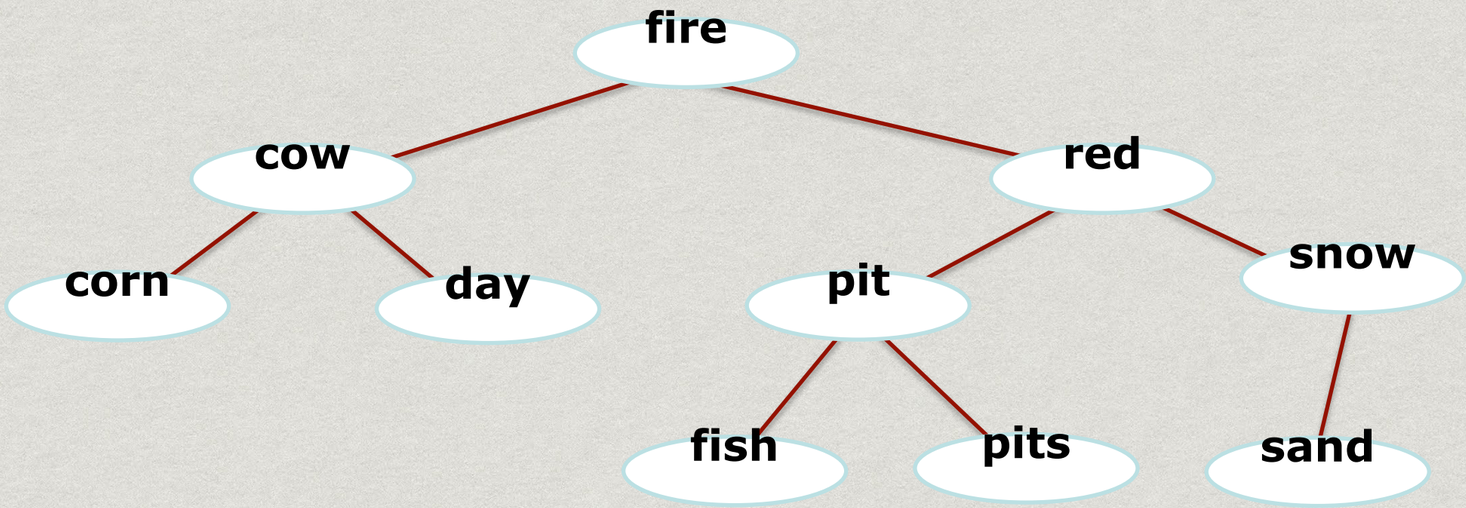
Searching in a dictionary

- How to implement word search?

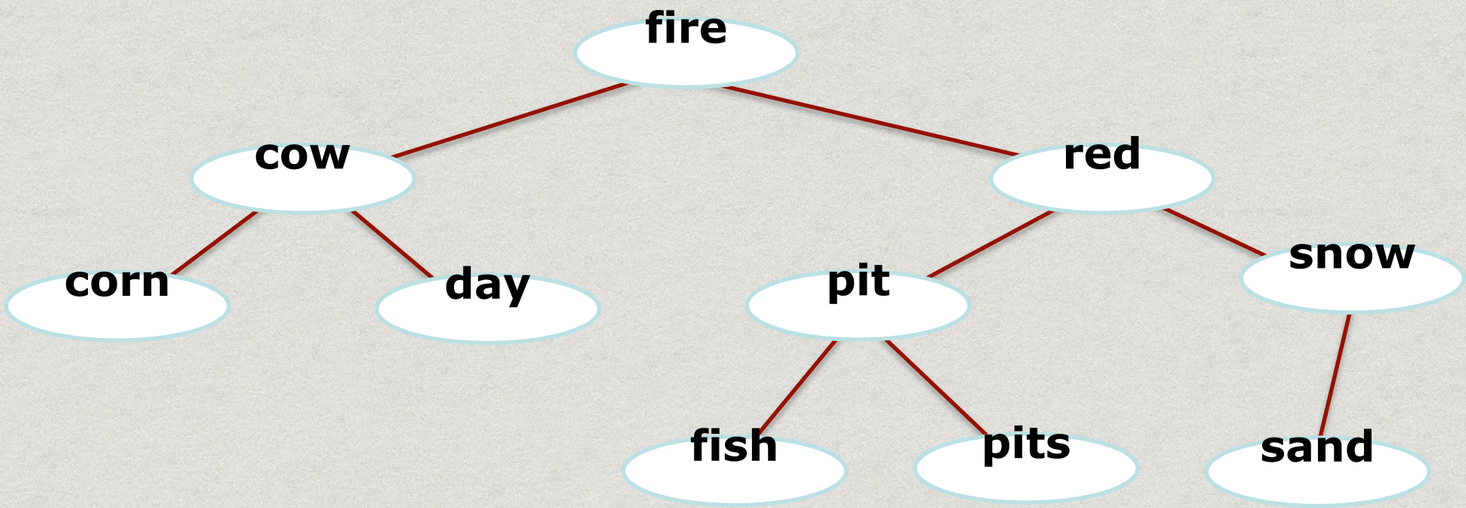
Option 1: tree



- How good is this?

WORST CASE O(#WORDS)
UNLESS WE CAN ACHIEVE GREAT BALANCE

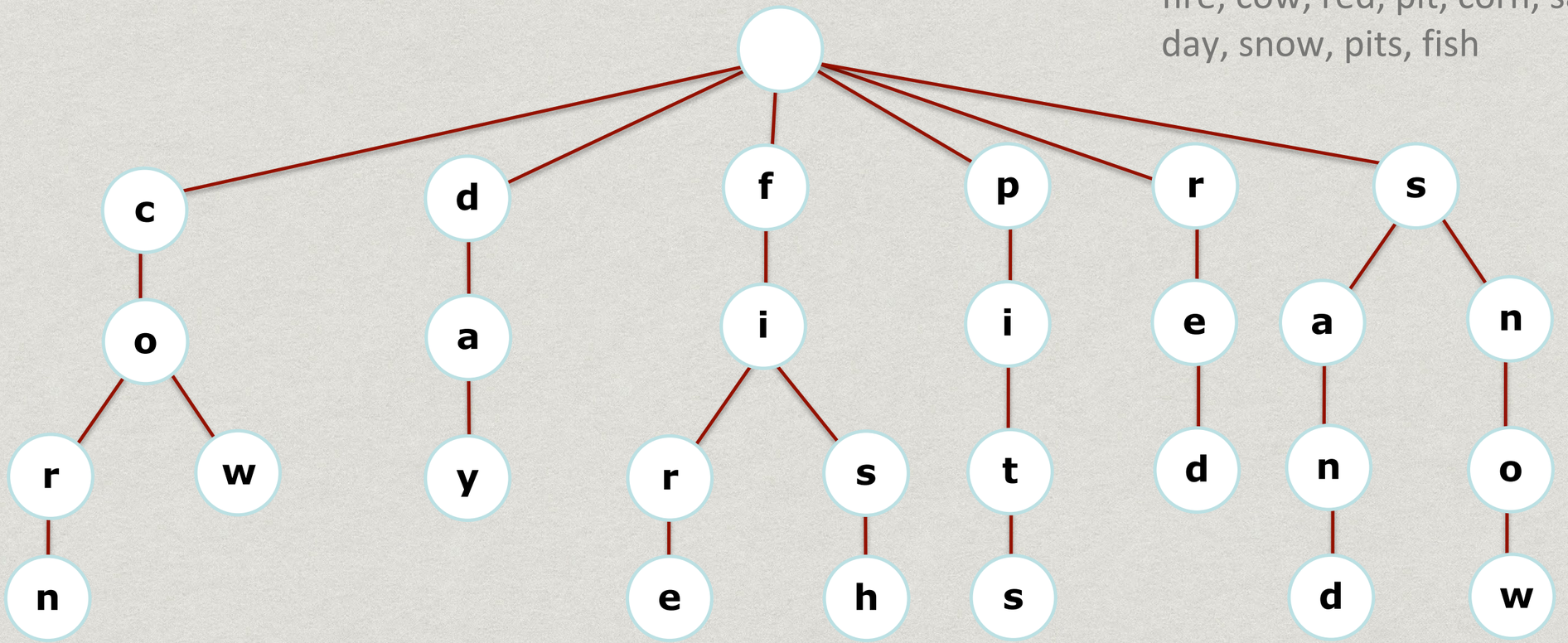
Can we do better?



- Let’s explore the advantages of our problem
  - Roughly 170.000 words, but only 26 characters!

Introducing Tries

fire, cow, red, pit, corn, sand, day, snow, pits, fish



- Tree with degree 26, each level has one letter

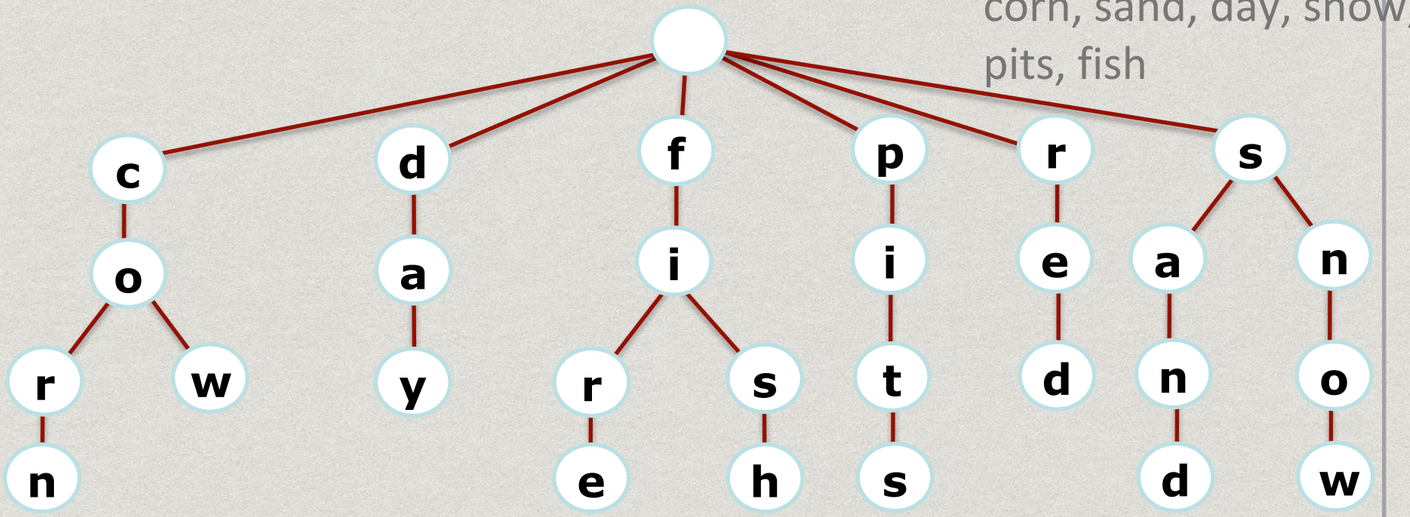
```c
struct TrieNode
{
    struct TrieNode *children[ALPHABET_SIZE];

    // isEndOfWord is true if the node represents
    // end of a word
    bool isEndOfWord;
};
```

```c
struct TrieNode *getNode(void)
{
    struct TrieNode *pNode = NULL;

    pNode = (struct TrieNode *)malloc(sizeof(struct TrieNode));

    if (pNode)
    {
        int i;

        pNode->isEndOfWord = false;

        for (i = 0; i < ALPHABET_SIZE; i++)
            pNode->children[i] = NULL;
    }

    return pNode;
}
```

Search

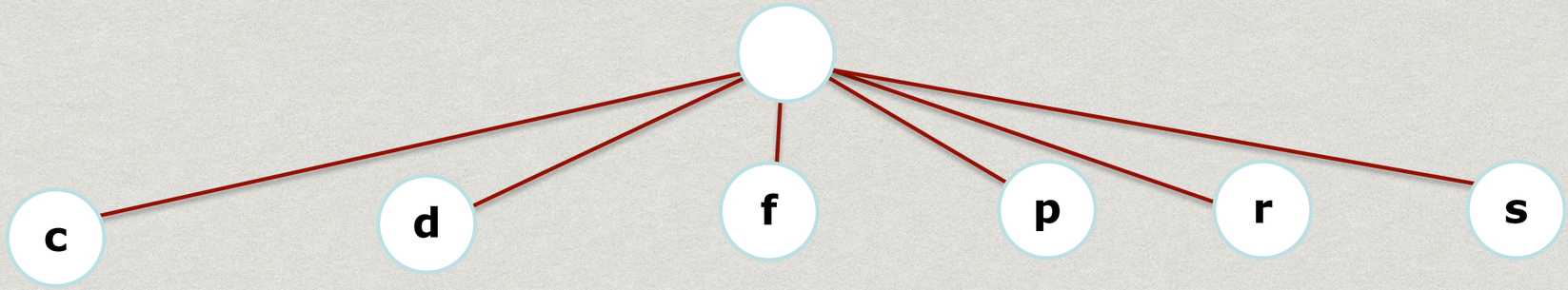
```
bool search(struct TrieNode *root, const char *key)
{
    int level;
    int length = strlen(key);
    int index;
    struct TrieNode *pCrawl = root;

    for (level = 0; level < length; level++)
    {
        index = CHAR_TO_INDEX(key[level]);

        if (!pCrawl->children[index])
            return false;

        pCrawl = pCrawl->children[index];
    }

    return (pCrawl->isEndOfWord);
}
```

fire, cow, red, pit, corn, sand, day, snow, pits, fish



PROBLEMS?

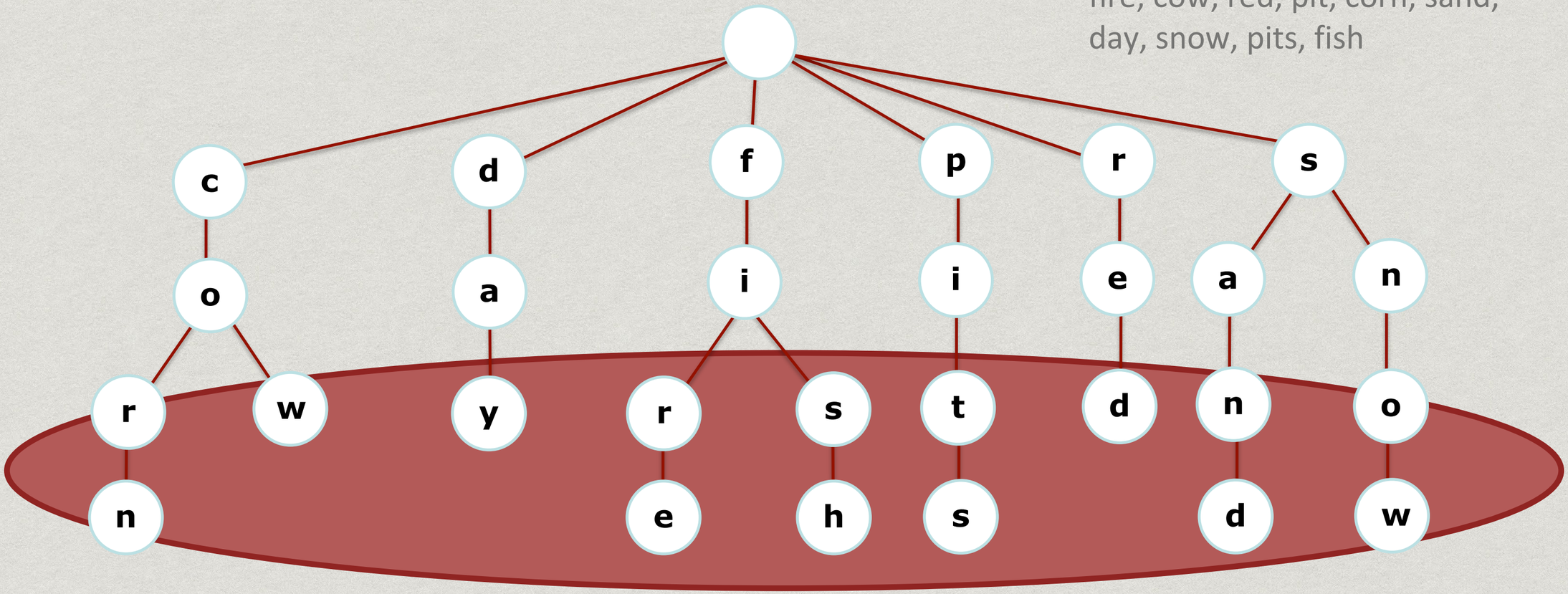
What does this mean



- In a binary tree k=2 -> Half of the pointers are NULL
- In an (English) Trie k=26 -> 96.1% of the pointers are NULL
- n nodes, k pointers each
  - nk pointers in total

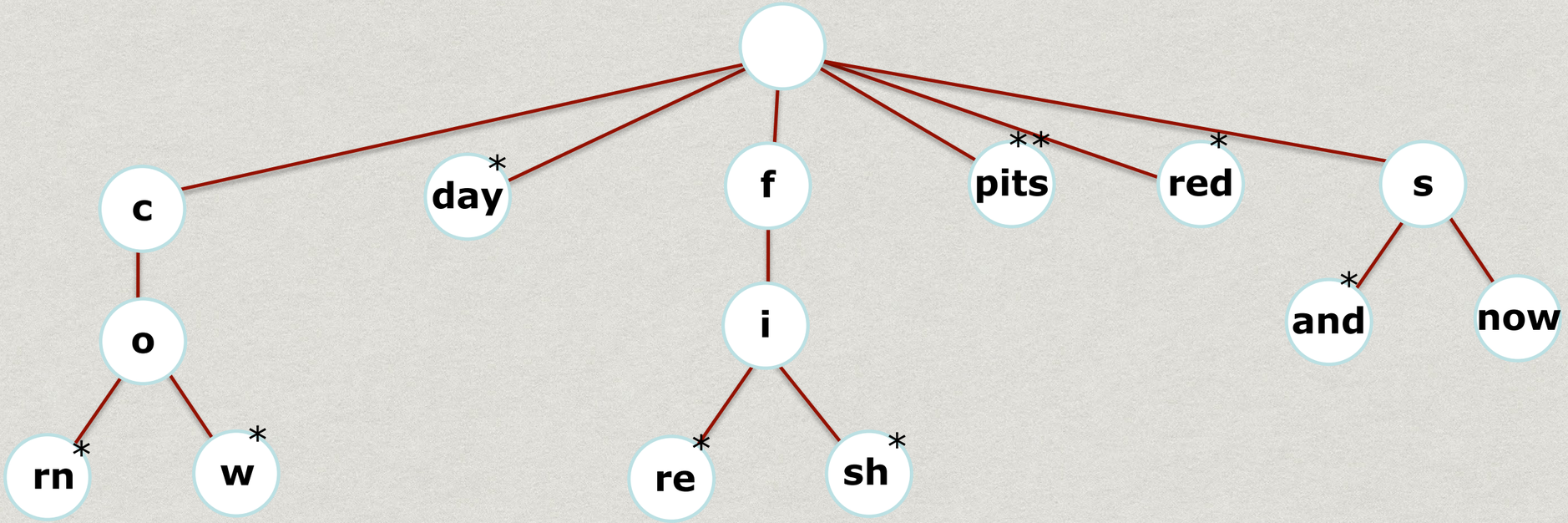
Can we save nodes?

fire, cow, red, pit, corn, sand, day, snow, pits, fish



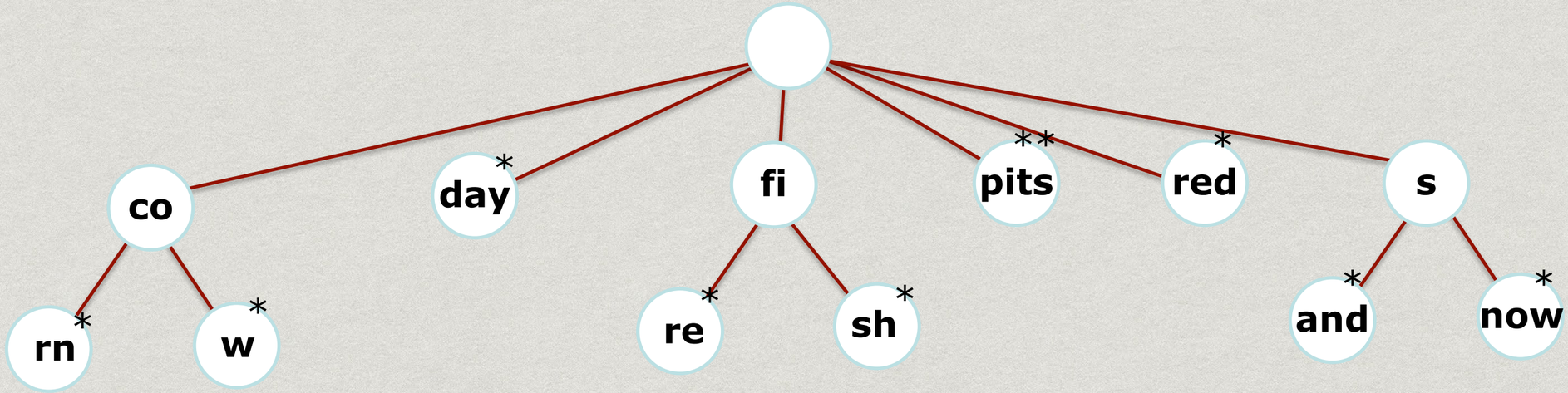
MANY PATHS AT THE END

Compressing tries



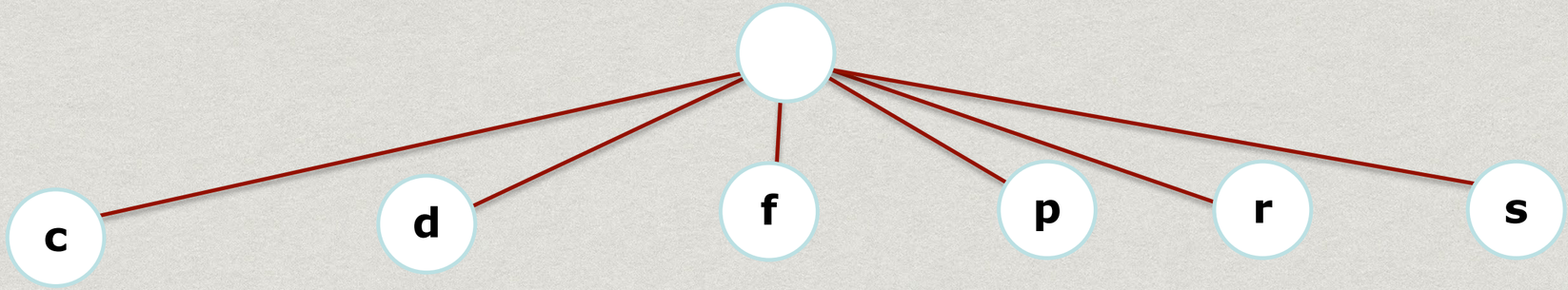
- Leaves without branches are collapsed into one

Let's take this one step forward



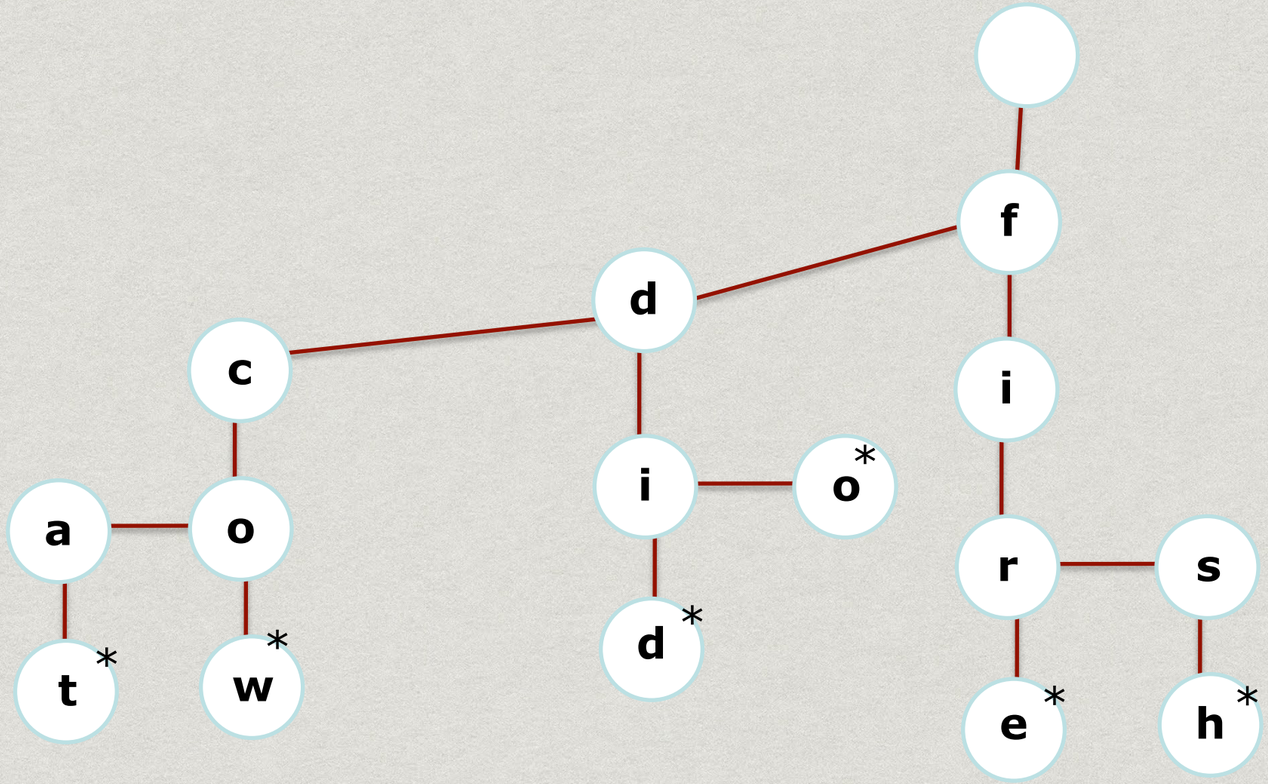
- Paths without branches can be collapsed as well
- Nodes must store additional characters

Mixing things up



- BSTs are great but have “slow search”
- Tries have fast search but consume much more memory
  - Can we find a middle ground?

BST+Tries= BSTries?



- Left child has smaller letter, Right child has larger letter
- **Middle** child has words that contain the letter