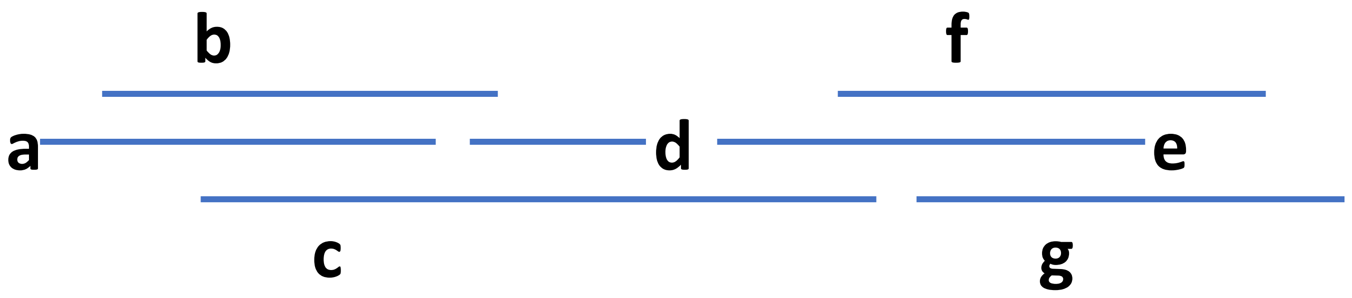
Analogy: Lectures (intervals) and classrooms (multiple resources)

IP: Set of intervals (Lectures)

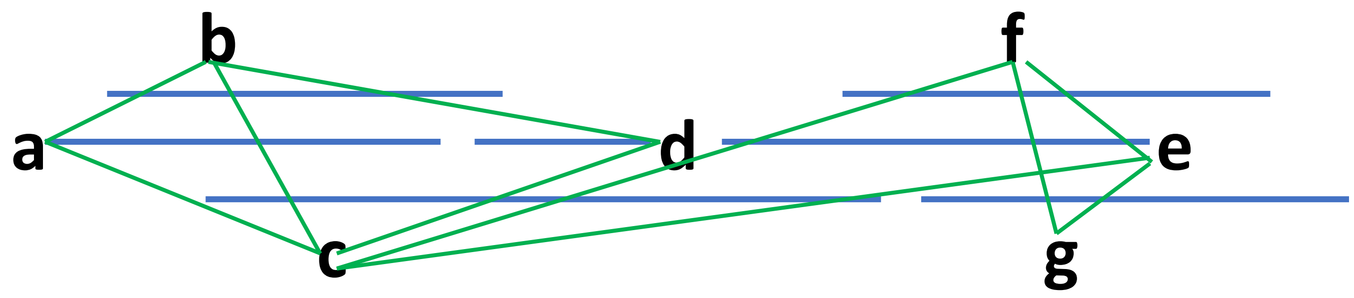
OP: Find the minimum number of resources such that all intervals got served.



Depth: d, we need at least d resources.

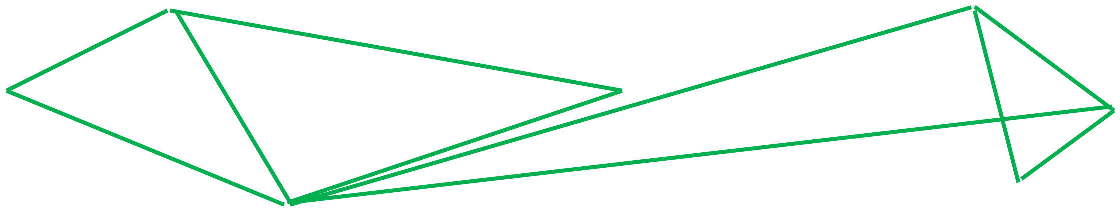


Prepared by *Pawan K. Mishra*

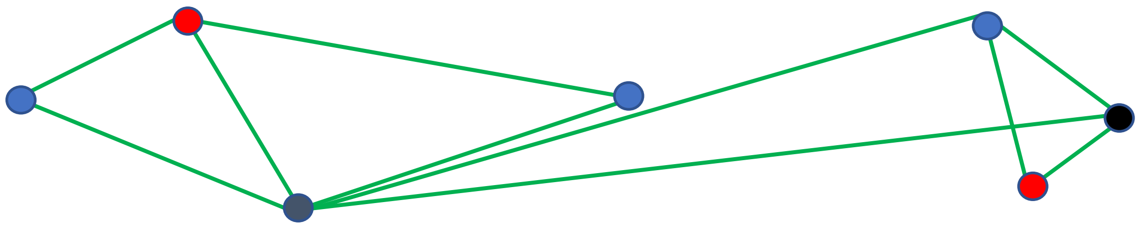


Prepared by *Pawan K. Mishra*

- Interval Graph,
- Let's solve coloring problem on the graph.



Coloring an interval graph (H.W. Check this problem)



Greedy Choices

First greedy choice …

Since last algorithm computes **maximum number of compatible intervals**, place in resource 1,

Repeat on remaining intervals, and puts in 2

Repeat until all intervals have a resource.

Counter example?

Greedy Choices

- Earliest finish time: ascending order of finish time

- Shortest interval: ascending order of (finish – starting) time

Greedy Choices

- Earliest finish time: ascending order of finish time

Counter example

- Shortest interval: ascending order of finish-starting time

Counter example

Algorithm with greedy choice

EARLIESTSTARTTIMEFIRST(n, s_1, s_2, …, s_n, f_1, f_2, …, f_n)

SORT lectures by start time so that $s_1 \leq s_2 \leq \ldots \leq s_n$.

$d \leftarrow 0$ number of allocated classrooms

FOR $j = 1$ TO n

  IF lecture j is compatible with some classroom

    Schedule lecture j in any such classroom k.

  ELSE

    Allocate a new classroom $d + 1$.

    Schedule lecture j in classroom $d + 1$.

    $d \leftarrow d + 1$

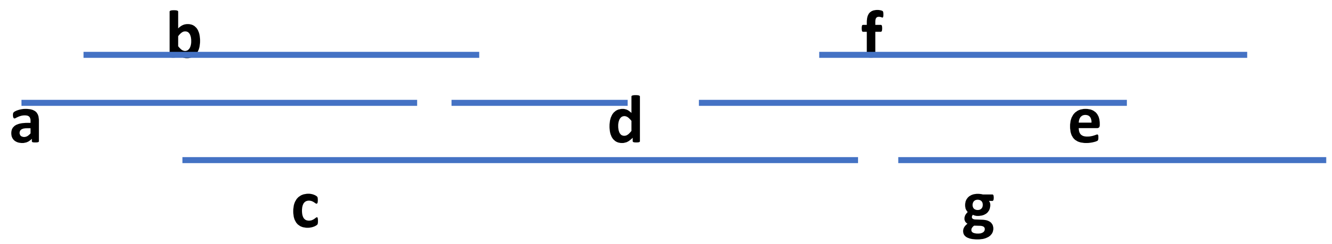
RETURN schedule.

Prepared by Pawan K. Mishra

Example

Sort by start time,

For an interval, give it to the resource where it can fit.

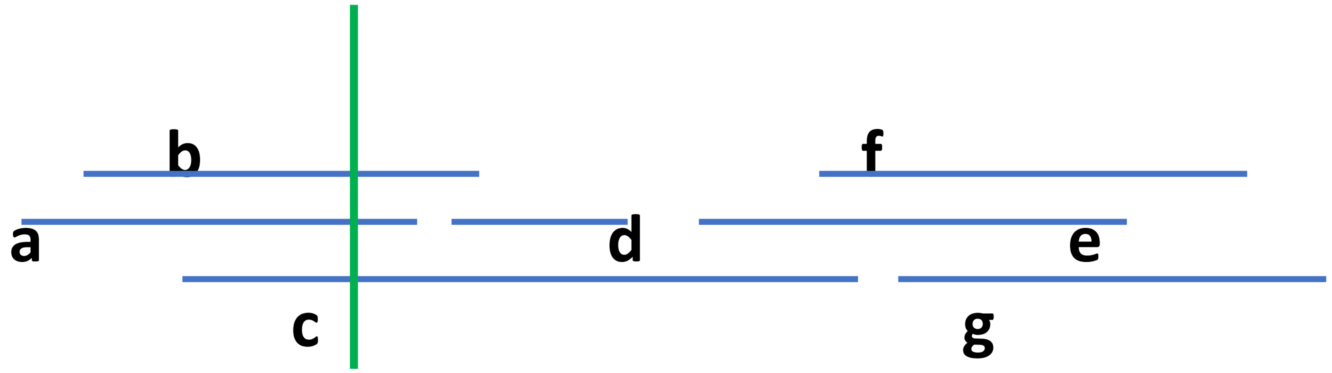


a, b, c, d, e, f, g– Sorted with start time. Resource 1,2,3,1,1,2,3 --- 3 resources required

Example

Sort by start time,

For an interval, give it to the resource where it can fit.



a, b, c, d, e, f, g– Sorted with start time. Resource 1,2,3,1,1,2,3 --- 3 resources required

Algorithm uses “d” resources

Proof by contradiction.

W.L.O.G assume that algorithm takes d+1 number of resources.

Fact: d is the depth

Let X=[a,b] be the interval which served by (d+1)-th resource by the algo.

1. Why X is being getting served at that phase of the algo?
2. Why X is being served by (d+1)-th resource ?

Algorithm uses “d” resources

Proof by contradiction.

W.L.O.G assume that algorithm takes d+1 number of resources.

Fact: d is the depth

Let X=[a,b] be the interval which served by (d+1)-th resource by the algo.

1. Why X is being getting served at that phase of the algo?
2. Why X is being served by (d+1)-th resource ?

All d resources are occupied

Greedy Choice

Algorithm uses “d” resources

Proof by contradiction.

W.L.O.G assume that algorithm takes d+1 number of resources.

Fact: d is the depth

Let X=[a,b] be the interval which served by (d+1)-th resource by the algo.

1. Why X is being getting served at that phase of the algo?
2. Why X is being served by (d+1)-th resource ?

All d resources are occupied, Greedy Choice

So, at time “a”, the starting of interval X, all “d” resources are occupied. Thus, we need d+1 resources. This implies that at time “a”, the depth is more than “d”, which is contradiction to the fact. Hence proved.

Algorithm gives optimal result.

Algorithm takes number of resource equal to the depth, and we know that any feasible solution requires at least resources equal to the number of depth.

P3: Job Scheduling

Time in the system for a job: Waiting time + Processing time

IP: Set of n jobs={j_1,j_2,…j_n} with processing time P(j_1), P(j_2), …P(j_n), and a single resource.

OP: Schedule jobs on one resource s.t. it minimizes the total time in the system.

Example

Job 1- 5 units , Job 2- 10 units, Job 3- 4 units

[1,2,3]- 5+(5+10)+(5+10+4)=39

[1,3,2]- 5+(5+4)+(5+4+10)=33

[2,1,3]- 10+(10+5)+(10+5+4)=44

[2,3,1]- 10+(10+4)+(10+4+5)=43

[3,1,2]- 4+(4+5)+(4+5+10)=32

[3,2,1]- 4+(4+10)+(4+10+5)=37

Example

Job 1- 5 units , Job 2- 10 units, Job 3- 4 units

[1,2,3]- 5+(5+10)+(5+10+4)=39

[1,3,2]- 5+(5+4)+(5+4+10)=33

[2,1,3]- 10+(10+5)+(10+5+4)=44

[2,3,1]- 10+(10+4)+(10+4+5)=43

[3,1,2]- 4+(4+5)+(4+5+10)=32

[3,2,1]- 4+(4+10)+(4+10+5)=37

[3,1,2] is optimal

Developing Intuition

Some arbitrary order of jobs…

Finish time of job 1 = t_1

Finish time of job 2 = $t_1 + t_2$

Finish time of job 3 = $t_1 + t_2 + t_3$

.

.

.

Finish time of job n = $t_1 + t_2 + \ldots + t_n$

Total Finishing Time = $nt_1 + (n-1)t_2 + \ldots + t_n$

Developing Intuition

Some arbitrary order of jobs…

Finish time of job 1 = t_1

Finish time of job 2 = $t_1 + t_2$

Finish time of job 3 = $t_1 + t_2 + t_3$

Can you guess the greedy choice ?

.

.

.

Finish time of job n = $t_1 + t_2 + \ldots + t_n$

Total Finishing Time = $nt_1 + (n-1)t_2 + \ldots + t_n$

Developing Intuition

Some arbitrary order of jobs…

Finish time of job 1 = t_1

Finish time of job 2 = $t_1 + t_2$

Finish time of job 3 = $t_1 + t_2 + t_3$

.

.

.

Finish time of job n = $t_1 + t_2 + \ldots + t_n$

Total Finishing Time = $nt_1 + (n-1)t_2 + \ldots + t_n$

Can you guess the greedy choice ?

Shortest Job First

The only schedule that minimizes the total time in the system is one that schedules jobs in nondecreasing order by service time.

Use exchange trick…..with contradiction

Assume that it is not true.

Fact: OPT is an optimal solution.

Note that in OPT, there exists two consecutive jobs X and Y, s.t. X is being served before Y, and P(x) > P(y).

Else OPT will be same schedule which follows the shortest job sequence order.

Now suppose we interchange X and Y in OPT, and else remain same. What happens?

The only schedule that minimizes the total time in the system is one that schedules jobs in nondecreasing order by service time.

Use exchange trick…..with contradiction

Assume that it is not true.

Fact: OPT is an optimal solution.

Note that in OPT, there exists two consecutive jobs X and Y, s.t. X is being served before Y, and P(x) > P(y).

Else OPT will be same schedule which follows the shortest job sequence order.

Now suppose we interchange X and Y in OPT, and else remain same. What happens?

Order in OPT …………………. XY………………….

New order OPT' …………………. YX………………….

The only schedule that minimizes the total time in the system is one that schedules jobs in nondecreasing order by service time.

Use exchange trick…..with contradiction

Assume that it is not true.

Fact: OPT is an optimal solution.

Note that in OPT, there exists two consecutive jobs X and Y, s.t. X is being served before Y, and P(x) > P(y).

Else OPT will be same schedule which follows the shortest job sequence order.

Now suppose we interchange X and Y in OPT, and else remain same. What happens?

Order in OPT	………………… XY………………….	No need to worry about these jobs
New order OPT’	………………… YX………………….	

The only schedule that minimizes the total time in the system is one that schedules jobs in nondecreasing order by service time.

Use exchange trick…..with contradiction

Assume that it is not true.

Fact: OPT is an optimal solution.

Note that in OPT, there exists two consecutive jobs X and Y, s.t. X is being served before Y, and P(x) > P(y).

Else OPT will be same schedule which follows the shortest job sequence order.

Now suppose we interchange X and Y in OPT, and else remain same. What happens?

Order in OPT	………………… XY………………….	In OPT, to do Job Y, need to wait till Job X being served.
New order OPT’	………………… YX………………….	In OPT’, to do Job X, need to wait till Job Y being served.

Prepared by Pawan K. Mishra

The only schedule that minimizes the total time in the system is one that schedules jobs in nondecreasing order by service time.

Use exchange trick…..with contradiction

Assume that it is not true.

Fact: OPT is an optimal solution.

Note that in OPT, there exists two consecutive jobs X and Y, s.t. X is being served before Y, and P(x) > P(y).

Else OPT will be same schedule which follows the shortest job sequence order.

Now suppose we interchange X and Y in OPT, and else remain same. What happens?

Order in OPT	………………… XY…………………	In OPT, to do Job Y, need to wait till Job X being served.
New order OPT’	………………… YX…………………	In OPT’, to do Job X, need to wait till Job Y being served.

Total Time (OPT’)= Total Time(OPT)-P(X)+P(Y)

The only schedule that minimizes the total time in the system is one that schedules jobs in nondecreasing order by service time.

Use exchange trick…..with contradiction

Assume that it is not true.

Fact: OPT is an optimal solution.

Note that in OPT, there exists two consecutive jobs X and Y, s.t. X is being served before Y, and P(x) > P(y).

Else OPT will be same schedule which follows the shortest job sequence order.

Now suppose we interchange X and Y in OPT, and else remain same. What happens?

Order in OPT …………………. XY…………………. In OPT, to do Job Y, need to wait till Job X being served.

New order OPT' …………………. YX…………………. In OPT', to do Job X, need to wait till Job Y being served.

Total Time (OPT')= Total Time(OPT)-P(X)+P(Y) . Now, we know that P(x) > P(y). Thus, Total Time (OPT') < Total Time(OPT)

CONTADICTION TO THE OPTIMALITY

P4. Job Scheduling with m servers

GENERALIZE THE PREVIOUS IDEA TO SOLVE THIS PROBLEM.

P5. Fractional Knapsack Problem

I/P: n items, each item i has profit p_i and size s_i

: a bag with capacity B

O/P: maximize the profit without violating the capacity constraint.

Need to fit items in bag, here we can use fraction of items…

P5. Fractional Knapsack Problem

Prepared by Pawan K. Mishra

P5. Fractional Knapsack Problem

Unless, he is Vijay Mallya

P5. Fractional Knapsack Problem

I/P: n items, each item i has profit p_i and size s_i

: a bag with capacity B

O/P: maximize the profit without violating the capacity constraint.

Need to fit items in bag, here we can use fraction of items…

Find x_1 for item i_i , x_2 for item i_2 , ….. , x_n for item i_n s.t.

$$x_1 s_1 + x_2 s_2 + \dots + x_n s_n \le B$$

$$\text{Each } 0 \le x_i \le 1$$

GOAL: Max $x_1 p_1 + x_2 p_2 + \dots + x_n p_n$

P5. Fractional Knapsack Problem

I/P: n items, each item i has profit p_i and size s_i

: a bag with capacity B

O/P: maximize the profit without violating the capacity constraint.

Need to fit items in bag, here we can use fraction of items…

Another variant 0/1 knapsack…where fraction of items are not allowed.

Greedy choices

Increasing profit

Decreasing size

Lets find a better greedy choice….

Let suppose that bag is full.

Can we remove some fraction of an item i by adding some fraction of item j

Such that profit increases?

Lets find a better greedy choice….

Let suppose that bag is full.

Can we remove some fraction of an item i by adding some fraction of item j
Such that profit increases?

Fractions: y_1 for item i_i , y_2 for item i_2 , ….. , y_n for item i_n

$y_1\, s_1 + y_2\, s_2 + \ldots + y_n\, s_n = B$ (by assumption)

Lets find a better greedy choice….

Let suppose that bag is full.

Can we remove some fraction of an item k by adding some fraction of item l

Such that profit increases?

y_1 for item i_i , y_2 for item i_2 , ….. , y_n for item i_n

$y_1 s_1 + y_2 s_2 + \ldots. + y_n s_n = B$ (by assumption)

y_1	y_2	y_k	y_l	y_n
y_1	y_2	$(y_k - e')$	$(y_l + e)$	y_n

Lets find a better greedy choice….

Let suppose that bag is full.

Can we remove some fraction of an item k by adding some fraction of item l

Such that profit increases?

y_1 for item i_i , y_2 for item i_2 , ….. , y_n for item i_n

$$y_1 s_1 + y_2 s_2 + \ldots + y_n s_n = B \quad \text{(by assumption)}$$

y_1	y_2	y_k	y_l	y_n
y_1	y_2	$(y_k - e')$	$(y_l + e)$	y_n

$$y_1 s_1 + y_2 s_2 + \ldots + (y_k - e')s_k + \ldots + (y_l + e)s_l + y_n s_n = B$$

Lets find a better greedy choice….

Let suppose that bag is full.

Can we remove some fraction of an item i by adding some fraction of item j

Such that profit increases?

y_1 for item i_i , y_2 for item i_2 , ….. , y_n for item i_n

$y_1 s_1 + y_2 s_2 + \ldots + y_n s_n = B$ (by assumption)

y_1	y_2	y_k	y_l	y_n
y_1	y_2	$(y_k - e')$	$(y_l + e)$	y_n

$$y_1 s_1 + y_2 s_2 + \ldots + (y_k - e')s_k + \ldots + (y_l + e)s_l + y_n s_n = B$$

$$y_1 s_1 + y_2 s_2 + \ldots + y_n s_n + (-e's_k + es_l) = B$$

$$B + (-e's_k + es_l) = B$$

$$es_l = e's_k \Rightarrow \frac{s_l}{s_k} = \frac{e'}{e}$$

Lets find a better greedy choice….

Let suppose that bag is full.

Can we remove some fraction of an item i by adding some fraction of item j

Such that profit increases?

y_1 for item i_i, y_2 for item i_2, ….. , y_n for item i_n

$y_1 s_1 + y_2 s_2 + \ldots + y_n s_n = B$ (by assumption)

y_1	y_2	y_k	y_l	y_n
y_1	y_2	$(y_k - e')$	$(y_l + e)$	y_n

$y_1 s_1 + y_2 s_2 + \ldots + (y_k - e')s_k + \ldots + (y_l + e)s_l + y_n s_n = B$

$y_1 s_1 + y_2 s_2 + \ldots + y_n s_n + (-e's_k + es_l) = B$

$B + (-e's_k + es_l) = B$

$es_l = e's_k \Rightarrow \frac{s_l}{s_k} = \frac{e'}{e}$

- Profit

New profit

$= y_1 p_1 + y_2 p_2 + \ldots + (y_k - e')p_k + \ldots + (y_l + e)p_l + y_n p_n$

$=$ old profit $- e'p_k + ep_l$

$=$ old profit $+ ep_l - e'p_k$

New profit > old profit ??

Prepared by *Pawan K. Mishra*

Lets find a better greedy choice....

Let suppose that bag is full.

Can we remove some fraction of an item i by adding some fraction of item j

Such that profit increases?

y_1 for item i_i , y_2 for item i_2 , ….. , y_n for item i_n

$y_1 s_1 + y_2 s_2 + \ldots + y_n s_n = B$ (by assumption)

y_1	y_2	y_k	y_l	y_n
y_1	y_2	$(y_k - e')$	$(y_l + e)$	y_n

$$y_1 s_1 + y_2 s_2 + \ldots + (y_k - e')s_k + \ldots + (y_l + e)s_l + y_n s_n = B$$

$$y_1 s_1 + y_2 s_2 + \ldots + y_n s_n + (-e's_k + es_l) = B$$

$$B + (-e's_k + es_l) = B$$

$$es_l = e's_k \Rightarrow \frac{s_l}{s_k} = \frac{e'}{e}$$

- Profit

New profit

$$= y_1 p_1 + y_2 p_2 + \ldots + (y_k - e')p_k + \ldots + (y_l + e)p_l + y_n p_n$$

$= \text{old profit} - e'p_k + ep_l$

$= \text{old profit} + ep_l - e'p_k$

New profit > old profit ??

if $ep_l - e'p_k > 0$

$\mathbf{p_l / p_k > e'/e}$

Lets find a better greedy choice....

Let suppose that bag is full.

Can we remove some fraction of an item i by adding some fraction of item j

Such that profit increases?

y_1 for item i_i , y_2 for item i_2 , ... , y_n for item i_n

$y_1 s_1 + y_2 s_2 + \ldots + y_n s_n = B$ (by assumption)

y_1	y_2	y_k	y_l	y_n
y_1	y_2	$(y_k - e')$	$(y_l + e)$	y_n

$y_1 s_1 + y_2 s_2 + \ldots + (y_k - e')s_k + \ldots + (y_l + e)s_l + y_n s_n = B$

$y_1 s_1 + y_2 s_2 + \ldots + y_n s_n + (-e's_k + es_l) = B$

$B + (-e's_k + es_l) = B$

$es_l = e's_k \Rightarrow \dfrac{s_l}{s_k} = \dfrac{e'}{e}$

- Profit

New profit

$= y_1 p_1 + y_2 p_2 + \ldots + (y_k - e')p_k + \ldots + (y_l + e)p_l + y_n p_n$

$=$ old profit $- e'p_k + ep_l$

$=$ old profit $+ ep_l - e'p_k$

New profit > old profit ??

if $ep_l - e'p_k > 0$

$p_l / p_k > e'/e$

$p_l / p_k > e'/e = s_l / s_k$

$p_l / s_l > p_k / s_k$

Algorithm

For each item i, $\text{score}_i = \text{profit}_i / \text{size}_i$

Order items by decreasing scores.

Pick them in this order till bag is filled

Algorithm

For each item i, $score_i = profit_i / size_i$

Order items by decreasing scores.

Pick them in this order till bag is filled

1, 1 , 1, 1, 1, α , 0, 0, 0, 0, 0

(0, 1)

Proof of correctness

Exchange argument… using contradiction

Suppose for contradiction that solution proposed by greedy algorithm $X_1, X_2, \ldots, X_n$ in not an optimal solution.

Fact: Take an optimal solution fractions $Z_1, Z_2, \ldots, Z_n$ which differ at some position with greedy solution.

Let i be there first index at which $X_i \neq Z_i$. By the design of our algorithm, it must be that $X_i > Z_i$.

By the optimality of OPT, there must exist an item $j > i$ such that $Z_j > X_j$.

Consider a new solution $O' = \{Z_1', Z_2', \ldots, Z_n'\}$ where $Z_k' = Z_k$, except at $k \neq i,j$.

Suppose we increase $Z_i' = Z_i' + e_1$ and decrease $Z_j' = Z_j - e_2$. Here $e_1, e_2 > 0$.

Prepared by *Pawan K. Mishra*

Proof of correctness

Exchange argument… using contradiction

Suppose for contradiction that solution proposed by greedy algorithm $X_1, X_2, \ldots, X_n$ in not an optimal solution.

Fact: Take an optimal solution fractions $Z_1, Z_2, \ldots, Z_n$ which differ at some position with greedy solution.

Let i be there first index at which $X_i \neq Z_i$. By the design of our algorithm, it must be that $X_i > Z_i$.

By the optimality of OPT, there must exist an item $j > i$ such that $Z_j > X_j$.

Consider a new solution $O' = \{Z_1', Z_2', \ldots, Z_n'\}$ where $Z_k' = Z_k$, except at $k \neq i,j$.

Suppose we increase $Z_i' = Z_i' + e_1$ and decrease $Z_j' = Z_j - e_2$. ➔ $e_1 s_i = e_2 s_j$.

Proof of correctness

Exchange argument… using contradiction

Suppose for contradiction that solution proposed by greedy algorithm $X_1, X_2,\ldots, X_n$ in not an optimal solution.

Fact: Take an optimal solution fractions $Z_1, Z_2,\ldots, Z_n$ which differ at some position with greedy solution.

Let i be there first index at which $X_i \neq Z_i$. By the design of our algorithm, it must be that $X_i > Z_i$.

By the optimality of OPT, there must exist an item j > i such that $Z_j > X_j$.

Consider a new solution $O'=\{Z_1', Z_2',\ldots, Z_n'\}$ where $Z_k' = Z_k$, except at $k \neq i,j$.

Suppose we increase $Z_i'=Z_i'+e_1$ and decrease $Z_j'=Z_j-e_2$. ➔ $e_1 s_i = e_2 s_j$.

O' has better profit than O?

Proof of correctness

Exchange argument… using contradiction

Suppose for contradiction that solution proposed by greedy algorithm $X_1, X_2, \ldots, X_n$ in not an optimal solution.

Fact: Take an optimal solution fractions $Z_1, Z_2, \ldots, Z_n$ which differ at some position with greedy solution.

Let i be there first index at which $X_i \neq Z_i$. By the design of our algorithm, it must be that $X_i > Z_i$.

By the optimality of OPT, there must exist an item j > i such that $Z_j > X_j$.

Consider a new solution $O' = \{Z_1', Z_2', \ldots, Z_n'\}$ where $Z_k' = Z_k$, except at $k \neq i,j$.

Suppose we increase $Z_i' = Z_i' + e_1$ and decrease $Z_j' = Z_j - e_2$. ➔ $e_1 s_i = e_2 s_j$.

Do O' has better profit than O?

new profit by $O' = Z_1' p_1 + Z_2' p_2 + \ldots + (Z_i' + e_1) p_i + \ldots.. + (Z_j' - e_2) p_j + \ldots. + Z_n' p_n$

$= \text{old profit by } O + e_1 p_i - e_2 p_j$

$= \text{old profit by } O + e_1 p_i - (e_1 s_i/s_j) p_j \quad = \text{old profit by } O + e_1 p_i - (e_1 s_i) p_j/s_j$

$= \text{old profit by } O + s_i e_1 (p_i/s_i) - (e_1 s_i) p_j/s_j = \text{old profit by } O + s_i e_1 (p_i/s_i - p_j/s_j)$

Proof of correctness

Exchange argument… using contradiction

Suppose for contradiction that solution proposed by greedy algorithm $X_1, X_2, \ldots, X_n$ in not an optimal solution.

Fact: Take an optimal solution fractions $Z_1, Z_2, \ldots, Z_n$ which differ at some position with greedy solution.

Let i be there first index at which $X_i \neq Z_i$. By the design of our algorithm, it must be that $X_i > Z_i$.

By the optimality of OPT, there must exist an item j > i such that $Z_j > X_j$.

Consider a new solution $O' = \{Z_1', Z_2', \ldots, Z_n'\}$ where $Z_k' = Z_k$, except at $k \neq i,j$.

Suppose we increase $Z_i' = Z_i' + e_1$ and decrease $Z_j' = Z_j - e_2$. ➔ $e_1 s_i = e_2 s_j$.

Do O' has better profit than O?

because of our algo

this is > 0

new profit by $O' = Z_1' p_1 + Z_2' p_2 + \ldots + (Z_i' + e_1) p_i + \ldots + (Z_j' - e_2) p_j + \ldots + Z_n' p_n$

$= \text{old profit by } O + e_1 p_i - e_2 p_j$

$= \text{old profit by } O + e_1 p_i - (e_1 s_i / s_j) p_j \quad = \text{old profit by } O + e_1 p_i - (e_1 s_i) p_j / s_j$

$= \text{old profit by } O + s_i e_1 (p_i / s_i) - (e_1 s_i) p_j / s_j = \text{old profit by } O + s_i e_1 (p_i/s_i - p_j/s_j)$

Proof of correctness

Exchange argument… using contradiction

Suppose for contradiction that solution proposed by greedy algorithm $X_1, X_2,\ldots, X_n$ in not an optimal solution.

Fact: Take an optimal solution fractions $Z_1, Z_2,\ldots, Z_n$ which differ at some position with greedy solution.

Let i be there first index at which $X_i \neq Z_i$. By the design of our algorithm, it must be that $X_i > Z_i$.

By the optimality of OPT, there must exist an item j > i such that $Z_j > X_j$.

Consider a new solution $O' = \{Z_1', Z_2',\ldots, Z_n'\}$ where $Z_k' = Z_k$, except at $k \neq i,j$.

Suppose we increase $Z_i' = Z_i' + e_1$ and decrease $Z_j' = Z_j - e_2$. ➔ $e_1 s_i = e_2 s_j$.

Do O' has better profit than O?

because of our algo

this is > 0

new profit by $O' = Z_1' p_1 + Z_2' p_2 + \ldots + (Z_i' + e_1) p_i + \ldots + (Z_j' - e_2) p_j + \ldots + Z_n' p_n$

= old profit by O + $e_1 p_i - e_2 p_j$

= old profit by O + $e_1 p_i - (e_1 s_i/s_j) p_j$ = old profit by O + $e_1 p_i - (e_1 s_i) p_j/s_j$

= old profit by O + $s_i e_1 (p_i/s_i) - (e_1 s_i) p_j/s_j$ = old profit by O + $s_i e_1 \mathbf{(p_i/s_i - p_j/s_j)}$

new profit by O' > old profit by O . Hence the contradiction to the optimality

Features and Bugs of the Greedy paradigm

- Easy to come up with one or more greedy algorithms

- Easy to analyse the running time

- Hard to establish correctness

- □↦□□□□□□ ⋮∘↔ □□□□□□ ↦ℓ□∘□↔ℏ∓← ↦□ □∘↔ ↦ℓ↦□← ⇕∘□⇕↔

Features and Bugs of the Greedy paradigm

- Easy to come up with one or more greedy algorithms
- Easy to analyse the running time
- Hard to establish correctness
- Warning: Most greedy algorithms are not always correct.

Future Greedy Algorithms on Graphs

Prim’s Algorithm

Kruskal’s Algorithm

Dijkstra‘s Algorithm

P6. Find minimum vertex cover on trees

Sort with minimum degree and store in L

Select node n with minimum degree

Delete all edges adjacent to the node n, remove node n too

Update L

P6. Find minimum vertex cover on trees

Sort with minimum degree and store in L

Select node n with minimum degree

Delete all edges adjacent to the node n, remove node n too

Update L

Counter Example

P7. Find maximum independent set on trees

Sort with maximum degree and store in L

Select node n with maximum degree

Delete all the nodes adjacent to n, and n too.

Update L

Prove the correctness via contradiction (**exchange trick**)

P8. Minimizing Lateness

IP: Single resource processes one job at a time. Job j requires t_j units of processing time and deadline is at time d_j.

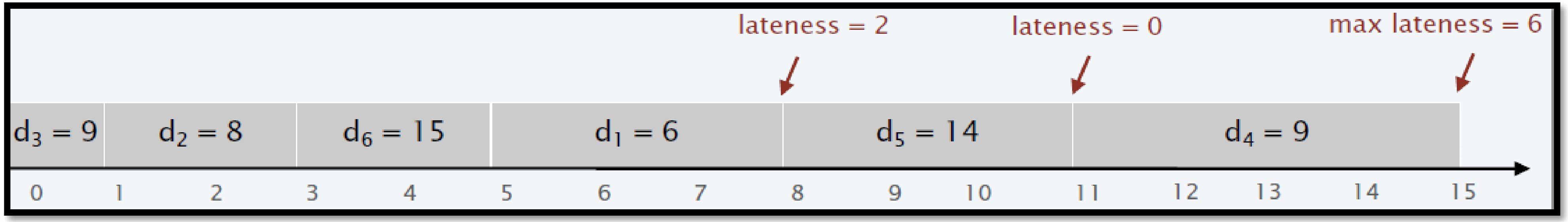
If j starts at time s_j, it finishes at time $f_j = s_j + t_j$.

Lateness of job j: $L_j = \max\{0, f_j - d_j\}$.

Goal: schedule all jobs to minimize maximum lateness $L = \max L_j$.

Example

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15



Greedy choices

Counter examples

- Shortest processing time first

  Ascending order of processing time t_j

	1	2
t_j	1	10
d_j	100	10

- Smallest slack first

  Ascending order of $d_j - t_j$

	1	2
t_j	1	10
d_j	2	10

Greedy choice which works

- This works.

SORT n jobs so that $d_1 \leq d_2 \leq \ldots \leq d_n$.

$t \leftarrow 0$

FOR $j = 1$ TO n

  Assign job j to interval $[t, t + t_j]$.

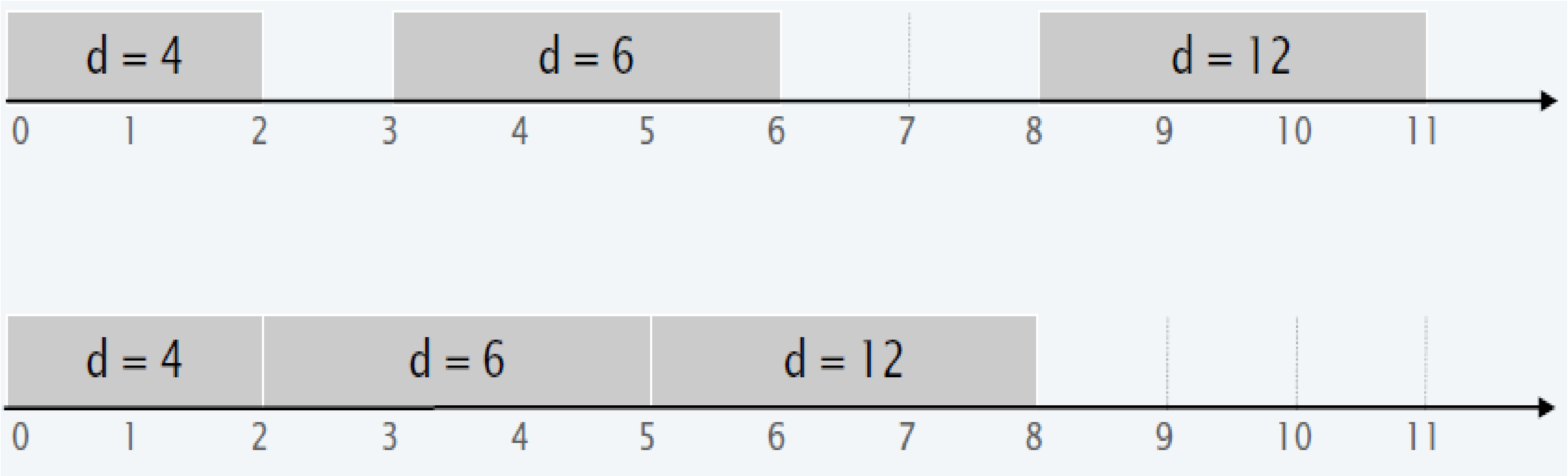
  $s_j \leftarrow t$; $f_j \leftarrow t + t_j$

  $t \leftarrow t + t_j$

RETURN intervals $[s_1, f_1], [s_2, f_2], \ldots, [s_n, f_n]$.

Few observations

Observation 1 . There exists an optimal schedule with no idle time.



Few observations

Observation 1 . There exists an optimal schedule with no idle time.

Observation 2. The greedy schedule has no idle time.

Few observations

An inversion in schedule S is a pair of jobs i and j such that deadline time of i is less than j , i.e., $d_i < d_j$,

but j scheduled before i.

Observation 3. Greedy schedule has no inversions.

Observation 4. If a schedule (with no idle time) has an inversion, it has one with a pair of inverted jobs scheduled consecutively.

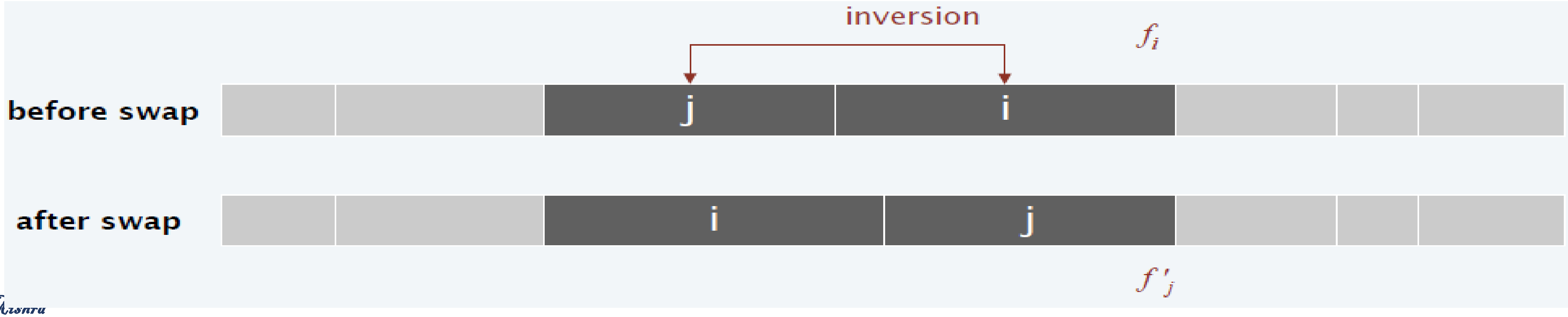
Claim: Swapping adjacently scheduled inverted jobs doesn't increase lateness but reduces #inversions by one

Proof: Let L_a and L_a' denotes the lateness of job a before and after the swap respectively.

$L=\max\{L_a\}$=maximum lateness before swap $L'=\max\{L'_a\}$=maximum lateness after swap

Clearly, $L_k= L_k'$ for all $k \neq i, j$. (no change in finish timing)

Also, clearly $L_i' \leq L_i$. (i moved early)



Prepared by Pawan K. Mishra

Claim: Swapping adjacently scheduled inverted jobs doesn't increase lateness but reduces #inversions by one

Proof: Let L_a and L_a' denotes the lateness of job a before and after the swap respectively.

$L=\max\{L_a\}$=maximum lateness before swap $L'=\max\{L'_a\}$=maximum lateness after swap

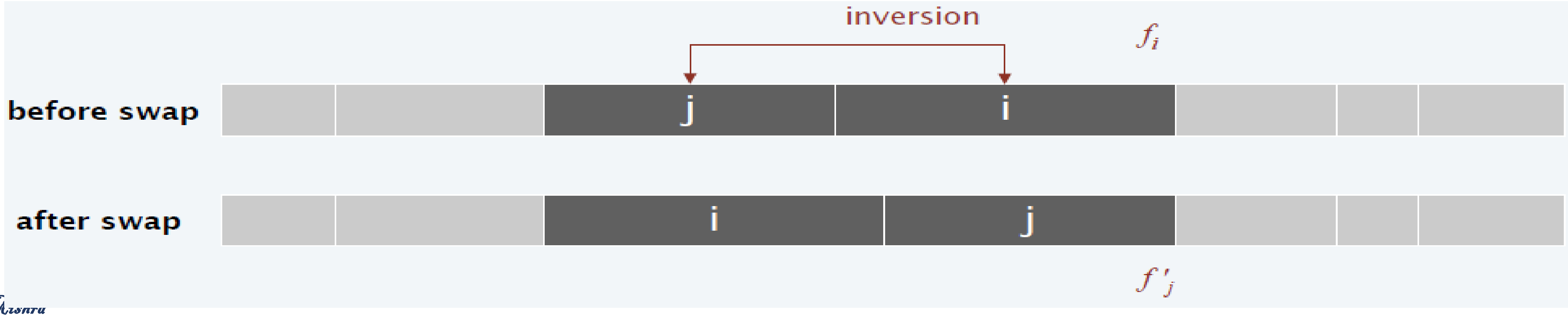
Clearly, $L_k = L_k'$ for all $k \neq i, j$.

Also, clearly $L_i' \leq L_i$.

$L_j' = f_j' - d_j = f_i - d_j \leq f_i - d_i = L_i$

(due to the inversions)

$L' = \max\{ L_i', L_j', L_k'\} \leq \max\{ L_i, L_i, L_k\} \leq L$



Prepared by Pawan K. Mishra

Greedy schedule S is optimal.

Use exchange trick…..with contradiction

Suppose for contradiction that S in not an optimal solution.

FACT: Consider an optimal schedule OPT with fewest inversions among all optimal schedules

Greedy schedule S is optimal.

Use exchange trick…..with contradiction

Suppose for contradiction that S in not an optimal solution.

FACT: Consider an optimal schedule OPT with fewest inversions among all optimal schedules

Because S is not optimal, OPT has inversions.

By observation 1, OPT has no idle time.

By Observation 4 , it has an adjacent inversion (i,j).

By Observation 5, swapping the adjacent pair keeps the schedule optimal but reduces the # inversions by 1.

Hence contradiction to the fact that OPT is an optimal solution with fewest inversions among all optimal solutions. Thus, the theorem.

P9. Huffman code

Kleinberg and Tardos: Algorithm design

Prepared by Pawan K. Mishra

P9. Huffman code

Computer Data Encoding:

How do we represent data in binary?

Historical Solution:

Fixed length codes.

Encode every symbol by a unique binary string of a fixed length.

Examples: ASCII (7 bit code),

EBCDIC (8 bit code), …

ASCII Example:

AABCAA

A	A	B	C	A	A
1000001	1000001	1000010	1000011	1000001	1000001

Total space usage in bits:

Assume an ℓ bit fixed length code.

For a file of n characters

Need $n\ell$ bits.

Variable Length codes

Idea: In order to save space, use less bits for frequent characters and more bits for rare characters.

Example: suppose alphabet of 3 symbols:

{ A, B, C }.

suppose in file: 1,000,000 characters.

Need 2 bits for a fixed length

code for a total of

2,000,000 bits.

Variable Length codes - example

Suppose the frequency distribution of the characters is:

A	B	C
999,000	500	500

Encode:

A	B	C
0	10	11

Note that the code of A is of length 1, and the codes for B and C are of length 2

Total space usage in bits:

Fixed code: 1,000,000 x 2 = 2,000,000

Variable code: 999,000 x 1
\+ 500 x 2
500 x 2

1,001,000

A savings of almost 50%

How do we decode?

In the fixed length, we know where every character starts, since they all have the same number of bits.

Example: A = 00

B = 01

C = 10

00|00|00|01|01|10|10|10|01|10|01|00|00|10|10

A A A B B C C C B C B A A C C

How do we decode?

In the variable length code, we use an idea called Prefix code, where no code is a prefix of another.

Example: A = 0
B = 10
C = 11

None of the above codes is a prefix of another.

How do we decode?

Example: A = 0

B = 10

C = 11

So, for the string:

A A A B B C C C B C B A A C C the encoding:

0 0 0101011111101110 0 01111

Prefix Code

Example: A = 0

B = 10

C = 11

Decode the string

0 0 101011111110111 0 0 01111

A A B B C C C B C B A A C C

Example

$S=\{a,b,c,d,e\}$ with $f(a)= 0.32$, $f(b)= 0.25$, $f(c)= 0.20$, $f(d)= 0.18$, $f(e)= 0.05$

Example

S={a,b,c,d,e} with f(a)= 0.32 , f(b)= 0.25 , f(c)= 0.20 , f(d)= 0.18, f(e)= 0.05

CBE: Constant bit encoding, 3 bits,

Average number of bits per letter

= 0.32*3+ 0.25*3 +0.20*3+ 0.18*3+ 0.05*3=3

λ_1:Prefix code: a=11, b=01, c=001, d=10,e=000,

Average number of bits per letter

= 0.32*2+0.25*2+0.20*3+0.18*2+0.05*3=2.25

Example

S={a,b,c,d,e} with f(a)= 0.32 , f(b)= 0.25 , f(c)= 0.20 , f(d)= 0.18, f(e)= 0.05

CBE: Constant bit encoding, 3 bits**,**

Average number of bits per letter

= 0.32*3+ 0.25*3 +0.20*3+ 0.18*3+ 0.05*3=3

λ_1:Prefix code: a=11, b=01, c=001, d=10,e=000,

Average number of bits per letter

= 0.32*2+0.25*2+0.20*3+0.18*2+0.05*3=2.25

λ_2 :Prefix code: a=11, b=01, c=01, d=001,e=000,

Average number of bits per letter

=0.32*2+0.25*2+0.20*2+0.18*2+0.05*3=2.23

Example

S={a,b,c,d,e} with f(a)= 0.32 , f(b)= 0.25 , f(c)= 0.20 , f(d)= 0.18, f(e)= 0.05

CBE: Constant bit encoding, 3 bits,

ABL(CBE)= 0.32*3+ 0.25*3 +0.20*3+ 0.18*3+ 0.05*3=3

λ_1:Prefix code: a=11, b=01, c=001, d=10,e=000,

ABL(λ_1)=**0.32*2+0.25*2+0.20*3+0.18*2+0.05*3=2.25**

λ_2 :Prefix code: a=11, b=01, c=01, d=001,e=000,

ABL(λ_2)=**0.32*2+0.25*2+0.20*2+0.18*2+0.05*3=2.23**

Given a set $S=\{x_1, x_2, \ldots, x_n\}$ of n characters and frequency of each character x_i is $f(x_i)$.

And $f(x_1)+f(x_2)+\ldots+f(x_n)=1$

Find a prefix code of S, λ, such that $\lambda(x_1)f(x_1)+ \lambda(x_2)f(x_2) +\ldots+ \lambda(x_n)f(x_n)$ is minimized.

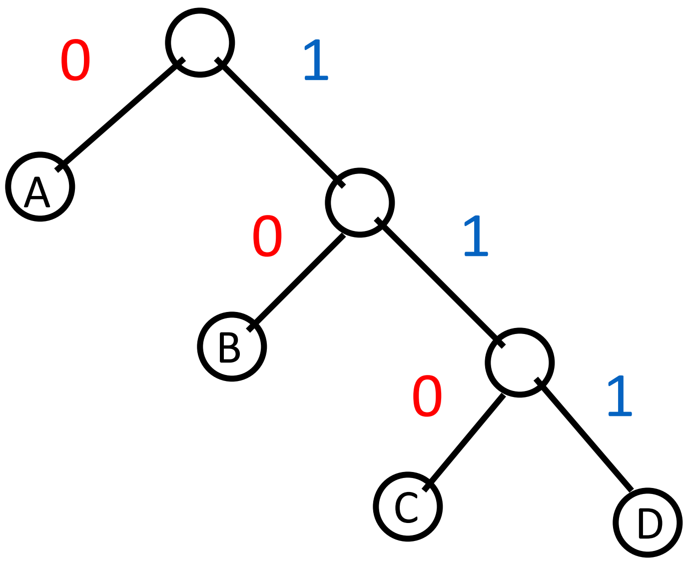
$$ABL(\lambda) = \lambda(x_1)f(x_1)+ \lambda(x_2)f(x_2) +\ldots+ \lambda(x_n)f(x_n)$$

Idea

Consider a binary tree with no. of leaves same as the no. of characters, and with:

0 meaning a left turn

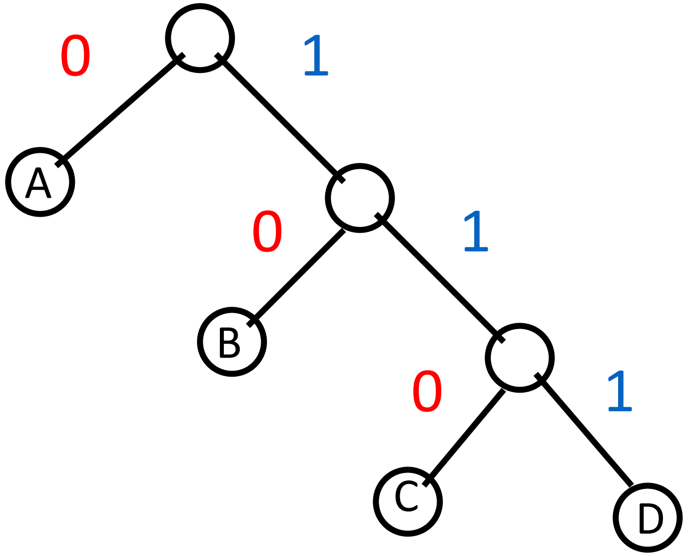
1 meaning a right turn.



Idea

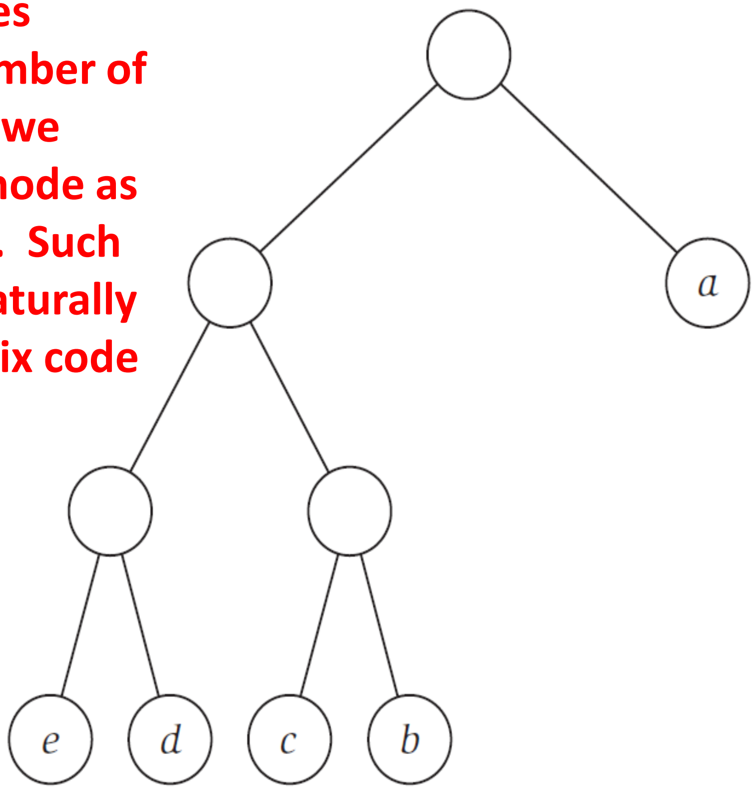
Consider a binary tree with no. of leaves same as the no. of characters, and with:

A : 0
B : 10
C : 110
D : 111

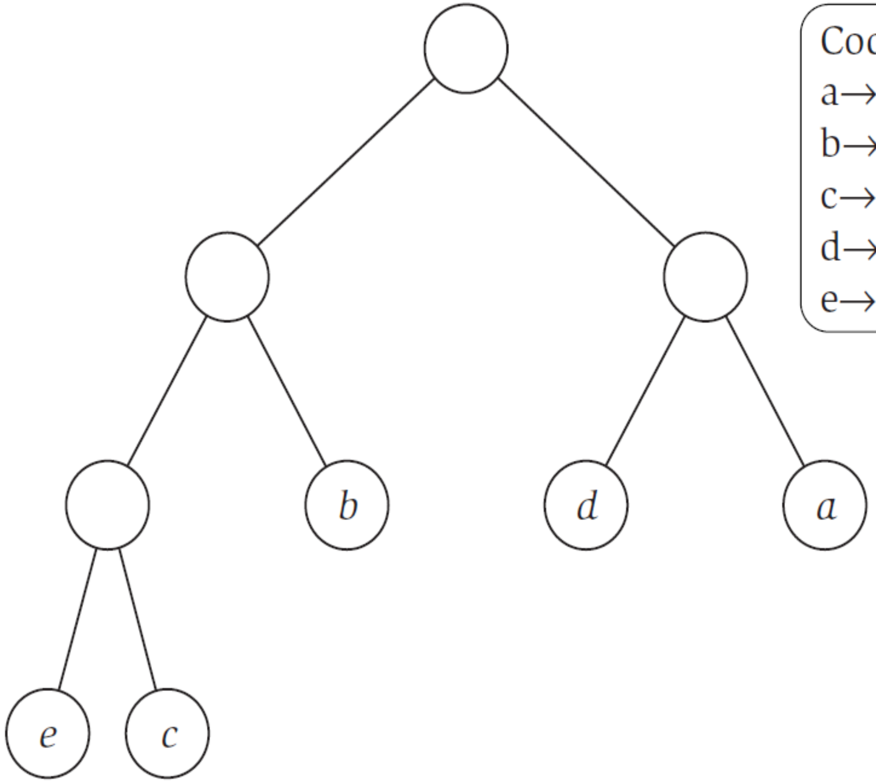


Representing prefix code as binary tree....

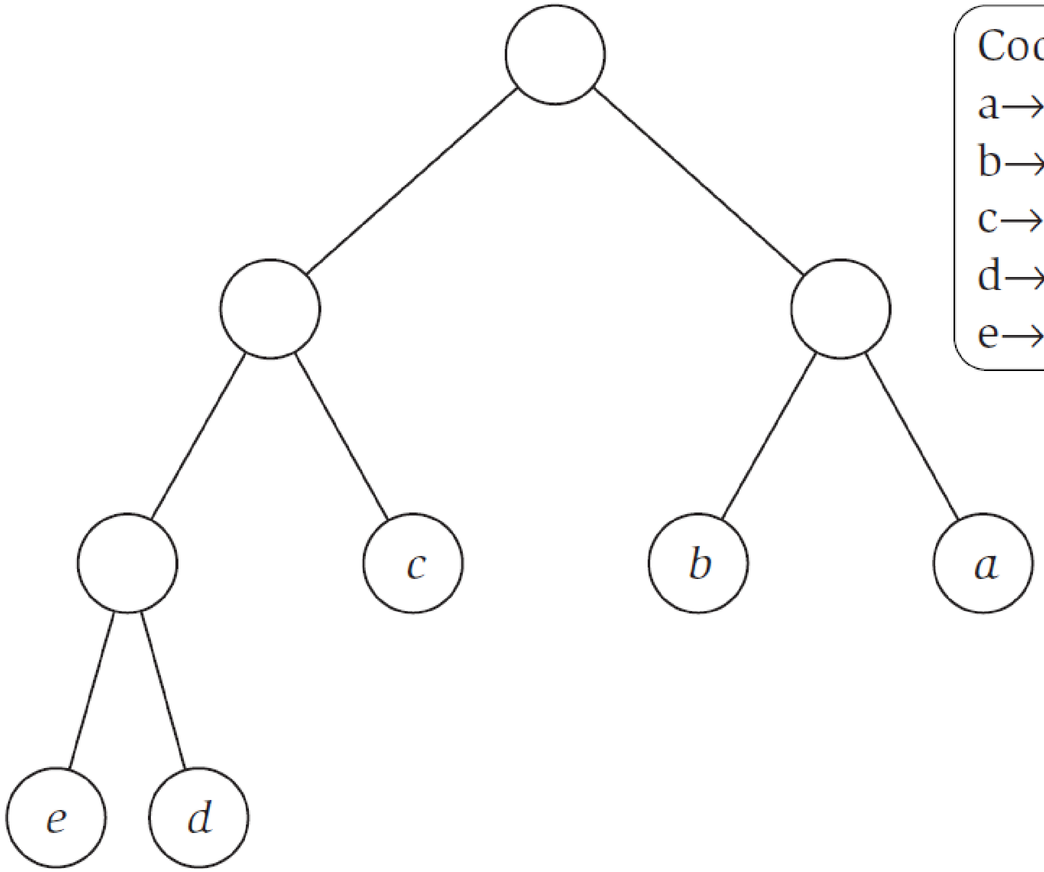
Binary tree, where number of leaves equal to the number of characters, and we label each leaf node as a character of S. Such labelled tree, naturally describes a prefix code of S.



Code:
a→1
b→011
c→010
d→001
e→000



Code:
a→11
b→01
c→001
d→10
e→000



Code:
a→11
b→10
c→01
d→001
e→000

Another way to define the same problem

Prefix tree is equivalent to binary tree with n leaves, where n is the number of character.

goal is to minimize $h(x_1)f(x_1)+ h(x_2)f(x_2) +\ldots+ h(x_n)f(x_n)$, where h(.) is the height/depth of a node in the binary tree.

INTUTION to proceed further

Observation 1. The encoding of S constructed from T is prefix code.

One liner proof for the above statement in the Tardos book

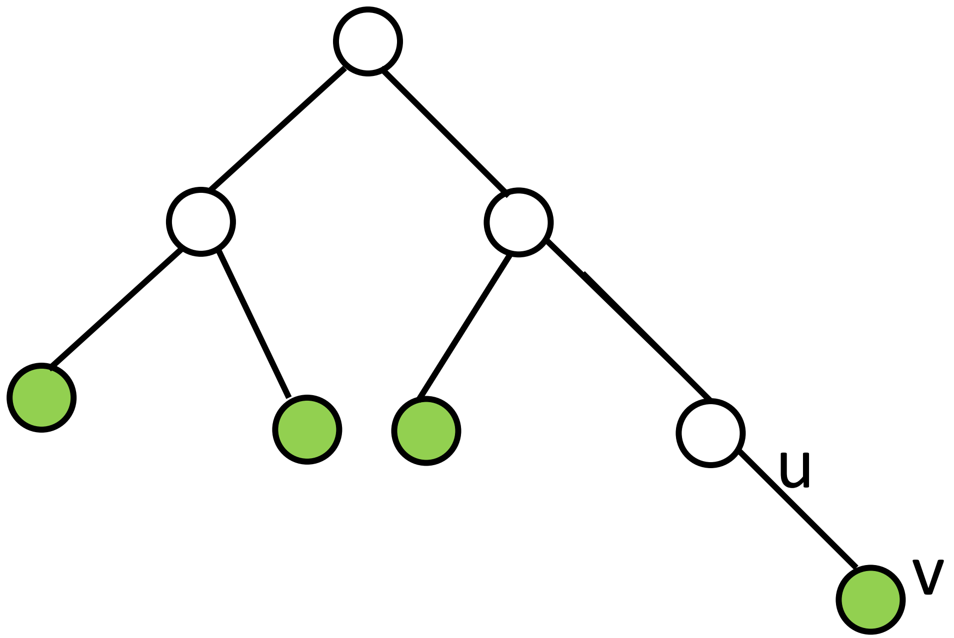
This is a prefix code, since each of the leaves has a path ending in it, without continuation.

Observation 2. Similarly, given a prefix code we can construct binary tree recursively.

Observation 3. The binary tree corresponding to the optimal prefix code is full (each node that is not a leaf has two children).

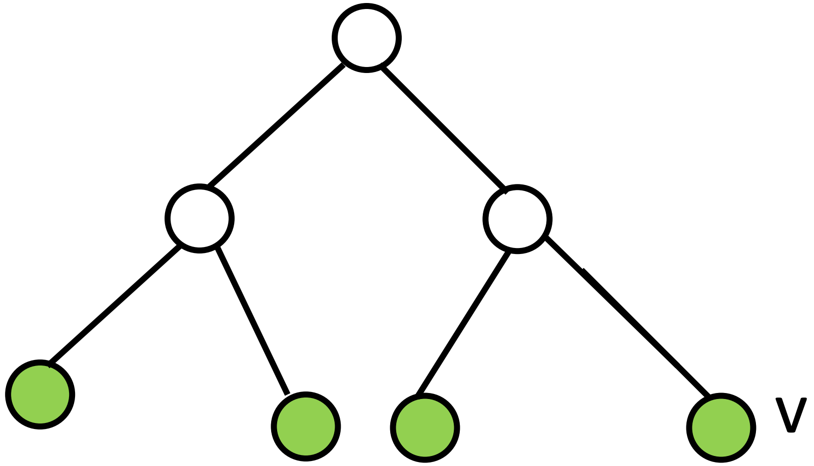
Observation 3. The binary tree corresponding to the optimal prefix code is full (each node that is not a leaf has two children).

Let binary tree T denote optimal prefix code, and suppose it contains a node u with exactly one child v.



Observation 3. The binary tree corresponding to the optimal prefix code is full (each node that is not a leaf has two children).

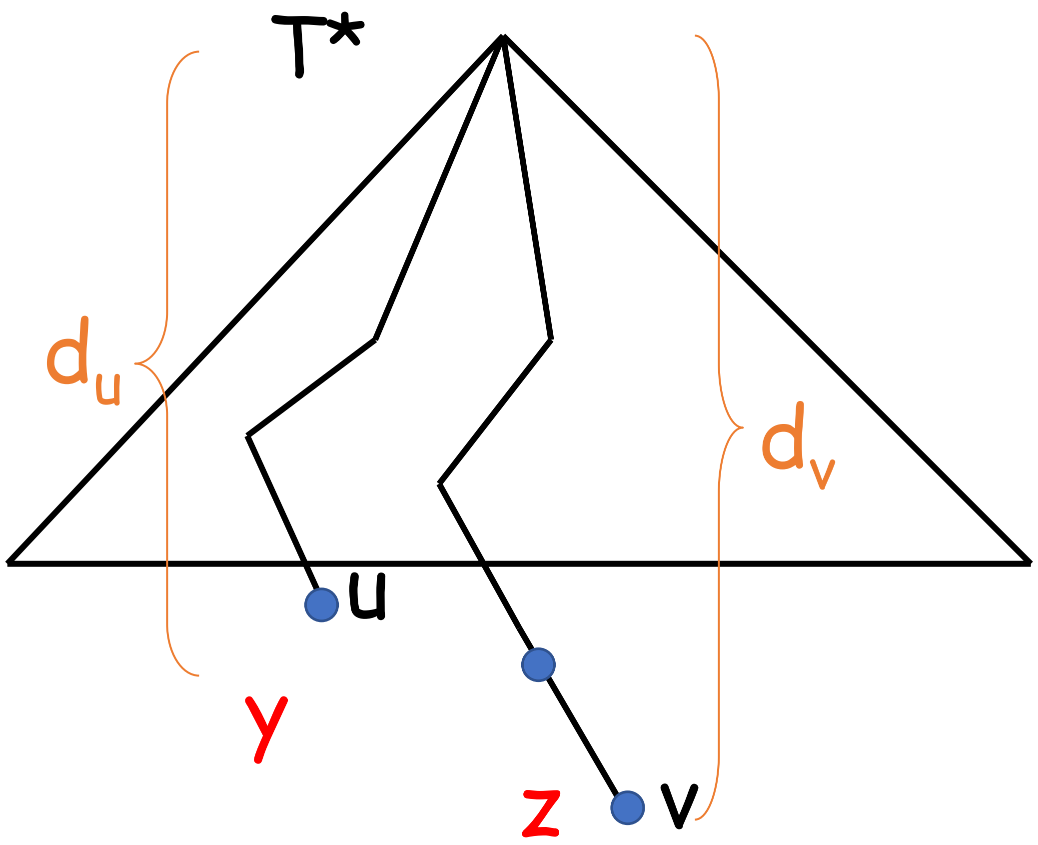
Let binary tree T denote optimal prefix code, and suppose it contains a node u with exactly one child v.



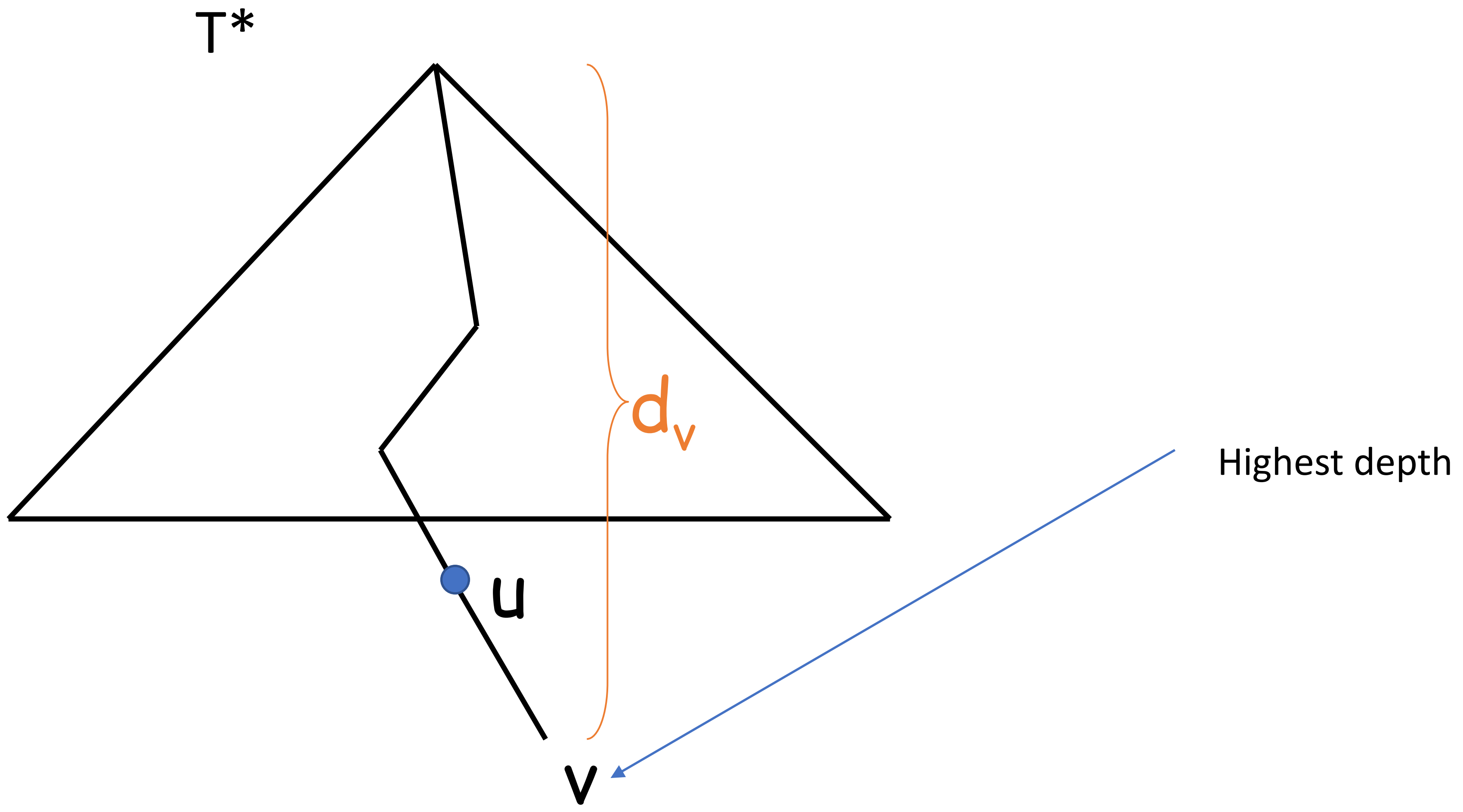
EXCHANGE u AND v to get T’

This will give new optimal prefix tree T’, contradiction to the optimality o f T.

Observation 4. Suppose that u and v are two leaf nodes of optimal prefix tree T*, such that depth(u) < depth(v). Further suppose that in labelling of T* corresponding to an optimal prefix code, leaf node u is labelled with character y of S and leaf node v is labelled with z of S, **then f(y) > f(z).**



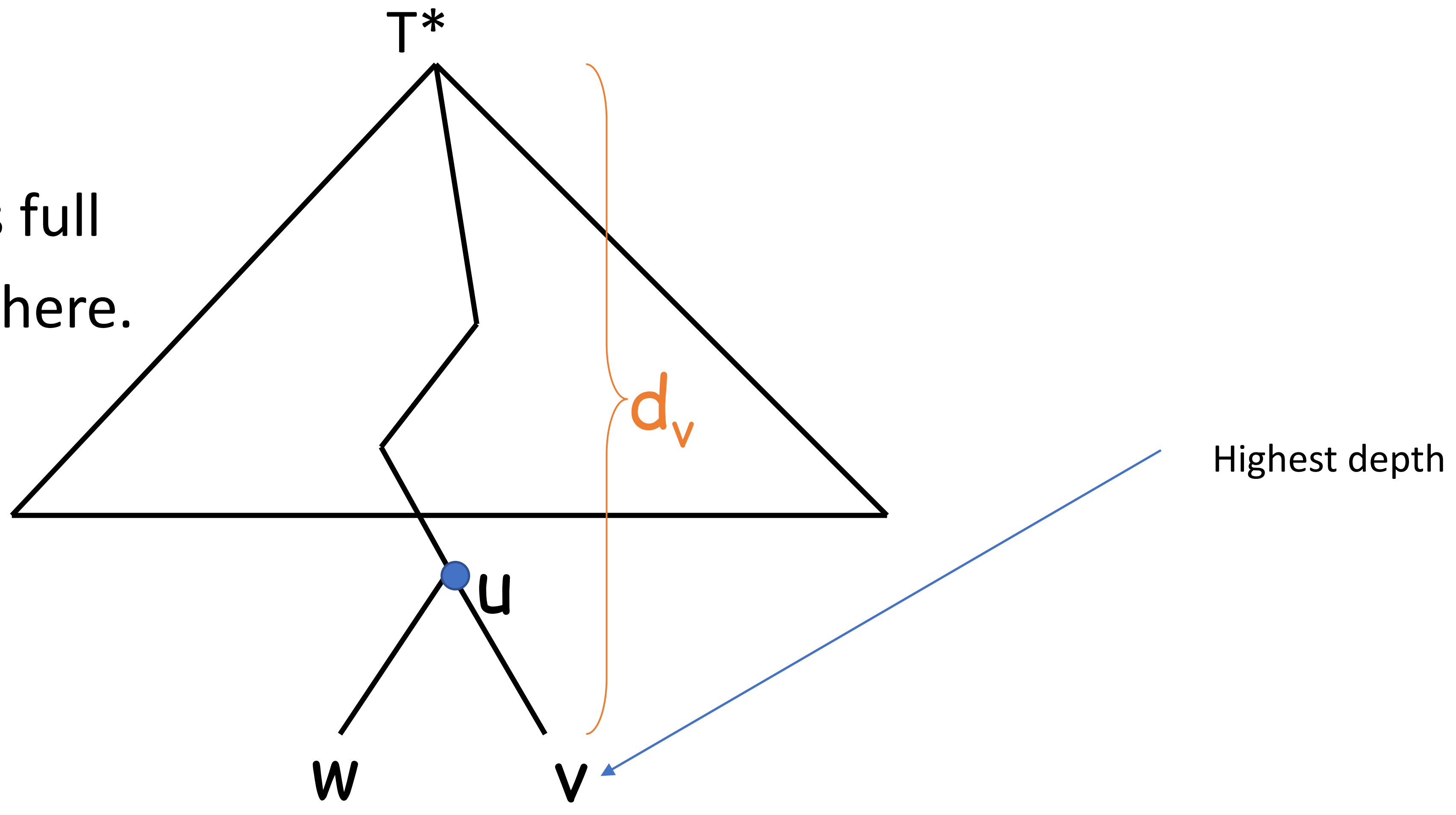
Observation 5. At the highest depth, there will be at least two leaf nodes



Prepared by Pawan K. Mishra

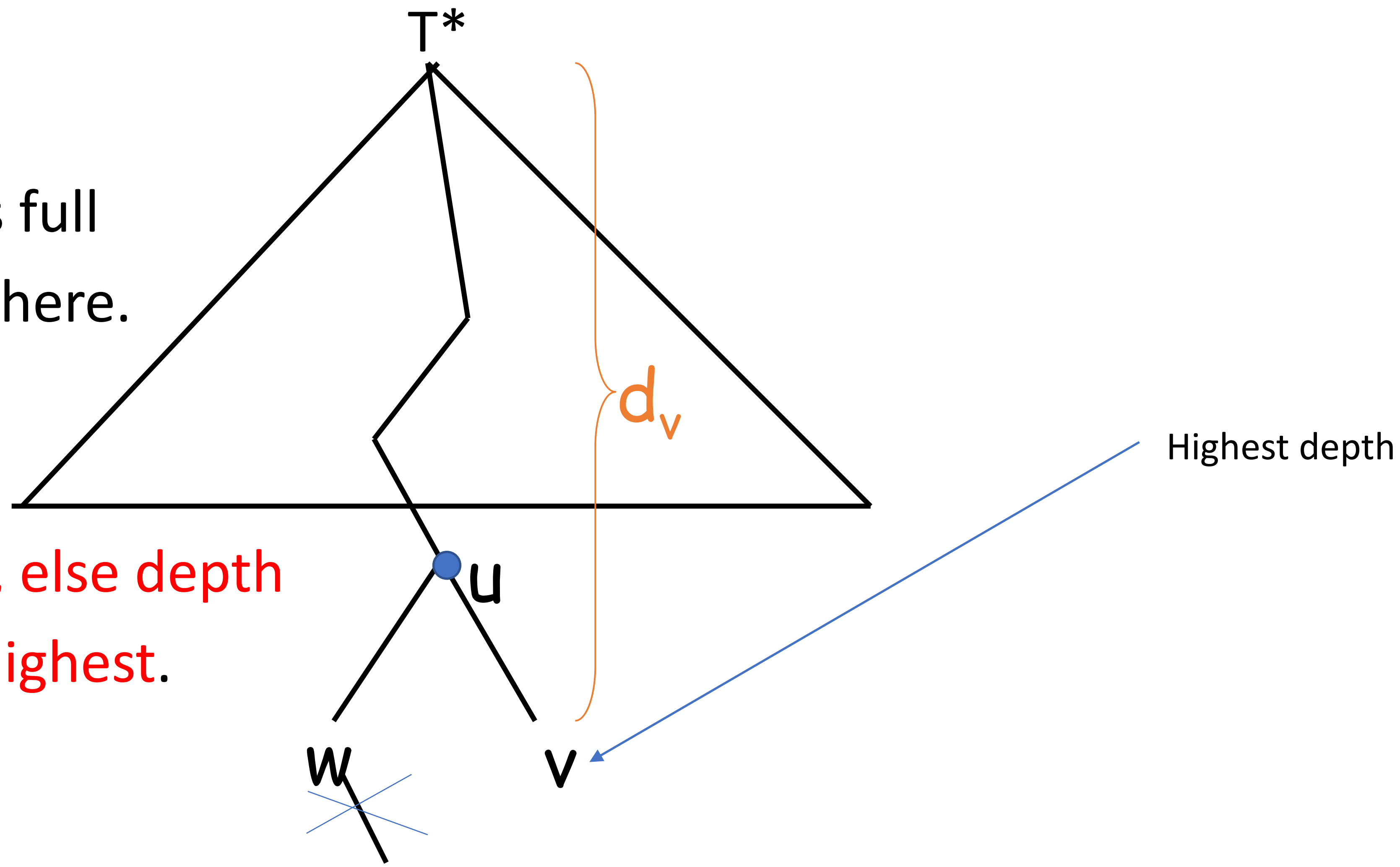
Internal node is full

So, w must be there.



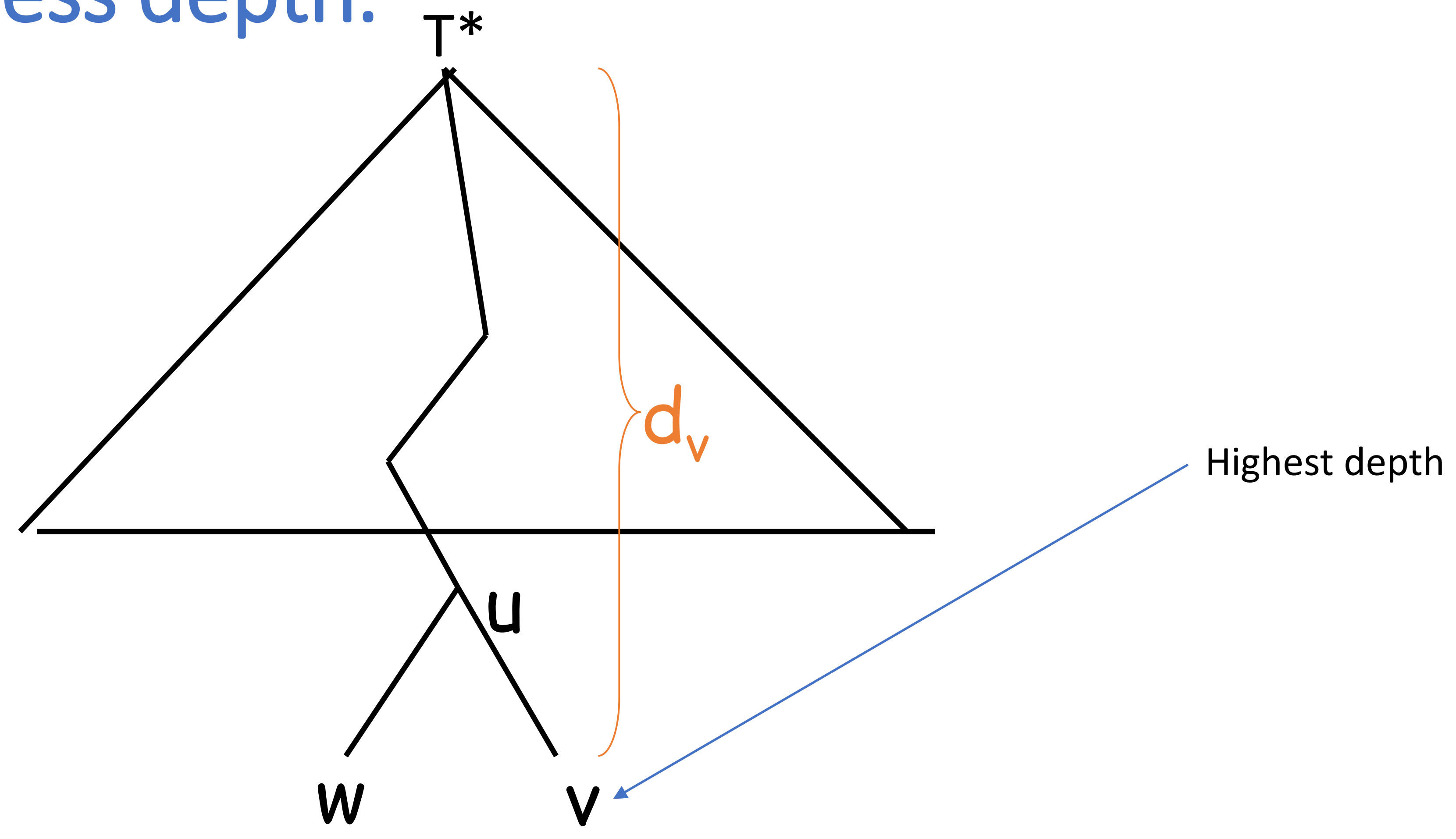
Internal node is full

So, w must be there.

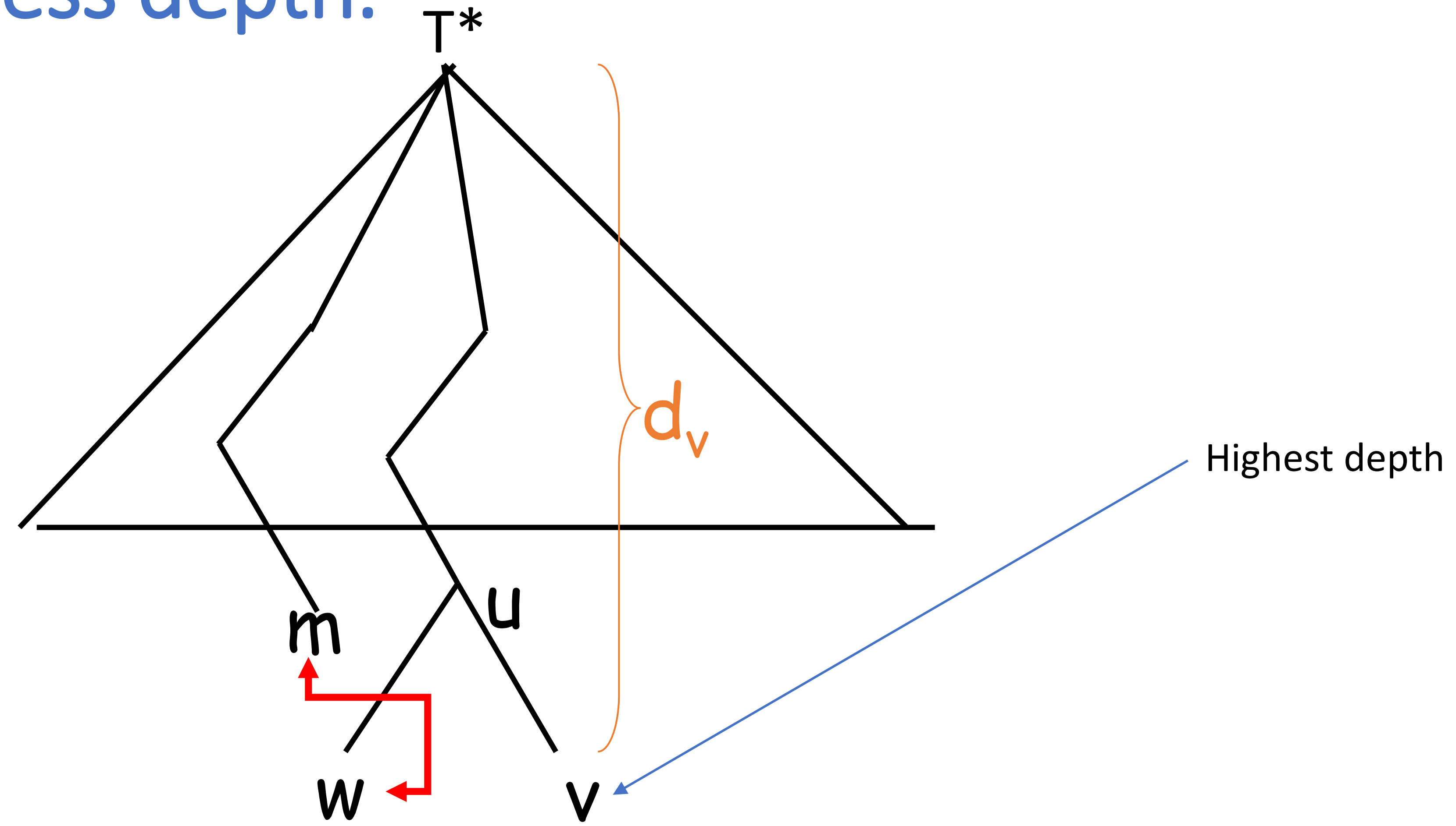


W must be leaf, else depth of v is not the highest.

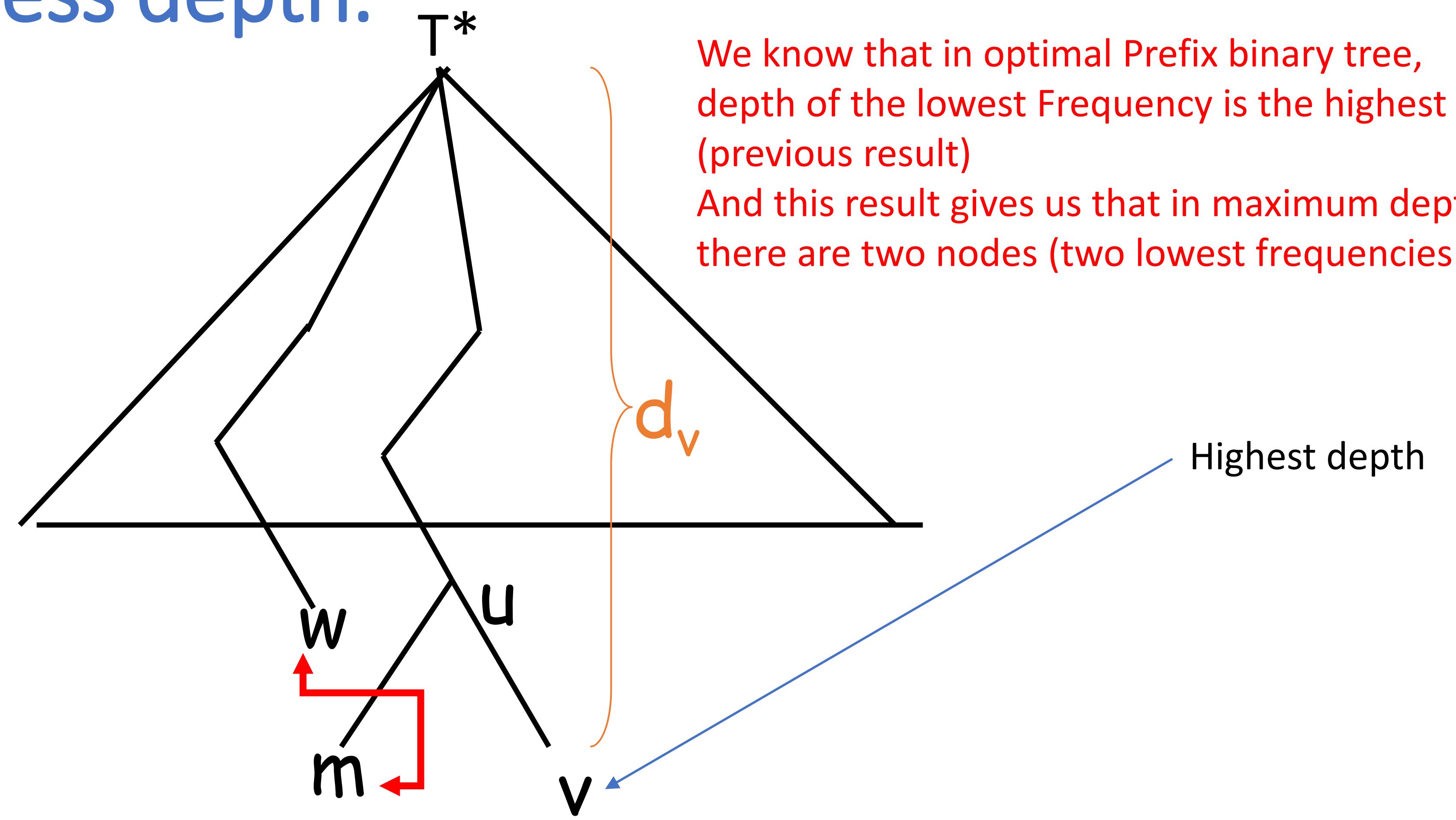
Observation 6. At the highest depth, there will be at least two leaf nodes in optimal prefix tree, and there frequency will be low, compare to any leaf node with less depth.



Observation 6. At the highest depth, there will be at least two leaf nodes in optimal prefix tree, and there frequency will be low, compare to any leaf node with less depth.



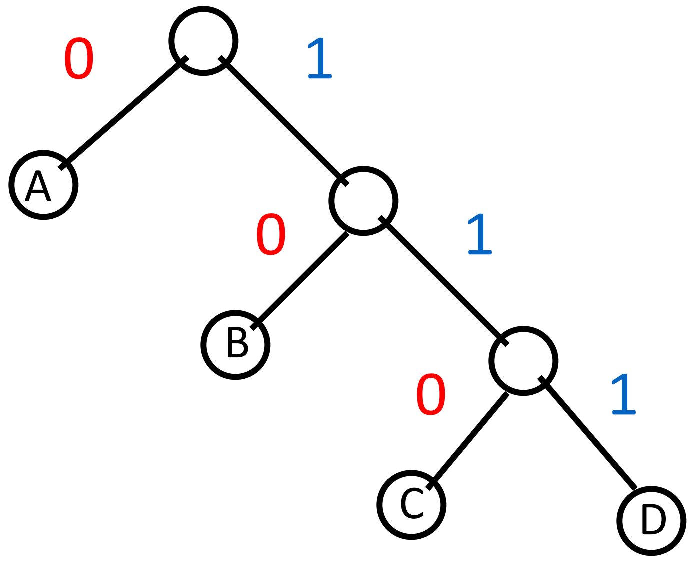
Observation 6. At the highest depth, there will be at least two leaf nodes in optimal prefix tree, and there frequency will be low, compare to any leaf node with less depth.



We know that in optimal Prefix binary tree, depth of the lowest Frequency is the highest (previous result)
And this result gives us that in maximum depth there are two nodes (two lowest frequencies…)

Observe:

1. This is a prefix code, since each of the leaves has a path ending in it, without continuation.
2. If the tree is full then we are not "wasting" bits.
3. If we make sure that the more frequent symbols are closer to the root then they will have a smaller code.



Greedy Algorithm:

1. Consider all pairs: <frequency, symbol>.

2. Choose the two lowest frequencies, and make them siblings, with the root having the combined frequency.

3. Iterate.

Greedy Algorithm Example:

Alphabet: A, B, C, D, E, F

Frequency table:

We can have frequency in values also

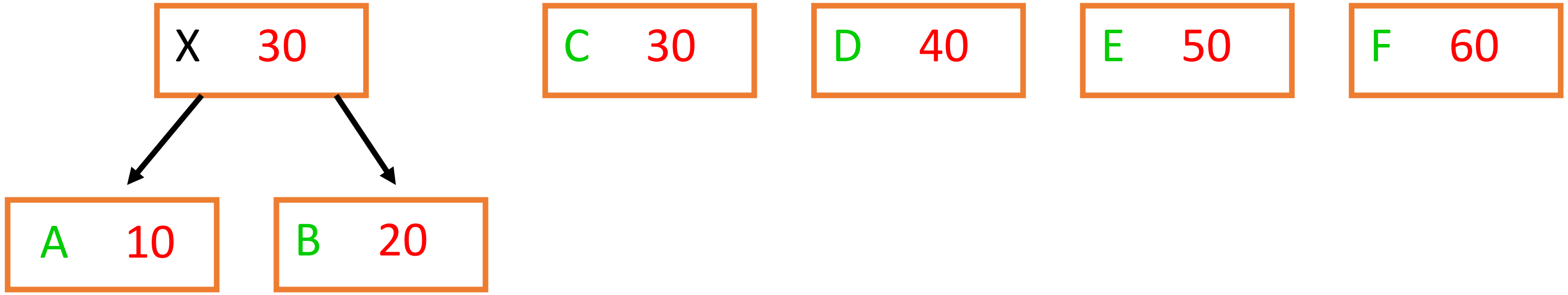
A	B	C	D	E	F
10	20	30	40	50	60

Total File Length: 210

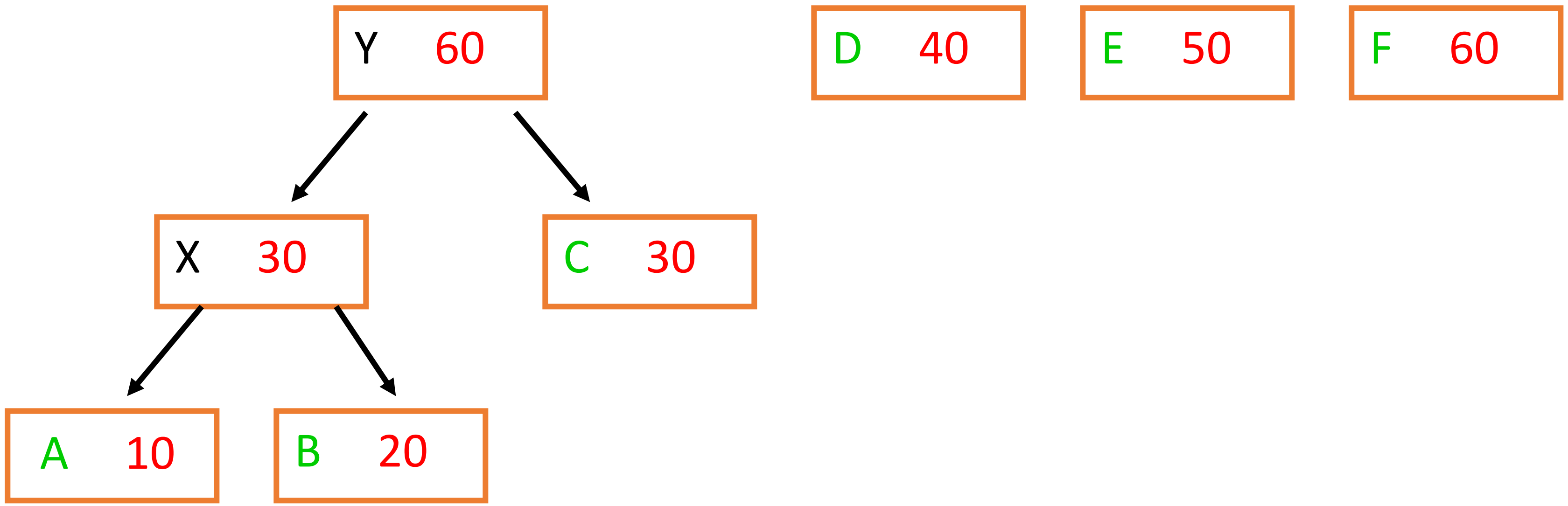
Algorithm Run:

A 10	B 20	C 30	D 40	E 50	F 60

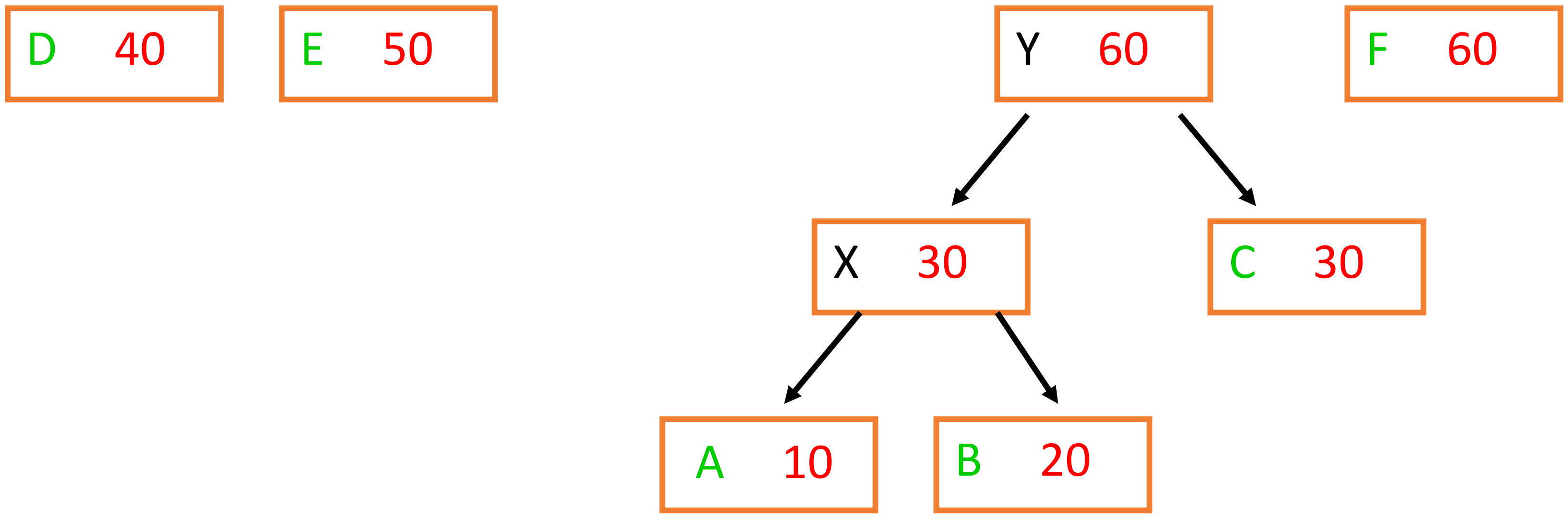
Algorithm Run:



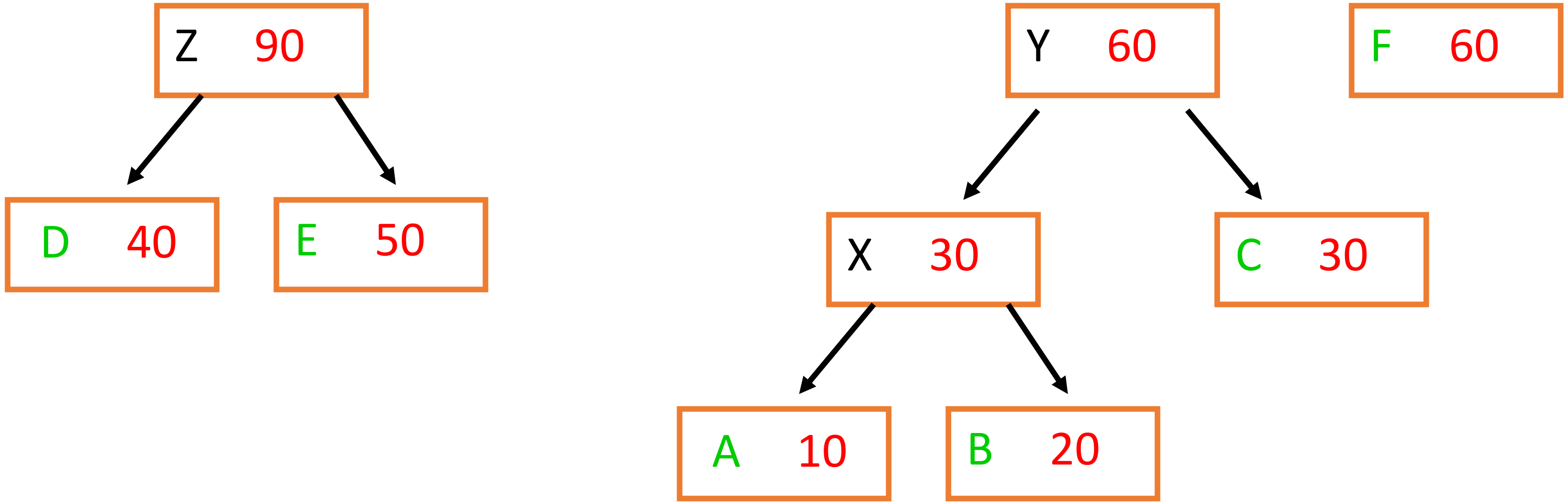
Algorithm Run:



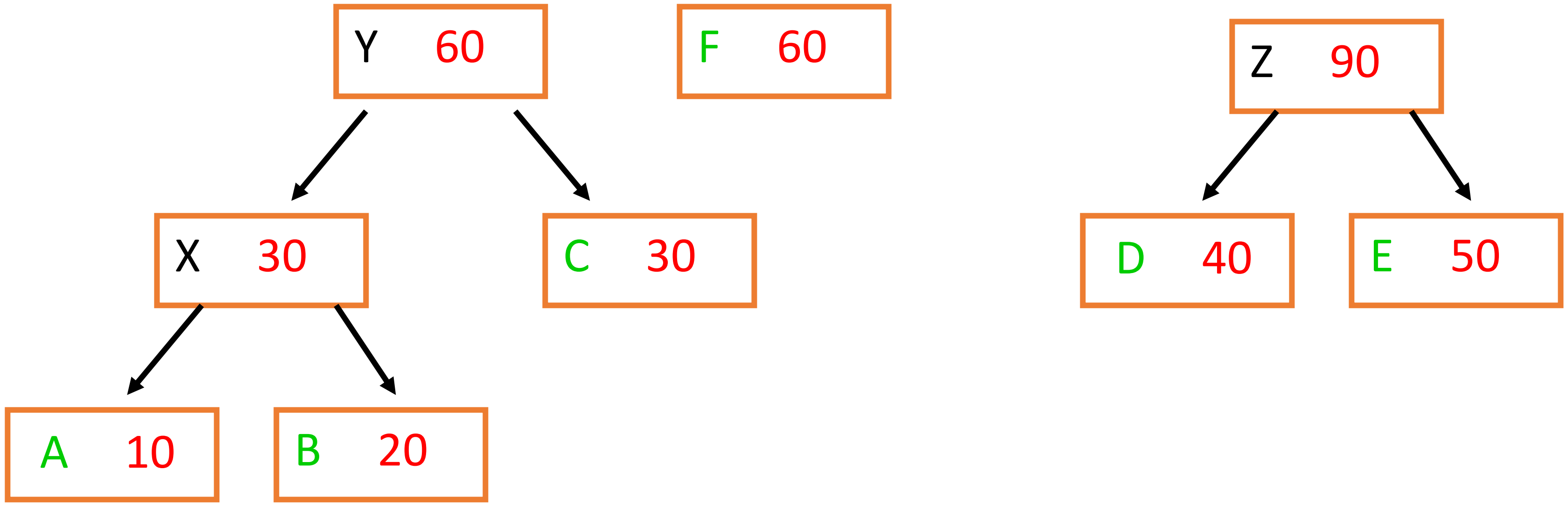
Algorithm Run:



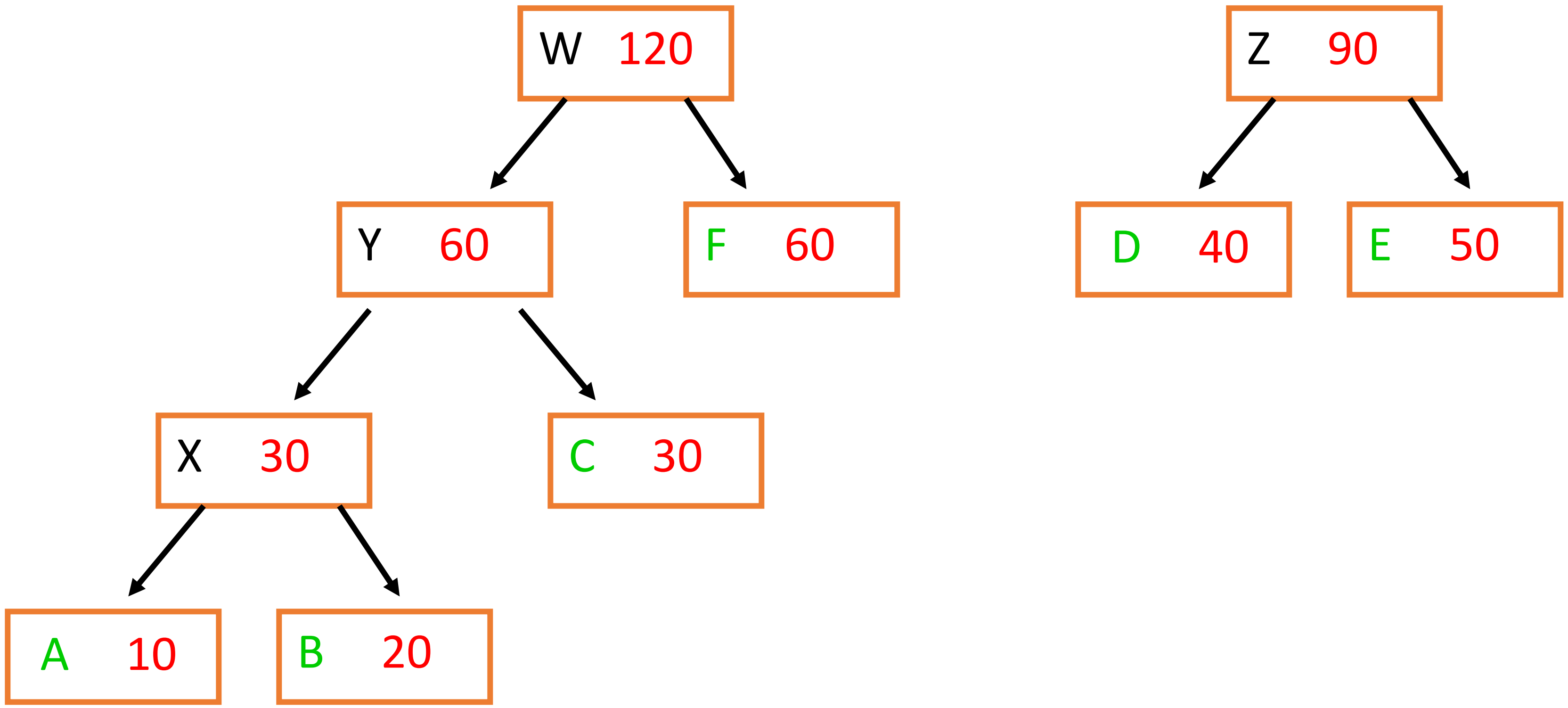
Algorithm Run:



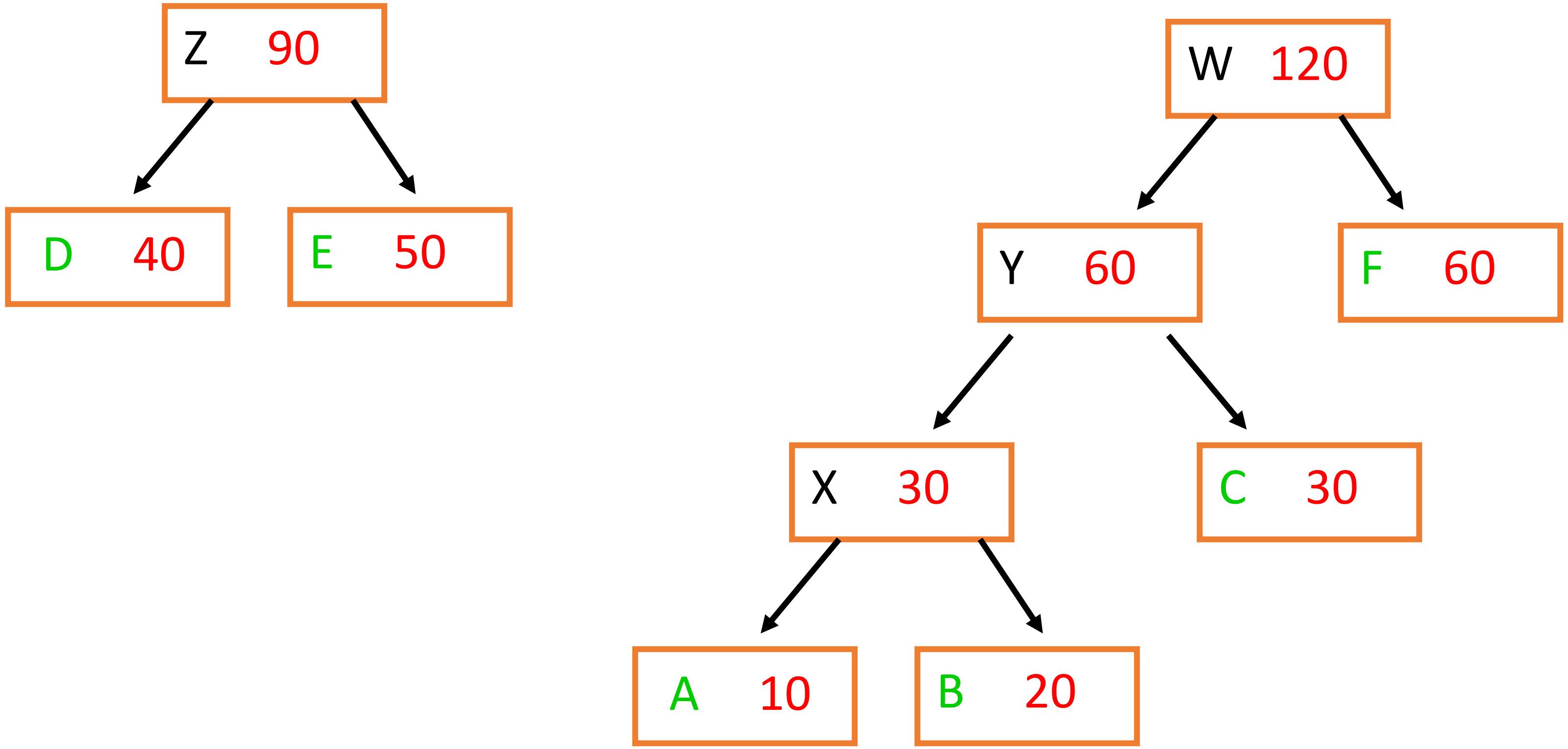
Algorithm Run:



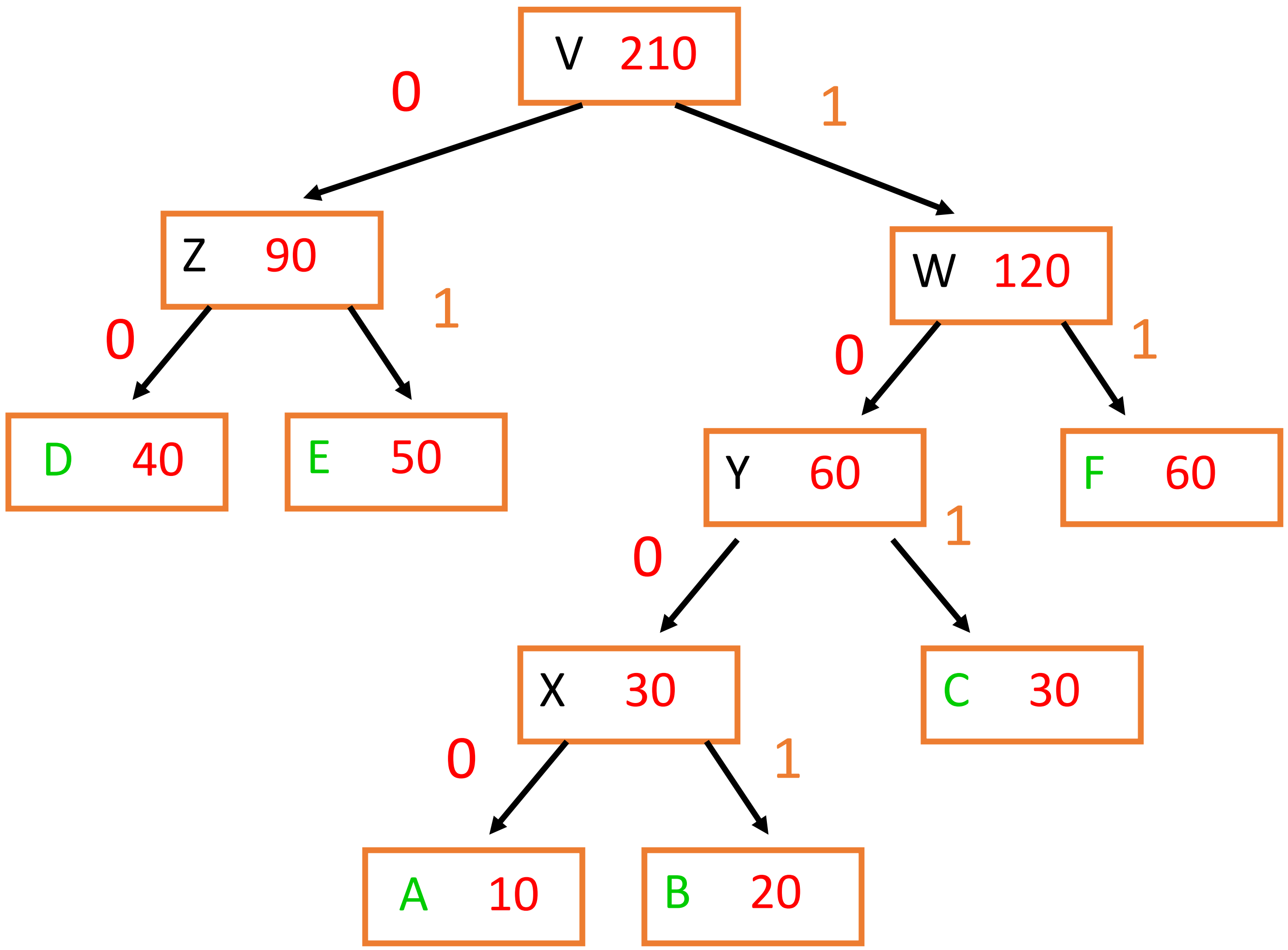
Algorithm Run:



Algorithm Run:

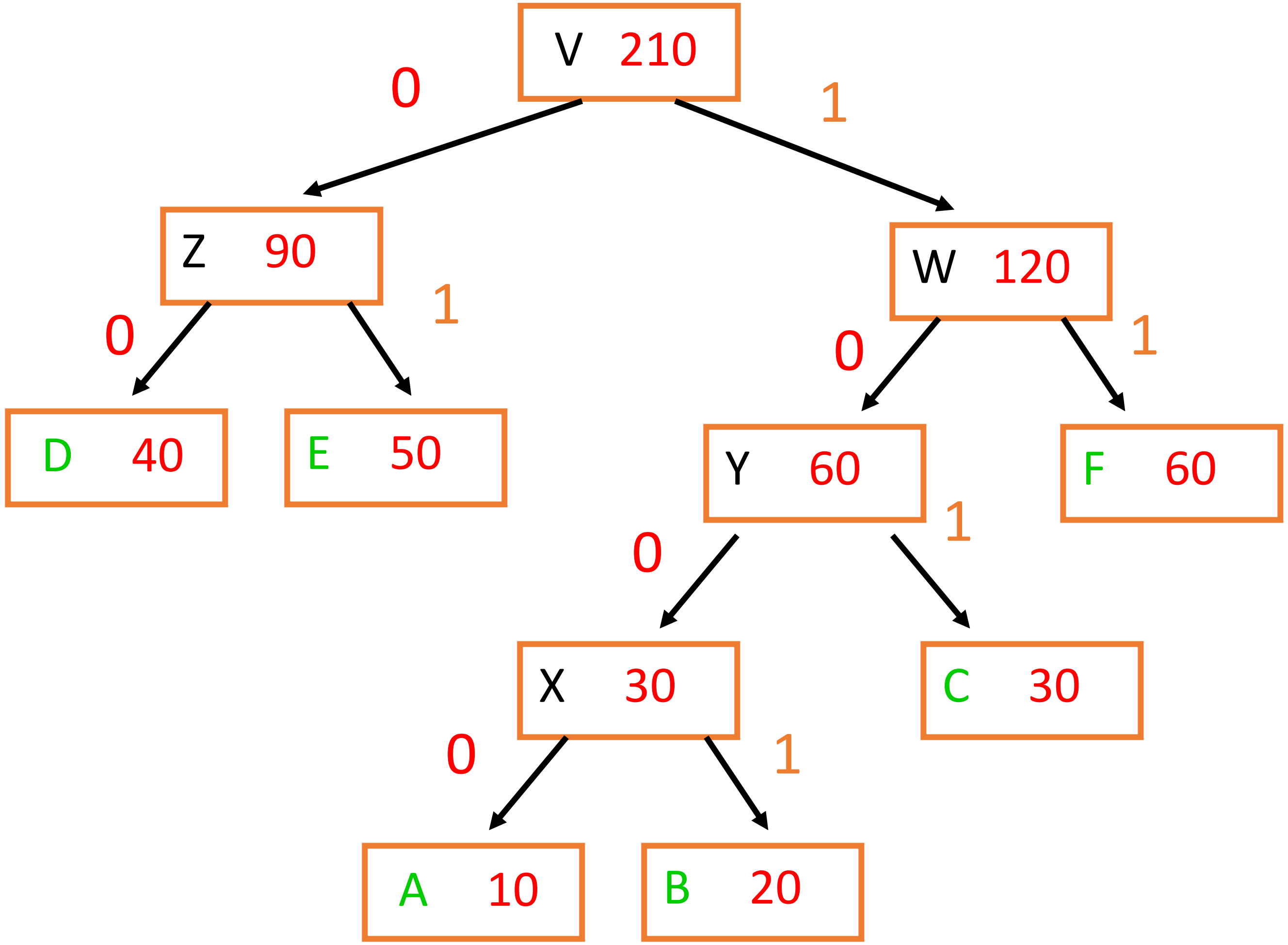


Algorithm Run:



The Huffman encoding:

A: 1000
B: 1001
C: 101
D: 00
E: 01
F: 11



File Size: 10x4 + 20x4 + 30x3 + 40x2 + 50x2 + 60x2 =
40 + 80 + 90 + 80 + 100 + 120 = 510 bits

Note the savings:

The Huffman code:
Required 510 bits for the file.

Fixed length code:
Need 3 bits for 6 characters.
File has 210 characters.

Total: 630 bits for the file.

Note also:

For uniform character distribution:

The Huffman encoding will be equal to the fixed length encoding.

Why?

Assignment.

Formally, the algorithm:

Initialize trees of a single node each.

Keep the roots of all subtrees in a priority queue.

Iterate until only one tree left:
Merge the two smallest frequency subtrees into a single subtree with two children, and insert into priority queue.

Algorithm time:

Each priority queue operation (e.g. heap): $O(\log n)$

In each iteration: one less subtree.

Initially: n subtrees.

Total: $O(n \log n)$ time.

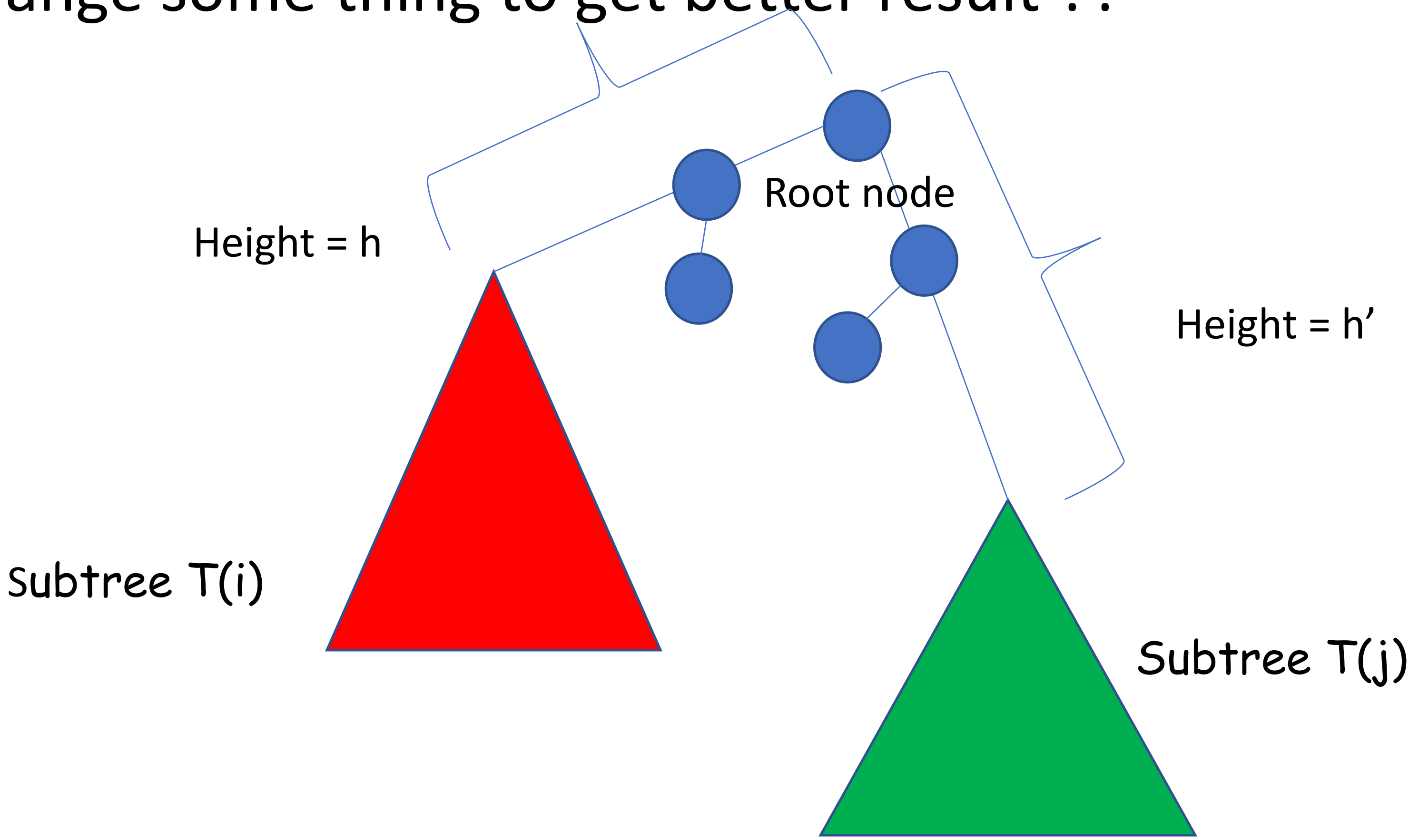
ANOTHER WAY TO PROCEED

HOW TO PROCEED

Assume we have some binary tree with n nodes as leaves, can we exchange subtrees to get better result ??

HOW TO PROCEED

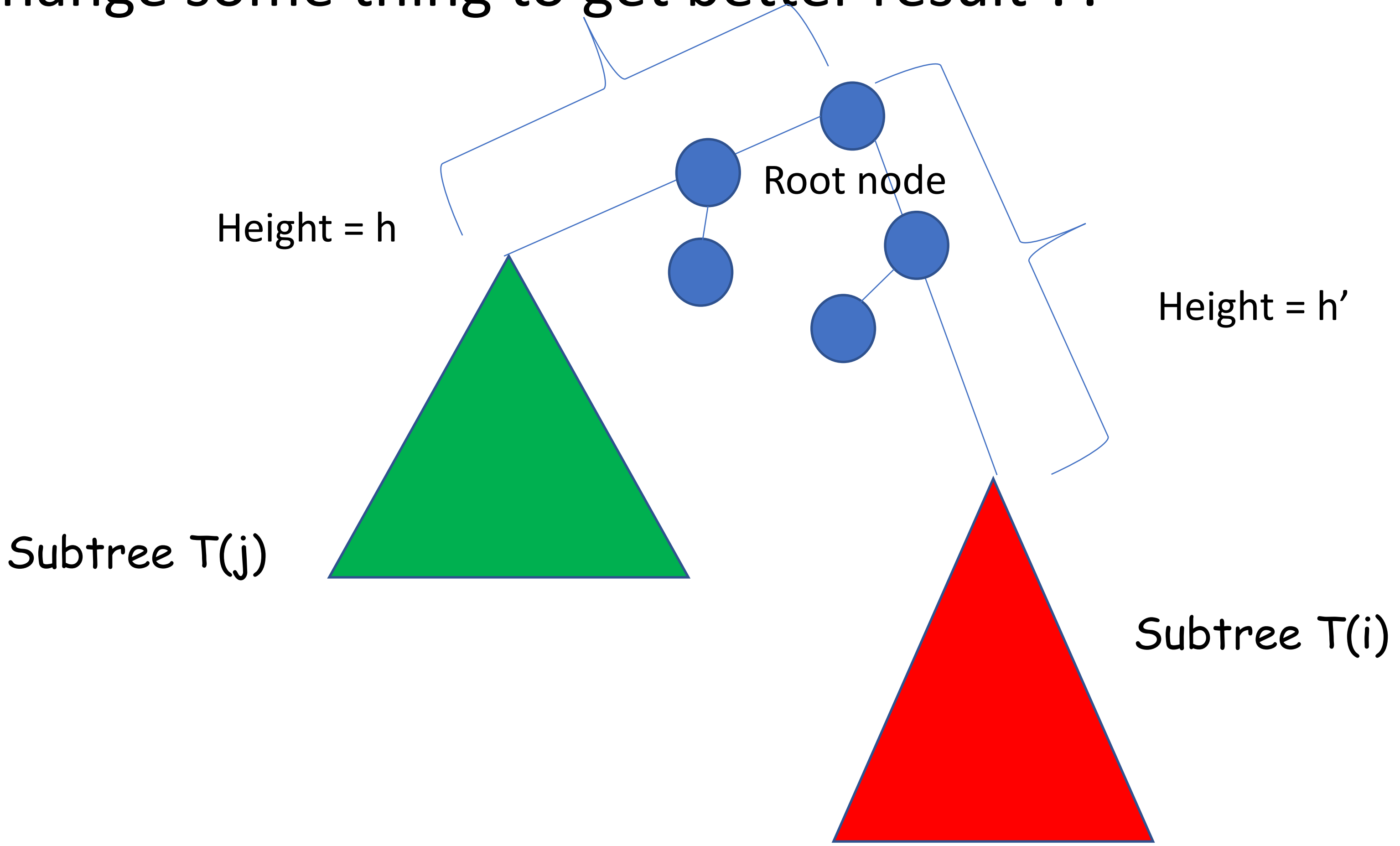
Assume we have some binary tree with n nodes as leaves, can we exchange some thing to get better result ??



Prepared by Pawan K. Mishra

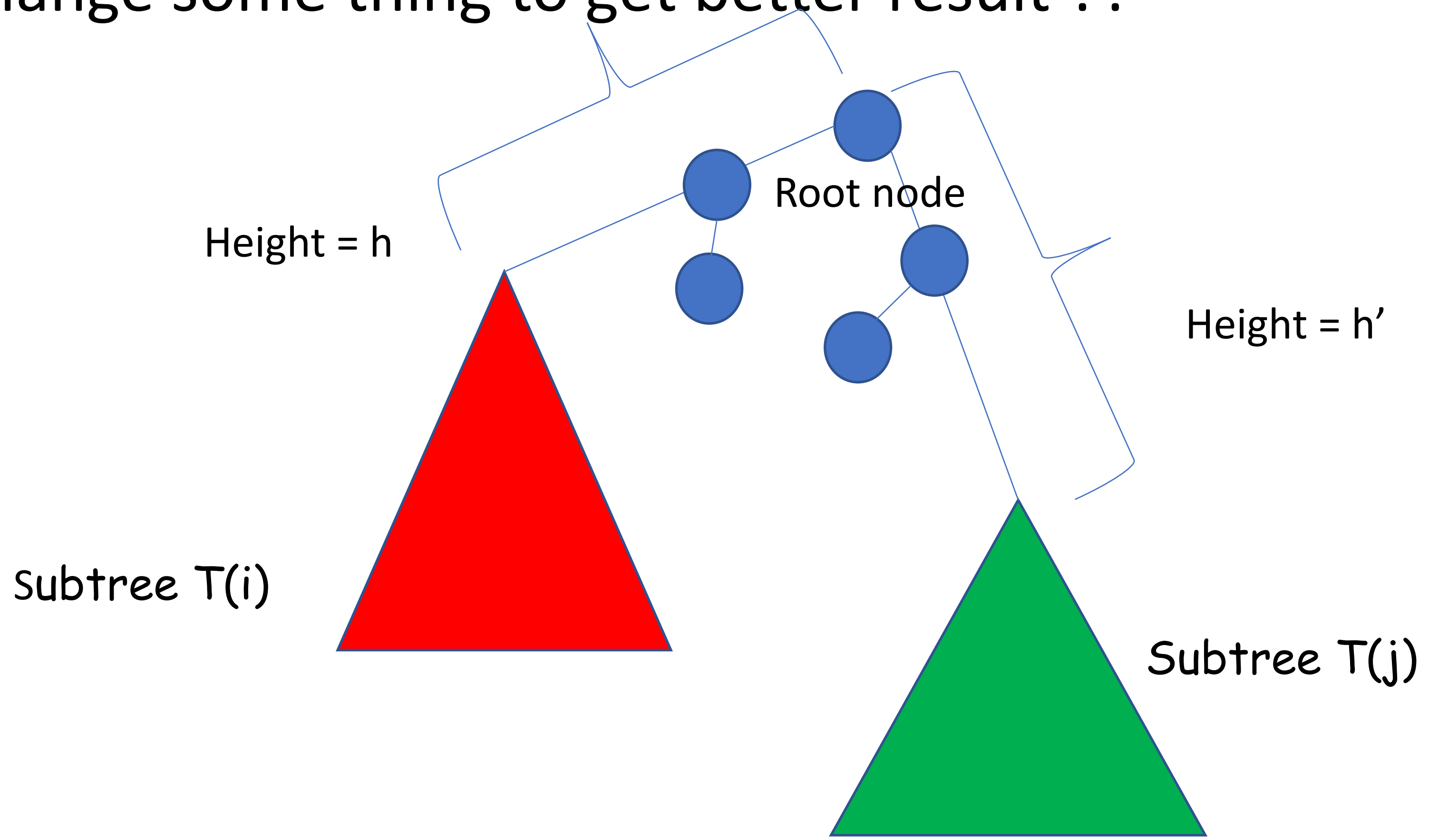
HOW TO PROCEED

Assume we have some binary tree with n nodes as leaves, can we exchange some thing to get better result ??



HOW TO PROCEED

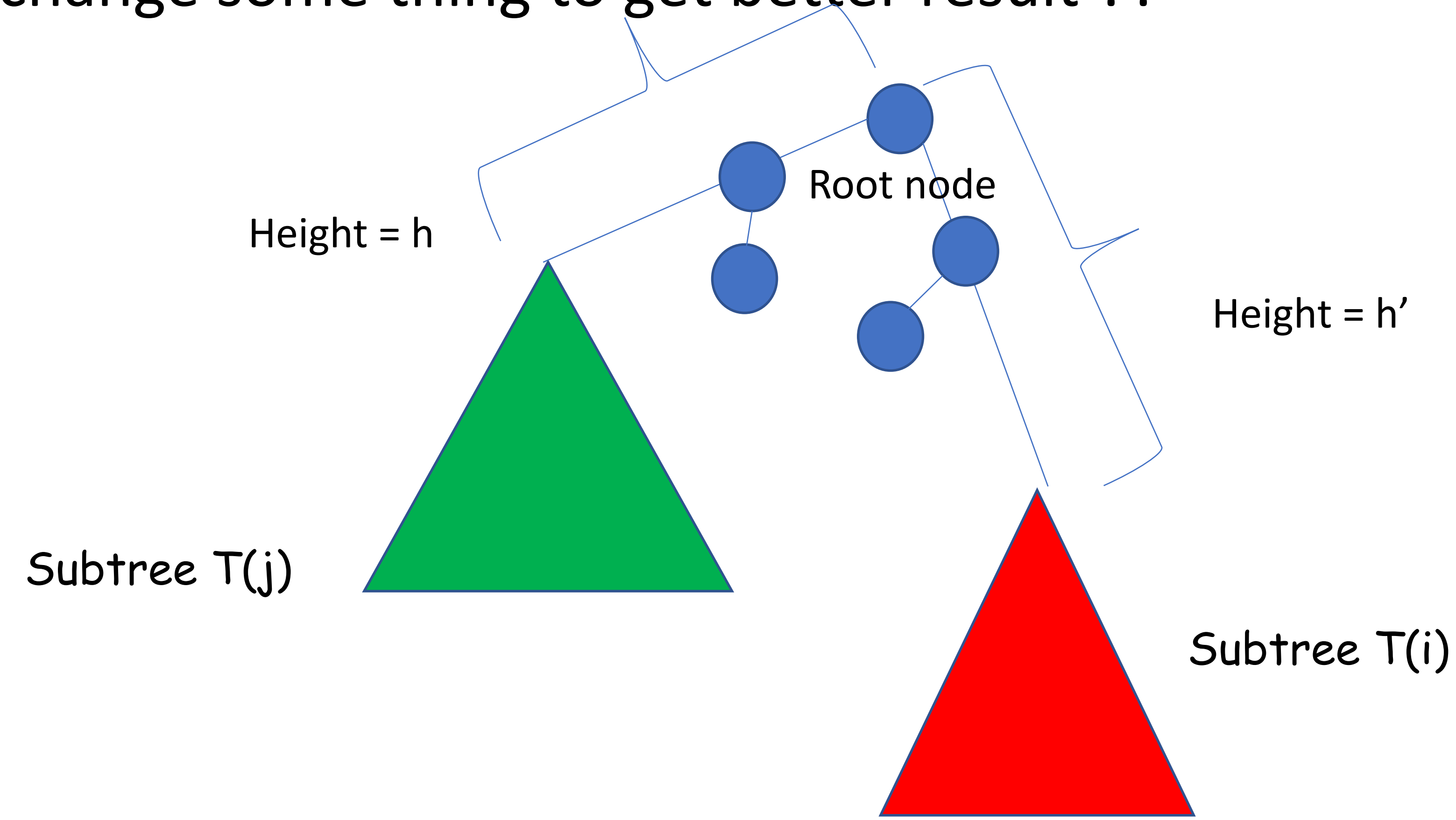
Assume we have some binary tree with n nodes as leaves, can we exchange some thing to get better result ??



Old result= extra + $\sum_{e \in T(i)} f(e)\,(h+le) + \sum_{c \in T(j)} f(c)\,(h'+l'_c)$

HOW TO PROCEED

Assume we have some binary tree with n nodes as leaves, can we exchange some thing to get better result ??



New result= extra + $\sum_{c \in T(j)} f(c)\,(h+l'_c) + \sum_{e \in T(i)} f(e)\,(h'+le)$

- Old result= extra + $\sum_{e \in T(i)} f(e)\,(h+le) + \sum_{c \in T(j)} f(c)\,(h'+l'_c)$

New result= extra + $\sum_{c \in T(j)} f(c)\,(h+l'_c) + \sum_{e \in T(i)} f(e)\,(h'+le)$

Old result – new result = $(h'-h)\left(\sum_{c \in T(j)} f(c) - \sum_{e \in T(i)} f(e)\right)$

- Old result= extra + $\sum_{e \in T(i)} f(e)\,(h+le) + \sum_{c \in T(j)} f(c)\,(h'+l'_c)$

New result= extra + $\sum_{c \in T(j)} f(c)\,(h+l'_c) + \sum_{e \in T(i)} f(e)\,(h'+le)$

Old result – new result = $(h'-h)\left(\sum_{c \in T(j)} f(c) - \sum_{e \in T(i)} f(e)\right)$

We know this h'-h > 0

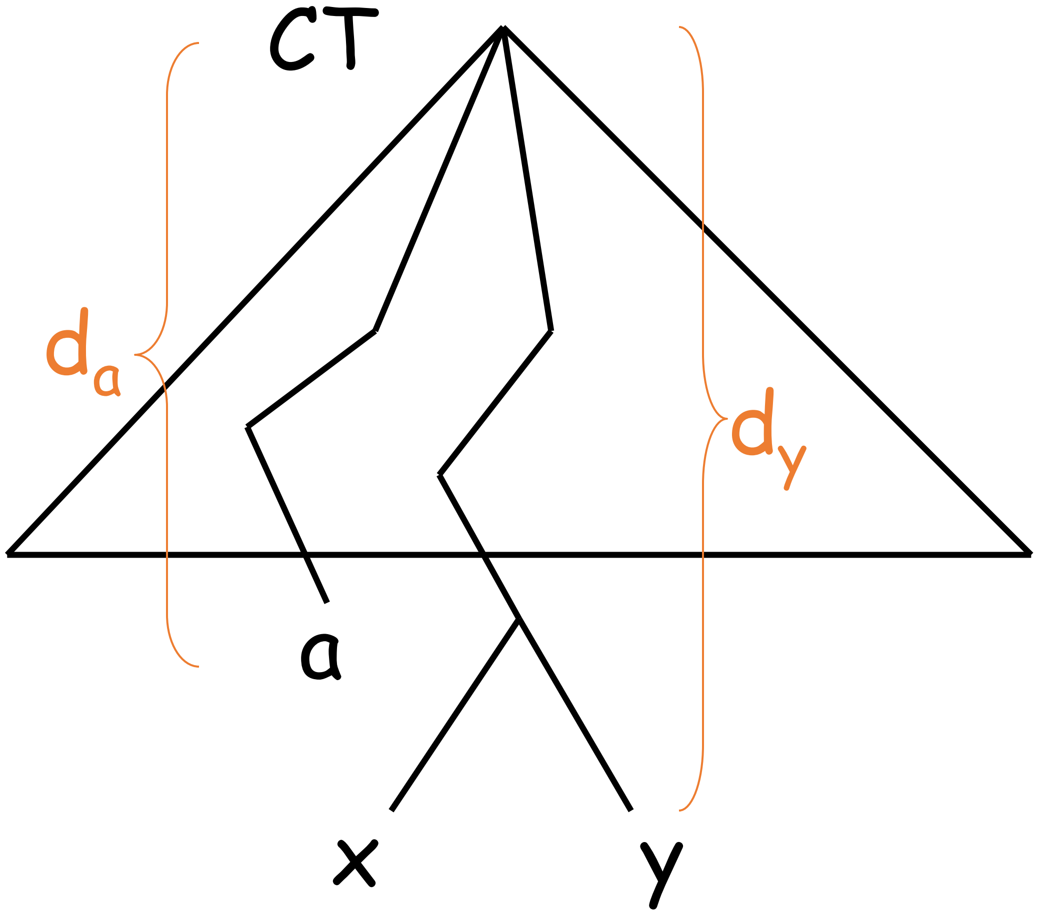
- Old result= extra + $\sum_{e \in T(i)} f(e)\,(h+le) + \sum_{c \in T(j)} f(c)\,(h'+l'_c)$

New result= extra + $\sum_{c \in T(j)} f(c)\,(h+l'_c) + \sum_{e \in T(i)} f(e)\,(h'+le)$

Old result – new result = $\underbrace{(h'-h)\left(\sum_{c \in T(j)} f(c) - \sum_{e \in T(i)} f(e)\right)}_{\text{positive, then only we can say exchange worked}}$

We know this h'-h > 0

Algorithm correctness:

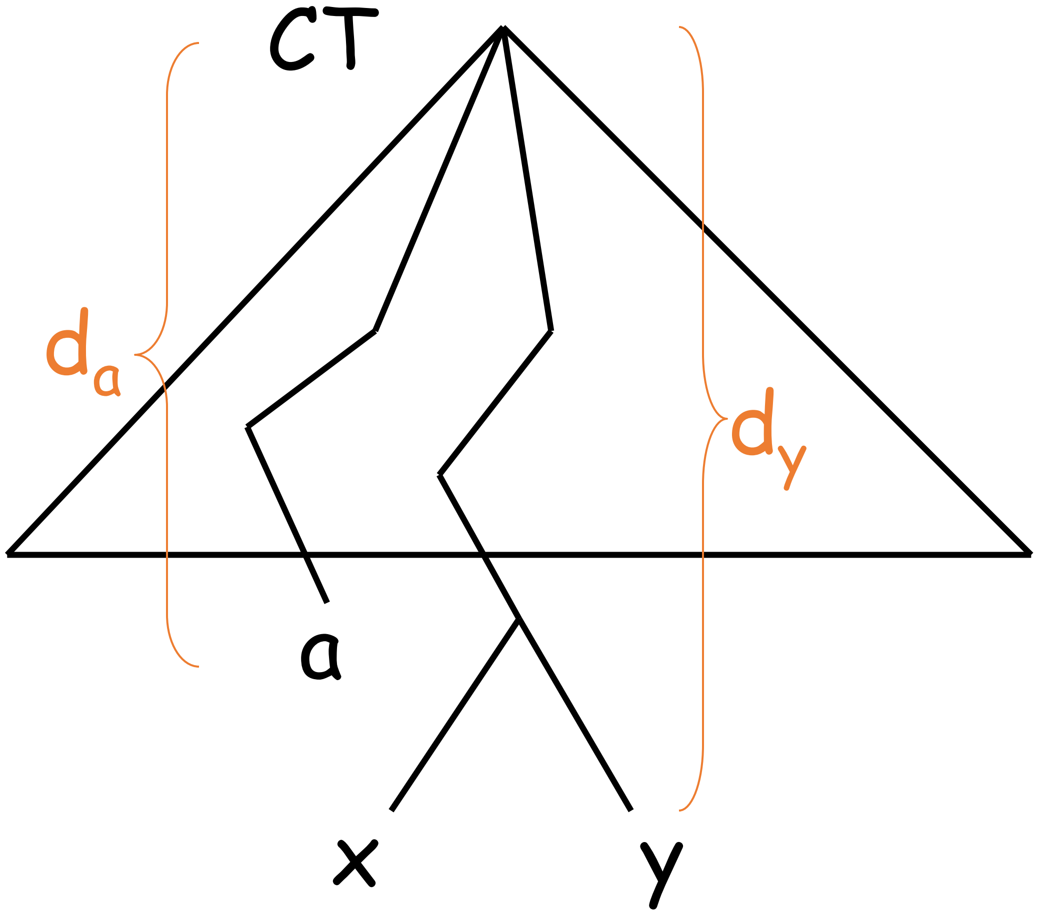


We know about depth and frequency:

$d_a \leq d_y$

$f_a \leq f_y$

Algorithm correctness:



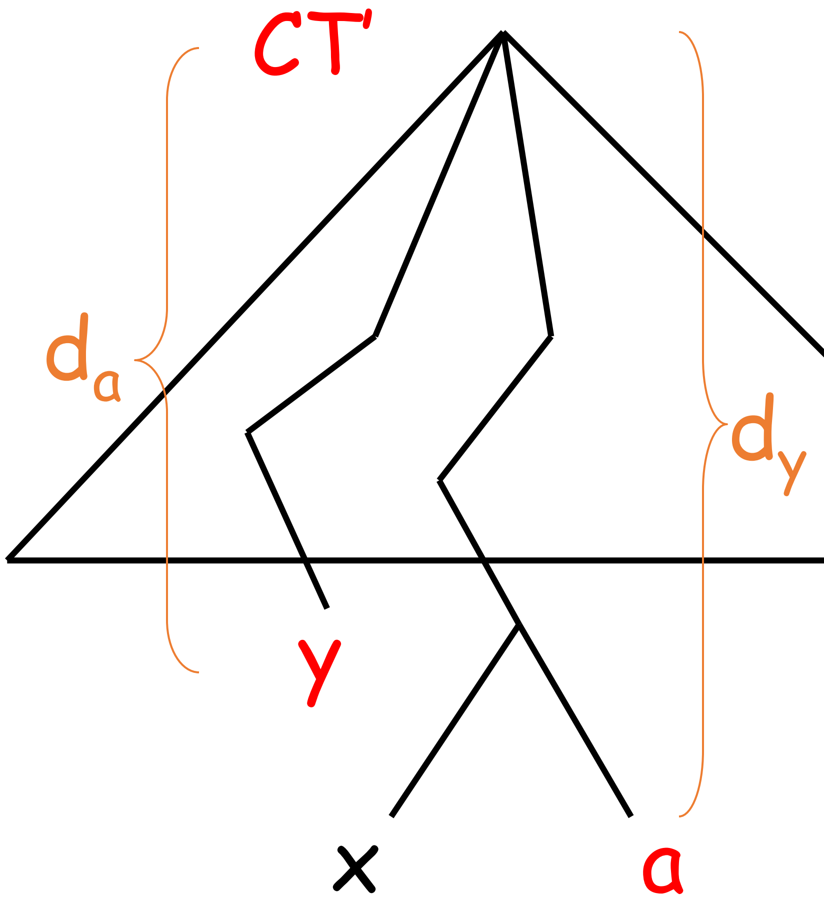
We also know about code tree CT:

$\sum_{\sigma} f_\sigma d_\sigma$

is smallest possible.

Now exchange a and y.

Algorithm correctness:



$$Cost(CT) = \sum_{\sigma} f_\sigma d_\sigma =$$

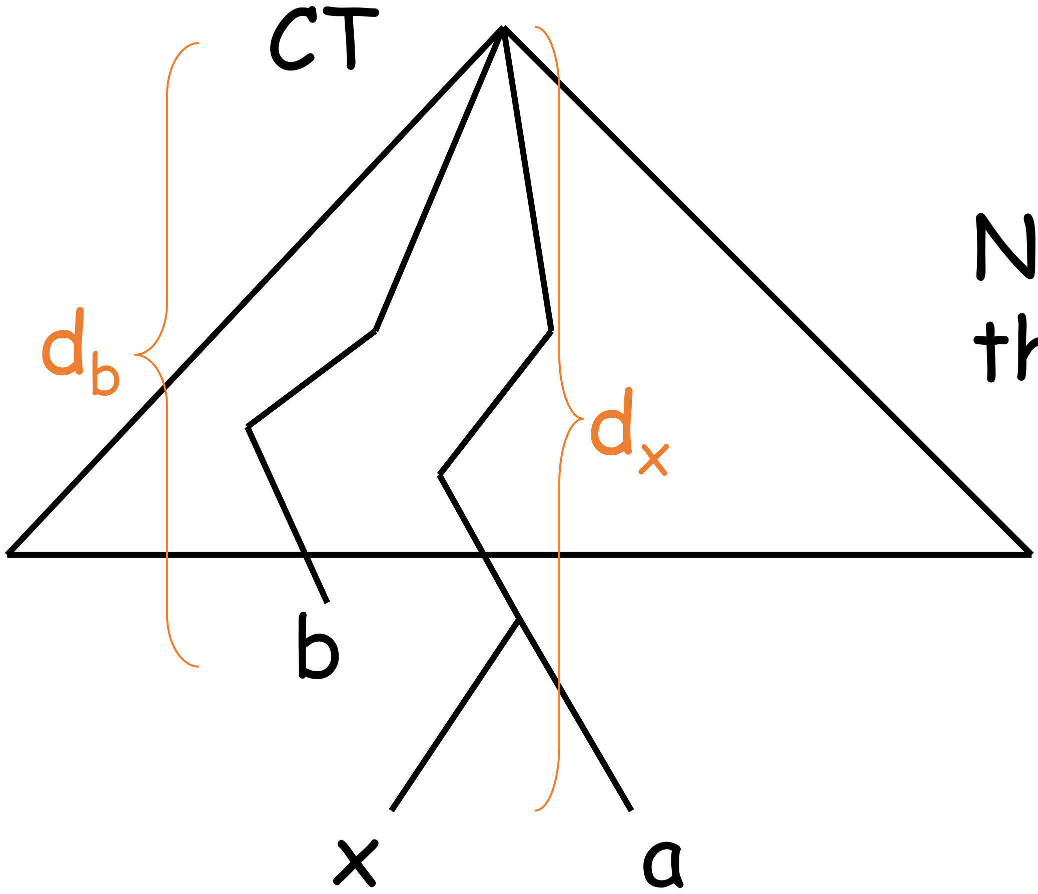
$$\sum_{\sigma \neq a,y} f_\sigma d_\sigma + f_a d_a + f_y d_y \geq$$

($d_a \leq d_y$, $f_a \leq f_y$
Therefore
$f_a d_a \geq f_y d_a$ and
$f_y d_y \geq f_a d_y$)

$$\sum_{\sigma \neq a,y} f_\sigma d_\sigma + f_y d_a + f_a d_y =$$

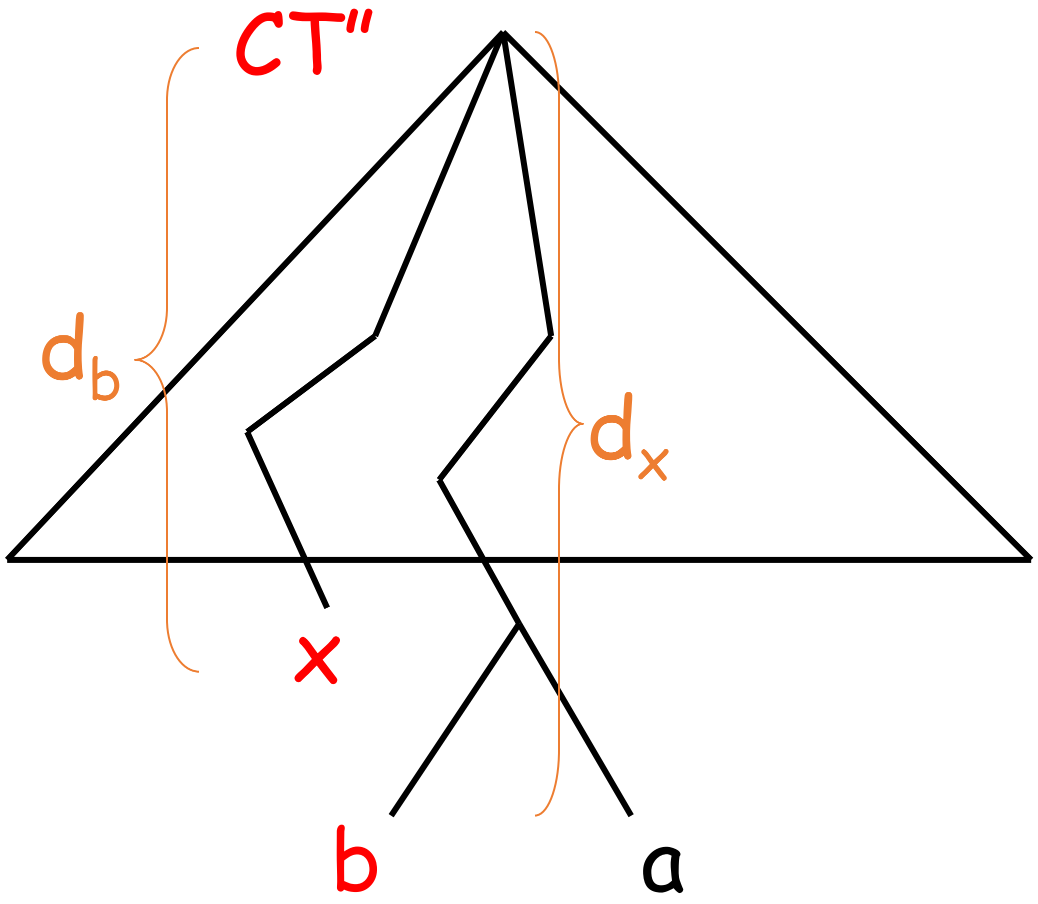
$$cost(CT')$$

Algorithm correctness:



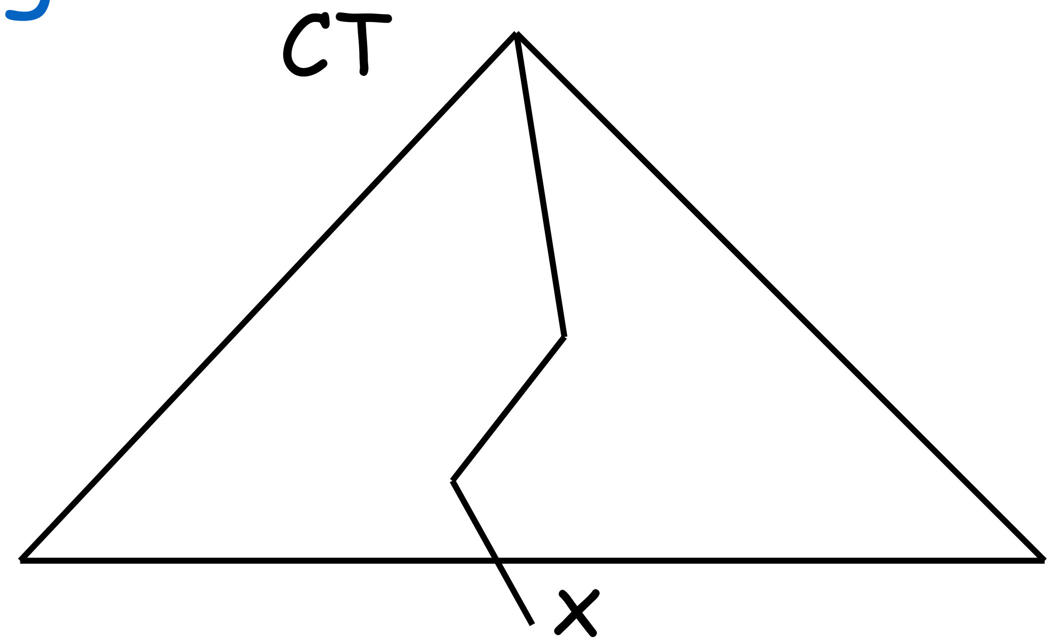
Now do the same thing for b and x

Algorithm correctness:

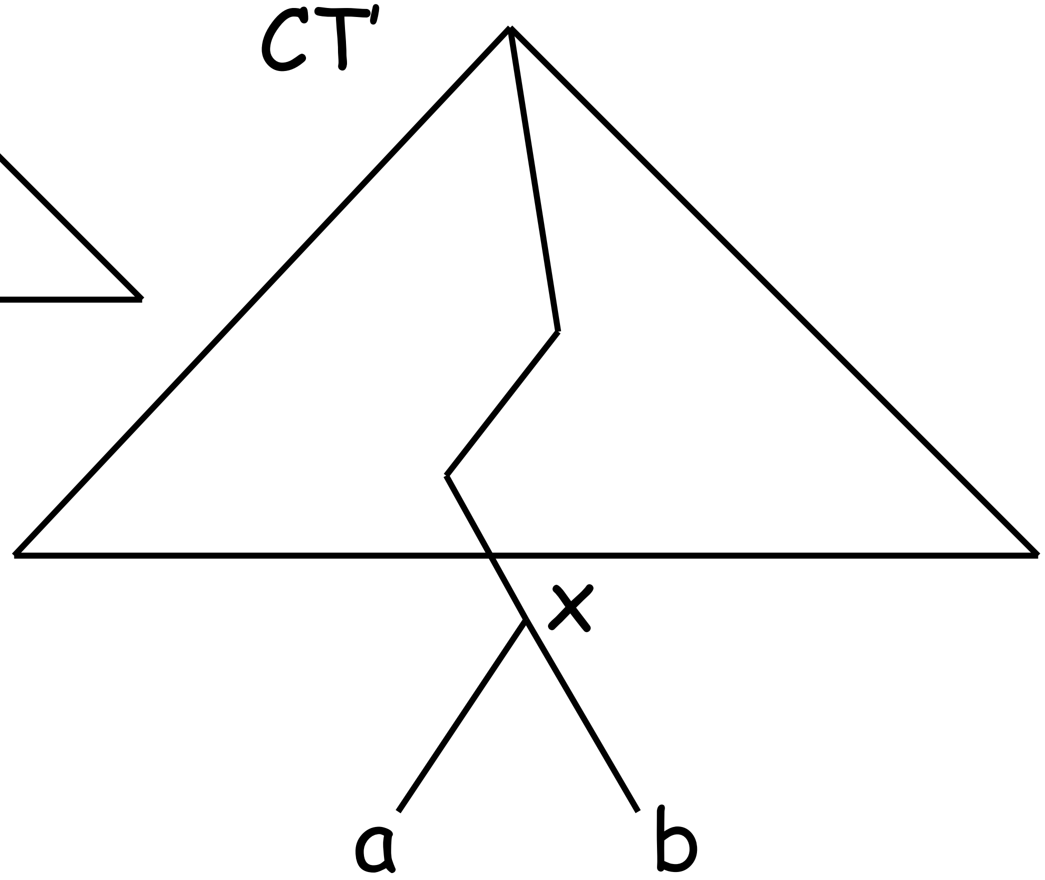


And get an optimal code tree where a and b are sibling with the longest paths

Algorithm correctness:



$$f_x = f_a + f_b$$



Algorithm correctness:

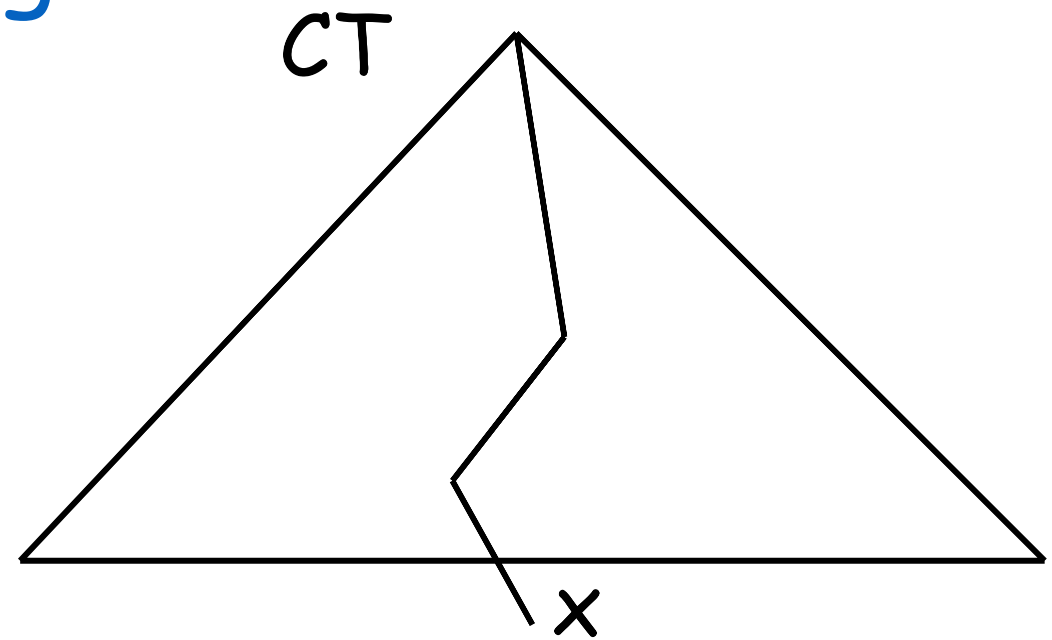
$$cost(CT') = \sum_{\sigma} f_\sigma d'_\sigma = \sum_{\sigma \neq a,b} f_\sigma d'_\sigma + f_a d'_a + f_b d'_b =$$

$$\sum_{\sigma \neq a,b} f_\sigma d'_\sigma + f_a(d_x+1) + f_b\,(d_x+1) =$$

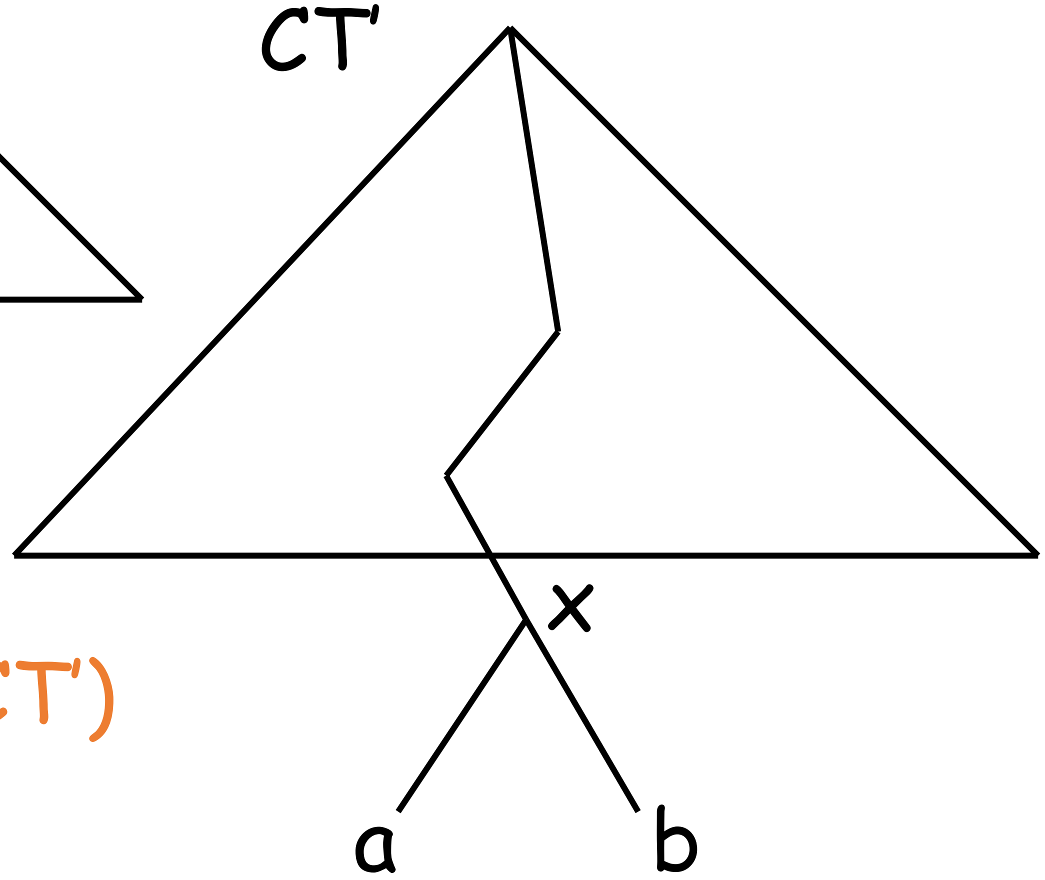
$$\sum_{\sigma \neq a,b} f_\sigma d'_\sigma + (f_a + f_b)(d_x+1) =$$

$$\sum_{\sigma \neq a,b} f_\sigma d_\sigma + f_x(d_x+1) + f_x = cost(CT) + f_x$$

Algorithm correctness:



$f_x = f_a + f_b$



$\text{cost}(CT) + f_x = \text{cost}(CT')$

Algorithm correctness:

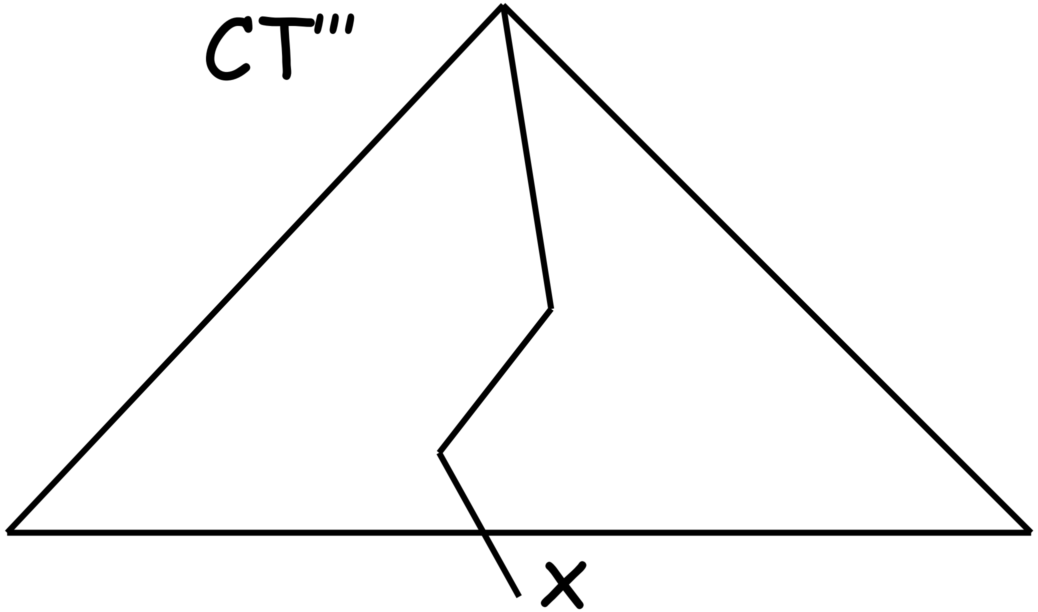
Assume CT' is not optimal.

By the previous lemma there is a tree CT'' that is optimal, and where a and b are siblings. So

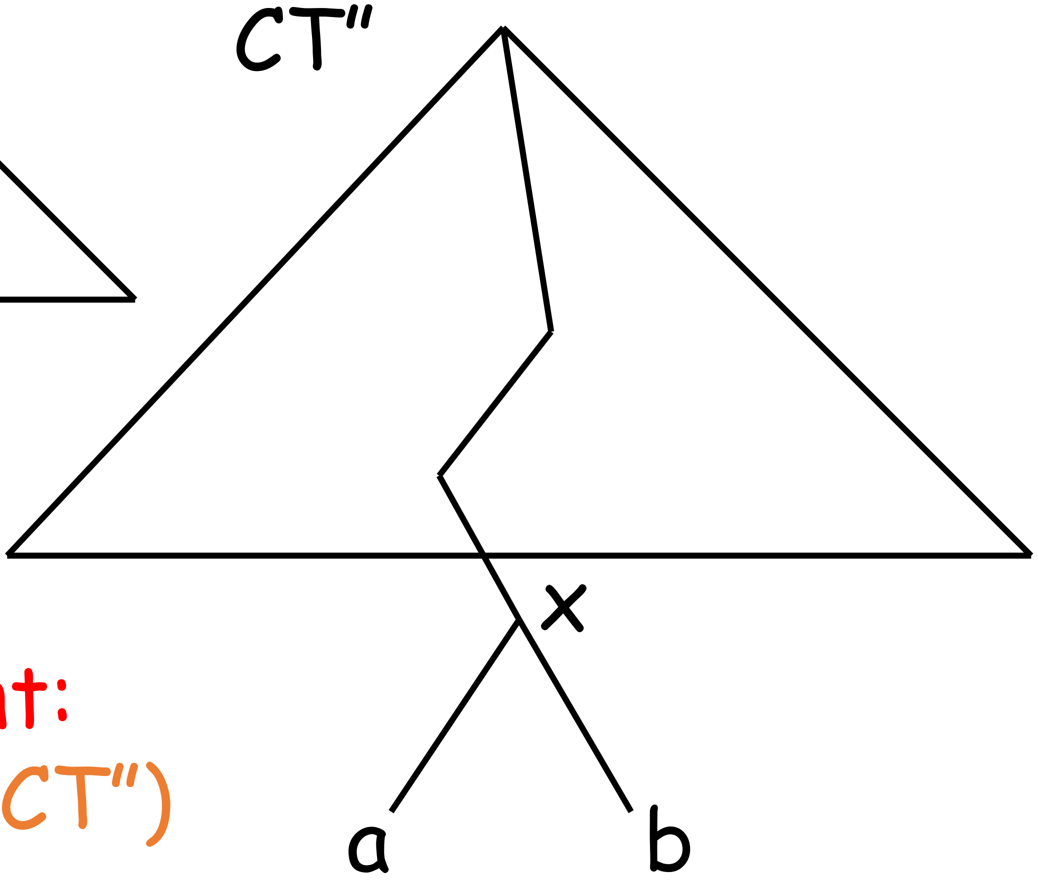
cost(CT'') < cost(CT')

Algorithm correctness:

Consider



$f_x = f_a + f_b$



By a similar argument:

$cost(CT''') + f_x = cost(CT'')$

Algorithm correctness:

We get:

$$\mathrm{cost}(CT''') = \mathrm{cost}(CT'') - f_x < \mathrm{cost}(CT') - f_x$$
$$= \mathrm{cost}(CT)$$

and this contradicts the minimality of $\mathrm{cost}(CT)$.