CS216: Program and Data Representation
University of Virginia Computer Science
Spring 2006 David Evans

Lecture 23: Review / Fast Dictionaries

http://www.cs.virginia.edu/cs216

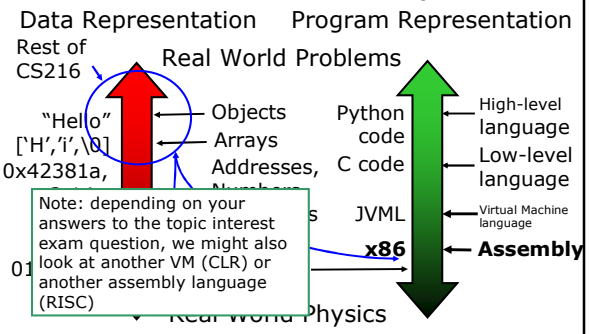
Announcements

- PS7 Comments will be posted later today
- Exam 2 will be posted Thursday after 5pm
- Office hours: Today: 2-3pm; Tomorrow: 10-11am
- After Thursday, I will start charging storage fees on uncollected graded assignments:
  - Exam 1: 1 point per page per day
  - Problem Sets: 1 star color per week

Exam 2 Review Questions

- JVML: Do istore_1 and astore_1 share the same memory location?
- Memory management: Explain memory leaks
- Complexity classes: What is NP-Complete?

CS216 Roadmap



Fast Dictionaries

- Problem set 2, question 5...

  "You may assume Python's dictionary type provides lookup and insert operations that have running times in $O(1)$."

- Class 6: fastest possible search using binary comparator is $O(\log n)$

  Can Python really have an $O(1)$ lookup?

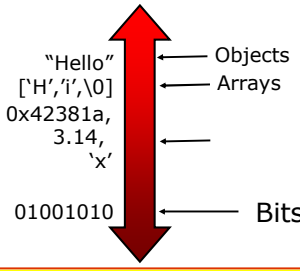
Fast Dictionaries

Data Representation

- If the keys can be anything?

No – best one comparison can do is eliminate ½ the elements

The keys must be bits, so we can do better!



Lookup Table

Key	Value
000000	"red"
000001	"orange"
000010	"blue"
000011	null
000100	"green"
000101	"white"
...	...

Works great...unless the key space is sparse.

Sparse Lookup Table

- Keys: names (words of up to 40 7-bit ASCII characters)
- How big a table do we need?

40 * 7 = 280
2^{280} = ~$1.9*10^{84}$ entries

We need lookup tables where many keys can map to the same entry

Hash Table

- Hash Function:
  h: $Key \rightarrow [0, m\text{-}1]$

Here:
$h = \text{firstLetter}(Key)$

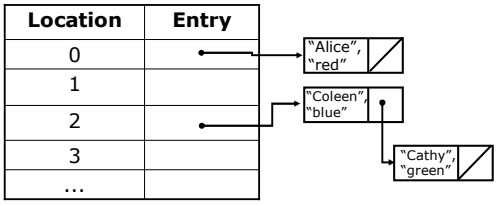
Location	Key	Value
0	"Alice"	"red"
1	"Bob"	"orange"
2	"Coleen"	"blue"
3	null	null
4	"Eve"	"green"
5	"Fred"	"white"
...	...	...
m-1	"Zeus"	"purple"

Collisions

- What if we need both "Colleen" and "Cathy" keys?

Separate Chaining

- Each element in hash table is not a <key, value> pair, but a list of pairs



Hash Table Analysis

- Lookup Running Time?

Worst Case: $\Theta(N)$
N entries, all in same bucket
Hopeful Case: $O(1)$
Most buckets with < c entries

Requirements for "Hopeful" Case

- Function h is well distributed for key space

  for a randomly selected $k \in K$,

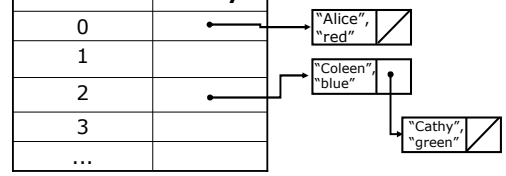
  probability $(h(k) = i) = 1/m$

- Size of table (m) scales linearly with N
  - Expected bucket size is $\Theta(N/m)$

Finding a good h can be tough (more next class)

Saving Memory

Location	Entry
0	
1	
2	
3	
…	



Can we avoid the overhead of all those linked lists?

Linear Open Addressing

Location	Key	Value
0	"Alice"	"red"
1	"Bob"	"orange"
2	"Coleen"	"blue"
3	"Cathy"	"yellow"
4	"Eve"	"green"
5	"Fred"	"white"
6	"Dave"	"red"
…		

Sequential Open Addressing

```
def lookup (T, k):
  i = hash (k)
  while (not looped all the way around):
    if T[i] == null:
      return null
    else if T[i].key == k:
      return T[i].value
    else
      i = i + 1 mod T.length
```


Problems with Sequential

- "Primary Clustering"
  - Once there is a full chunk of the table, anything hash in that chunk makes it grow
  - Note that this happens even if h is well distributed
- Improved strategy?

  Don't look for slots sequentially

  i = i + s mod T.length

Doesn't help – just makes clusters appear scattered

Double Hashing

- Use a second hash function to look for slots

  i = i + hash2 (K) mod T.length
- Desirable properties of hash2:
  - Should eventually try all slots

    result of hash2(K) should be relatively prime to m

    (Easiest to make m prime)
  - Should be independent from hash

Charge (Announcements)

- PS7 Comments will be posted later today
- Exam 2 will be posted Thursday after 5pm
- Office hours: Today: 2-3pm; Tomorrow: 10-11am
- After Thursday, I will start charging storage fees on uncollected graded assignments:
  - Exam 1: 1 point per page per day
  - Problem Sets: 1 star color per week