Design & Analysis of Algorithms
COMP 482 / ELEC 420

John Greiner

Python Dictionaries

```
d = {"snow" : 7, "apple" : 3, "white" : 20}
d["white"] = 30
d["prince"] = 16
d["apple"]
```

Same idea as Java/C++ hash map, C# dictionary, Perl hash, …

How is this "magic" implemented?

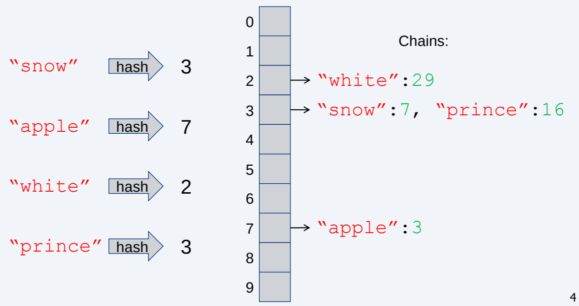
Combination of Ideas

- Hash table & hash function
- Dynamic table & amortization

To do:
[CLRS] 11,17
#4

Hash Tables & Hash Functions

```
d = {"snow" : 7, "apple" : 3, "white" : 20}
d["white"] = 29
d["prince"] = 16
d["apple"]
```



Access Time Depends on Chain Length

? What property do we want of our hash function? ?

? How long is each chain? ?

? How much time per access? ?

Creating a Good Hash Function is Difficult

Generally, just use those in libraries.

```
class string:
   def __hash__(self):
      if not self:
         return 0 # empty
      value = ord(self[0]) << 7
      for char in self:
         value = c_mul(1000003, value) ^ ord(char)
      value = value ^ len(self)
      if value == -1: # reserved error code
         value = -2
      return value
```

`key.__hash__() % table_size`

Dynamic Table Motivation

Typically, don't know how much data we'll have.

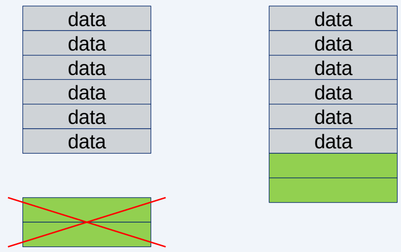
- Want underlying hash table to grow, so average chain size is bounded.
- Want to retain constant-time indexing.

Focus on the latter goal first.

- We'll have to do a little extra to combine hash tables & dynamic tables.

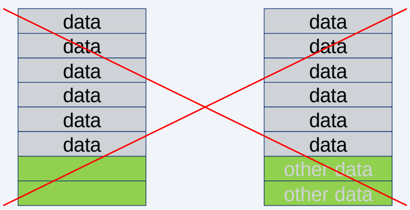
Adding Data when Dynamic Table is Full

Must be contiguous for constant-time indexing.



Adding Data when Dynamic Table is Full

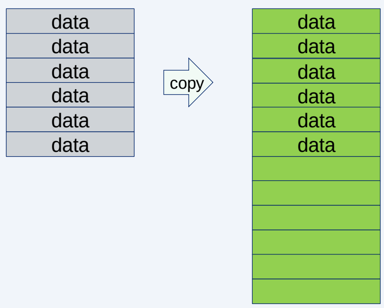
What's wrong with just using space at end of array?



That memory might already be in use.

Adding Data when Dynamic Table is Full

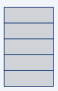
So, grab needed space elsewhere & copy everything.



Double the space.

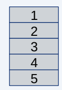
Cost of a Series of Operations

Initially: Table size = 5, table empty



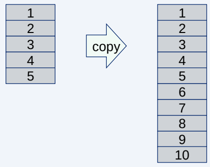
Cost of a Series of Operations

Add 5 data items, cost = 5



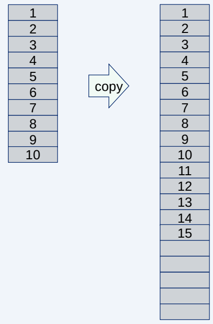
Cost of a Series of Operations

Add 5 more data items, cost = 10



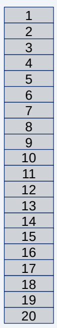
Cost of a Series of Operations

Add 5 more data items, cost = 15



Cost of a Series of Operations

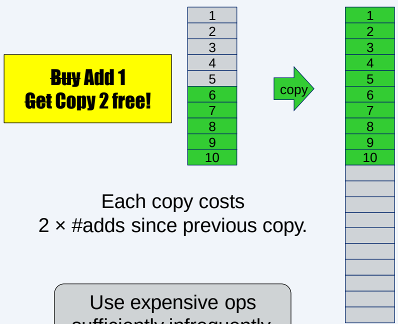
Add 5 more data items, cost = 5



<u>Total costs:</u>

Copying = 15
Adding = <u>20</u>
Total = 35

Cost of Copying is Proportional to Adding



Each copy costs
2 × #adds since previous copy.

Use expensive ops
sufficiently infrequently.

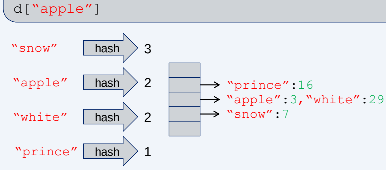
Amortized Cost

Cost of series of n operations = m
→
Each operation has *amortized cost* = m/n

Dynamic tables: O(1) amortized time to add data

Dynamic Hash Tables

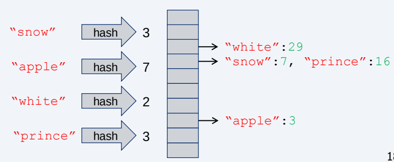
```
d = {"snow" : 7, "apple" : 3, "white" : 20}
d["white"] = 29
d["prince"] = 16
d["apple"]
```



Hash table "full enough".

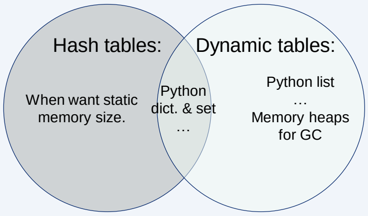
Load factor = #items/size

Must rehash all data.



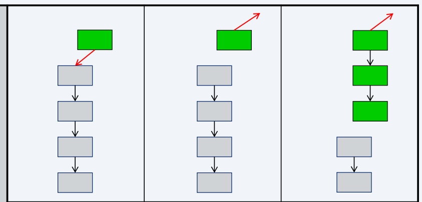
Hash Table & Dynamic Table Odds & Ends

Hash table size: prime or power-of-2?
Chaining vs. open addressing
Dynamic table expansion factor
Dynamic table contraction
Python's dictionary/hash table implementation



19

Multipop

3 ops:			
	Push(S,x)	Pop(S)	Multi-pop(S,k)
Worst-case cost:	O(1)	O(1)	O(min(|S|,k) = O(n)

Amortized cost: ?

20

Incrementing Binary Counter

Counter	Bits changed in increment
0	1
1	2
10	1
11	3
100	1
101	2
110	1
111	4
1000	1
1001	2
1010	1
1011	3
1100	1
1101	2
1110	1
1111	5

Another way to sum the costs?

Bit position	# times bit changes
0	16
1	8
2	4
3	2

$$\le \sum_{i=0}^{\lfloor \lg n \rfloor} \left\lfloor \frac{n}{2^i} \right\rfloor = O(?)$$

21

Amortized Analysis Approaches

Aggregate:

1. For all sequences of m ops, find maximum sum of actual costs.
   Potentially difficult to know actual costs or bound well.
2. Result is sum/m.

Sufficient for many commonly-used examples.

22

Amortized Analysis Approaches

Accounting:

1. Compute actual costs c_i of each kind of op.
2. Define accounting costs $\hat{c}_i$ of each kind of op, such that can assign credits $\hat{c}_i$-c_i consistently to data elts.
   Can "overpay" on some ops, and use credits to "underpay" on other ops later.
   Poor definitions lead to loose bounds.
3. O(max acct. cost.)

23

Amortized Analysis Approaches

Potential:

1. Define potential function Φ(D), such that $\Phi(D_i) \geq \Phi(D_0)$.
   Essentially assigns "credits" to data structure, rather than operations.
   Poor definition leads to loose bounds.
2. Calculate accounting costs $\hat{c}_i = c_i + \Delta\Phi$ of each kind of op.
3. O(max acct. cost.)

Complicated approach necessary for more complicated data structures.

24