CSE 326
Lecture 21

Spring Quarter 1995

Rehashing

- Rehashing means
  1. initializing a new hash table
  2. Scanning the old hash table and reinserting into the new hash table.
- When to rehash
  - an insertion fails in closed hashing
  - too many probes are occuring in closed hashing
  - load factor reaches some λ too close to 1 in closed hashing.
  - load factor reaches 1 in open hashing.
  - too many items marked deleted.
- Although rehashing is costly at the time it is done, it is not expensive in the long run. That is the amortized cost of the operations is still a constant.

Rehashing Example

- Open hashing - $h_1(x) = x \bmod 5$ rehashes to $h_2(x) = x \bmod 11$

$\lambda=1$

0	1	2	3	4
25		37	83	
		52	98	

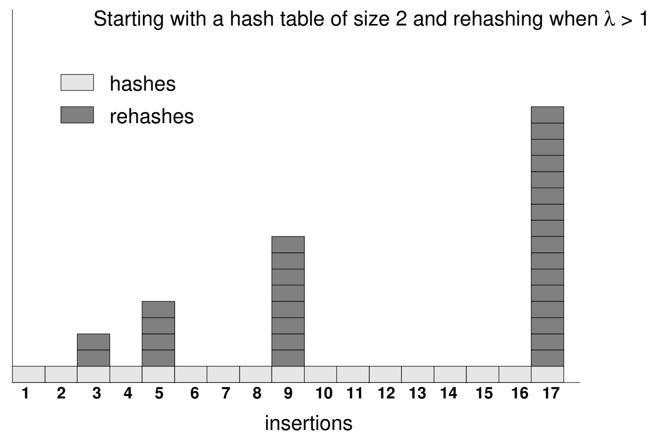
$\lambda=5/11$

0	1	2	3	4	5	6	7	8	9	10
			25	37		83		52		98

Rehashing Example

- Let us look only at the case of insertions and finds in open hashing.
- Start with a fixed size hash table of size s_0.
- If an insertion causes the load factor to exceed 1 then approximately double the size of the hash table. If the table is size s_i then its new size is $s_{i+1} = 2\, s_i$ approximately.
- Time analysis:
  1. If a find or insertion (without rehashing) is done the time is 1.
  2. If an insertion is done and rehashing is done for the i-th time then the time is $s_i = 2^{i-1}\, s_0$ approximately.

Rehashing Picture



Amortized Analysis

- Let us count hash function calculations for doing n insertions where the intital table is size 2. When $\lambda > 1$ hash table size is doubled.

Insertion	Per Oper	Cum
1-2	1	2
3	3	5
4	1	6
5	5	11
6-8	1	14
9	9	23
10-16	1	30
17	17	47
18-32	1	62
33	33	95
34-64	1	

Analysis

- $2^k+1 \le n \le 2^{k+1}$

  Hashes = n

  Rehashes = $2^k + 2^{k-1} + ... + 2 = 2^{k+1} - 2$

  Total = $n + 2^{k+1} - 2 < 3n$
- Example:

  n = 33 Total = 33 + 64 - 2 = 95

  n = 60 Total = 60 + 64 - 2 = 122

Tic-Tac-Toe

- In our tic-tac-toe project notice that the number of valid game boards is smaller than 3^9, the number of possible game boards. Your implementation uses an array of size 3^9 = 19,683.
- Question: Is using an array that large the right way to implement the storage of best moves?
- How many valid game boards are there?

$$1 + 9 + 9 \cdot 8 + \frac{9 \cdot 8}{2} \cdot 7 + \frac{9 \cdot 8}{2} \cdot \frac{7 \cdot 6}{2} + \frac{9 \cdot 8 \cdot 7}{3 \cdot 2} \cdot \frac{7 \cdot 6}{2} + \frac{9 \cdot 8 \cdot 7}{3 \cdot 2} \cdot \frac{6 \cdot 5 \cdot 4}{3 \cdot 2}$$

$$\frac{9 \cdot 8 \cdot 7 \cdot 6}{4 \cdot 3 \cdot 2} \cdot \frac{5 \cdot 4 \cdot 3}{3 \cdot 2} + \frac{9 \cdot 8 \cdot 7 \cdot 6}{4 \cdot 3 \cdot 2} \cdot \frac{5 \cdot 4 \cdot 3 \cdot 2}{4 \cdot 3 \cdot 2} + \frac{9 \cdot 8 \cdot 7 \cdot 6 \cdot 5}{5 \cdot 4 \cdot 3 \cdot 2} = 6,550$$

- The table is about 1/3 full, and 2/3 is wasted.

What are the Options for Storing and Using the Table?

- Leave the table as is.

  ++ fast look up, no comparisons per look up
  - Storage, 19,683 entries or 19,683 pointers and 6,550 entries
- Sort and store valid boards in an array with 6,550 entries

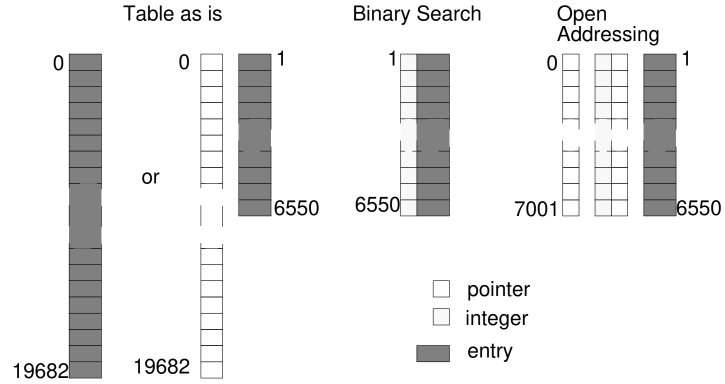
  - Slower look up, binary search , $\log_2 6{,}550 = 13$ comparisons per look up
  + Storage 6,550 integers and 6,550 entries
- Hash table - open hashing. Use $h(x) = x \bmod 7{,}001$ for $\lambda = 1$.

  + On average about $1 + \lambda/2 = 1.5$ comparisons per look up.
  - Storage 6,550 entries, 6,550 integers, and 6,550 + 7,001 pointers.
- Hash table - closed hashing using linear probing. Use $h(x) = x \bmod 13{,}103$ (prime?) for $\lambda = 1/2$.

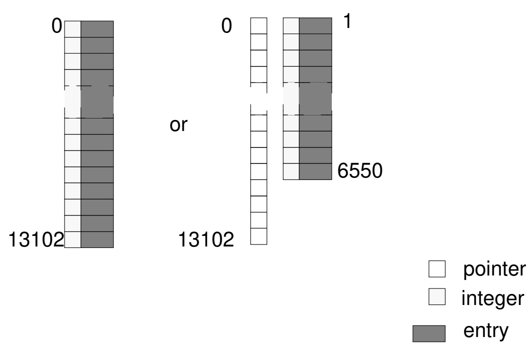
  + On average about $1/2(1 + 1/(1-\lambda)) = 1.5$ comparisons per look up.

  - Storage for 13,103 integers and 13,103 entries or (13,103 pointers and 6,550 entries and 6,650 integers)

Storage Layouts for Data Structures



Closed Hashing with Linear Probing



Which Data Structure Would You Pick?

- Table as is: 16
- Binary Search: 2
- Open Hashing (chaining): 2
- Closed Hashing with Linear Probing: 6

- 16/26 chose table as is which is a good choice. All choices are basically ok. In most cases table as it will not be the best because the key space is too large.