南開大學
Nankai University

计算机学院

算法导论期末设计报告

公司项目分配与团队管理

姓名：赵元鸣

学号：2211757

专业：计算机科学与技术

2024 年 6 月 6 日

目录

1 问题简介

在现代企业管理中，团队与项目的高效匹配是提升生产力和项目成功率的关键因素。高效的团队和项目匹配不仅可以最大化资源利用率，提升工作效率，还能增强团队成员的工作满意度和积极性。然而，如何在多个团队和项目之间进行最佳匹配，同时考虑团队和项目的偏好、项目的优先级以及项目之间的依赖关系，是一个复杂且具有挑战性的问题。传统的手工分配方法不仅耗时费力，而且难以保证分配结果的最优性和公平性。手工分配容易受到人为因素的影响，导致分配结果的不合理性，甚至可能引发团队内部的矛盾和不满。此外，手工分配难以处理大量数据和复杂的依赖关系，无法适应现代企业管理的需求。因此，开发一个智能化的团队与项目匹配系统显得尤为重要。

本实验旨在设计并实现一个团队与项目匹配系统，通过考虑团队和项目的偏好、项目的优先级以及项目之间的依赖关系，实现团队与项目的最佳匹配。该系统不仅能自动化地完成匹配过程，还能保证分配结果的合理性和公平性。系统的设计目标是提高匹配过程的效率和准确性，减少人为干预，确保每个团队都能获得最适合的项目，从而提升整体工作效率和项目成功率。

2 实现思路

实现团队与项目匹配系统的思路主要分为三个核心部分：数据输入处理、项目排序算法和团队与项目的匹配算法。首先，系统需要获取用户输入的团队和项目的基本信息。这包括团队和项目的数量、名称、偏好、项目之间的依赖关系以及项目的优先级权重。用户输入的数据将作为系统进行后续计算的基础。因此，必须设计一个友好的用户输入界面，确保数据输入的准确性和完整性。具体步骤包括输入团队和项目的数量，用户首先输入团队和项目的数量，系统根据数量初始化相应的数据结构。接着，用户依次输入每个团队和项目的名称，系统将这些名称存储在哈希映射中，以便后续使用。然后，用户输入每个团队对所有项目的偏好顺序，以及每个项目对所有团队的偏好顺序。这些偏好信息将存储在矩阵中。用户还需要输入项目之间的依赖关系，系统将这些依赖关系存储在邻接表中，并更新每个项目的入度。最后，用户输入每个项目的优先级权重，系统将这些权重存储在数组中。在获取到所

有输入数据后，系统首先需要对项目进行排序，以确保依赖关系得到正确处理。我们采用拓扑排序算法来实现这一功能。拓扑排序的具体步骤包括初始化队列，将所有入度为 0 的项目加入队列；处理队列，从队列中依次取出项目，并将其加入排序结果中。对于每个取出的项目，减少其依赖项目的入度，如果某个依赖项目的入度变为 0，则将其加入队列。最后，如果排序结果中的项目数量不等于项目总数，则说明存在循环依赖，系统应提示错误并终止执行。

在项目排序完成后，系统将按照排序结果和项目的优先级权重进行团队与项目的匹配。我们采用 Gale-Shapley 算法来实现这一功能。具体步骤包括初始化，将所有团队标记为未匹配状态，并初始化每个项目的匹配团队为-1。接着，对于每个未匹配的团队，按照其偏好顺序依次尝试匹配项目。如果项目未被匹配，则直接匹配；如果项目已被匹配，则根据项目的偏好顺序决定是否替换当前匹配的团队。重复上述过程，直到所有团队都匹配到项目为止。通过上述步骤，系统能够实现团队与项目的高效匹配，并保证分配结果的合理性和公平性。最终，系统将匹配结果和项目优先级顺序输出到文件和控制台，以便用户查看和分析。

算法部分伪代码如下：

Algorithm 1 项目分配算法

Input: 团队数量 $TEAM\_COUNT$,项目数量 $PROJECT\_COUNT$,团队的项目偏好 $teamPreferences$, 项目的团队偏好 $projectPreferences$

Output: 团队的项目匹配 $teamMatch$，项目的团队匹配 $projectMatch$

```
function allocateProjects(teamMatch, projectMatch)
    初始化一个全为真的团队空闲列表 teamFree，长度为 TEAM_COUNT
    初始化空闲团队数量 freeCount 为 TEAM_COUNT
    while 仍有团队空闲 do
        找出第一个空闲的团队 team
        for 每个项目 i，直到团队 team 不再空闲 do
            获取团队 team 偏好的项目 project
            if 项目 project 还未被分配 then
                将项目 project 分配给团队 team
                将团队 team 标记为已分配
                空闲团队数量 freeCount 减 1
            else
                获取当前被分配项目 project 的团队 currentTeam
                判断项目 project 是否更偏好新的团队 team
                for 每个团队 j，直到找到项目 project 更偏好的团队或者当前团队 do
                    if 项目 project 更偏好团队 team then
                        将项目 project 重新分配给团队 team
                        将团队 team 标记为已分配，将团队 currentTeam 标记为空闲
                    end if
                end for
            end if
        end for
    end while
end function
```

Algorithm 2 项目排序算法

Input: 项目数量 $PROJECT\_COUNT$，项目的入度 $inDegree$，项目的邻接表 adj

Output: 项目的拓扑排序结果 $order$

```
function sortProjects
    order ← 空列表
    q ← 空队列
    for 每个项目 i 从 0 到 PROJECT_COUNT − 1 do
        if inDegree[i] == 0 then
            q.push(i)
        end if
```

```
    end for
    while 队列 q 不为空 do
        u ← q.front()
        q.pop()
        order.push_back(u)
        for 每个 v 在 adj[u] 中 do
            inDegree[v] ← inDegree[v] − 1
            if inDegree[v] == 0 then
                q.push(v)
            end if
        end for
    end while
    if order.size() ≠ PROJECT_COUNT then
        cerr ← "Error :        "
        exit(1)
    end if
    return order
end function
```

3 代码实现

3.1 实验环境

1. cpp 版本：199711
2. 开发环境：VisualStudio2022 Community
3. 系统：Windows11 22621.3593

表 1: 软件环境

1. CPU:AMD Ryzen 9 7945HX with Radeon Graphics 2.50 GHz
2.GPU:RTX 4060
3. 电脑型号：联想拯救者 R9000P

表 2: 硬件环境

3.2 代码展示

测试代码所用的用例和输入输出格式都在提交的压缩包中，可供查阅

变量定义

```cpp
// 定义团队和项目的数量
int TEAM_COUNT;
int PROJECT_COUNT;

// 团队和项目的详细信息
```

```
unordered_map<int, string> teamNames;
unordered_map<int, string> projectNames;

// 团队对项目的偏好
vector<vector<int>> teamPreferences;
vector<vector<int>> projectPreferences;

// 项目之间的依赖关系（拓扑排序）
vector<vector<int>> adj;
vector<int> inDegree;

// 项目优先级权重
vector<int> projectWeights;
```

用固定格式获取用户输入

```
void getInput() {
    cout << "请输入团队数量: ";
    cin >> TEAM_COUNT;
    cout << "请输入项目数量: ";
    cin >> PROJECT_COUNT;

    teamPreferences.resize(TEAM_COUNT, vector<int>(PROJECT_COUNT));
    projectPreferences.resize(PROJECT_COUNT, vector<int>(TEAM_COUNT));
    adj.resize(PROJECT_COUNT);
    inDegree.resize(PROJECT_COUNT, 0);
    projectWeights.resize(PROJECT_COUNT);

    cout << "请输入每个团队的名称:" << endl;
    for (int i = 0; i < TEAM_COUNT; i++) {
        cout << "团队 " << i << " 名称: ";
        string name;
        cin >> name;
        teamNames[i] = name;
    }

    cout << "请输入每个项目的名称:" << endl;
    for (int i = 0; i < PROJECT_COUNT; i++) {
        cout << "项目 " << i << " 名称: ";
        string name;
        cin >> name;
        projectNames[i] = name;
    }

    cout <<
        "请输入团队对项目的偏好（每行输入一个团队对所有项目的偏好，按优先级从高到低排列）:"
        << endl;
    for (int i = 0; i < TEAM_COUNT; i++) {
        for (int j = 0; j < PROJECT_COUNT; j++) {
```

```
            cin >> teamPreferences[i][j];
        }
    }

    cout <<
        "请输入项目对团队的偏好（每行输入一个项目对所有团队的偏好，按优先级从高到低排列）:"
        << endl;
    for (int i = 0; i < PROJECT_COUNT; i++) {
        for (int j = 0; j < TEAM_COUNT; j++) {
            cin >> projectPreferences[i][j];
        }
    }

    cout << "请输入项目之间的依赖关系（格式：项目A 项目B 表示项目A依赖于项目B，输入-1
        -1结束）:" << endl;
    while (true) {
        int a, b;
        cin >> a >> b;
        if (a == -1 && b == -1) break;
        if (a < 0 || a >= PROJECT_COUNT || b < 0 || b >= PROJECT_COUNT) {
            cerr << "无效的项目编号！" << endl;
            continue;
        }
        adj[b].push_back(a);
        inDegree[a]++;
    }

    cout << "请输入每个项目的优先级权重（按项目编号顺序）:" << endl;
    for (int i = 0; i < PROJECT_COUNT; i++) {
        cin >> projectWeights[i];
    }
}
```

进行项目分配

```
void allocateProjects(vector<int>& teamMatch, vector<int>& projectMatch) {
    vector<bool> teamFree(TEAM_COUNT, true);
    int freeCount = TEAM_COUNT;

    while (freeCount > 0) {
        int team;
        for (team = 0; team < TEAM_COUNT; team++) {
            if (teamFree[team]) break;
        }

        for (int i = 0; i < PROJECT_COUNT && teamFree[team]; i++) {
            int project = teamPreferences[team][i];

            if (projectMatch[project] == -1) {
```

```
                projectMatch[project] = team;
                teamMatch[team] = project;
                teamFree[team] = false;
                freeCount--;
            }
            else {
                int currentTeam = projectMatch[project];
                bool preferNewTeam = false;
                for (int j = 0; j < TEAM_COUNT; j++) {
                    if (projectPreferences[project][j] == team) {
                        preferNewTeam = true;
                        break;
                    }
                    if (projectPreferences[project][j] == currentTeam) break;
                }

                if (preferNewTeam) {
                    projectMatch[project] = team;
                    teamMatch[team] = project;
                    teamFree[team] = false;
                    teamFree[currentTeam] = true;
                }
            }
        }
    }
}
```

进行项目排序

```
vector<int> sortProjects() {
    vector<int> order;
    queue<int> q;

    for (int i = 0; i < PROJECT_COUNT; i++) {
        if (inDegree[i] == 0) {
            q.push(i);
        }
    }

    while (!q.empty()) {
        int u = q.front();
        q.pop();
        order.push_back(u);

        for (int v : adj[u]) {
            if (--inDegree[v] == 0) {
                q.push(v);
            }
        }
```

```
    }

    if (order.size() != PROJECT_COUNT) {
        cerr << "Error: 依赖关系中存在循环! " << endl;
        exit(1);
    }

    return order;
}
```

将运行结果储存到文件内以及命令行显示菜单

```
void outputToFile(const vector<int>& teamMatch, const vector<int>& order) {
    ofstream outFile("output.txt");
    if (!outFile) {
        cerr << "无法打开输出文件! " << endl;
        return;
    }

    outFile << "团队分配结果:" << endl;
    for (int i = 0; i < TEAM_COUNT; i++) {
        outFile << "团队 " << teamNames[i] << " -> 项目 " <<
            projectNames[teamMatch[i]] << endl;
    }

    outFile << "项目优先级顺序:" << endl;
    for (int i : order) {
        outFile << "项目 " << projectNames[i] << " ";
    }
    outFile << endl;

    outFile.close();
}

// 显示菜单
void showMenu() {
    cout << "请选择操作:" << endl;
    cout << "1. 输入数据" << endl;
    cout << "2. 执行匹配并输出结果" << endl;
    cout << "3. 退出" << endl;
}
```

4 结果总结

4.1 样例分析

我们通过设计了 4 个不同情况的测试用例，逐步验证了程序的正确性，这里以测试用例 2 为例，我们来看一下程序运行的过程。用例 2 如下：

测试用例 2：

输入：

- 团队数量：4
- 项目数量：4
- 团队名称：TeamA, TeamB, TeamC, TeamD
- 项目名称：Project1, Project2, Project3, Project4
- 团队对项目的偏好：
  - TeamA：Project1, Project2, Project3, Project4
  - TeamB：Project2, Project3, Project4, Project1
  - TeamC：Project2, Project3, Project1, Project4
  - TeamD：Project4, Project1, Project2, Project3
- 项目对团队的偏好：
  - Project1：TeamA, TeamB, TeamC, TeamD
  - Project2：TeamB, TeamC, TeamD, TeamA
  - Project3：TeamA, TeamD, TeamC, TeamB
  - Project4：TeamD, TeamA, TeamB, TeamC
- 项目之间的依赖关系：
  - Project2 依赖 Project1
  - Project3 依赖 Project2
  - Project4 依赖 Project1
- 项目优先级权重：
  - Project1：10
  - Project2：20
  - Project3：10
  - Project4：60

在测试用例 2 中，我们有 4 个团队（TeamA、TeamB、TeamC 和 TeamD）和 4 个项目（Project1、Project2、Project3 和 Project4）。每个团队和项目都有各自的偏好列表。例如，TeamA 最喜欢 Project1，TeamB 最喜欢 Project2，TeamC 最喜欢 Project2，TeamD 最喜欢 Project4。项目对团队的偏好也各不相同：Project1 最喜欢 TeamA，Project2 最喜欢 TeamB，Project3 最喜欢 TeamA，Project4 最喜欢 TeamD。此外，项目之间存在依赖关系：Project2 依赖 Project1，Project3 依赖 Project2，Project4 依赖 Project1。这意味着在开始 Project2 之前，必须先完成 Project1；同样，Project3 依赖于 Project2，而 Project4 依赖于 Project1。每个项目还有不同的优先级权重，分别为 Project1（10），Project2（20），Project3（10），和 Project4（60）。

首先，使用 Gale-Shapley 算法进行稳定匹配。算法从 TeamA 开始，让 TeamA 选择它最喜欢的 Project1。如果 Project1 没有被其他团队匹配，那么 TeamA 与 Project1 匹配。同样地，TeamB 选择

它最喜欢的 Project2，并与其匹配，因为 Project2 也没有被其他团队匹配。接下来，TeamC 选择它最喜欢的 Project2，但由于 Project2 已经被 TeamB 匹配，TeamC 会选择其次喜欢的 Project3，并成功匹配。最后，TeamD 选择它最喜欢的 Project4，并与其匹配。通过这个过程，我们得到了初步的匹配结果：TeamA 与 Project1 匹配，TeamB 与 Project2 匹配，TeamC 与 Project3 匹配，TeamD 与 Project4 匹配。

接下来，我们进行拓扑排序以考虑项目的依赖关系。首先计算每个项目的入度，发现 Project1 和 Project4 的入度为 0，表明它们没有依赖关系，因此可以首先处理这两个项目。将它们添加到队列中，然后开始处理队列中的项目。处理 Project1 后，减少所有依赖于 Project1 的项目的入度，即 Project2 和 Project4。Project4 已经处理完毕，无需进一步操作。Project2 的入度变为 0，接着处理 Project2，减少依赖于 Project2 的 Project3 的入度，最终 Project3 的入度也变为 0，可以处理。

结合优先级权重，我们最终的项目处理顺序为 Project4（权重 60），Project1（权重 10），Project2（权重 20），和 Project3（权重 10）。这种顺序不仅满足了项目之间的依赖关系，还考虑了每个项目的重要性。因此，分配结果为 TeamA 处理 Project1，TeamB 处理 Project2，TeamC 处理 Project3，TeamD 处理 Project4。整个过程确保了团队与项目的匹配是稳定的，同时项目的处理顺序也遵循了依赖关系和优先级权重。

经运行检验，结果与预期完全相同。执行结果如图所示：

```
请输入团队数量：4
请输入项目数量：4
请输入每个团队的名称：
团队 0 名称：A
团队 1 名称：B
团队 2 名称：C
团队 3 名称：D
请输入每个项目的名称：
项目 0 名称：Project1
项目 1 名称：Project2
项目 2 名称：Project3
项目 3 名称：Project4
请输入团队对项目的偏好（每行输入一个团队对所有项目的偏好，按优先级从高到低排列）：
0 1 2 3
1 2 3 0
1 2 0 3
3 0 1 2
请输入项目对团队的偏好（每行输入一个项目对所有团队的偏好，按优先级从高到低排列）：
 2 1 3 0
1 2 0 3
0 3 2 1
3 0 1 2
请输入项目之间的依赖关系（格式：项目A 项目B 表示项目A依赖于项目B，输入-1 -1结束）：
1 0
2 1
3 0
-1 -1
请输入每个项目的优先级权重（按项目编号顺序）：
10
20
10
60
请选择操作：
1. 输入数据
2. 执行匹配并输出结果
3. 退出
2
团队分配结果：
团队 A -> 项目 Project1
团队 B -> 项目 Project2
团队 C -> 项目 Project3
团队 D -> 项目 Project4
项目优先级顺序：
项目 Project4 项目 Project2 项目 Project1 项目 Project3
```

图 4.1: 用例 2 执行结果

4.2 算法复杂度分析

这段代码的主要部分包含两个算法：拓扑排序和稳定匹配。拓扑排序用于确定项目之间的依赖关系，而稳定匹配则用于将团队与项目进行匹配。

首先，我们来看项目排序。所使用的拓扑排序的时间复杂度为 O(V+E)，其中 V 代表顶点数量，E 代表边的数量。在这个程序中，顶点对应的是项目的数量（PROJECT_COUNT），边对应的是项目之间的依赖关系数量。因此，拓扑排序的时间复杂度可以表示为 O(PROJECT_COUNT + 依赖关系的数量)。

其次，我们来看项目与团队的匹配算法。这个算法的时间复杂度为 O(n2)，其中 n 代表团队的数量（TEAM_COUNT）。在最坏的情况下，每个团队可能需要考虑所有的项目才能找到最优的匹配，因此稳定匹配算法的时间复杂度为 O(TEAM_COUNT2)。

综合上述分析，整个程序的时间复杂度可以表示为 O(m + k + n2)，其中 m 代表 PROJECT_COUNT，k 代表依赖关系的数量，n 代表 TEAM_COUNT。。这意味着，随着项目数量、项目之间的依赖关系数量以及团队数量的增加，程序的运行时间将会相应地增加。

4.3 改进思路思考

在本次实验中，我认为有几个方向可以进行改进。首先，我们可以优化项目和团队的匹配算法。当前的算法主要基于项目的优先级权重、团队的偏好和项目对团队的偏好。这种方法虽然简单，但可能并不总是能得到最优的分配结果。我们可以考虑引入更复杂的匹配算法，例如使用加权稳定匹配算法或多目标优化算法，以便在考虑更多因素的情况下得到更优的分配结果。虽然本次实验使用的是稳定婚姻问题的 Gale-Shapley 算法，这种算法可以保证得到稳定的匹配结果，但在实际应用中，引入更多的复杂性和权衡可能会进一步提高匹配的效果。

其次，我们可以引入更多的项目和团队特性。目前的算法只考虑了项目的优先级权重、项目之间的依赖关系、团队的偏好以及项目对团队的偏好。但是，项目和团队可能还有其他的重要特性，如项目的难度、所需技能、预算限制、完成时间以及团队的技能水平和经验。如果我们能够将这些特性也纳入算法中，匹配过程将更加全面和细致，从而使得项目分配更加公平和合理。例如，对于复杂项目，应优先分配给具备相应技能和经验的团队；而对于时间紧迫的项目，应优先分配给可以快速响应和高效完成任务的团队。

此外，我们还可以改进算法的灵活性和可扩展性。当前的算法在面对动态变化的环境时可能显得不足，例如新项目的加入、现有项目的变更或团队成员的变动。通过开发更加灵活和适应性强的算法，我们可以实时更新匹配结果，确保资源配置始终处于最优状态。并且考虑到实际应用中的多样化需求，我们可以在算法中引入更多的用户定义参数和约束条件，允许管理人员根据具体需求调整匹配策略。这将使得算法不仅能在通用场景中发挥作用，还能在特定场景中实现个性化优化，从而提高系统的实用性和用户满意度。

4.4 使用思路拓展

这个项目的应用场景广泛。我们可以将他用于各个不同的领域。首先，它可用于公司内部的项目分配。每个团队都有自己擅长的领域，每个项目也需要特定的技能。通过您的项目，可以高效地将团队与项目进行匹配，使得每个团队都能在其擅长的领域发挥出最大的效能，从而提高整个公司的效率。

其次，这个项目也可以应用于学校的课程分配。每个学生都有自己喜欢的课程，每个课程也有其特定的学习要求。通过此项目，可以将学生与课程进行有效匹配，使得每个学生都能学到他们感兴趣的课程，提高学生的学习兴趣和效率。

在医疗行业中，医生和病人的匹配问题是非常重要的。每个医生都有他们擅长的疾病类型和治疗方法，每个病人也有他们特定的病症和治疗需求。通过此项目，可以将医生和病人进行有效匹配，使得每个病人都能得到最适合他们的医疗服务。

此外，这个项目也可以应用于社区服务。每个社区都有不同的需求，每个志愿者也有他们擅长的服务领域。通过此项目，可以将志愿者与社区需求进行有效匹配，使得每个社区都能得到最需要的服务，提升社区服务的质量和效率。

4.5 项目总结

通过这个项目我也了解到，如何发现现实生活中的实际需求，并根据这些需求设计并实现一个有效的系统，这是一项非常重要的技能。同时，我深入理解了稳定匹配算法和拓扑排序的原理和应用，如何合理的设计测试用例等等知识。通过代码编写也提高了我的编程技能和问题解决能力。

最后感谢这门算法导论课程以及苏老师和助教老师，我学习了许多基本的算法和数据结构，这些知识虽然对于我来说学习起来有些难度，但是却是一个计算机学生必不可少的理论素质的训练。此外，我也学习了如何分析和优化算法的性能，这对我设计高效的算法非常有帮助。希望我带着这门课程带给我的思考继续前进，也祝算法导论课程越办越好。