**项目说明文档**

**数据结构课程设计**

**——关键活动**

### 作 者 姓 名 尹诚成

学 号 2351279

指 导 教 师 张 颖

学 院 专 业 计算机科学与技术学院 软件工程



二〇二三 年 十二 月 十三 日

**目录**

[1.项目分析 1](#_bookmark0)

* 1. .[项目背景分析 1](#_bookmark1)
  2. .[项目需求分析 1](#_bookmark2)
  3. .[项目功能分析 1](#_bookmark3)

1.3.1.项目任务统计功能 1

1.3.2.拓扑排序判断功能 1

1.3.3.关键活动计算功能 1

1.3.4.异常处理功能 2

[2.项目设计 2](#_bookmark12)

* 1. .[数据结构设计 2](#_bookmark13)
  2. .[结构体与类设计 2](#_bookmark14)

2.2.1 Edge结构体的设计 2

2.2.2.Vertex结构体设计 2

2.2.3.Graphlnk类的设计 3

2.2.4.ActivityNode类的设计 3

2.2.5.KeyActivities类的设计 3

* 1. .[项目主体架构设计 4](#_bookmark32)

[3.项目功能实现 5](#_bookmark33)

3.1.项目主体架构的实现 5

3.2.拓扑排序判断功能的实现 5

3.3.关键活动计算功能的实现 6

3.4.关键活动输出功能的实现 6

3.5.异常处理功能的实现 6

4.项目测试 7

4.1.输入任务交接点个数功能测试 7

4.2.输入任务数量功能测试 7

4.3.输入任务相关信息功能测试 8

4.4.拓扑排序判断功能测试 8

4.5.关键活动计算与输出功能测试 9

1. **项目分析**
   1. 项目背景分析

在现代工程项目管理中，合理的任务调度是确保项目按时完成的关键。任务调度不仅要考虑每个任务的开始和结束时间，还需要考虑任务之间的依赖关系。有些任务必须在其他任务完成后才能开始，而有些任务则可以并行执行。此外，不同的任务可能需要不同的时间来完成，且有些任务对于项目的整体进度至关重要，被称为“关键活动”。如果关键活动延误，整个项目的工期都会受到影响。

为了优化资源分配和确保项目按时完成，需要开发一个程序来判定给定的任务调度方案是否可行，并计算出完成整个项目所需的最短时间。同时，该程序还需要能够识别并输出所有的关键活动，以便项目管理者能够重点关注这些任务，确保项目的顺利进行。

* 1. 项目需求分析

基于以上背景分析，本项目需要实现需求如下：

(1)设计一个关键活动计算系统，能够输入不同任务的任务节点和工期；

(2)根据输入的任务调度信息，判断方案是否可行；若可行，则需计算出任务完成的最短时间，并识别出所有关键活动，以便管理者能集中关注这些任务；

(3)提供友好的用户界面，方便用户输入数据和查看结果；

(4)系统需要具备良好的稳定性和安全性，能够处理非法输入等异常情况。

* 1. 项目功能分析

本项目旨在开发一个任务调度程序，并考虑用户界面设计以实现一个高效的工程项目任务管理系统。下面对项目的功能进行详细分析。

* + 1. 项目任务统计功能

统计工程项目所有的任务调度，包括每个任务的起始节点、终止接点和任务所需的时间，并构建对应的网络。

* + 1. 拓扑排序判断功能

在关键活动计算中，拓扑排序判断功能通过确定任务的执行顺序和依赖关系，确保项目调度的合理性。这一功能能够有效识别出任务之间的前后关系，并为后续的早晚开始时间计算奠定基础。以下是拓扑排序的具体实现步骤及其重要性：

* + 1. 关键活动计算功能

关键活动计算功能专注于识别哪些任务对项目的完工时间至关重要，也即关键活动。该功能的实现步骤包括：

(1)最早开始/完成时间计算：采用拓扑排序等算法，计算每个任务的最早开始时间和最早完成时间，从而确定任务在项目进度中最早可以开始的时间点。

(2)最晚开始/完成时间计算：通过逆向遍历，计算每个任务的最晚开始时间和最晚完成时间，确保任务仍能在项目截止日期之前完成。

(3)关键活动判定：通过比较每个任务的最早开始时间和最晚开始时间，判断是否存在关键活动。若某个任务的最早开始时间与最晚开始时间相同，则该任务被认为是关键活动。

(4)输出关键活动及详细信息：将所有识别出的关键活动按优先顺序输出，并提供详细的任务信息，以便于项目管理者进行重点关注和资源分配。

* + 1. 异常处理功能

实现异常处理机制，处理用户可能输入的非法信息，确保系统的稳定性和安全性。

1. **项目设计**

2.1.数据结构设计

基于项目分析，为了实现任务调度和关键活动的计算，采用邻接表而不使用邻接矩阵的数据结构来表示图，原因如下：

(1)适应稀疏图场景：在任务调度的场景中，任务之间的依赖关系通常呈稀疏状态。使用邻接表可以有效地表示只有少量边（依赖关系）的图，而不需要为所有可能的连接分配额外空间。相比于邻接矩阵，邻接表在空间利用上更加高效。

(2)动态性和灵活性：邻接表允许在运行时动态地增加顶点和边，这对于任务调度系统来说是非常重要的。随着项目需求的变化，可能需要频繁地增加新的任务及其依赖关系，邻接表结构能更轻松地应对这些变化。

(3)方便的遍历操作：在执行诸如拓扑排序等算法时，邻接表提供了按邻接关系遍历的便利。每个节点只需跟踪其连接的边，能够快速访问相关的邻接顶点，有助于加速算法的执行。

2.2.结构体与类设计

2.2.1. Edge结构体的设计

Edge 结构体用于表示图中的边的信息。其数据成员，构造函数定义及含义如下：

int otherVertex：表示与当前边相连的另一个顶点。

E cost：边的权值，

Edge<T, E>\* link：指向下一个边的指针，形成链表结构，这是为了在一个顶点可以有多条边的情况下，能够有效地连接和遍历所有相关边。

Edge()：默认构造函数，初始化 otherVertex 为 -1，cost 为 0，link 为 NULL，保证在未分配或使用前，边的状态是安全的。

Edge(int num, E weight)：带参数的构造函数，允许在创建边的同时指定连接的顶点和权值，使得结构体在使用时更加灵活和方便。

2.2.2. Vertex结构体的设计

Vertex 结构体用于表示图中的顶点，其数据成员定义及含义如下：

T data：表示顶点的数据，用于标识或描述该顶点的一些信息，比如顶点的名称或编号。

Edge<T, E>\* firstEdge：指向第一个边的指针，用于连接到从该顶点出发的边。通过这个指针，可以快速访问与该顶点相关联的所有边，有助于在图的遍历中高效查找有效连接。

2.2.3.Graphlnk 类的设计

Graphlnk 类是一个用于表示图的基本结构，封装了图的顶点和边，并提供了一系列操作图的功能。该类的设计目的是为各种图算法提供必要的数据结构支持，通过封装图的基本操作，使得图的构建和管理变得简单而高效，易于与算法结合使用。它提供了一个清晰的接口来进行图的操作，确保在处理图数据时的安全性和效率。其数据成员、构造函数、析构函数、公有成员函数定义及含义如下：

Vertex<T, E> nodeTable\*：指向一个 Vertex 类型的数组，表示图中的所有顶点。

int numVertices：表示图中顶点的数量。

int numEdges：表示图中边的数量。

Graphlnk(int n)：构造函数，接受一个整数参数 n 表示顶点的数量。初始化 numVertices 和 numEdges，并动态分配 nodeTable 数组以存储图的所有顶点。每个顶点的初始状态会被设置，确保在使用前是安全的。

~Graphlnk()：析构函数，负责释放所有动态分配的元素，确保不发生内存泄漏。

int getFirstNeighbor(int v)：获取顶点 v 的第一个邻接点。如果存在，则返回该邻接点的编号。

int getNextNeighbor(int v1, int v2)：获取从顶点 v1 到顶点 v2 的下一个邻接点。

int getWeight(int v1, int v2)：返回顶点 v1 和 v2 之间边的权重。如果不存在这样的边，则返回 0。

bool insertVertex(T vertex)：在图中插入一个新顶点。

bool insertEdge(int v1, int v2, E weight)：在顶点 v1 和 v2 之间插入一条边，设置边的权重。

2.2.4.ActivityNode 结构体的设计

ActivityNode 结构体用于表示任务调度中单个任务的相关信息，其数据成员和运算符重载定义及含义如下：

int vertex1：表示任务开始的交接点编号。

int vertex2：表示任务完成的交接点编号。

int day：表示完成该任务所需的时间。

friend std::istream& operator >>(std::istream& in, ActivityNode& k)：重载输入运算符，输入开始和完成的交接点以及所需的时间。

friend std::ostream& operator <<(std::ostream& out, ActivityNode k)：重载输出运算符，以便于将任务信息以一种易读的格式输出，格式为 "vertex1->vertex2"，方便用户查看任务之间的关系。

2.2.5.KeyActivities 类的设计

KeyActivites 类继承自 Graphlnk<T, E>，专门用于管理和计算与关键活动相关的操作。通过设计  KeyActivites 类，本项目能够有效管理任务调度及其依赖关系，支持关键活动的识别和计算。同时强化了用户输入和输出的功能，提升了系统的友好性和可用性。其数据成员、构造函数、析构函数、公有成员函数定义及含义如下：

ActivityNode \*activity：动态数组，用于存储所有活动的信息。

ActivityNode \*keyActivity：动态数组，用于存储项目中的关键活动。

Int \*topologicalVertex：存储经过拓扑排序后的顶点序列，以便于后续的计算。

int keyActivityCount：记录关键活动的数量。

int totalTime：储存项目的总时间。

KeyActivites()：默认构造函数，初始化图的顶点和边的数目为0，其他指针初始化为NULL，确保在对象创建时不会引用未分配的内存。

KeyActivites(int N, int M)：带参数的构造函数，接受任务交接点数量（N）和任务数量（M），并根据这些参数动态分配必要的数组，准备接受任务信息。

~KeyActivites()：释放动态分配的内存，确保没有内存泄漏。

bool topologicalSort()：实现拓扑排序算法，以确定任务的执行顺序，判断调度是否可行，并在可行的情况下生成按顺序排列的顶点数组。

void criticalActivity()：根据拓扑排序的结果和活动的早晚开始时间，计算关键活动。

void outputCriticalActivity()：打印关键活动和整个项目需要的最短时间。

2.3.项目主体架构设计

项目主体架构设计为：

(1)创建任务调度图；

(2)输入任务交接点数量和任务数量；

(3)输入任务信息；

(4)进行拓扑排序判断任务调度是否可行；

(5)计算关键活动；

(6)输出关键活动；

(7)退出程序。

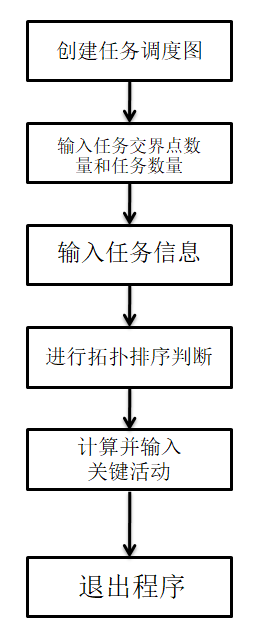


图 2.3 项目主体架构设计流程图

**3.项目功能实现**

3.1.项目主体架构的实现

项目主体架构实现思路为：

(1)创建任务调度图：在程序的开头，通过打印欢迎信息向用户介绍系统，为用户创建任务调度系统的过程提供指引；

(2)输入任务交接点数量和任务数量：使用inputInteger函数提示用户输入任务交界点数目和任务的数量，从而创建KeyActivites实例；

(3)输入任务信息：在构造函数中，输入每个单元活动的相关信息，包括起始和终止节点，完成任务所需的天数；

(4)进行拓扑排序判断任务调度是否可行：使用方法KeyActivity.topologicalSort判断当前任务调度是否可行。若可行，则计算关键活动并输出；若不可行，则输出提示信息并结束程序；

(5)计算关键活动：使用方法KeyActivity.criticalActivity来计算任务调度中哪些活动属于关键活动，并把关键活动信息存储在数组keyActivity中；

(6)输出关键活动：使用方法KeyActivity.outputCriticalActivity：来输出完成整个项目所需的最小天数以及所有关键活动的信息

(7)退出程序。

3.2.拓扑排序判断功能的实现

拓扑排序判断功能的函数为KeyActivities类的topologicalSort，拓扑排序判断功能的实现思路为：

(1)初始化数据结构：首先，定义一个栈 top 和一个计数数组 count，用于存储每个顶点的入度信息。top 指针用于跟踪当前顶点，而 count 数组则初始化为0，用于统计每个顶点的入度。

(2)统计入度：通过遍历图所有顶点，用 getFirstNeighbor 和 getNextNeighbor 方法来访问每个顶点的邻接点，并为每个邻接点增加入度计数。这一步骤确保了每个顶点的入度被正确记录，以便后续判断哪些顶点可以立即执行。

(3)将入度为0的顶点入栈：在入度统计完成后，遍历 count 数组，将所有入度为0的顶点推入栈中。入度为0的顶点表示这些任务不依赖于其他任务，可以立即进行处理。

(4)通过循环遍历图中的每个顶点，执行以下操作：

①检查栈 top 是否为空。如果栈为空，说明存在循环依赖或未处理的顶点，输出提示信息并返回不可行的结果。

②如果栈不为空，弹出栈顶元素（顶点），并将该顶点添加到 topologicalVertex 数组中，表示该顶点在拓扑排序中的位置。

③遍历当前顶点的所有邻接点，减少对应邻接点的入度计数。如果某个邻接点的入度减少后变为0，则将其推入栈中，继续处理。

(5)结束与资源释放：在完成所有顶点处理后，确认没有循环依赖，释放分配的内存，清理临时变量。

3.3.关键活动计算功能的实现

关键活动计算功能的函数为KeyActivities类的criticalActivity，关键活动计算功能的实现思路为：

(1)最早开始/完成时间计算：利用拓扑排序的结果，计算每个任务的最早开始时间和最早完成时间。初始化一个数组 eventEarlyTime 来存储每个任务的最早开始时间，所有任务的初始时间设为0。

对于每个顶点，通过 getFirstNeighbor 和 getNextNeighbor 方法访问其所有邻接的后继任务，根据边的权重（完成时间）更新后继任务的最早开始时间。如果从当前任务到后继任务的路径总时间超过了后继任务当前的最早时间，则进行更新。

(2)最晚开始/完成时间计算：基于完成时间，从项目的最后一个任务开始逆向遍历。初始化一个数组 eventLateTime，将所有任务的最晚开始时间初始设为最后任务的最早完成时间。

通过遍历所有任务的邻接前驱任务，计算每个任务的最晚开始时间。如果从后继任务到当前任务的路径时间超出当前任务的最晚完成时间，则更新当前任务的最晚开始时间。

(3)关键活动判定：在计算完每个任务的最早和最晚开始时间后，进行关键活动的判定。遍历所有活动，分别获取每个活动的早期和晚期完成时间。如果某个任务的最早开始时间与最晚开始时间相同，则这说明该任务无法延迟，成为关键活动。将符合条件的任务添加到 keyActivity 数组中。

将关键活动信息提供给项目管理者，以便他们对项目中的关键环节给予特别关注，做好资源分配和任务调度，以确保项目按时完成。

3.4.关键活动输出功能的实现

关键活动计算功能的函数为KeyActivities类的outputCriticalActivity，关键活动计算功能的实现思路为：

(1)初始化数据结构创建一个动态数组 activityStack，用于临时存储关键活动：使用一个整型变量 top 来管理栈的顶端。在栈初始化时，top 设置为 1，表示栈为空。

(2)输出项目的最短时间直接输出整个项目所需的总时间 totalTime。3. 输出关键活动

(3)关键活动的输出：在循环中遍历 keyActivity 数组，根据活动的排列情况决定是直接输出还是利用栈存输出。其中具体判断条件是检查当前活动和下一个活动的起始交接点编号：如果当前活动的起始点小于下一个活动的起始点，或者当前活动是最后一个活动，则输出当前活动。在输出活动之前，检查栈是否有活动存储，如果有，则逐一输出并将栈清空。

(4)使用栈存储活动:如果当前活动的起始点大于或等于下一个活动的起始点，则将当前活动入栈（即将其存入 activityStack），以待后续整理输出。

(5)清理动态内存:在完成输出后，使用 delete[] 释放之前动态分配的内存，以防止内存泄漏。

3.5.异常处理功能的实现

在进行KeyActivities类中私有数据成员等的动态内存申请时，程序使用new(std::nothrow)来尝试分配内存。new(std::nothrow)在分配内存失败时不会引发异常，而是返回一个空指针（NULL或nullptr），代码检查指针是否为空指针，如果为空指针，意味着内存分配失败，这时程序将执行以下操作：

(1)向标准错误流std::cerr输出一条错误消息"Error: Memory allocation failed."，指出内存分配失败；

(2)调用exit函数，返回错误码MEMORY\_ALLOCATION\_ERROR（通过宏定义方式定义为-1），用于指示内存分配错误，并导致程序退出。

**4.项目测试**

4.1.输入任务交接点个数功能测试

当输入合法时，程序继续运行。

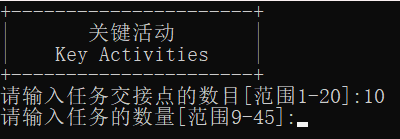


图 4.1.1 输入任务交接点个数功能测试（输入合法）

分别输入超过上下限的整数、浮点数、字符、字符串，可以验证程序对输入非法的情况进行了处理。

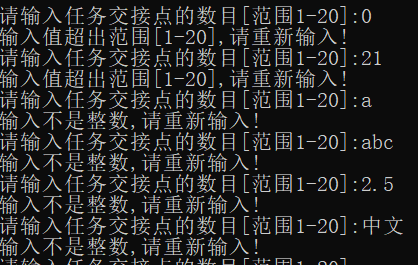


图 4.1.2 输入电网节点个数功能测试（输入非法）

4.2. 输入任务数量功能测试

分别输入超过上下限的整数、浮点数、字符、字符串，可以验证程序对输入非法的情况进行了处理。

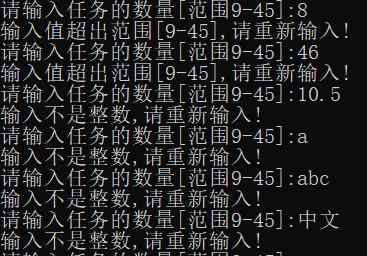


图 4.2.1 输入任务数量功能测试（输入非法）

当输入合法时，程序继续运行。

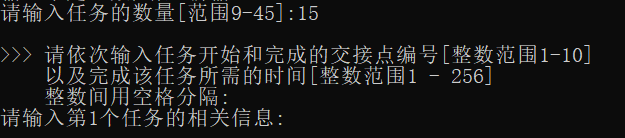


图 4.2.2 输入任务数量功能测试（输入合法）

4.3.输入任务相关信息功能测试

分别输入超过上下限的任务节点、任务完成天数、和两个节点相同的情况，可以验证程序对输入非法的情况进行了处理。

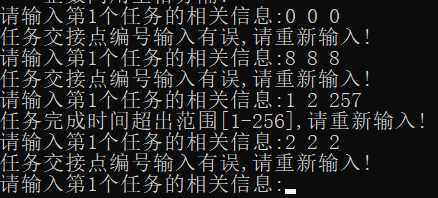


图 4.3.1 输入任务相关信息功能功能测试（输入非法）

当输入合法时，程序继续运行。

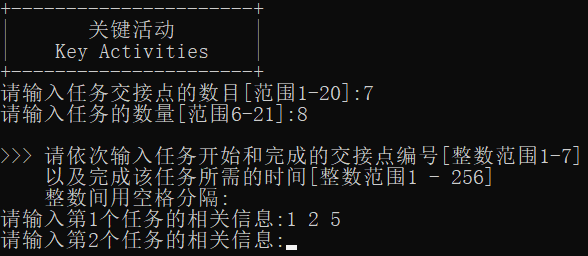


图 4.3.2 输入任务相关信息功能测试（起始节点为B）

4.4.拓扑排序判断功能测试

输入存在循环依赖的任务调度，可以验证程序对方案的可行性进行了验证。

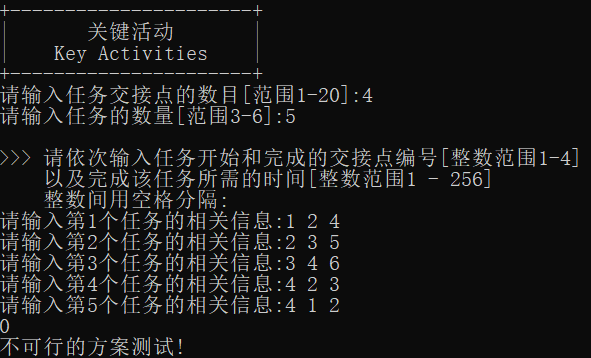


图 4.4 拓扑排序判断功能测试（起始节点为C）

4.5.关键活动计算与输出功能测试

输入存在关键活动的任务调度，可以验证程序能够正确计算完成任务所需的最小时间，并输出关键活动信息。

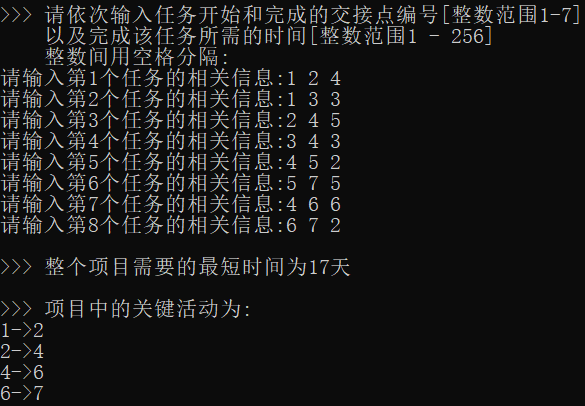


图 4.5.1 关键活动计算与输出功能测试（测试数据1）

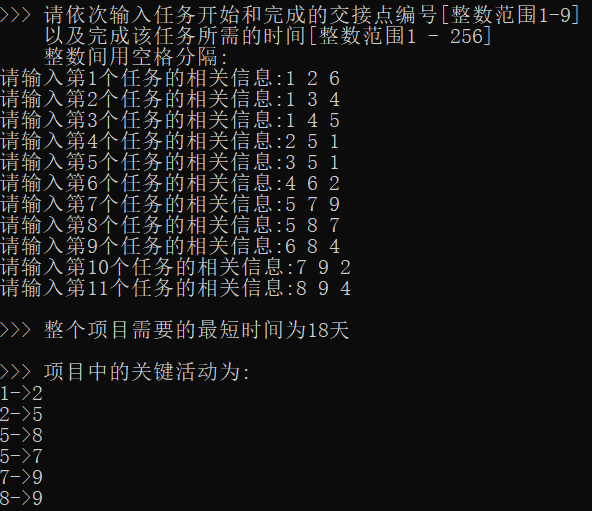


图 4.5.2 关键活动计算与输出功能测试（测试数据2）