---

1. Type that "I thoroughly read the course syllabus" to acknowledge that your did so.

   I thoroughly read the course syllabus

2. Consider the recurrence relation $F(0) = F(1) = 2$ and for $n > 1$

$$F(n) = \sum_{i=1}^{n-1} F(i)F(i-1)$$

   We consider the problem of computing $F(n)$ from n.

   (a) Show that a naive implementation of this recursion takes an exponential time in n. In other words, show that the run time is $\Omega(c^n)$ for some constant $c > 1$.

   solution

   note that

$$\begin{aligned} T(n) &= T(n-1) + T(n-2) + 1 \approx 2T(n-2) + 1 \\ &= 2 \cdot 2T(n-4) + 1 \\ &= 2^m T(n-2m) + 1 \end{aligned} \tag{1}$$

   the plus one represents the constant time required to call the function.

   we can it takes 2^m time to run, but only ends when $m = n$, thus it takes approximately 2^n time to run.

   (b) Explain how, by not recomputing the same $F(i)$ value twice (i.e., using dynamic programming), one can obtain an algorithm for this problem that runs in $O(n^2)$ time.

   solution

   we want A[i] = F[i]F[i-1]

```
1: A is a list of len n, init to 0
2: F is a list of len n, init to 0
3: F[0] = F[1] = 2
4: A[1] = 2·2
5: for i in {2, ..., n} do
6:     A[i]= F[i-1] + F[i-2]
7:     for j in {1, ..., i} do
8:         F[i]+=A[j]
9: return F[n]
```

   note the loop on line 5 iterates n times, and within it line 7 iterates n times as well, thus the program runs in n^2 time

   (c) Give an algorithm for this problem that only has $O(n)$ run time.

   solution

   if we store all $F[i]$, then finding the next term takes $O(1)$ since it requires computing one extra term, which takes

```
1: A is a list of len n, init to 0
2: F is a list of len n, init to 0
```

```
F[0] = F[1] = 2
A[1] = 2 · 2
for i in {2, ..., n} do
    A[i] = F[i-1] + F[i-2]
    F[i] = F[i-1] + A[i]
return F[n]
```

this removes one loop 7 in algo 1 thus it runs in $O(n)$

3. Informally in this problem we consider how to divide the chapters in a story into volumes (think the 7 volumes of Harry Potter or however many Game of Thrones volumes there will be) so as to equalize the size of the volumes. The input consists of positive integers $x_1, \ldots, x_n$ and k. Here x_i is the number of pages in chapter i, and k is the desired number of volumes. The problem is determine which chapters go into each of the k volumes so as to minimize the difference between the most number of pages in any volumes, and the least number of pages in any of the volumes. Of course you can not reorder the story. Give an algorithm whose running time is bounded by a polynomial in n. You running time should not depend on the number of pages in the chapters.

   Hint: This approach prioritizes simplicity over optimal runtime.

   A volume is composed of a contiguous subset of chapters, so its total page count belongs to the set:

$$P := \left\{ \sum_{i=a}^{b} x_i \mid 1 \leq a \leq b \leq n \right\}$$

   Consider every pair of values (L, H) from this set such that $L \leq H$. For each pair, check if it's possible to partition the chapters into exactly k volumes, where each volume's page count is between L and H. We call such a pair (L, H) a **feasible pair**. The goal is to find a feasible pair (L, H) that minimizes the difference $H - L$.

   (a) Give a polytime algorithm to determine if a given pair (L, H) is feasible (assume $L, H > 0$. Explain why the run time is polynomial in n. Be concise.

   Solution

```
S[i] = ∑_{a=1}^{i} x_a
new array A, size k by n. set all values to []
for j ∈ {1, ..., n} do
    if L≤S[j]≤H then
        A[0][j].append(0)
for i ∈ {0, ..., k − 1} do
    for j ∈ {1, ..., n} do
        if A[i][j] is nonempty then
            for k ∈ {j + 1, ..., n} do
                if L ≤ S[k] − S[j] ≤ H then
                    A[i+1][k].append(j)
if A[-1][-1] is nonempty then
    return True
return False
```

note that $S[a] - S[b]$ is the number of pages between the chapters b to a.

step 3 is initializing the first row of A. the loop determines if the chapter j can be in the 1st volume. the each row of A is answering the question, is it possible for this chapter to be in the $i + 1$th volume

the loop from step 6 to 11 is answering, if the volume started at chapter j, is it possible for chapter k to be in book $i + 1$.

the for loop on step 6 contributes a constant time since k is fixed. the for loop on step 7 contributes $O(n)$, and the for loop on 9 also contributes $O(n)$, thus teh total loop on steps 6 to 11 run in $O(n^2)$ time.

(b) Given a feasible pair (L, H), explain how to find a partitioning of chapters into k volumes so that every volume's page count is between L and H.

Solution

starting from A[-1][-1], pick any element a inside that array. $a+1$ to the final chapter forms the final book. repeat the same process from A[-2][a] until you reach the final row, in which you make sure you start from the 1st chapter.

There must exist a path because it was necessary for A[-1][-1] to be nonempty

4. Consider the problem where the input is a collection of n train trips within Germany. For the ith trip T_i you are given the date d_i of that trip, and the non-discounted fare f_i for that trip. The German railway system sells a Bahncard for B Euros that entitles you to a 50% fare reduction on all train travel within Germany within L days of purchase. The problem is to determine when to buy a Bahncard to minimize the total cost of your travel.

For example if the input was
d_1 = January 11, 1997, $f_1 = 20$ Euros,
d_2 = February 11, 1998, $f_2 = 200$ Euros,
d_3 = January 11, 1999, $f_3 = 200$ Euros,
d_4 = March 13, 1999, $f_4 = 100$ Euros,
d_5 = February 11, 2002, $f_5 = 200$ Euros, and
d_6 = January 11, 2003, $f_6 = 600$ Euros, $B = 240$
, and $L = 365$ then you might buy a Bahncard on February 11, 1998, and February 11, 2002, resulting in a total cost of 1200 Euros (240 * 2 + 20 + (200 + 200) * 0.5 + 100 + (200+ 600) * 0.5). If you only purchase a Bahncard on February 11, 2002, you would pay a total of 1160 Euros (240 + 20 + 200 + 200 + 100 + (200+ 600) * 0.5).

Give a polynomial time algorithm for this problem. The running time of your algorithm should be independent of B and L. For convenience, assume that the dates are given as integers, and $d_1 = 1 < d_2 < d_3 < \ldots$.

solution

consider a matrix C w dimension $n \times n$ where n is the total number of trips. the first index i represents the price of the first i trips, given that the last pass was bought on the second index j.

note for

$$C[j][i] = \min_{j'<i}(C[i-1][j']) + B + 0.5 \sum_{i'=j+1}^{i} \delta(i', j)d_i + \sum_{i'=j+1}^{i} \delta'(i', j)d_i \tag{2}$$

where $\delta(i, j)$ is the indicator function for when a trip i is within the card timeframe of j and $\delta'(i, j)$ is the indicator function for when the trip is outside the timeframe of a card bought at trip j.