Problem Set 9

All parts are due on November 29, 2018 at 11PM. Please write your solutions in the LaTeX and Python templates provided. Aim for concise solutions; convoluted and obtuse descriptions might receive low marks, even when they are correct. Solutions should be submitted on the course website, and any code should be submitted for automated checking on `alg.mit.edu`.

Problem 9-1. [28 points] **Tree Counting**

How many heaps, binary search trees, and AVL trees are there on n distinct integer keys? In this problem, you will describe an algorithm to compute each of these counts. Instead of analyzing each algorithm's running time, please report the asymptotic number of **arithmetic operations** performed by your algorithm (additions, subtractions, multiplications, and divisions).[1]

(a) [8 points] Recall, a **Binary Search Tree** is a binary tree with a key at each node, satisfying the BST property. There are 5 binary search trees on the $n = 3$ distinct keys $\{60, 0, 6\}$. Describe a dynamic-programming algorithm to compute the number of different binary search trees on a given set of n distinct integer keys, using at most $O(n^2)$ arithmetic operations.

Solution:

1. **Subproblems**
   - Doesn't matter what the keys are, as long as they have a unique order
   - $x(k)$: number of BSTs on a(ny) fixed set of k distinct keys
2. **Relate**
   - Root of subtree must be some key, the ith smallest (**Guess!**)
   - Left and right subtrees will have $i-1$ and $k-i$ keys respectively
   - $x(k) = \sum_{i=1}^{k} x(i-1) \cdot x(k-i)$
   - $x(k)$ only depends on subproblems with strictly smaller k, so acyclic
3. **Base**
   - $x(0) = 1$ (only one BST on zero keys)
4. **Solution**
   - $x(n)$ (number of BSTs on n distinct keys)
5. **Time**
   - $O(n)$ subproblems

[1] Numbers in this problem can grow too large to fit in a constant number of machine words, so arithmetic can take more than constant time. Thus instead, you should report the asymptotic number of arithmetic operations performed.

- $O(n)$ arithmetic operations per subproblem
- $O(n^2)$ operations in total

Rubric:

- 1 point for subproblem description
- 1 point for a correct recursive relation
- 1 point for correct base cases in terms of subproblems
- 1 point for correct solution in terms of subproblems
- 1 point for correct analysis of running
- 3 points if correct algorithm is $O(n^2)$
- Partial credit may be awarded

(b) [10 points] Recall, an **AVL Tree** is a binary tree with a key stored at each node, satisfying both the BST and AVL properties. There is only 1 AVL tree on the $n = 3$ distinct keys $\{60, 0, 6\}$. Describe an dynamic-progamming algorithm to compute the number of different AVL trees on a given set of n distinct integer keys, using at most $O(n^2 \log n)$ arithmetic operations.

Solution:

1. **Subproblems**
   - Same subproblems as BST counting, but expand subproblems to remember subtree height
   - $x(k, h)$: number of AVL trees on a fixed set of k distinct keys, with height h
2. **Relate**
   - Height-h AVL tree has subtrees with heights $h-1$ or $h-2$ (not both $h-2$)
   - $$x(k,h) = \sum_{i=1}^{k} \Big(\begin{array}{l} x(i-1,h-1) \cdot x(k-i,h-1) + \\ x(i-1,h-1) \cdot x(k-i,h-2) + \\ x(i-1,h-2) \cdot x(k-i,h-1) \end{array} \Big)$$
   - Subproblems $x(k, h)$ only depend on strictly smaller k and h, so acyclic
3. **Base**
   - $x(0, -1) = 1$ (exactly one AVL tree with height -1 on zero keys)
   - Otherwise $x(k, h) = 0$ for $k < 1$ or $h < 0$ (no other AVL trees with negative height or less than one key)
4. **Solution**
   - AVL tree on n keys has height at most $2 \log n$
   - So solution is $\sum_{-1}^{2 \log n} x(n, h)$
5. **Time**
   - Need to consider $0 \le k \le n$ and $-1 \le h \le 2 \log n$
   - $O(n \log n)$ subproblems

- $O(n)$ arithmetic operations per subproblem
- $O(n^2 \log n)$ operations in total

Rubric:

- 1 point for subproblem description
- 1 point for a correct recursive relation
- 1 point for correct base cases in terms of subproblems
- 1 point for correct solution in terms of subproblems
- 1 point for correct analysis of running
- 3 points if correct algorithm is $O(n^2 \log n)$
- Partial credit may be awarded

(c) [10 points] Recall, a **Binary Min Heap** is a left-justified, complete binary tree with a key at each node, satisfying the min-heap property. There are 2 binary min heaps on the $n = 3$ distinct keys $\{60, 0, 6\}$. Describe a dynamic-programming algorithm to compute the number of different binary min heaps on a given set of n distinct integer keys, using at most $O(n)$ arithmetic operations.

Solution:

1. **Subproblems**
   - A binary heap on k keys has:
     - height: $h(k) = \lceil \log_2(k+1) \rceil - 1$
     - number of keys in bottom level: $b(k) = (2^{h(k)} - 1) - k$
     - size of left subtree: $\ell(k) = (2^{h(k)-1} - 1) + \min\{b(k), 2^{h(k)-2}\}$
     - size of right subtree: $r(k) = k - 1 - \ell(k)$
   - Subtrees of binary heaps are also left-justified and complete
   - $\boxed{x(k)\text{: number of binary min heaps on a fixed set of } k \text{ distinct keys}}$
2. **Relate**
   - Root of subtree must be maximum of keys in subtree
   - Can split remaining keys into left and right subtrees arbitrarily
   - There are $k+1$ choose $\ell(k)$ possible key subsets for left subtree
   - $x(k) = \binom{k-1}{\ell(k)} \cdot x(\ell(k)) \cdot x(r(k))$
   - Subproblems $x(k)$ only depend on strictly smaller k, so acyclic
3. **Base**
   - $x(0) = x(1) = 1$ (exactly one binary min heap on zero or one key)
4. **Solution**
   - $x(n)$ (number of binary min heaps on n keys)
5. **Time**

- Each $h(k)$, $\ell(k)$, and $r(k)$ can be computed in $O(1)$ operations ($O(n)$ total operations) by first precomputing/memoizing 2^i, for $0 \leq i \leq 2\log n$ in $O(\log n)$ total operations
- Each $\binom{a}{b} = \frac{a!}{b!(a-b)!}$ can be computed in $O(1)$ arithmetic operations ($O(n)$ total operations) by precomputing/memoizing $k!$, for $1 \leq k \leq n$ in $O(n)$ total operations
- $O(n)$ subproblems
- $O(1)$ arithmetic operations per subproblem
- $O(n)$ operations in total

Rubric:

- 2 points for subproblem description
- 2 points for a correct recursive relation
- 1 point for correct base cases in terms of subproblems
- 1 point for correct solution in terms of subproblems
- 1 point for correct analysis of running
- 3 points if correct algorithm is $O(n)$
- Partial credit may be awarded

Problem 9-2. Treasureship! [10 points]

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**
   - Game board has n columns, each of height 2, and 2 rows, each of width n.
   - Let $v(x, y)$ denote the grid value at row y column x, for $0 \leq y \leq 1$, $0 \leq x \leq n$
   - Let $s(i, j)$ represent game board subset containing columns 1 to i of row 1, and colomns 1 to $i + j$ of row 2, for $-1 \leq j \leq 1$.
   - $\boxed{x(i, j)\text{: maximum score, only placing ships on board subset } s(i, j)}$
2. **Relate**
   - Either you get points from every grid square in column i or not

- $x(i,j) = \begin{cases} \max\{v(i,1) + v(i-1,1) + x(i-2,+1), x(i-1,0)\} & \text{if } j = -1 \\ \max\{v(i+1,2) + v(i,2) + x(i,-1), x(i,0)\} & \text{if } j = 1 \\ \max\{v(i,1) + v(i,2) + x(i-1,0), x(i,-1), x(i-1,+1)\} & \text{if } j = 0 \end{cases}$
- Subproblems $x(i,j)$ only depend on strictly lexiographically smaller (i,j), so acyclic.

3. **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!)
4. **Solution**
   - Solution is $x(n,0)$, the maximum considering all grid squares.
   - Store parent pointers to reconstruct ship locations
5. **Time**
   - # subproblems: $O(n)$
   - work per subproblem $O(1)$
   - $O(n)$ running time

Rubric:

- 2 points for subproblem description
- 2 points for a correct recursive relation
- 1 point for correct base cases in terms of subproblems
- 1 point for correct solution in terms of subproblems
- 1 point for correct analysis of running
- 3 points if correct algorithm is $O(n)$
- Partial credit may be awarded

Problem 9-3. Circleworld Politics [15 points]

Circleworld, an essentially one-dimensional circular nation surrounding a sun, is home to n residents (where n is odd). There are two political parties in Circleworld: the Maryonettes and the Itnizks, with the Maryonettes currently in power. Each resident is a member of one of these two parties and always votes for their party in any election. Mary Jander, the leader of the Maryonettes, lives at address 1 in Circleworld. Each other resident lives at a unique consecutively increasing integer address starting at Mary's house going around the circular world in one direction, so that Mary's next door neighbor in that direction lives at address 2, and her other next door neighbor in the opposite direction lives at address n.

Circleworld is currently divided into d districts that are each **odd-contiguous**. A district is odd-contiguous if it contains an odd number of residents who live at a contiguous sequence of addresses along the circular world. For example, Mary currently lives in district 1, comprising 7 residents who live at addresses $(n-1, n, 1, 2, 3, 4, 5)$. When an election is held, the residents in each district

cast a vote for their political party, and the party receiving the majority of votes within a district wins that district. Being the leader of the party in power, Mary is allowed to completely reassign the d districts in Circleworld prior to the upcoming election, under the condition that each of the new d districts is odd-contiguous, and that every resident is assigned to exactly one district. Given the address and political party of each resident in Circleworld, describe an efficient dynamic-programming algorithm to help Mary reassign districts so as to maximize the number of districts her party can win in the upcoming election.

Solution:

1. **Subproblems**
   - Want to maximize won districts.
   - All index arithmetic will be cyclic on the n addresses, so that e.g. $1 - 2 = n - 1$
   - Let $s(i, j)$ be contiguous cyclic subset from address i to address j
   - Let $w(i, j)$ be $-\infty$ if $|s(i, j)|$ is even; otherwise, if $|s(i, j)|$ is odd, let $w(i, j)$ be 1 if residents in $s(i, j)$ are majority Maryonettes, else 0
   - (Can naïvely compute all $O(n^2)$ values of $w(i, j)$ in $O(n^3)$ time, $O(n)$ time per $w(i, j)$)
   - If $j < i$, then $s(i, j) = (i, i + 1, \ldots, n, 1, \ldots, j - 1, j)$
   - $\boxed{x(i, j, k)\text{: max districts winable, dividing } s(i, j) \text{ into exactly } k \text{ odd-contiguous districts}}$
2. **Relate**
   - Address i is the start of a district, guess integer address a at end of district for $a \in s(i, j)$
   - Recurse on remainder, requiring fewer districts
   - $x(i, j, k) = \max\{w(i, a) + x(a + 1, j, k - 1) \mid a \in s(i, j)\}$
   - Subproblems $x(i, j, k)$ only depend on subproblems corresponding to cyclic subsets having strictly fewer residents, so acyclic
3. **Base**
   - $x(i, j, k) = -\infty$ if $|s(i, j)| < k$ (impossible to split $k - 1$ residents into k districts)
   - $x(i, j, 1) = w(i, j)$ (one district, no choice!)
4. **Solution**
   - Solution is $\max\{x(i, i - 1, d) \mid 1 \le i \le n\}$, (guess start of first district)
5. **Time**
   - # subproblems: dn^2
   - Then work per subproblem is $O(n)$
   - $O(n)$ work to compute solution from subproblems
   - $O(dn^3)$ total running time

Rubric:

- 3 points for subproblem description

- 3 points for a correct recursive relation
- 2 points for correct base cases in terms of subproblems
- 1 point for correct solution in terms of subproblems
- 1 point for correct analysis of running
- 5 points if correct algorithm is $O(dn^3)$
- Partial credit may be awarded

Problem 9-4. [12 points] **Bottle Breaker Remix**

After witnessing the popularity of *Green Zombie Atonement II* (GZA2), a competitor has released a knockoff videogame called *Blue Demon Confessions III* (BDC3). BDC3 features the exact same bottle-breaking minigame as in GZA2 (from PS8) with one key difference: whenever one or more bottles are hit and shatter on the ground, the remaining bottles **get pushed together**, so that the bottles on either side of the bottles that just broke **become adjacent**. As in GZA2, you may throw a ball to hit any single bottle or any pair of adjacent bottles; but in BDC3, you may hit adjacent bottle pairs even if they were not adjacent in the original lineup, to score the product of their labels. For example, if the lineup of bottle labels is $(5, -3, -5, 1, 2, 9, -4)$, then one sub-optimal[2] score attainable is 38: you could throw balls at pairs $(-3, -5)$, $(2, 9)$, and $(5, 1)$ (which become adjacent after $(-3, -5)$ are hit); or, you could throw a ball at pair $(2, 9)$, then throw a ball at 1 to get rid of that bottle, and then throw a ball at pair $(-5, -4)$, to achieve the same score. Given a line of n bottle labels, describe an efficient dynamic-programming algorithm to determine the maximum possible score.

Solution:

1. **Subproblems**
   - Points now depend on the order we hit the bottles
   - Let $l(i)$ denote the label of bottle $1 \leq i \leq n$, where n is number of bottles
   - $x(i, j)$: maximum score possible by hitting only bottles i to j
2. **Relate**
   - Either bottle i contributes to score or not
   - If not scoring, pairs with no bottle, recurse on $x(i + 1, j)$
   - If scoring, pairs with some bottle k for $1 < k \leq j$, recurse on two disjoint subsets
   - $x(i, j) = \max \left\{ \begin{array}{l} x(i + 1, j), \\ \max\{l(i) \cdot l(k) + x(i + 1, k - 1) + x(k + 1, j) \mid i < k \leq j\} \end{array} \right\}$
   - Subproblems $x(i, j)$ only depend on strictly smaller $j - i$, so acyclic
3. **Base**
   - Maximum score on 1 or fewer bottles is zero

[2]The optimal score for this lineup is 62, achievable by hitting pairs $(-3, -5)$, $(1, 2)$, and $(5, 9)$.

- $x(i, j) = 0$ for all $j - i < 1$

4. **Solution**
   - Solution is $x(1, n)$, the maximum considering all the bottles
5. **Time**
   - # subproblems: n^2
   - work per subproblem $O(n)$
   - $O(n^3)$ running time

Rubric:

- 2 points for subproblem description
- 3 points for a correct recursive relation
- 2 points for correct base cases in terms of subproblems
- 1 point for correct solution in terms of subproblems
- 1 point for correct analysis of running
- 3 points if correct algorithm is $O(n^3)$
- Partial credit may be awarded

Problem 9-5. Hidden Palindromes [35 points]

Cayson wants to send hidden messages to his friends Ack and Dirk, via Derrit, an online public message board. Cayson will post to the message board a seemingly random string of lowercase letters, containing a secret message within a palindrome subsequence. A **palindrome** is any string which is the same string when the order of its characters is reversed. String B is a **subsequence** of a string A if the letters of B appear in order in A, possibly with other letters interspersed. Given an online message board post A, Cayson's secret message B will be the **first half** of the longest palindrome subsequence of A. If the longest palindrome subsequence has an odd number of characters, the secret message will include the middle character. You may assume that the longest palindrome subsequence of one of Cayson's online message board posts is unique. For example, if Cayson's message board post is `"nweeyaoslsoitmarawtuitfsjaipdiwi"`, the (unique) longest subsequence palindrome is `"wasitaratisaw"`[3], with the secret message being `"wasitar"`[4].

(a) [10 points] Given a message board post from Cayson containing n characters, describe an $O(n^2)$-time algorithm to decode it and return Cayson's secret message.

Solution:

1. **Subproblems**

[3] Was it a rat I saw?
[4] Was it AR?

- Let A be Cayson's secret message with $|A| = n$
- Let A_i be the ith letter of Cayson's message board posting
- $x(i, j)$: max length of palindrome subsequence in substring from A_i to A_j
- for $1 \leq i \leq j \leq n$

2. **Relate**
   - If $A_i = A_j$, some longest palindrome subsequence uses both
   - Otherwise, no longest palindrome subsequence uses both A_i and A_j
   - $x(i, j) = \begin{cases} 1 + x(i+1, j-1) & \text{if } A_i = A_j \\ \max\{x(i+1, j), x(i, j-1)\} & \text{otherwise} \end{cases}$
   - Subproblems $x(i, j)$ only depend on strictly smaller $j - i$, so acyclic
3. **Base**
   - $x(i, j) = 0$ for all $j < i$ (no palindrome in a string with no letters)
4. **Solution**
   - Solution is $x(1, n)$, max length of palindrome subsequence in entire string
   - Store $O(1)$-sized parent pointers to reconstruct matched message
5. **Time**
   - # subproblems: n^2
   - work per subproblem $O(1)$
   - $O(n^2)$ running time

This solution uses at most $O(1)$ time and space per subproblem. If you use linear space or time per subproblem (like storing the longest palindrome subsequence or message per subproblem taking $O(n)$ space), your dynamic program will take $O(n^3)$ time.

A solution that many students tried that seems to work is to run **Longest Common Subsequence** on the input message post and its reverse. The problem here is that Longest Common Subsequence may return a string that is not a palindrome. For example, the string 'acbac' and its reverse contain three longest common subsequences: 'aba', 'cbc', and 'abc', one of which is **not a palindrome**. Any version of this approach not accompanied by a full and accurate proof of correctness should receive a **maximum of 5 points** total.

Rubric:

- 2 points for subproblem description
- 2 points for a correct recursive relation
- 1 point for correct base cases in terms of subproblems
- 1 point for correct solution in terms of subproblems
- 1 point for correct analysis of running
- 3 points if correct algorithm is $O(n^2)$
- Partial credit may be awarded

(b) [25 points] Implement the decoding algorithm described above in a Python function `decode_message`. You can download a code template containing some test cases from the website. Submit your code online at `alg.mit.edu`.

Solution:

```
def decode_message(A):                  # iterative bottom up
    'Return the first half of the longest palindrome subsequence of string A'
    n = len(A)
    memo, parent = {}, {}
    for k in range(-2, n):              # k = j - i topological sort order
        for i in range(n - k):
            j = i + k
            if j < i:                   # base case
                memo[(i, j)] = 0
            elif A[i] == A[j]:          # outer characters match!
                memo[(i, j)] = 1 + memo[(i + 1, j - 1)]
                parent[(i, j)] = (i + 1, j - 1)
            else:                       # outer characters don't match
                choice_1, choice_2 = memo[(i + 1, j)], memo[(i, j - 1)]
                if choice_2 < choice_1:
                    memo[(i, j)] = choice_1
                    parent[(i, j)] = (i + 1, j)
                else:
                    memo[(i, j)] = choice_2
                    parent[(i, j)] = (i, j - 1)
    message = []
    i, j = 0, n - 1
    while (i, j) in parent:
        i_, j_ = parent[(i, j)]
        if A[i] == A[j]:                # characters matched!
            message.append(A[i])
        i, j = i_, j_
    return ''.join(message)
```