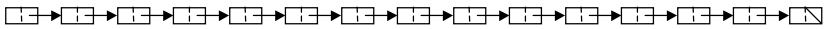
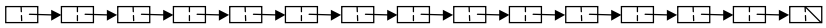


CSE 373: Hash Tables

Chapter 5

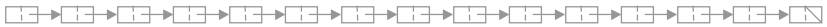


Motivation

Goal: The ability to store and retrieve information in O(1) time

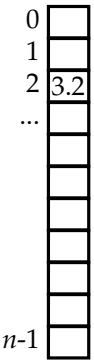
Current Solutions:

Hash Table Goal



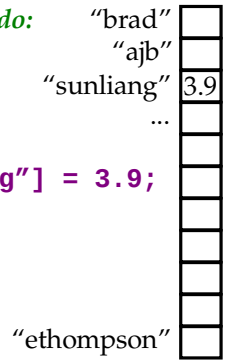
We can already do:

```
a[2] = 3.2;
```

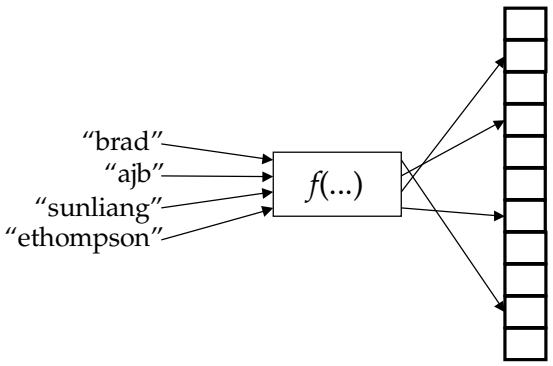


We'd like to do:

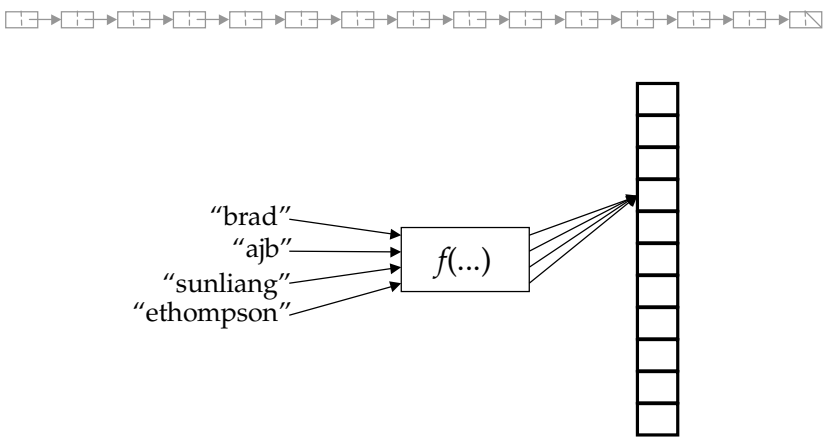
```
a["sunliang"] = 3.9;
```



Hash Table Approach



Hashing Conflicts



Hashing Integers

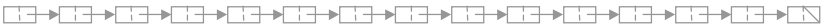
What's a simple scheme for hashing integers?

```
int HashInteger(int key,int tablesize) {


}
```

- advantages?
- disadvantages?

General Strategies



- Selecting a good hash function often depends on the set of possible keys
- Using a hash table whose size is a prime number tends to help reduce conflicts

Hashing Strings

General approach:

- convert string to integer
- "mod" integer by table size

Naive approach:

```
int HashString(char* key,int tablesize) {
  int total=0;
  while (*key) {
    total += *key;
  }
  return total%tablesize;
}
```

Problem with Naive Approach

```
HashString("bat",n)
= HashString("tab",n);
= HashString("rad",n);
```

Total Number of Possibilities ≈ 127×12 = 1524
Useful Number of Possibilities ≈ 26×12 = 312

Probably not good if hash table size or number of keys is greater than this…

Improved Approach

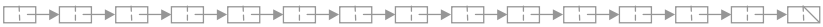
Read string as base 27 number:

1	27	729	
b	a	t	= 2×1 + 1×27 + 20×729 = 14,609
t	a	b	= 20×1 + 1×27 + 2×729 = 1,505
r	a	d	= 18×1 + 1×27 + 4×729 = 2,961

Advantages?

Disadvantages?

Other Ideas



Hash using only a subset of the characters…

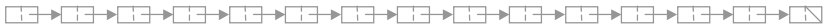
- first three?
- last three?
- middle three?
- first, middle, last?
- etc.

Hash Function Design Goals

- Hash to all slots in your table
- Avoid collisions
- Hash as evenly as possible
- Hash as quickly as possible

(Again, note that much of this may depend on the set of possible keys…)

Harsh Hash Reality



- No matter how good your hash function is, collisions will probably occur
- Thus, we also need a collision resolution strategy…
  - separate chaining
  - resizable hash table
  - open addressing

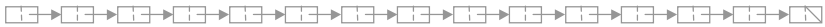
Hash Table Operations

- Main Operations:

```
void insert(HashedObj& key);
HashedObj& find(HashedObj& key);
void remove(HashedObj& key);
```

- Normal Creation/Deletion Operations
- No iteration, `FindMin()`/`Max()`, etc.
  (why?)

Alternate Hash Table Operations



- Main Operations:

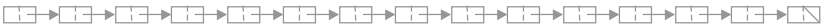
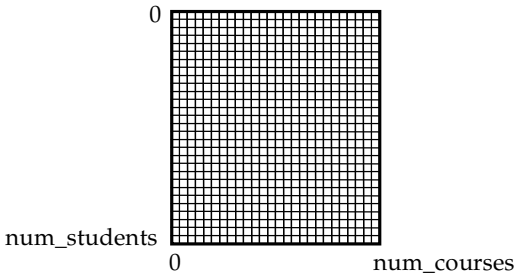
```
void insert(HashedObj& key,Object& data);
Object& find(HashedObj& key);
void remove(HashedObj& key);
```

- Similar to the normal interface, but associates some data with each key

Operator Analysis

	Hash Table	*List*	*BST*
problem size			
space			
`insert()`			
`find()`			
`remove()`			
other			

Flashback to Day 1



```
const int num_courses = 7000;
const int num_students = 33000;

typedef int registry[num_students][num_courses];
```