上海大学

SHANGHAI UNIVERSITY

<数据结构>研讨报告

总分 100		得分
报告	逻辑结构（40 分）	
	语言表达（30 分）	
	格式规范（20 分）	
	个人体会（10 分）	
合 计		

学　　院　计算机工程与科学学院

组　　名　范舒舰

题　　目　并查集的设计与实现

日　　期　2025-04-12

学号	姓名	专业	分工
23121677	范舒舰	网络空间安全	代码挑战
23120651	程铭	计算机科学与技术	基础代码结构
23121632	罗溴予	网络空间安全	测试、PPT 制作讲解
23120654	王子申	网络空间安全	测试优化、报告撰写

一、目的与要求

本次研讨的主要目的是通过实现并查集（UFSets）数据结构，掌握其在社交网络分析中的应用。具体要求包括：

1. 实现并查集的基本操作（查找、合并、路径压缩等）。
2. 应用并查集解决朋友圈划分问题。
3. 实现社交网络中的好友关系管理功能。

二、实验环境

1. Windows 11
2. Visual studio2022

三、实验内容

1. 实现并查集数据结构，包括基本操作和优化策略。
2. 从文件中读取用户数据并初始化并查集。
3. 实现社交网络功能，包括：
    - 查询直接好友关系
    - 查询间接好友关系
    - 统计朋友圈数量
    - 查找最大朋友圈
    - 添加新用户和好友关系
    - 删除用户和好友关系
    - 查找两个用户之间的连接路径

四、实验内容的设计与实现

[说明：选择本次研讨中最有设计技巧或特色的算法实现部分，使用必要的代码和文字进行介绍。避免大篇幅的复制粘贴源代码。 **]**

在基础实现的基础上，我们进行了以下存储优化：

1. **动态扩容机制**

    该函数实现了并查集的动态扩容能力，当元素数量超过当前容量时，会自动分配更大的存储空间并迁移现有数据，通过倍增策略（如 1.5 或 2 倍扩容）保证插入操作的平均时间复杂度为 O(1)，有效解决了数据集规模不确定时的内存管理问题。

```
void Resize(int newSize) {
    ElemNode<ElemType>* newSets = new ElemNode<ElemType>[newSize];
    // 数据迁移和初始化...
    delete[] sets;
    sets = newSets;
    size = newSize;
}
```

该机制在用户数量超过预设值时自动扩容，避免频繁重建集合。

2. **懒删除策略**

采用标记删除而非立即移除的方式处理删除操作，配合定期压缩机制来回收空间，将单次删除操作的时间复杂度优化到 O(1)，特别适合存在频繁删除插入交替操作的场景，在保持数据结构完整性的同时提高了操作效率。

算法深度优化

1. **混合查找策略**

通过概率控制（如 70%使用路径压缩）平衡了查找操作的即时开销和长期收益，既保留了路径压缩对后续查询的优化效果，又避免了每次查找都进行完整路径压缩带来的性能损耗，实现了整体性能的最优平衡。

```
int SmartFind(ElemType e) {
    if (rand() % 100 < 70)   // 70%概率使用路径压缩
        return CollapsingFind(e);
    else                             // 30%概率使用普通查找
        return Find(e);
}
```

通过概率控制平衡了路径压缩的开销和收益。

2. **自适应合并策略**

根据操作历史动态选择合并策略（查询多用按大小合并，合并多用按高度合并），使算法能自动适应不同的工作负载特征，无需人工干预即可在各种使用场景下保持稳定的性能表现。

```
void AdaptiveUnion(ElemType a, ElemType b) {
    if (queryCount > unionCount)   // 查询较多时
        WeightedUnion(a, b);          // 使用按大小合并
    else
        Union(a, b);                      // 使用按高度合并
```

```
}
```

该策略根据操作频次自动调整，提升整体性能。

应用场景扩展

针对社交网络特性，实现了以下扩展功能：

1. **朋友圈分析：**

   通过遍历根节点并统计各集合规模，提供朋友圈大小分布等群体特征分析，可识别关键用户群体和社区结构特征，为社交网络分析提供数据支持，时间复杂度与集合数量成正比。

```
void AnalyzeCircles() {
    map<int, int> sizeDistribution;
    for (int i = 0; i < size; ++i) {
        if (sets[i].parent < 0)  // 根节点
            sizeDistribution[-sets[i].parent]++;
    }
    // 输出朋友圈大小分布...
}
```

统计不同规模朋友圈的分布情况。

2. **关系推荐：**

   基于并查集的连通性检测，推荐同一朋友圈内尚未建立直接联系的用户，核心算法通过定位用户所在集合并筛选非直接好友实现，可扩展加入相似度计算等增强功能，为社交网络提供基础推荐能力。

```
vector<ElemType> RecommendFriends(ElemType user) {
    vector<ElemType> candidates;
    int root = Find(user);
    for (int i = 0; i < size; ++i) {
        if (Find(sets[i].data) == root && !IsFriend(user, sets[i].data))
            candidates.push_back(sets[i].data);
    }
    return candidates;
}
```

推荐同一朋友圈内尚未建立直接关系的用户。

五、收获与体会

[说明：每位组员撰写完成该项目后个人的收获和体会。**]**

程铭：通过解决朋友关系管理系统的问题，我深入理解了并查集在社交网络中的应用，学会了如何动态维护朋友圈数量。修改代码让我认识到精确计数的重要性，尤其是处理用户添加和关系合并时的逻辑严谨性。使用 circleCount 动态跟踪朋友圈数量，不仅解决了输出错误，还提升了系统实时性。这次实践让我体会到调试和优化的乐趣，同时增强了处理复杂数据结构的信心。未来，我会更注重代码的动态适应性，确保功能准确反映用户需求。

范书舰：在实现好友关系管理与用户删除功能时，我深刻体会到数据结构与算法协同设计的重要性。针对并查集无法动态断开的特性，采用"解散后重建"策略保障了关系解绑的正确性，同时通过双向链表维护好友关系确保数据一致性。同时，我优化了一部分代码，使用模块化设计重建函数提升了代码复用性，后续可探索更优的动态维护算法以提升系统效率。

罗溴予：在测试与介绍并查集算法的实现后，我意识到了这类代码容易发生错误的位置，同时在演讲后修正了自己对研讨内容的理解，对代码在实际生活中的应用有了更进一步的认识。

王子申：通过本次代码测试与修正，我深刻体会到算法优化与鲁棒性的重要性。动态扩容机制让我认识到内存管理的关键性，懒删除策略则展示了空间与时间的权衡艺术。在调试混合查找策略时，概率参数的微调让我理解了算法参数优化的精妙之处。最宝贵的是，测试过程中暴露的边界条件问题让我意识到完整测试用例的必要性，这比任何理论都更直观地教会了我如何编写健壮的代码。这些实践收获将深远影响我未来的编码习惯和工程思维。

六、组内互评

	范舒舰	程铭	罗溴予	王子申
范舒舰	/	100	100	100
程铭	100	/	100	100
罗溴予	100	100	/	100
王子申	100	100	100	/