Solutions to Homework Assignment 1
CS 535 Design and Analysis of Algorithms
Fall Semester, 2016

Solution:

1. Let \( G(i, j) \) be the maximal amount of gold we can collect by moving from point \((1,1)\) to point \((i, j)\).
   The recurrence relation is:
   \[
   G(i, j) = \begin{cases} 
   g_{1,1} & \text{if } i = j = 1 \\
   G(i, j - 1) + g_{i,j} & \text{if } i = 1, j > 1 \\
   G(i - 1, j) + g_{i,j} & \text{if } i > 1, j = 1 \\
   \max\{G(i - 1, j), G(i, j - 1)\} + g_{i,j} & \text{if } i > 1, j > 1 
   \end{cases}
   \]
   As an alternative, if you define \( G(i, j) \) be the maximal amount of gold we can collect by moving from point \((i, j)\) to point \((n, n)\). The recurrence relation would be
   \[
   G(i, j) = \begin{cases} 
   g_{1,1} & \text{if } i = j = n \\
   G(i, j + 1) + g_{i,j} & \text{if } i = n, j < n \\
   G(i + 1, j) + g_{i,j} & \text{if } i < n, j = n \\
   \max\{G(i + 1, j), G(i, j + 1)\} + g_{i,j} & \text{if } i < n, j < n
   \end{cases}
   \]
   Let \( g \) be a 2D array storing the gold amount of each grid point, which is given as input.

   **Unmemoized recursive algorithm:** Please see Algorithms 1 and 2 (a wrapper and the recursive algorithm).

   **Algorithm 1: UNMEMOIZED-GOLD-COLLECT\((g, n)\)**
   
   1: return \( \text{GOLD-COLLECT-RECURSIVE}(g, n, n) \)

   **Memoized iterative algorithm:** Please see Algorithm 3. Let \( G \) be a 2D array such that \( G[i][j] \) is the maximal amount of gold we can collect by moving from point \((1,1)\) to point \((i, j)\). Let \( \text{Prev} \) be a 2D boolean array to store the direction from previous spot to current spot: true for up, false for right.

   **Time complexity:** For the unmemoized recursive method, \( T(i, j) = T(i - 1, j) + T(i, j - 1) + c \). If we define \( S(i, j) = T(i, j) + c \), we have \( S(i, j) = S(i - 1, j) + S(i, j - 1) \). For simplicity, let’s assume \( S(2, 1) = S(1, 2) = 1 \), then this is exactly the recurrence of binomial coefficient, as shown in Figure 1. We can see Pascal’s triangle by rotating this figure 135 degrees clockwise. According to it, \( S(2, 2) = \binom{2}{1} \), \( S(3, 3) = \binom{4}{2} \), \( S(4, 4) = \binom{6}{3} \), and similarly we have \( S(n, n) = \binom{2n}{n-1} \). Thus \( T(n, n) = \Theta\left(\binom{2n}{n-1}\right) \sim \Theta\left(\frac{4^n}{\sqrt{n}}\right) \). For the memoized iterative method, we calculate values for each slot of the matrices \( G \) and \( \text{prev} \), so the complexity \( \Theta(n^2) \).

---

\(^1\)An asymptotic for the central binomial coefficient (https://en.wikipedia.org/wiki/Central_binomial_coefficient)
Algorithm 2 GOLD-COLLECT-RECURSIVE($g, i, j$)

1: if $i < 1$ or $j < 1$ then
2: return 0
3: if $i = 1$ and $j = 1$ then
4: return $g[1][1]$
5: if $i = 1$ and $j > 1$ then
6: return GOLD-COLLECT-RECURSIVE($g, i, j - 1$) + $g[i][j]$
7: if $i > 1$ and $j = 1$ then
8: return GOLD-COLLECT-RECURSIVE($g, i - 1, j$) + $g[i][j]$
9: $G1 = $GOLD-COLLECT-RECURSIVE($g, i - 1, j$)
10: $G2 = $GOLD-COLLECT-RECURSIVE($g, i, j - 1$)
11: if $G1 > G2$ then
12: return $G1 + g[i][j]$
13: else
14: return $G2 + g[i][j]$

Algorithm 3 MEMOIZED-GOLD-COLLECT($g, n$)

1: if $n = 1$ then
2: return 0
3: if $n = 1$ then
4: return $g[1][1]$
5: Let $G[1..n][1..n]$ be a $n \times n$ double array and $prev[1..n][1..n]$ be a $n \times n$ boolean array
6: $G[1][1] = g[1][1]$
7: for $j = 2$ to $n$ do
8: $G[1][j] = G[1][j - 1] + g[1][j]$
9: $prev[1][j] = \text{up}$ //true for up
10: for $j = 2$ to $n$ do
11: $G[j][1] = G[j - 1][1] + g[j][1]$
12: $prev[j][1] = \text{right}$ //false for right
13: for $i = 2$ to $n$ do
14: for $j = i$ to $n$ do
15: if $G[i][j - 1] > G[i - 1][j]$ then
16: $G[i][j] = G[i][j - 1] + g[i][j]$
17: $prev[i][j] = \text{up}$
18: else
19: $G[i][j] = G[i - 1][j] + g[i][j]$
20: $prev[i][j] = \text{right}$
21: for $j = i + 1$ to $n$ do
22: if $G[j - 1][i] > G[j][i - 1]$ then
23: $G[j][i] = G[j - 1][i] + g[j][i]$
24: $prev[j][i] = \text{right}$
25: else
26: $G[j][i] = G[j][i - 1] + g[j][i]$
27: $prev[j][i] = \text{up}$
28: Let $L$ be a empty sequence which will store the optimal sequence of steps
29: $i = n, j = n$
30: while $i > 1$ or $j > 1$ do
31: Append $prev[i][j]$ to $L$
32: if $prev[i][j] = \text{up}$ then
33: $j - -$
34: else
35: $i - -$
36: Reverse $L$ and print it
37: return $G[n][n]$
2. Denote the given string as $s$, let $M(i, j)$ be the minimal number of split substrings of $s[i, j]$ (the substring of $s$ from index $i$ to $j$, inclusively) such that each of the splits is a palindrome. The recurrence relation is

$$M(i, j) = \begin{cases} 
1 & \text{if } s[i, j] \text{ is a palindrome} \\
\min_{i \leq k < j} \{M(i, k) + M(k + 1, j)\} & \text{if } s[i, j] \text{ is not a palindrome}
\end{cases}$$

**Unmemoized recursive algorithm:** Please see Algorithms 4,5,6

**Algorithm 4** `bool isPalindrome(const char *s, int i, int j)`
1: while (i<j) //Ignore middle char in odd length strings do
2: if (s[i++] != s[j--]) then
3: return false;
4: return true;

**Algorithm 5** `int minPalindromesWrapper(const char *s)`
1: return `minPalindromesRecursive(s, 0, strlen(s)-1)`;

**Memoized iterative algorithm:** Please see Algorithms 4,7,8. Algorithm 7 recursively outputs how we split optimally.

**Time complexity:** For the unmemoized recursive method, let $T(n)$ be the worst case time for splitting a string of length $n$, then we have

$$T(n) = (T(1) + T(n-1)) + (T(2) + T(n-2)) + \cdots + (T(n-1) + T(1)) + cn$$

$$= cn + 2 \sum_{1 \leq i < n} T(i). \quad (1)$$

Then $T(n) - T(n-1) = c + 2 \sum_{1 \leq i < n} T(i) - 2 \sum_{1 \leq i < n-1} T(i) = c + 2T(n-1)$, so $T(n) = 3T(n-1) + c$. By solving the recurrence using annihilators, we get $T(n) = \Theta(3^n)$. For the memoized iterative method, the complexity is $O(n^3)$ since there are 3 for-loops.

3. **PhD problem:** For Problem 2, the code for iterative method was given in Algorithms 4,7,8. Non-trivial examples of input/output are as follows:
Algorithm 6 int minPalindromesRecursive(const char *s, i, j)
1: if (isPalindrome(s, i, j)) then
2: return true;
3: min=j-i+1;
4: for (k=i; k<j; k++) do
5: int sum = minPalindromesRecursive(s, i, k)+minPalindromesRecursive(s, k+1, j);
6: if (sum<min) then
7: min=sum;
8: return min;

Algorithm 7 int minPalindromesIterative(const char *s)
1: int i, j, k, len;
2: int n = strlen(s);
3: int **M; // Allocate n x n array (not shown)
4: int **P; // To track the split positions
6: for (len = 1; len≤n; len++) do
7: for (i = 0; i≤n - len; i++) do
8: j = i + len - 1;
9: if (ispalindrome(s, i, j)) then
10: M[i][j] = 1;
11: P[i][j] = -1; //no split needed
12: else
13: M[i][j] = len; // Assume worst-case. Improve on it below.
14: // Consider splitting string at each position.
15: for (k = i; k<j; k++) do
16: int sum = M[i][k] + M[k+1][j];
17: if (sum<M[i][j]) then
18: M[i][j] = sum; // Save minimum value
19: P[i][j] = k;
20: string str(s);
21: printSubstrings(P, str, 0, strlen(s));
22: return M[0][n-1]; // Return result for entire string

Algorithm 8 int printSubstrings(const int **P, const string str, i, j)
1: if (P[i][j]==-1) then
2: cout<<str.substr(i, j-i+1)<<" ",
3: else
4: printSubstrings(P, str, i, P[i][j]);
5: printSubstrings(P, str, P[i][j]+1, j);
• abababab → 2: aba, babab
• abaababa → 2: aba, ababa
• neveroddoreveneven → 3: neveroddoreven, eve, n
• aswastoobadihidaboot → 4: as, w, as, toobadihidaboot
• sawiwastoobadihidaboot → 2: sawiwas, toobadihidaboot

Grading Criteria

20 points in total. For each of the two problems, 10 points are split as follows:

• (2') recurrence relation (including base cases)
• (3') unmemoized recursive algorithm. Base cases should also be presented.
• (3') memoized iterative algorithm. Note you need to output not only the optimum value, but also the way that value is obtained.
• (1') complexity analysis for the unmemoized recursive algorithm. You must present how the complexity is derived, e.g., by solving the recurrence relation of $T(n)$ or other methods.
• (1') complexity analysis for the memoized iterative algorithm.