Lecture 24: Algorithmic Puzzles

- Two-Dimensional Search
- Reversal of Sort
- Lemonade Stand Placement
- Robotic Coin Collection with a twist

Two-Dimensional Search

\[
\begin{array}{cccc}
& & & \\
& & & \\
& & & \\
& & & \\
& & & \\
& & & \\
& & & \\
& & & \\
& & & \\
& & & \\
\end{array}
\]

Increasing order

\[n \times n\] set of cards with numbers hidden.

Want to search for a given number.

How to search to minimize worst-case number of cards to be turned over?
**Strategy 1**

Can we do some sort of binary search to get \( O(\log n^2) = O(\log n) \) algorithm?

Unfortunately not, 2D array is not sorted.

\[
\begin{array}{cccccc}
1 & 2 & 6 & 4 & 5 & 9 \\
4 & 5 & 9 & 1 & 13
\end{array}
\]

**Strategy 2**

\[
\begin{array}{cccccc}
\text{Incr} & n \text{ cards} & & & & \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow
\end{array}
\]

Searching for \( X \).

Look at top right number = TR

If \( X < TR \)

- Can't be in last column, move left to next column

Else if \( X > TR \)

- Can't be in first row, move down to next row

Else FOUND.

Worst case 2n-1 cards flipped.

\( k = n^2 \) \( O(\sqrt{k}) \) search algorithm.
Reversal of Sort

Suppose we have a sorted list of $n$ numbers in increasing order

\[ 1 \ 2 \ 3 \ \ldots \ n \]

We want to sort in DECREASING order using only pairwise swaps of numbers in $i$ and $i+2$ positions, for any $i$.

Is this possible? If so, how?

Observation

It won't work for even $n$ because we can never make number in position 1 go to an even location.

If $n$ is even, cannot get it to position $n$.

What about odd $n$?
**Strategy**

It will work for odd $n$.

Run insertion sort on $1, 3, \ldots, n$.

This will give us these numbers in the correct positions in reverse order – insertion sort only swaps adjacent numbers.

Then run insertion sort on $2, 4, \ldots, n-1$.

**Lemonade Stand Placement**

$n$ houses on a grid in locations $(x_i, y_i)$

Want to place a lemonade stand at $(x, y)$ such that

$$\sum_{i=1}^{n} |x_i-x| + |y_i-x|$$

is minimized.

How?
Observation
Can independently minimize $x$ then $y$.
Assume $x_1, x_2, \ldots, x_n$ in non-decreasing order (can be equal).
Assume $n$ is odd (even case similar).
Look at $n = 3$ case $x_1, x_2, x_3$.
Ignore $x_2$ for now. $x_1 \uparrow x_3$

$|x_1 - x| + |x_3 - x|$ is constant!

How to minimize $|x_2 - x|$? Choose $x = x_2$.

Strategy
Have:

$|x_1 - x| + |x_2 - x| + \ldots + |x_{n-1} - x| + |x_n - x|$

Write as:

$(|x_1 - x| + |x_n - x|) +
\left( |x_2 - x| + |x_{n-1} - x| \right) +
\ldots +
|x_{\lceil n/2 \rceil} - x|$

$x$ between $x_1$ and $x_n$ minimizes this

$x = x_{\lfloor n/2 \rfloor}$ minimizes this and all the other terms!

Just pick median $x_{\lfloor n/2 \rfloor}$ value

$\smile$
Robotic Coin Collection revisited

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>G</td>
</tr>
<tr>
<td>G</td>
<td>G</td>
<td>G</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>G</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Robots start at top left and move to bottom right only making right/down moves.

Minimize # robots required to pick up ALL coins.

Dynamic Programming Strategy

Send 1st robot and collect maximum # coins. Coins are removed.
Second robot collects max # coins ...
 Does this minimize the # robots? NO

In example above, first robot collects 7 coins: 5 in row 2 and coins in column 8
Can do better!

(2,1) (2,2) (2,4) (4,4) (4,5) (5,8) 6 coins
(1,3) (1,6) (2,7) (2,8) (3,8) 5 coins
Observation

Two occupied cells are disjoint if one is to the bottom left of another. Cells that are disjoint from each other need different robots.

The largest maximal set of disjoint cells is a lower bound on the number of robots required, as each cell in the set is disjoint from all others.

Algorithm

Given the lower bound above, an algorithm that always achieves this lower bound is optimal. Algorithm finds largest maximal sets, and picks the bottom-most coin from each of the sets. This clearly reduces the largest maximal set size by 1. Repeat.

Need to show that we can always pick the bottom-most coins from each of the largest maximal sets.
Proof Sketch

You are not responsible for this material. It is 6.046-level material and is meant to be a preview 😊

To show the claim, we need to show that the bottom-most coins in each of the largest maximal sets are not disjoint. If coins are at \((x_1, y_1), (x_2, y_2)\ldots (x_k, y_k)\)

If coins are at \((x_1, y_1), (x_2, y_2)\ldots (x_k, y_k)\)

If coins are at \((x_1, y_1), (x_2, y_2)\ldots (x_k, y_k)\)

If coins are at \((x_1, y_1), (x_2, y_2)\ldots (x_k, y_k)\)

Without loss of generality, assume for the \(k\) largest maximal sets, assume without loss of generality that \(x_1 \leq x_2 \ldots \leq x_k\).

Without loss of generality that \(x_1 \leq x_2 \ldots \leq x_k\).

Without loss of generality that \(x_1 \leq x_2 \ldots \leq x_k\).

Without loss of generality that \(x_1 \leq x_2 \ldots \leq x_k\).

This means that \(y_1 > y_2 > \ldots y_k\) else we will be able to add a coin to one of the "largest" sets resulting in a contradiction.

A robot will therefore be able to pick up 

\((x_1, y_1), (x_2, y_2)\ldots (x_k, y_k)\)

Reducing the size of the largest sets by 1.