CS208 Lab2 Practice: Stable Matching

12312110 李轩然

DDL: Mar.16

Task 1: Gale-Shapley Algorithm Implementation

C++ Code

```cpp
#include <iostream>
#include <vector>
#include <unordered_map>
#include <queue>
using namespace std;

void stable_matching(int n, vector<string>& men, vector<string>& women,
                     unordered_map<string, vector<string>>& men_pref,
                     unordered_map<string, vector<string>>& women_pref) {
    unordered_map<string, int> men_map, women_map;
    for (int i = 0; i < n; i++)
    {
        men_map[men[i]] = i;
        women_map[women[i]] = i;
    }

    vector<vector<int>> men_pref_idx(n, vector<int>(n));
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            men_pref_idx[i][j] = women_map[men_pref[men[i]][j]];

    vector<vector<int>> women_rank(n, vector<int>(n));
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            women_rank[i][women_map[women_pref[women[i]][j]]] = j;

    vector<int> men_to_women(n, -1);
    vector<int> women_to_men(n, -1);

    queue<int> men_free;
    for (int i = 0; i < n; i++)
        men_free.push(i);

    while (!men_free.empty())
```

```
    {
        int man = men_free.front();
        men_free.pop();

        for (int woman_idx : men_pref_idx[man])
        {
            if (women_to_men[woman_idx] == -1) {
                men_to_women[man] = woman_idx;
                women_to_men[woman_idx] = man;
                break;
            } else {
                int current_man = women_to_men[woman_idx];
                if (women_rank[woman_idx][man] < women_rank[woman_idx]
[current_man]) {
                    men_to_women[current_man] = -1;
                    men_free.push(current_man);
                    men_to_women[man] = woman_idx;
                    women_to_men[woman_idx] = man;
                    break;
                }
            }
        }
    }

    for (int i = 0; i < n; i++)
        cout << men[i] << " " << women[men_to_women[i]] << '\n';
}

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n;
    cin >> n;

    vector<string> men(n), women(n);
    for (int i = 0; i < n; i++)
        cin >> men[i];
    for (int i = 0; i < n; i++)
        cin >> women[i];

    unordered_map<string, vector<string>> men_pref;
    for (int i = 0; i < n; i++)
    {
        men_pref[men[i]] = vector<string>(n);
        for (int j = 0; j < n; j++)
            cin >> men_pref[men[i]][j];
    }
```

```
    unordered_map<string, vector<string>> women_pref;
    for (int i = 0; i < n; i++)
    {
        women_pref[women[i]] = vector<string>(n);
        for (int j = 0; j < n; j++)
            cin >> women_pref[women[i]][j];
    }

    stable_matching(n, men, women, men_pref, women_pref);
    return 0;
}
```

Notion

We use `ios::sync_with_stdio(false)` and `cin.tie(nullptr)` to turn off synchronization of the C++ standard library with the C standard library to speed up input and output. In this way, when $N = 1000$, we can accumulate the runtime from around $1.2s$ to around $0.8s$!

Task 2: Construct Test Cases

Requirement

Design **at least 10 test cases** covering the following categories:

Category	Purpose
Basic Functionality	Validate correctness for simple, non-conflicting preferences.
Edge Cases	Test minimum input size ((N=1)), extreme preferences, and edge values.
Multiple Stable Solutions	Scenarios with multiple valid stable matchings.
Conflict Scenarios	Cyclic preferences (e.g., (B1→G1→B2→G2→B1)).
Performance & Scalability	Ensure efficiency for $N \geq 1000$.
Random-Scale & Distribution Tests	Validate robustness with programmatically generated inputs of varying sizes.

Construction

1. Basic Functionality

Input1: simple, non-conflicting

```
5
Victor Wyatt Xavier Yancey Zeus
Amy Bertha Clare Diane Erika
Bertha Amy Diane Erika Clare
Diane Bertha Amy Clare Erika
Bertha Erika Clare Diane Amy
Amy Diane Clare Bertha Erika
Bertha Diane Amy Erika Clare
Zeus Victor Wyatt Yancey Xavier
Xavier Wyatt Yancey Victor Zeus
Wyatt Xavier Yancey Zeus Victor
Victor Zeus Yancey Xavier Wyatt
Yancey Wyatt Zeus Xavier Victor
```

Output1:

```
Victor Amy
Wyatt Clare
Xavier Bertha
Yancey Erika
Zeus Diane
```

2. Edge Cases

Input2: minimum

```
1
man
woman
woman
man
```

Output2:

```
man woman
```

Input3: extreme preferences

```
4
m1 m2 m3 m4
w1 w2 w3 w4
w1 w4 w3 w2
w1 w3 w2 w4
w1 w3 w4 w2
w1 w2 w4 w3
m2 m3 m4 m1
m2 m4 m1 m3
m2 m1 m3 m4
m2 m4 m3 m1
```

Output3:

```
m1 w4
m2 w1
m3 w3
m4 w2
```

Explanation: All men prefer the same woman, all women prefer the same man.

Input4: edge values

```
4
m1 m2 m3 m4
w1 w2 w3 w4
w1 w4 w3 w2
w1 w3 w2 w4
w1 w3 w4 w2
w1 w2 w4 w3
m1 m3 m4 m2
m3 m4 m1 m2
m4 m1 m3 m2
m1 m4 m3 m2
```

Output4:

```
m1 w4
m2 w1
m3 w3
m4 w2
```

3. Multiple Stable Solutions

Input5: multiple valid stable matchings

```
3
m1 m2 m3
w1 w2 w3
w1 w2 w3
w2 w1 w3
w1 w3 w2
m1 m2 m3
m2 m1 m3
m1 m3 m2
```

Output5:

```
m1 w1
m2 w2
m3 w3
```

Explanation:

```
m1 w1
m2 w3
m3 w2
```

and

```
m1 w2
m2 w1
m3 w3
```

are also the valid matchings, and our output is stable.

4. Conflict Scenarios

Input6:

```
3
m1 m2 m3
w1 w2 w3
w1 w3 w2
w2 w3 w1
w3 w1 w2
m2 m1 m3
m3 m2 m1
m1 m3 m2
```

Output6:

```
m1 w1
m2 w2
m3 w3
```

Explanation: In this case, there exists cyclic preferences:
$m1- > w1- > m2- > w2- > m3- > w3- > m1.$

5. Performance & Scalability

Input7-10: Here is the code which can output different sizes of input cases, just by changing const N. We can take `N = 100, 500, 1000, 2000` as four examples.

```
#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>
#include <random>
using namespace std;

vector<string> generateNames(const string& prefix, int n) {
    vector<string> names;

    for (int i = 1; i <= n; ++i)
        names.push_back(prefix + to_string(i));

    return names;
}

vector<vector<string>> generatePreferences(const vector<string>& names,
mt19937& rng) {
    vector<vector<string>> preferences;

    for (size_t i = 0; i < names.size(); ++i)
    {
        vector<string> pref = names;
        shuffle(pref.begin(), pref.end(), rng);
        preferences.push_back(pref);
    }

    return preferences;
}

int main()
{
    const int N = 1000; //also can be 100, 500, 2000, 4 cases have been
```

```
tried
    mt19937 rng(random_device{}());

    vector<string> men = generateNames("m", N);
    vector<string> women = generateNames("w", N);

    vector<vector<string>> men_pref = generatePreferences(women, rng);
    vector<vector<string>> women_pref = generatePreferences(men, rng);

    ofstream outFile("input.txt");
    if (!outFile) {
        cerr << "Error: Could not open file input.txt for writing." <<
endl;
        return 1;
    }
    filesystem::path filePath = filesystem::current_path() / "input.txt";
        cout << "Input file will be saved to: " << filePath << endl;

    outFile << N << "\n";
    for (const string& man : men)
        outFile << man << " ";
    outFile << "\n";

    for (const string& woman : women)
        outFile << woman << " ";
    outFile << "\n";

    for (size_t i = 0; i < men.size(); ++i)
    {
        for (const string& woman : men_pref[i]) {
            outFile << woman << " ";
        }

        outFile << "\n";
    }

    for (size_t i = 0; i < women.size(); ++i)
    {
        for (const string& man : women_pref[i]) {
            outFile << man << " ";
        }

        outFile << "\n";
    }

    outFile.close();
    return 0;
}
```

input_1000.txt

Output7-10: We can find the runtime of this code when `N = 1000` is around $0.9s$. Morever, here is the modified code which adds time counting, and input and output files.

```
#include <iostream>
#include <vector>
#include <unordered_map>
#include <queue>
#include <fstream>
#include <chrono>
using namespace std;
using namespace std::chrono;

void stable_matching(int n, vector<string>& men, vector<string>& women,
                     unordered_map<string, vector<string>>& men_pref,
                     unordered_map<string, vector<string>>& women_pref) {
    unordered_map<string, int> men_map, women_map;
    for (int i = 0; i < n; i++)
    {
        men_map[men[i]] = i;
        women_map[women[i]] = i;
    }

    vector<vector<int>> men_pref_idx(n, vector<int>(n));
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            men_pref_idx[i][j] = women_map[men_pref[men[i]][j]];

    vector<vector<int>> women_rank(n, vector<int>(n));
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            women_rank[i][women_map[women_pref[women[i]][j]]] = j;

    vector<int> men_to_women(n, -1);
    vector<int> women_to_men(n, -1);

    queue<int> men_free;
    for (int i = 0; i < n; i++)
        men_free.push(i);

    while (!men_free.empty())
    {
        int man = men_free.front();
        men_free.pop();

        for (int woman_idx : men_pref_idx[man]) {
```

```
            if (women_to_men[woman_idx] == -1) {
                men_to_women[man] = woman_idx;
                women_to_men[woman_idx] = man;
                break;
            } else {
                int current_man = women_to_men[woman_idx];
                if (women_rank[woman_idx][man] < women_rank[woman_idx]
[current_man]) {
                    men_to_women[current_man] = -1;
                    men_free.push(current_man);
                    men_to_women[man] = woman_idx;
                    women_to_men[woman_idx] = man;
                    break;
                }
            }
        }
    }

    ofstream outFile("output.txt");
    if (!outFile) {
        cerr << "Error: Could not open output.txt for writing." << endl;
        return;
    }

    for (int i = 0; i < n; i++)
        outFile << men[i] << " " << women[men_to_women[i]] << '\n';
    outFile.close();
}

int main()
{
    auto start = high_resolution_clock::now();
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    ifstream inFile("input.txt");
    if (!inFile) {
        cerr << "Error: Could not open input.txt for reading." << endl;
        return 1;
    }

    int n;
    inFile >> n;
    vector<string> men(n), women(n);
    for (int i = 0; i < n; i++)
        inFile >> men[i];
    for (int i = 0; i < n; i++)
        inFile >> women[i];

    unordered_map<string, vector<string>> men_pref;
```

```
    for (int i = 0; i < n; i++) {
        men_pref[men[i]] = vector<string>(n);
        for (int j = 0; j < n; j++) {
            inFile >> men_pref[men[i]][j];
        }
    }

    unordered_map<string, vector<string>> women_pref;
    for (int i = 0; i < n; i++) {
        women_pref[women[i]] = vector<string>(n);
        for (int j = 0; j < n; j++) {
            inFile >> women_pref[women[i]][j];
        }
    }
    inFile.close();
    stable_matching(n, men, women, men_pref, women_pref);
    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<milliseconds>(stop - start);
    cout << "Runtime: " << duration.count() << "ms" << endl;
    return 0;
}
```

Output when `N = 1000` : output_1000.txt

6. Random-Scale & Distribution Tests

Just using the same method in (5).

Task 3: Test Your Algorithm

1. Manual Verification

- Just have a look at input5 and output5 in task 2, and we find that if one case has mutiple answers, the output will always be the same, which means **Gale-Shapley Algorithm is stable**.

2. Diff-Based Testing

Brute-Force C++ Code

```
#include <iostream>
```

```cpp
#include <vector>
#include <unordered_map>
#include <algorithm>
using namespace std;

bool isStable(const vector<int>& matching, const vector<vector<int>>&
boyPref, const vector<vector<int>>& girlPrefRank) {
    int N = matching.size();
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; ++j) {
            if (i == j) continue;
            int boy1 = i;
            int girl1 = matching[i];
            int boy2 = j;
            int girl2 = matching[j];

            bool boyPrefersGirl2 = false;
            for (int k = 0; k < N; ++k) {
                if (boyPref[boy1][k] == girl2) {
                    boyPrefersGirl2 = true;
                    break;
                }
                if (boyPref[boy1][k] == girl1) {
                    break;
                }
            }

            bool girlPrefersBoy1 = girlPrefRank[girl2][boy1] <
girlPrefRank[girl2][boy2];

            if (boyPrefersGirl2 && girlPrefersBoy1) {
                return false;
            }
        }
    }
    return true;
}

int main()
{
    int N;
    cin >> N;

    vector<string> boys(N);
    for (int i = 0; i < N; ++i) {
        cin >> boys[i];
    }

    vector<string> girls(N);
    for (int i = 0; i < N; ++i) {
```

```
        cin >> girls[i];
    }

    unordered_map<string, int> boyIndex;
    for (int i = 0; i < N; ++i) {
        boyIndex[boys[i]] = i;
    }

    unordered_map<string, int> girlIndex;
    for (int i = 0; i < N; ++i) {
        girlIndex[girls[i]] = i;
    }

    vector<vector<int>> boyPref(N, vector<int>(N));
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; ++j) {
            string girl;
            cin >> girl;
            boyPref[i][j] = girlIndex[girl];
        }
    }

    vector<vector<int>> girlPref(N, vector<int>(N));
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; ++j) {
            string boy;
            cin >> boy;
            girlPref[i][j] = boyIndex[boy];
        }
    }

    vector<vector<int>> girlPrefRank(N, vector<int>(N));
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; ++j) {
            girlPrefRank[i][girlPref[i][j]] = j;
        }
    }

    vector<int> matching(N);
    for (int i = 0; i < N; ++i) {
        matching[i] = i;
    }

    do {
        if (isStable(matching, boyPref, girlPrefRank)) {
            for (int i = 0; i < N; ++i) {
                cout << boys[i] << " " << girls[matching[i]] << endl;
            }
            break;
        }
```

```
    } while (next_permutation(matching.begin(), matching.end()));

    return 0;
}
```

- By using **Brute-Force Program**, we automatically find that the output of **Gale-Shapley-based Program** is alway valid.

Referred Questions:

Q1: *Does diff-based testing remain feasible for a little larger inputs (e.g., N > 10 or N > 20)?*
Q2: *How to test when N exceeds a threshold (e.g., N > 10 or N > 20)?*

- **Q1:** Usable but low efficiency. The time complexity of **Brute-Force Program** is about $O(N!)$, thus the test is not feasible when `N > 20` or so.
- **Q2:** Direct stability verification. Just iterating through all man-woman pairs to check for blocking pairs, and the time complexity will be $O(N^2)$.

3. Performance Testing

- The time complexity of this program is $O(N^2)$.
- According to the input 7-10 in task 2, when `N = 1000`, the output is: `Runtime: 965ms`, which is less than 1 second.
- Morever, we can test that when `N = 100`, the output is: `Runtime: 5ms`; when `N = 500`, the output is: `Runtime: 139ms`; when `N = 2000`, the output is: `Runtime: 2282ms`.

Task 4: Fairness Discussion

Description

Does the result differ if **boys propose first vs. girls propose first**? Explain why?

Analysis

NO.
Because when changing boys and girls, we can just treat it as we exchange boys'

names and girls' names, which does not influence our result.

Partly input and output files

- **N = 100**
  input_100.txt
  output_100.txt
- **N = 500**
  input_500.txt
  output_500.txt
- **N = 1000**
  input_1000.txt
  output_1000.txt
- **N = 2000**
  input_2000.txt
  output_2000.txt