Hash Tables and Elementary Data Structures

Chapters 10 and 11

Elementary Data Structures

- What structures did we cover in CS300?
- What operations are performed on these structures?

Languages without Pointers

- How can we implement pointers, as in linked lists, in languages that do not have pointers?

Multiple Array Representation

Single Array Representation

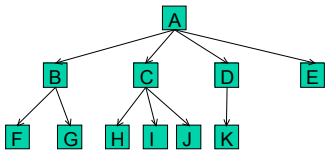
Representing Rooted Trees

- p
- left
- right
- If p[x] = Nil then x is the root
- If node x has no children then:
  - left[x] = Nil and right[x] = Nil
- Root of tree T is root[T]
- If root[T] is Nil then the tree is empty

Rooted Trees with Unbounded Branching

- What if a node in a tree can have up to
  - 3 children?
  - 4 children?
  - 5 children? Etc.
- When the number of children is unbounded we have no idea how many child nodes to create
- Can we use binary trees to represent unbounded trees?

Rooted Trees with Unbounded Branching



Bit Manipulation

```
#include <iostream>

using namespace std;

typedef unsigned long Bits;

int main()
{
  Bits val1, val2, val, result;
  cout.setf(ios::showbase);
  cout.setf(ios::hex, ios::basefield);

  cout << "Enter bit pattern to be complemented: ";
  cin >> val;
  result = ~val;
  cout << "Val: " << val << ", Not val: " << result << endl;
```

And

```
cout << "Enter two bit patterns to be Anded: "
     << endl;
cout << "Val1 :";
cin >> val1;
cout << "Val2 :"; cin >> val2;
result = val1 & val2;
cout << "Val1 : " << val1 << ", Val2 : "
     << val2 << ", And gives " << result
     << endl;
```


Or

```
cout << "Enter two bit patterns to be Ored: "
        << endl;
cout << "Val1 :";
cin >> val1;
cout << "Val2 :";
cin >> val2;
result = val1 | val2;
cout << "Val1 : " << val1 << ", Val2 : "
     << val2 << ", Or gives " << result
     << endl;
```


XOR

```
cout << "Enter two bit patterns to be Xored: "
     << endl;
cout << "Val1 :";
cin >> val1;
cout << "Val2 :";
cin >> val2;
result = val1 ^ val2;
cout << "Val1 : " << val1 << ", Val2 : "
     << val2 << ", Xor gives " << result
     << endl;
```

Output

```
C:\WINDOWS\system32\cmd.exe
Enter bit pattern to be complemented: 6
Val: 0x6, Not val: 0xfffffff9
Enter two bit patterns to be Anded :
Val1 :3
Val2 :5
Val1 : 0x3, Val2 : 0x5, And gives 0x1
Enter two bit patterns to be Ored :
Val1 :3
Val2 :5
Val1 : 0x3, Val2 : 0x5, Or gives 0x7
Enter two bit patterns to be Xored :
Val1 :3
Val2 :5
Val1 : 0x3, Val2 : 0x5, Xor gives 0x6
Press any key to continue . . .
```


Bit Shifting

```
Bits val, result;
int i;
cout.setf(ios::showbase);
cout.setf(ios::hex,ios::basefield);
cout << "Enter val ";
cin >> val;
cout << "Enter shift amount ";
cin >> std::hex >> i;
result = val << i;
cout << "Left shifting gives " << result << endl;
result = val >> i;
cout << "Right shifting gives " << result << endl;
```


Bit Shifting

```
C:\WINDOWS\system32\cmd.exe
Enter val 23
Enter shift amount 2
Left shifting gives 0x8c
Right shifting gives 0x8
Press any key to continue . . .
```

Hashing

- It is common to need to generate a “key value” that summarizes or characterizes a complex data type
- This is called hashing and is something of an art form

Hashing a String

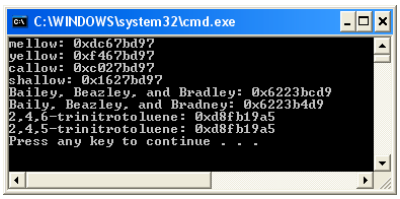
- The key should depend on every character in the string
- The hash key is going to be a mix-up of the bits of the characters in the string
- XOR is good because it depends on both inputs
- The pattern is going to combine bits from all of the characters by a loop that xors the next character into a key, then moves the key to the left a little to fill up a long int

Hashing a String

- When you move the key left, some bits fall off the end. These are the bits that encode the first few characters in the string.
- With a long string, the result could end up depending only on the last few characters in the string.
- How can we avoid the problem?
  - Save the bits that fall off and feed them back in on the right, xoring them with the new char data

Example of Problem

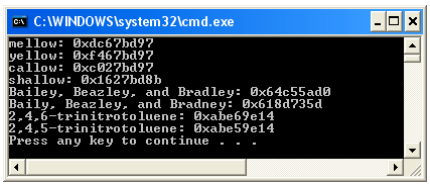
- Here's what happens when the bits fall off the end



Hash String Function

```
Bits HashString(const char str[])
{
  Bits Result = 0;
  int n = strlen(str);
  Bits Top5Bits = 0xf8000000;
  Bits Carry = 0x0;
  const int kleftmove = 5;
  const int krightmove = 27;
  for(int i = 0; i < n; i++)
  {
    Carry = Result & Top5Bits;
    Carry = Carry >> krightmove;
    Result = Result << kleftmove;
    Result ^= Carry;
    Result ^= str[i];
  }
  return Result;
}
```


Working Hash String



Hash String

- Would it be possible to have two different strings hash to the same key?
- What can you use the hash key for?

Simple Hash Table

- Use modulo arithmetic
  - How? Why?
- This could cause hash collisions

Handling Collisions

- Techniques for handling collisions include:
  - Chaining
  - Linear Probing

$$h' : U \rightarrow \{0,1,\ldots,m-1\}.$$
$$h(k,i) = (h'(k) + i) \bmod m$$
$$for \quad i = 0,1,\ldots,m-1$$

  - Quadratic Probing

$$h(k,i) = (h'(k) + c_1 i + c_2 i^2) \bmod m$$

Handling Collisions

- Double Hashing
  - One of the best methods for open addressing

    $$h(k,i) = (h_1(k) + ih_2(k)) \bmod m$$

  - Example:

    $$h_1(k) = k \bmod 13$$
    $$h_2(k) = 1 + (k \bmod 11)$$

    - Where will the key 14 be inserted

Index	Key
0	
1	79
2	
3	
4	69
5	98
6	
7	72
8	
9	
10	
11	50
12	