Bloom filters
h1(x) = r
h1(y) = r-1
f1 x = 1
if 0 it definitely doesn't exist, if 1 it might exist.
h2(x) = s
f2 s = 1
74000 words = 40killobytes*10 = 400 KB

unsigned char T[max]; 8 bits in an unsigned char log2(8) = 3
8*max bits in T
Bk T[k>>3] is same as k/8 kinda you shift over 3 bites on a number which is the same as dividing by 8. Do this because it's faster.

01 <<(k & 07)
k& 07 gives you the first 3 digits, k>>3 gives you the numbers after the first 3 digits (aka number divided by 8)
00000111 I want these bits

(T[k>>3] & 01 <<(k & 07))

ASK ARJUN ABOUT THIS!!!! HE SEEMS TO UNDERSTAND
(T[k>>3] &(k & 07)) & 01 This allows you to find the bit GETBIT
T[k>>3] |= (01 << (k &07))
T[k>>3] &= ~(01 << (k & 07))

37

1001 | 01
k>>2 | k & 03

length of char vector is num of things/8 +1

We're gonna use c++ apparently it looks like Java.
Make a class called bitmap:
Make sure to set everything to 0.

Only ever read the file once.

make dictionary
bad : ungood

Binary trees:
Parent: left child: right child
l<p<r
  2
1 3
is a binary tree, left is less, right is more

```
find(T,X)
	if(T==NULL)
		return not found
	else if(T ->key == x)
		return T
	else if(T->key > x)
		return find(T->left,x)
	else
		return find(T->right,x)
```

insert(T,num) //you can always insert as a leaf, always.

As HW write the code for find and insert for Binary trees.

```
inorder(T)
	if(T !=null)
		inorder(T->left)
		print T->key
		inorder(T->right)
preorder(T)
	if(T !=null)
		print T->key
		preorder(T->left)
		preorder(T->right)
postorder(T)
	if(T !=null)
		postorder(T->left)
		postorder(T->right)
		print T->key
```

Maybe look up treaded binary trees

Huffman coding:
this is our alphabet

a	b	c d e f g
7	42	

Huffman coding will be on the final probably

DFS: depth first search
BFS: breadth first search

DFS(preorder traversal) recursion, recursion like stacks
BFS is more like a queue, look at the thing, inque it's children.

While not empty Q:
    x = dequeue
    find? return
    enqueue children
this works for any number of children as does depth first search.

We're doing graphs!!!!!
Nodes aka vertices
Lines are edges