Lecture 11, 15 September 2026

Madhavan Mukund

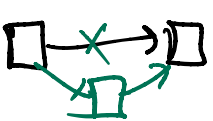
https://www.cmi.ac.in/~madhavan

Programming and Data Structures with Python

Arrays, Lists, Dictionaries

Arrays : Contiguous storage, "random" access
Insertion/deletion is expensive

Lists : Flexible storage, spend time prop to i to access $l[i]$, Insertion/deletion is constant time, "re-plumbing"



Python "lists" are arrays

Extra space - double the allocation & copy the elements

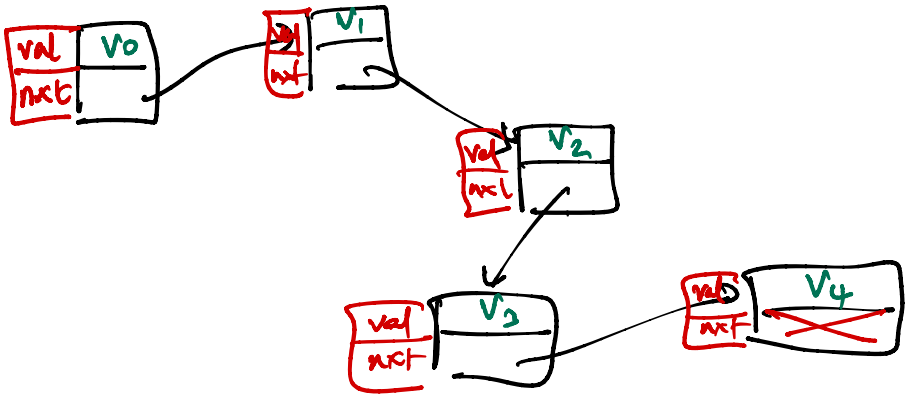
"Amortized" cost analysis

Dictionaries

key k $\rightsquigarrow$ hash $h(k)$ $\rightsquigarrow$ location

- Inserts are constant time (assuming no collisions!)

Lists



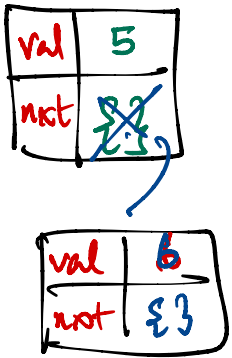
Dictionary implementation

Empty list: dl = {}

Insert 5

dl["val"] = 5

dl["nxt"] = {}

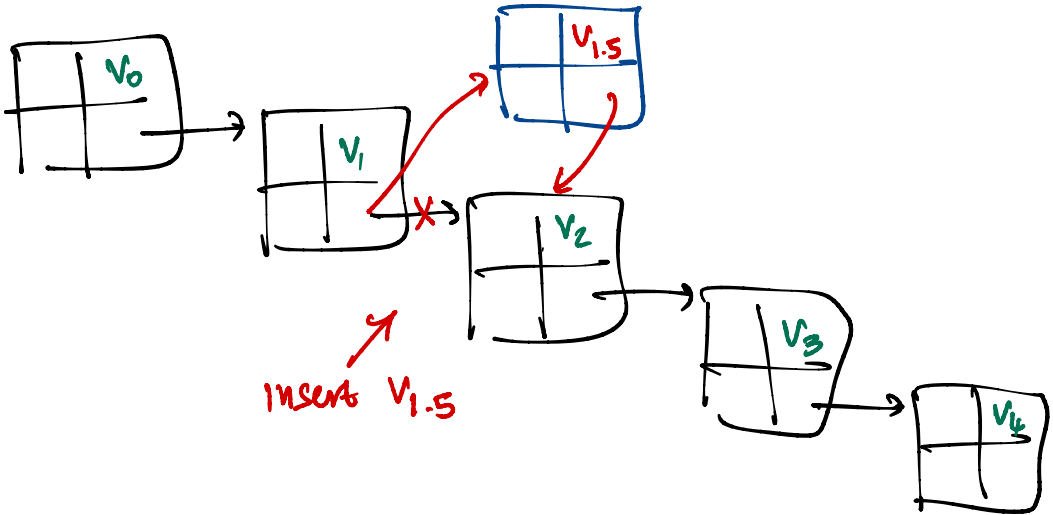


Append 7

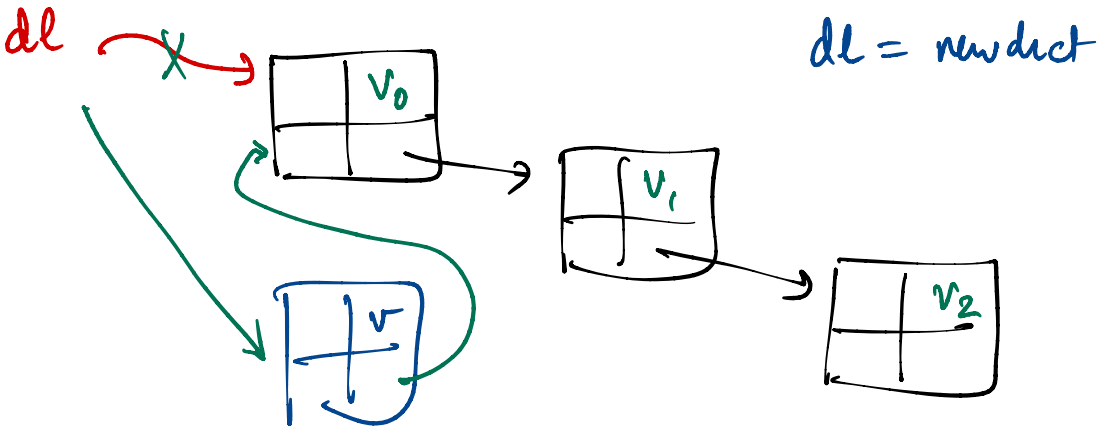
dl["next"]["val"] = 5

dl["nxt"]["nxt"] = {}

Insert



l.insert(0, v)

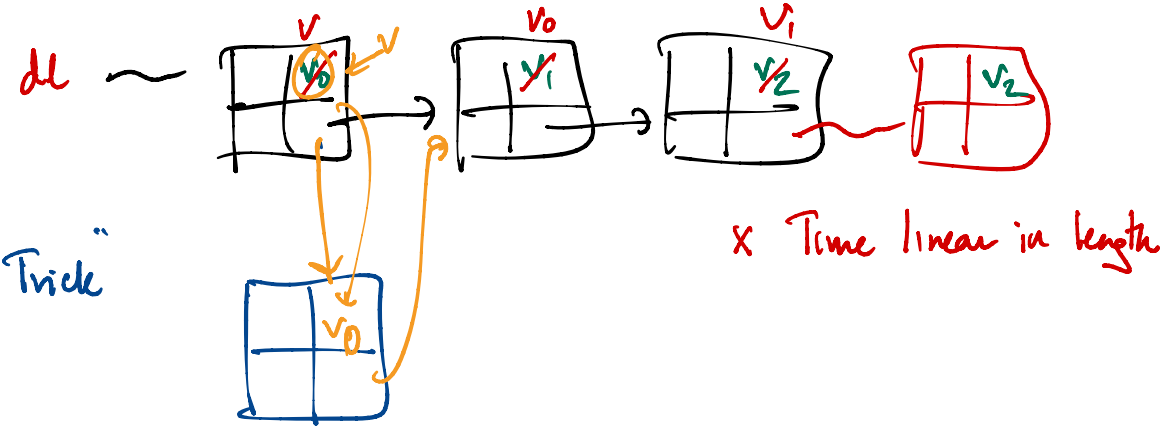


recall

```
def badconcat(l1,l2):
    l1 = l1+l2
    return
```

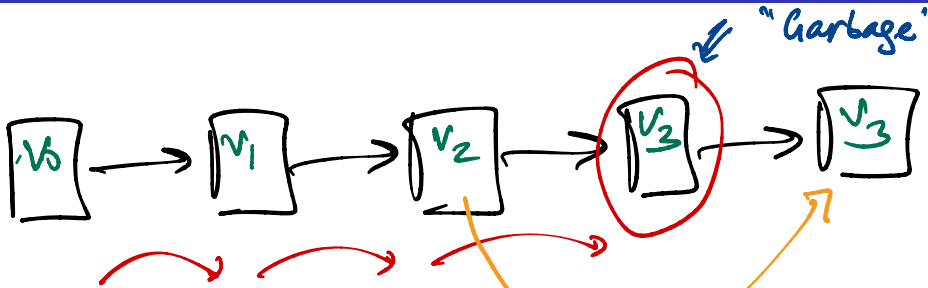
(vs l1.extend(l2))

new local l1


dl ~
v
v0
v
v0
v1
v1
v2
v2
v0
"Trick"
x Time linear in length

delete

Del. v_3



look ahead — if next value is v_3, bypass

Inductive definitions

Proof by induction

Prove for $n = 0$ "Base Case"

If it holds for n, it holds for $n+1$ "Induction step"

$$\sum_{i=0}^{n} i = \frac{n(n+1)}{2} + n+1$$

Factorial

Base case: $0! = 1$

$(n+1)!$: $(n!) \times (n+1)$

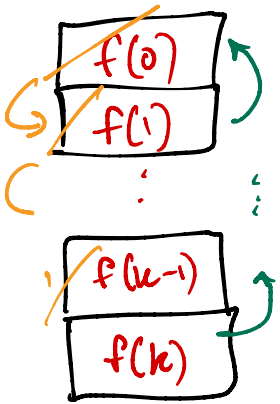
"Compute" $f(k)$, where f is defined inductively.

Base case? $k = 0$ → return $f(0)$

$k > 0$ → compute $f(k-1)$

"combine" $f(k-1)$, k to get $f(k)$

f(k) → compute f(k-1) → compute f(k-2)



f(0)

"Recursion"

Data structures are also inductive

List is

either empty

or a value followed by a list

Append v to l

if l is empty

    "create" a dictionary with v

otherwise

    append v to l["next"]