Additional notes on hash tables

Open Addressing with Linear Probing

Handout for F15:Homework:H09

aka: Open-Address Hashing, Closed Hashing. Suppose we are hashing Student objects consisting of name and a two digit perm, using `H(s)=s.perm % 10` as our hash function. Here is how the table would evolve after a series of results where we use open hashing with linear probing.

Insert (Alice,34)

[0]	
[1]	
[2]	
[3]	
[4]	**(Alice, 34)**
[5]	
[6]	
[7]	
[8]	
[9]	

Insert (Bob, 44)

[0]	
[1]	
[2]	
[3]	
[4]	(Alice, 34)
[5]	**(Bob, 44)**
[6]	
[7]	
[8]	
[9]	

Insert (Chris,29)

[0]	
[1]	
[2]	
[3]	
[4]	(Alice, 34)
[5]	(Bob, 44)
[6]	
[7]	
[8]	
[9]	**(Chris, 29)**

Insert (Dani, 25)

[0]	
[1]	
[2]	
[3]	
[4]	(Alice, 34)
[5]	(Bob, 44)
[6]	**(Dani, 25)**
[7]	
[8]	
[9]	(Chris, 29)

Insert (Erica, 39)

[0]	**(Erica,39)**
[1]	
[2]	
[3]	
[4]	(Alice, 34)
[5]	(Bob, 44)
[6]	(Dani, 25)
[7]	
[8]	
[9]	(Chris, 29)

Insert (Fred, 72)

[0]	(Erica,39)
[1]	
[2]	**(Fred, 72)**
[3]	
[4]	(Alice, 34)
[5]	(Bob, 44)
[6]	(Dani, 25)
[7]	
[8]	
[9]	(Chris, 29)

More examples

These examples are the type you'd get on a hwk or exam. To save space, we show only the perm numbers, and we show only the FINAL hash table after a sequence of inserts for: (Alex, 3), (Blair, 14), (Chris, 15), (Dakota, 92), (Emory, 65), (Fawn, 35). Note the difference from the different hash functions. As an aside, a quick way to compute the mod of a sequence of numbers is to use Python's lambda function and map feature, as shown at right.

```
>>> perms=[3,14,15,92,65,35]
>>> map(lambda x:x%10, perms)
[3, 4, 5, 2, 5, 5]
>>> map(lambda x:x%9, perms)
[3, 5, 6, 2, 2, 8]
>>> map(lambda x:x%11, perms)
[3, 3, 4, 4, 10, 2]
>>> def f(x):
...  return ((10*x) + x%2)%9
...
>>> map(f,perms)
[4, 5, 7, 2, 3, 0]
>>>
```

H(s)=s.perm%10

[0]	
[1]	
[2]	92
[3]	3
[4]	14
[5]	15
[6]	65
[7]	35
[8]	
[9]	

H(s)=s.perm%9

[0]	
[1]	
[2]	92
[3]	3
[4]	65
[5]	14
[6]	15
[7]	
[8]	35

H(s)=s.perm%11

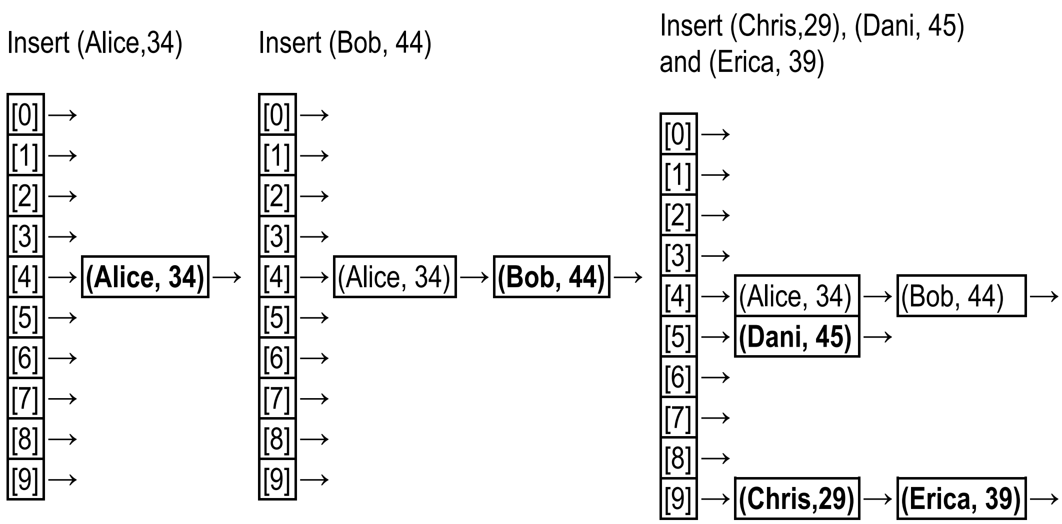
[0]	
[1]	
[2]	35
[3]	3
[4]	14
[5]	15
[6]	92
[7]	
[8]	
[9]	
[10]	65

f(x)=(10*x + x%2)%9
H(s)=f(s.perm)

[0]	35
[1]	
[2]	92
[3]	65
[4]	3
[5]	14
[6]	
[7]	15
[8]	

Chained Hashing

aka Separate Chaining, Open Hashing, Hashing with Closed Addressing. Now suppose we are, as before, hashing Student objects consisting of name and a two digit perm, using `H(s)=s.perm % 10` as our hash function—but this time using separate chaining with a linked list at each array element. Here is how the table would evolve after a series of insert operations.



More Separate Chaining examples

These examples are the type you'd get on a hwk or exam. To save space, we show only the perm numbers, and we show only the FINAL hash table after a sequence of inserts for: (Alex, 3), (Blair, 14), (Chris, 15), (Dakota, 92), (Emory, 65), (Fawn, 35). Note the difference from the different hash functions. As an aside, a quick way to compute the mod of a sequence of numbers is to use Python's lambda function and map feature, as shown at right.

```
>>> perms=[3,14,15,92,65,35]
>>> map(lambda x:x%10, perms)
[3, 4, 5, 2, 5, 5]
>>> map(lambda x:x%9, perms)
[3, 5, 6, 2, 2, 8]
>>> def f(x):
...  return ((10*x) + x%2)%9
...
>>> map(f,perms)
[4, 5, 7, 2, 3, 0]
>>>
```