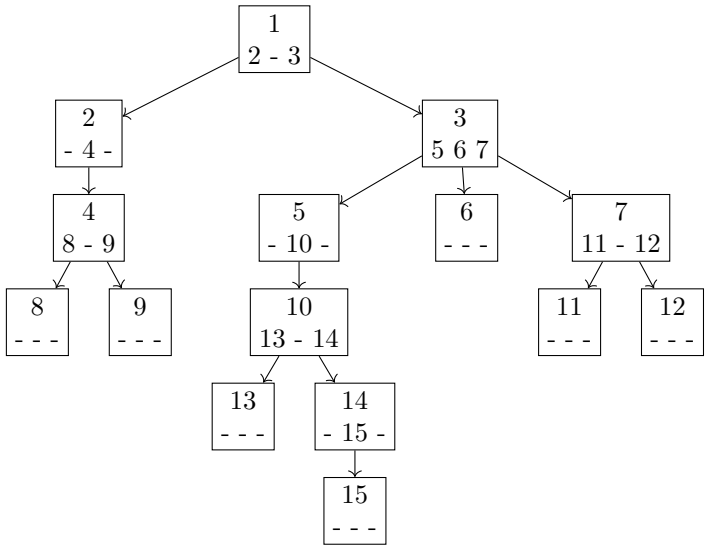
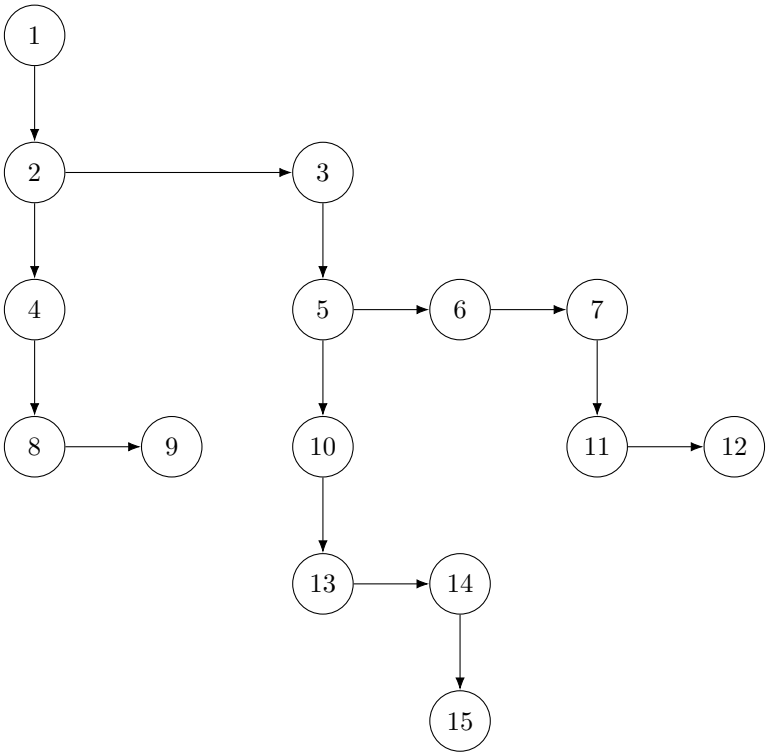
1.

node	leftmost child	right sibling
1	2	
2	4	3
3	5	
4	8	
5	10	6
6		7
7	11	
8		9
9		
10	13	
11		12
12		
13		14
14	15	
15		

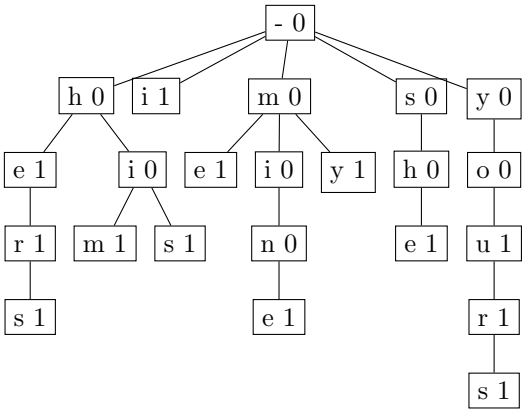
2.





The trie needs the memory of an `int` and an array of three `pNODE`s. The leftmost-child-right-sibling representation needs the memory of an `int` and two `pNODE`s.

`3.`



4.

a) There would be $10,000,000$ nodes in the trie.

b) There are 26 pointers for each node. Hence the trie requires $(2+4(26))10^7$ bytes.

c) There are 10^7 nodes and 2.6×10^8 pointers. Hence there are 2.5×10^8 `NULL` pointers, which takes $4 \times 2.5 \times 10^8$ bytes of memory.

5. There are two pointers for each node. Hence the tree requires $(2+4(2))10^7 = 10^8$ bytes. There are 2×10^7 pointers. We know that $n - 1 = 10^7 - 1$ pointers are non-`NULL`. Thus $(2 \times 10^7) - (10^7 - 1) = 10^7 + 1$ pointers are `NULL`, which means $4(10^7 + 1)$ bytes are taken.

6. We assume that we can select any representation we want. What seems to be the simplest is to hold whatever field of data and a pointer to the parent. The idea is that we go through all the parents c of x, asking if there exists a path from y upward to c. If there is a path then c must be the lowest common ancestor of x and y. Otherwise we find the lowest common ancestor of the parent of x, and y. We do so until x becomes `NULL`, meaning x is not in the same tree as y.

```
typedef struct NODE *pNODE;
struct NODE {
    int num;
    pNODE parent;
};

pNODE lca(pNODE x, pNODE y) {
    if (x == NULL) return NULL;
    else if (path_exists(x, y)) return x;
    else return lca(x->parent, y);
}

BOOLEAN path_exists(pNODE x, pNODE y) {
    if (y == NULL) return FALSE;
    else if (y == x) return TRUE;
    else return path_exists(x, y->parent);
}
```