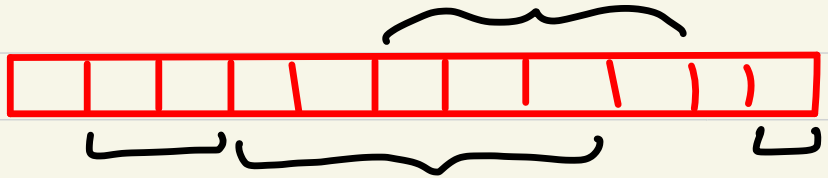
Q => Given an array of characters, print all possible subsequences of the given array.

example → ['a', 'b', 'c']

ans →
'a'
'b'
'c'
'ab'
'bc'
'ac'
'abc'
'' → empty

Subsets Vs Subsequence Vs Subarray Vs Substring

Subarray → A subarray is a contiguous cross-section of an array and it maintains the relative order of the elements.



Ex → [1, 2, 3, 4]

1
2
3
4
1, 2
2, 3
3, 4
1, 2, 3
2, 3, 4
1, 2, 3, 4

for an array of size n,
there are approx $\frac{n \times (n+1)}{2}$
non empty Subarrays.

x, ..., y ⟶ [x, ..., y]

#Substrings→ A substring is a contiguous cross-section of a string and it maintains the relative order of the elements.

Subsets → A subset contains elements of the given array, where they are not bound to be in contiguous order and the relative ordering of the elements doesn't matter.

2 2 2 2

ex → [1, 2, 3, 4].

element

pick ← → avoid

{1} {2} {3} {4}

{1,2} {3,1} {4,1} {2,3}

{2,4} {3,4} {1,2,3} {1,2,4}

{3,1,4} {2,3,4} {4,2,3 }

{ } → empty

for a given array → we have 2^n subsets

where n is the length of the given array.

[1, 2, 3]

✓	✓	✓	→	{1, 2, 3}
✓	X	✓	→	{1, 3}
✓	X	X		{1}
X	✓	X		{2}
X	X	✓		{3}
✓	✓	X		{1, 2}
X	✓	✓		{2, 3}
X	X	X		{} → empty

[ϕ]

→ ϕ

Subsequences → A subset contains elements of the given array, where they are not bound to be in contiguous order and the relative ordering of the elements does matter.

[1, 2, 3] → {3, 1} is not a subsequence.

{1} {2} {3}

{1, 2} {2, 3} {1, 3}

{1, 2, 3} { } empty

→ Subsets & subsequences can be defined for both arrays & strings.

Str = "abc"

→ "a"
"b"
"c"
"ab"
"bc"
"ac"
"abc"
"" → empty.

Q => Given an array of characters, print all possible subsequences of the given array.

example → ['a', 'b', 'c']

?

Recursively

ans →

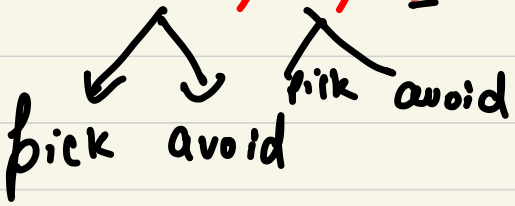
'a'
'b'
'c'
'ab'
'bc'
'ac'
'abc'
''

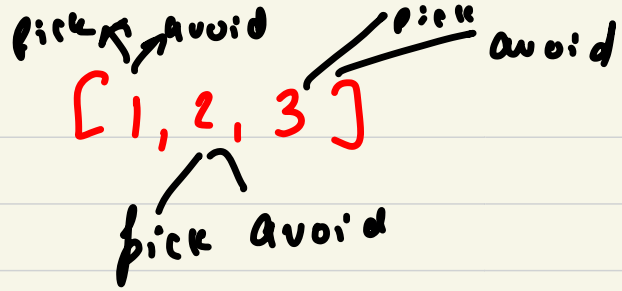
→ empty

⇒ We have an array of length n, then total no. of subsequences will be 2^n.

↳ picking elements in non contiguous order

$[a_1, a_2, a_3 \ldots\ldots a_n]$

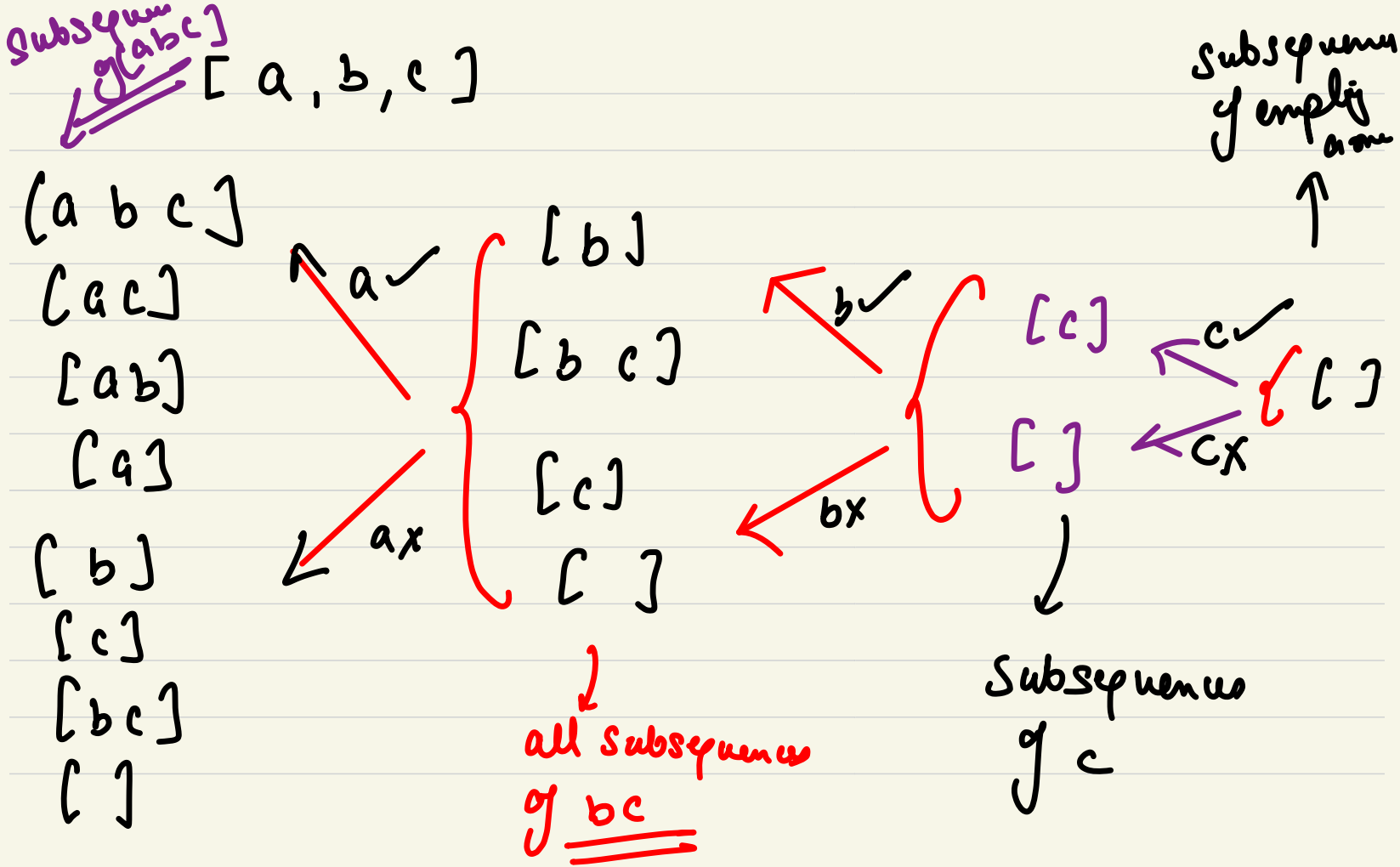




2 2 2 ..... 2 → 2^n

[a_1 a_2 a_3 ..... a_n]

We want to recursively write a func^n f, to print all subsequences.


Subsequence of [abc]
[a, b, c]
Subsequence of empty array
[a b c]
[a c]
[a b]
[a]
[b]
[c]
[b c]
[]
a
ax
[b]
[b c]
[c]
[]
b
bx
[c]
[]
c
cx
[]
all subsequences of bc
Subsequence of c

if we want to get all the subsequences of a given array, by considering the property that every element got 2 choices → pick
→ avoid

$f(arr, i, result) = f(arr, i+1, result + arr[i])$ — pick

$f(arr, i+1, result)$ — avoid

f ( ) function prints all subsequences of a given array from index $[i, n-1]$

result → string → storing all the subsequences

i → int → to iterate on the array

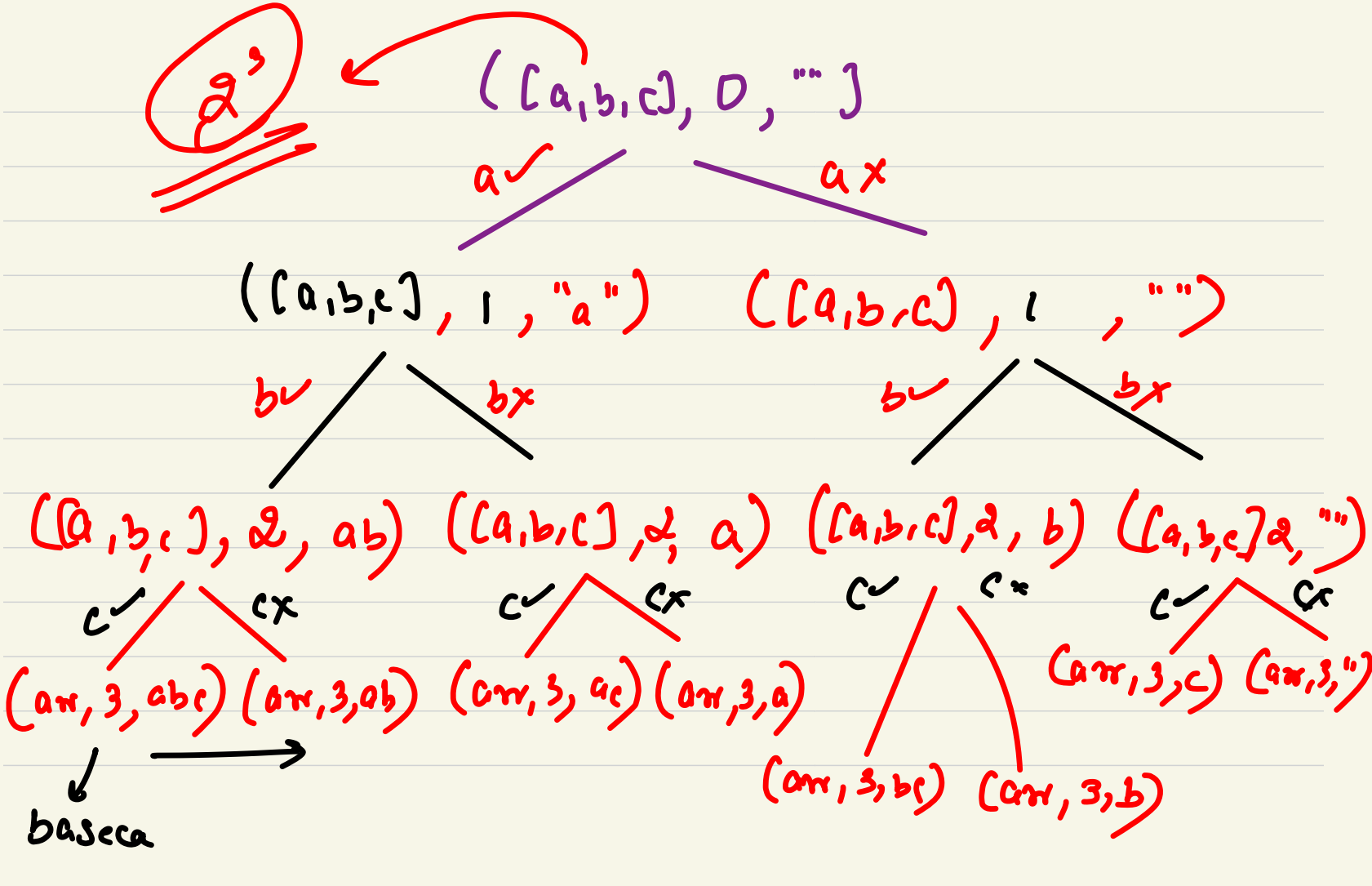
arr → array under consideration

(n → length of array)

Base Case

when array is exhausted

```
if (i == arr.length) {
    print(result)
    return
}
```


a'
([a,b,c], 0, "")
a✓
a✗
([a,b,c], 1, "a")
([a,b,c], 1, "")
b✓
b✗
b✓
b✗
([a,b,c], 2, ab)
([a,b,c], 2, a)
([a,b,c], 2, b)
([a,b,c], 2, "")
c✓
c✗
c✓
c✗
c✓
c✗
(aw, 3, abc)
(aw, 3, ab)
(aw, 3, ac)
(aw, 3, a)
(aw, 3, bc)
(aw, 3, b)
(aw, 3, c)
(aw, 3, "")
basecа

2 → a b c

3 → d e f

⋮

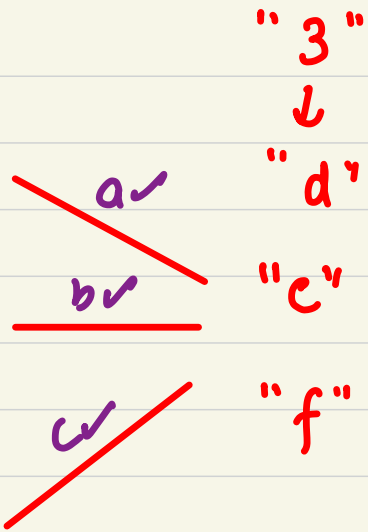
9 → w x y z

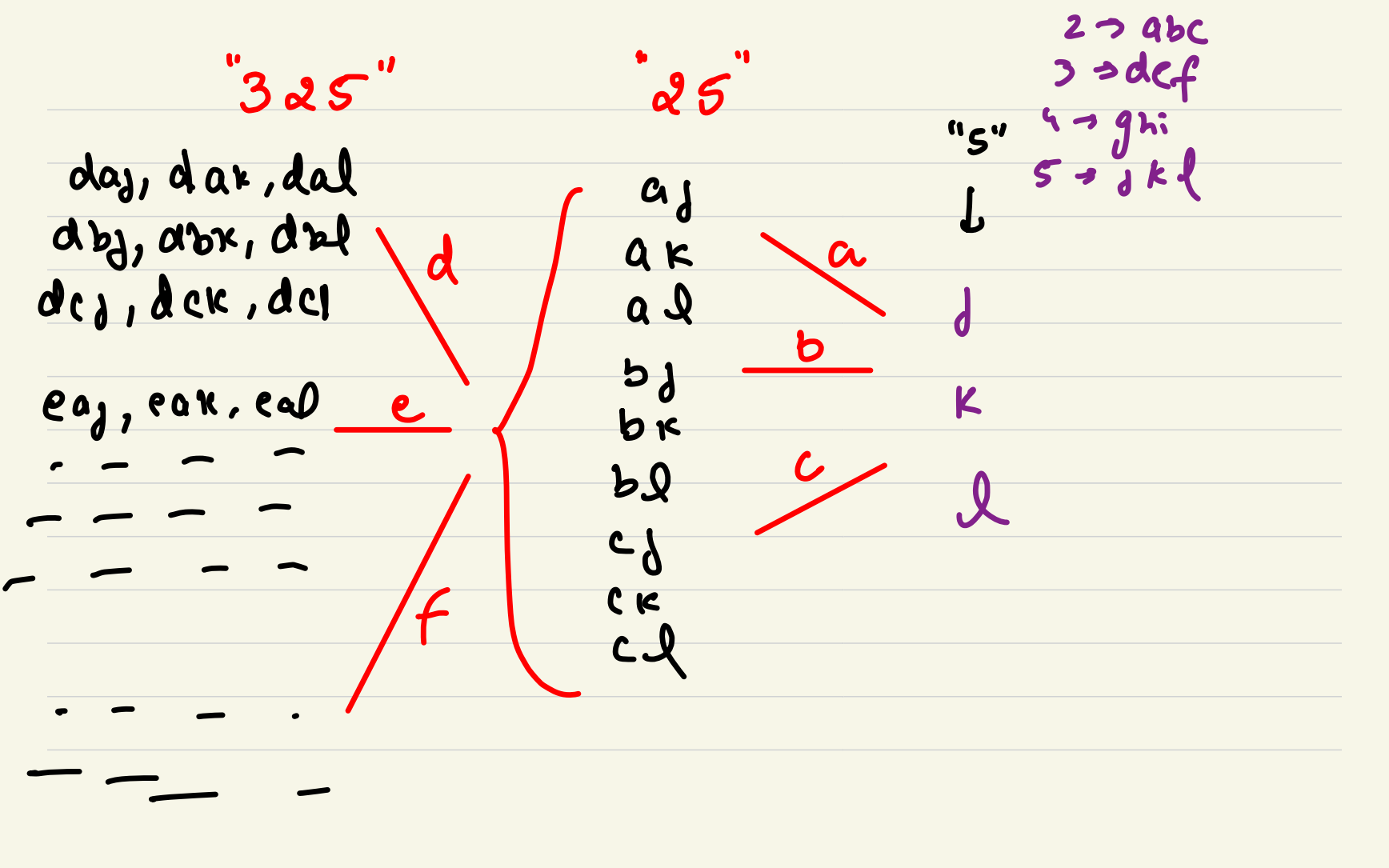
i/p → "23"

1 2 S 6

"23"

ad
ae
af
bd
be
bf
cd
ce
cf




"325"
"25"
"5"
2 → abc
3 → def
4 → ghi
5 → jkl
daj, dak, dal
dbj, dbk, dbl
dcj, dck, dcl
eaj, eak, eal
d
e
f
aj
ak
al
bj
bk
bl
cj
ck
cl
a
b
c
j
k
l

$f(str, i, result) = f(str, i+1, result + map[str[i]][k])$

$\forall k \in [0, map[str[i]].len-1]$

print all the combinations

for the given string str,

from index i to n-1

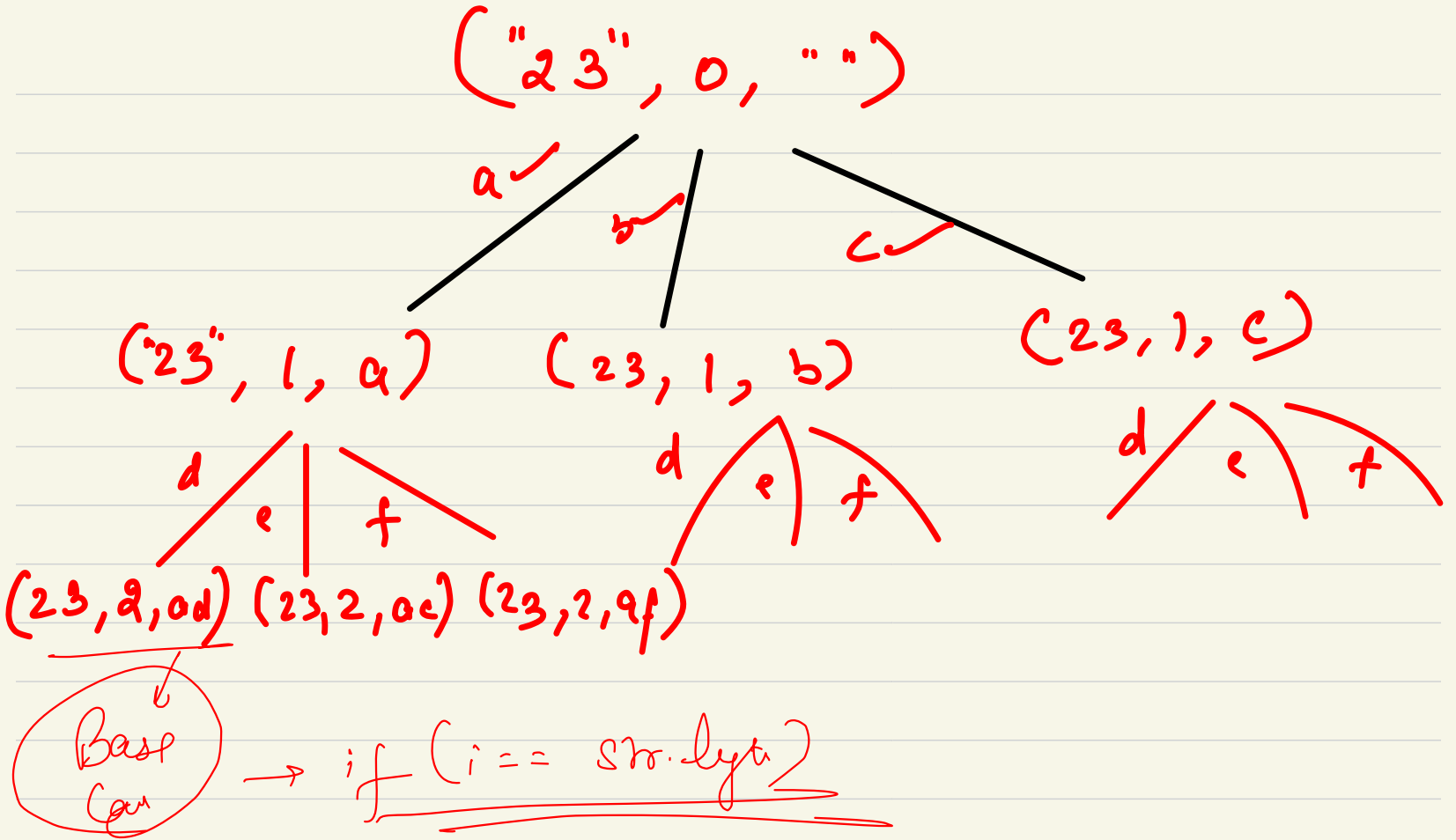
for (k=0; k < len; k++)

i:=0

map = { 2 : [a, b, c]
3 : [d, e, f]
:
9 : [w, x, y, z] }

→ map[str[i]]

array of chars


("23", 0, "")
a
b
c
("23", 1, a)
(23, 1, b)
(23, 1, c)
d
e
f
d
e
f
d
e
f
(23,2,ad)
(23,2,ae)
(23,2,af)
Base Case
→ if (i == str.length)