

# 东北大学计算机科学与工程学院

## 数据结构课程设计报告

题目 男女运动员最佳组合

课题组长 时智贤

课题组成员 王韵然 杨宗昊

专业名称 计算机科学与技术

班级 计 2401 2402 2403

指导教师 李发明

2025 年 12 月

# 课程设计任务书

题目:

男女运动员最佳组合

问题描述:

设有  $N$  个男羽毛球运动员和  $N$  个女羽毛球运动员, 现组成  $N$  对男女混合最佳组合。每个男运动员对每个女运动员都有一个满意度排序, 用矩阵  $mf[0:n-1][0:n-1]$  表示。 $mf[i][j]$  表示第  $i$  个男运动员对第  $j$  个女运动员的满意度, 满意度值越高, 满意程度越高。同理, 每个女运动员对每个男运动员也有一个满意度排序, 用矩阵  $fm[0:n-1][0:n-1]$  表示。男女运动员之间的一个完全匹配称为一个组合。

设计要求:

设计对于给定的满意度, 求最佳组合的程序, 使得满意度总和达到最大。

- (1) 采用 STL 的一维向量类构造构造二维向量矩阵。
- (2) 应用基本运算, 设计算法求解。

指导教师签字:

年 月 日

# 课题程序设计分工

学号	姓名	程序设计函数原型、类	功能说明
20245131	时智贤	<pre> MatchingResult* bruteSearch(const SatisfactionMatrix&amp; matrix); AlgorithmComparison* verify_hungarian_result(const SatisfactionMatrix&amp; matrix, const MatchingResult&amp; hungarian_result); bool loadMatrixFromFile(const string&amp; filename, SatisfactionMatrix&amp; matrix); bool saveMatrixToFile(const string&amp; filename, const SatisfactionMatrix&amp; matrix); void activate(GtkApplication* app, gpointer user_data); void on_generate_clicked(GtkWidget* widget, gpointer data); void on_solve_clicked(GtkWidget* widget, gpointer data); void on_load_clicked(GtkWidget* widget, gpointer data); void on_save_clicked(GtkWidget* widget, gpointer data); void display_matrix_in_grid(GtkWidget* grid, const vector&lt;vector&lt;int&gt;&gt;&amp; matrix, int start_row, const string&amp; title); void update_result_display(GtkWidget* textview, const MatchingResult&amp; result); void </pre>	<p>暴力搜索</p> <p>验证匈牙利算法的结果</p> <p>读写文件</p> <p>GTK 应用程序激活</p> <p>"生成随机矩阵"按钮</p> <p>"求解最佳组合"按钮</p> <p>"加载文件"按钮</p> <p>"保存文件"按钮</p> <p>在网格中显示矩阵</p> <p>更新结果显示</p>
20245190	王韵然	<pre> class SatisfactionMatrix { public:     SatisfactionMatrix(int size = 0);     void resize(int size);     void randomGenerate(); };  class MatchingResult { public:     MatchingResult(int size = 0); }; </pre>	<p>矩阵数据结构</p> <p>调整矩阵大小</p> <p>随机生成矩阵</p> <p>配对结果</p>
20245248	杨宗昊	<pre> class HungarianAlgorithm { private:     bool dfs(int u); public:     HungarianAlgorithm();     MatchingResult solve(const SatisfactionMatrix&amp; matrix); }; </pre>	<p>匈牙利算法</p> <p>DFS 深度优先搜索, 为某一位男队员匹配, 返回成功与否</p> <p>计算配对结果</p>

# 课题报告分工

章节	内容	完成人
1 课题概述	1.1 课题任务 1.2 课题原理 1.3 相关知识	时智贤
2 需求分析	2.1 课题调研 2.2 用户需求分析	王韵然
3 方案设计	3.1 总体功能设计 3.2 数据结构设计 3.3 函数原型设计 3.4 输入输出设计 3.5 主算法设计 3.6 用户界面设计	杨宗昊
4 方案实现	4.1 开发环境与工具 4.2 程序设计关键技术 4.3 个人设计实现（按组员分工） 4.3.1 4.3.2 4.3.3	王韵然 杨宗昊 时智贤
5 测试与调试	5.1 个人测试（按组员分工） 5.1.1 5.1.2 5.1.3 5.2 组装与系统测试 5.3 系统运行	时智贤 王韵然 杨宗昊
6 课题总结	6.1 课题评价 6.2 团队协作 6.3 下一步工作 6.4 个人设计心得（按组员分工） 6.4.1 6.4.2 6.4.3	时智贤 王韵然 杨宗昊

# 目录

- 1 课题概述
  - 1.1 课题任务
  - 1.2 课题原理
  - 1.3 相关知识
- 2 需求分析
  - 2.1 课题调研
  - 2.2 用户需求分析
- 3 方案设计
  - 3.1 总体功能设计
  - 3.2 数据结构设计
  - 3.3 函数原型设计
  - 3.4 主算法设计
  - 3.5 用户界面设计
  - 3.6 输入输出设计
- 4 方案实现
  - 4.1 开发环境与工具
  - 4.2 程序设计关键技术
  - 4.3 个人设计实现（按组员分工）
- 5 测试与调试
  - 5.1 个人测试（按组员分工）
  - 5.2 组装与系统测试
  - 5.3 系统运行
- 6 课题总结
  - 6.1 课题评价
  - 6.2 团队协作

6.3 个人设计小结（按组员分工）

7 课题设计文档

B-1 课程设计报告（电子版）

B-2 源程序代码（\*.H, \*.CPP）

C.1 运行环境说明

## 1 课题概述

### 1.1 课题任务

设计一个用于求解男女羽毛球运动员最佳配对的系统，根据双方满意度矩阵（男对女、女对男），采用匈牙利算法求解最优匹配，使得双方总满意度最大化。系统需支持矩阵数据输入、算法求解、结果验证及 GUI 展示。

### 1.2 课题原理

**满意度矩阵：**男对女满意度矩阵  $mf$  与女对男满意度矩阵  $fm$ 。

**总满意度矩阵：** $total[i][j] = mf[i][j] + fm[j][i]$ ，表示男  $i$  与女  $j$  配对的总满意度。

**匈牙利算法：**一种求解指派问题的经典算法，通过调整期望值寻找最大权重完美匹配。

**暴力搜索验证：**当  $n \leq 10$  时，通过枚举所有排列验证匈牙利算法的正确性。

### 1.3 相关知识

组合优化与指派问题

匈牙利算法 (Kuhn – Munkres 算法)

STL 容器 (vector、algorithm)

GTK+图形界面开发

文件 I/O 与数据验证

## 2 需求分析

### 2.1 课题调研

现有配对系统多基于单一满意度，本系统考虑双方满意度，更符合实际场景。

教学场景需可视化展示算法过程与结果对比。

系统支持小规模暴力验证，增强算法可信度。

### 2.2 用户需求分析

用户角色	需求描述
教练员	快速生成最佳配对方案，支持数据导入导出
学生	理解匈牙利算法原理，可视化查看匹配过程
开发者	可扩展算法，支持更大规模问题

### 3 方案设计

#### 3.1 总体功能设计

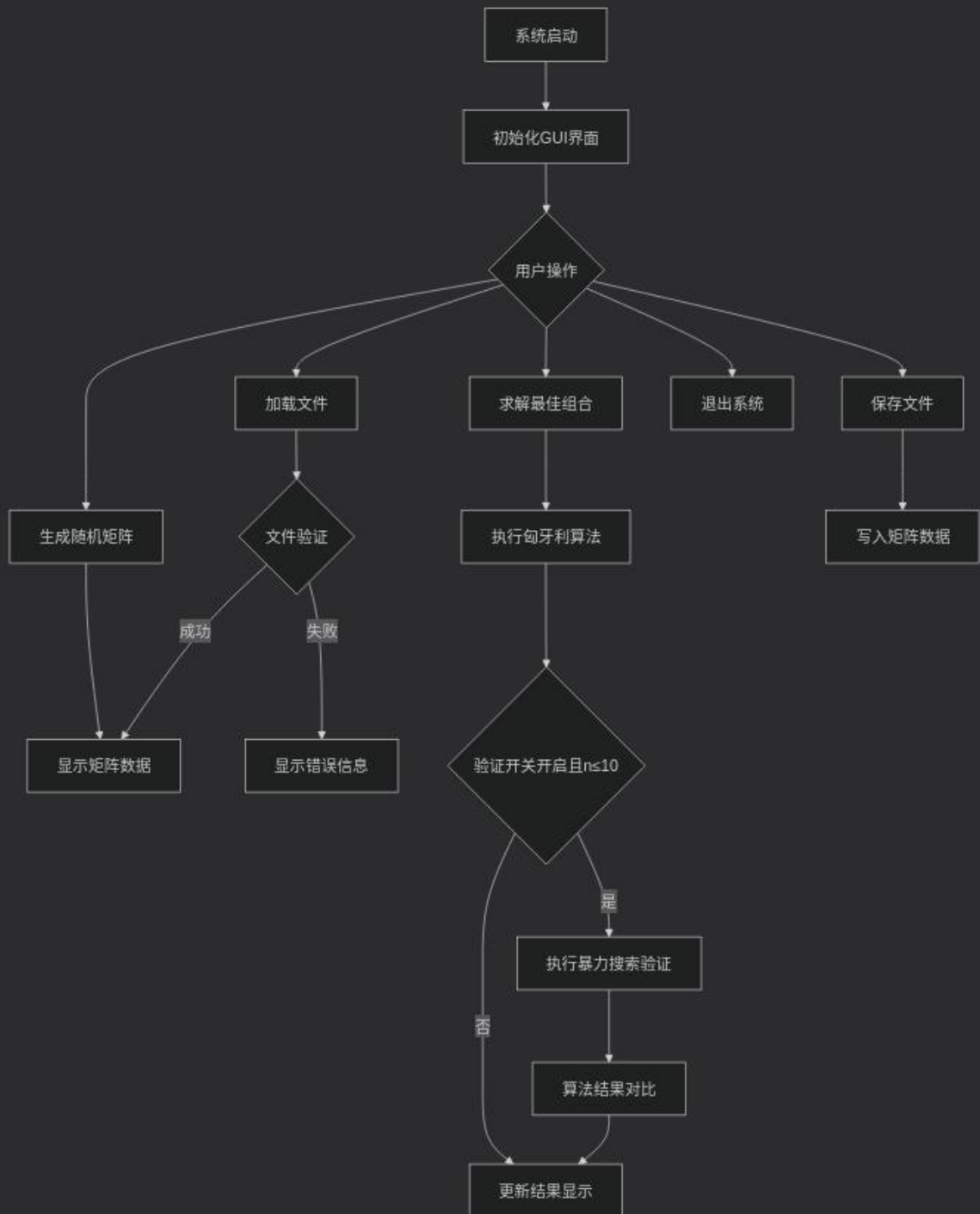
**矩阵管理：**生成、加载、保存满意度矩阵

**算法求解：**匈牙利算法求解最佳配对

**结果验证：**暴力搜索验证 ( $n \leq 10$ )

**GUI 显示：**矩阵可视化、结果展示、对比信息

设计流程图：

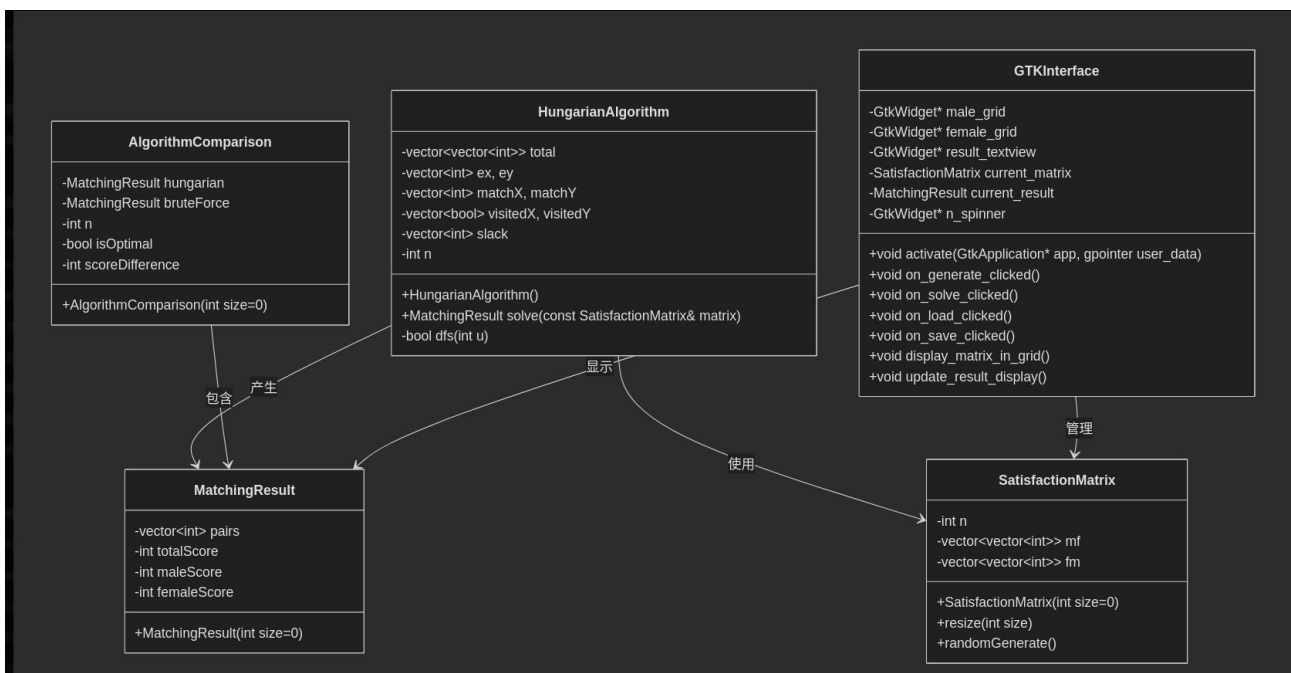


### 3.2 数据结构设计

cpp

```
class SatisfactionMatrix {  
    int n; // 运动员数量  
    vector<vector<int>>> mf; // 男对女满意度矩阵  
    vector<vector<int>>> fm; // 女对男满意度矩阵  
};  
  
class MatchingResult {  
    vector<int> pairs; // 配对结果 pairs[i]=j 表示男 i 配女 j  
    int totalScore; // 总满意度  
    int maleScore; // 男方总满意度  
    int femaleScore; // 女方总满意度  
};
```

类图:



### 3.3 函数原型设计

类别	函数名	功能
矩阵操作	randomGenerate	随机生成满意度矩阵
算法求解	HungarianAlgorithm::solve	匈牙利算法求解
验证功能	bruteSearch	暴力搜索最优解
验证功能	verify_hungarian_result	验证算法结果
文件操作	loadMatrixFromFile/saveMatrixToFile	读写矩阵文件

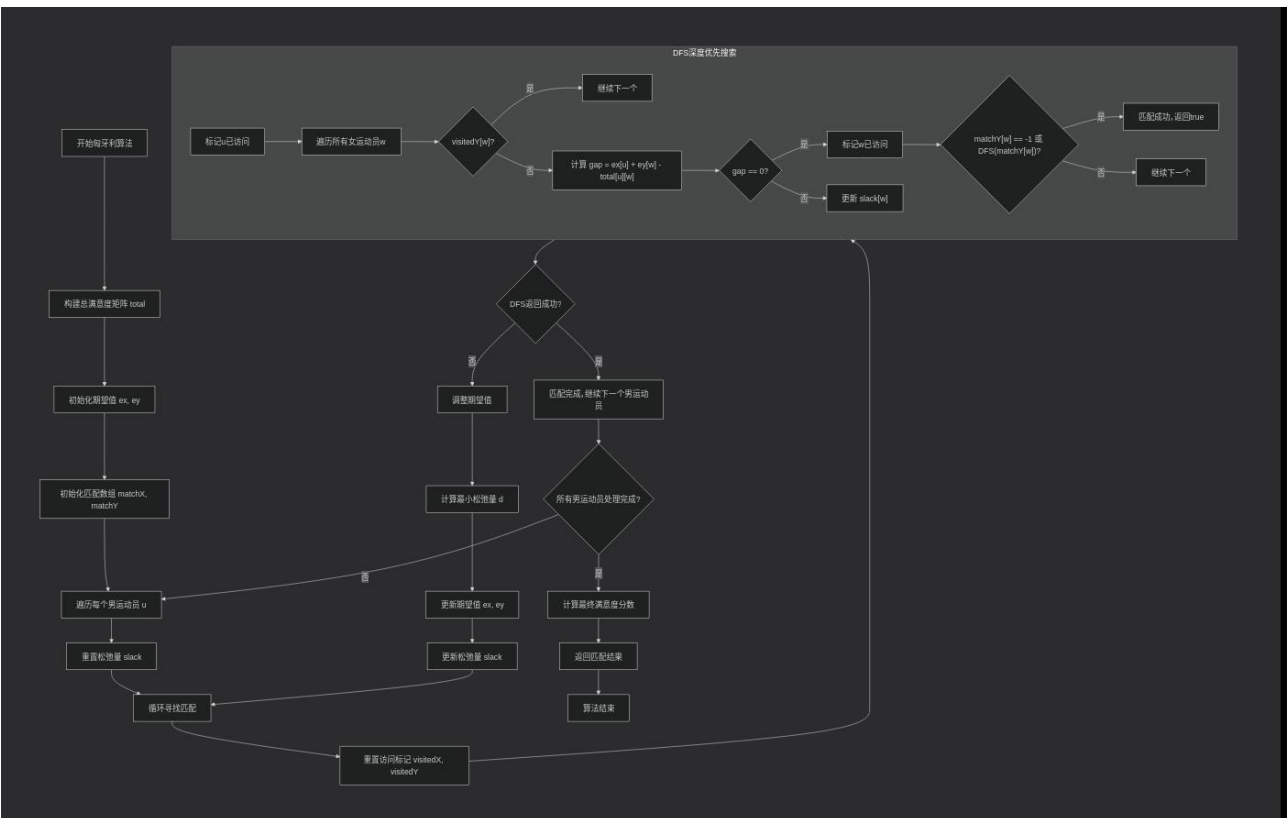
### 3.4 主算法设计

匈牙利算法:

1. 初始化期望值  $ex$ 、 $ey$
2. 为每个男运动员寻找匹配 (DFS 增广)
3. 若匹配失败, 调整期望值继续
4. 重复直至所有男运动员匹配成功

暴力搜索: 枚举所有排列, 计算总满意度, 保留最大值。

算法流程图:





### 3.5 用户界面设计

控制面板：参数设置、功能按钮

矩阵显示区：男对女、女对男矩阵

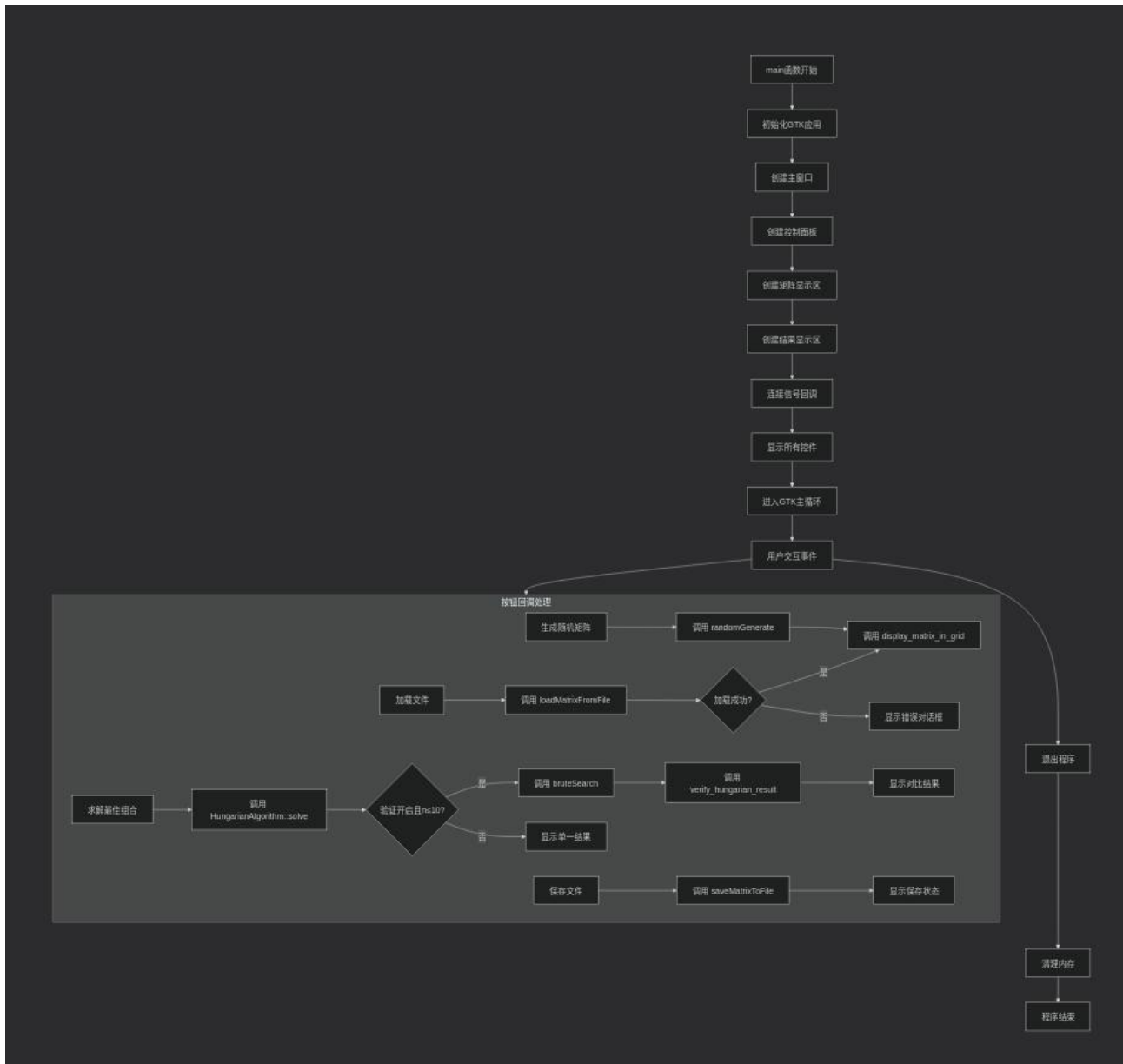
结果显示区：匹配结果、满意度分数、验证信息

### 3.6 输入输出设计

输入：随机生成、文件加载（TXT 格式）

输出：文件保存、GUI 结果展示、控制台日志

整体程序流程图：



## 4 方案实现

### 4.1 开发环境与工具

操作系统: Linux / Windows

编译器: g++ (C++11)

开发工具: VS Code

依赖库: GTK+3.0

### 4.2 程序设计关键技术

STL 二维向量表示矩阵

匈牙利算法 DFS 实现

全排列枚举 (next\_permutation)

GTK+网格布局与文本框显示

### 4.3 个人设计实现 (按组员分工)

#### 4.3.1 王韵然设计实现

负责设计并实现满意度矩阵和配对结果的数据结构。在 SatisfactionMatrix 类中, 使用 `vector<vector<int>>` 存储男对女、女对男两个满意度矩阵, 支持动态调整大小和随机生成。在 MatchingResult 类中, 使用 `vector<int>` 存储配对关系, 并封装男、女及总满意度分数。此外, 还设计了 AlgorithmComparison 结构体, 用于存储匈牙利算法与暴力搜索的结果对比, 支持验证功能。

#### 4.3.2 杨宗昊设计实现

负责实现基于 KM 算法的匈牙利算法, 用于求解最佳配对问题。在 HungarianAlgorithm 类中, 设计了 dfs 方法进行深度优先搜索, 通过维护 ex 和 ey 期望值以及 slack 松弛量, 动态调整配对策略。算法的核心在于: 当遇到期望值不匹配时, 调整男/女期望值, 逐步逼近最优匹配。最终通过求解得到的总满意度矩阵 total, 计算出男、女及总满意度分数, 并封装成 MatchingResult 返回。

#### 4.3.3 时智贤设计实现

负责实现文件读写、GUI 界面及算法验证功能。在文件模块中, 设计了 loadMatrixFromFile 和 saveMatrixToFile, 支持从文本文件读取和保存满意度矩阵, 并进行格式校验。在 GUI 部分, 使用 GTK 构建图形界面, 包括矩阵显示网格、控制按钮和结果文本框, 支持随机生成、加载、保存、求解及验证开关。验证模块通过 bruteSearch 暴力搜索对比匈牙利算法结果, 确保算法正确性。

## 5 测试与调试

### 5.1 个人测试（按组员分工）

#### 5.1.1 王韵然测试

测试了 SatisfactionMatrix 的 resize 和 randomGenerate 功能，确保矩阵维度正确且随机数在合理范围内。通过手动构建简单矩阵（如对角矩阵），验证配对结果是否与预期一致。同时，测试了 MatchingResult 的初始化和赋值逻辑，确保分数计算准确。对于 AlgorithmComparison，模拟了匈牙利算法与暴力搜索结果不一致的情况，验证其是否能正确标记最优性并计算差值。

#### 5.1.2 杨宗昊测试

编写了多组测试用例，包括小规模（ $n=3$ ）和中规模（ $n=6$ ）矩阵，验证算法是否返回最大满意度配对。通过手动计算预期最优解，对比算法输出结果，确保正确性。同时，测试了算法对极端情况（如全零矩阵、全满矩阵、随机矩阵）的适应性，并验证了 dfs 函数在匹配失败时的期望调整逻辑是否正常，确保算法收敛且不陷入死循环。

#### 5.1.1 时智贤测试

测试了文件读写功能，创建了多个测试文件（含合法与非法数据），确保程序能正确解析并处理错误。GUI 测试中，逐一验证了各按钮功能是否正常，矩阵显示是否清晰，结果更新是否及时。验证功能方面，测试了  $n \leq 10$  时暴力搜索与匈牙利算法结果的一致性，以及验证开关能否正确启用或禁用对比功能，确保界面与逻辑交互无误。

### 5.2 组装与系统测试

整合算法、GUI、IO 模块

测试不同规模矩阵（ $n=1 \sim 20$ ）的求解性能

验证界面交互与结果显示

### 5.3 系统运行

#### 1. 编译命令:

```
bash
```

```
g++ *.cpp -o run.exe `pkg-config --cflags --libs gtk+-3.0`
```

#### 2. 运行界面:

设置运动员数量，点击“生成随机矩阵”

点击“求解最佳组合”显示匹配结果

可加载/保存矩阵文件（如 sample.txt）

启用验证后显示算法对比信息

羽毛球运动员最佳组合系统

运动员数量 (N): 4 生成随机矩阵 求解最佳组合 加载文件 保存文件 验证已启用

男对女满意度矩阵

	女1	女2	女3	女4
男1	43	97	26	57
男2	65	75	86	87
男3	64	88	100	40
男4	38	44	46	43

女对男满意度矩阵

	男1	男2	男3	男4
女1	71	61	5	70
女2	0	29	93	87
男3	75	89	7	37
男4	36	24	95	89

匹配结果

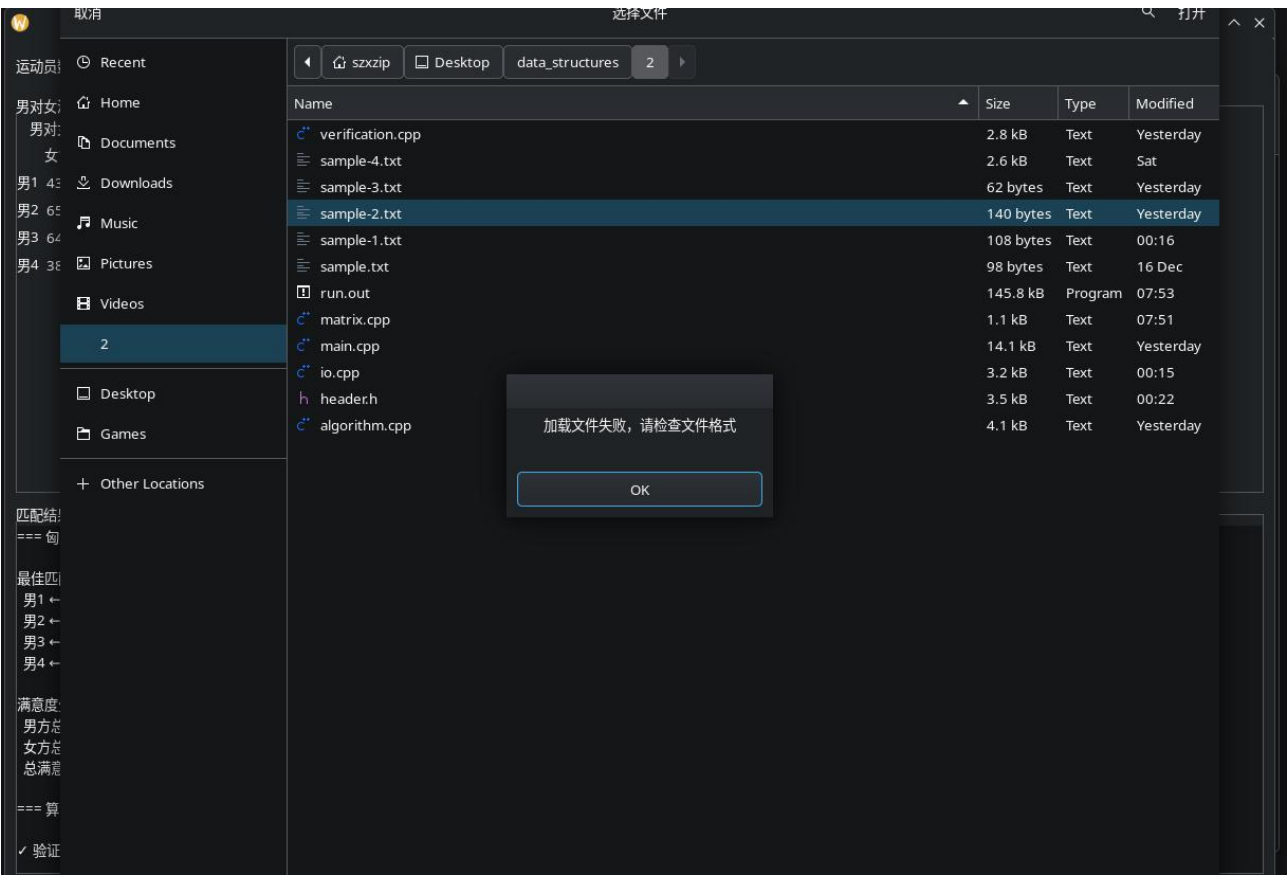
=== 匈牙利算法结果 ===

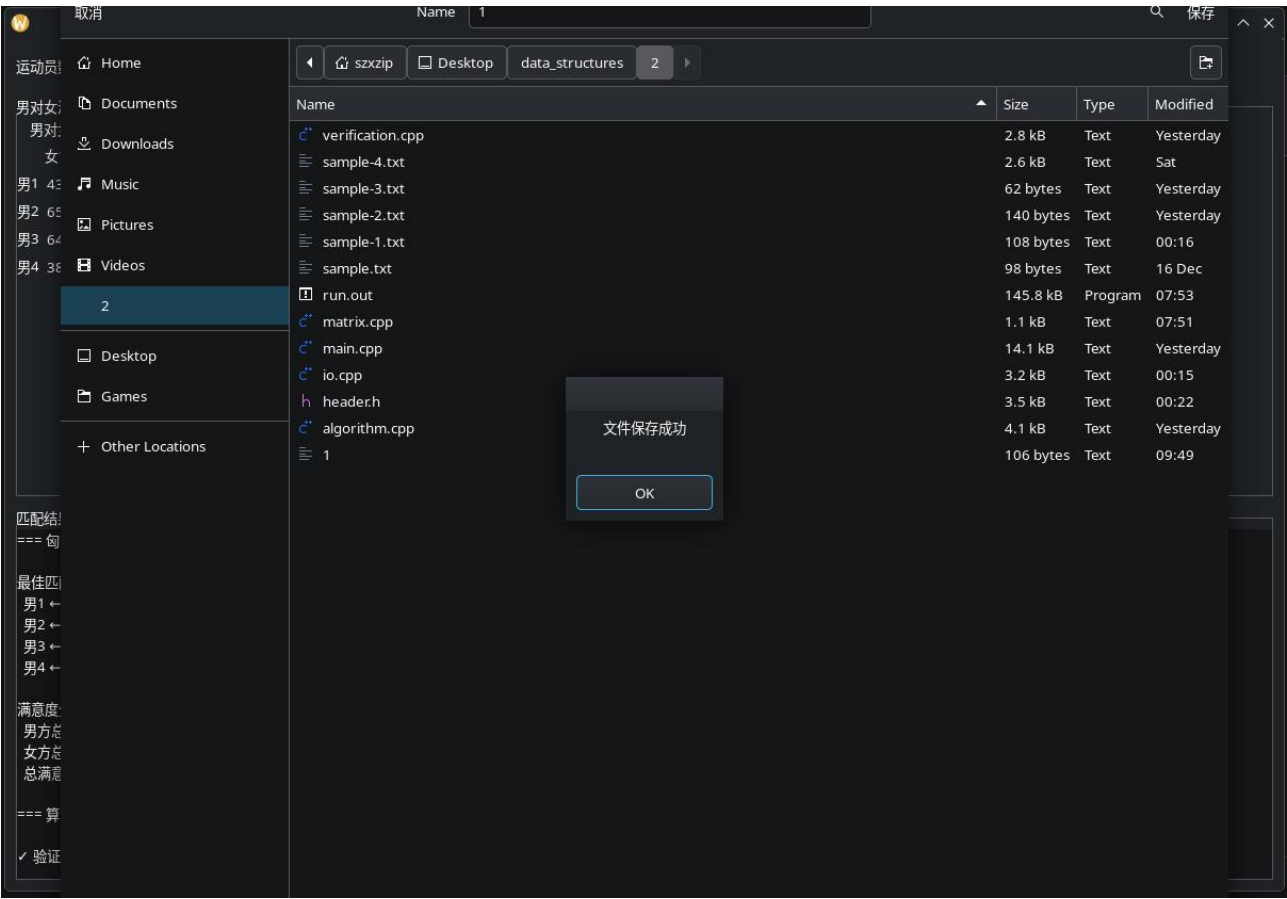
最佳匹配:  
男1 ↔ 女1  
男2 ↔ 女3  
男3 ↔ 女2  
男4 ↔ 女4

满意度分数:  
男方总满意度: 260  
女方总满意度: 342  
总满意度: 602

=== 算法验证结果 ===

✓ 验证通过: 匈牙利算法找到了最优解





羽毛球运动员最佳组合系统

运动员数量 (N): 6 - + 生成随机矩阵 求解最佳组合 加载文件 保存文件 验证已禁用

男对女满意度矩阵

男对女满意度矩阵

	女1	女2	女3	女4	女5	女6
男1	32	15	77	95	38	37
男2	27	77	31	80	37	33
男3	21	100	37	22	81	94
男4	76	65	89	4	24	7
男5	78	17	39	22	93	94
男6	97	65	20	67	78	42

女对男满意度矩阵

女对男满意度矩阵

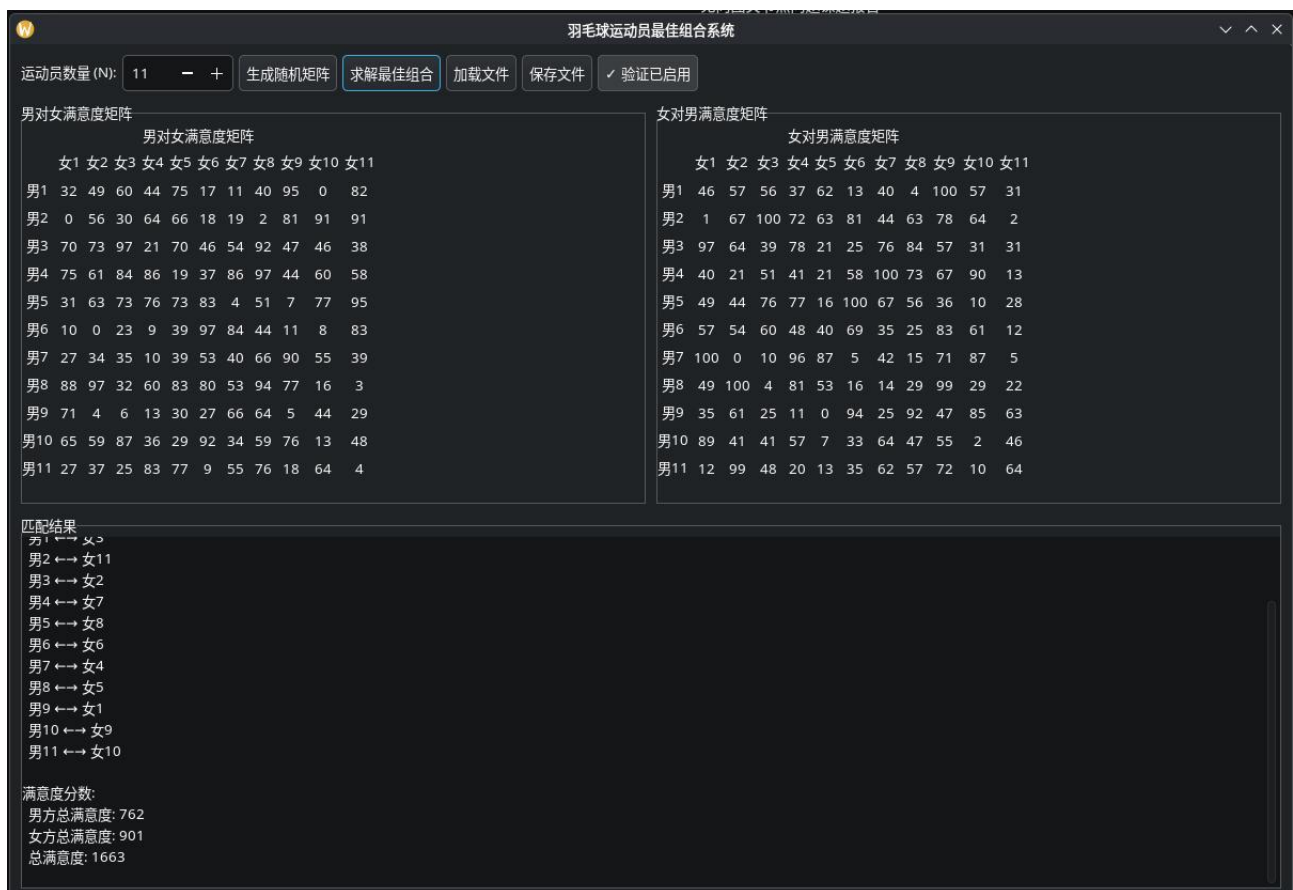
	男1	男2	男3	男4	男5	男6
女1	5	58	72	8	69	24
女2	90	92	30	30	86	76
女3	77	68	8	69	38	57
女4	54	14	69	16	47	21
女5	53	81	50	60	89	30
女6	16	43	24	62	98	67

匹配结果

=== 匈牙利算法结果 ===

最佳匹配:  
男1 ↔ 女4  
男2 ↔ 女2  
男3 ↔ 女5  
男4 ↔ 女3  
男5 ↔ 女6  
男6 ↔ 女1

满意度分数:  
男方总满意度: 533  
女方总满意度: 387  
总满意度: 920



## 6 课题总结

### 6.1 课题评价

优点:

算法实现准确，支持双方满意度最大化

GUI 界面清晰，支持矩阵可视化

验证机制增强算法可信度

不足:

暴力搜索仅支持  $n \leq 10$

界面布局在大矩阵时可能拥挤

### 6.2 团队协作

分工明确：算法、数据结构、GUI、验证各模块独立开发

使用 Git 进行版本控制与合并

定期讨论接口设计与集成问题

## 6.3 个人设计小结（按组员分工）

### 6.3.1 王韵然设计小结

通过设计矩阵与结果类，认识到良好封装对代码可读性和可维护性的重要性。使用 STL 容器简化了动态内存管理，但也需注意边界检查和数据一致性。配对结果的结构设计便于后续扩展与比较，如增加验证功能。此次实践中，体会到数据结构不仅是数据的容器，更是算法与业务逻辑之间的桥梁，合理设计能显著提升系统整体效率。

### 6.3.2 杨宗昊设计小结

通过本次实现，深刻理解了 KM 算法在二分图匹配问题中的优越性，尤其是期望值与松弛量的引入，使算法能高效逼近最优解。在实际编码中，需注意初始化期望值和访问标记的重置，防止状态污染。同时，算法需处理  $n=0$  边界情况，确保鲁棒性。这次实践不仅巩固了算法知识，也体会到优化算法与数据结构设计对性能的重要性。

### 6.3.3 时智贤设计小结

通过实现文件读写与 GUI 界面，掌握了数据持久化与用户交互设计的基本方法。文件解析需考虑格式兼容性与错误处理，避免程序崩溃。GUI 设计需注重布局合理与操作流畅，提升用户体验。验证功能的加入增强了系统的可信度，但需控制暴力搜索的规模 ( $n \leq 10$ )，避免性能问题。此次实践认识到，一个完整系统不仅需要高效算法，还需友好界面与可靠数据支撑。

## 7 附录

### B 课题设计文档

#### B-1 课程设计报告（电子版）

#### B-2 源程序代码（\*.H, \*.CPP）

### header.h

```
#ifndef HEADER_H
#define HEADER_H

#include <algorithm>
#include <cmath>
#include <cstdlib>
#include <ctime>
#include <fstream>
#include <gtk/gtk.h>
#include <iostream>
#include <sstream>
#include <string>
#include <vector>

using namespace std;
```

```

extern bool enable_verification; // 是否启用验证

// 矩阵数据结构 (B)
class SatisfactionMatrix {
public:
    // 满意度取值为 0-100
    vector<vector<int>>> mf; // 男对女满意度矩阵 (n x n)
    vector<vector<int>>> fm; // 女对男满意度矩阵 (n x n)
    int n; // 运动员数量

    SatisfactionMatrix(int size = 0); // 构造函数
    void resize(int size); // 调整矩阵大小
    void randomGenerate(); // 随机生成矩阵
};

// 配对结果 (B)
class MatchingResult {
public:
    vector<int> pairs; // 配对结果, pairs[i]=j 表示男 i 配女 j
    int totalScore; // 总满意度 (男+女)
    int maleScore; // 男方总满意度
    int femaleScore; // 女方总满意度
    MatchingResult(int size = 0); // 构造函数
};

// 匈牙利算法 (A)
class HungarianAlgorithm {
private:
    vector<vector<int>>> total; // 总的期望矩阵 就是把男女对彼此的期望相加
    vector<int> ex, ey; // 男女期望
    vector<int> matchX, matchY; // 匹配结果 matchX[i]=j 表示男 i 匹配女 j
    vector<bool> visitedX, visitedY; // DFS 搜索时的访问标记
    vector<int> slack; // 松弛量, 用于调整期望
    int n; // 问题规模
    bool dfs(int u); // DFS 深度优先搜索, 为某一位男队员匹配, 返回成功与否

public:
    HungarianAlgorithm(); // 构造函数
    MatchingResult solve(const SatisfactionMatrix& matrix); // 计算配对结果
};

// 暴力搜索 (C)
MatchingResult* bruteSearch(const SatisfactionMatrix& matrix);

```

```

// 算法结果对比 (C)
struct AlgorithmComparison {
    MatchingResult hungarian; // 匈牙利算法结果
    MatchingResult bruteForce; // 暴力搜索结果
    int n; // 问题规模
    bool isOptimal; // 匈牙利算法是否找到最优解
    int scoreDifference; // 分数差值 (暴力-匈牙利)

    AlgorithmComparison(int size = 0);
};

AlgorithmComparison* verify_hungarian_result(const SatisfactionMatrix& matrix, const MatchingResult&
hungarian_result); // 验证匈牙利算法的结果

/* *
 * 读写文件 (C)
 *
 * sample.txt:
 * 第一行: n (运动员数量)
 * n 行: 男评价女, 男行女列 (n × n)
 * n 行: 女评价男, 女行男列 (n × n)
 */
bool loadMatrixFromFile(const string& filename, SatisfactionMatrix& matrix);
bool saveMatrixToFile(const string& filename, const SatisfactionMatrix& matrix);

// GUI (C)
void activate(GtkApplication* app, gpointer user_data); // GTK 应用程序激活
void on_generate_clicked(GtkWidget* widget, gpointer data); // "生成随机矩阵"按钮
void on_solve_clicked(GtkWidget* widget, gpointer data); // "求解最佳组合"按钮
void on_load_clicked(GtkWidget* widget, gpointer data); // "加载文件"按钮
void on_save_clicked(GtkWidget* widget, gpointer data); // "保存文件"按钮
void display_matrix_in_grid(GtkWidget* grid, const vector<vector<int>>& matrix, int start_row, const string&
title); // 在网格中显示矩阵
void update_result_display(GtkWidget* textview, const MatchingResult& result); // 更新结果显示
void update_result_display_with_comparison(GtkWidget* textview, const MatchingResult& result, const
AlgorithmComparison& comparison); // 更新结果显示 (带对比信息)

#endif // HEADER_H

```

## algorithm.cpp

```

#include "header.h"

```

```

HungarianAlgorithm::HungarianAlgorithm()
: n(0)
{
    // 构造函数初始化
    cout << "HungarianAlgorithm 构造函数: 完成! " << endl;
}

bool HungarianAlgorithm::dfs(int u)
{
    visitedX[u] = true; // 表示该男选手已访问

    for (int w = 0; w < n; w++) {
        if (visitedY[w]) { // 跳过已经访问过的女选手, 防止重复递归
            continue;
        }

        int gap = ex[u] + ey[w] - total[u][w]; // 计算男女选手期望只和与总期望矩阵对应项的差

        if (gap == 0) {
            visitedY[w] = true; // 记录该女选手已经被访问

            if (matchY[w] == -1 || dfs(matchY[w])) { // 如果该女选手还没有匹配搭档或者她现在的搭档
                // 可以再另寻一个女选手组队
                matchX[u] = w;
                matchY[w] = u;
                return true;
                cout << "深度优先搜索: 完成! " << endl;
            }
        } else {
            slack[w] = min(slack[w], gap); // 更新这位女选手的松弛量, 可以理解为还差多少期望可以找到
            // 一个搭档
        }
    }

    cout << "深度优先搜索: 失败! " << endl;
    return false;
}

MatchingResult HungarianAlgorithm::solve(const SatisfactionMatrix& matrix)
{
    // 处理 n=0 边界情况
    if (matrix.n == 0) {

```

```

    MatchingResult result(0);
    return result;
}

n = matrix.n; // 问题规模为 n 人

total.resize(n, vector<int>(n, 0)); // 构建满意度矩阵
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        // 总满意度 = 男对女的满意度 + 女对男的满意度
        total[i][j] = matrix.mf[i][j] + matrix.fm[j][i];
    }
}

ex.resize(n, 0); // 初始化期望值数组
ey.resize(n, 0);
matchX.resize(n, -1); // 初始化匹配数组
matchY.resize(n, -1);
visitedX.resize(n, false);
visitedY.resize(n, false);
slack.resize(n, 0);

for (int m = 0; m < n; m++) { // 设置男选手的期望值，等于他与最合适的女选手之间的期望和
    for (int w = 0; w < n; w++) {
        ex[m] = max(ex[m], total[m][w]);
    }
}

// 为每一个男选手寻找搭档
for (int u = 0; u < n; u++) {

    // 每次为新的男选手寻找匹配时，重置所有女选手松弛量为最大值,因为后面要找最小
    for (int v = 0; v < n; v++) {
        slack[v] = 100;
    }

    while (1) { // 进入循环，直到匹配完成
        for (int i = 0; i < n; i++) { // 对于每一个男性选手的匹配过程都要重置两边选手的访问标记
            visitedX[i] = false;
            visitedY[i] = false;
        }

        if (dfs(u)) { // 找到搭档就可以退出循环了
            break;
        }
    }
}

```

```

// 没有找到的话，需要调整期望值
int d = 100; // d 表示需要调整的期望大小
for (int v = 0; v < n; v++) { // 找最小松弛量
    if (!visitedY[v]) { // 从未访问的女选手中找最小 slack 作为 d 值
        d = min(d, slack[v]);
    }
}

for (int j = 0; j < n; j++) {
    if (visitedX[j]) { // 本轮参与匹配的男选手减少期望值
        ex[j] -= d;
    }

    if (visitedY[j]) { // 本轮参与匹配的女选手增加期望值
        ey[j] += d;
    } else { // 本轮未参与的女选手 slack 降低
        slack[j] -= d;
    }
}
}

// 上面做完了匹配的任务，下面是计算结果了
int res = 0; // 分别计算总，男，女满意度
int mSat = 0;
int fSat = 0;
MatchingResult result(n);

for (int i = 0; i < n; i++) {
    if (matchX[i] == -1 || matchY[i] == -1) // 理论上不会没有匹配，以防万一跳过未被匹配的项
        continue;
    res += total[i][matchX[i]];
    mSat += matrix.mf[i][matchX[i]];
    fSat += matrix.fm[i][matchY[i]];
    result.pairs[i] = matchX[i];
}
result.totalScore = res;
result.maleScore = mSat;
result.femaleScore = fSat;

return result;
}

```

**io.cpp**

```

#include "header.h"

// 从文件加载矩阵数据
bool loadMatrixFromFile(const string& filename, SatisfactionMatrix& matrix)
{
    ifstream file(filename);
    if (!file.is_open()) {
        cerr << "错误: 无法打开文件 " << filename << endl;
        return false;
    }

    int n = 0; // 运动员数量
    file >> n; // 若开头不合规, 则之后的判定会拒绝通过

    // 运动员数量 1-20
    if (n <= 0 || n > 20) {
        cerr << "错误: 无效的运动员数量! " << endl;
        file.close();
        return false;
    }

    matrix.resize(n);

    int lineNumber = 1; // 当前行号
    string line;
    getline(file, line); // 清除第一行换行符 (和多余字符, 如果有的话)

    // 读取男对女矩阵
    for (int i = 0; i < n; i++) {
        lineNumber++;

        if (!getline(file, line)) {
            cerr << "错误: 无法读取第 " << lineNumber << " 行! " << endl;
            file.close();
            return false;
        }

        // 将行转化为字符串流
        istringstream iss(line);

        for (int j = 0; j < n; j++) {
            if (!(iss >> matrix.mf[i][j])) { // 读入数组
                cerr << "错误: 第 " << lineNumber << " 行格式不正确! " << endl;
                file.close();
            }
        }
    }
}

```

```

        return false;
    }
    if (matrix.mf[i][j] < 0 || matrix.mf[i][j] > 100) {
        cerr << "错误: 第 " << lineNumber << " 行格式不正确! " << endl;
        file.close();
        return false;
    }
}

// 读取女对男矩阵
for (int i = 0; i < n; i++) {
    lineNumber++;

    if (!getline(file, line)) {
        cerr << "错误: 无法读取第 " << lineNumber << " 行! " << endl;
        file.close();
        return false;
    }

    // 将行转化为字符串流
    istringstream iss(line);

    for (int j = 0; j < n; j++) {
        if (!(iss >> matrix.fm[i][j])) { // 读入数组
            cerr << "错误: 第 " << lineNumber << " 行格式不正确! " << endl;
            file.close();
            return false;
        }
        if (matrix.fm[i][j] < 0 || matrix.fm[i][j] > 100) {
            cerr << "错误: 第 " << lineNumber << " 行格式不正确! " << endl;
            file.close();
            return false;
        }
    }
}

file.close();
return true;
}

// 保存矩阵数据到文件
bool saveMatrixToFile(const string& filename, const SatisfactionMatrix& matrix)
{
    ofstream file(filename);

```

```

if (!file.is_open()) {
    cerr << "错误: 无法创建文件 " << filename << endl;
    return false;
}

// 保存运动员数量
file << matrix.n << endl;

// 保存男对女矩阵
for (int i = 0; i < matrix.n; i++) {
    for (int j = 0; j < matrix.n; j++) {
        file << matrix.mf[i][j] << " ";
    }
    file << endl;
}

// 保存女对男矩阵
for (int i = 0; i < matrix.n; i++) {
    for (int j = 0; j < matrix.n; j++) {
        file << matrix.fm[i][j] << " ";
    }
    file << endl;
}

file.close();
return true;
}

```

## matrix.cpp

```

#include "header.h"

// 构造函数
SatisfactionMatrix::SatisfactionMatrix(int size)
    : n(size) // 初始化矩阵维度
{
    resize(size); // 调用 resize 函数分配内存
}

// 调整矩阵大小
void SatisfactionMatrix::resize(int size)
{
    n = size; // 更新矩阵维度
    mf.resize(size);
}

```

```

        fm.resize(size);

    // 为每个子向量分配空间并初始化为 0
    for (int i = 0; i < size; i++) {
        mf[i].resize(size, 0);
        fm[i].resize(size, 0);
    }
}

// 随机生成满意度矩阵
void SatisfactionMatrix::randomGenerate()
{
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            // 生成男性 i 对女性 j 的满意度分数 (1 到 maxScore)
            mf[i][j] = rand() % 101;
            // 生成女性 j 对男性 i 的满意度分数 (1 到 maxScore)
            fm[i][j] = rand() % 101;
        }
    }
}

// 构造函数
MatchingResult::MatchingResult(int size)
: pairs(size, -1) // 初始化配对数组, 所有元素设为-1 (表示未配对)
, totalScore(0)
, maleScore(0)
, femaleScore(0)
{
}

```

## verification.cpp

```

#include "header.h"

// 暴力搜索 (n <= 10 可用)
MatchingResult* bruteSearch(const SatisfactionMatrix& matrix)
{
    int n = matrix.n;

    if (n > 10) {
        cerr << "n 过大, 暴力搜索已禁用! " << endl;
        return nullptr;
    }
}

```

```

// 构建总满意度矩阵
vector<vector<int>> totalMatrix(n, vector<int>(n, 0));
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        totalMatrix[i][j] = matrix.mf[i][j] + matrix.fm[j][i];
    }
}

// 初始化最优结果
MatchingResult* bestResult = new MatchingResult();
bestResult->totalScore = 0;

// 生成初始排列
vector<int> permutation(n);
for (int i = 0; i < n; i++) {
    permutation[i] = i;
}

// 枚举所有排列
do {
    int totalScore = 0;
    int maleScore = 0;
    int femaleScore = 0;

    // 计算当前排列的总满意度
    for (int i = 0; i < n; i++) {
        int j = permutation[i]; // 男 i 配对女 j
        maleScore += matrix.mf[i][j];
        femaleScore += matrix.fm[j][i];
        totalScore += totalMatrix[i][j];
    }

    // 更新最优解
    if (totalScore > bestResult->totalScore) {
        bestResult->totalScore = totalScore;
        bestResult->maleScore = maleScore;
        bestResult->femaleScore = femaleScore;
        bestResult->pairs = permutation;
    }

} while (next_permutation(permutation.begin(), permutation.end())); // 遍历所有排列并写入数组, 完全升序 -> 完全降序

return bestResult;

```

```

}

AlgorithmComparison::AlgorithmComparison(int size)
    : hungarian(size)
    , bruteForce(size)
    , n(size)
    , isOptimal(true)
    , scoreDifference(0)
{
}

// 验证匈牙利算法的结果
AlgorithmComparison* verify_hungarian_result(const SatisfactionMatrix& matrix, const MatchingResult&
hungarian_result)
{

    AlgorithmComparison* comparison = new AlgorithmComparison(matrix.n);
    comparison->hungarian = hungarian_result;
    comparison->n = matrix.n;

    // 检查是否启用验证
    if(!enable_verification) {
        cerr << "验证功能已禁用。" << endl;
        return nullptr;
    }

    // 检查 n 是否过大
    if (comparison->n > 10) {
        cerr << "n > 10, 验证功能已禁用。" << endl;
        return nullptr;
    }

    // 使用暴力搜索求解
    MatchingResult* bruteResult = bruteSearch(matrix);

    if (bruteResult != nullptr) {
        comparison->bruteForce = *bruteResult;
        comparison->scoreDifference = bruteResult->totalScore - hungarian_result.totalScore;
        comparison->isOptimal = (comparison->scoreDifference == 0);
        delete bruteResult;
    } else {
        cerr << "验证失败" << endl;
        return nullptr;
    }
}

```

```

    return comparison;
}

```

## main.cpp

```

#include "header.h"

// 全局变量定义
GtkWidget* male_grid; // 男对女矩阵显示网格
GtkWidget* female_grid; // 女对男矩阵显示网格
GtkWidget* result_textview; // 结果显示文本框
SatisfactionMatrix current_matrix; // 当前处理的矩阵数据
MatchingResult current_result; // 当前匹配结果
GtkWidget* n_spinner; // 运动员数量选择器

bool enable_verification = true; // 默认启用验证

// 在网格中显示矩阵
void display_matrix_in_grid(GtkWidget* grid, const vector<vector<int>>& matrix, int start_row, const string& title)
{
    // 清除旧内容
    GList* children = gtk_container_get_children(GTK_CONTAINER(grid));
    for (GList* child = children; child != NULL; child = child->next) {
        gtk_widget_destroy(GTK_WIDGET(child->data));
    }
    g_list_free(children);

    int n = matrix.size();
    if (n == 0)
        return;

    // 添加标题
    GtkWidget* title_label = gtk_label_new(title.c_str());
    gtk_grid_attach(GTK_GRID(grid), title_label, 0, start_row, n + 1, 1);

    // 添加列标题
    for (int j = 0; j < n; j++) {
        string label_text = "女" + to_string(j + 1);
        GtkWidget* col_label = gtk_label_new(label_text.c_str());
        gtk_grid_attach(GTK_GRID(grid), col_label, j + 1, start_row + 1, 1, 1);
    }
}

```

```

// 添加行标题和矩阵内容
for (int i = 0; i < n; i++) {
    string row_label_text = "男" + to_string(i + 1);
    GtkWidget* row_label = gtk_label_new(row_label_text.c_str());
    gtk_grid_attach(GTK_GRID(grid), row_label, 0, start_row + 2 + i, 1, 1);

    for (int j = 0; j < n; j++) {
        string value = to_string(matrix[i][j]);
        GtkWidget* value_label = gtk_label_new(value.c_str());
        gtk_grid_attach(GTK_GRID(grid), value_label, j + 1, start_row + 2 + i, 1, 1);
    }
}

gtk_widget_show_all(grid);
}

// 更新结果显示
void update_result_display(GtkWidget* textview, const MatchingResult& result)
{
    GtkTextBuffer* buffer = gtk_text_view_get_buffer(GTK_TEXT_VIEW(textview));
    gtk_text_buffer_set_text(buffer, "", -1);

    GtkTextIter iter;
    gtk_text_buffer_get_start_iter(buffer, &iter);

    // 匈牙利算法结果
    string result_text = "=== 匈牙利算法结果 ===\n\n";
    result_text += "最佳匹配:\n";

    for (size_t i = 0; i < result.pairs.size(); i++) {
        if (result.pairs[i] != -1) {
            result_text += "  男" + to_string(i + 1) + "  $\longleftrightarrow$  女" + to_string(result.pairs[i] + 1) + "\n";
        }
    }

    result_text += "\n 满意度分数:\n";
    result_text += "  男方总满意度: " + to_string(result.maleScore) + "\n";
    result_text += "  女方总满意度: " + to_string(result.femaleScore) + "\n";
    result_text += "  总满意度: " + to_string(result.totalScore) + "\n";

    gtk_text_buffer_insert(buffer, &iter, result_text.c_str(), -1);
}

// 更新结果显示 (有对比信息)
void update_result_display_with_comparison(GtkWidget* textview, const MatchingResult& result, const

```

AlgorithmComparison& comparison)

```
{

    GtkTextBuffer* buffer = gtk_text_view_get_buffer(GTK_TEXT_VIEW(textview));
    gtk_text_buffer_set_text(buffer, "", -1);

    GtkTextIter iter;
    gtk_text_buffer_get_start_iter(buffer, &iter);

    // 匈牙利算法结果
    string result_text = "=== 匈牙利算法结果 ===\n\n";
    result_text += "最佳匹配:\n";

    for (size_t i = 0; i < result.pairs.size(); i++) {
        if (result.pairs[i] != -1) {
            result_text += "  男" + to_string(i + 1) + "  $\longleftrightarrow$  女" + to_string(result.pairs[i] + 1) + "\n";
        }
    }

    result_text += "\n 满意度分数:\n";
    result_text += "  男方总满意度: " + to_string(result.maleScore) + "\n";
    result_text += "  女方总满意度: " + to_string(result.femaleScore) + "\n";
    result_text += "  总满意度: " + to_string(result.totalScore) + "\n";

    result_text += "\n=== 算法验证结果 ===\n\n";

    if (comparison.isOptimal) {
        result_text += "✓ 验证通过: 匈牙利算法找到了最优解\n\n";
    } else {
        result_text += "✗ 验证未通过: 匈牙利算法未找到最优解\n";
        result_text += "  暴力搜索最优解: " + to_string(comparison.bruteForce.totalScore) + "\n";
        result_text += "  分数差值: " + to_string(comparison.scoreDifference) + "\n\n";
    }

    result_text += "暴力搜索结果:\n";
    result_text += "  总满意度: " + to_string(comparison.bruteForce.totalScore) + "\n";

    gtk_text_buffer_insert(buffer, &iter, result_text.c_str(), -1);
}

// 生成随机矩阵按钮
void on_generate_clicked(GtkWidget* widget, gpointer data)
{
    int n = gtk_spin_button_get_value_as_int(GTK_SPIN_BUTTON(n_spinner));
```

```

if (n <= 0 || n > 20) {
    GtkWidget* dialog = gtk_message_dialog_new(NULL,
        GTK_DIALOG_MODAL,
        GTK_MESSAGE_ERROR,
        GTK_BUTTONS_OK,
        "运动员数量应在 1-20 之间");
    gtk_dialog_run(GTK_DIALOG(dialog));
    gtk_widget_destroy(dialog);
    return;
}

current_matrix.resize(n);
current_matrix.randomGenerate();

display_matrix_in_grid(male_grid, current_matrix.mf, 0, "男对女满意度矩阵");
display_matrix_in_grid(female_grid, current_matrix.fm, 0, "女对男满意度矩阵");
}

// 求解按钮
void on_solve_clicked(GtkWidget* widget, gpointer data)
{
    if (current_matrix.n == 0) {
        GtkWidget* dialog = gtk_message_dialog_new(NULL,
            GTK_DIALOG_MODAL,
            GTK_MESSAGE_WARNING,
            GTK_BUTTONS_OK,
            "请先生成或加载矩阵数据");
        gtk_dialog_run(GTK_DIALOG(dialog));
        gtk_widget_destroy(dialog);
        return;
    }

    // 使用匈牙利算法求解
    HungarianAlgorithm solver;
    current_result = solver.solve(current_matrix);

    // 验证结果 & 更新 GUI 显示
    if (enable_verification == true && current_matrix.n <= 10) {
        AlgorithmComparison* comparison = verify_hungarian_result(current_matrix, current_result);
        update_result_display_with_comparison(result_textview, current_result, *comparison);
    } else {
        update_result_display(result_textview, current_result);
    }
}

```

```

// 加载文件按钮
void on_load_clicked(GtkWidget* widget, gpointer data)
{
    GtkWidget* dialog = gtk_file_chooser_dialog_new("选择文件",
        NULL,
        GTK_FILE_CHOOSER_ACTION_OPEN,
        "取消", GTK_RESPONSE_CANCEL,
        "打开", GTK_RESPONSE_ACCEPT,
        NULL);

    if (gtk_dialog_run(GTK_DIALOG(dialog)) == GTK_RESPONSE_ACCEPT) {
        char* filename = gtk_file_chooser_get_filename(GTK_FILE_CHOOSER(dialog));

        if (loadMatrixFromFile(filename, current_matrix)) {
            display_matrix_in_grid(male_grid, current_matrix.mf, 0, "男对女满意度矩阵");
            display_matrix_in_grid(female_grid, current_matrix.fm, 0, "女对男满意度矩阵");
            gtk_spin_button_set_value(GTK_SPIN_BUTTON(n_spinner), current_matrix.n);
        } else {
            GtkWidget* error_dialog = gtk_message_dialog_new(NULL,
                GTK_DIALOG_MODAL,
                GTK_MESSAGE_ERROR,
                GTK_BUTTONS_OK,
                "加载文件失败， 请检查文件格式");
            gtk_dialog_run(GTK_DIALOG(error_dialog));
            gtk_widget_destroy(error_dialog);
        }

        g_free(filename);
    }

    gtk_widget_destroy(dialog);
}

// 保存文件按钮
void on_save_clicked(GtkWidget* widget, gpointer data)
{
    if (current_matrix.n == 0) {
        GtkWidget* dialog = gtk_message_dialog_new(NULL,
            GTK_DIALOG_MODAL,
            GTK_MESSAGE_WARNING,
            GTK_BUTTONS_OK,
            "没有数据可保存， 请先生成或加载矩阵");
        gtk_dialog_run(GTK_DIALOG(dialog));
        gtk_widget_destroy(dialog);
    }
}

```

```

        return;
    }

    GtkWidget* dialog = gtk_file_chooser_dialog_new("保存文件",
        NULL,
        GTK_FILE_CHOOSER_ACTION_SAVE,
        "取消", GTK_RESPONSE_CANCEL,
        "保存", GTK_RESPONSE_ACCEPT,
        NULL);

    gtk_file_chooser_set_do_overwrite_confirmation(GTK_FILE_CHOOSER(dialog), TRUE);

    if (gtk_dialog_run(GTK_DIALOG(dialog)) == GTK_RESPONSE_ACCEPT) {
        char* filename = gtk_file_chooser_get_filename(GTK_FILE_CHOOSER(dialog));
        saveMatrixToFile(filename, current_matrix);

        GtkWidget* success_dialog = gtk_message_dialog_new(NULL,
            GTK_DIALOG_MODAL,
            GTK_MESSAGE_INFO,
            GTK_BUTTONS_OK,
            "文件保存成功");
        gtk_dialog_run(GTK_DIALOG(success_dialog));
        gtk_widget_destroy(success_dialog);

        g_free(filename);
    }

    gtk_widget_destroy(dialog);
}

// 验证开关按钮
void on_verify_toggled(GtkToggleButton* button, gpointer data)
{
    enable_verification = gtk_toggle_button_get_active(button);

    // 在按钮上显示当前状态
    if (enable_verification) {
        gtk_button_set_label(GTK_BUTTON(button), "✓ 验证已启用");
    } else {
        gtk_button_set_label(GTK_BUTTON(button), "✗ 验证已禁用");
    }
}

// GTK 应用程序激活

```

```

void activate(GtkApplication* app, gpointer user_data)
{
    // 创建主窗口
    GtkWidget* window = gtk_application_window_new(app);
    gtk_window_set_title(GTK_WINDOW(window), "羽毛球运动员最佳组合系统");
    gtk_window_set_default_size(GTK_WINDOW(window), 1200, 800);
    gtk_container_set_border_width(GTK_CONTAINER(window), 10);

    // 创建主垂直布局
    GtkWidget* main_box = gtk_box_new(GTK_ORIENTATION_VERTICAL, 10);
    gtk_container_add(GTK_CONTAINER(window), main_box);

    // 控制面板
    GtkWidget* control_box = gtk_box_new(GTK_ORIENTATION_HORIZONTAL, 5);
    gtk_box_pack_start(GTK_BOX(main_box), control_box, FALSE, FALSE, 0);

    // 运动员数量选择
    GtkWidget* n_label = gtk_label_new("运动员数量 (N):");
    gtk_box_pack_start(GTK_BOX(control_box), n_label, FALSE, FALSE, 0);

    // 创建 SpinButton
    GtkAdjustment* adj = gtk_adjustment_new(4, 1, 20, 1, 5, 0);
    n_spinner = gtk_spin_button_new(adj, 1, 0);
    gtk_box_pack_start(GTK_BOX(control_box), n_spinner, FALSE, FALSE, 0);

    // 按钮
    GtkWidget* generate_button = gtk_button_new_with_label("生成随机矩阵");
    g_signal_connect(generate_button, "clicked", G_CALLBACK(on_generate_clicked), NULL);
    gtk_box_pack_start(GTK_BOX(control_box), generate_button, FALSE, FALSE, 0);

    GtkWidget* solve_button = gtk_button_new_with_label("求解最佳组合");
    g_signal_connect(solve_button, "clicked", G_CALLBACK(on_solve_clicked), NULL);
    gtk_box_pack_start(GTK_BOX(control_box), solve_button, FALSE, FALSE, 0);

    GtkWidget* load_button = gtk_button_new_with_label("加载文件");
    g_signal_connect(load_button, "clicked", G_CALLBACK(on_load_clicked), NULL);
    gtk_box_pack_start(GTK_BOX(control_box), load_button, FALSE, FALSE, 0);

    GtkWidget* save_button = gtk_button_new_with_label("保存文件");
    g_signal_connect(save_button, "clicked", G_CALLBACK(on_save_clicked), NULL);
    gtk_box_pack_start(GTK_BOX(control_box), save_button, FALSE, FALSE, 0);

    GtkWidget* verify_toggle = gtk_toggle_button_new_with_label("✓ 验证已启用");
    gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(verify_toggle), enable_verification);
}

```

```

g_signal_connect(verify_toggle, "toggled", G_CALLBACK(on_verify_toggled), NULL);
gtk_box_pack_start(GTK_BOX(control_box), verify_toggle, FALSE, FALSE, 0);

// 创建水平布局放置两个矩阵
GtkWidget* matrix_box = gtk_box_new(GTK_ORIENTATION_HORIZONTAL, 10);
gtk_box_pack_start(GTK_BOX(main_box), matrix_box, TRUE, TRUE, 0);

// 男对女矩阵网格
GtkWidget* male_frame = gtk_frame_new("男对女满意度矩阵");
gtk_box_pack_start(GTK_BOX(matrix_box), male_frame, TRUE, TRUE, 0);

GtkWidget* male_scroll = gtk_scrolled_window_new(NULL, NULL);
gtk_container_add(GTK_CONTAINER(male_frame), male_scroll);
gtk_scrolled_window_set_policy(GTK_SCROLLED_WINDOW(male_scroll),
    GTK_POLICY_AUTOMATIC, GTK_POLICY_AUTOMATIC);

male_grid = gtk_grid_new();
gtk_grid_set_row_spacing(GTK_GRID(male_grid), 5);
gtk_grid_set_column_spacing(GTK_GRID(male_grid), 5);
gtk_container_add(GTK_CONTAINER(male_scroll), male_grid);

// 女对男矩阵网格
GtkWidget* female_frame = gtk_frame_new("女对男满意度矩阵");
gtk_box_pack_start(GTK_BOX(matrix_box), female_frame, TRUE, TRUE, 0);

GtkWidget* female_scroll = gtk_scrolled_window_new(NULL, NULL);
gtk_container_add(GTK_CONTAINER(female_frame), female_scroll);
gtk_scrolled_window_set_policy(GTK_SCROLLED_WINDOW(female_scroll),
    GTK_POLICY_AUTOMATIC, GTK_POLICY_AUTOMATIC);

female_grid = gtk_grid_new();
gtk_grid_set_row_spacing(GTK_GRID(female_grid), 5);
gtk_grid_set_column_spacing(GTK_GRID(female_grid), 5);
gtk_container_add(GTK_CONTAINER(female_scroll), female_grid);

// 结果显示区域
GtkWidget* result_frame = gtk_frame_new("匹配结果");
gtk_box_pack_start(GTK_BOX(main_box), result_frame, FALSE, FALSE, 0);
gtk_widget_set_size_request(result_frame, -1, 350);

GtkWidget* result_scroll = gtk_scrolled_window_new(NULL, NULL);
gtk_container_add(GTK_CONTAINER(result_frame), result_scroll);

result_textview = gtk_text_view_new();
gtk_text_view_set_editable(GTK_TEXT_VIEW(result_textview), FALSE);

```

```

gtk_text_view_set_wrap_mode(GTK_TEXT_VIEW(result_textview), GTK_WRAP_WORD);
gtk_container_add(GTK_CONTAINER(result_scroll), result_textview);

// 初始生成一个矩阵
on_generate_clicked(NULL, NULL);

gtk_widget_show_all(window);
}

int main(int argc, char** argv)
{
    GtkApplication* app;
    int status;

    app = gtk_application_new("org.gtk.athlete.matching", G_APPLICATION_DEFAULT_FLAGS);
    g_signal_connect(app, "activate", G_CALLBACK(activate), NULL);
    status = g_application_run(G_APPLICATION(app), argc, argv);
    g_object_unref(app);

    return status;
}

```

## sample.txt

```

4
92 85 78 91
88 94 86 89
79 87 95 82
91 84 88 93
89 92 85 87
86 90 93 88
84 89 91 90
93 87 86 92

```

### C.1 运行环境说明

## 部署 Windows GUI, 以及编译

> 提示: GUI 采用 GTK3, 可参考下面步骤来配置环境。

1. 安装 [msys2](https://www.msys2.org/)。(用来创建一个 linux 环境)

2. 打开 MSYS2 **\*\*UCRT64\*\***。(有好几个环境, ucrt64 更常用些)

> 以下都是在此 ucrt64 环境中执行

3. 安装 gtk3。

```
...  
pacman -S mingw-w64-ucrt-x86_64-gtk3  
...
```

4. 安装 C++ 编译工具。

```
...  
pacman -S mingw-w64-ucrt-x86_64-toolchain base-devel  
...
```

5. 切换到项目的目录。

```
...  
cd X:/path/to/codes  
...
```

`cd` = change directory。`X:` 是盘符, 比如 C 盘。`/` 用来代替 `\` (linux 和 win 的路径分隔符不一样)。

比如

```
...  
cd C:/users/me/Desktop/data_structures/1/  
...
```

6. 编译。

```
...  
g++ -o run.exe sample1.cpp sample2.cpp `pkg-config --cflags --libs gtk+-3.0` -mwindows  
...
```

`g++` 是 C++ 编译工具。`-o run.exe` 输出可执行文件。`sample1.cpp` 编译 sample1.cpp。`pkg-config --cflags --libs gtk+-3.0` 添加 gtk3。