hello, section!

week 4

pset 3 recap

structs and typedef

structs and typedef

how would you represent this in code?

structs and typedef

```
dog fido; ???
```

structs and typedef

```
int age;
string breed;
string fluff;
bool good;
```

dog

structs and typedef

```
int age = 1;
string breed = "golden";
string fluff = "max";
bool good = true;
```

dog fido;

structs and typedef

```
typedef struct dog
{
    int age;
    string breed;
    string fluff;
    bool good;
}
node;
```

structs and typedef

```
typedef struct dog
{
    int age;
    string breed;
    string fluff;
    bool good;
}
dog;
```

```
dog fido;
fido.age = 1;
fido.breed = "golden";
fido.fluff = "max";
fido.good = true;
```

structs and typedef

```
typedef struct dog
{
    int age;
    string breed;
    string fluff;
    bool good;
}
dog;
```

```
dog fido =
    {.age = 1,
     .breed = "golden",
     .fluff = "max",
     .good = true};
```

structs and typedef

```
dog pepper =
    {.age = 3,
     .breed = "pug",
     .fluff = "min",
     .good = true};
```

```
dog fido =
    {.age = 1,
     .breed = "golden",
     .fluff = "max",
     .good = true};
```

structs and typedef

```
dog* pepper = malloc(sizeof(dog));
pepper->age = 1;
pepper->breed = "pug";
pepper->fluff = "min";
pepper->good = true;
```

structs and typedefs

`malloc()`
global

heap

stack

local
variables

linked lists

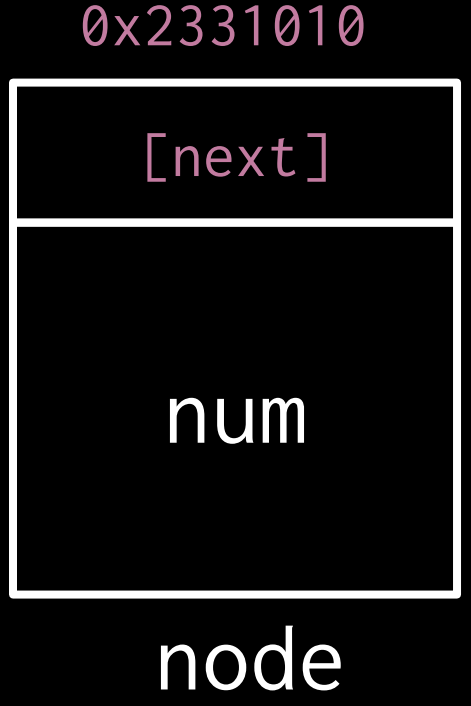
node

linked lists

```
typedef struct node
{
   int num;
   struct node* next;
}
node;
```

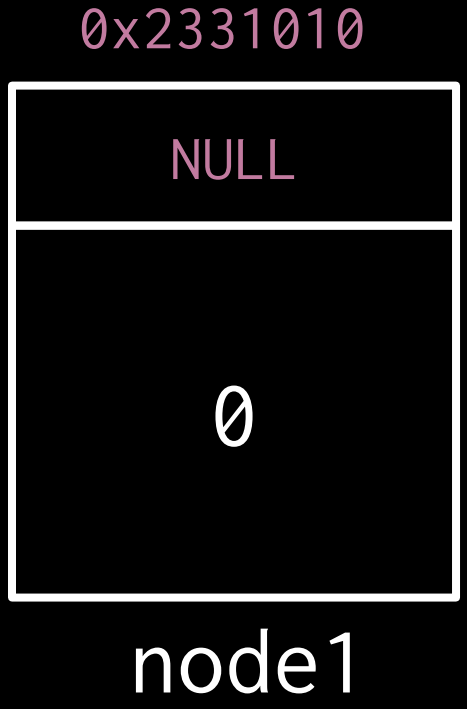
linked lists

```
typedef struct node
{
   int num;
   struct node* next;
}
node;
```



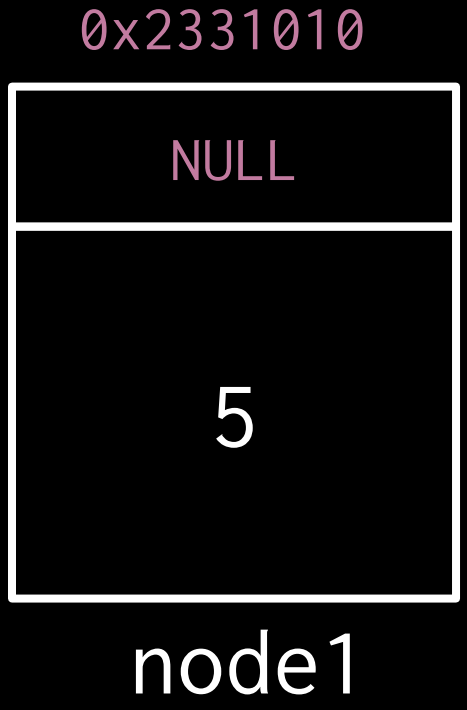
linked lists

```
node* node1 = malloc(sizeof(node));
```



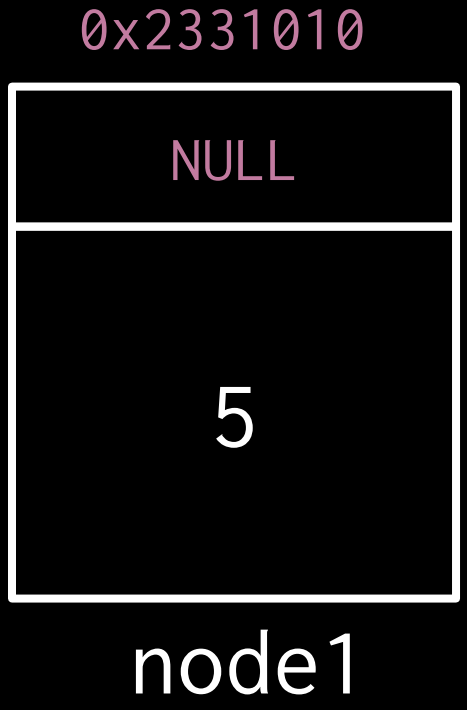
linked lists

```
node* node1 = malloc(sizeof(node));
*node1.num = 5;
```



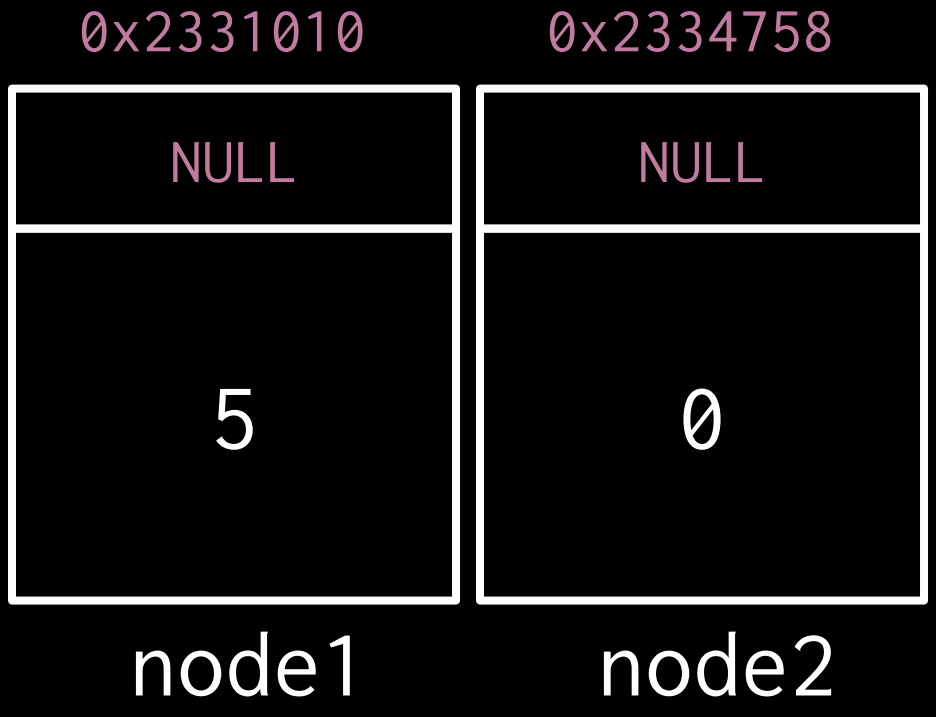
linked lists

```
node* node1 = malloc(sizeof(node));
node1->num = 5;
```



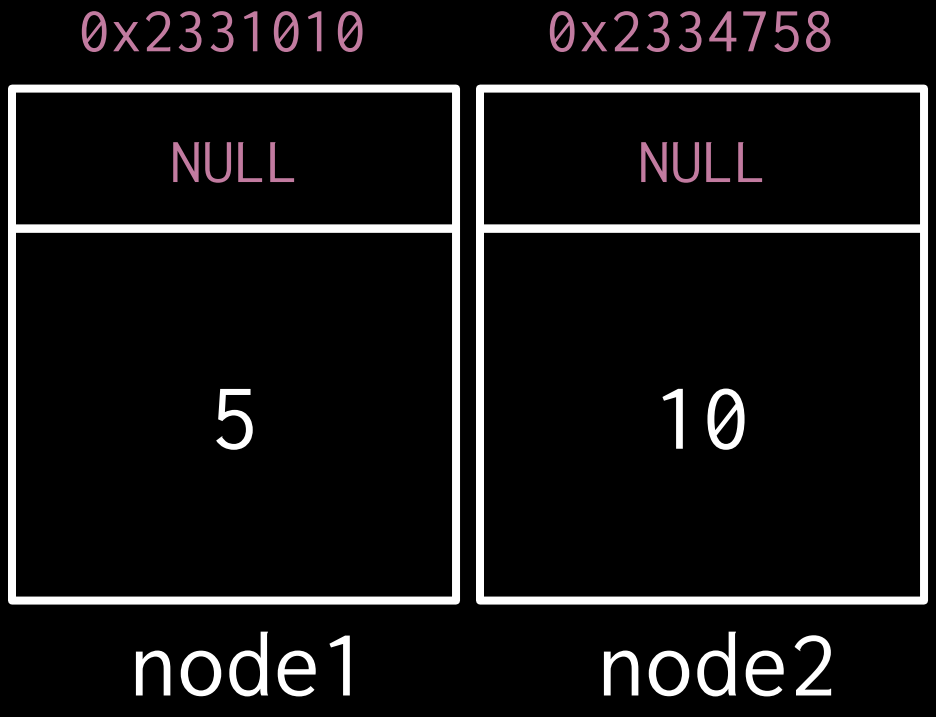
linked lists

```
node* node2 = malloc(sizeof(node));
```



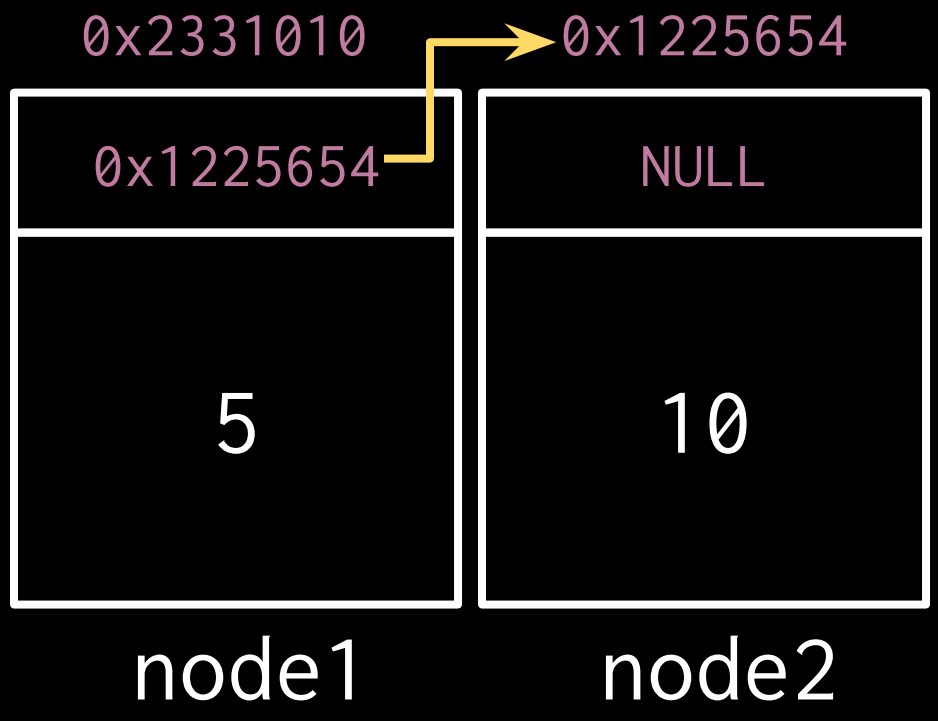
linked lists

```
node* node2 = malloc(sizeof(node));
node2->num = 10;
```



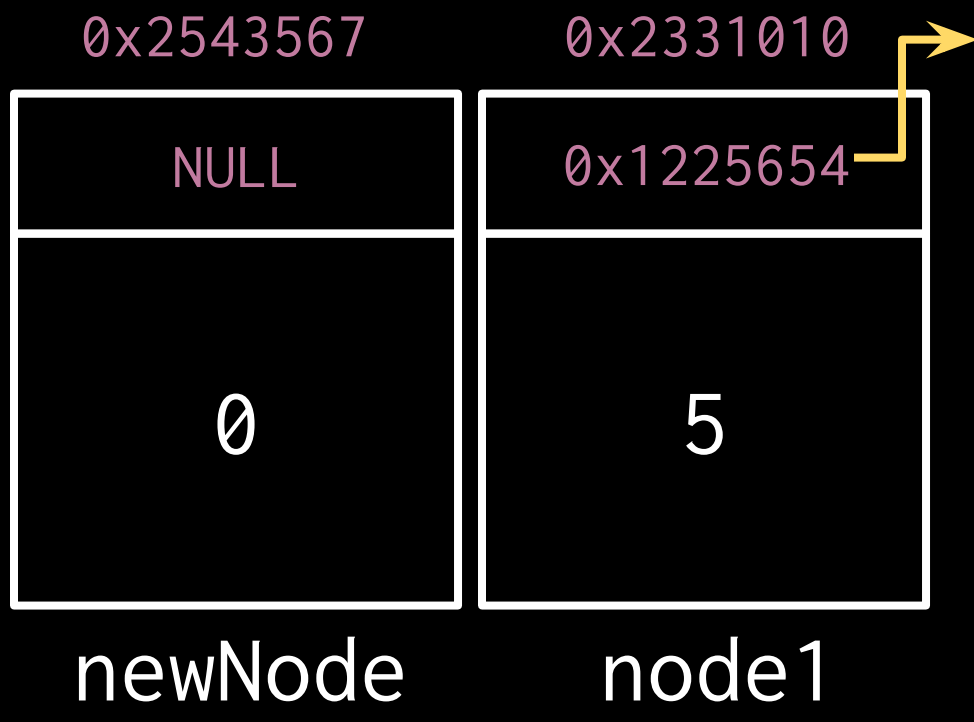
linked lists

```
node* node2 = malloc(sizeof(node));
node2->num = 10;
node1->next = node2;
```



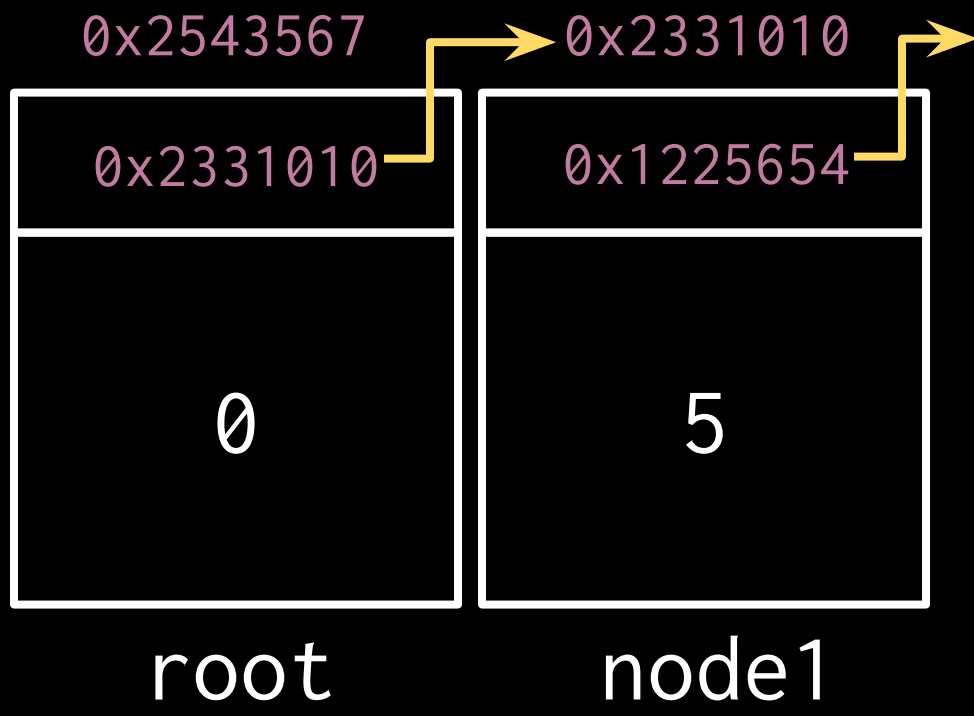
linked lists

```
node* root = malloc(sizeof(node));
```

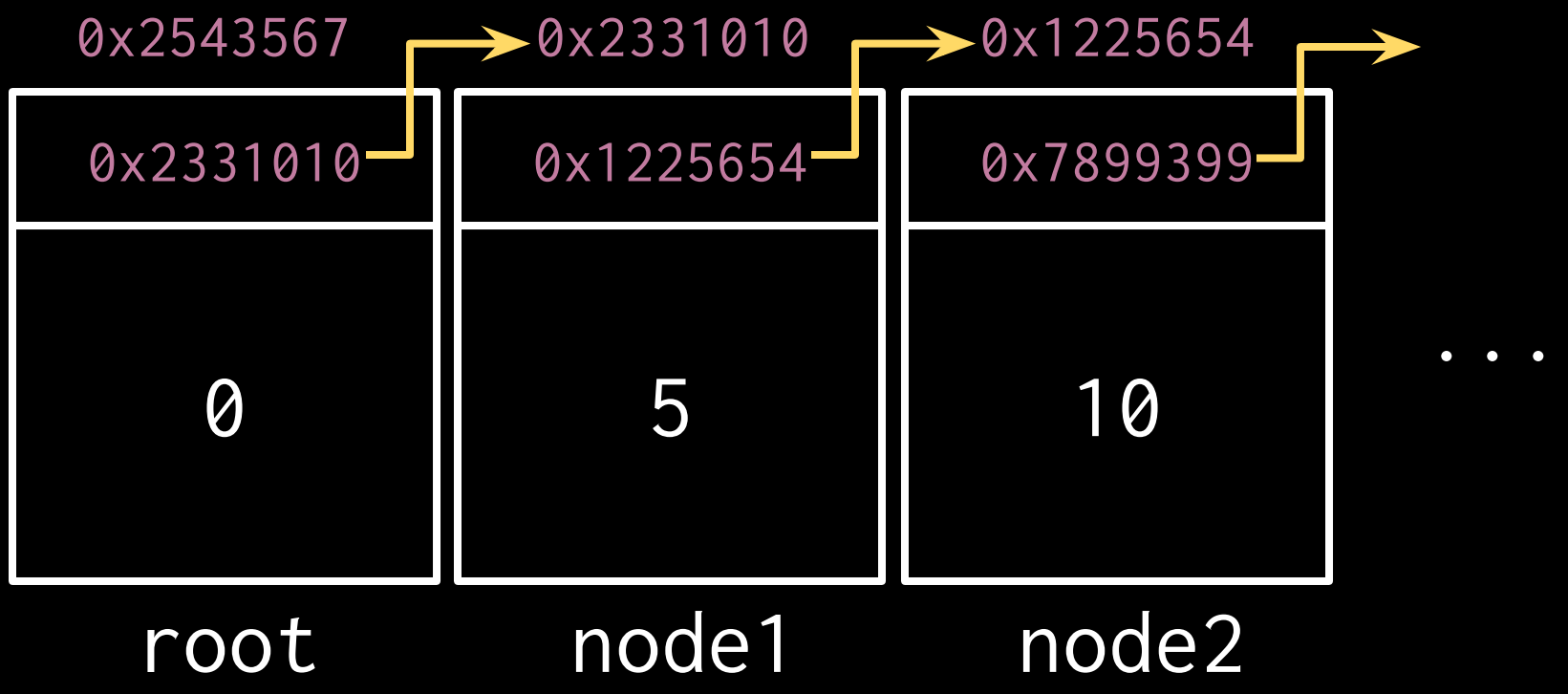


linked lists

```
node* root = malloc(sizeof(node));
root->next = node1;
```



linked lists



why linked lists?

why linked lists?

pros:

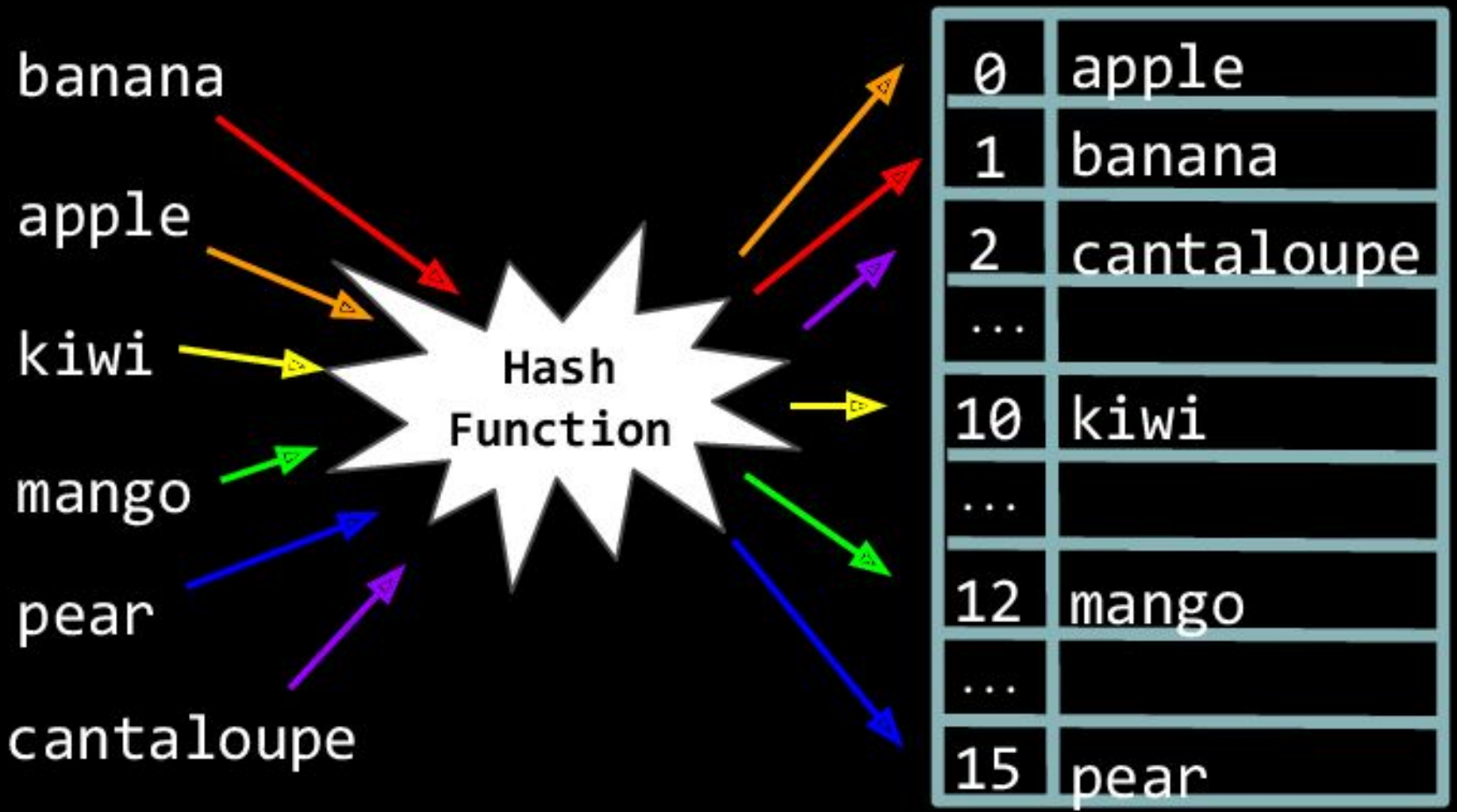
- insertion
- deletion
- flexibility

cons:

- lookup
- fragmentation

hash tables

hash tables


banana
apple
kiwi
mango
pear
cantaloupe
Hash Function
0 apple
1 banana
2 cantaloupe
...
10 kiwi
...
12 mango
...
15 pear

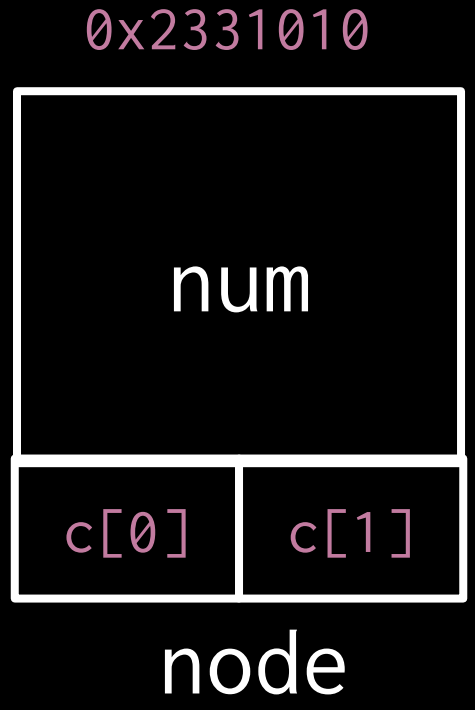
trees

trees

```
typedef struct node
{
   int num;
   struct node* children[2];
}
node;
```

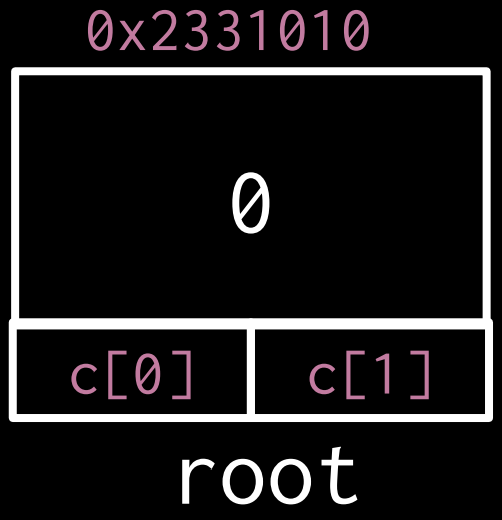
trees

```
typedef struct node
{
   int num;
   struct node* children[2];
}
node;
```



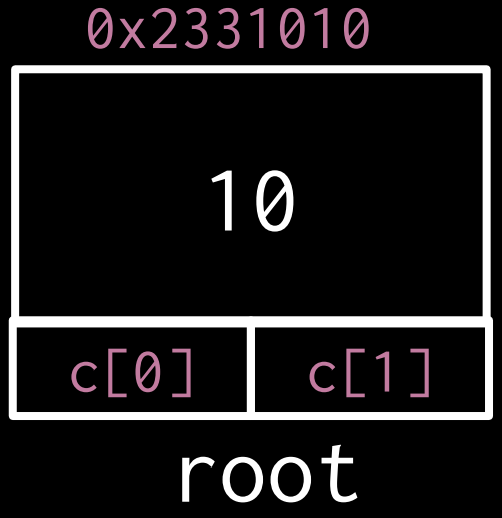
trees

```
node* root = malloc(sizeof(node));
```



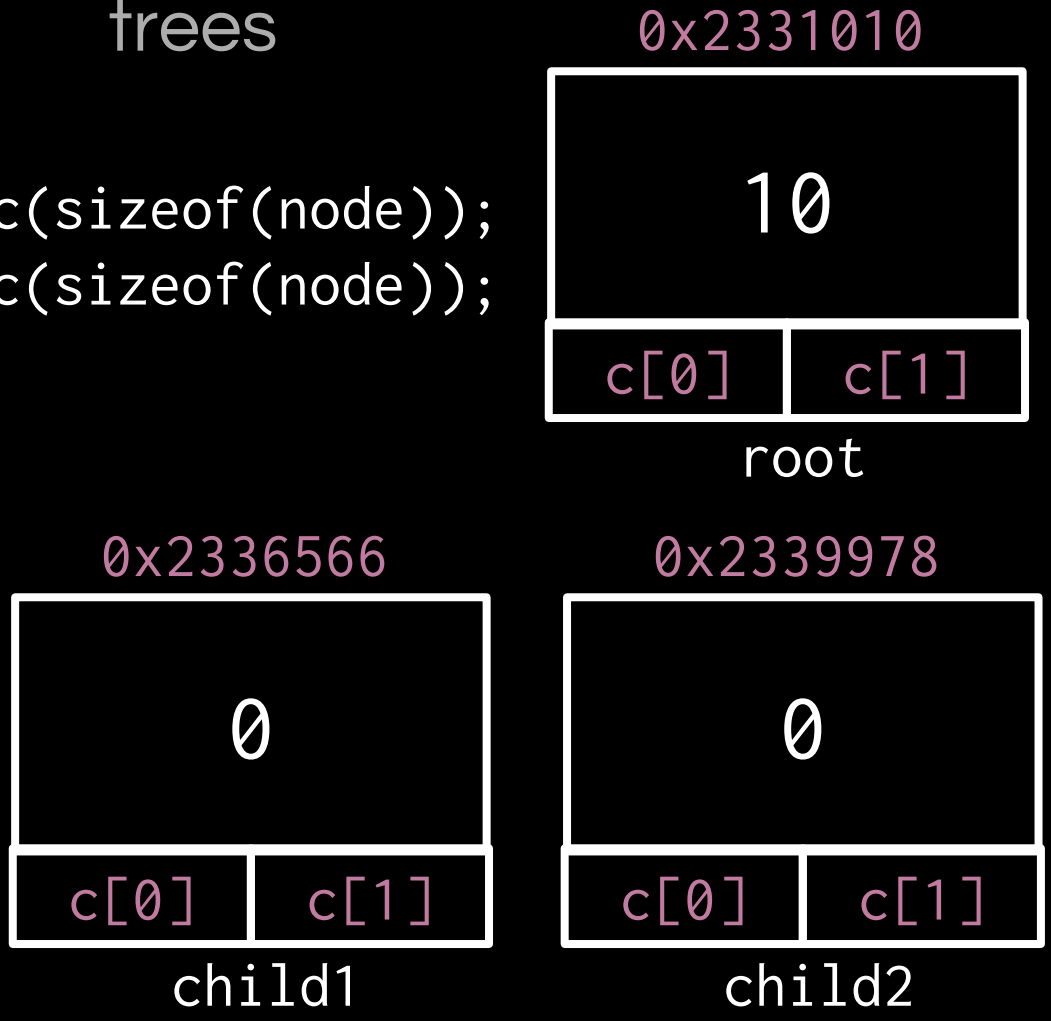
trees

```
node* root = malloc(sizeof(node));
root->num = 10;
```



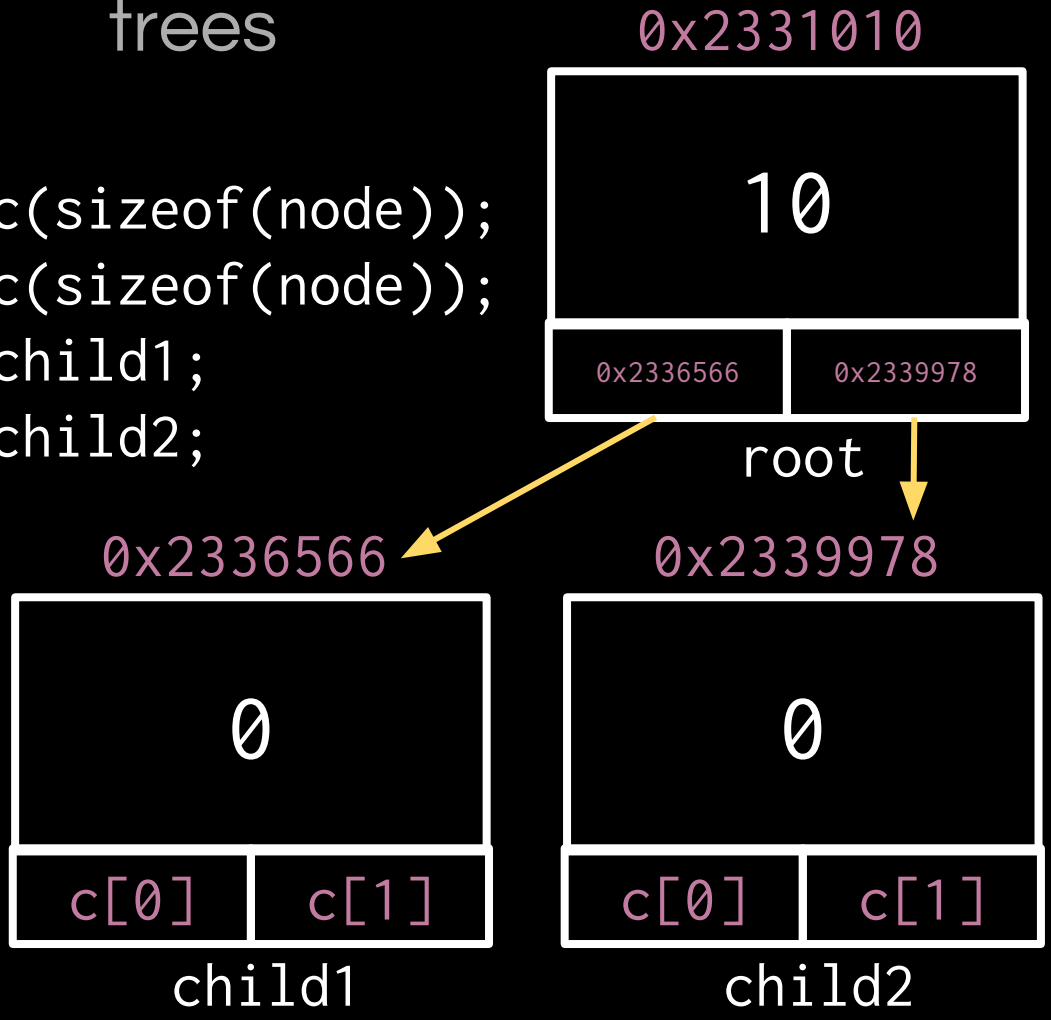
trees

```
...
node* child1 = malloc(sizeof(node));
node* child2 = malloc(sizeof(node));
```



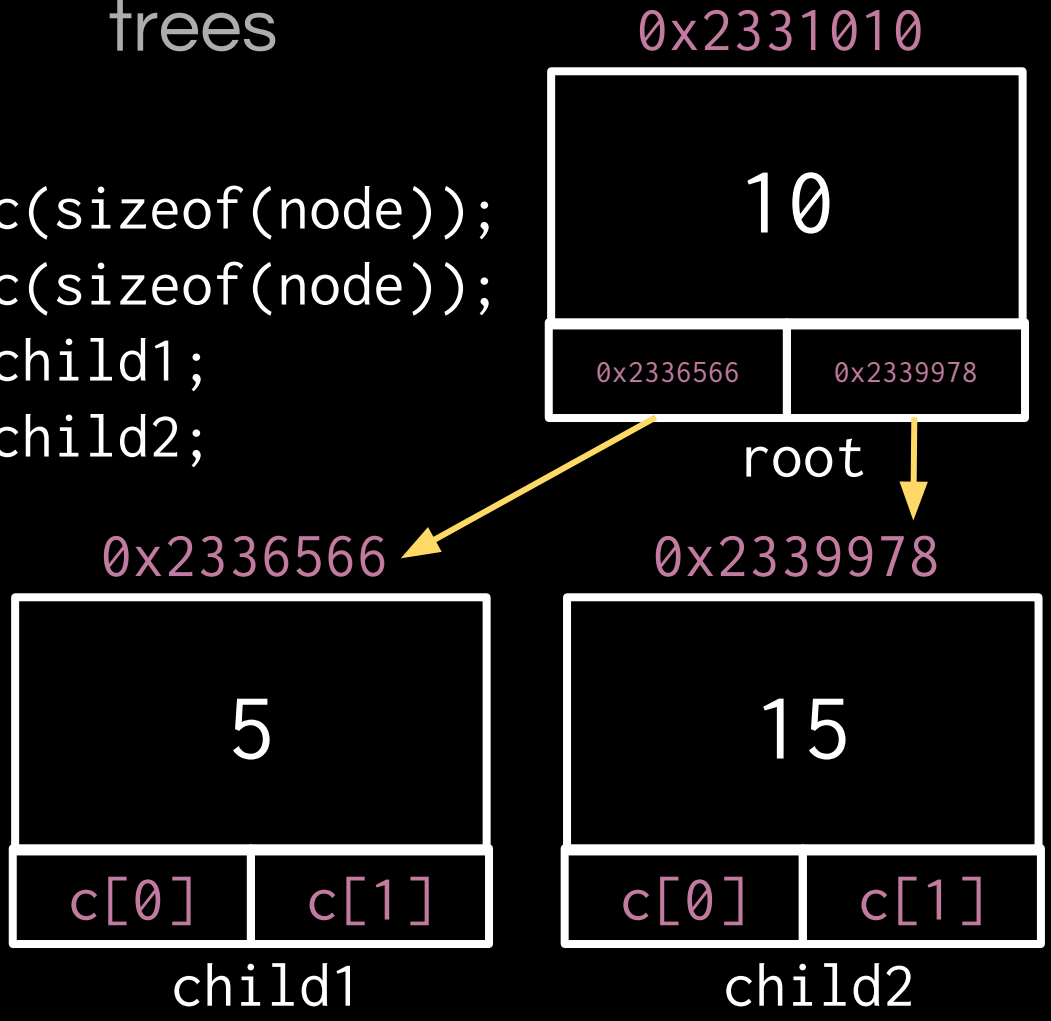
trees

```
...
node* child1 = malloc(sizeof(node));
node* child2 = malloc(sizeof(node));
root->children[0] = child1;
root->children[1] = child2;
```



trees

```
...
node* child1 = malloc(sizeof(node));
node* child2 = malloc(sizeof(node));
root->children[0] = child1;
root->children[1] = child2;
child1->num = 5;
child2->num = 15;
```



tries

tries

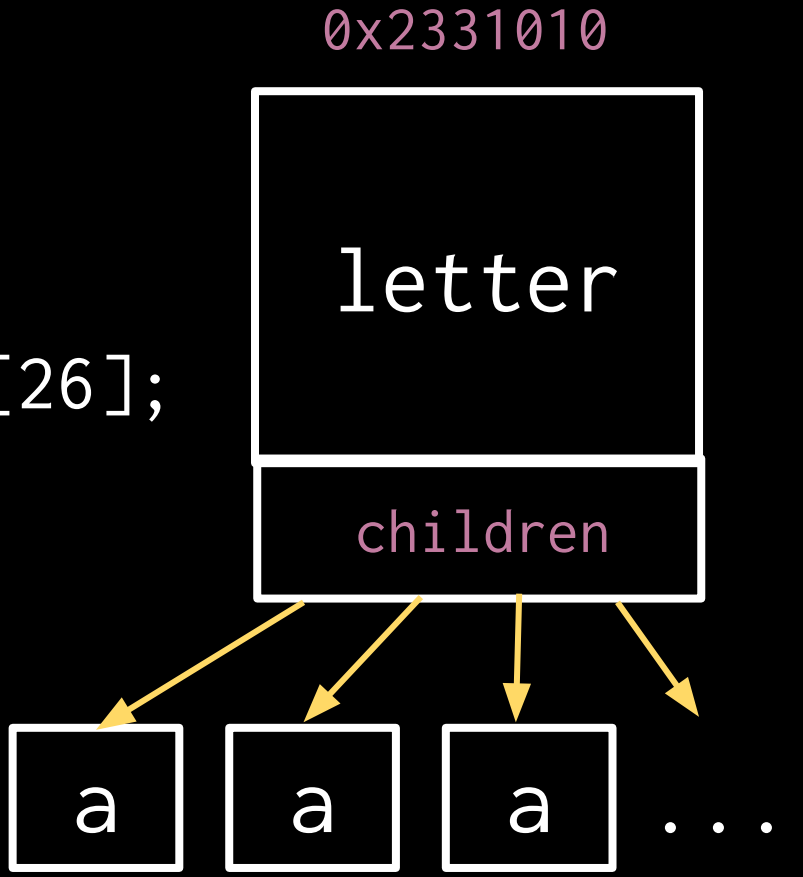
```
typedef struct node
{
   char letter;
   struct node* children[26];
}
node;
```

tries

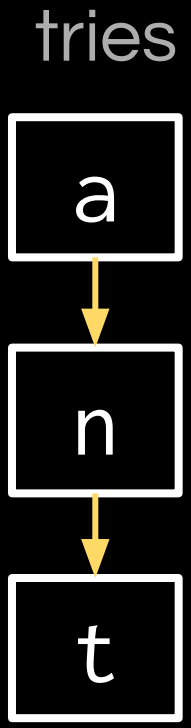
```
typedef struct node
{
   char letter;
   struct node* children[26];
}
node;
```

tries

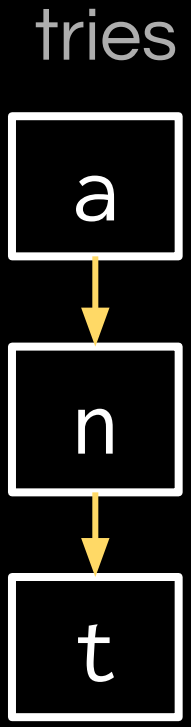
```
typedef struct node
{
   char letter;
   struct node* children[26];
}
node;
```



```
insert(“ant”);
```

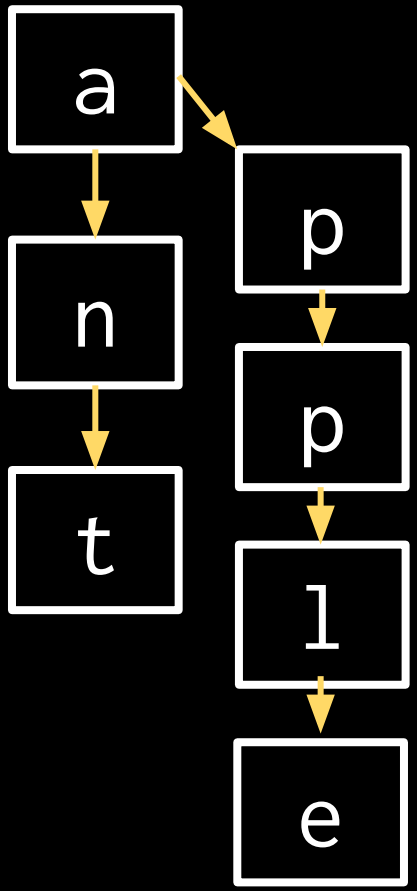


```
insert(“apple”);
```



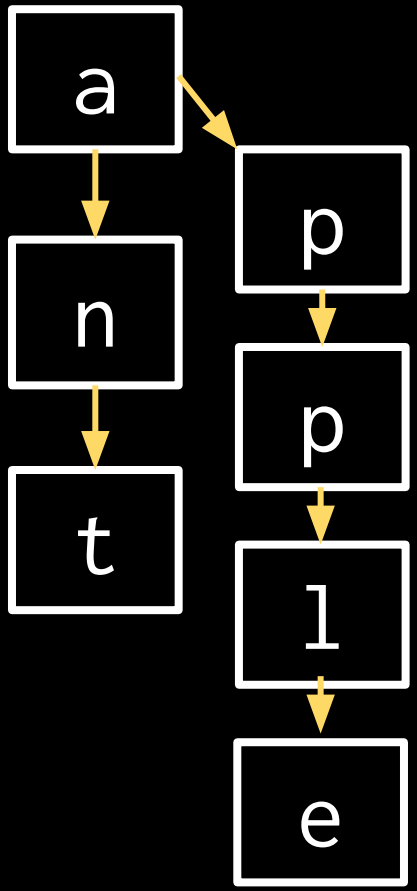
tries

```
insert("apple");
```



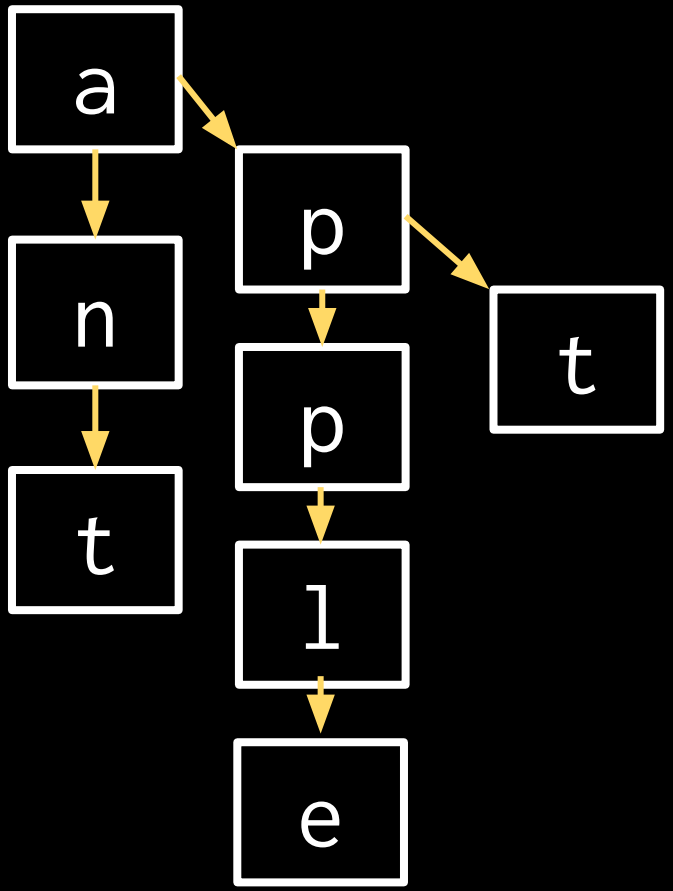
tries

```
insert(“apt”);
```



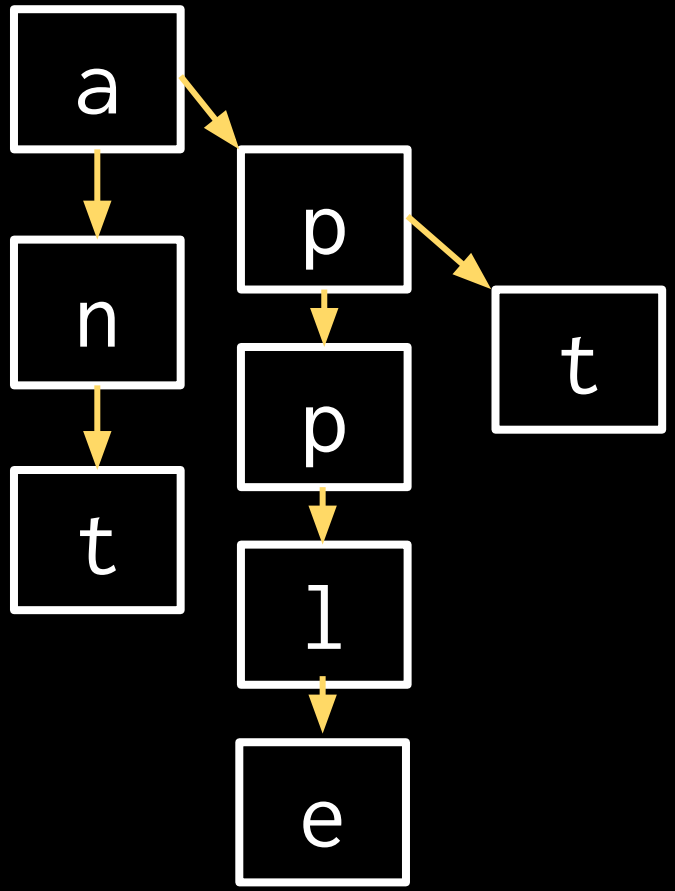
tries

```
insert(“apt”);
```



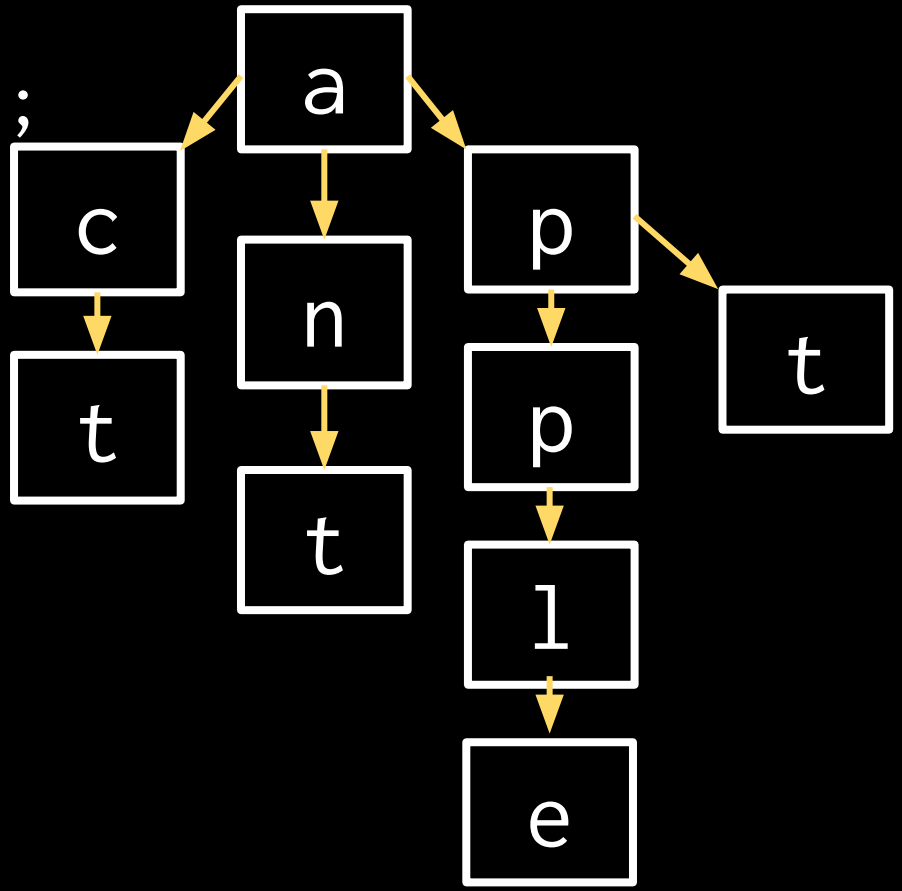
tries

```
insert(“act”);
```



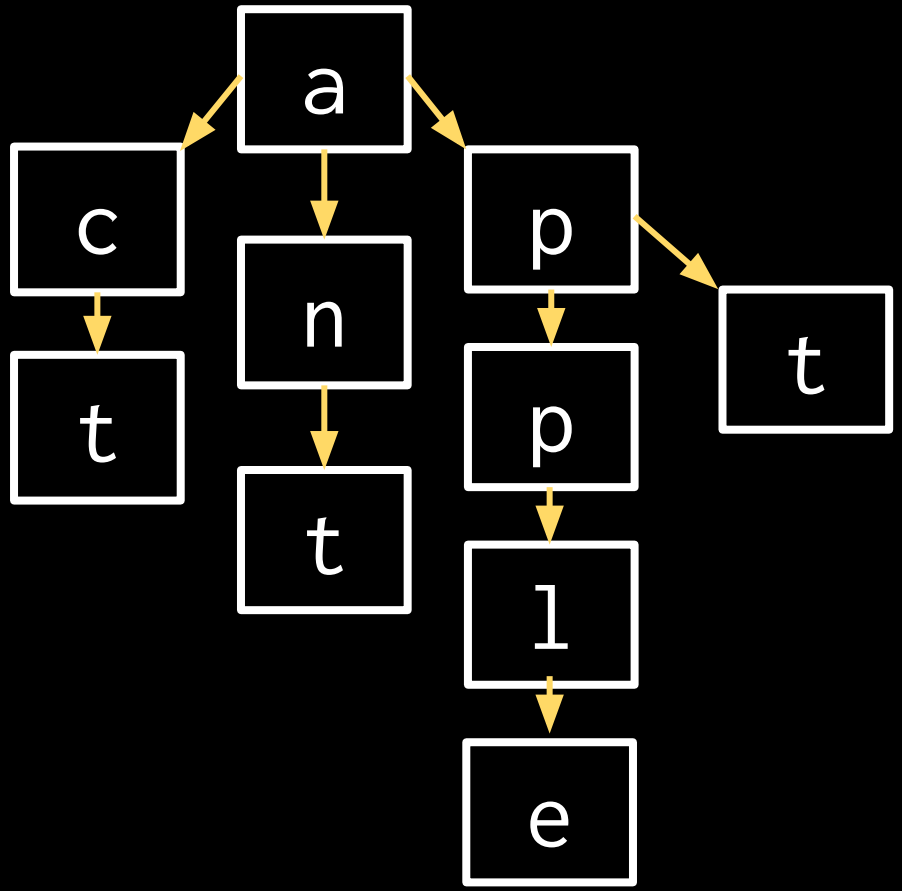
tries

```
insert(“act”);
```



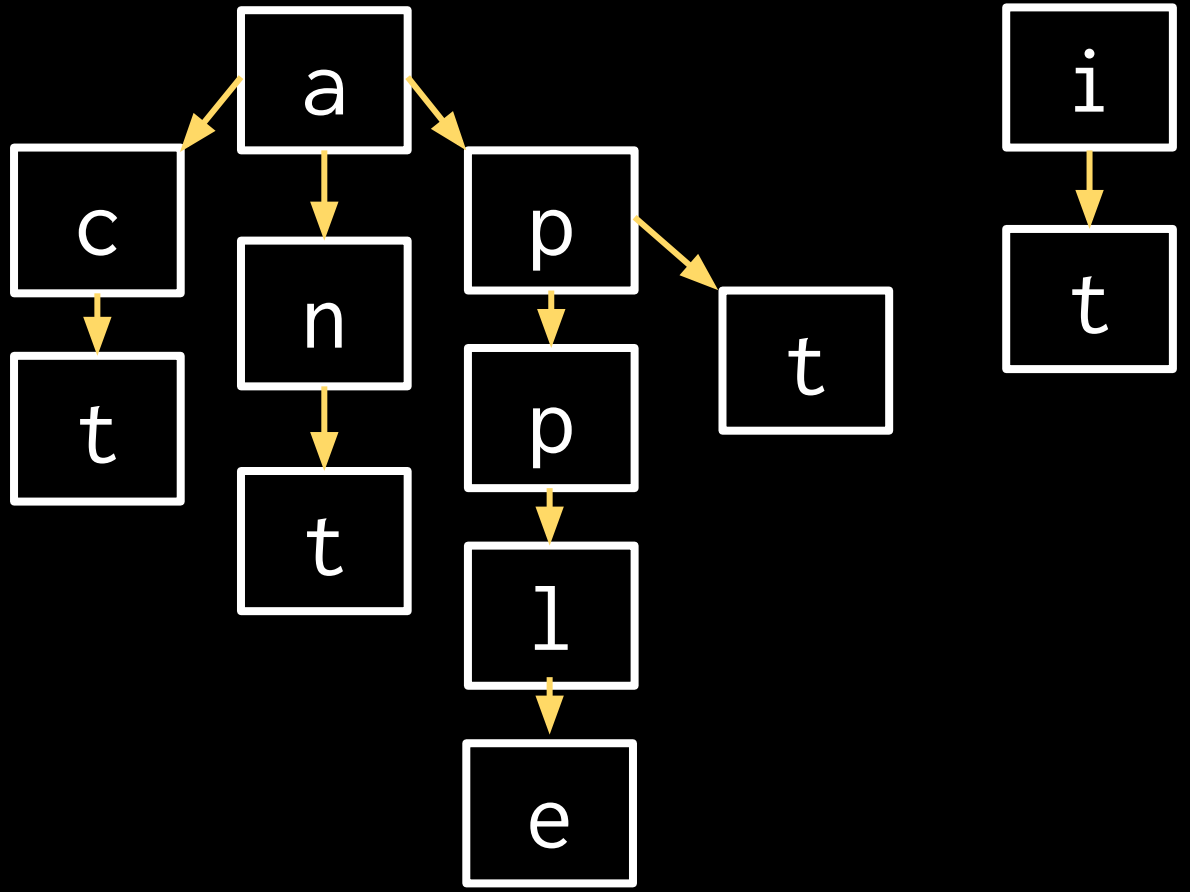
tries

```
insert("it");
```



tries

```
insert("it");
```



why tries?

why tries?

pros:

- lookup
- sort
- insertion

cons:

- deletion
- space

pset 4 requirements

- a **pointer** is just an ***address***
- how do multiple files work together?
- nodes (???)