The Search Problem

Suppose we have a collection of records, say R_1, R_2, …, R_N, each of which contains a unique key value, say k_i.

Given a particular key value, K, the *search problem* is to locate the record R_i such that k_i equals K (or to determine that no such record exists in the collection).

A *successful* search in one in which a matching record is located; an *unsuccessful* search is one in which no such record is found (or exists within the collection).

An *exact match search* is a search for the record that matches s specified key value; a *range search* is a search for all records whose key values fall within a specified set of key values.

Linear Search

In a *linear search*, the records are typically stored in a linear structure such as an array or a list, and the records are examined one-by-one in sequential order until a match is found or the last record has been examined and rejected.

It should be immediately obvious that the best case is that we will examine one record (and hence do one comparison of key values).

It should also be immediately obvious that the worst case is that we will examine all N records, performing N comparisons of key values, and find a match in the final comparison or find there is no match.

If we assume that each record is equally likely to match the specified key value, then it is probably intuitively obvious that the average case will involve going about half-way through the collection, and hence doing about N/2 comparisons of key values.

Binary Search

In a *binary search*, the records are typically stored <u>in sorted order</u> in a linear structure such as an array or a list, and the records are examined according to the following scheme:

```
0  set the active list to be the entire list
1  calculate the index of the middle-most element in the active list
2  compare the middle-most element to the target key value
3  if it's a match, stop
4  if the target key value is less than the key for the middle-most element, reset the active
   list to be the lower half of the current active list
5  otherwise, reset the active list to be the top half of the current active list
6  goto step 1
```

Binary Search Cost

For a rough analysis of the cost of binary search, note that each comparison that does not result a match does eliminate about half of the remaining elements.

So, if we start with N elements in the list, we can say:

```
After comparison #        approx # elements NOT eliminated
----------------------------------------------------
      1                            N/2
      2                            N/4
      3                            N/8
    . . .                         . . .
    log2N                           1
```

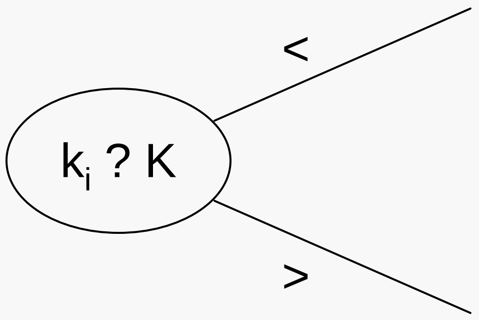
So, in the worst case, it appears that a binary search will require about $1 + \log_2 N$ comparisons of key values.

Lower Bound on Cost of Searching

So, the question seems to be, can we state a lower bound on the number of comparisons of key values that any possible search algorithm would require (say, in its worst case)?

Assume that we have a search algorithm that operates simply by comparing key values.

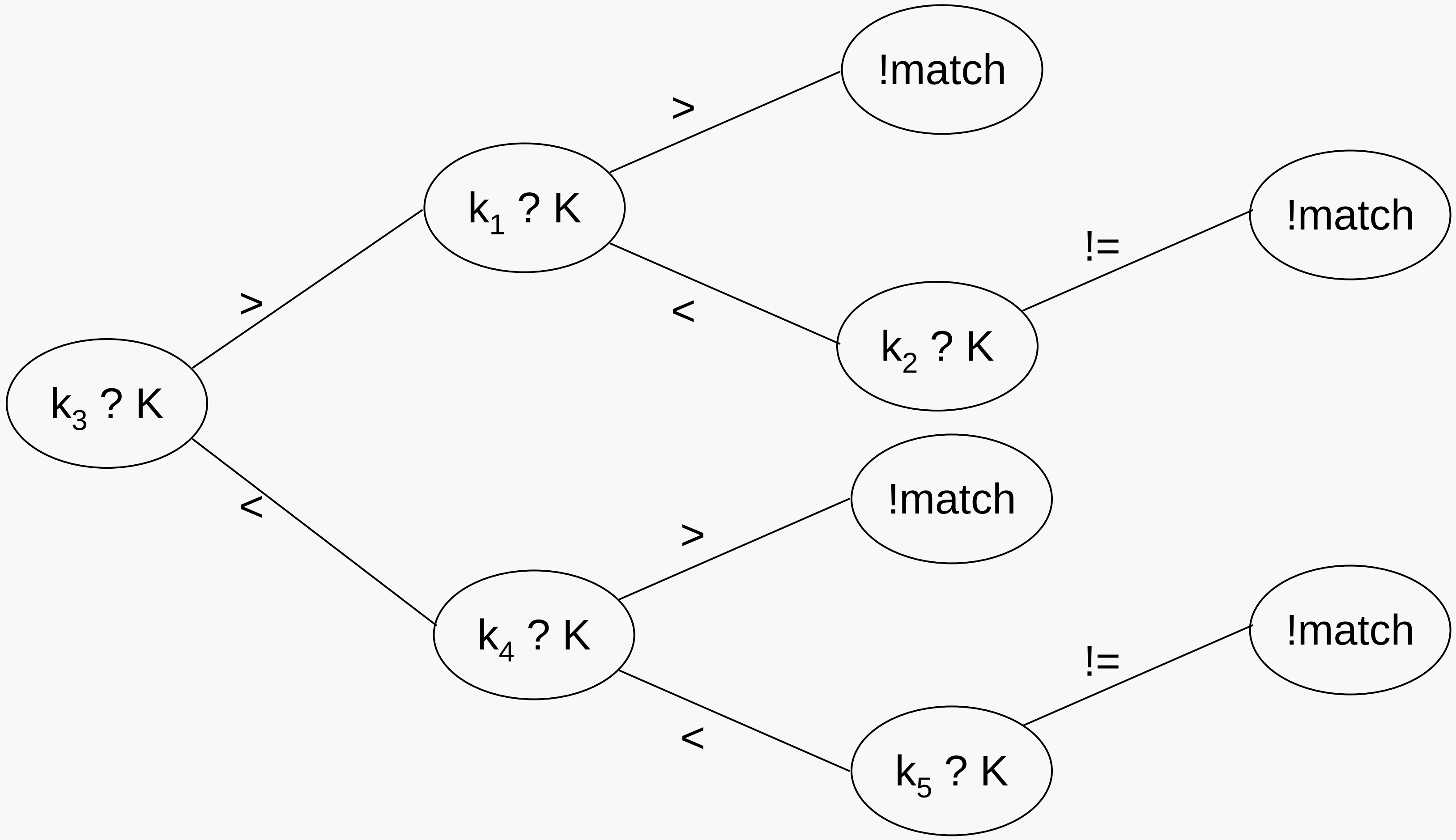
We can model the search process by using a binary tree, in which each node represents the result of comparing two key values:



If the comparison results in equality, then the search terminates, so we don't need an edge to model that.

Decision Tree Model of Searching

Consider binary search on a collection of 5 records:



So how does this help?

Decision Tree Analysis

The decision tree must contain 1 node for each possible conclusion of the search.

For a successful search, that requires N nodes.

For an unsuccessful search, that requires some additional nodes to represent the various ways in which failure may be detected. Without being too precise, say this is no more than N + 1 additional nodes.

Then:

For a successful search, the decision tree must have at least $\lceil \log (N + 1) \rceil$ levels.

So the worst case search would require $\lceil \log (N + 1) \rceil$ key comparisons.

For an unsuccessful search, the decision tree must have at least $1 + \lceil \log (N + 1) \rceil$ levels, so the worst case search would require $\lceil \log (N + 1) \rceil$ key comparisons.

So, to simplify, any search algorithm that operates strictly on key comparisons must require about log N key comparisons in its worst case.

Conclusions

We have two goals then:

First, we want the ability to achieve the theoretical minimum search cost with a flexible data structure --- and an array or other linear structure does not seem to qualify.

Second, we would like to have a way to beat the theoretical minimum cost… so we would like to know if there are ways to search that do not depend purely on key comparisons.