CSCI-UA.0380-001
Programming Challenges

Sean McIntyre

Class 04: Search

Today's agenda

- Homework discussion
- Lecture
- Break (~2:30-2:45pm)
- Practice (~2:45pm-4:15pm)
- Discussion of problems

Homework discussion

Splitting Numbers

- Create two numbers by dividing the bits up between the given number
- e.g., <u>11010110</u>

  n1 = 10010010

  n2 = 01000100

Newspaper

- You're given a list of prices for each character. Output the total price of a given article.

- Data structure problem

Searching

Searching

- What we'll look at today:
  - Iterative: Loops, Permutations, and Subsets
  - Recursive backtracking
  - State-space search

Searching with loops

- Problem:
  - Determine if *N* is a perfect square
    - 1 <= *N* <= 10,000

Searching with loops

- Problem:
  - Determine if *N* is a perfect square
    - 1 <= *N* <= 10,000
- Solution:
  - Math!

Searching with loops

- Problem:
  - Determine if *N* is a perfect square
    - 1 <= *N* <= 10,000
- Solution:
  - Math!
  - Complete search!

Searching with loops

- Problem:
  - Find all pairs of 5-digit numbers that between them use the digits 0 through 9 once such that *abcde* / *fghij* = *N*
    - 2 <= *N* <= 79
    - Each letter represents a different digit

Searching with loops

- Problem:
  - Find all pairs of 5-digit numbers that between them use the digits 0 through 9 once such that *abcde* / *fghij* = *N*
    - 2 <= *N* <= 79
    - Each letter represents a different digit
- Solution:
  - Complete search!

Searching with permutations

- Problem:
  - *n* friends go to a movie and sit in a row with *n* consecutive open seats.
  - There are *m* seating constraints, i.e., two people *a* and *b* must be at most (least) *c* seats apart
  - $0 < n <= 8$ and $0 <= m <= 20$

Searching with permutations

- Problem:
  - *n* friends go to a movie and sit in a row with *n* consecutive open seats.
  - There are *m* seating constraints, i.e., two people *a* and *b* must be at most (least) *c* seats apart
  - $0 < n <= 8$ and $0 <= m <= 20$
- Solution:
  - Try all permutations / complete search!

Searching with combinations

- Problem:
  - A dam has 1 <= *n* <= 20 water gates to let out water when necessary. Using each gate has a flow rate and damage cost when used.
  - Open the gates so that a total flow rate is achieved at minimal total damage cost.

Searching with combinations

- Problem:
  - A dam has 1 <= *n* <= 20 water gates to let out water when necessary. Using each gate has a flow rate and damage cost when used.
  - Open the gates so that a total flow rate is achieved at minimal total damage cost.
- Solution:
  - Try all combinations / complete search!

Searching with recursive backtracking

- Problem:
  - Place 8 queens on an 8x8 chessboard and count the number of solutions with a queen at (a, b)
  - No queens can attack each other
  - The “N queens problem”

Searching with recursive backtracking

- Problem:
  - Place 8 queens on an 8x8 chessboard and count the number of solutions with a queen at (*a*, *b*)
  - No queens can attack each other
  - The “N queens problem”
- Naive solution:
  - 8x8 = 64 cells, choose 8 of them and test.
    - 64 choose 8 ~= 4 billion = too much

Searching with recursive backtracking

- Pruning the search space:
  - Two queens cannot be in the same column, so place a queen in each column
    - Represented as a set of digits 1-8. The index of the digit is the column, the digit is the row.
    - 8^8 ~= 17 million = better

Searching with recursive backtracking

- Pruning the search space:
  - Two queens cannot be in the same column, so place a queen in each column
    - Represented as a set of digits 0-7. The index of the digit is the column, the digit is the row.
    - 8^8 ~= 17 million = better
  - Two queens cannot be in the same <u>row</u>
    - Represented as a set of digits 1-8, <u>each digit unique</u>.
    - 8! = 40,320 = good

Searching with recursive backtracking

- Pruning the search space:
  - Two queens cannot be on the same diagonal
    - Reduces the search space further.
    - Solutions built with recursive backtracking can preemptively ignore placing queens on diagonals

Searching with recursive backtracking

```
int queens[] = new int[8]; int a, b;

boolean place(int r, int c) {
  for (int prev = 0; prev < c; prev++) { // Check previously placed queens
    if (queens[prev] == r || (abs(queens[prev] – r) == abs(prev – c))) {
        return false; // If here then previous queen attacks (r, c)
    }
  }
  return true;
}

void backtrack(int c) {
  if (c == 8) {
    if (queens[b] == a) printSolution(queens);
    return;
  }

  for (int r = 0; r < 8; r++) { // Try all possible rows for this column
    if (place(r, c)) {  // True if (r, c) is a valid placement for a queen
      queens[c] = r;    // Place a queen here
      backtrack(c + 1); // Recurse
    }
  }
}
```

Searching the state space

- Problem:
  - Given *n* paragraphs from 1 to *n*, arrange them in order of 1, 2, …, *n*
  - Operations: cut and paste
    - You cannot cut twice before pasting, but you can cut several paragraphs in a row
  - What is the minimum number of steps?

Searching the state space

- Solution:
  - Breadth-first search (BFS)
  - $O(|V| + |E|)$
  - Plain breadth-first search is 9! * 9^3 = 265 million = too much

Searching the state space

- Solution:
  - Meet-in-the-middle BFS
  - Two breadth-first searches from either end
    - The graphs only need to be 4 deep, so the search space is significantly pruned
    - Generate all states that can be reached from either end
    - The moment a state is shared, add the depth from the start point and end point

Searching the state space

- Solution:
  - This example is intended as an illustration of how to analyze the runtime of a complete search
  - And pruning

Practice

For next class

- Readings:
  - Sections 3.1 and 3.2
- Exercises:
  - 3x problems + 2x bonus problems on website
- Next week:
  - More on BFS, greedy, and binary search