Exhaustive Search and Backtracking

Christopher R. Genovese

Department of Statistics & Data Science

Thu 30 Oct 2025
Session #18

Plan

Recap

Plan

Recap

Backtracking

Plan

Recap

Backtracking

Activity

Announcements

- In documents: lecture notes, fpc updated draft
- Look at `fp_concepts.examples.huffman` and `fp_concepts.examples.divide_conquer`
- **Reading**:
  - Zippers
    - You could have invented Zippers
    - Huet Zippers (optional but interesting)
    - Zippers as Derivatives (optional, for later)
  - Starting sequence on categories
    - What is Category Theory?
    - Definitions and Examples
    - What is a Functor? Part 1
    - What is a Functor? Part 2
    - Fibonacci Functor
    - Natural Transformations
- **Homework**:
  - **zippers** assignment due Tue 04 Nov.
  - Mini-assignment today

Plan

Recap

Backtracking

Activity

Revisiting Last Time

Look at `fp_concepts.examples.huffman` and `fp_concepts.examples.divide_conquer`.

For the first, consider design, use of language affordances, streaming versus str, and more.

For the second, remember our Divide-and-Conquer Framework.

Revisiting Last Time

```
data DandC problem solution = record DandC where
  isTrivial : problem -> Bool
  divide : problem -> Either problem (problem, problem)
  conquered : problem -> solution   -- trivial case or error thrown
  conquer : solution -> solution -> solution

BinaryTree a = Leaf a | Branch (BinaryTree a) a (BinaryTree a)

unfold : (b -> Either a (b, a, b)) -> b -> BinaryTree a
unfold gen seed =
  case gen seed of
    Left base          -> Leaf base
    Right (sl, p, sr) -> Branch (unfold gen s1) p (unfold gen s1)

fold : (a -> s) -> (s -> s -> s) -> BinaryTree a -> s
fold trivial combine (Leaf p) = trivial p
fold trivial combine (Branch p1 p2) =
  combine (fold trivial combine p1) (fold trivial combine p2)

divideAndConquer : (DandC p s) -> p -> s
divideAndConquer (DandC {divide, conquered, conquer}) initial =
  unfold divide initial |> fold conquered conquer
```

Plan

Recap

Backtracking

Activity

Backtracking and Exhaustive Search

Exhaustive search is a strategy for optimization problems where we consider every possible candidate and choose the best. This applies when strategies like greedy algorithms and divide-and-conquer are not enough.

```
exhaustive : Candidate -> List Component -> Candidate
exhaustive c_0 = minWith cost . candidates c_0
```

Heuristics in the details of candidate selection can make a big impact on the effectiveness of this strategy. In some cases, we might also be able assess the candidates in a way that lets us stop early with a "good enough" solution.

Backtracking and Exhaustive Search

Backtracking is a strategy for exhaustive search that is especially useful for solving *constraint satisfaction* problems. In constraint satisfaction, we have several constraints on possible solutions, and we must try possible solutions until we find one that satisfies all the constraints.

We search by trying all candidates (ideally with care). When we find one that is either apparently non-optimal or that violates the constrains, we *backtrack* to a previous state and try new candidates from there.

There are many problems that fit this mold. Some examples:

Room assignments We have k classes that have to be fit into n rooms. Each class can only fit into rooms large enough for it. There are a limited number of time slots available for each room. We must assign classes to rooms so all classes get a time slot and fit in their room.

Graph coloring Color each node in a graph with one of k colors, such that no node is connected to another node of the same color. (Includes map coloring.)

Logic programming Specify a problem as a set of logical expressions or rules and find values of variables which make the expressions true.

We will spend time with one of two more examples (Dominoes and Sudoku) today.

Plan

Recap

Backtracking

Activity

Activity

Main activity **dominoes**, in problem bank.

(Alternate: **sudoku**.)

The End