L10 5800

feb 18/21 2022

shelat

Greedy is only good for certain problems

	start	end
sy3333	2	3.25
en1612	1	4
ma1231	3	4
Cs5800	3.5	4.75
cs4800	4	5.25
cs6051	4.5	6
sy3100	5	6.5
Cs1234	7	8

How many non-overlapping courses can you take?

problem statement

$$(a_1, \ldots, a_n)$$

$$(s_1, s_2, \ldots, s_n)$$ starting times.

$$(f_1, f_2, \ldots, f_n)$$ (sorted) $s_i < f_i$

end times

find largest subset of activities $C=\{a_i\}$ such that

(compatible)

for any i, j $i \neq j$

$s_j > f_i$

problem statement

$$(a_1, \ldots, a_n)$$

$$(s_1, s_2, \ldots, s_n)$$

$(f_1, f_2, \ldots, f_n)$ (sorted) $s_i < f_i$

find largest subset of activities C={a_i} such that

(compatible)

For any two activities $a_i, a_j, i < j$ the start time of a_j is after the finish time of a_i.

problem statement

$$(a_1, \ldots, a_n)$$

$$(s_1, s_2, \ldots, s_n)$$

$$(f_1, f_2, \ldots, f_n)$$ (sorted) $$s_i < f_i$$

find largest subset of activities C={a_i} such that

(compatible)

$$a_i, a_j \in C, i < j$$

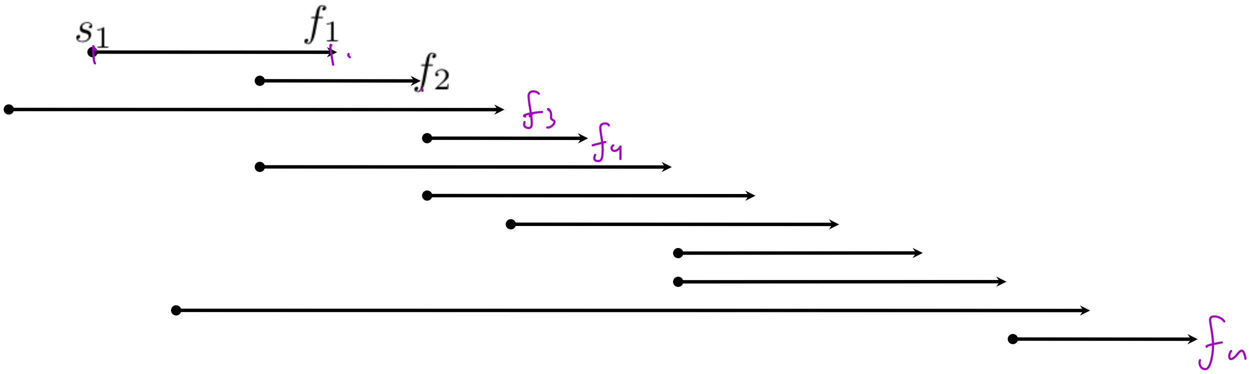
$$f_i \leq s_j$$

problem statement

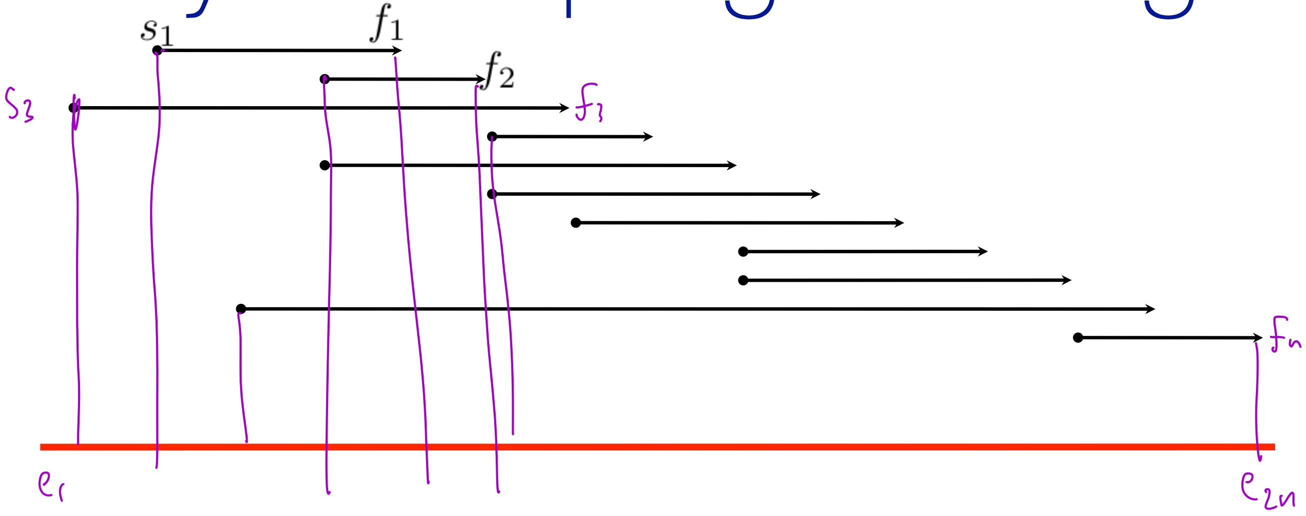
$$(a_1, \ldots, a_n)$$

$$(s_1, s_2, \ldots, s_n)$$

$$(f_1, f_2, \ldots, f_n) \text{ (SORTED)} \quad s_i < f_i$$

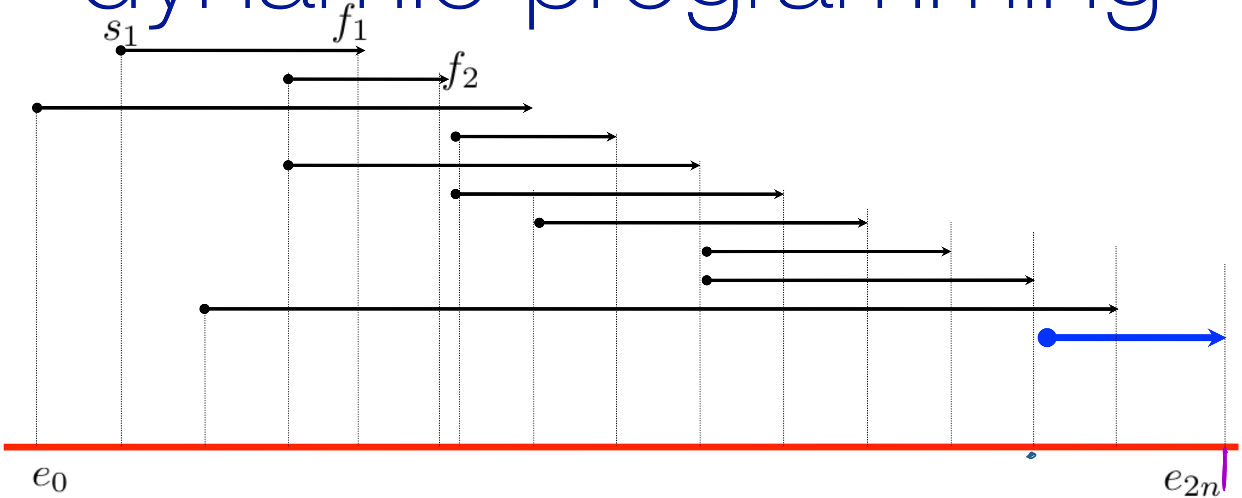


dynamic programming



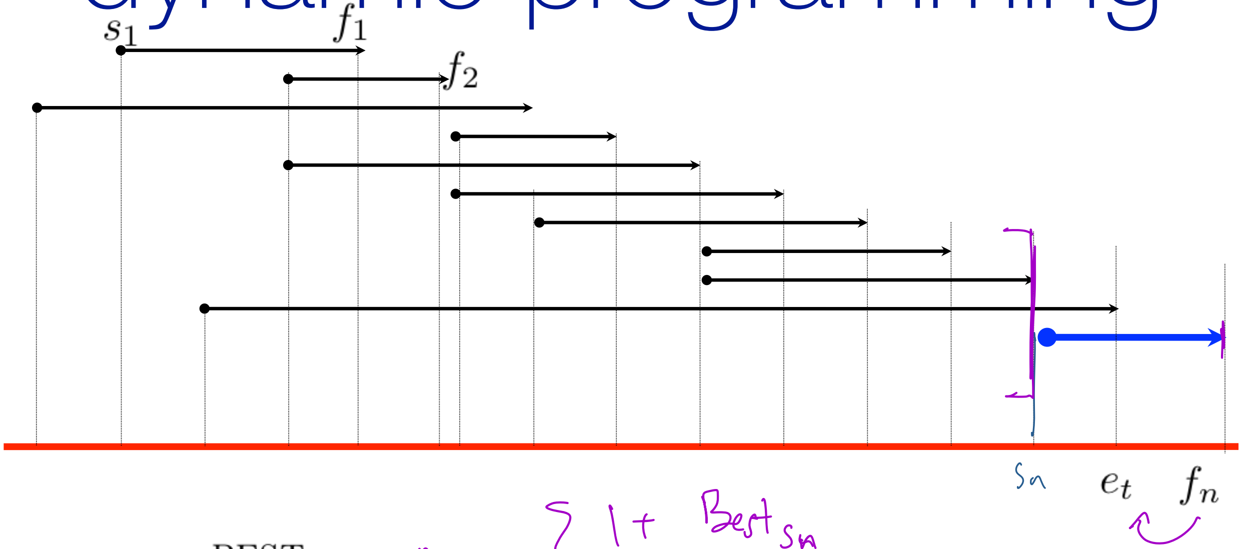
Lets draw all of the events on a timeline.

dynamic programming



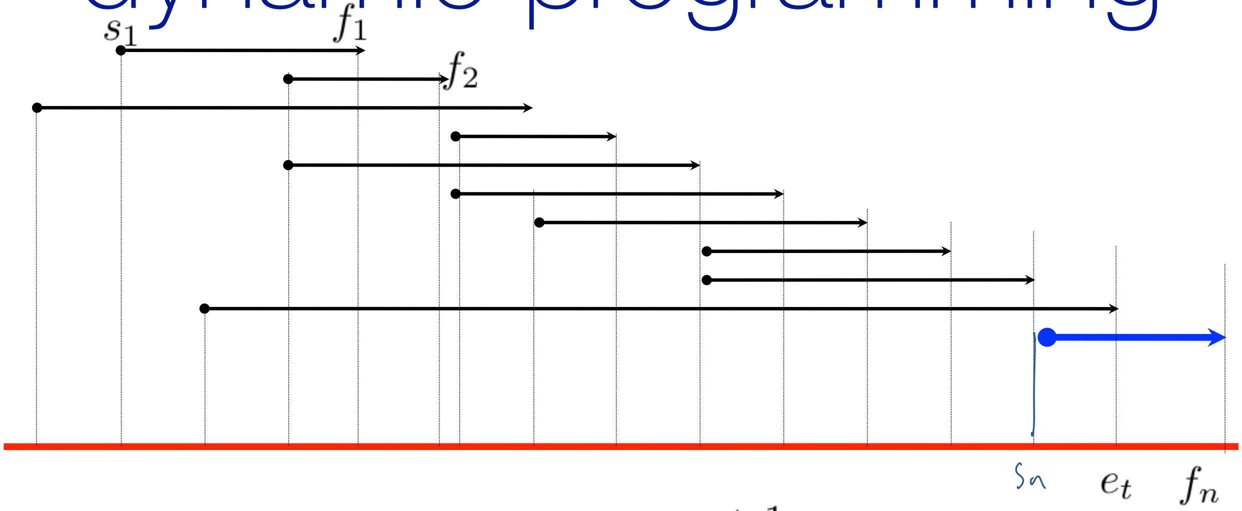
$Best_{2n} =$ Maximum number of non-overlapping activities possible among the first 2n events.

dynamic programming



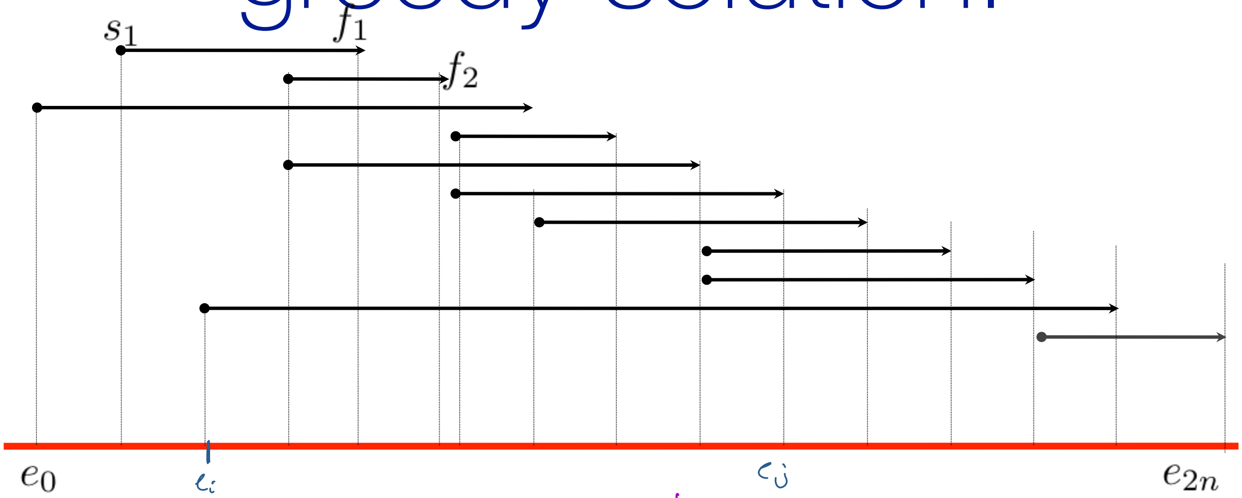
$$\mathrm{BEST}_{f_n} = \max \begin{cases} 1 + \mathrm{Best}_{s_n} \\ \mathrm{Best}_{e(f_n)-1} \end{cases}$$

dynamic programming



$$\text{BEST}_{f_n} = \max \begin{cases} \text{BEST}_{s_n} + 1 & \text{in: } a_n \\ \text{BEST}_{e_t} & \text{out: } a_n \end{cases}$$

greedy solution:

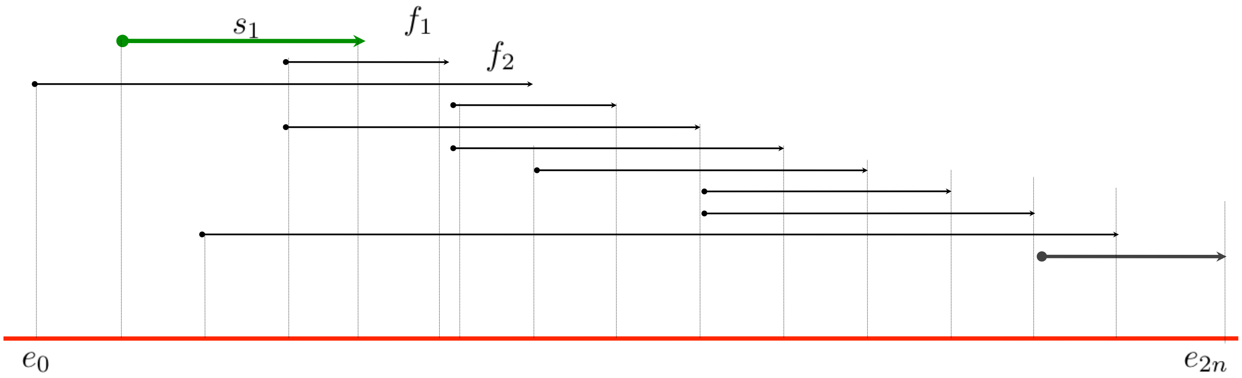


DEFINITION:

$\text{soltn}_{i,j} =$ max set of non-overlapping activities between events i, j.

GOAL: $\text{SOLTN}_{0,2n}$

greedy solution:



claim: the first action to finish in e[i,j] is always part of some $\textsc{soltn}_{i,j}$

claim: the first action to finish in e[i,j] is always part of some $\text{SOLTN}_{i,j}$

PROOF: Consider some optimal $\text{SOLTN}_{i,j}$.

Let a^* be the first action to finish in $e[i,j]$.

If $a^* \in \text{SOLTN}_{i,j}$, then the claim holds.

If $a^* \notin \text{SOLTN}_{i,j}$, let a be the first to finish in $\text{SOLTN}_{i,j}$

Consider $S_{i,j} = \text{SOLTN}_{i,j} - \{a\} \cup \{a^*\}$

① $|S_{i,j}| = |\text{SOLTN}_{i,j}|$

② $S_{i,j}$ is valid solution, nonoverlapping. Because

$e_{a^*} < e_a$. So a^* does not overlap with any events.

claim: the first action to finish in e[i,j] is always part of some $\text{SOLTN}_{i,j}$

PROOF:

Consider $\text{soltn}_{i,j}$ and let a^* be the first activity to finish in e[i,j].

If $a^* \in \text{soltn}_{i,j}$, then the claim follows.

If not, let a be the activity that finishes first in $\text{soltn}_{i,j}$.

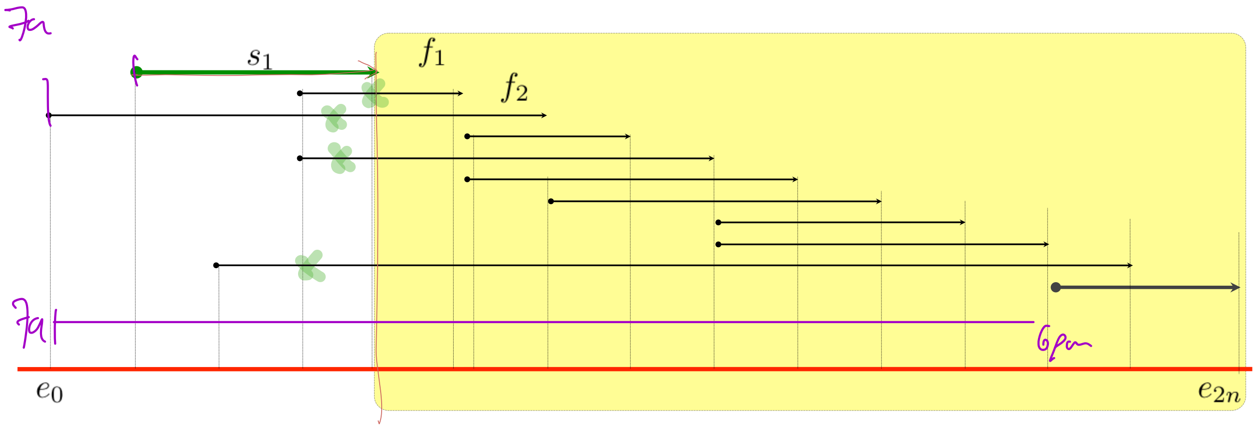
Consider a new solution that replaces a with a^*.

set union

$$\text{soltn}^*_{i,j} = \text{soltn}_{i,j} - \{a\} \cup \{a^*\}$$

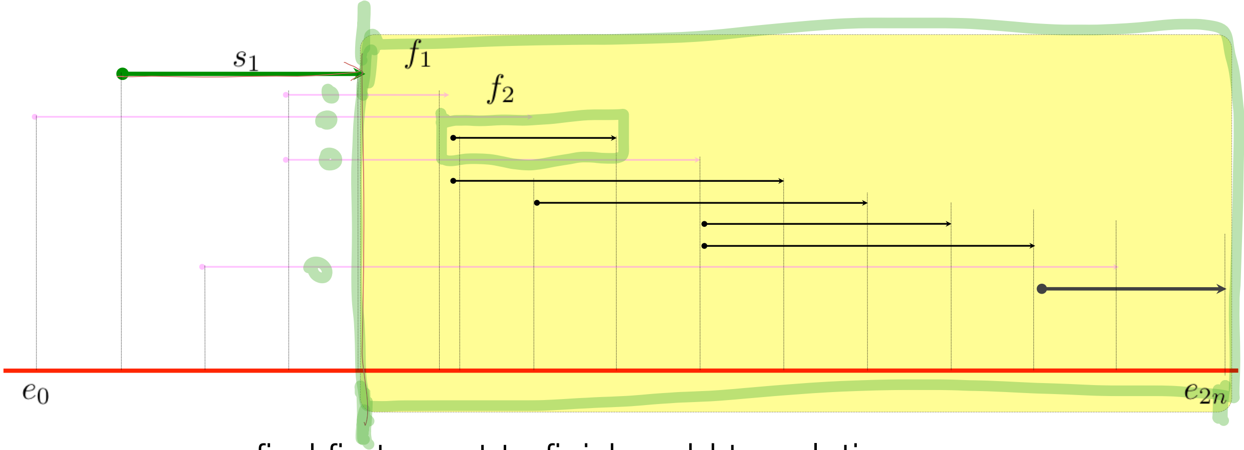
This new set is valid because a^* finishes before a and thus does not overlap with any activities. This new solution also has the same size and is therefore also optimal too.

greedy solution:



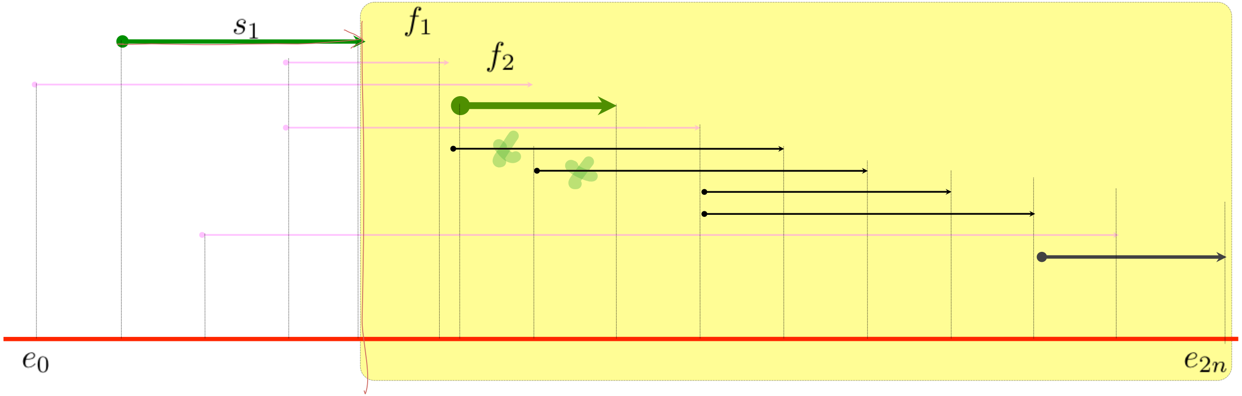
algorithm: find first event to finish. add to solution.
remove conflicting events.
continue.

greedy solution:



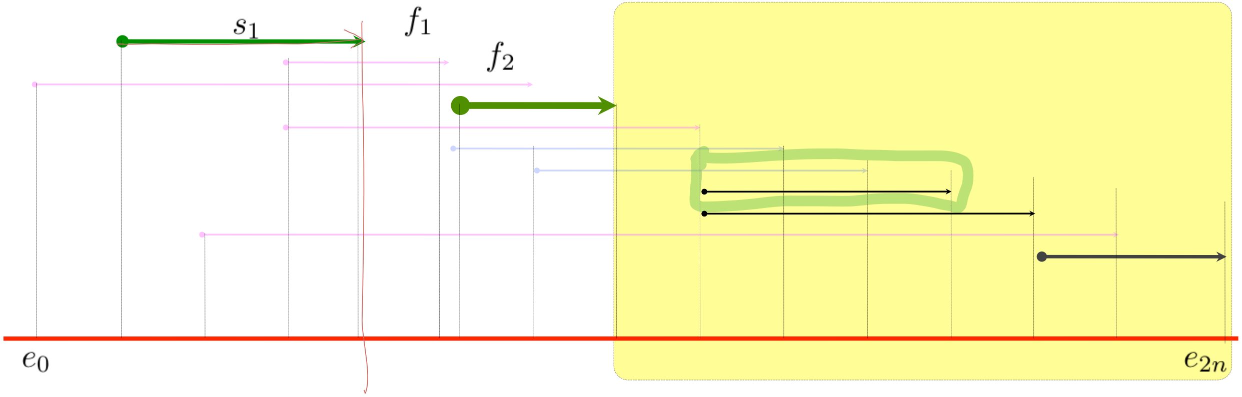
algorithm: find first event to finish. add to solution.
remove conflicting events.
continue.

greedy solution:



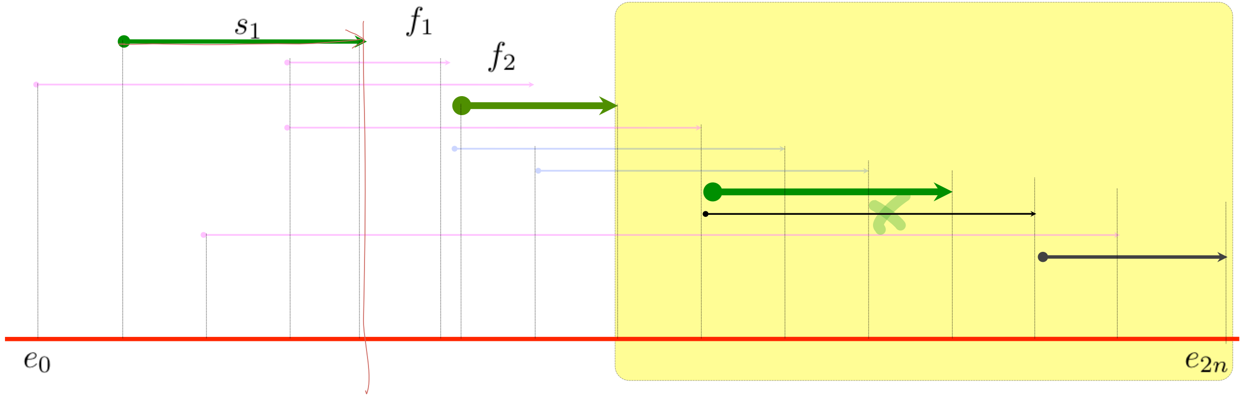
algorithm: find first event to finish. add to solution.
remove conflicting events.
continue.

greedy solution:



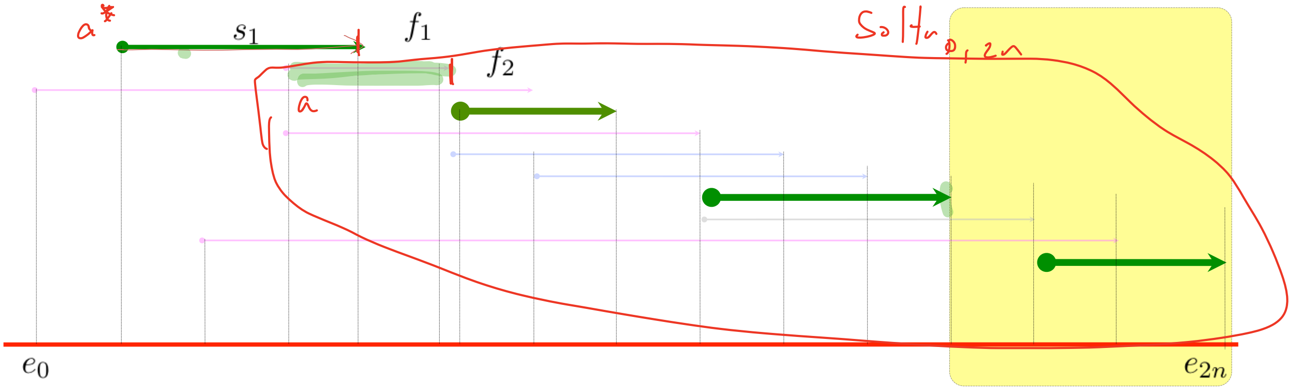
algorithm: find first event to finish. add to solution.
remove conflicting events.
continue.

greedy solution:



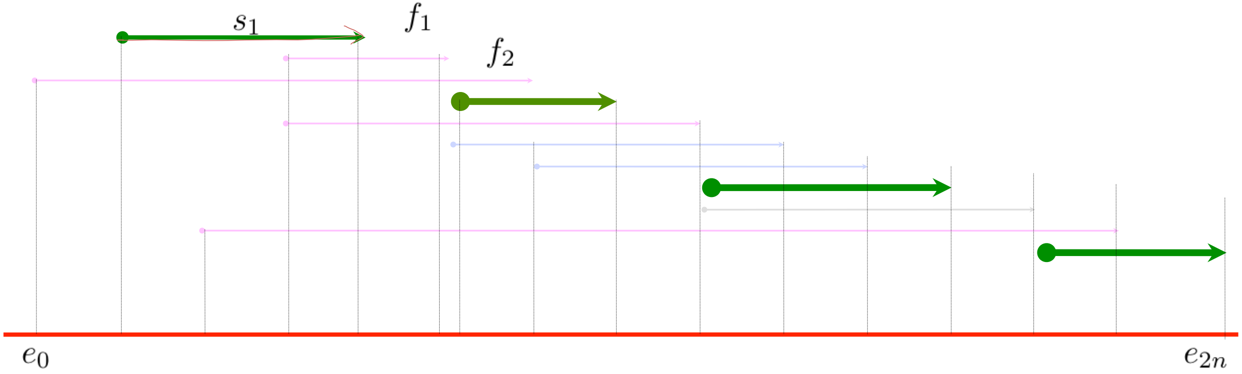
algorithm: find first event to finish. add to solution.
remove conflicting events.
continue.

greedy solution:



algorithm: find first event to finish. add to solution.
remove conflicting events.
continue.

greedy solution:



algorithm: find first event to finish. add to solution.
remove conflicting events.
continue.

running time

algorithm: find first event to finish. add to solution.
remove conflicting events.
continue.

$(f_1, f_2, \ldots, f_n)$ (sorted) $s_i < f_i$

Recap

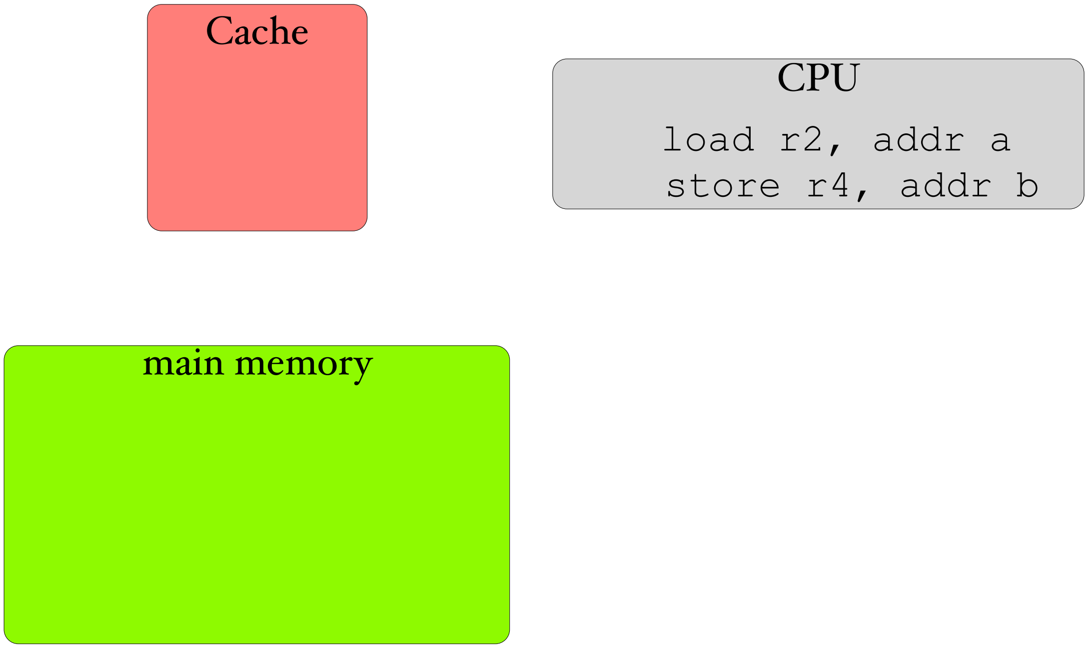
The main idea in this algorithm was the “exchange argument.”

We were able to identify an item (first to finish) that must be part of *some* optimal solution by exchanging this element with one that we can identify in any optimal solution.

Since its easy to identify the item that is first to finish, our algorithm is conversely simple, “greedy.”

caching

cache hit



question:

question:

How do we manage a fully-associate cache?

When it is full, which element do we replace?

problem statement

input:

output:

cache is

problem statement

input: K, the size of the cache
$d_1, d_2, \ldots, d_m$ memory accesses

output: schedule for that cache that minimizes # of cache misses while satisfying requests

cache is fully associative, line size is 1

contrast with reality

contrast with reality

In a real program, we may not know the future memory access patterns.

Some caches have additional restrictions, like line-size, associativity, etc.

Belady eviction rule

Belady eviction rule

Replace the element in the cache that is accessed "farthest into the future"

example

cache

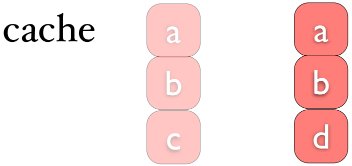
a

b

c

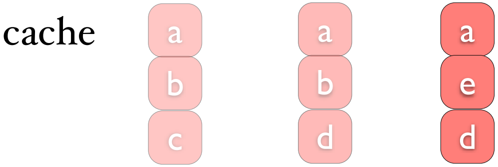
a b c d a d e a d b a e c e a

example



a b c d a d e a d b a e c e a

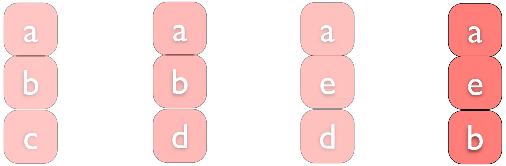
example



a b c d a d e a d b a e c e a

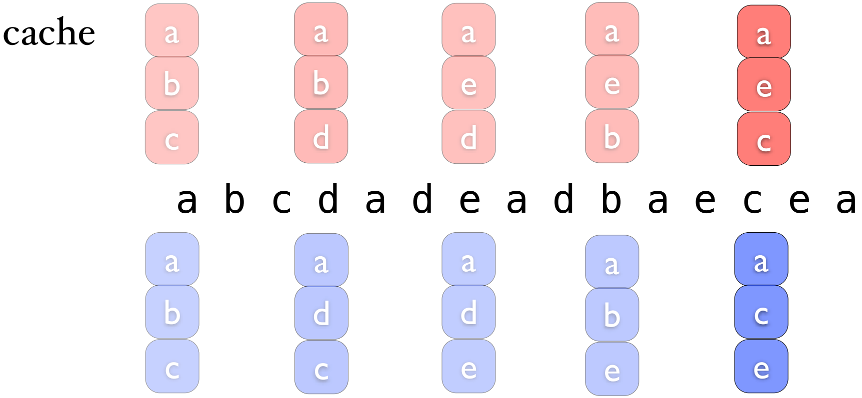
example

cache



a b c d a d e a d b a e c e a

example


cache
a b c
a b d
a e d
a e b
a e c
a b c d a d e a d b a e c e a
a b c
a d c
a d e
a b e
a c e

Surprising theorem

Surprising theorem

The schedule S_{ff} produced by the Belady "farthest in the future" eviction rule is optimal.

schedule

Schedule for access pattern $d_1,d_2,...,d_n$:

Reduced schedule:

schedule

Schedule for access pattern $d_1,d_2,\ldots,d_n$:

A list of instructions for each access that is either "NOP" or "evict x for y"

Reduced schedule:

schedule

Schedule for access pattern $d_1,d_2,\ldots,d_n$:

A list of instructions for each access that is either "NOP" or "evict x for y"

Reduced schedule:

A schedule in which"evict x for y" instruction only occurs when y is accessed.

schedule

Schedule for access pattern $d_1, d_2, \ldots, d_n$:

A list of instructions for each access that is either "NOP" or "evict x for y"

Reduced schedule:

A schedule in which"evict x for y" instruction only occurs when y is accessed.

Note: any schedule can be transformed into a reduced schedule with the same or fewer cache misses.

(Idea: starting at the end, defer "evict...t" until y is read)

Exchange lemma

Exchange lemma

Let S be a reduced schedule that agrees with S_{ff} on the first j accesses.

Then there exists a schedule S' that agrees with S_{ff} on the first j+1 accesses and has the same or fewer misses.

Some optimal
schedule.

S^*

S_{ff}

Some optimal schedule.

S^* S_1

Agrees with S_{ff} on the first access.

S_{ff}

Some optimal schedule.

S^* S_1 S_2

Agrees with S_{ff} on the first access.

Agrees with S_{ff} on the first two accesses.

S_{ff}

Some optimal schedule.

S^* S_1 S_2 S_3 S_{n-1} S_{ff}

Agrees with S_{ff} on the first access.

Agrees with S_{ff} on the first two accesses.

Agrees with S_{ff} on the first three accesses.

S_{ff} has the same number of cache misses as S^*.

Proof of Lemma

Let S be a reduced sched that agrees with S_{ff} on the first j items.
There exists a reduced sched **S'** that agrees with S_{ff} on the first j+1 items and has the same or fewer #misses as S.

Proof of Lemma

Let S be a reduced sched that agrees with S_{ff} on the first j items.
There exists a reduced sched **S'** that agrees with S_{ff} on the first j+1 items and has the same or fewer #misses as S.

At time j, both S and S_{ff} have the same state.
Let d be the element accessed at time j+1.

Proof of lemma

State of the cache after J operations under the two schedules.

easy case 1

easy case 2

Proof of lemma

State of the cache after J operations under the two schedules.

easy case 1 d is in the cache.

easy case 2

Proof of lemma

State of the cache after J operations under the two schedules.

easy case 1 d is in the cache.

Both S and S_{ff} agree since both do NOPs at j+1.

easy case 2

Proof of lemma

State of the cache after J operations under the two schedules.

easy case 1 d is in the cache.

Both S and S_{ff} agree since both do NOPs at j+1.

easy case 2 d is not in the cache, but both "evict e for d."

Proof of lemma

State of the cache after J operations under the two schedules.

easy case 1 d is in the cache.

Both S and S_{ff} agree since both do NOPs at j+1.

easy case 2 d is not in the cache, but both “evict e for d.”

Both S and S_{ff} agree at j+1.

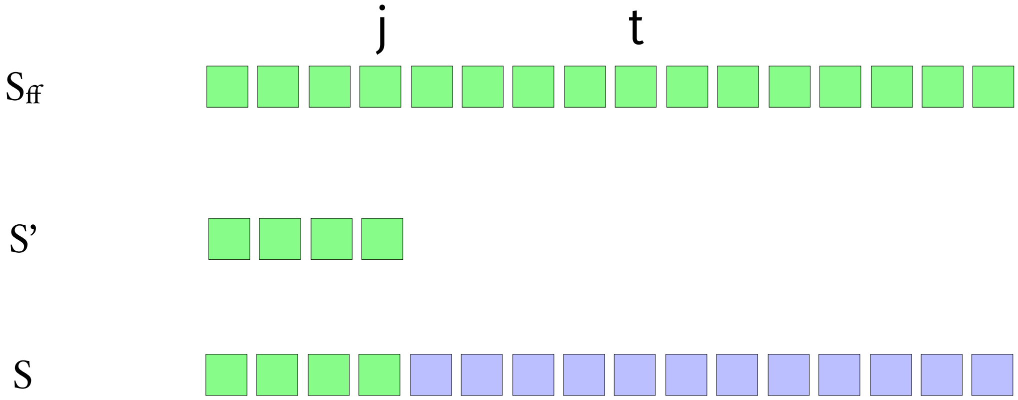
Proof of lemma

case 3

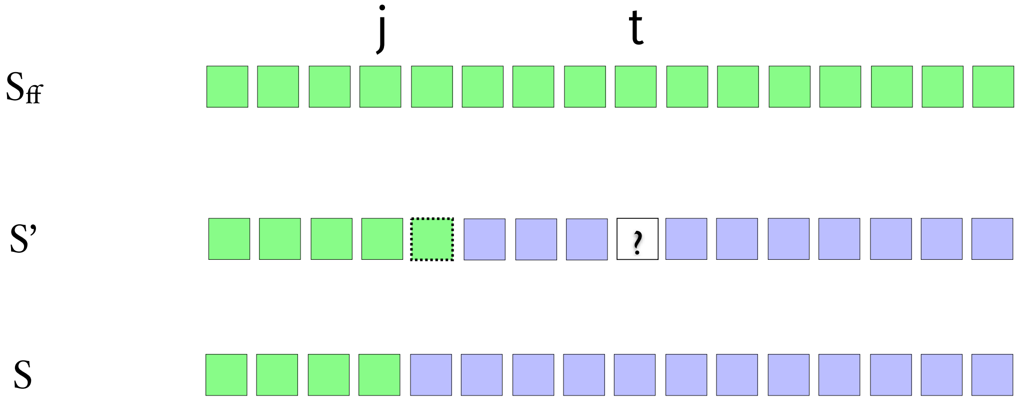
Proof of lemma

case 3 S evicts "e for d", and S_{ff} evicts "e for f"

Timeline



Timeline



Copy j+1 from S_{ff}. Then copy from S until t (the first time that either e or f are accessed). Then copy from S until the end.

Proof of lemma

Let t be the first access that either e or f are accessed.

What if t=e:

Proof of lemma

what if t=e ?

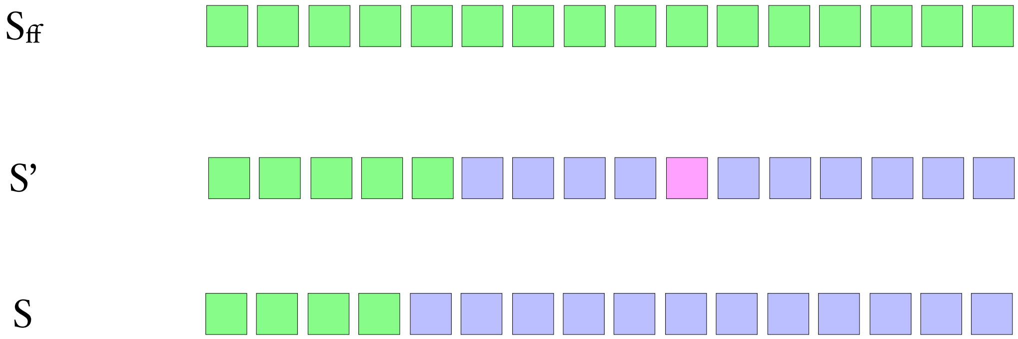
Proof of lemma

what if t=f ?

Proof of lemma

what if t is neither e nor f ?

What have we shown



Let S be a reduced sched that agrees with S_{ff} on the first j items.
There exists a reduced sched **S’** that agrees with S_{ff} on the first j+1 items and has the same or fewer #misses as S.

Let S be a reduced sched that agrees with S_{ff} on the first j items.
There exists a reduced sched **S'** that agrees with S_{ff} on the first j+1 items and has the same or fewer #misses as S.

S^* $\qquad\qquad$ S_{ff}

Recap

The greedy algorithm is quite simple.

But the analysis for why the solution works is more subtle and complicated.

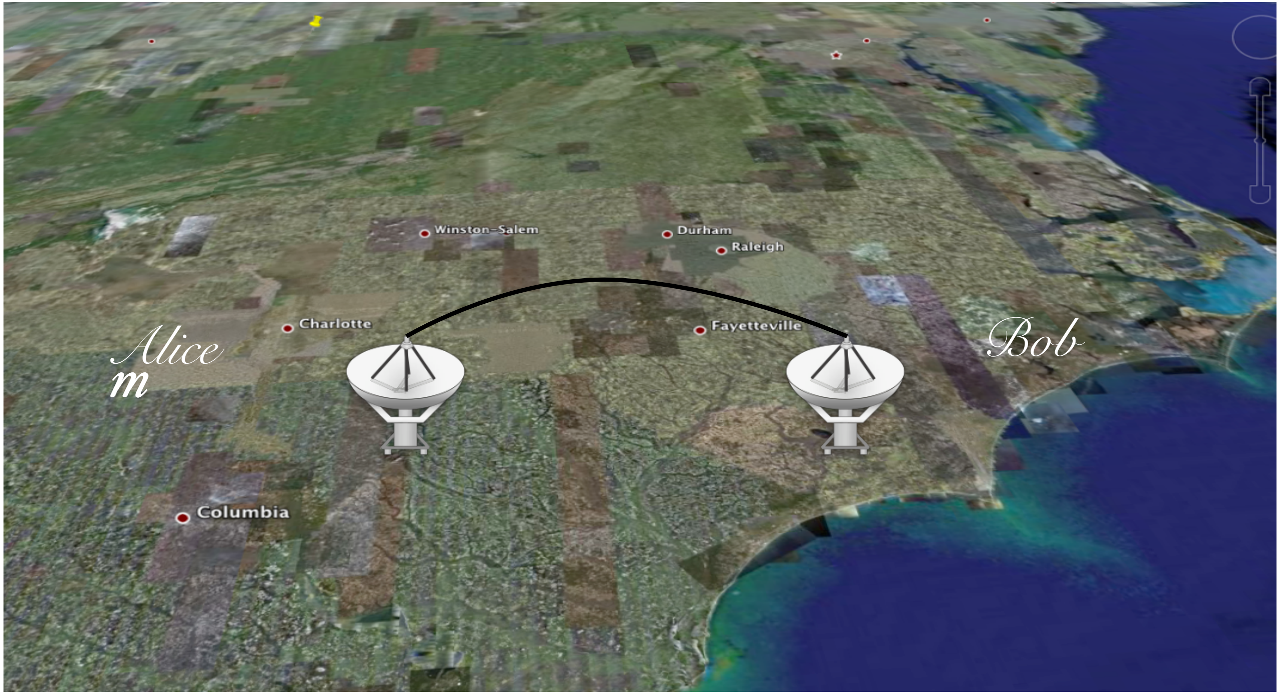
In this case, we had to apply the exchange lemma multiple times to prove optimality.

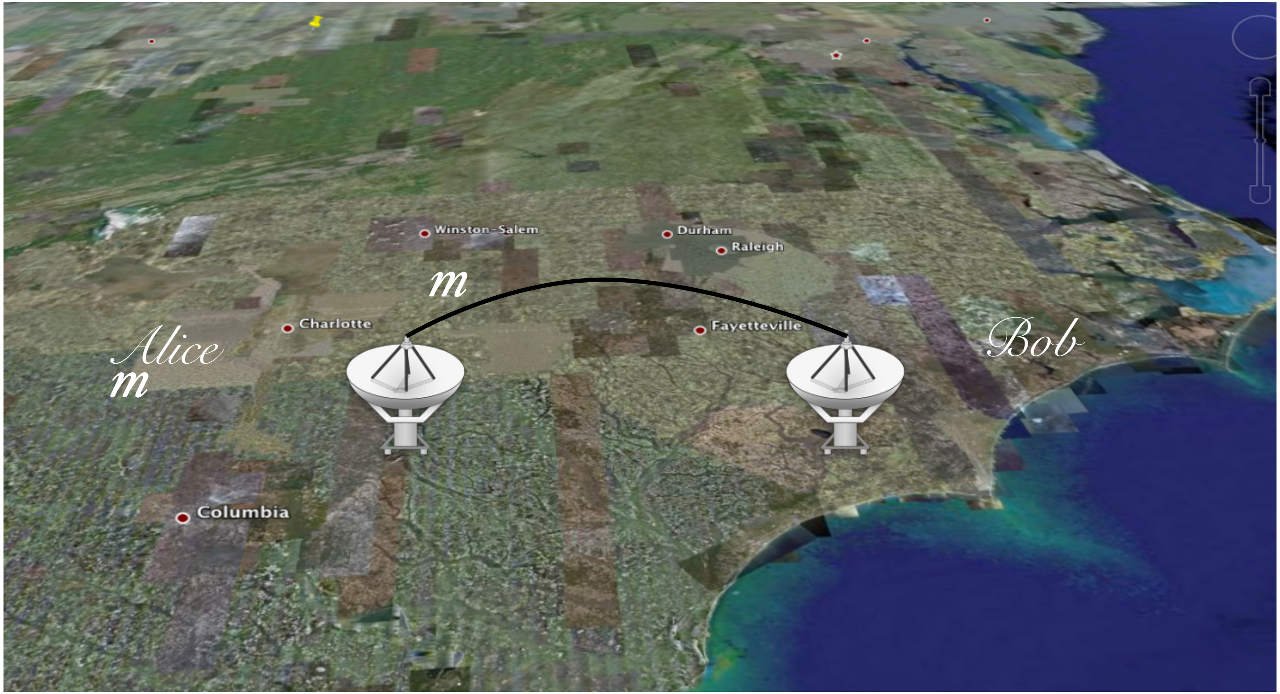
Huffman

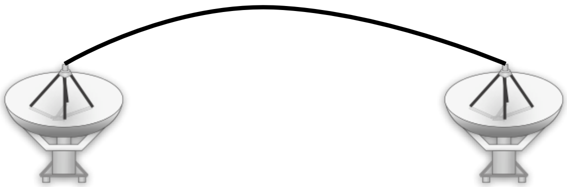
L10

CS4800


image: wikimedia

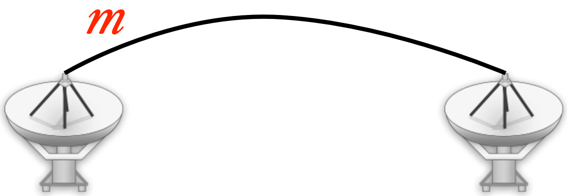

Winston-Salem
Durham
Raleigh
Charlotte
Fayetteville
Alice
m
Bob
Columbia


Winston-Salem
Durham
Raleigh
m
Charlotte
Fayetteville
Alice
m
Bob
Columbia



MOSCOW — President Vladimir V. Putin's typically theatrical order to withdraw the bulk of Russian forces from Syria, a process that the Defense Ministry said it began on Tuesday, seemingly caught Washington, Damascus and everybody in between off guard — just the way the Russian leader likes it.

By all accounts, Mr. Putin delights at creating surprises, reinforcing Russia's newfound image as a sovereign, global heavyweight and keeping him at the center of world events.



MOSCOW — President Vladimir V. Putin's typically theatrical order to withdraw the bulk of Russian forces from Syria, a process that the Defense Ministry said it began on Tuesday, seemingly caught Washington, Damascus and everybody in between off guard — just the way the Russian leader likes it.

By all accounts, Mr. Putin delights at creating surprises, reinforcing Russia's newfound image as a sovereign, global heavyweight and keeping him at the center of world events.

$c \in C$	f_c	T
e:	235	
i:	200	
o:	170	
u:	87	
p:	78	
g:	47	
b:	40	
f:	24	

881

$c \in C$	f_c	T	ℓ_c
e:	235	000	3
i:	200	001	3
o:	170	010	3
u:	87	011	3
p:	78	100	3
g:	47	101	3
b:	40	110	3
f:	24	111	3

881

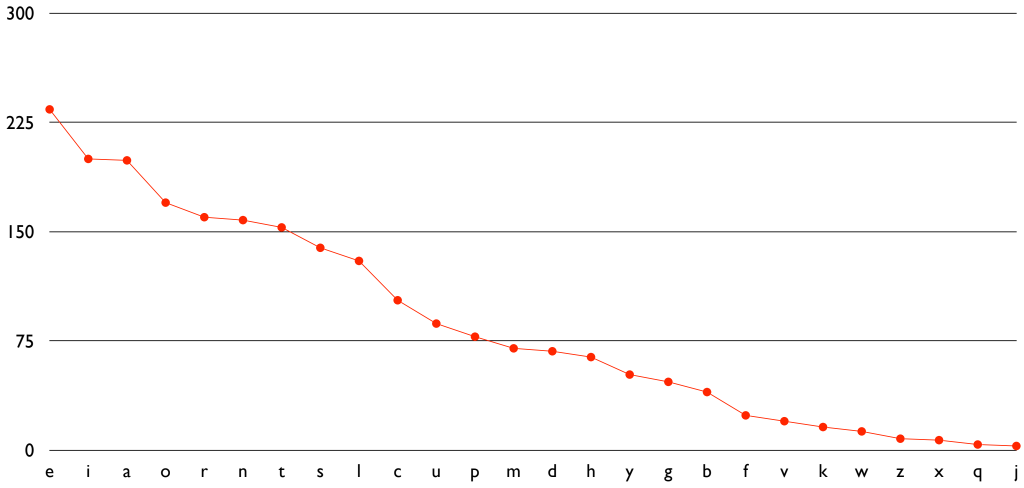
def: cost of an encoding

$$B(T, \{f_c\}) = \sum_{c \in C} f_c \cdot \ell_c$$

$c \in C$	f_c	T	ℓ_c
e:	235	000	3
i:	200	001	3
o:	170	010	3
u:	87	011	3
p:	78	100	3
g:	47	101	3
b:	40	110	3
f:	24	111	3
	881		

character frequency

```
e: 234803
i: 200613
a: 198938
o: 170392
r: 160491
n: 158281
t: 152570
s: 139238
l: 130172
c: 103307
 u: 87211
 p: 78077
 m: 70504
 d: 68007
 h: 64165
 y: 51527
 g: 47011
 b: 40351
 f: 24110
 v: 20103
 k: 16012
 w: 13825
  z: 8439
  x: 6926
  q: 3729
  j: 3075
```



Morse code

image http://en.wikipedia.org/wiki/Morse_code

Morse code


International Morse Code
- 1 dash = 3 dots.
- The space between parts of the same letter = 1 dot.
- The space between letters = 3 dots.
- The space between words = 7 dots.

def: prefix-free code

def: prefix-free code

$$\forall x, y \in C, x \neq y \implies \textsc{code}(x) \text{ not a prefix of } \textsc{code}(y)$$

def: prefix code

$$\forall x, y \in C, x \neq y \implies \textsc{code}(x) \text{ not a prefix of } \textsc{code}(y)$$

```
e: 235    0
i: 200    10
o: 170    110
u: 87     1110
p: 78     11110
g: 47     111110
b: 40     1111110
f: 24     11111110
```

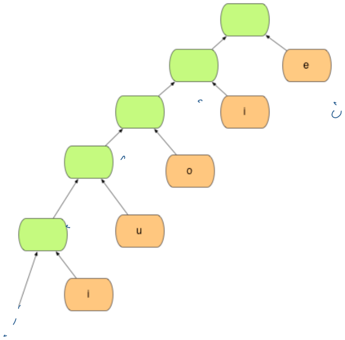
decoding a prefix code

e:	235	0
i:	200	10
o:	170	110
u:	87	1110
p:	78	11110
g:	47	111110
b:	40	1111110
f:	24	11111110

111111010111110

code to binary tree

e:	235	0
i:	200	10
o:	170	110
u:	87	1110
p:	78	11110
g:	47	111110
b:	40	1111110
f:	24	11111110

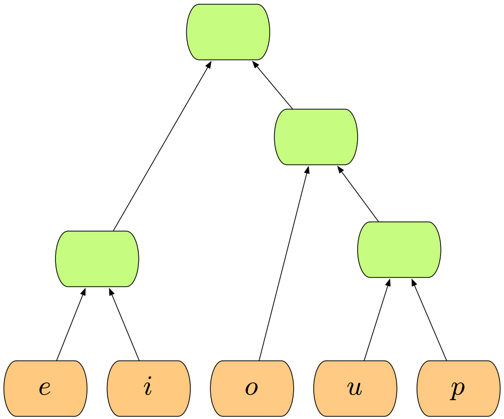


11111101011110

prefix code

binary tree

use tree to encode



$c \in C$	f_c	T	ℓ_c
e:	235	00	2
i:	200	01	2
o:	170	10	2
u:	87	110	3
p:	78	111	3

goal

GIVEN THE

goal

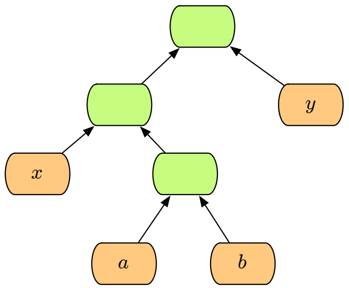
(all frequencies are > 0)

GIVEN THE CHARACTER FREQUENCIES $\{f_c\}_{c \in C}$

PRODUCE A PREFIX CODE T WITH SMALLEST COST

$$\min_T B(T, \{f_c\})$$

property

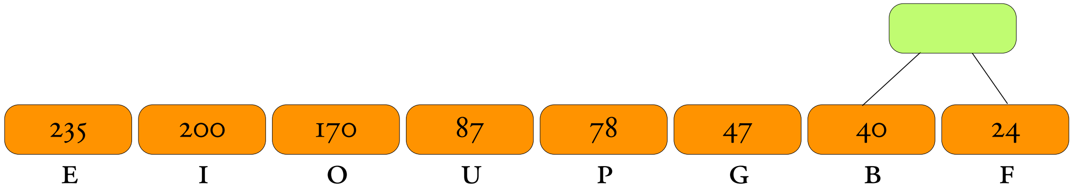


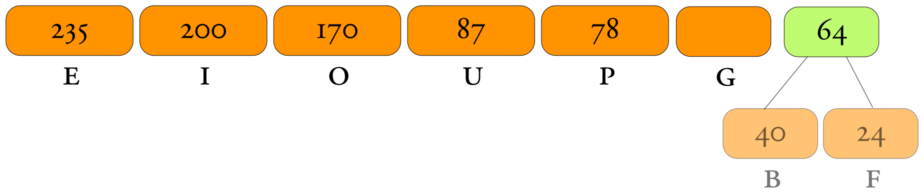
LEMMA:OPTIMAL TREE MUST BE FULL.

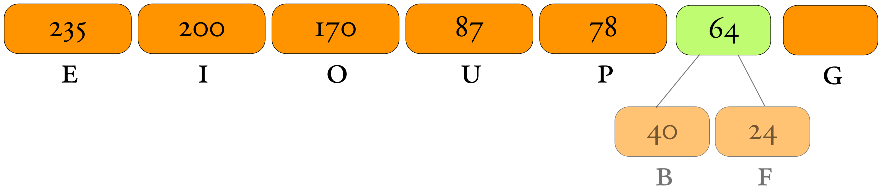
divide & conquer?

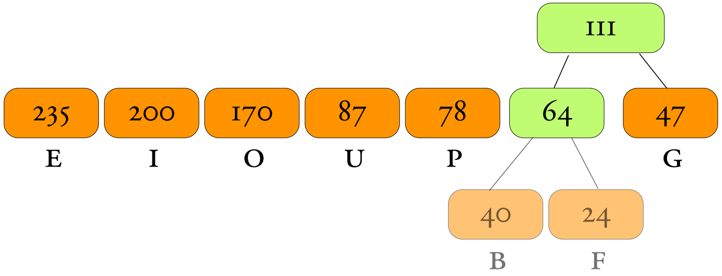
counter-example

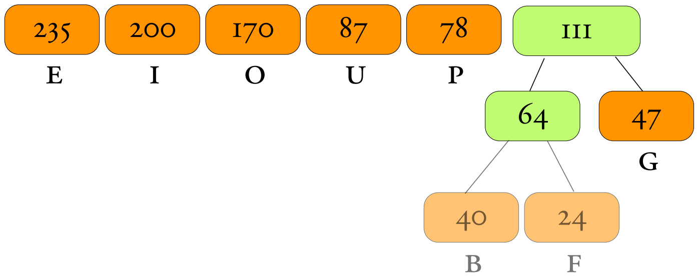
```
e: 32
i: 25
o: 20
u: 18
p: 5
```

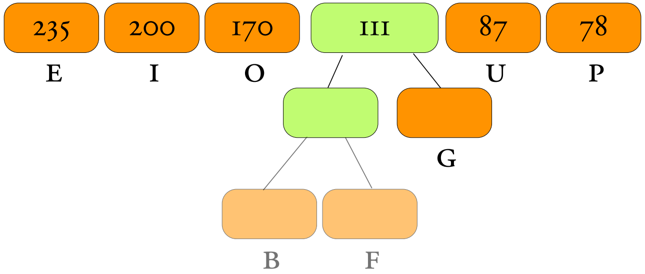

235
200
170
87
78
47
40
24
E
I
O
U
P
G
B
F


235
E
200
I
170
O
87
U
78
P
G
64
40
B
24
F

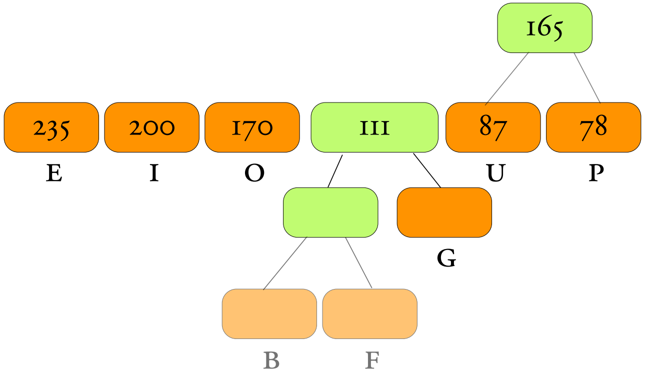

235
E
200
I
170
O
87
U
78
P
64
G
40
B
24
F

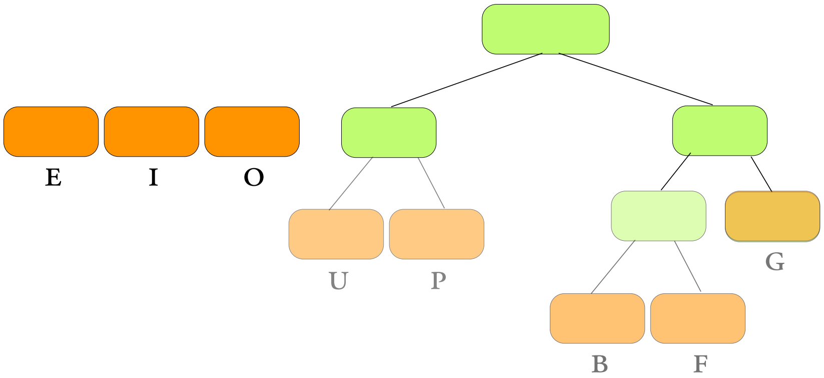

111
235
200
170
87
78
64
47
E
I
O
U
P
G
40
24
B
F

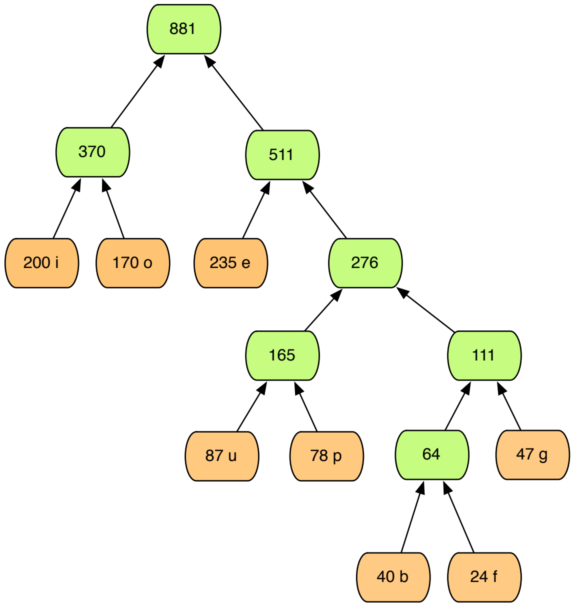

235
E
200
I
170
O
87
U
78
P
111
64
47
G
40
B
24
F

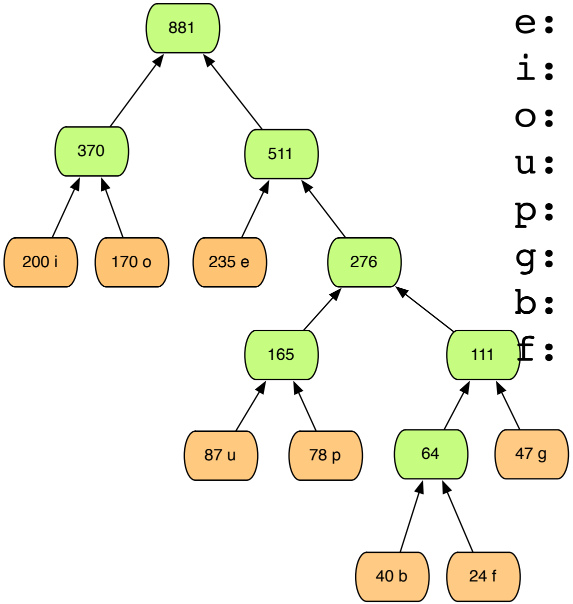


235
200
170
111
87
78
E
I
O
U
P
G
B
F

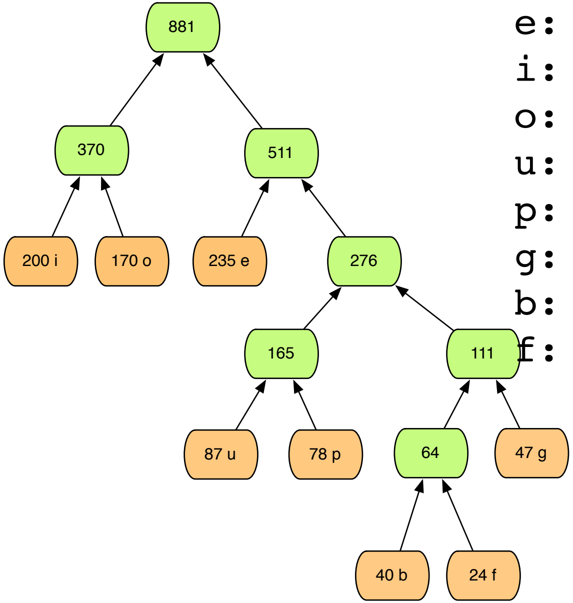

165
235
200
170
111
87
78
E
I
O
U
P
G
B
F


E
I
O
U
P
B
F
G


881
370
511
200 i
170 o
235 e
276
165
111
87 u
78 p
64
47 g
40 b
24 f



e:	235	01
i:	200	11
o:	170	10
u:	87	0011
p:	78	0010
g:	47	0000
b:	40	00011
f:	24	00010



e:	235	01	470
i:	200	11	400
o:	170	10	340
u:	87	0011	348
p:	78	0010	312
g:	47	0000	188
b:	40	00011	200
f:	24	00010	120
			2378

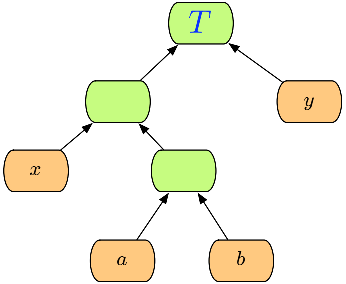
objective

exchange argument

LEMMA:

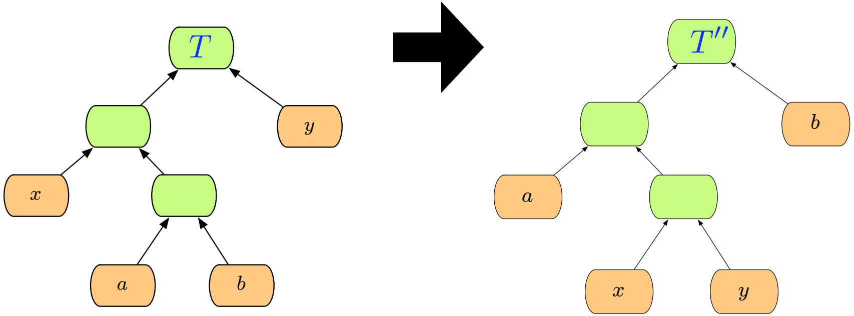
exchange argument

LEMMA: Let $x, y \in C$ be characters with smallest frequencies f_x, f_y. There exists an optimal prefix code T'' for C in which x, y are siblings. That is, the codes for x, y have the same length and only differ in the last bit.



exchange argument

LEMMA: Let $x, y \in C$ be characters with smallest frequencies f_x, f_y. There exists an optimal prefix code T'' for C in which x, y are siblings. That is, the codes for x, y have the same length and only differ in the last bit.



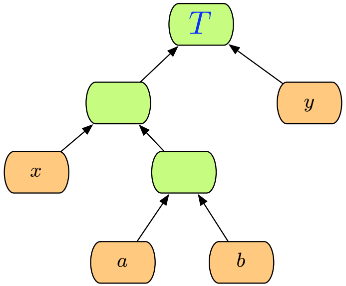
exchange argument

LEMMA: Let $x, y \in C$ be characters with smallest frequencies f_x, f_y. There exists an optimal prefix code T'' for C in which x, y are siblings. That is, the codes for x, y have the same length and only differ in the last bit.

PROOF:

exchange argument

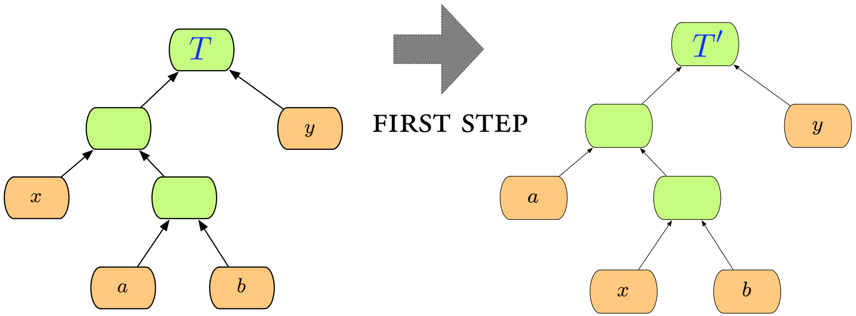
LEMMA: Let $x, y \in C$ be characters with smallest frequencies f_x, f_y. There exists an optimal prefix code T'' for C in which x, y are siblings. That is, the codes for x, y have the same length and only differ in the last bit.



FIRST STEP

exchange argument

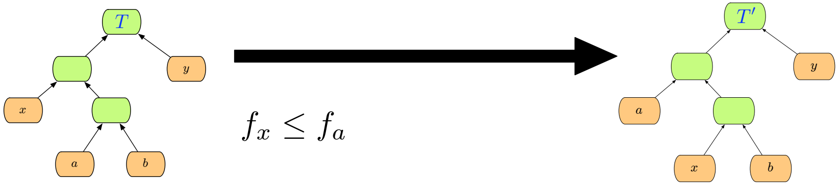
LEMMA: Let $x, y \in C$ be characters with smallest frequencies f_x, f_y. There exists an optimal prefix code T'' for C in which x, y are siblings. That is, the codes for x, y have the same length and only differ in the last bit.



$f_a \leq f_b$ $\quad$ $f_x \leq f_a$

$f_x \leq f_y$ $\quad$ $f_y \leq f_b$

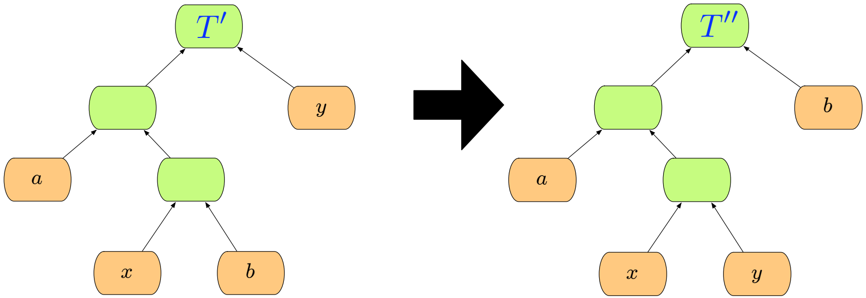
T
y
x
a
b
T'
y
a
x
b



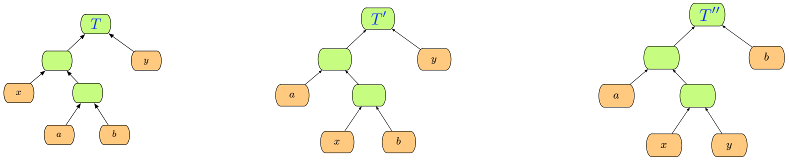
$$B(T) = \sum_c f_c \ell_c + f_x \ell_x + f_a \ell_a \qquad B(T') = \sum_c f_c \ell'_c + f_x \ell'_x + f_a \ell'_a$$

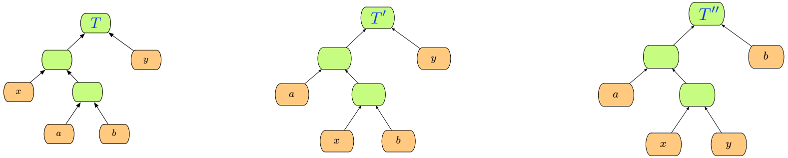
$$B(T) - B(T') \geq 0$$

exchange argument



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


T
x
y
a
b
T′
a
y
x
b
T″
a
b
x
y

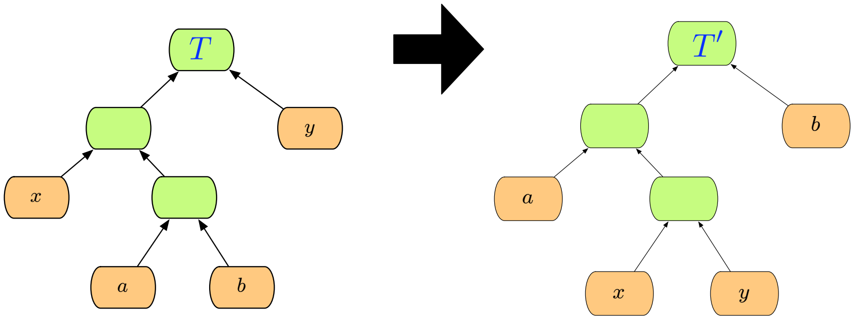


$$B(T) - B(T') \geq 0 \qquad B(T') - B(T'') \geq 0$$

T'' IS ALSO OPTIMAL

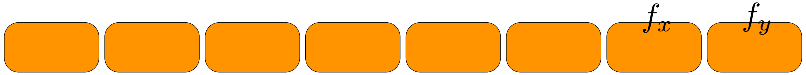
exchange argument

LEMMA: Let $x, y \in C$ be characters with smallest frequencies f_x, f_y. There exists an optimal prefix code T'' for C in which x, y are siblings. That is, the codes for x, y have the same length and only differ in the last bit.

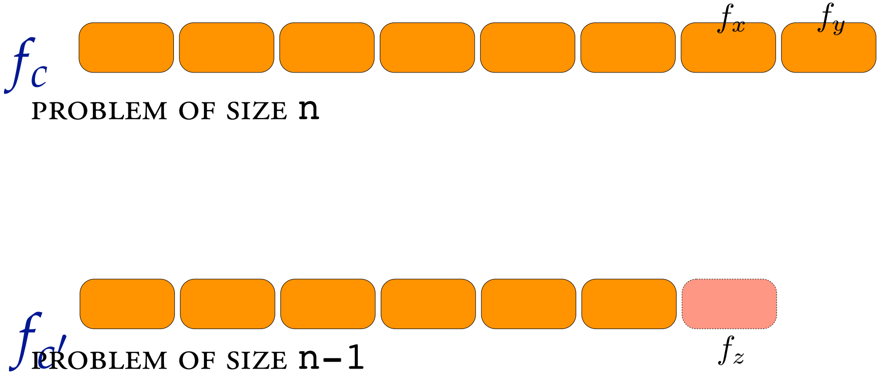


optimal sub-structure

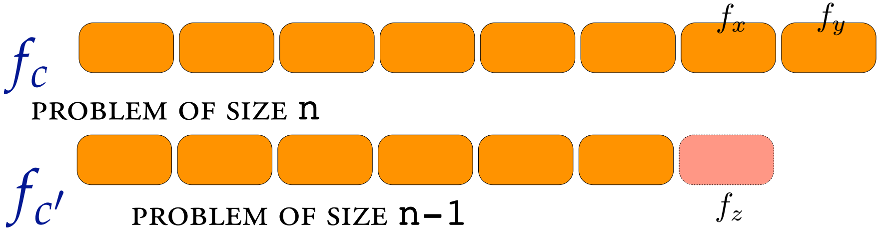
f_c



optimal sub-structure

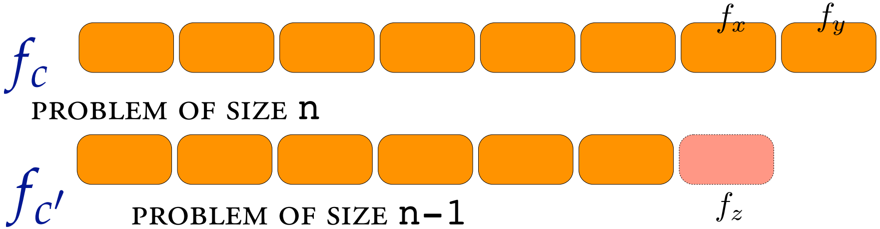


optimal sub-structure

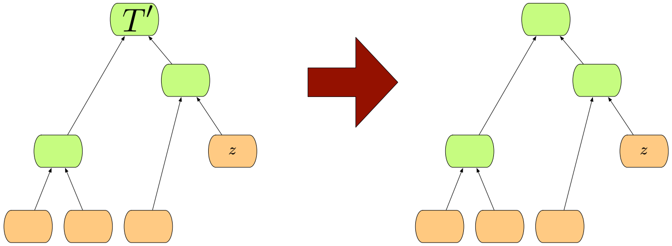


LEMMA:

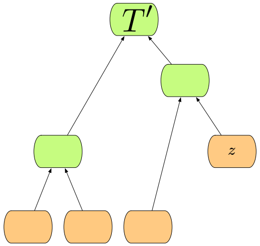
optimal sub-structure



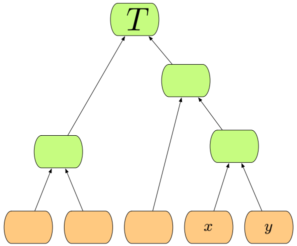
Lemma: The optimal solution for T consists of computing an optimal solution for T' and replacing the left z with a node having children x, y.



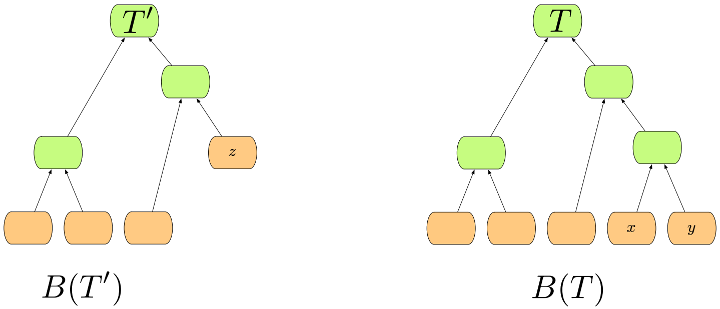
T'
z
z



$B(T')$



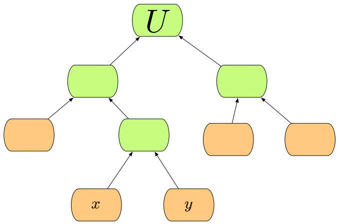
$B(T)$



$$B(T') = B(T) - f_x - f_y$$

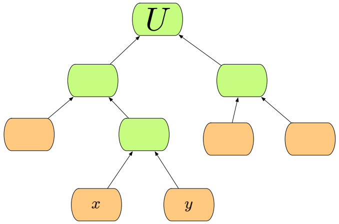
Suppose T is not optimal

Suppose T is not optimal

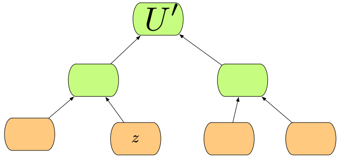


$$B(U) < B(T)$$

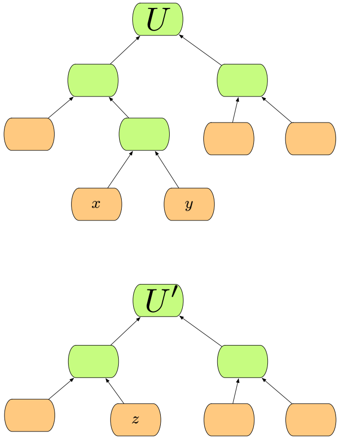
Suppose T is not optimal



$$B(U) < B(T)$$



Suppose T is not optimal



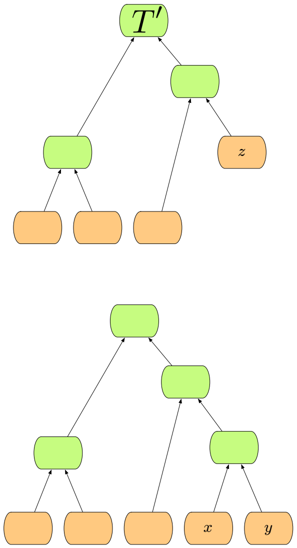
$$B(U) < B(T)$$

$$B(U') = B(U) - f_x - f_y$$

< B(T) - FX - FY

But this implies that B(T') was not optimal.

therefore



summary of argument