Theory HW 1 Q7

Team 12
Department of Computer Science
University of Houston

I. Question 7

Write a greedy algorithm to compute the set cover with input k subsets of letters. Show an input example where the algorithm is slow. Show an input example where the algorithm is fast.

II. Pseudo-code

GreedySetCover
Input: A ground set U and collection of subsets S Output: A collection of sets C that covers U

```
function GREEDYSETCOVER(U, S)
C = null
while C != U do
{
A = element of S that maximizes number of uncovered elements
C = C U A
}
return C
```

III. Algorithm

```
while GroundSet <> [] do
begin
  bestIndex := 0;
  bestCount := 0;
  for i := 1 to NumberOfSubsets do
  begin
    if not used[i] then
    begin
      tempCount := CountIntersection(S[i], U);
      if tempCount > bestCount then
      begin
        bestCount := tempCount;
        bestIndex := i;
      end;
    end;
  end;
```

IV. Input examples

Using the explanation from our textbook. We know, assuming there is a set of subsets that form the optimal cover, that there is at least one subset that that has at least (uncovered elements) / (number of subsets in optimum cover). Since this algorithm selects the subset that contains the most uncovered elements. Each iteration of the algorithm will cover that many elements. Thus, if we have fewer, larger subsets that include more values from our ground set, the program will iterate fewer times than if we have many subsets with few values. For example, for ground set U a,b,c,d,e,f,g,h, our program will run more quickly with subsets a, b, c, d, e, f, g, h than subsets a, b, c, d, e, f, g, h.

V. Time Complexity

The time complexity of this algorithm is theta(tlog(n/t)) where t is the number of uncovered elements at beginning of iteration n and n/t represents the number of elements that are covered each iteration.

VI. AI Tools Used

ChatGPT v3.5 was used to generate the Pascal code. Algorithms used come from Pandurangan 2022.

VII. Question 9

Show an example of MCM with n = 8 matrices where DP produces about the same result with a naive left-right multiplication. Show an example of MCM with n = 8 matrices where DP produces signficantly better performance than a naive left-right multiplication. Rewrite the MCM algortihm with recursion, instead of loops

VIII. Pseudo-code

NaiveMCM
Input: A sequence of matrix dimensions: p = p0..pn Output: The optimal cost to multiply Ai..Aj

```
function NaiveMCM(p)
if i == j
return 0
m[i,j] = infinity
for k = i to j - 1 do
{
q = NaiveMCM(p, i, k) + NaiveMCM(p, k+1, j) + PartialProducts
}
if q ¡ m[i,j] then
m[i,j] = q
return m[i,j]
```

IterativeMCM
Input: A sequence of matrix dimensions p Output: The optimal cost to multiply A1..An

```
function IterativeMCM(p)
for i = 1 to n do
m[i,i] = 0
for len = 2 to n do
{
```

```
for i = 1 to n - len + 1 do
{
j = i + len - 1
m[i,j] = infinity
for k = i to j - 1 do
q = m[i,k] + m[k+1,j] + PartialProducts
if q ¡ m[i,j] then
m[i,j] = q
}
}
return m[1,n]
```

MemoizationMCM

Input: A sequence of matrix dimensions p Output: The optimal cost to multiply A1..An

```
function MemoizationMCM(p)
return MemoizationMCM(p, 1, n)
function MCMHelper(p, i, j)
{
if m[i,j] == infinity
if i == j then
m[i,j] = 0
else
{
for k = i to j - 1 do
left = MCMHelper(p, i, k)
right = MCMHelper(p, k+1, j
total = left + right + PartialProducts
if total ¡ m[i,j] then
m[i,j] = total
return m[i,j]
}
}
return m[1,n]
```

IX. Algorithm

```
{ Function to multiply two matrices (Naive) }
function NaiveMCM(A: Matrix; RA, CA: integer;
B: Matrix; RB, CB: integer;
var Result: Matrix): boolean;
var
  i, j, k: integer;
begin
  if CA <> RB then
  begin
    writeln('Matrix multiplication
            not possible');
    NaiceMCM := false;
    exit;
  end;

  { Initialize result matrix }
  for i := 1 to RA do
    for j := 1 to CB do
      Result[i, j] := 0;

  { Perform multiplication }
  for i := 1 to RA do
    for j := 1 to CB do
      for k := 1 to CA do
        Result[i, j] := Result[i, j]
        + (A[i, k] * B[k, j]);

  NaiveMCM := true;
end;

function IterativeMCM(n: integer): LongInt;
begin
  { Initialize diagonal elements to 0,
  as cost of multiplying a single matrix is zero }
  for i := 1 to n do
    m[i, i] := 0;

  { len is the length of the subchain
  being considered (starts from 2
                  matrices up to n) }
  for len := 2 to n do
  begin
    for i := 1 to n - len + 1 do
    begin
      j := i + len - 1;
      m[i, j] := INF; { Initialize to
                        a large value }

      for k := i to j - 1 do
      begin
        q := m[i, k] + m[k + 1, j] +
        (p[i - 1] * p[k] * p[j]);

        if q < m[i, j] then
          m[i, j] := q;
      end;
    end;
  end;

  { The minimum cost to multiply A1..An
  is stored in m[1, n] }
  IterativeMCM := m[1, n];
end;


{ Recursive function with
memoization to compute MCM }
function RecursiveMCM
(i, j: integer): LongInt;
var
  k, q: integer;
begin
  if i = j then
    Exit(0); { Single matrix has
                zero multiplication cost }
```

```
  if m[i, j] <> -1 then
    Exit(m[i, j]); { Return already
                     computed value }

  m[i, j] := INF; { Initialize to infinity }

  for k := i to j - 1 do
  begin
    q := RecursiveMCM(i, k) +
    RecursiveMCM(k + 1, j) +
    (p[i - 1] * p[k] * p[j]);

    if q < m[i, j] then
      m[i, j] := q;
  end;

  Exit(m[i, j]);
end;
```

X. Input examples

If all matrixes had the same dimensions, or were already arranged optimally. Then the naive implementation and dynamic implementations would produce the same results. If the matrixes all had different dimensions and were not arranged optimally than dynamic programming would produce far better results. The left columns below represents input of matrices that would result in similar performance between algorithms, the column on the right represents input that would benefit from dynamic programming.

```
1 = 10x10                   1 = 10x100
2 = 10x10                   2 = 20x20
3 = 10x10                   3 = 10x500
4 = 10x10                   4 = 50x50
5 = 10x10                   5 = 120x40
6 = 10x10                   6 = 30x30
7 = 10x10                   7 = 100x100
8 = 10x10                   8 = 10x10
```

XI. Time Complexity

The time complexity of this algorithm is $theta(n^2)$. We get this result because will iterate n times, each iteration will consist of n-k steps. We are able to achieve this because we do not solve the same problems more than once.

XII. AI Tools Used

ChatGPT v3.5 was used to generate the Pascal code. Algorithms used come from Pandurangan 2022.

XIII. Question 10 part 1

Optimized recursion on overlapping subproblems: Fibonacci numbers: write a function with forward recursion simulating a loop, going from 1 to n, exploiting a table with partial results or 2 temporary variables. Then write a 2nd version with recursion, but storing the partial results with the memoization technique. Then write the slow classical recursive version with exponential. Show step by step examples choose some n random value s.t. 8 ¡ n ¡ 12 (different teams should choose different n!).

XIV. ForwardRecursiveFibonacci

```
procedure ForwardFibonacci(n: Integer;
                           a, b: Integer);
var
  temp: Integer;
begin
  if n > 0 then
  begin
    temp := a + b;  { Compute the next
                      Fibonacci number }
    Write(temp, ' '); { Output the
                        Fibonacci number }
    ForwardFibonacci(n - 1, b, temp);
    {^^ Recursive call with updated values }
  end;
end;
```

XV. MemoizationFibonacci

```
function MemoizationFibonacci
(n: Integer): Integer;
begin
  if memo[n] <> -1 then  { Check if
                           the value is
                           already computed }
    Exit(memo[n]);

  if (n = 0) then
    memo[n] := 0
  else if (n = 1) then
    memo[n] := 1
  else
    memo[n] := MemoizationFibonacci(n - 1) +
    MemoizationFibonacci(n - 2);
    { Store computed value }

  Exit(memo[n]);
end;
```

XVI. SlowFibonacci

```
function Fibonacci(n: Integer): Integer;
begin
  if (n = 0) then
    Fibonacci := 0
  else if (n = 1) then
    Fibonacci := 1
  else
    Fibonacci := Fibonacci(n - 1) +
    Fibonacci(n - 2);  { Recursive calls }
end;
```

XVII. Time Complexity

Forward Fibonacci utilizing the temporary variable has theta(n) of n^2 as it does not need to recalculate every Fibonacci number leading up to the one we are looking for, it calculates the Fibonacci number by storing the previous 2 variables in temporary variables. Memoization Fibonacci takes this a step further by storing Fibonacci values in a table that can be referenced every time the procedure is called. This procedure also has n^2 run time, but in practice is much faster than simple forward recursion. The slow recursive version has theta(n) of 2^n.

XVIII. AI Tools Used

ChatGPT v3.5 was used to generate the Pascal code. Algorithms used come from Pandurangan 2022.

XIX. Question 10 part 2

Optimized recursion on overlapping subproblems: Show a step by step example aligning two sequences of length 10 combining 5 symbols with optimized recursion (dynamic prog.) algorithm varying the gap score and mismatch scores.

XX. Pseudo-code

SimScoreDP
Input: sequences s and t of lengths m and n
Output: sim(s, t)

```
function SimScoreDP(s, t)
for i = 0 to m do
a[i, 0] = -i
for j = 0 to n do
a[0,j] = -j
for i = 1 to m do
{
for j = 1 to n do
{
both = a[i - 1, j - 1] + score(s[i], t[j])
left = a[i, j - 1] + score(-, t[j])
right = a[i - 1, j] + score(s[i], -)
a[i, j] = max(both, left, right)
}
}
return a[m, n]
```

XXI. Algorithm

```
{ Function implementing dynamic programming
to compute similarity score }
function SimScoreDP(s, t: string): integer;
var
  i, j: integer;
begin
  m := Length(s);
  n := Length(t);

  { Base case initialization }
  for i := 0 to m do
    a[i, 0] := -i; { Deletion penalty }
  for j := 0 to n do
    a[0, j] := -j; { Insertion penalty }

  { Fill DP table using the
  given recurrence }
  for i := 1 to m do
  begin
    for j := 1 to n do
    begin
      a[i, j] := Max(
        { Diagonal (match/mismatch) }
        a[i - 1, j - 1] + Score(s[i], t[j]),
        { Left (insertion) }
        a[i, j - 1] + Score('-', t[j]),
        { Up (deletion) }
        a[i - 1, j] + Score(s[i], '-')
      );
    end;
  end;

  { Return final similarity score }
  SimScoreDP := a[m, n];
end;
```

XXII. Input examples

```
Input
s = AABCCBAABC
t = ACCBBAABCC
Output
AABCC-BAAB-C
-A-CCBBAABCC
Score: 12

Input
s = ABCDEABCDE
t = EDCBAEDCBA
Output
ABCDEA-BCDE
EDC-BAEDCBA
Score: -2

Input
s = AAABBBCCDD
t = BBBAAACCDD
Output
AAABBB---CCDD
---BBBAAACCDD
Score: 8
```

XXIII. Time Complexity

When implimented in this way, the time complexity of this algorithm is theta(n^2)

XXIV. AI TOOLS USED

ChatGPT v3.5 was used to generate the Pascal code. Algorithms used come from Pandurangan 2022.