Question 5: The egyptian tomb

Team-ID: —

Team-Name: Super Secret

Bearbeiter/-innen dieser Aufgabe:
Sigma

November 12, 2024

Contents

1 Solution Idea

The solution to this problem utilizes mathematical equations and inequalities and follows an approach similar to what is known as the branch-and-bound algorithm. Initially, I applied this technique intuitively, later discovering that it aligns closely with the principles of branch and bound. It utilizes a forward selection approach, similar to a greedy algorithm, to make locally optimal choices. When encountering roadblocks, backtracking is used to explore alternative paths. Additionally, the algorithm is bound to limit exploration of paths that are unlikely to lead to an optimal solution, ensuring more efficient search and decision-making.

Our first significant discovery was identifying mathematical equations and inequalities that pinpoint the only two fastest possible movement options at any given section. To fully understand this, we need to examine a table showing the time frequency values for each section.

Opening Number	Multiplication table of 5	Multiplication table of 8	Multiplication table of 12
Excluded	0	0	0
1-Opens	5	8	12
2-Closes	10	16	24
3-Opens	15	24	36
4-Closes	20	32	48

Table 1: Time Frequency Values for Each Section

We exclude 0 because all doors will be closed at that point, and skipping this step slightly optimizes the algorithm.

The first key point is that if the opening number of a door in a section is odd, the door is open. Based on this, we can calculate the opening number for the next possible state and determine whether movement to the next section is allowed by checking if this number is even or odd.

Variables

To help clarify the logic, let's define the following variables:

- C_t: The current time
- B_n: The base time for the next section
- B_c: The base time for the current section
- S_c: The historical step time in the current section
- S_p: The historical step time in the previous section

Using these variables, we'll examine three possible cases to determine movement.

Case 1: Fastest Movement (Immediate Run to the Next Section)

The fastest possible movement occurs when the door in the next section is already open, allowing immediate access. This condition can be represented by the equation:

$$\left\lfloor \frac{C_t}{B_n} \right\rfloor$$

Here, we check whether this results in an even or odd number. If the result is odd, the door is open, allowing movement to the next section. If it's even, this attempt fails, and we proceed to the next case.

Case 2: Delayed Movement (Waiting for the Door to Open in the Next Section)

If the door in the next section isn't immediately open, we consider a slightly delayed movement, where we wait for the next opportunity to enter that section before the door in the current section closes. This is represented by the inequality:

$$B_c \left(\left\lfloor \frac{C_t}{B_c} \right\rfloor + 1 \right) > B_n \left(\left\lfloor \frac{C_t}{B_n} \right\rfloor + 1 \right)$$

In other words, this checks if we could reach the next section before the door in the current section closes. If this results in a true output, the door in the next section will be open, allowing us to move forward. If the result is a false output, however, movement fails under this condition, and we must consider the next option.

Case 3: Backward Propagation (Returning to the Previous Section)

If both forward movement options have failed, we turn to the possibility of waiting longer in the previous section so that the door in the current section will open again in the next cycle, allowing us to try movement from there. This condition is represented by the inequality:

$$B_c \left(\left\lfloor \frac{S_c}{B_c} \right\rfloor + 2 \right) < B_p \left(\left\lfloor \frac{S_p}{B_p} \right\rfloor + 1 \right)$$

This checks if we could have waited in the previous section long enough to catch the next cycle of the door opening in the current section. If this section has a true outcome, we take another step backward. Otherwise, we restart the forward selection process.

These three fundamental steps are what we use to optimize for the quickest execution time.

2 Implementation

The algorithm was implemented in Python and only uses the inbuilt math libraries. All the steps mentioned above are made functions so that we can repeatedly use them. The pseudo code of the program looks like this:

Algorithm 1 Egyptian_Tomb (Recursive with Backtracking)

```
0: procedure EGYPTIAN_TOMB(ANZAHL_QUADER, QUADER_FREQUENZ)
0:   aktuellerIndex ← 1
0:   aktuelleZeit ← QUADER_FREQUENZ[0]
0:   schrittListe ← [aktuelleZeit]
0:   while true do
0:     if FastestMovement(aktuelleZeit, aktuellerIndex) then
0:       aktuellerIndex ← aktuellerIndex + 1
0:       schrittListe.append(aktuelleZeit)
0:     else if DelayedMovement(aktuelleZeit, aktuellerIndex) then
0:       aktuelleZeit ← nächste gültige Zeit für QUADER_FREQUENZ[aktuellerIndex]
0:       aktuellerIndex ← aktuellerIndex + 1
0:       schrittListe.append(aktuelleZeit)
0:     else
0:       (schrittListe, aktuellerIndex) ← BackwardPropagation(schrittListe, aktuellerIndex)
0:       aktuelleZeit ← schrittListe[-1]
0:     end if
0:     if aktuellerIndex = ANZAHL_QUADER then
0:       print(schrittListe)
0:       anweisungen ← GenerateInstructions(schrittListe)
0:       print(anweisungen)
0:       break
0:     end if
0:   end while
0: end procedure=0
```

3 Examples

Here are the algorithm results based on the input examples from the website.

Example 1

Output: Warte 51 Minuten, laufe zum Grabmal
Historical Steps: [51, 51, 51, 51, 51]
Execution Time: 0.0000 seconds (practically negligible)

Example 2

Output: Warte 510000 Minuten, laufe zum Grabmal
Historical Steps: [510000, 510000, 510000, 510000, 510000]
Execution Time: 0.0000 seconds (practically negligible)

Example 3

Output: Warte 20 Minuten, laufe in den Abschnitt 1, Warte 2 Minuten, laufe in den Abschnitt 6, Warte 13 Minuten, laufe zum Grabmal
Historical Steps: [20, 22, 22, 22, 22, 22, 35, 35, 35, 35]
Execution Time: 0.0000 seconds (practically negligible)

Example 4

Output: Warte 3242835 Minuten, laufe in den Abschnitt 1, Warte 60360 Minuten, laufe in den Abschnitt 2, Warte 360 Minuten, laufe in den Abschnitt 5, Warte 37695 Minuten, laufe in den Abschnitt 6, Warte 51407 Minuten, laufe in den Abschnitt 8, Warte 36216 Minuten, laufe in den Abschnitt 9, Warte 4024 Minuten, laufe zum Grabmal
Historical Steps:
[3242835, 3303195, 3303555, 3303555, 3303555, 3341250, 3392657, 3392657, 3428873, 3432897]
Execution Time: 0.0020 seconds

Example 5

Output: Warte 199166868 Minuten, laufe in den Abschnitt 1, Warte 1675 Minuten, laufe in den Abschnitt 2, Warte 332 Minuten, laufe in den Abschnitt 4, Warte 540 Minuten, laufe in den Abschnitt 5, Warte 4130 Minuten, laufe in den Abschnitt 8, Warte 1174 Minuten, laufe in den Abschnitt 9, Warte 1341 Minuten, laufe in den Abschnitt 10, Warte 636 Minuten, laufe in den Abschnitt 13, Warte 1173 Minuten, laufe in den Abschnitt 14, Warte 7238 Minuten, laufe in den Abschnitt 15, Warte 622 Minuten, laufe in den Abschnitt 18, Warte 1623 Minuten, laufe in den Abschnitt 19, Warte 1568 Minuten, laufe in den Abschnitt 20, Warte 1285 Minuten, laufe in den Abschnitt 21, Warte 1011 Minuten, laufe in den Abschnitt 29, Warte 834 Minuten, laufe in den Abschnitt 36, Warte 864 Minuten, laufe in den Abschnitt 37, Warte 102 Minuten, laufe in den Abschnitt 38, Warte 5991 Minuten, laufe in den Abschnitt 40, Warte 21 Minuten, laufe in den Abschnitt 45, Warte 2193 Minuten, laufe in den Abschnitt 46, Warte 189 Minuten, laufe in den Abschnitt 51, Warte 1698 Minuten, laufe in den Abschnitt 52, Warte 2031 Minuten, laufe in den Abschnitt 53, Warte 186 Minuten, laufe in den Abschnitt 54, Warte 2252 Minuten, laufe in den Abschnitt 55, Warte 2539 Minuten, laufe in den Abschnitt 56, Warte 1314 Minuten, laufe in den Abschnitt 57, Warte 17 Minuten, laufe in den Abschnitt 58, Warte 1770 Minuten, laufe in den Abschnitt 63, Warte 253 Minuten, laufe in den Abschnitt 66, Warte 327 Minuten, laufe in den Abschnitt 68, Warte 2828 Minuten, laufe in den Abschnitt 69, Warte 3733 Minuten, laufe in den Abschnitt 70, Warte 2247 Minuten, laufe in den Abschnitt 71, Warte 6548 Minuten, laufe in den Abschnitt 72, Warte 1921 Minuten, laufe in den Abschnitt 80, Warte 3325 Minuten, laufe zum Grabmal
Historical Steps:
[199166868, 199168543, 199168875, 199168875, 199169415, 199173545, 199173545, 199173545, 199174719, 199176060, 199176696, 199176696, 199176696, 199177869, 199185107, 199185729, 199185729, 199185729, 199187352, 199188920, 199190205, 199191216, 199191216, 199191216, 199191216, 199191216, 199191216, 199191216, 199191216, 199192050, 199192050, 199192050, 199192050, 199192050, 199192050, 199192050, 199192914, 199193016, 199199007, 199199007, 199199028, 199199028, 199199028, 199199028, 199199028, 199201221, 199201410, 199201410, 199201410, 199201410, 199201410, 199203108, 199205139, 199205325, 199207577, 199210116, 199211430, 199211447, 199213217, 199213217, 199213217, 199213217, 199213217, 199213470, 199213470, 199213470, 199213797, 199213797, 199216625, 199220358, 199222605, 199229153, 199231074, 199231074, 199231074, 199231074, 199231074, 199231074, 199231074, 199231074, 199234399, 199234399, 199234399, 199234399, 199234399]
Execution Time: 0.5944 seconds

Edge Case Example

Input:
Number of Sections: 4
Section Frequencies: 20, 10, 4, 7

Output: Warte 20 Minuten, laufe in den Abschnitt 1, Warte 10 Minuten, laufe in den Abschnitt 2, Warte 6 Minuten, laufe zum Grabmal
Historical Steps: [20, 30, 36, 36]
Execution Time: 0.0000 seconds (practically negligible)

4 Source Code

The most important parts of our code are the following:

```
1
ANZAHL_QUADER = GRABMAL_DATA_LIST[0]
3 QUADER_FREQUENZ = GRABMAL_DATA_LIST[1:]

5 def pruefeQuader(aktuelleZeit, aktuellerIndex):
    quaderWert = aktuelleZeit//QUADER_FREQUENZ[aktuellerIndex]
7     if quaderWert % 2 == 0:
        return False
9     else:
        return True
11
```

```
def pruefeFortschritt(aktuelleZeit, aktuellerIndex):
    naechsterQuaderWert1 = (aktuelleZeit//QUADER_FREQUENZ[aktuellerIndex-1]+1)*QUADER_FREQUENZ[aktuellerIndex-1]
    naechsterQuaderWert2 = (aktuelleZeit//QUADER_FREQUENZ[aktuellerIndex]+1)*QUADER_FREQUENZ[aktuellerIndex]
    if naechsterQuaderWert1 > naechsterQuaderWert2:
        return True
    else:
        return False

def rueckwaertsPropagatione(schrittListe, aktuellerIndex):
    if aktuellerIndex == 0:
        return False

    naechsterQuaderWert1 = (schrittListe[aktuellerIndex-1]//QUADER_FREQUENZ[aktuellerIndex-1]+2)*QUADER_FREQUENZ[aktuellerIndex-1]
    naechsterQuaderWert2 = (schrittListe[aktuellerIndex-2]//QUADER_FREQUENZ[aktuellerIndex-2]+1)*QUADER_FREQUENZ[aktuellerIndex-2]
    if naechsterQuaderWert1 < naechsterQuaderWert2:
        return False
    else:
        return True

def rueckwaertsPropagatioSchleife(schrittListe, aktuellerIndex):
    tempSchrittListe = schrittListe.copy()
    while True:
        if rueckwaertsPropagatione(schrittListe, aktuellerIndex) == False:
            if len(schrittListe) == 0:
                schrittListe.append((tempSchrittListe[aktuellerIndex]//QUADER_FREQUENZ[aktuellerIndex]+2)*QUADER_FREQUENZ[aktuellerIndex])
                aktuellerIndex += 1
            else:
                quaderWert = (schrittListe[aktuellerIndex-1]//QUADER_FREQUENZ[aktuellerIndex-1]+2)*QUADER_FREQUENZ[aktuellerIndex-1]
                schrittListe.pop()
                schrittListe.append(quaderWert)
            return schrittListe, aktuellerIndex
        else:
            aktuellerIndex -= 1
            schrittListe.pop()

def erstelleAnweisungen(zeitmarken):
    anweisungen = []
    anweisungen.append(f"Warte {zeitmarken[0]} Minuten")
    anweisungen.append(f"laufe in den Abschnitt 1")
    for i in range(1, len(zeitmarken)):
        warteMinuten = zeitmarken[i] - zeitmarken[i-1]
        anweisungen.append(f"Warte {warteMinuten} Minuten")
        if i < len(zeitmarken)-1:
            anweisungen.append(f"laufe in den Abschnitt {i+1}")
        else:
            anweisungen.append("laufe zum Grabmal")

    ergebnisListe = []
```

```
    index = 0

    while index < len(anweisungen):
        if anweisungen[index] == "Warte 0 Minuten":
            if ergebnisListe:
                ergebnisListe.pop()
        else:
            ergebnisListe.append(anweisungen[index])
        index += 1
    anweisungen = ergebnisListe
    return ', '.join(anweisungen)

def rekursiveAlgorithmus():
    aktuellerIndex = 1
    aktuelleZeit = QUADER_FREQUENZ[aktuellerIndex-1]
    schrittListe = [aktuelleZeit]
    while True:
        if pruefeQuader(aktuelleZeit, aktuellerIndex):
            aktuellerIndex += 1
            schrittListe.append(aktuelleZeit)
        elif pruefeFortschritt(aktuelleZeit, aktuellerIndex):
            aktuelleZeit = (aktuelleZeit//QUADER_FREQUENZ[aktuellerIndex
    ]+1)*QUADER_FREQUENZ[aktuellerIndex]
            aktuellerIndex += 1
            schrittListe.append(aktuelleZeit)
        else:
            schrittListe, aktuellerIndex = rueckwaertsPropagatioSchleife
    (schrittListe, aktuellerIndex)
            aktuelleZeit = schrittListe[-1]
        if aktuellerIndex == ANZAHL_QUADER:
            print(schrittListe)
            anweisungen = erstelleAnweisungen(schrittListe)
            print(anweisungen)
            break
```