Recitation 17

Treasureship!

The new boardgame Treasureship is played by placing 2×1 ships within a $2 \times n$ rectangular grid. Just as in regular battleship, each 2×1 ship can be placed either horizontally or vertically, occupying exactly 2 grid squares, and each grid square may only be occupied by a single ship. Each grid square has a positive or negative integer value, representing how much treasure may be acquired or lost at that square. You may place as many ships on the board as you like, with the score of a placement of ships being the value sum of all grid squares covered by ships. Design an efficient dynamic-programming algorithm to determine a placement of ships that will maximize your total score.

Solution:

1. **Subproblems**

   - The game board has n columns of height 2 (alternatively 2 rows of width n)
   - Let $v(x, y)$ denote the grid value at row y column x, for $y \in \{1, 2\}$ and $x \in \{1, \ldots, n\}$
   - Guess how to cover the right-most square(s) in an optimal placement
   - Can either:
     - not cover,
     - place a ship to cover vertically, or
     - place a ship to cover horizontally.
   - After choosing an option, the remainder of the board may not be a rectangle
   - Right side of board looks like one of the following cases:

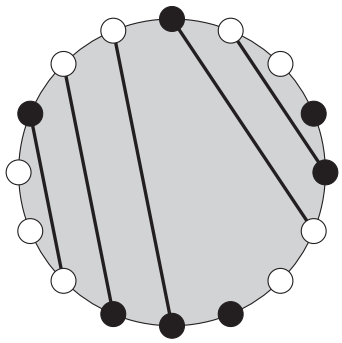
```
(0) #####   (+1) #####   (-1) ####    (+2) #####   (-2) ###    row 2
    #####        ####         #####        ###          #####  row 1
```

   - Exists optimal placement where no two ships aligned horizontally on top of each other
   - Proof: cover instead by two vertical ships next to each other!
   - So actually only need first three cases above: 0, $+1$, -1
   - Let $s(i, j)$ represent game board subset containing columns 1 to i of row 1, and columns 1 to $i + j$ of row 2, for $j \in \{0, +1, -1\}$
   - $\boxed{x(i, j)\text{: maximum score, only placing ships on board subset } s(i, j)}$
   - for $i \in \{0, \ldots, n\}$, $j \in \{0, +1, -1\}$

2. **Relate**
   - If $j = +1$, can cover right-most square with horizontal ship or leave empty
   - If $j = -1$, can cover right-most square with horizontal ship or leave empty
   - If $j = 0$, can cover column i with vertical ship or not cover one of right-most squares
   - $x(i,j) = \begin{cases} \max\{v(i,1) + v(i-1,1) + x(i-2,+1), x(i-1,0)\} & \text{if } j = -1 \\ \max\{v(i+1,2) + v(i,2) + x(i,-1), x(i,0)\} & \text{if } j = +1 \\ \max\{v(i,1) + v(i,2) + x(i-1,0), x(i,-1), x(i-1,+1)\} & \text{if } j = 0 \end{cases}$
3. **Topo**
   - Subproblems $x(i,j)$ only depend on strictly smaller $2i + j$, so acyclic.
4. **Base**
   - $s(i,j)$ contains $2i + j$ grid squares
   - $x(i,j) = 0$ if $2i + j < 2$ (can't place a ship if fewer than 2 squares!)
5. **Original**
   - Solution is $x(n,0)$, the maximum considering all grid squares.
   - Store parent pointers to reconstruct ship locations
6. **Time**
   - # subproblems: $O(n)$
   - work per subproblem $O(1)$
   - $O(n)$ running time

Wafer Power

A start-up is working on a new electronic circuit design for highly-parallel computing. Evenly-spaced along the perimeter of a circular wafer sits n ports for either a power source or a computing unit. Each computing unit needs energy from a power source, transferred between ports via a wire etched into the top surface of the wafer. However, if a computing unit is connected to a power source that is too close, the power can overload and destroy the circuit. Further, no two etched wires may cross each other. The circuit designer needs an automated way to evaluate the effectiveness of different designs, and has asked you for help. Given an arrangement of power sources and computing units plugged into the n ports, describe an $O(n^3)$-time dynamic programming algorithm to match computing units to power sources by etching non-crossing wires between them onto the surface of the wafer, in order to maximize the number of powered computing units, where wires may not connect two adjacent ports along the perimeter. Below is an example wafer, with non-crossing wires connecting computing units (white) to power sources (black).



Solution:

1. **Subproblems**

   - Let $(a_1, \ldots, a_n)$ be the ports cyclically ordered counter-clockwise around the wafer, where ports a_1 and a_n are adjacent
   - Let a_i be True if the port is a computing unit, and False if it is a power source
   - Want to match opposite ports connected by non-crossing wires
   - If match across the wafer, need to independently match ports on either side (substrings!)
   - $x(i, j)$: maximum number of matchings, restricting to ports a_k for all $k \in \{i, \ldots, j\}$
   - for $i \in \{1, \ldots, n\}$, $j \in \{i-1, \ldots, n\}$
   - $j - i + 1$ is number of ports in substring (allow $j = i - 1$ as an empty substring)

2. **Relate**
   - Guess what port to match with **first port** in substring. Either:
     - first port does not match with anything, try to match the rest;
     - first port matches a port in the middle, try to match each side independently.
   - Non-adjacency condition restricts possible matchings between i and some port t:
     - if $(i, j) = (1, n)$, can't match i with last port n or 2, so try $t \in \{3, \ldots, n-1\}$
     - otherwise, just can't match i with $i+1$, so try $t \in \{i+2, \ldots, j\}$
   - Let $m(i, j) = 1$ if $a_i \neq a_j$ and $m(i, j) = 0$ otherwise (ports of opposite type match)
   - $x(1, n) = \max\{x(2, n)\} \cup \{m(1, t) + x(2, t-1) + x(t+1, n) \mid t \in \{3, \ldots, n-1\}\}$
   - $x(i, j) = \max\{x(i+1, j)\} \cup \{m(i, t) + x(i+1, t-1) + x(t+1, j) \mid t \in \{i+2, \ldots, j\}\}$
3. **Topo**
   - Subproblems $x(i, j)$ only depend on strictly smaller $j - i$, so acyclic
4. **Base**
   - $x(i, j) = 0$ for any $j - i + 1 \in \{0, 1, 2\}$ (no match within 0, 1, or 2 adjacent ports)
5. **Original**
   - Solve subproblems via recursive top down or iterative bottom up.
   - Solution to original problem is $x(1, n)$.
   - Store parent pointers to reconstruct matching (e.g., choice of t or no match at each step)
6. **Time**
   - # subproblems: $O(n^2)$
   - work per subproblem: $O(n)$
   - $O(n^3)$ running time

6.006 Introduction to Algorithms

Spring 2020