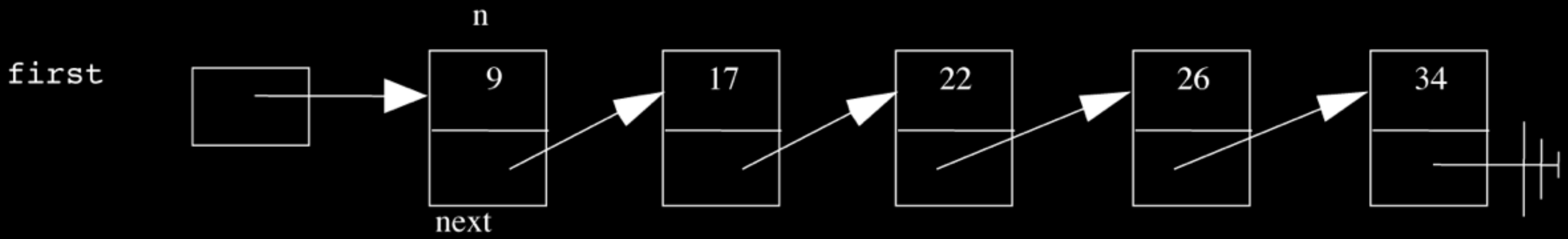


http://cs.calvin.edu/books/c++/ds/1e/

delete

insert

search

...

```
bool search(int n, node* list)
{
    node* ptr = list;
    while (ptr != NULL)
    {
        if (ptr->n == n)
        {
            return true;
        }
        ptr = ptr->next;
    }
    return false;
}
```


THANKS FOR YOUR

```
push

pop

...
```

```
typedef struct
{
    int numbers[CAPACITY];
    int size;
}
stack;
```

```
typedef struct
{
    int* numbers;
    int size;
}
stack;
```

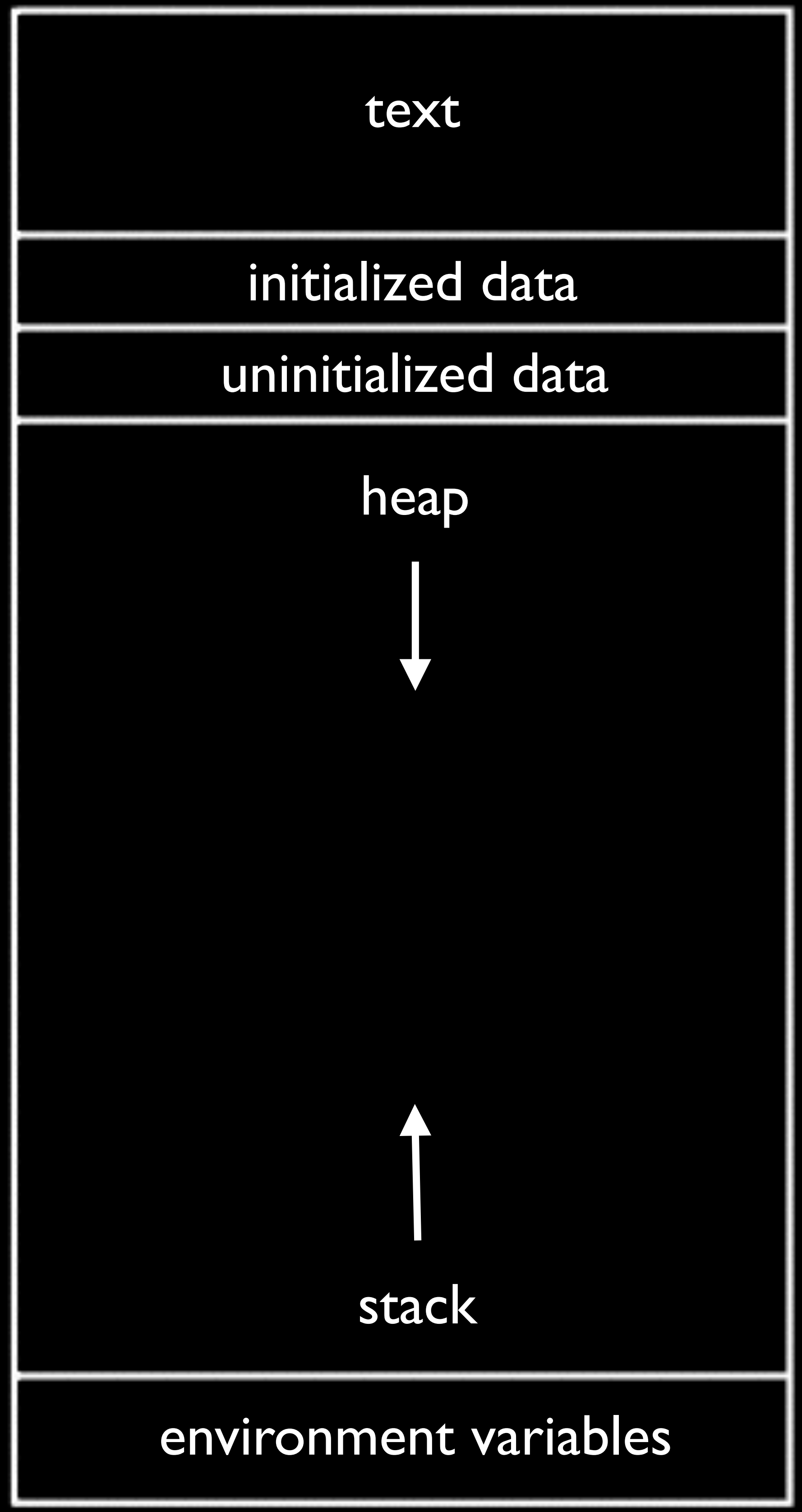

engadget

```
enqueue

dequeue

...
```

```
typedef struct
{
    int front;
    int numbers[CAPACITY];
    int size;
}
queue;
```

```
typedef struct
{
    int front;
    int* numbers;
    int size;
}
queue;
```

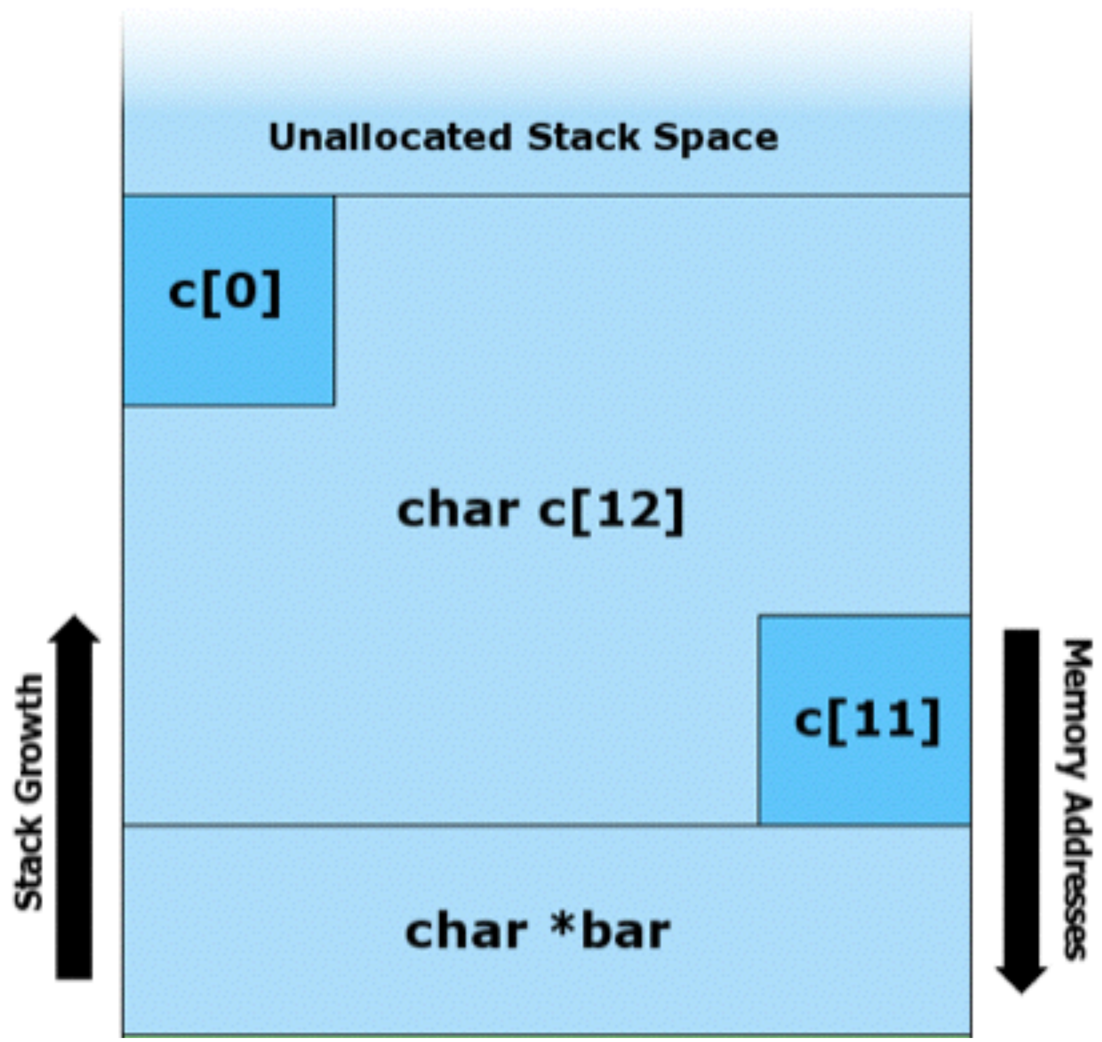
Jack Learns the Facts About Queues and Stacks


text
initialized data
uninitialized data
heap
stack
environment variables

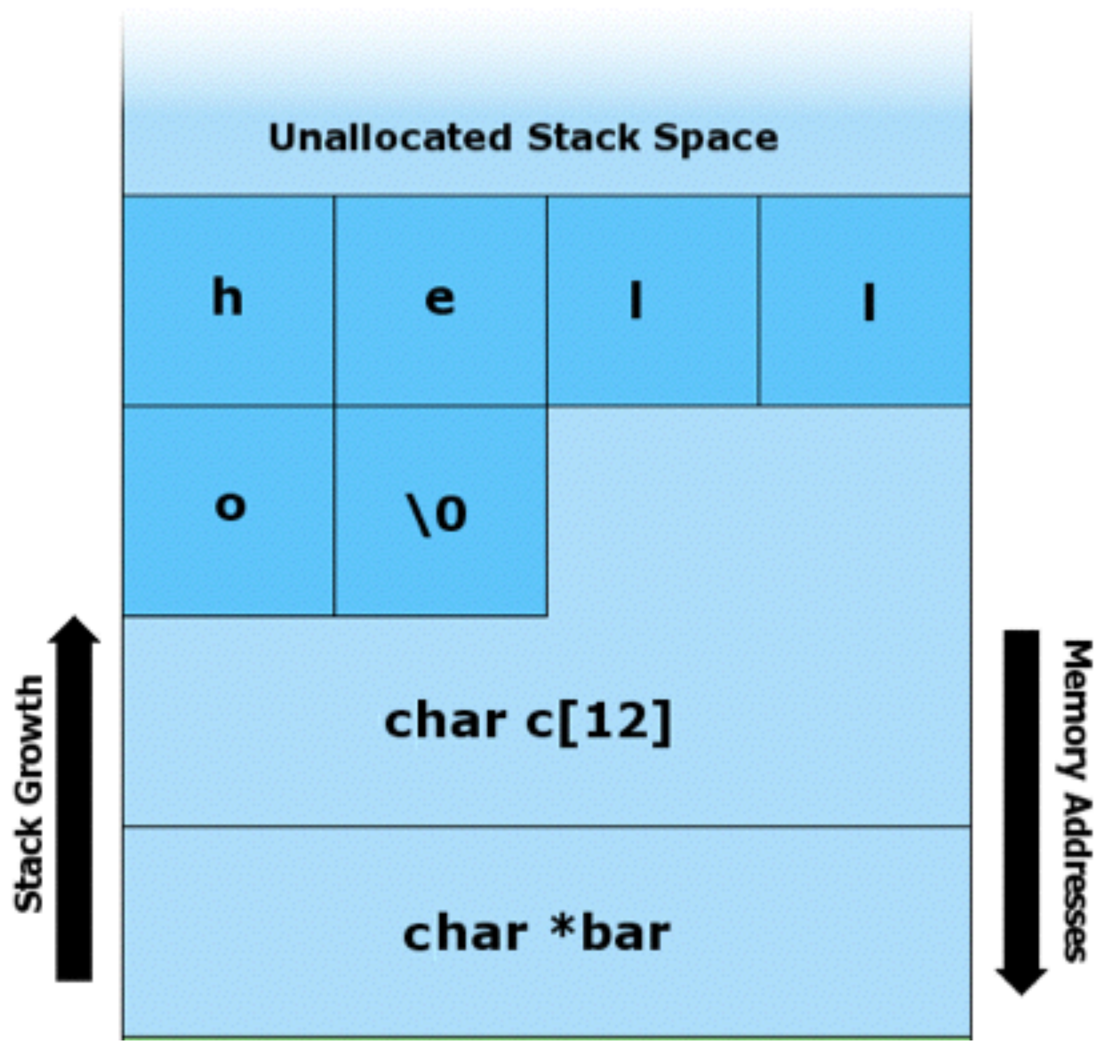
buffer overflow

```c
#include <string.h>

void f(char* bar)
{
    char c[12];
    strncpy(c, bar, strlen(bar));
}

int main(int argc, char* argv[])
{
    f(argv[1]);
}
```

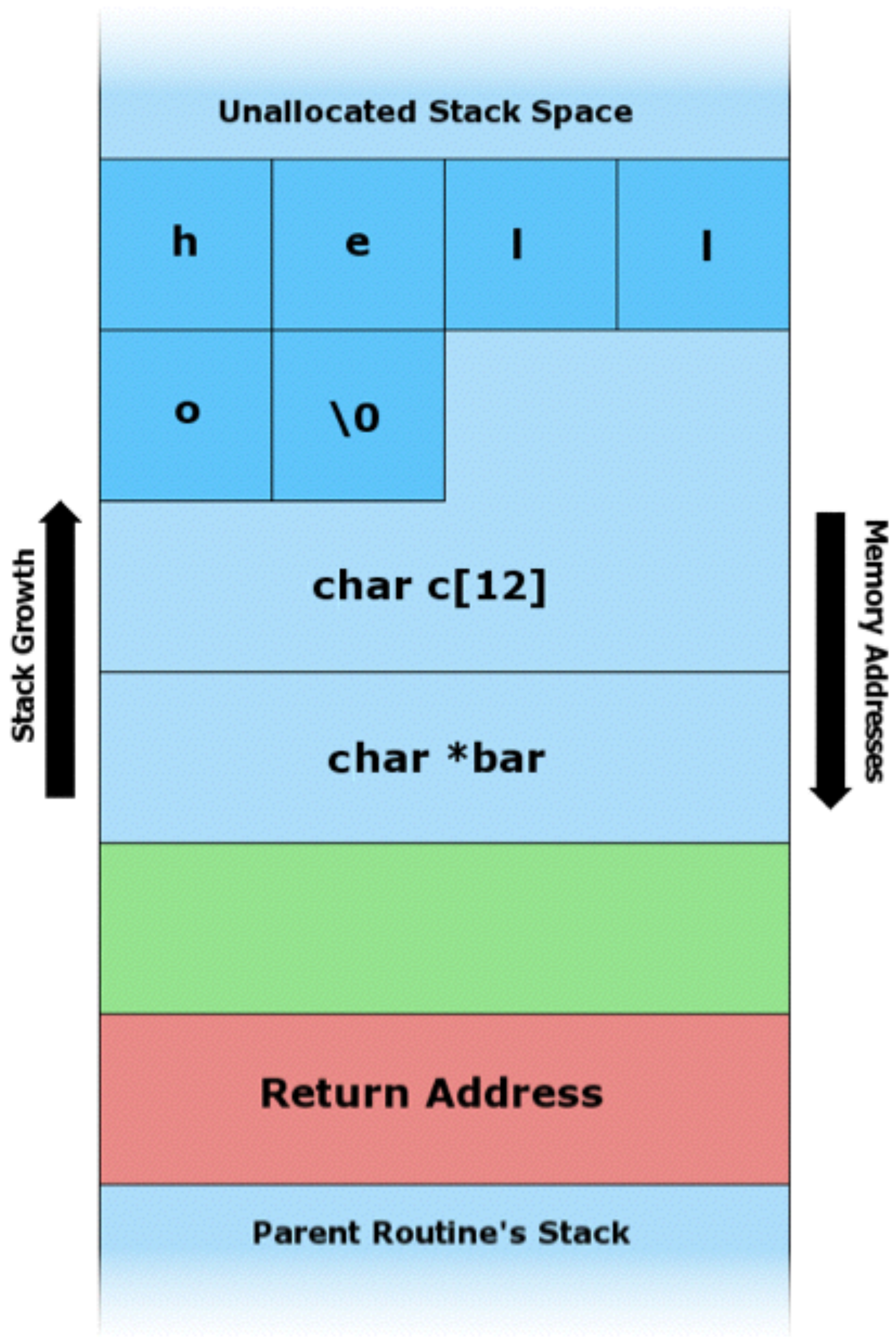
adapted from http://en.wikipedia.org/wiki/Stack_buffer_overflow



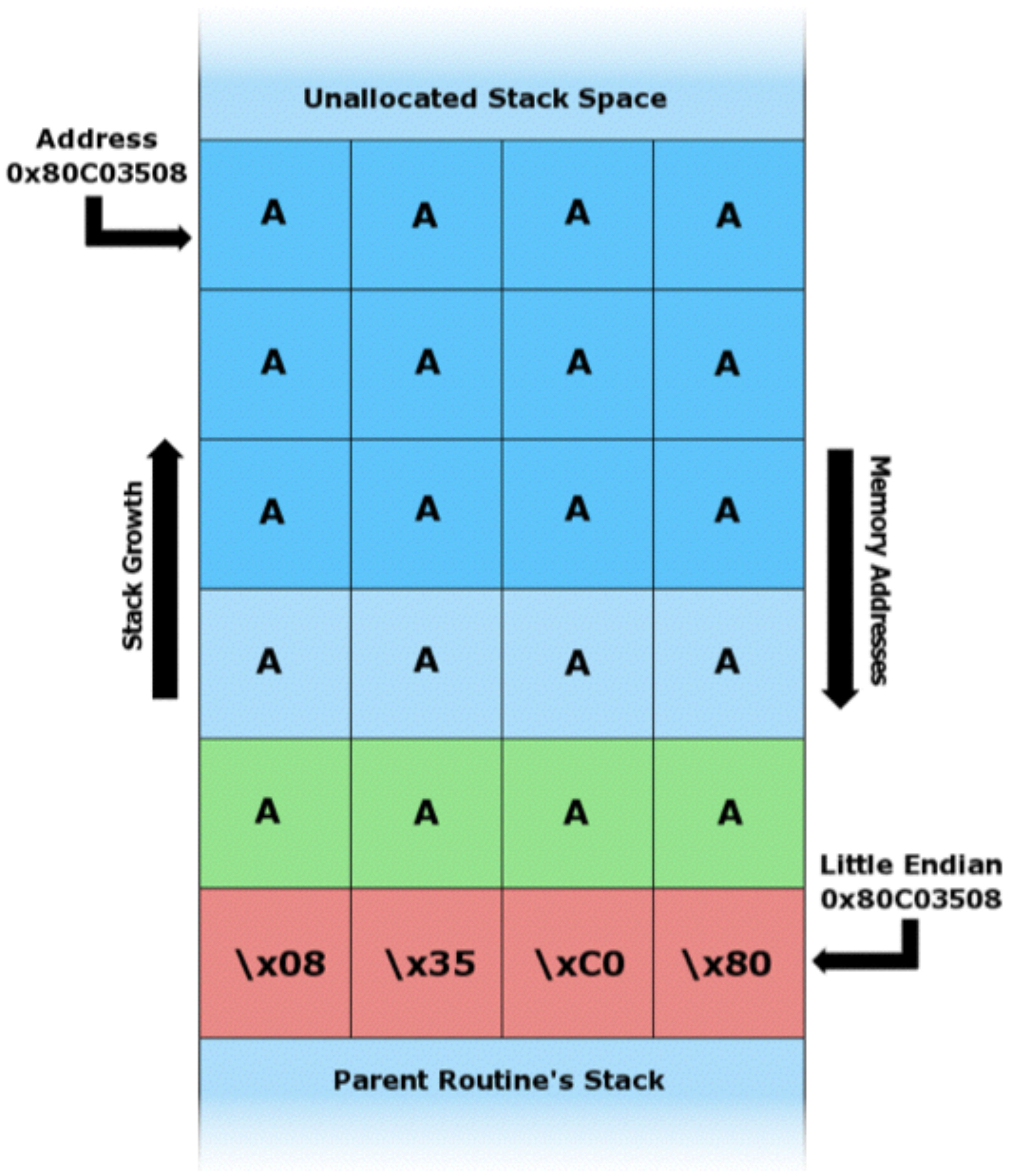
adapted from http://en.wikipedia.org/wiki/Stack_buffer_overflow



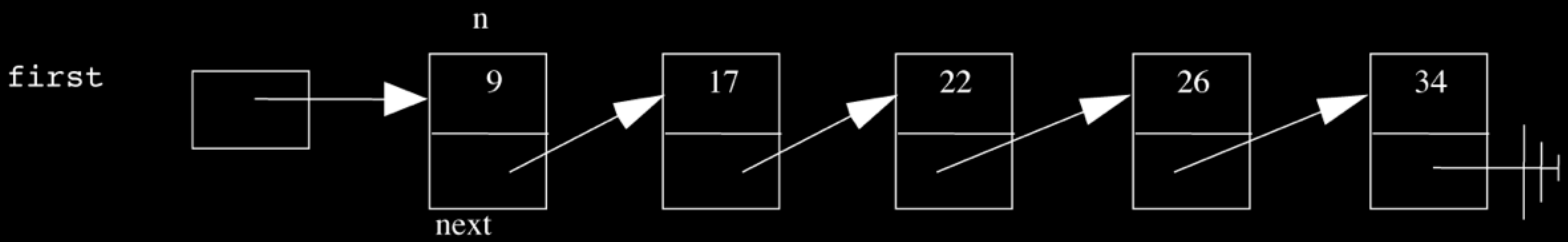
adapted from http://en.wikipedia.org/wiki/Stack_buffer_overflow



adapted from http://en.wikipedia.org/wiki/Stack_buffer_overflow



adapted from http://en.wikipedia.org/wiki/Stack_buffer_overflow


first
n
9
17
22
26
34
next

$$O(n)$$

$O(\log n)$

$O(1)$

22	33	44	55	66	77	88

22	33	44	55	66	77	88

tree

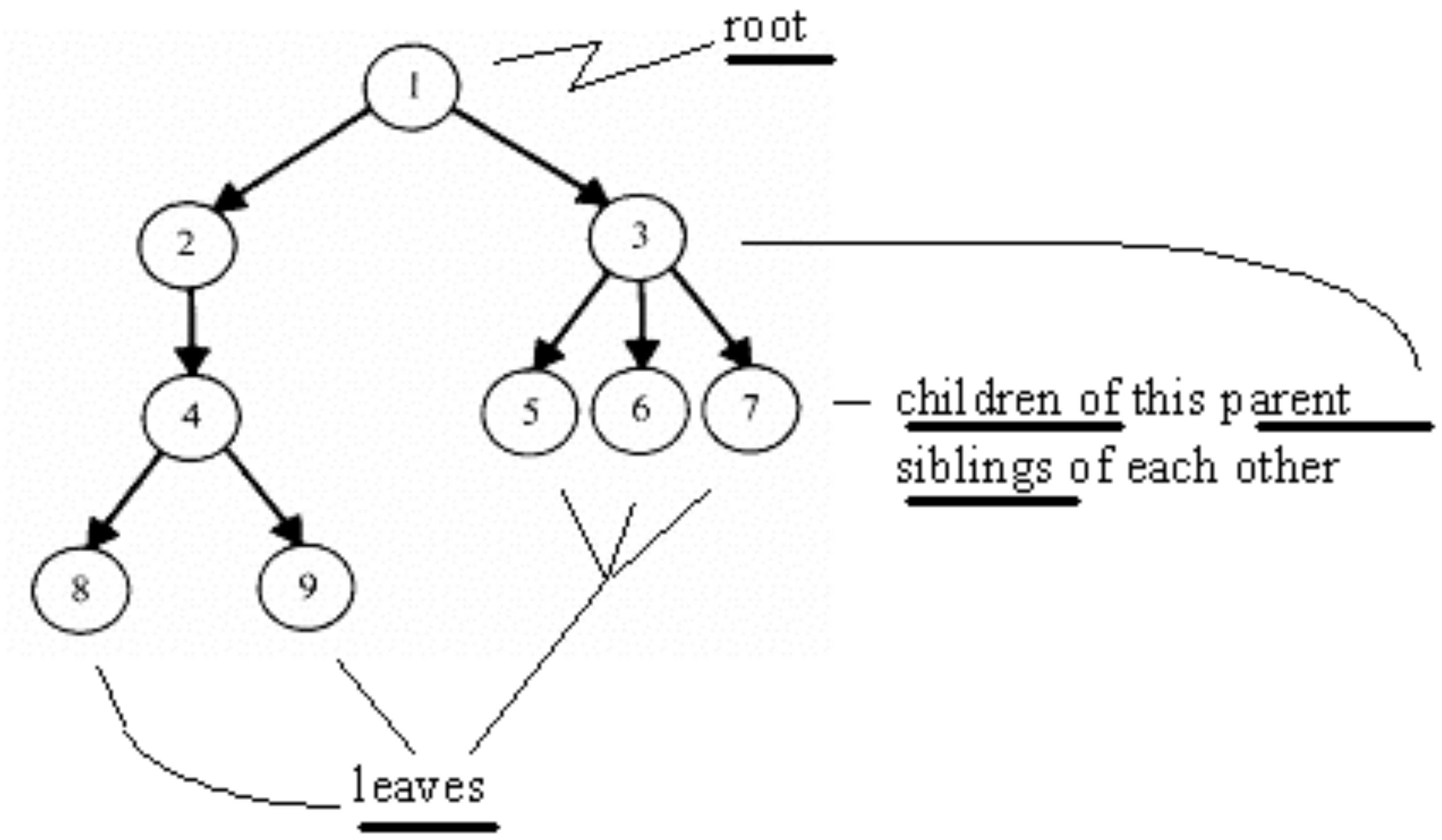


Figure by Larry Nyhoff.

binary search tree

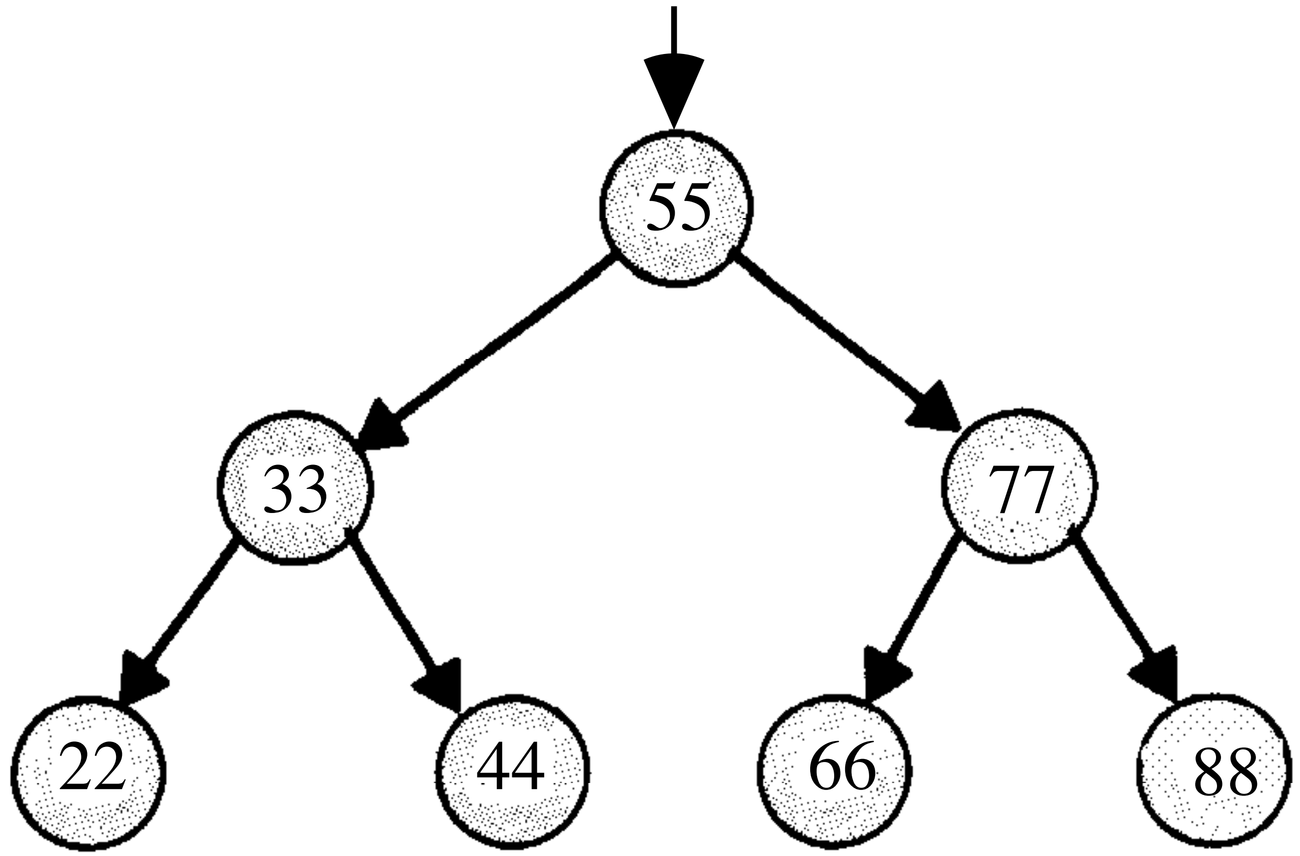
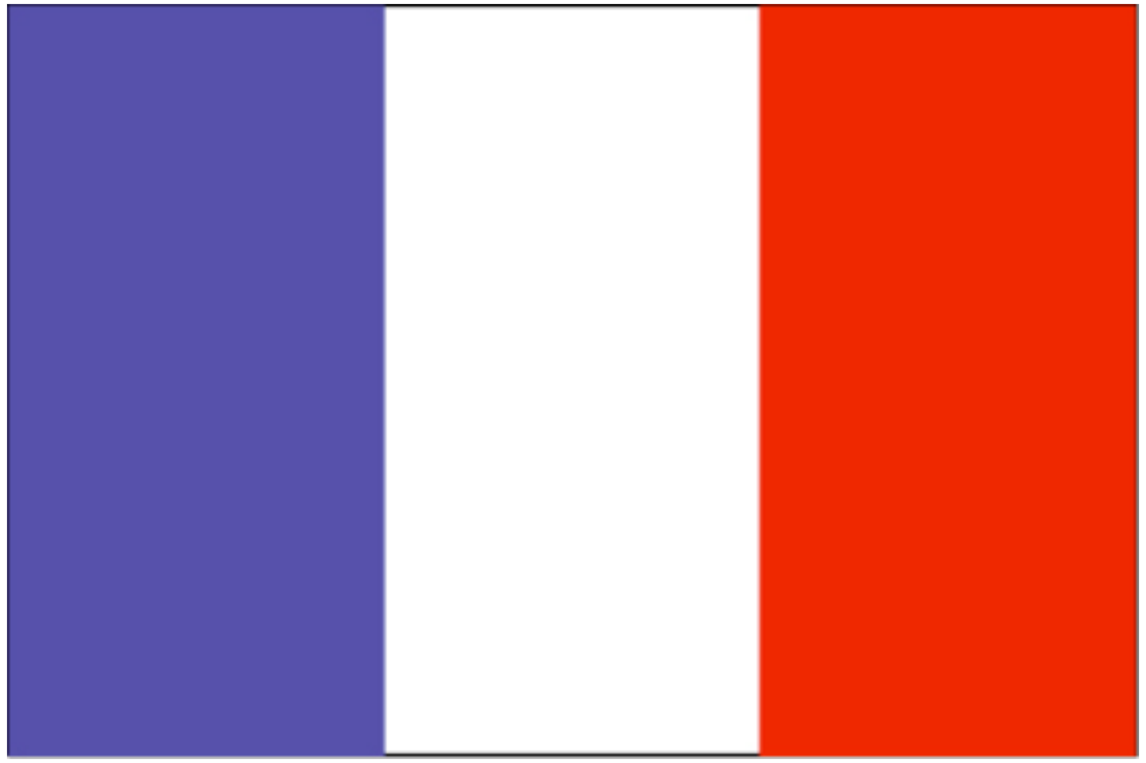


Figure from http://cs.calvin.edu/books/c++/ds/1e/.

```
typedef struct node
{
    int n;
    struct node* left;
    struct node* right;
}
node;
```

```
bool search(int n, node* tree)
{
    if (tree == NULL)
    {
        return false;
    }
    else if (n < tree->n)
    {
        return search(n, tree->left);
    }
    else if (n > tree->n)
    {
        return search(n, tree->right);
    }
    else
    {
        return true;
    }
}
```

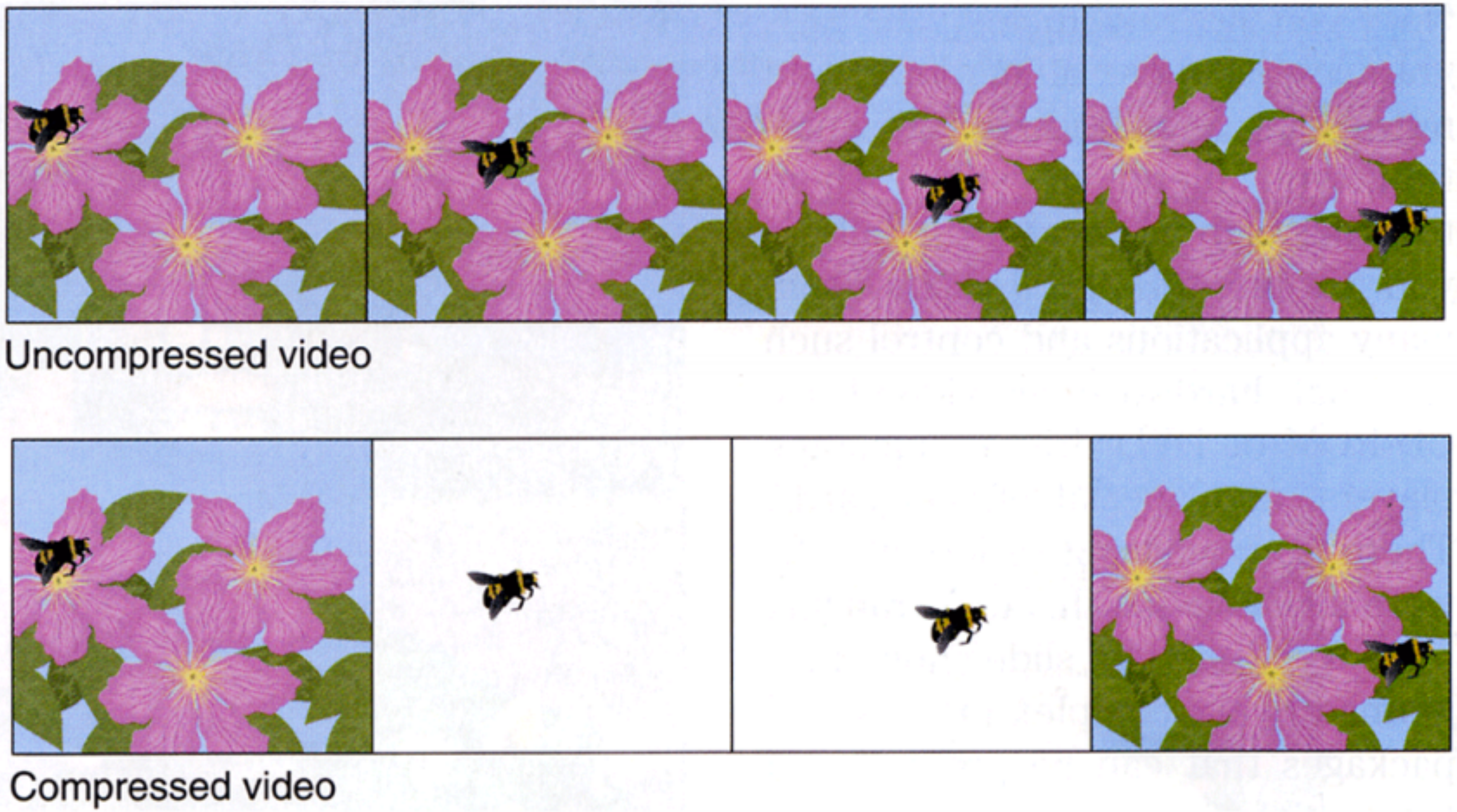
http://www.worldatlas.com/webimage/flags/flagcnty.htm



http://www.worldatlas.com/webimage/flags/flagcnty.htm

Uncompressed video

Dennis P. Curtin, *et al.*, *Information Technology: The Breaking Wave*



Dennis P. Curtin, *et al.*, *Information Technology: The Breaking Wave*

ASCII

A	B	C	D	E	F	G	H	I	J	K	L	M
65	66	67	68	69	70	71	72	73	74	75	76	77
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
78	79	80	81	82	83	84	85	86	87	88	89	90

morse code

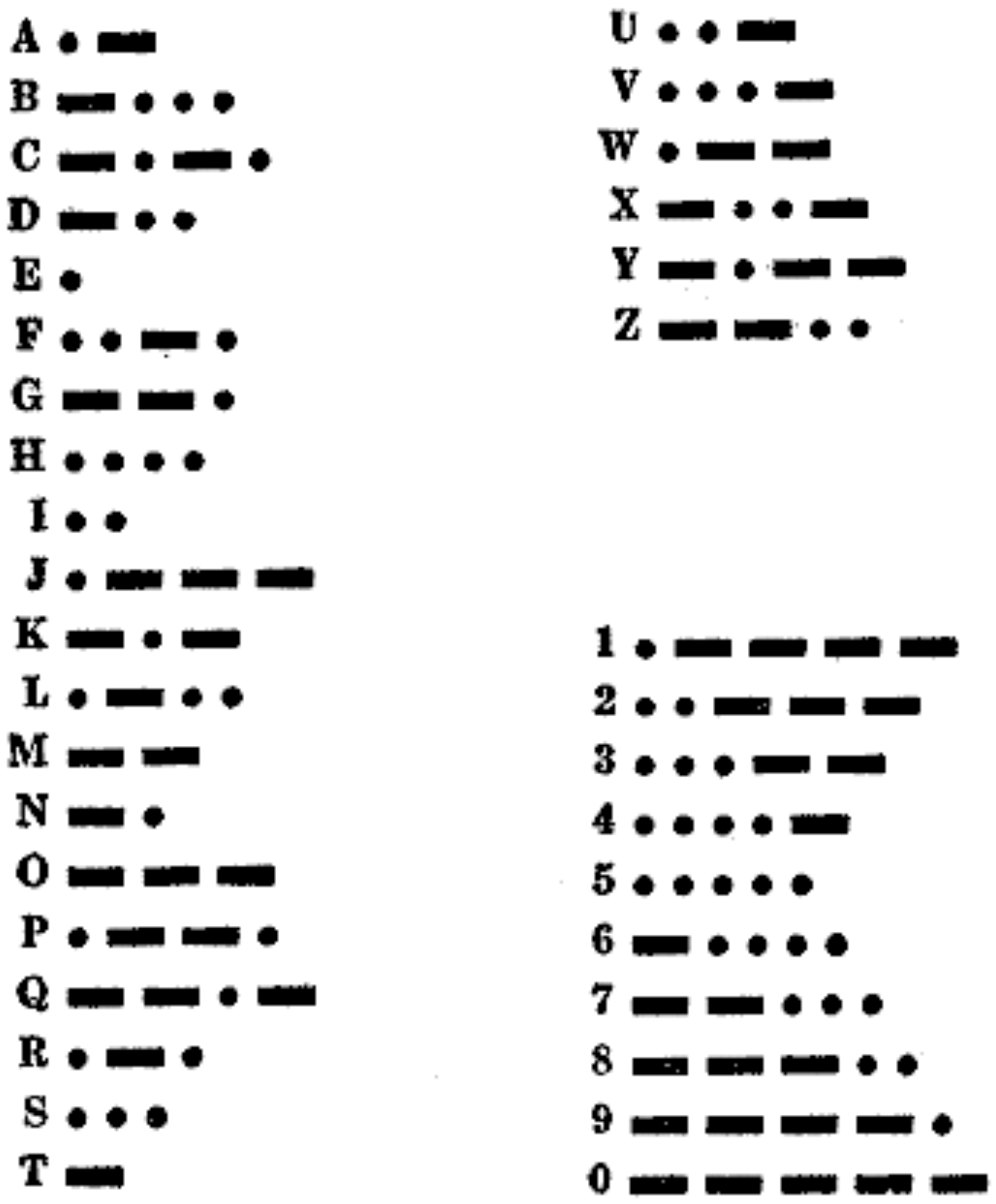


Image adapted from Wikipedia.

“ECEABEADCAEDEEEECEADEEEEEDBAAEABDBBAAEAAAC
DDCCEABEEDCBEEDEAEEEEEAEEDBCEBEEADEAEEDAEBC
DEDEAEEDCEEAEEE”

“ECEABEADCAEDEEEECEADEEEEEDBAAEABDBBAAEAAAC DDCCEABEEDCBEEDEAEEEEEAEEDBCEBEEADEAEEDAEBC DEDEAEEDCEEAEEE”

character	A	B	C	D	E
frequency	0.2	0.1	0.1	0.15	0.45

Figure by Larry Nyhoff.

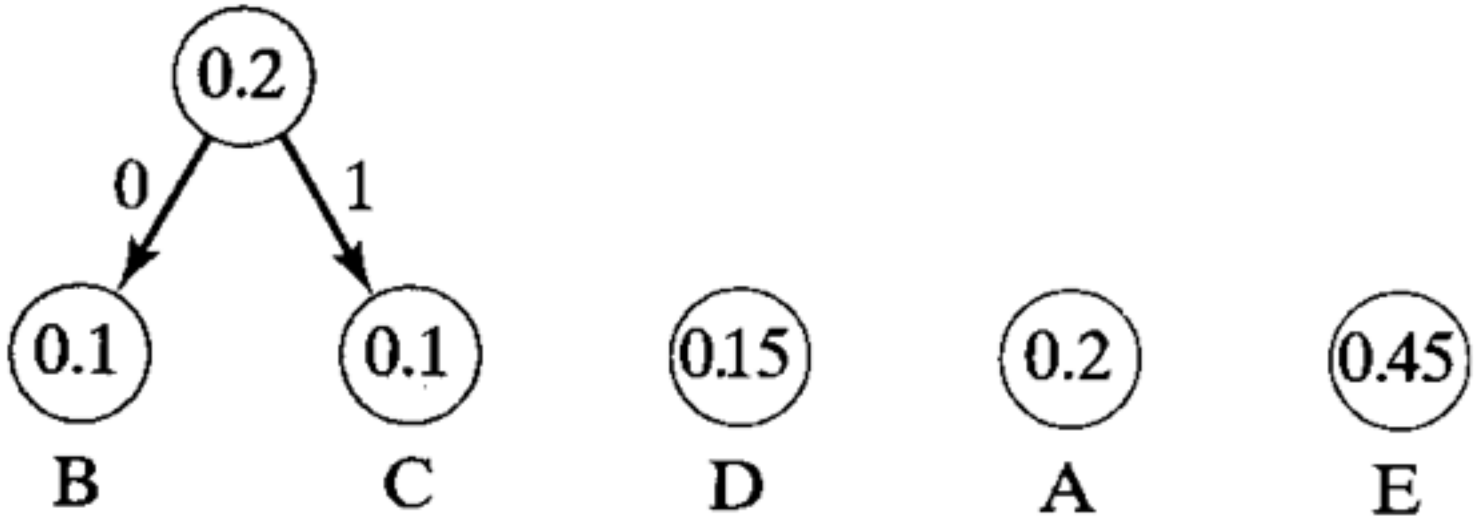


Figure by Larry Nyhoff.

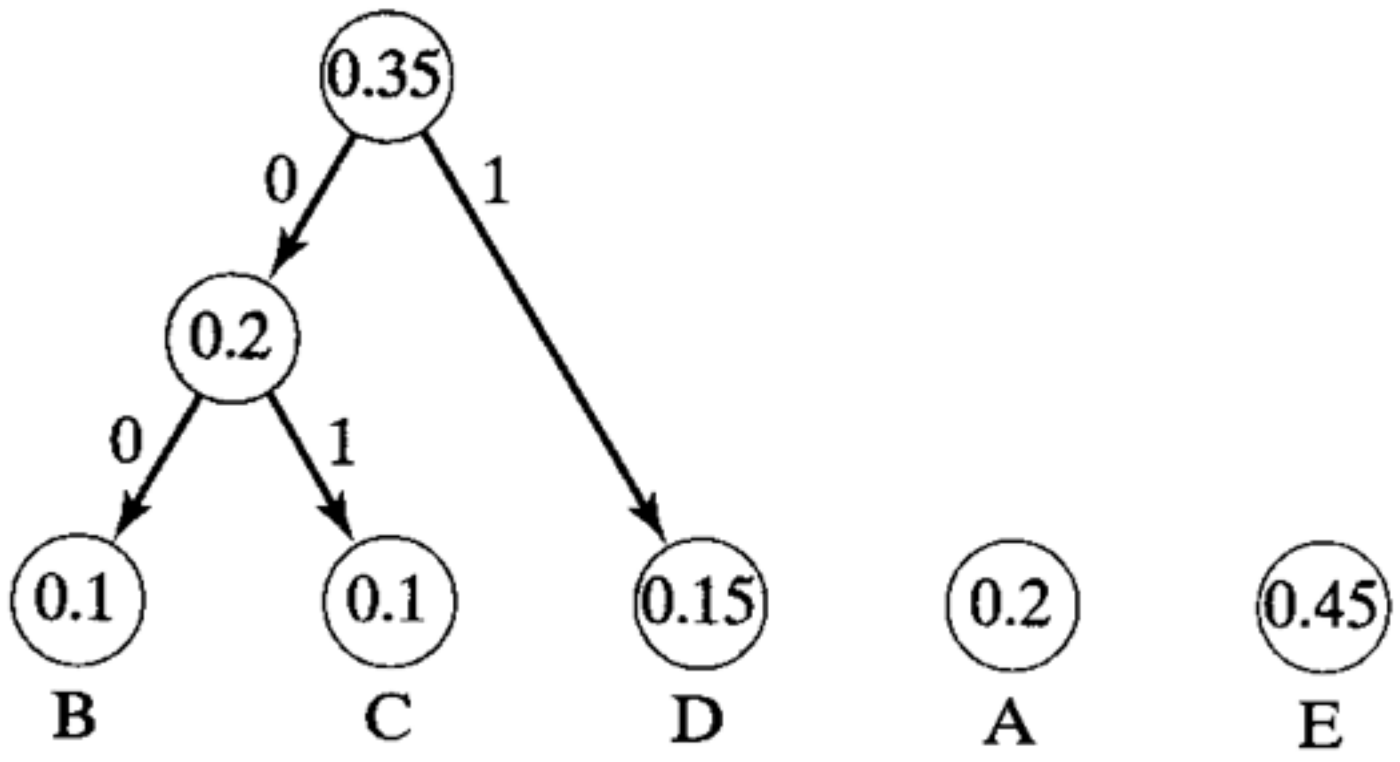


Figure by Larry Nyhoff.

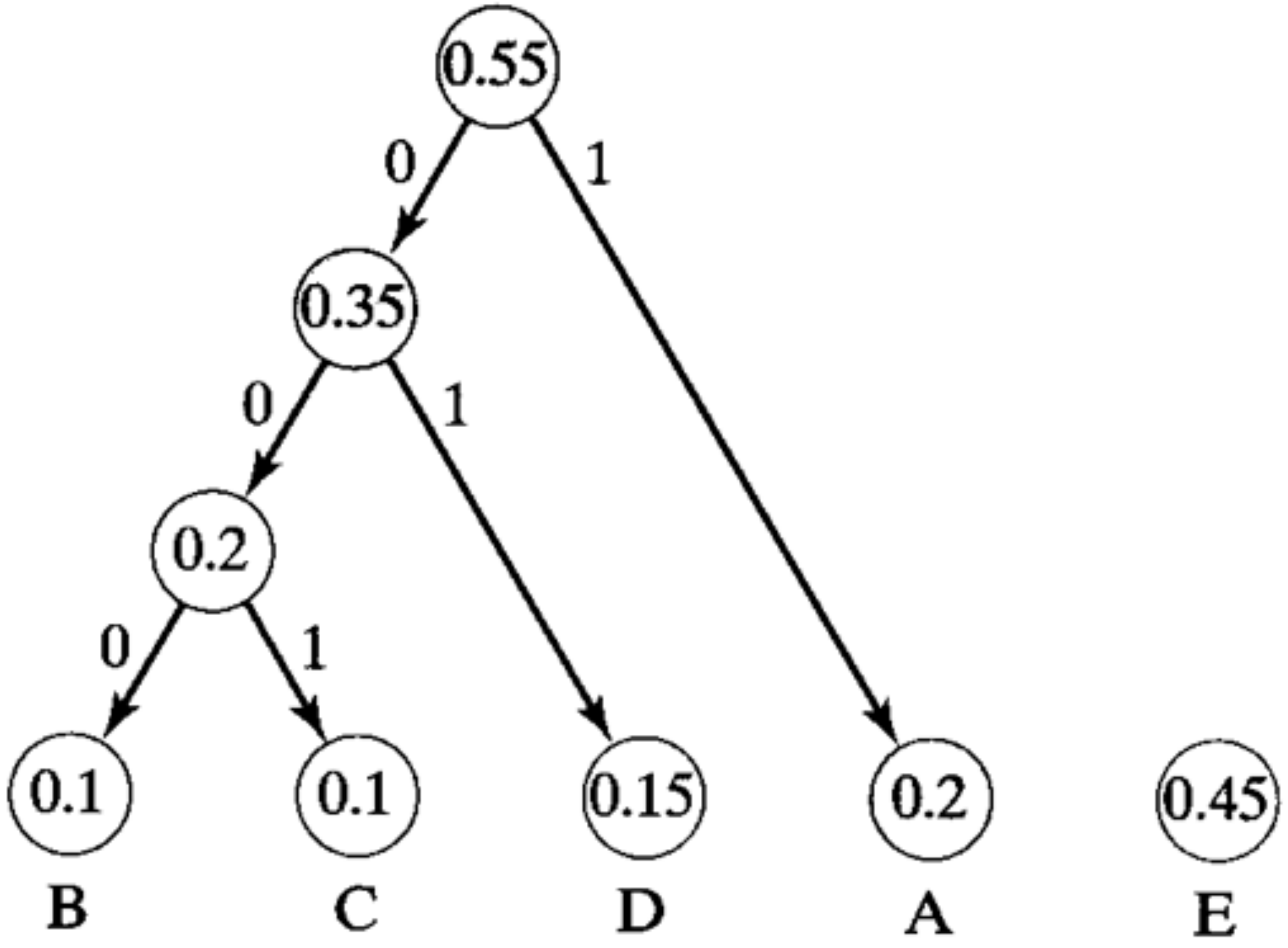


Figure by Larry Nyhoff.

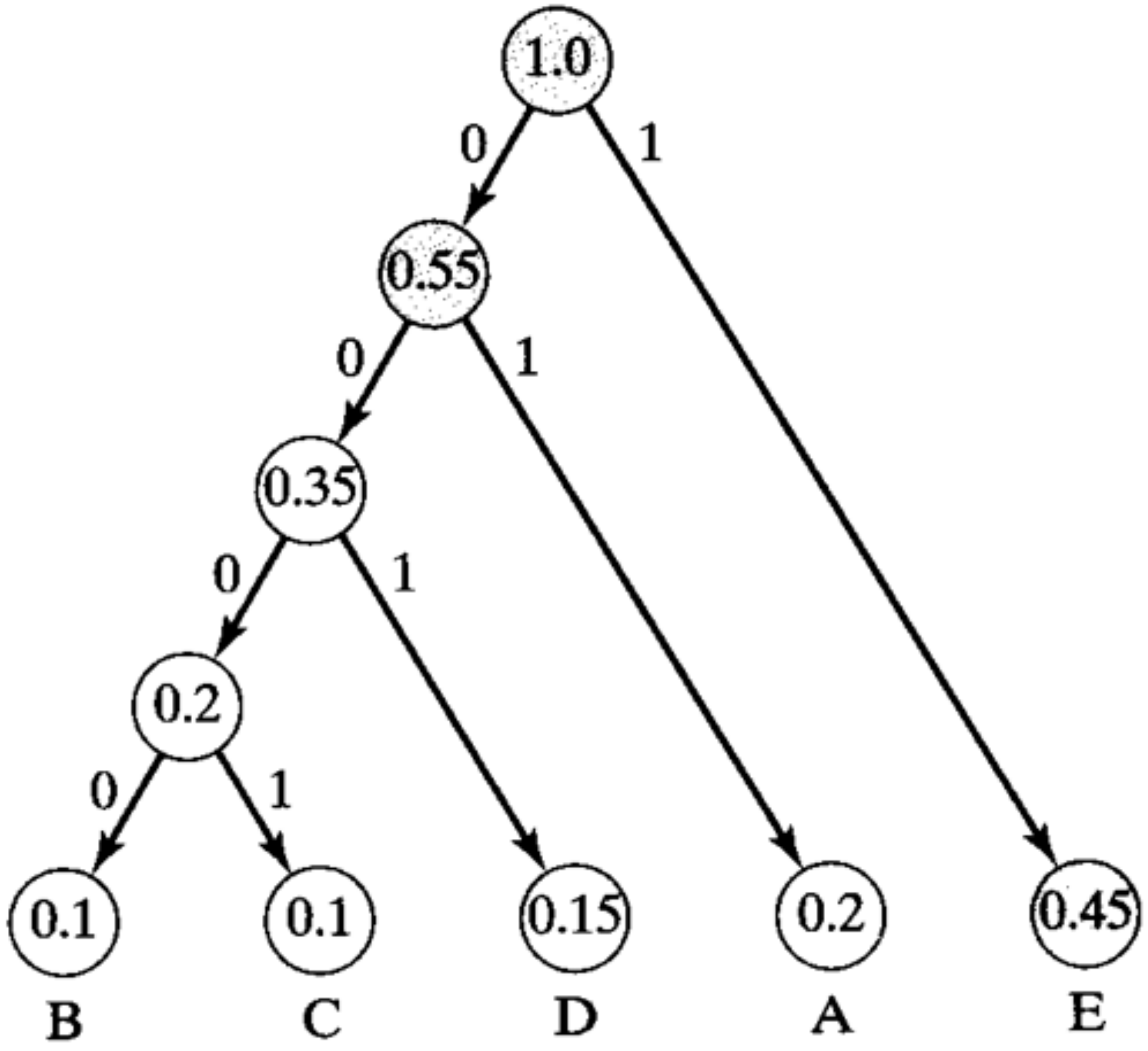


Figure by Larry Nyhoff.

“ECEABEADCAEDEEEECEADEEEEEDBAAEABDBBAAEAAAC
DDCCEABEEDCBEEDEAEEEEEAEEDBCEBEEADEAEEDAEBC
DEDEAEEDCEEAEEE”

character	A	B	C	D	E
frequency	0.2	0.1	0.1	0.15	0.45

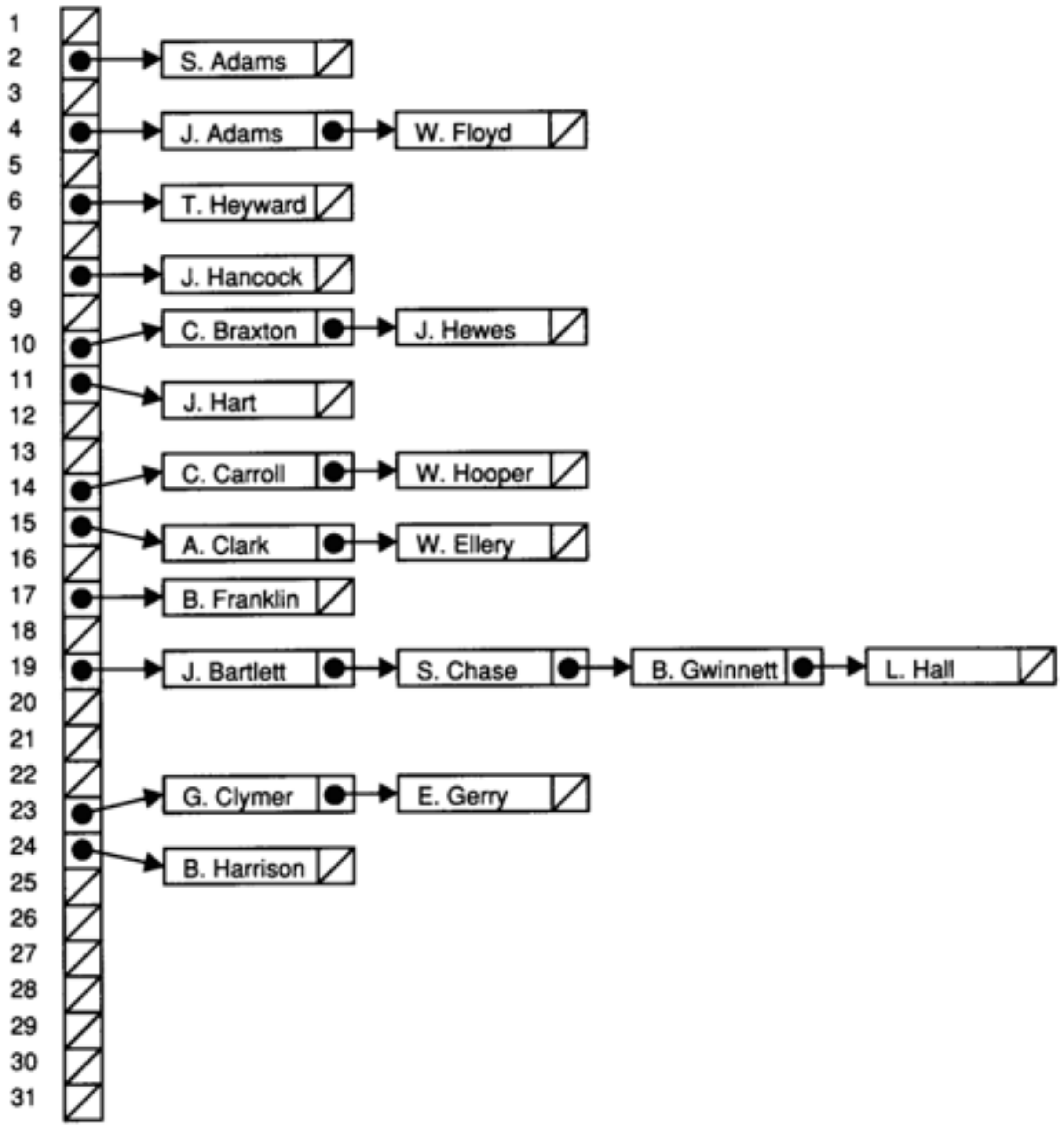
```
typedef struct node
{
    char symbol;
    float frequency;
    struct node* left;
    struct node* right;
}
node;
```

table[0]	
table[1]	
table[2]	
table[3]	
table[4]	
table[5]	
table[6]	
	⋮
table[24]	
table[25]	

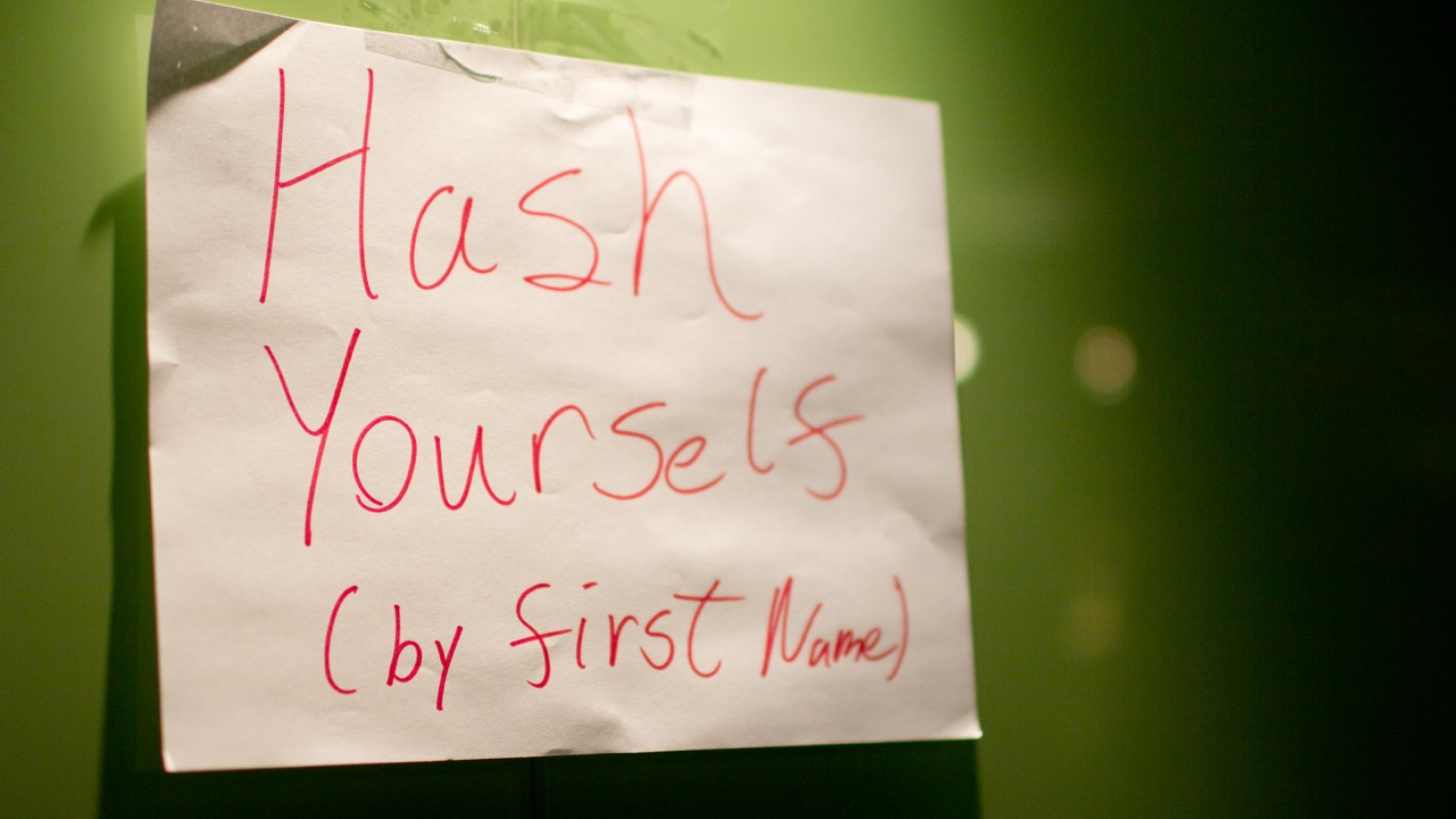
linear probing

table[0]	
table[1]	
table[2]	
table[3]	
table[4]	
table[5]	
table[6]	
	⋮
table[n-1]	

separate chaining




STAFF
BPG


Hash
Yourself
(by first Name)

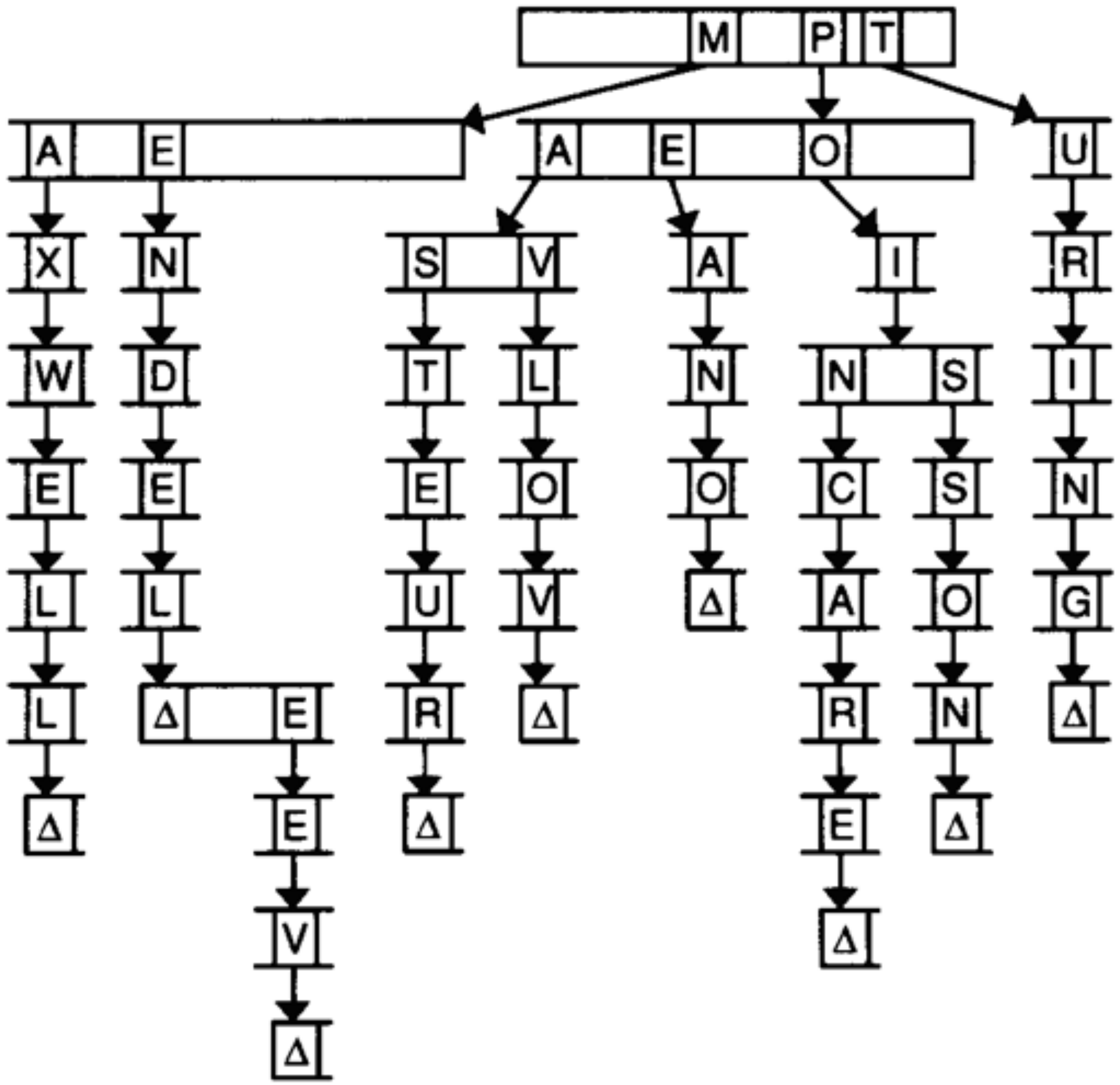


Figure from Lewis and Denenberg's Data Structures & Their Algorithms.

```
typedef struct node
{
    bool word;
    struct node* children[27];
}
node;
```