CS/ECE 374 Sec A ✧ Spring 2025

☙ Homework 6 ❧

Due Wednesday, Mar 12th, 2025 at 9am

- You can work in a group of up to **three** students. Read the instructions on the course website for additional details.
- **Submit your solutions electronically on the course Gradescope site as PDF files.** Submit a separate PDF file for each numbered problem. If you plan to typeset your solutions, please use the LaTeX solution template on the course web site. If you must submit scanned handwritten solutions, please use a black pen on blank white paper and a high-quality scanner app (or an actual scanner, not just a phone camera).
- Late submissions will be accepted (for 75% credit) until midnight the day of the deadline.

☞ Some important course policies ☜

- **You may use any source at your disposal**—paper, electronic, or human—but you ***must*** cite ***every*** source that you use, and you ***must*** write everything yourself in your own words. See the academic integrity policies on the course web site for more details including the policy on using AI tools.
- **Avoid the Two Deadly Sins!** Any homework or exam solution that breaks any of the following rules will be given an ***automatic zero***, unless the solution is otherwise perfect. Yes, we really mean it. We're not trying to be scary or petty (Honest!), but we do want to break a few common bad habits that seriously impede mastery of the course material.
  - Always give complete solutions, not just examples.
  - Always declare all your variables, in English. In particular, always describe the specific problem your algorithm is supposed to solve.

See the course web site for more information.

If you have any questions about these policies,
please don't hesitate to ask in class, in office hours, or on Ed.

---

1. Given a sequence $a_1, a_2, \ldots, a_n$ of n distinct numbers in an array A, an *inversion* is a pair $i < j$ such that $a_i > a_j$. Note that a sequence has no inversions if and only if it is sorted in ascending order. We saw in lecture an $O(n \log n)$ algorithm to count the number of inversions in a given sequence (this is also described in the Kleinberg-Tardos book on algorithms). We consider two generalizations.

   - (5 pts) Call a pair $i < j$ a *significant* inversion if $a_i > 10a_j$. Describe an $O(n \log n)$ time algorithm to count the number of significant inversions in a given sequence.
   - (5 pts) Consider a further generalization. In addition to the sequence $a_1, \ldots, a_n$ we are given weights $w_1, \ldots, w_n$ where $w_i \geq 1$ for each i. Now call a pair $i < j$ a significant inversion if $a_i > w_j a_j$. Describe an $O(n \log n)$ time algorithm to count the number of significant inversions given the sequences a and w in arrays A and W. Partial credit for an $O(n \log^2 n)$ time algorithm. Note that this part generalizes the previous one so you can just describe a solution for this part and explain why it yields an algorithm for the previous part.

2. (10 pts) Problem 13 in Jeff's chapter on DP. https://jeffe.cs.illinois.edu/teaching/algorithms/book/03-dynprog.pdf.

3. **Not to submit:** Let $a_1, a_2, \ldots, a_n$ be a sequence of numbers (could be positive or negative) and let B be a number. You wish to decide whether the sequence can be split/partitioned into contiguous subsequences such that cube of the sum of numbers in each subsequence is at most B. For instance if the sequence is $2, 10, -1, 11$ and $B = 100$ then the answer is no because the smallest value that can be obtained for the subsequence that contains 11 is $(11 - 1)^3 = 1000$. The answer is yes if $B = 1000$ where the sequence is split into three contiguous subsequences: $[2]$, $[10]$ and $[-1, 11]$. Describe an algorithm for this problem. Your algorithm, if it outputs yes, should also output a split of the sequence that shows the correctness of the answer.

4. **Not to submit:** The McKing chain wants to open several restaurants along Red street in Shampoo-Banana. The possible locations are at $L_1, L_2, \ldots, L_n$ where L_i is at distance m_i meters from the start of Red street. Assume that the street is a straight line and the locations are in increasing order of distance from the starting point (thus $0 \leq m_1 < m_2 < \ldots < m_n$). McKing has collected some data indicating that opening a restaurant at location L_i will yield a profit of p_i independent of where the other restaurants are located. However, the city of Shampoo-Banana has a zoning law which requires that any two McKing locations should be D or more meters apart. Describe an algorithm that McKing can use to figure out the maximum profit it can obtain by opening restaurants while satisfying the city's zoning law.

5. **Not to submit:** Problem 16 in Jeff's chapter on DP. https://jeffe.cs.illinois.edu/teaching/algorithms/book/03-dynprog.pdf. A clarification on the problem: Mr. Fox has to go straight through the obstacle course and visit each booth exactly once on their way.

Solved Problem

5. A *shuffle* of two strings X and Y is formed by interspersing the characters into a new string, keeping the characters of X and Y in the same order. For example, the string **BANANAANANAS** is a shuffle of the strings BANANA and **ANANAS** in several different ways.

BANANAANANAS BAN**ANA**ANA**NAS** B**A**N**AN**A**A**NA**NA**S

Similarly, the strings **PRODGYRNAMAMMIINCG** and **DYPRONGARMAMMICING** are both shuffles of DYNAMIC and **PROGRAMMING**:

PROD**G**Y**R**NAM**AMMI**I**N**C**G** **DY**PRO**N**G**A**R**M**AMM**IC**ING**

Given three strings $A[1..m]$, $B[1..n]$, and $C[1..m+n]$, describe and analyze an algorithm to determine whether C is a shuffle of A and B.

Solution: We define a boolean function $\mathit{Shuf}(i, j)$, which is TRUE if and only if the prefix $C[1..i+j]$ is a shuffle of the prefixes $A[1..i]$ and $B[1..j]$. This function satisfies the following recurrence:

$$\mathit{Shuf}(i,j) = \begin{cases} \text{TRUE} & \text{if } i = j = 0 \\ \mathit{Shuf}(0, j-1) \wedge (B[j] = C[j]) & \text{if } i = 0 \text{ and } j > 0 \\ \mathit{Shuf}(i-1, 0) \wedge (A[i] = C[i]) & \text{if } i > 0 \text{ and } j = 0 \\ \big(\mathit{Shuf}(i-1, j) \wedge (A[i] = C[i+j])\big) & \\ \quad \vee \big(\mathit{Shuf}(i, j-1) \wedge (B[j] = C[i+j])\big) & \text{if } i > 0 \text{ and } j > 0 \end{cases}$$

We need to compute $\mathit{Shuf}(m, n)$.

We can memoize all function values into a two-dimensional array $\mathit{Shuf}[0..m][0..n]$. Each array entry $\mathit{Shuf}[i, j]$ depends only on the entries immediately below and immediately to the right: $\mathit{Shuf}[i-1, j]$ and $\mathit{Shuf}[i, j-1]$. Thus, we can fill the array in standard row-major order. The original recurrence gives us the following pseudocode:

```
SHUFFLE?(A[1..m], B[1..n], C[1..m+n]):
  Shuf[0,0] ← TRUE
  for j ← 1 to n
      Shuf[0,j] ← Shuf[0,j-1] ∧ (B[j] = C[j])
  for i ← 1 to n
      Shuf[i,0] ← Shuf[i-1,0] ∧ (A[i] = B[i])
      for j ← 1 to n
          Shuf[i,j] ← FALSE
          if A[i] = C[i+j]
              Shuf[i,j] ← Shuf[i,j] ∨ Shuf[i-1,j]
          if B[i] = C[i+j]
              Shuf[i,j] ← Shuf[i,j] ∨ Shuf[i,j-1]
  return Shuf[m,n]
```

The algorithm runs in $O(mn)$ *time*. ■

Rubric: Max 10 points: Standard dynamic programming rubric. No proofs required. Max 7 points for a slower polynomial-time algorithm; scale partial credit accordingly.

Standard dynamic programming rubric. 10 points divided as follows

- 3 points for a clear and correct English description of the recursive function you are trying to evaluate. (Otherwise, we don't even know what you're *trying* to do.)
  - -1 for naming the function "OPT" or "DP" or any single letter.
  - – No credit if the description is inconsistent with the recurrence.
  - – No credit if the description does not explicitly describe how the function value depends on the named input parameters.
  - – No credit if the description refers to internal states of the eventual dynamic programming algorithm, like "the current index" or "the best score so far". The function must have a well-defined value that depends *only* on its input parameters (and constant global variables).
  - – An English explanation of the *recurrence* or *algorithm* does not qualify. We want a description of *what* your function returns, not (here) an explanation of *how* that value is computed.
- 4 points for a correct recurrence, described either using mathematical notation or as pseudocode for a recursive algorithm.
  - 1 for base case(s). $-\frac{1}{2}$ for one *minor* bug, like a typo or an off-by-one error.
  - 3 for recursive case(s). -1 for each *minor* bug, like a typo or an off-by-one error.
  - -2 for greedy optimizations without proof, even if they are correct.
  - – **No credit for the rest of the problem if the recursive case(s) are incorrect.**
- 3 points for iterative details
  - + 1 for describing an appropriate memoization data structure
  - + 1 for describing a correct evaluation order; a clear picture is usually sufficient. If you use nested for loops, be sure to specify the nesting order.
  - + 1 for correct time analysis. (It is not necessary to state a space bound.)

- For problems that ask for an algorithm that computes an optimal *structure*—such as a subset, partition, subsequence, or tree—an algorithm that computes only the *value* or *cost* of the optimal structure is sufficient for full credit, unless the problem specifically says otherwise.
- Official solutions usually include pseudocode for the final iterative dynamic programming algorithm, ***but iterative pseudocode is not required for full credit***. If your solution includes iterative pseudocode, you do not need to separately describe the recurrence, memoization structure, or evaluation order. But you ***do*** still need and English description of the underlying recursive function (or equivalently, the contents of the memoization structure). ***Perfectly correct iterative pseudocode, with no explanation or time analysis, is worth at most 6 points out of 10.***
- Partial credit for incomplete solutions depends on the running time of the ***best possible*** completion (up to the target running time). For example, consider a solution that contains *only* a clear English description of a function, with no recurrence or iterative details. If the described function *can* be developed into an algorithm with the target running time, the solution is worth 3 points; however, if the function leads

to an algorithm that is slower than the target time by a factor of n, the solution could be worth only 2 points ($=$ 70% of 3, rounded).