Algorithm PA2 Report

(I) Data Structures

- struct DpMatrix
  This is a data structure that stores the mps solution
  To reduce memory overhead, I noticed that the maximum size of chords is 90000, which only needs 17bits to store the solution to MPS. Therefore, I use a unsigned short(0~65565) triangular matrix - lsb_16 and a bool triangular matrix msb to represent a number from 0~90000, while the value stored in msb is the most significant bit of the solution. Such method will take 3 bytes for each entry, reducing ¼ space compared to simply using unsigned int.

  e.g. if we want the mps solution from point i to j (i<j)
  dpMatrix.get(i,j) will return a unsigned integer {msb[i][j], lsb_16[i][j]}

  In my implementation, I use dynamic allocated 2D array to implement DpMatrix. To further reduce memory overhead, using vector<vector<bool>> to represent msb is one possible way, since in bool** each bool will take one byte (8bits), while vector<bool> will perform space optimization that will pack the bool entries together, such that each bool takes only 1 bit. Such method will take 2.25 bytes on average for each entry.

- Chord information
  I use a chord[2*chord_number] to implement a look-up table for chords' information. Given a chord (i, j), simply set chord[i] = j and chord[j] = i.

(II) Discussion

- Top-down v.s. Bottom-up approach
  In dynamic programming, if we don't need to solve every sub-problem, top-down will be much more efficient then bottom-up approach, while the MPS problem is the case.