problems motivating classes, idioms, CS, ...

- Find a “path” from *white* to *house*
  - smart start stark stack slack black blank bland brand braid brain
  - ➤ shortest path, changing one letter at a time
  - ➤ issues? questions? ideas?

- Fast searching in the context of insertions (deletions)
  - ➤ binary search in array is very quick, # comparisons for one million strings? big-Oh? problems?
Word ladders

Each word that’s k-away appears on the queue before (k+1)-away words and before any word that’s n-away where n > k

How to remember the word ladder? How did we get on the queue?

Avoid putting words on more than once
Word ladders

- **operations/functions/behavior**
  - load words
  - find ladder
  - per “word” operations?
  - others?

- **In designing a class, some points to ponder**
  - behavior first, then state
  - collaborating classes
  - private/instance variables vs local/function variables
OO programming, concepts and patterns

- How to “watch” the ladder being formed, need a ladder-observer
- See example code for postfix observing
- Key idea to observers: inheritance
  - Sub class derives or inherits from the base class
  - The sub class “is-a” base class, can be used-as-a
  - Would facilitate word-tracking
  - Syntax is ugly in C++, but not too bad, key word is virtual
  - Virtual function calls resolved at runtime, not compile time
Fast search in the presence of insert (delete) ops

- big-Oh complexity of vector search, insert
  ➤ how does binary search work, what’s needed?
  ➤ what about insert, two parts: find and insert

- big-Oh complexity of linked-list search, insert

- combining the best features of both: fast search, fast insert, how are these realized in each data structure?
Binary search trees

● Use nodes like linked-lists, but with two pointers

```c
struct Tree {
    string info;
    Tree * left;
    Tree * right;
};
```

● If tree is balanced, with one comparison to root node, how much of the tree is eliminated?

● How are values added? Printed?
Tree insertion

- traverse left or right to find where value goes

```c
void insert(Tree * t, const string & value) {
    Tree * trail = 0;
    while (t != 0) {
        trail = t;
        if (value < t->info) t = t->left;
        else                 t = t->right;
    }
    trail->______ = ________;
}

● note: code is flawed
```
Tree insertion recursively

- insert a value into a tree, special cases?

```c
void insert(Tree * & t, const string & value) {
    if (t == 0) {
    
    } else if (value < t->info) insert(t->left, value);
    else insert(t->right, value);
}
```