LAB 8

HOMEWORK

Task 1: Basic Signature Matching

```
#include <iostream>
#include <vector>
#include <string>

using namespace std;

bool canMatchFile(const string& file, const vector<string>& signatures)
{
    int n = file.length();
    vector<bool> dp(n + 1, false);
    dp[0] = true; // Empty string is always matchable

    for (int i = 1; i <= n; ++i) {
        for (const string& sig : signatures) {
            int len = sig.length();
```

```
            if (i >= len && dp[i - len]) {
                if (file.substr(i - len, len) == sig) {
                    dp[i] = true;
                    break; // No need to check other signatures
                }
            }
        }
    }

    return dp[n];
}

int main() {
    string file = "abcdabcd";
    vector<string> signatures = {"ab", "cd", "abcd"};

    if (canMatchFile(file, signatures)) {
        cout << "File can be matched with the signatures." << endl;
    } else {
        cout << "File cannot be matched with the signatures." << endl;
    }
```

```
    return 0;
}
```

Task 2: Report the Sequence of Signatures Used

```
#include <iostream>
#include <vector>
#include <string>
#include <unordered_map>
#include <stack>

using namespace std;

bool canMatchFile(const string& file, const vector<string>&
signatures, vector<string>& result) {
    int n = file.length();
    vector<bool> dp(n + 1, false);
    vector<int> parent(n + 1, -1);        // Stores where we came
from
```

```
    vector<int> sigIndex(n + 1, -1);      // Stores which signature
was used
    dp[0] = true;

    for (int i = 1; i <= n; ++i) {
        for (int j = 0; j < signatures.size(); ++j) {
            const string& sig = signatures[j];
            int len = sig.length();
            if (i >= len && dp[i - len]) {
                if (file.substr(i - len, len) == sig) {
                    dp[i] = true;
                    parent[i] = i - len;
                    sigIndex[i] = j;
                    break;
                }
            }
        }
    }

    if (!dp[n]) return false;
```

```
    // Backtrack to get the sequence of signatures used
    int index = n;
    stack<string> sequence;

    while (index > 0) {
        int sigIdx = sigIndex[index];
        if (sigIdx == -1) break;
        sequence.push(signatures[sigIdx]);
        index = parent[index];
    }

    // Store result in correct order
    while (!sequence.empty()) {
        result.push_back(sequence.top());
        sequence.pop();
    }

    return true;
}
```

```
int main() {
    string file = "abcdabcd";
    vector<string> signatures = {"ab", "cd", "abcd"};
    vector<string> result;

    if (canMatchFile(file, signatures, result)) {
        cout << "File can be matched with the signatures." << endl;
        cout << "Sequence of signatures used:" << endl;
        for (const string& sig : result) {
            cout << sig << " ";
        }
        cout << endl;
    } else {
        cout << "File cannot be matched with the signatures." <<
endl;
    }

    return 0;
}
```

Task 3: Wildcard Support

```
#include <iostream>
#include <vector>
#include <string>
#include <unordered_map>
#include <stack>

using namespace std;

// Function to match a signature with the wildcard '?' support
bool matchesWithWildcard(const string& file, int start, const string& sig) {
    int len = sig.length();
    if (start + len > file.length()) return false;  // If signature goes beyond file bounds

    for (int i = 0; i < len; ++i) {
        if (sig[i] != '?' && sig[i] != file[start + i]) {
            return false;  // Character mismatch or non-wildcard mismatch
        }
```

```
    }
    return true;
}

bool canMatchFile(const string& file, const vector<string>& signatures,
vector<string>& result) {
    int n = file.length();
    vector<bool> dp(n + 1, false);
    vector<int> parent(n + 1, -1);        // Stores where we came from
    vector<int> sigIndex(n + 1, -1);      // Stores which signature was
used
    dp[0] = true;

    for (int i = 1; i <= n; ++i) {
        for (int j = 0; j < signatures.size(); ++j) {
            const string& sig = signatures[j];
            int len = sig.length();
            if (i >= len && dp[i - len]) {
                if (matchesWithWildcard(file, i - len, sig)) {
                    dp[i] = true;
                    parent[i] = i - len;
                    sigIndex[i] = j;
```

```
                break;  // No need to check other signatures
            }
        }
    }
}

if (!dp[n]) return false;

// Backtrack to get the sequence of signatures used
int index = n;
stack<string> sequence;

while (index > 0) {
    int sigIdx = sigIndex[index];
    if (sigIdx == -1) break;
    sequence.push(signatures[sigIdx]);
    index = parent[index];
}

// Store result in correct order
while (!sequence.empty()) {
    result.push_back(sequence.top());
```

```
        sequence.pop();
    }

    return true;
}

int main() {
    string file = "abcdabcd";
    vector<string> signatures = {"ab?", "cd", "abcd"};

    vector<string> result;

    if (canMatchFile(file, signatures, result)) {
        cout << "File can be matched with the signatures." << endl;
        cout << "Sequence of signatures used:" << endl;
        for (const string& sig : result) {
            cout << sig << " ";
        }
        cout << endl;
    } else {
        cout << "File cannot be matched with the signatures." << endl;
    }
```

```
    return 0;
}
```

Task 4: Performance Test

#include #include #include #include #include #include #include #include

using namespace std; using namespace chrono;

matchesWithWildcard(const string& file, int start, const string& sig) { int len = sig.length(); if (start + len > file.length()) return false; // If signature goes beyond file bounds

```
for (int i = 0; i < len; ++i) {
    if (sig[i] != '?' && sig[i] != file[start + i]) {
        return false;  // Character mismatch or non-wildcard
mismatch
    }
}
return true;
```

}

bool canMatchFile(const string& file, const vector& signatures, vector& result) { int n = file.length(); vector dp(n + 1, false); vector parent(n + 1,

-1); // Stores where we came from vector sigIndex(n + 1, -1); // Stores which signature was used dp[0] = true;

```
for (int i = 1; i <= n; ++i) {
    for (int j = 0; j < signatures.size(); ++j) {
        const string& sig = signatures[j];
        int len = sig.length();
        if (i >= len && dp[i - len]) {
            if (matchesWithWildcard(file, i - len, sig)) {
                dp[i] = true;
                parent[i] = i - len;
                sigIndex[i] = j;
                break;  // No need to check other signatures
            }
        }
    }
}

if (!dp[n]) return false;
int index = n;
stack<string> sequence;

while (index > 0) {
    int sigIdx = sigIndex[index];
    if (sigIdx == -1) break;
    sequence.push(signatures[sigIdx]);
    index = parent[index];
}

while (!sequence.empty()) {
    result.push_back(sequence.top());
    sequence.pop();
}
```

```
return true;


}
```

generateRandomString(int length

{ const string chars = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"; string result; result.reserve(length); random_device rd; mt19937 gen(rd()); uniform_int_distribution<> dis(0, chars.size() - 1);

```
for (int i = 0; i < length; ++i) {
    result += chars[dis(gen)];
}
return result;


}
```

generateRandomSignatures(int numSignatures, int maxLength)

{ vector signatures;

for (int i = 0; i < numSignatures; ++i) { int length = rand() % maxLength + 1;

signatures.push_back(generateRandomString(length)); }

return signatures; }

int main() { srand(time(0));

```
string file = generateRandomString(10000);
```

```
vector<string> signatures = generateRandomSignatures(500,
5);
auto start = high_resolution_clock::now();

vector<string> result;
bool match = canMatchFile(file, signatures, result);


auto end = high_resolution_clock::now();
auto duration = duration_cast<milliseconds>(end - start);

if (match) {
    cout << "File can be matched with the signatures." <<
endl;
    cout << "Sequence of signatures used: ";
    for (const string& sig : result) {
        cout << sig << " ";
    }
    cout << endl;
} else {
    cout << "File cannot be matched with the signatures." <<
endl;
}


cout << "Performance Test Completed in: " <<
duration.count() << " milliseconds." << endl;

return 0;


}
```