# Programming Assignment #2: Maximum Planar Subs Problem

## B10901085 李昀儒

- **Algorithm**

  Use dynamic programming to solve this problem: use M[i][j] to store answer of subproblem M(i,j) and C[i][j] to get the corresponding non-overlapped chords.
  For M[i][j],
  if vertices i and j share the same chord, M[i][j] = 1+M[i+1][j-1]. (case1)
  Otherwise, if the vertex shares the same chord with vertex i is not in [i,j](case2),
  M[i][j] = M[i+1][j].
  Otherwise, if the vertex shares the same chord with vertex i is vertex i+1(case3),
  M[i][j] = 1+M[i+2][j].
  Otherwise, if the vertex shares the same chord with vertex i is vertex j-1(case4),
  M[i][j] = 1+M[i+1][j-2].
  Otherwise, compare value of M[i+1][j] (not choosing the chord with one end at vertex i(same as case2)) and 1+M[i+1][(another vertex of chord whose one end is at vertex i)-1]+M [(another vertex of chord whose one end is at vertex i)+1][j] (choosing the chord with one end at vertex i (case5)) and store the larger one into M[i][j].
  Then, we can get the numbers of maximum non-overlapped chords in M[0][2n-1].

  In order to get the corresponding chords, we use 2d array C[i][j] and a recursive function f(start,end) to get chords under different scenarios..
  If M[i][j] is in case1, then C[i][j] stores it and f(i,j) prints chord with vertex i and calls f(i+1,j).
  If it's in case2, then C[i][j] records it and f(i,j) calls f(i+1,j).
  If it's in case3, then C[i][j] records it and f(i,j) prints chord with vertex i and calls f(i+2,j).
  If it's in case4, then C[i][j] records it and f(i,j) prints chord with vertex i and calls f(i+1,j-2).
  If it's in case5, then C[i][j] records it and f(i,j) prints chord with vertex i, calls f(i+1, (another vertex of chord)-1), and calls f((another vertex of chord)+1,j) respectively.
  This way, we can have all the desired chords printed with first endpoint sorted in increasing order.

This is top-down method and it’s more efficient than buttom-up method since buttom-up method needs to fill every entries in 2d array M while top-down doesn’t.

- **Time complexity analysis**

  For n chords, since it need to initialize 2d array with size 2n*2n in advance, it time complexity is o(n^2), and every subproblem of M(0,2n-1) will be computed for at most once, so the time complexity is O(n^2). Hence we can know that the time complexity of the algorithm is $\Theta(n^2)$.