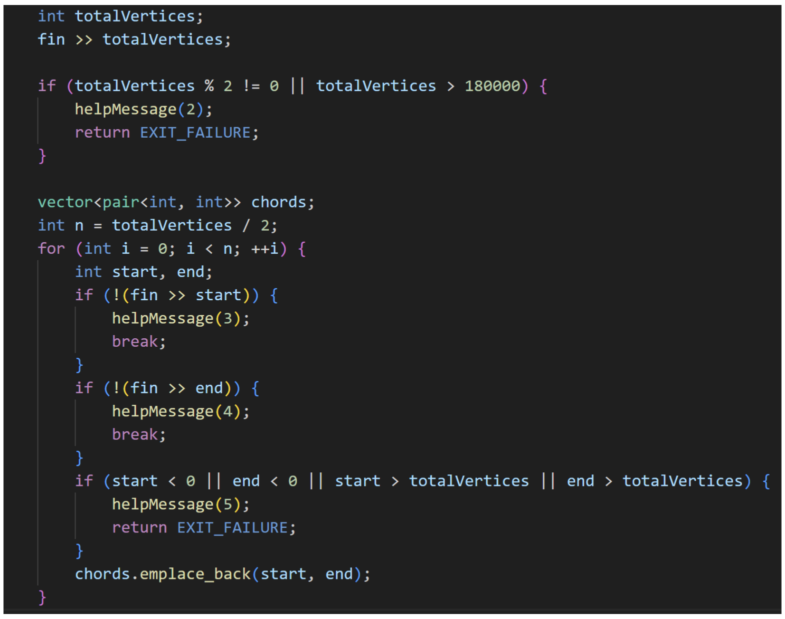
Algorithm PA2 Report

侯奕安 b11202014

April 26, 2025

1 Input

In main.cpp, I stored the input chords in `vector<pair<int, int>>` chords by applying `chords.emplace_back(start, end)`. If an invalid format occurs, `void helpMessage(int errorCode)` will be called, and the corresponding error message will be output according to `errorCode`. After storing input chords, the function `solver()` in class `MPSolver` will be called to calculate the desired solution.



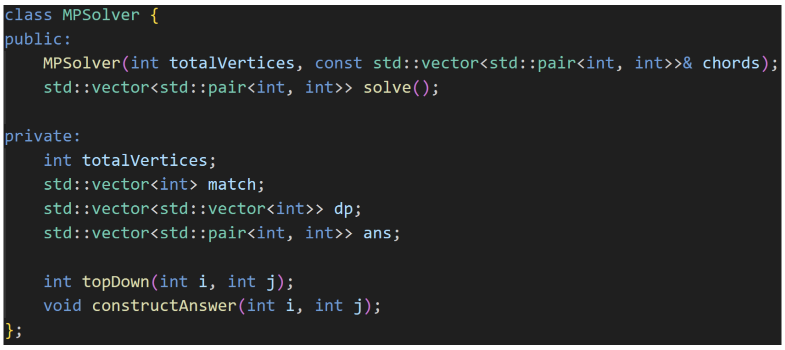
```
int totalVertices;
fin >> totalVertices;

if (totalVertices % 2 != 0 || totalVertices > 180000) {
    helpMessage(2);
    return EXIT_FAILURE;
}

vector<pair<int, int>> chords;
int n = totalVertices / 2;
for (int i = 0; i < n; ++i) {
    int start, end;
    if (!(fin >> start)) {
        helpMessage(3);
        break;
    }
    if (!(fin >> end)) {
        helpMessage(4);
        break;
    }
    if (start < 0 || end < 0 || start > totalVertices || end > totalVertices) {
        helpMessage(5);
        return EXIT_FAILURE;
    }
    chords.emplace_back(start, end);
}
```

Figure 1: handling input

2 Fucntions and Algorithms

2.1 Class Introduction

I used a class to solve the problem:



```cpp
class MPSolver {
public:
    MPSolver(int totalVertices, const std::vector<std::pair<int, int>>& chords);
    std::vector<std::pair<int, int>> solve();

private:
    int totalVertices;
    std::vector<int> match;
    std::vector<std::vector<int>> dp;
    std::vector<std::pair<int, int>> ans;

    int topDown(int i, int j);
    void constructAnswer(int i, int j);
};
```

Figure 2: functions and variables in the class

1. `vector<pair<int, int>> solve()`: Calls `topDown` and `constructAnswer` to calculate the answer.
2. `vector<int, int> match`: Stores `match[chords[i].fisrt] = chords[i].second` and vice versa, achieving $O(1)$ search complexity.
3. `vector<vector<int>> dp`: Triangular array that stores the dp value.
4. `vector<pair<int, int>> ans`: Stores endpoints of the selected chords.
5. `int topDown(int i, int j)`: Uses a top-down method to apply dynamic programming to solve MPS on the given invertal (i, j).
6. `void constructAnswer(int i, int j)`: Constructs a solution based on the dp array.

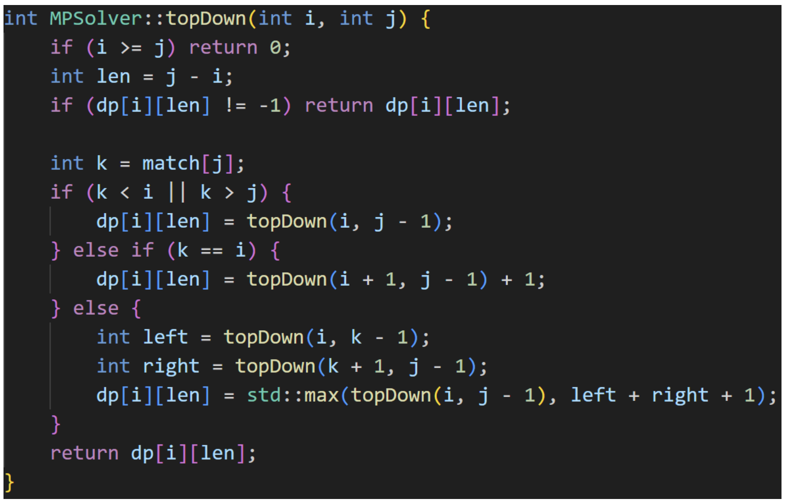
2.2 Algorithm

The algorithm I implemented is the same as the one mentioned in HW2. Given a set C of N chords of a circle, consider chord ij, there are three cases of recursion of solution $M(i, j)$ depending on the chord itself:

1. chord $kj \in C$, $k \notin [i, j]$: $M(i, j) = M(i, j-1)$
2. chord $kj \in C$, $k \in [i, j]$: $M(i, j) = max\{M(i, j-1),\ M(k+1, j-1) + M(i, k-1) + 1\}$

3. chord $ij \in C$: $M(i, j) = 1 + M(i + 1, j - 1)$

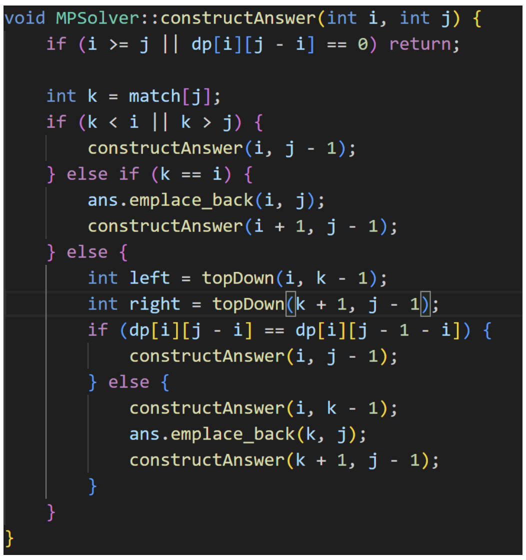
Hence, the top-down dp is shown as the following:



```
int MPSolver::topDown(int i, int j) {
    if (i >= j) return 0;
    int len = j - i;
    if (dp[i][len] != -1) return dp[i][len];

    int k = match[j];
    if (k < i || k > j) {
        dp[i][len] = topDown(i, j - 1);
    } else if (k == i) {
        dp[i][len] = topDown(i + 1, j - 1) + 1;
    } else {
        int left = topDown(i, k - 1);
        int right = topDown(k + 1, j - 1);
        dp[i][len] = std::max(topDown(i, j - 1), left + right + 1);
    }
    return dp[i][len];
}
```

Figure 3: Algorithm of Top-Down DP

As for constructing the answer, `constructAnswer(int i, int j)` will emplace the selected pair of endpoints at the end of `ans` by checking contents in `dp`, and the logic is similar to `topDown(int i, int j)`. Notice that the base case is whenever $i \geq j$ or `dp[i][j - i] = 0`, and the function will return.

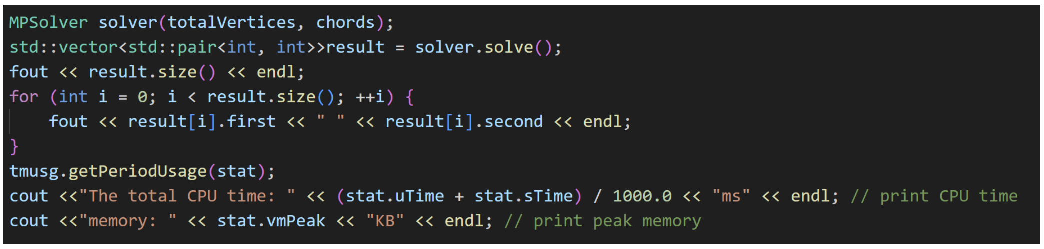


```cpp
void MPSolver::constructAnswer(int i, int j) {
    if (i >= j || dp[i][j - i] == 0) return;

    int k = match[j];
    if (k < i || k > j) {
        constructAnswer(i, j - 1);
    } else if (k == i) {
        ans.emplace_back(i, j);
        constructAnswer(i + 1, j - 1);
    } else {
        int left = topDown(i, k - 1);
        int right = topDown(k + 1, j - 1);
        if (dp[i][j - i] == dp[i][j - 1 - i]) {
            constructAnswer(i, j - 1);
        } else {
            constructAnswer(i, k - 1);
            ans.emplace_back(k, j);
            constructAnswer(k + 1, j - 1);
        }
    }
}
```

Figure 4: Constructing Answer

3 Output

After `ans` is correctly extended, the main funtion will output the sorted answer into the output file, and also display the CPU time and memory usage in terminal.



```cpp
MPSolver solver(totalVertices, chords);
std::vector<std::pair<int, int>>result = solver.solve();
fout << result.size() << endl;
for (int i = 0; i < result.size(); ++i) {
    fout << result[i].first << " " << result[i].second << endl;
}
tmusg.getPeriodUsage(stat);
cout <<"The total CPU time: " << (stat.uTime + stat.sTime) / 1000.0 << "ms" << endl; // print CPU time
cout <<"memory: " << stat.vmPeak << "KB" << endl; // print peak memory
```

Figure 5: Output Handling

4 Comparing CPU Time and Memory Usage

Table 1: CPU Time and Memory Usage of Each Input Size

size	time (ms)	memory (KB)
12	4.645	5896
1000	10.546	7876
10000	490.798	201700
30000	4350.17	1765268
100000	59337.3	19609948
180000	220453	63393048

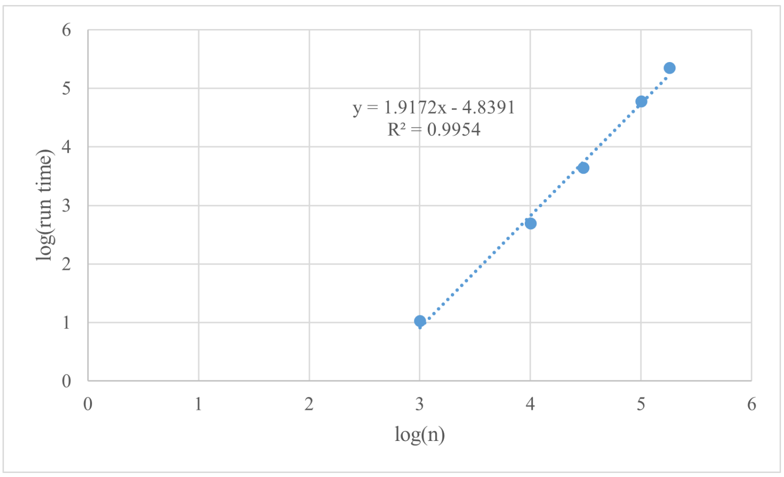


Figure 6: log(time) vs. log(n)

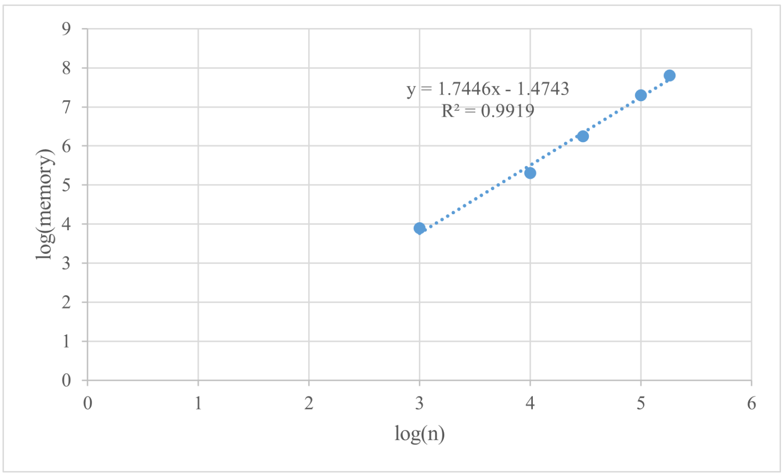
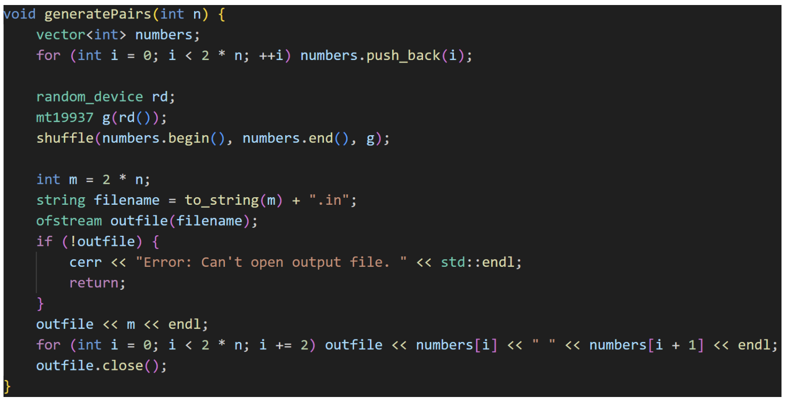


Figure 7: log(memory) vs. log(n)

The above figures show that the relation between time complexity and input size is $T = O(n^2)$, while space complexity and input size follow a polynomial relationship with power less than 2: $M \approx O(n^{1.75}) = O(n^2)$.

5 Some Additional Work

Because there are no public tests with the maximum number of total vertices (180000), I used a .cpp file to randomly generate endpoint pairs of chords to simulate the running time of my program. This file is also used to generate more sample inputs to plot figure 6 and figure 7.



```cpp
void generatePairs(int n) {
    vector<int> numbers;
    for (int i = 0; i < 2 * n; ++i) numbers.push_back(i);

    random_device rd;
    mt19937 g(rd());
    shuffle(numbers.begin(), numbers.end(), g);

    int m = 2 * n;
    string filename = to_string(m) + ".in";
    ofstream outfile(filename);
    if (!outfile) {
        cerr << "Error: Can't open output file. " << std::endl;
        return;
    }
    outfile << m << endl;
    for (int i = 0; i < 2 * n; i += 2) outfile << numbers[i] << " " << numbers[i + 1] << endl;
    outfile.close();
}
```

Figure 8: code for generating inputs

6 Obversation

First of all, if I naively applied the $n \times n$ array for dp, the total memory usage would exceed the 65 GB limit. Hence, I used a triangular array with size $\frac{n\times(n+1)}{2}$ to fit the requirement.

Secondly, during my implementation, the most difficult challenge is to optimize the code and speed up the running time. From the beginning, my program simply recursively called the functions in a bottom-up manner, and it took about 20 minutes to finish the largest test case, which is devastatingly slow. Therefore, I've tried several approaches:

1. Wrap Variables into a Class: I wanted to avoid functions sending references to each other whenever they were called, so I create the `MPSolver` class, wraping `dp`, `ans` into it. However, this approach had limited improvement. For instance, `100000.in` was only 5 to 10 seconds faster.

2. Change to Top-Down Manner: This is the method with best improvement on running time. Top-down method only traverse the essential subproblems, so the irrelevant ones won't be touched, saving a tremendous amount of time. For instance, `100000.in` took more than 5 minutes for bottom-up method, but only took about 1 minute for top-down method.

7 Reminder

Occasionally, the reported memory usage may appear abnormally high (e.g. 21 TB). This is due to an unknown glitch and does not reflect actual memory consumption. This issue also occurs on my personal laptop, which obviously does not have such memory capacity.
If this happens, try the following steps to resolve it:

```
    make clean
    make
```

Other possible solutions:

1. Reconnecting to the Server
2. Delete and Clone the Repo

(To TA: PLEASE DON'T DEDUCT POINTS if you encounter this QAQ This memory issue is random and difficult to reproduce consistently. It may disappear by simply recompiling or re-uploading a file. I've investigated this extensively but haven't found a consistent pattern. Since this problem doesn't affect the function of the program, I kindly ask that this anomaly not be counted against my score.)