CS 112 – Introduction to Computing II
Wayne Snyder
Computer Science Department
Boston University

Today
State-space search

Next:
Refinements to MinMax Search
Reading is on the web site
State Space Search: Basic Notions

Let’s question one of our basic assumptions:

**DO NODES in TREES and HAVE TO EXIST?**

Well.... not really....many computations follow a “tree structure” without having to actually construct trees of explicit nodes....

---

**Binary Search in an array:**

- 22  33  44  55  66  77  88

**Search in a BST:**

---

Of course, we **DO** have a data structure to search, it just isn’t exactly a tree....

Are data structures necessary for search?
Here’s another example of binary search in a “tree structure” …. without any data structure to search…..

Suppose we want to find the inverse of a function f(n) which we know to be non-decreasing (monotonic); we can search the “tree” of possibilities without constructing a tree:

```java
int inverse(double r, double n, double lo, double hi) {
    if( Math.abs(n*Math.log2(n) - r) < 0.01)
        return n;
    double mid = (lo + hi)/2;
    if(f(mid) <= r)
        return inverse(r, n, lo, mid);
    else
        return inverse(r, n, mid, hi);
}
```

```java
double f(double x) {
    return x*Math.log2(x);
}
```

// called like this:
inverse(5, 5, 0, 5)
State Space Search: Basic Notions

Here’s another example of binary search in a “tree structure”…. without any data structure to search…..

Suppose we want to find the inverse of a function f(n) which we know to be non-decreasing (monotonic); we can search the “tree” of possibilities without constructing a tree:

```c
int inverse(double r, double n, double lo, double hi) {
    if( abs(n*Math.log2(n) – r) < 0.01)
        return n;
    double mid = (lo + hi)/2;
    if(f(mid) ≤ r)
        return inverse(r, n, lo, mid);
    else
        return inverse(r, n, mid, hi);
}

double f(double x) {
    return x *Math.log2(x);
}
```

// called like this:
inverse(5, 5, 0, 5)
State Space Search: Basic Notions

Here’s another example of binary search in a “tree structure” … without any data structure to search…..

Suppose we want to find the inverse of a function \( f(n) \) which we know to be non-decreasing (monotonic); we can search the “tree” of possibilities without constructing a tree:

```java
int inverse(double r, double n, double lo, double hi) {
    if( abs(n*Math.log2(n) – r) < 0.01)
        return n;
    double mid = (lo + hi)/2;
    if(f(mid) ≤ r)
        return inverse(r, mid, lo, mid);
    else
        return inverse(r, mid, mid, hi);
}

double f(double x) {
    return x *Math.log2(x);
}

// called like this:
inverse(5, 5, 0, 5)
```

<table>
<thead>
<tr>
<th>N</th>
<th>N*\text{log}(N)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>11.6096405</td>
</tr>
<tr>
<td>2.5</td>
<td>3.30482024</td>
</tr>
<tr>
<td>3.75</td>
<td>7.15083973</td>
</tr>
<tr>
<td>3.125</td>
<td>5.13705059</td>
</tr>
<tr>
<td>2.8125</td>
<td>4.19583683</td>
</tr>
<tr>
<td>2.96875</td>
<td>4.66050884</td>
</tr>
<tr>
<td>3.046875</td>
<td>4.89733455</td>
</tr>
<tr>
<td>3.0859375</td>
<td>5.01683588</td>
</tr>
<tr>
<td>3.06640625</td>
<td>4.95699548</td>
</tr>
<tr>
<td>3.07617188</td>
<td>4.98689332</td>
</tr>
<tr>
<td>3.08105469</td>
<td>5.00185902</td>
</tr>
</tbody>
</table>
DO NODES IN TREES and HAVE TO EXIST?  NO!

The parameters to the function call are in effect “nodes” in the computation, and are not explicit nodes linked by explicit pointers; but they are essentially the same. They are created by need as the computation proceeds.

NOTE: The tree is infinite!
The same idea can be used in many examples of search where:

- The search space of imaginary nodes is very large, perhaps infinite;
- You are going to search it only once (or a small number of times), to solve a particular problem.

This is called **State Space Search** where:

( imaginary) nodes == values of parameters in method calls == states of computation

You are searching among the various states of the problem, and you “create the children” of a node “by need” as you call a method recursively.

Many puzzles and games can be solved this way.....
Missionaries and Cannibals: Three missionaries and three cannibals are on the left bank of a river and have a boat in which they must cross to the right side, but the boat only holds two people and can not cross the river by itself. Furthermore, if the number of cannibals on either side of the bank is every greater than the number of missionaries, the cannibals will overwhelm and eat the missionaries. How can they all get to the other side without anyone spoiling his dinner?
State Space Search: Examples of State Space Problems

If we collapse the duplicates in the tree down into a graph (nodes and links which can point to anything, even make cycles among links), then:

Each node encodes the current STATE of the problem, and contains the number of missionaries and cannibals on each side: int[] S = {3, 3, 0, 0}. // 3M, 3C on left, 0M, 0C on right

The edges in the graph (it is undirected) are the ways that the boat can go to the other side and change the state of the problem.

You can search for the solution by doing traversal without actually constructing nodes:
The Eight Puzzle: Slide numbered tiles until they are in order:
The Eight Queens Problems: Place 8 queens on a chess board so that no one queen can attack another:

![Chessboard with queens](image)

*Figure 3.5* Almost a solution to the 8-queens problem. (Solution is left as an exercise.)