Problem Set 5
Erdun E
Nov 05, 2024

CS 5800: Algorithm
Problem Set 5

Exercises

6.1 Question

A contiguous subsequence of a list S in a subsequence made up of consecutive elements of S. For instance, if S is

5, 15, -30, 10, 05, 40, 10,

then 15, -30, 10 is a contiguous subsequence but 5, 15, 40 is not. Given a linear-time algorithm for the following task:

Input: A list of numbers, $a_1, a_2, ..., a_n$**.**

Output: The contiguous subsequence of maximum sum (a subsequence of length zero has sum zero)

For the preceding example, the answer would be 10, -5, 40, 19 with a sum of 55.

(Hint: For each $j \in \{1, 2, ..., n\}$**), consider contiguous subsequences ending exactly at position j.)**

6.1 Solution

Step 1: Define the state

- Let dp[i] represent the maximum sum of the contiguous subsequence and ends at index i.

Step 2: Define the transition formula

For each i, there are two choices that

- Extend the previous subsequence end at i - 1 to include arr[i], that dp[i - 1] + arr[i]
- Or start a new subsequence begin at arr[i]
- So the transition formula is dp[i] = max(dp[i - 1] + arr[i], arr[i])

Step 3: Track the start and end index of the maximum subsequence

- Add start and end variables to track the start and end index of the maximum sum subsequence.
- Use a tempStart variable to track the start index of a potential new subsequence. when decide to start a new subsequence at arr[i], set tempStart to i
- Each time dp[i] makes a new maximum sum, update maxSum, start, and end accordingly.

Step 4: Initial Value

- Set dp[0] = arr[0], and start and end to 0, and maxSum to dp[0]

Step 5: Traverse the Array and Apply the Transition Formula

- Starting from the second element, apply the transition formula and update maxSum, start, and end as necessary.

Step 6: Return the Results

- After the traversal, maxSum will store the maximum sum, and arr[start:end + 1] will represent the contiguous subarray that yields this sum.

Based on the above analysis we can get the pseudocode as:

Algorithm 1 The Contiguous Subsequence Of Maximum Sum

```
Input  : An integer array $arr$ of length $n$
Output: A pair containing the maximum sum and the contiguous subsequence
if $n = 0$ then
    return $(0, \text{empty list})$
Initialize $dp[0] \leftarrow arr[0]$
Initialize $maxSum \leftarrow dp[0]$
Initialize $start \leftarrow 0$, $end \leftarrow 0$, and $tempStart \leftarrow 0$
for $i \leftarrow 1$ to $n - 1$ do
    if $dp[i-1] + arr[i] > arr[i]$ then
        $dp[i] \leftarrow dp[i-1] + arr[i]$
    else
        $dp[i] \leftarrow arr[i]$
        $tempStart \leftarrow i$
    if $dp[i] > maxSum$ then
        $maxSum \leftarrow dp[i]$
        $start \leftarrow tempStart$
        $end \leftarrow i$
Initialize an empty list $subsequence$
for $i \leftarrow start$ to $end$ do
    Add $arr[i]$ to $subsequence$
return $(maxSum, subsequence)$
```

Execute the pseudocode step by step:

Step 1: Initial

- dp[0] = 5
- maxSum = 5
- start = 0,end = 0
- tempStart = 0

Step 2: Index i = 1

- arr[1] = 15
- dp[0] + arr[1] = 5 + 15 = 20
- dp[1] = 20
- maxSum = 20, start = 0, end = 1

Step 3: Index i = 2

- arr[2] = -30
- dp[1] + arr[2] = 20 - 30 = -10
- dp[2] = -10
- tempStart = 2
- maxSum remains 20

Step 4: Index i = 3

- arr[3] = 10
- dp[2] + arr[3] = -10 + 10 = 0
- dp[3] = 10
- tempStart = 3
- maxSum remains 20

Step 5: Index i = 4

- arr[4] = -5
- dp[3] + arr[4] = 10 - 5 = 5
- dp[4] = 5
- tempStart remains 3
- maxSum remains 20

Step 6: Index i = 5

- arr[5] = 40
- dp[4] + arr[5] = 5 + 40 = 45
- dp[5] = 45
- maxSum = 45, start = 3, end = 5

Step 7: Index i = 6

- arr[6] = 10
- dp[5] + arr[6] = 45 + 10 = 55
- dp[6] = 55
- maxSum = 55, start = 3, end = 6

Final Result

- Maximum Sum: 55
- Contiguous Subsequence with Maximum Sum: [10, -5, 40, 10]

Summarized in the table:

Index i	Array Value $arr[i]$	$dp[i-1] + arr[i]$	$dp[i]$	tempStart	maxSum	start	end
0	5	N/A	5	0	5	0	0
1	15	20	20	0	20	0	1
2	-30	-10	-10	2	20	0	1
3	10	0	10	3	20	0	1
4	-5	5	5	3	20	0	1
5	40	45	45	3	45	3	5
6	10	55	55	3	55	3	6

6.5 Question

Pebbling a checkerboard. We are given a checkerboard which has 4 rows and n columns, and has an integer written in each square. We are also given a set of 2n pebbles, and we want to place some or all of these on the checkerboard (each pebble can be placed on exactly one square) so as to maximize the sum of the integers in the squares that are covered by pebbles. There is one constraint: for a placement of pebbles to be legal, no two of them can be on horizontally or vertically adjacent squares (diagonal adjacency is fine)

(a) Determine the number of legal patterns that can occur in any column (in isolation, ignoring the pebbles in adjacent columns) and describe these patterns.

Call two patterns compatible if they can be placed on adjacent columns to form a legal placement. Let us consider subproblems consisting of the first k columns $1 \leq k \leq n$. Each subproblem can be assigned a type, which is the pattern occurring in the last column.

(b) Using the notions of compatibility and type, give an O(n)-time dynamic programming algorithm for computing an optimal placement

6.5 Solution

Solution (a)

Step 1: Identify Legal Patterns for a Single Column

- Given in a single 4-row column, pebbles cannot be placed in adjacent rows. Define each row as:
- 1 pebble or 0 pebble

Step 2: Legal Patterns

- There are 7 legal patterns for a column:
- Pattern 1: $[0, 0, 0, 0]$ – No pebbles
- Pattern 2: $[1, 0, 0, 0]$ – Pebble in 1st row
- Pattern 3: $[0, 1, 0, 0]$ – Pebble in 2nd row
- Pattern 4: $[0, 0, 1, 0]$ – Pebble in 3rd row
- Pattern 5: $[0, 0, 0, 1]$ – Pebble in 4th row
- Pattern 6: $[1, 0, 1, 0]$ – Pebbles in 1st and 3rd rows
- Pattern 7: $[0, 1, 0, 1]$ – Pebbles in 2nd and 4th rows

Step 3: Dynamic Programming Approach

- Define $dp[c][p]$ as the maximum sum for column c with pattern p. Transition depends on:
- Adding values from the current column pattern p
- Maximizing based on compatible patterns in the previous column

Step 4: Final Solution:

- Compute the highest $dp[n-1][p]$ for the last column to get the maximum sum, ensuring no adjacency violations.

Step 5: Table of Legal Patterns

Pattern	Representation	Description
1	[0, 0, 0, 0]	No pebbles
2	[1, 0, 0, 0]	Pebble in 1st row
3	[0, 1, 0, 0]	Pebble in 2nd row
4	[0, 0, 1, 0]	Pebble in 3rd row
5	[0, 0, 0, 1]	Pebble in 4th row
6	[1, 0, 1, 0]	Pebbles in 1st and 3rd rows
7	[0, 1, 0, 1]	Pebbles in 2nd and 4th rows

Solution (b)

Algorithm 2 Optimal Pebble Placement on Checkerboard

Input : A $4 \times n$ integer matrix *board*, and a set of patterns and their compatibilities
Output: The maximum achievable sum from placing pebbles legally

```
Define $patterns \leftarrow \{7 \text{ legal patterns for a column}\}$
Initialize $dp[1][p] \leftarrow$ sum of values covered by pattern $p$ in column 1 for each pattern $p$
for $c \leftarrow 2$ to $n$ do
    for each pattern $p$ in patterns do
        $dp[c][p] \leftarrow \max(dp[c-1][q] + \text{sum}(p, c))$, where $q$ is compatible with $p$
$maxSum \leftarrow \max(dp[n][p]$ for all $p \in patterns)$
return $maxSum$
```

Step 1: Define Legal Patterns

- From part (a), we've identified 7 legal patterns for each column, which are following:
- Pattern 1: $[0,0,0,0]$
- Pattern 2: $[1,0,0,0]$
- Pattern 3: $[0,1,0,0]$
- Pattern 4: $[0,0,1,0]$
- Pattern 5: $[0,0,0,1]$
- Pattern 6: $[1,0,1,0]$
- Pattern 7: $[0,1,0,1]$

Step 2: Compatibility of Patterns

- Define two patterns as compatible if they can legally appear in adjacent columns.
- For instance, patterns with overlapping pebble rows.

Step 3: Dynamic Programming Table Setup

- Let $dp[c][p]$ represent the maximum achievable sum for column c ending with pattern p.
- p ranges from 1 to 7, representing the legal patterns.
- The final solution will be the maximum of $dp[n][p]$ across all patterns p in the last column.

Step 4: Transition Formula

- For each column c and each pattern p in that column:
- dp[c][p] = max {dp[c-1][q] + sum(p, c)}
- $\text{sum}(p, c)$: Sum of values covered by pattern p in column c.
- q: Pattern in column $c - 1$ that is compatible with p.

Step 5: Initialization and Base Case

- For the first column $c = 1$:
- dp[1][p] = sum(p, 1)
- $\forall p \in \{1, 2, \ldots, 7\}$
- Initialize each $dp[1][p]$ with the sum of integers in column 1 covered by pattern p.

Step 6: Fill the DP Table

- Iterate through each column $c = 2$ to n.
- Compute $dp[c][p]$ using the transition formula, ensuring compatibility with previous columns.

Step 7: Extract Optimal Solution

- The maximum achievable sum for the entire board is:
- $\max\{dp[n][p] \mid p \in \{1, 2, \ldots, 7\}\}$
- This provides the maximum sum with all placement constraints.

Step 8: Final Algorithm Complexity

- The algorithm runs in $O(n)$ time.
- Each column computation involves a constant number of patterns that 7 patterns with fixed compatibility checks.

Step 9: Table of Legal Patterns and Compatibility

Pattern	Representation	Compatible Patterns
1	[0, 0, 0, 0]	All patterns
2	[1, 0, 0, 0]	Patterns 1, 3, 4, 5, 7
3	[0, 1, 0, 0]	Patterns 1, 2, 4, 5, 6
4	[0, 0, 1, 0]	Patterns 1, 2, 3, 5, 7
5	[0, 0, 0, 1]	Patterns 1, 2, 3, 4, 6
6	[1, 0, 1, 0]	Patterns 1, 3, 5
7	[0, 1, 0, 1]	Patterns 1, 2, 4