



同濟大學
TONGJI UNIVERSITY

数据结构课程设计

项目说明文档

关键活动

姓 名： 马小龙

学 号： 2353814

学 院： 计算机科学与技术学院（软件学院）

专 业： 软件工程

指导教师： 张颖

二〇二四年十一月二十五日

目录

第 1 章 项目分析.....	1
1.1 项目背景分析.....	1
1.3 项目功能分析.....	1
1.3.1 关键活动寻找功能.....	1
1.3.2 异常处理功能.....	2
第 2 章 项目设计.....	3
2.1 数据结构设计.....	3
2.2 结构体与类设计.....	3
2.2.1 关键路径点（关键活动）设计.....	3
2.2.1.1 概述.....	3
2.2.1.2 结构体定义.....	3
2.2.1.3 数据成员.....	4
2.2.1.4 构造函数.....	4
2.2.2 有向图边（Edge）设计.....	4
2.2.2.1 概述.....	4
2.2.2.2 结构体定义.....	4
2.2.2.3 数据成员.....	4
2.2.2.4 构造函数.....	4
2.2.3 关键路径（CriticalPath）设计.....	4
2.2.3.1 概述.....	4
2.2.3.2 CriticalPath 类定义.....	5
2.2.3.3 数据成员.....	5
2.2.3.4 构造函数与析构函数.....	5
2.2.3.5 成员函数.....	5
2.2.4 有向图（Graph）设计.....	6
2.2.4.1 概述.....	6
2.2.4.2 Graph 类定义.....	6
2.2.4.3 构造函数与析构函数.....	6
2.2.4.4 私有数据成员.....	6
2.2.4.5 公有成员函数.....	7
2.3 项目框架设计.....	7
2.3.1 项目框架流程图.....	7
2.3.2 项目框架流程.....	8
第 3 章 项目功能实现.....	9
3.1 项目主体架构.....	9

3.1.1 实现思路.....	9
3.1.2 流程图.....	9
3.1.3 核心代码.....	9
3.1.4 示例.....	11
3.2 关键活动寻找功能.....	11
3.2.1 实现思路.....	11
3.2.2 流程图.....	12
3.2.3 核心代码.....	12
3.2.4 示例.....	13
3.3 异常处理功能.....	13
3.3.1 输入非法的异常处理.....	14
3.3.1.1 任务交接点数目与任务数量输入非法的异常处理.....	14
3.3.1.2 完成任务所需要的时间输入非法的异常处理.....	14
3.3.2 动态内存申请失败的异常处理.....	15
第4章 项目测试.....	17
4.1 输入功能测试.....	17
4.2 关键活动寻找功能.....	17
第5章 相关说明.....	19
5.1 编程语言.....	19
5.2 Windows 环境.....	19
5.3 Linux 环境.....	19

第 1 章 项目分析

1.1 项目背景分析

在项目管理和工程规划领域中，任务调度问题是一种典型的优化问题。随着项目规模的增大，任务之间的依赖关系变得复杂，如何在满足所有约束条件的同时，计算出完成整个项目所需的最短时间，是项目的核心目标之一。

关键路径法（Critical Path Method, CPM）是一种有效的工具，能够帮助分析项目任务的依赖关系，计算任务的最早开始时间和最晚完成时间，并识别出对项目整体工期起决定性作用的任务。这些关键任务的确定可以协助管理者合理分配资源，规避潜在的工期风险。

1.2 项目需求分析

需要开发一个程序，用于解决任务调度问题，计算一个给定工程项目的最短完成时间，并输出对工期有决定性影响的关键活动。

程序需要从用户输入的交接点数目和任务描述中，构建任务的依赖关系网络并判断调度方案的可行性。若方案可行，程序应输出项目的总工期和所有关键活动；若不可行，则输出 0。

技术需求包括支持 Windows 和 Linux 等操作系统，采用 C++ 等主流编程语言，采用合适的数据结构。

1.3 项目功能分析

本项目旨在开发一套用于解决任务调度相关问题的程序，以解决在任务调度过程中的各种挑战，计算出任务完成的最短时间。系统支持任务交接点数量、任务数量以及交接点间任务完成需要的时间，从而计算出关键活动。程序主要功能为关键活动寻找功能，以下为对功能的详细介绍：

1.3.1 关键活动寻找功能

程序能够依据输入的数据，借助 AOE 网络，即将图的边当做任务，顶点当任务交接点，以及拓扑排序和逆拓扑排序，计算出关键活动。

1.3.2 异常处理功能

程序对各种异常进行了基本处理，以提升系统稳定性。

第 2 章 项目设计

2.1 数据结构设计

本项目主要采用 AOE 网络作为主要数据结构，用来存储任务调度点以及任务、计算出关键活动，主要是基于以下方面：

1.适合描述任务间的依赖关系：在项目调度中，任务之间通常存在先后关系，某些任务必须在其他任务完成后才能开始。AOE 网络通过有向图中的边表示任务，节点表示任务的起始和终止调度点，这种结构能够清晰直观地描述任务的依赖关系。

2.便于进行拓扑排序与环检测：AOE 网络是有向无环图（DAG），可以通过拓扑排序快速验证任务依赖的可行性。若图中存在环，则说明调度方案存在循环依赖，任务无法按计划完成。

3.支持关键路径算法的核心计算：关键路径算法需要确定每个任务的最早开始时间（ES）和最晚完成时间（LF）。AOE 网络提供了完整的信息，通过对图的正向和反向遍历，可以高效地计算出这些时间参数，从而确定项目的最短工期和关键活动。

4.便于计算工期和优化资源配置：通过 AOE 网络中的权重（任务所需时间），可以直接计算出项目的总工期。同时，通过识别关键活动，管理者能够聚焦于对工期有直接影响的任务，合理配置资源，优化项目执行。

2.2 结构体与类设计

2.2.1 关键路径点（关键活动）设计

2.2.1.1 概述

Path 用于储存信息，包括当前路径点的起点与终点和下一个路径点的地址。

2.2.1.2 结构体定义

```
struct Path {  
    int start;  
    int end;  
    Path *next;
```

```
Path(int s = -1, int e = -1) : start(s), end(e) { next = nullptr; }
};
```

2.2.1.3 数据成员

int start; 存储当前路径点的起点

int end; 存储当前路径点的终点

Path *next; 存储下一个路径点的地址

2.2.1.4 构造函数

```
Path(int s = -1, int e = -1) : start(s), end(e) { next = nullptr; }
```

构造函数，根据输入的信息，初始化数据域与指针域。

2.2.2 有向图边（Edge）设计

2.2.2.1 概述

Path 用于储存信息，包括目标顶点、边的权重和指向下一条边的指针。

2.2.2.2 结构体定义

```
struct Edge
{
    int to;
    int weight;
    Edge *next;
    Edge(int t, int w, Edge *n = nullptr): to(t), weight(w), next(n) {}
};
```

2.2.2.3 数据成员

int to; 当前边的目标顶点

int weight; 当前边的权重

Edge *next; 指向下一条边的地址

2.2.2.4 构造函数

```
Edge(int t, int w, Edge *n = nullptr)
```

构造函数，根据输入的信息，初始化数据域与指针域

2.2.3 关键路径（CriticalPath）设计

2.2.3.1 概述

该通用模板类 `CriticalPath` 用于表示关键路径。用于存储路径是否生成成功、以及关键路径的总花费，同时通过链表的方式存储计算产生的关键活动，形成关键路径。该类主要提供插入函数，满足路径点的插入工作。

2.2.3.2 CriticalPath 类定义

```
class CriticalPath {
public:
    bool success;
    Path *paths;
    int path_count;
    int total_cost;
    CriticalPath();
    CriticalPath(const CriticalPath &other);
    ~CriticalPath();
    void CopyPaths(const Path *other_paths); // 深拷贝路径链表
    void ClearPaths(); // 清除所有路径节点
    void AddPath(int start, int end); // 添加路径
};
```

2.2.3.3 数据成员

`bool success;` 存储关键路径是否生成成功
`Path *paths;` 存储关键路径点的起点和终点对
`int path_count;` 关键路径点对数量
`int total_cost;` 存储实现该路径的总花费

2.2.3.4 构造函数与析构函数

`CriticalPath();`
 构造函数，初始化数据域和指针域。
`CriticalPath(const CriticalPath &other);`
 拷贝构造函数，实现将该类的对象作为返回值返回；
`~CriticalPath();`
 析构函数，释放内存资源。

2.2.3.5 成员函数

`void CopyPaths(const Path *other_paths);`

辅助拷贝构造函数，实现路径的深拷贝

```
void ClearPaths();
```

辅助析构函数，实现清除所有路径节点内存

```
void AddPath(int start, int end);
```

向关键路径中添加路径点，即关键活动

2.2.4 有向图（Graph）设计

2.2.4.1 概述

存储 AOE 网络的数据结构，提供拓扑排序和关键活动计算功能。

2.2.4.2 Graph 类定义

```
class Graph {
private:
    int vertices;
    Edge **adj_list;
    int *in_degree;
    int *earliest;
    int *latest;
    int *topo_order;
    int top_count;
public:
    Graph(const int v); // 构造函数
    ~Graph(); // 析构函数
    void AddEdge(int from, int to, int weight); // 添加边
    bool TopologicalSort(); // 拓扑排序
    CriticalPath FindCriticalPath(); // 计算关键路径
};
```

2.2.4.3 构造函数与析构函数

```
Graph(const int v);
```

构造函数，初始化数据成员，为数组分配动态内存

```
~Graph();
```

析构函数，释放数组、邻接表所占内存，防止内存泄漏

2.2.4.4 私有数据成员

int vertices;顶点数

Edge **adj_list;邻接表，用于存储有向边关系

int *in_degree; 存储每个顶点的入度

int *earliest; 存储最早开始时间

int *latest; 存储最晚开始时间

int *topo_order;存储拓扑排序结果

int top_count;记录当前拓扑排序顶点数量

2.2.4.5 公有成员函数

void AddEdge(int from, int to, int weight);

向有向图中添加边，完善有向图

bool TopologicalSort();

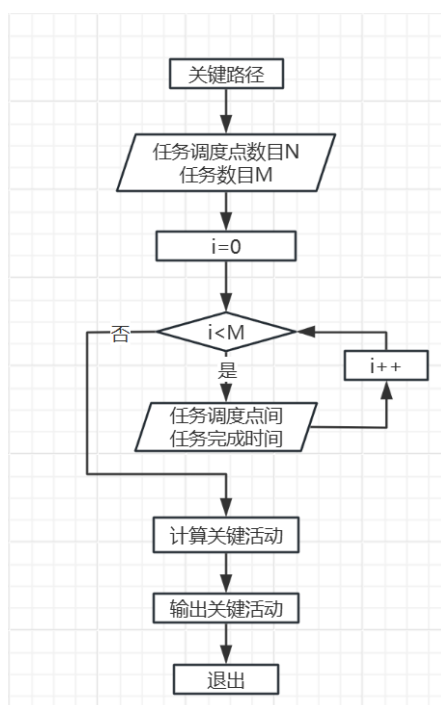
对当前有向图中的顶点进行拓扑排序

CriticalPath FindCriticalPath();

计算关键活动，得到关键路径

2.3 项目框架设计

2.3.1 项目框架流程图



2.3.2 项目框架流程

- 1.进入关键活动项目。
- 2.输入任务交接点的数目 N 以及任务数量 M 。
- 3.根据任务交界点数目和任务数量，添加任务调度点之间的任务的完成时间；
- 4.计算出关键活动以及项目完成总花费时间
- 5.输出总花费时间以及关键活动。
- 6.退出程序。

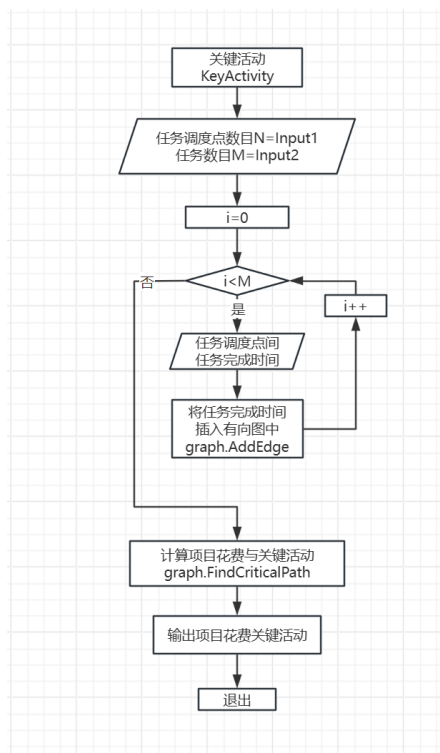
第3章 项目功能实现

3.1 项目主体架构

3.1.1 实现思路

- 1.进入 KeyActivity 函数以进入关键活动系统。
- 2.调用 Input1 函数分别输入任务交接点数目 N、任务数量 M，据此初始化有向图。
- 3.根据任务数量，调用 Input2 函数输入两个任务交接点任务完成时间，并插入有向图中。
- 4.调用 graph.FindCriticalPath()函数，计算项目总花费与关键活动。
- 6.输出完成项目总花费与关键活动；
- 5.退出系统。

3.1.2 流程图



3.1.3 核心代码

```

void KeyActivity()
{
    std::cout << "+-----+\n";
    std::cout << "|      关键活动      |\n";
    std::cout << "|      Key Activity   |\n";
    std::cout << "+-----+\n";
    bool **have_visited;
    const int N = Input1(true, "请输入任务交接点数目[2-100]:\n");
    const int M = Input1(false, "请输入任务数量:\n");
    Graph graph(N);
    have_visited=new(std::nothrow) bool*[N+1];
    assert(have_visited);
    for (int i=0;i<N+1;i++) {
        have_visited[i]=new bool[N+1];
        assert(have_visited[i]);
    }
    for (int i=0;i<N+1;i++) {
        for (int j=0;j<N+1;j++) {
            have_visited[i][j]=false;
        }
    }
    std::cout<<"请输入任务开始和完成设计的交接点编号以及完成该任务
所需要的时间: \n";
    for(int i = 0 ; i < M ; i++) {
        int src,dst,value;
        Input2(N,i ,src,dst, value);
        if (have_visited[src][dst]) {
            std::cout<<"任务调度点"<<src<<"与"<<dst<<"之间已经有任务!
";

            i--;
            continue;
        }
        have_visited[src][dst]=true;
        graph.AddEdge(src,dst,value);
    }
}

```

```

    }
    CriticalPath result=graph.FindCriticalPath();
    std::cout << "关键路径总花费: ";
    if(!result.success) {
        std::cout<<'0'<<" \n";
        return;
    }
    std::cout<<result.total_cost << "\n";
    for (Path *path = result.paths; path!= nullptr; path = path->next)
        std::cout << path->start << "->" << path->end << "\n";
    for (int i=0;i<N+1;i++)
        delete[] have_visited[i];
    delete [] have_visited;
}

```

3.1.4 示例

```

关键活动
Key Activity
请输入任务交接点数目[2-100]:
9
请输入任务数量:
11
请输入任务开始和完成设计的交接点编号以及完成该任务所需要的时间:
1 2 6
1 3 4
1 4 5
2 5 1
3 5 1
4 6 2
5 7 9
5 8 7
6 8 4
7 9 2
8 9 4
关键路径总花费: 18
1->2
2->5
5->8
5->7
7->9
8->9

```

3.2 关键活动寻找功能

3.2.1 实现思路

关键活动寻找功能的函数为 `CriticalPath Graph::FindCriticalPath()`，实现的思路为：

1. 调用 `TopologicalSort()` 方法对图的顶点进行拓扑排序，若排序成功，继续进行关键路径的计算；排序失败失败，则打印错误信息并返回空的 `CriticalPath` 对象。环路的存在意味着任务调度方案不可行。

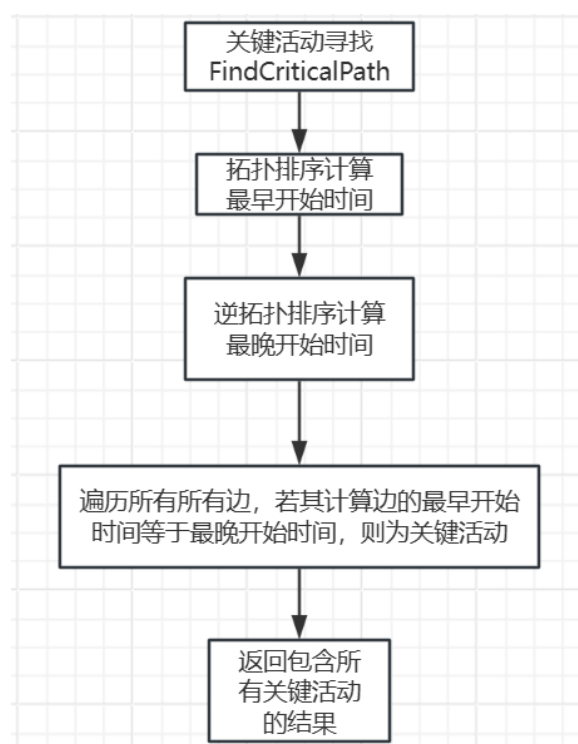
2. 使用 `memset` 将所有顶点的最晚开始时间 `latest` 初始化为无穷大，将最后一个顶点的最晚开始时间设置为其最早开始时间 `earliest`，并用其值更新项目的总工期 `result.total_cost`。

3. 按拓扑排序的逆序遍历顶点集合，从最后一个顶点开始向前计算。对每个顶点的所有出边，计算通过该边到达的终点的最晚开始时间。更新当前顶点的最晚开始时间，使其等于所有邻接顶点最晚开始时间减去边权重中的最小值。

4. 遍历所有边，对每条边，计算边的最早开始时间 (e) 和最晚开始时间 (l)，其中， $e = \text{earliest}[i]$ (起点的最早开始时间)， $l = \text{latest}[\text{neighbor}] - \text{edge} \rightarrow \text{weight}$ (终点的最晚开始时间减去边的权重)。若 $e == l$ ，说明该边是关键活动，加入结果对象 `result` 的路径列表。

5. 如果至少存在一条关键路径 (`result.paths != nullptr`)，将 `result.success` 标记为 `true`，返回包含关键路径信息的 `result` 对象。

3.2.2 流程图



3.2.3 核心代码

```

inline CriticalPath Graph::FindCriticalPath()
{

```

```

CriticalPath result;
if (!TopologicalSort()) {
    std::cout << "有向图中存在环路，构建拓扑排序失败！" << std::endl;
    return result; // 如果拓扑排序失败，返回空结果
}
memset(latest, 0x3f, sizeof(int) * vertices);
latest[topo_order[top_count - 1]] = earliest[topo_order[top_count - 1]];
result.total_cost = earliest[topo_order[top_count - 1]];
for (int i = top_count - 1; i >= 0; i--) {
    int curr = topo_order[i];
    for (Edge *edge = adj_list[curr]; edge != nullptr; edge = edge->next) {
        int neighbor = edge->to;
        latest[curr] = std::min(latest[curr], latest[neighbor] - edge->weight);
    }
}
for (int i = 0; i < vertices; i++) {
    for (Edge *edge = adj_list[i]; edge != nullptr; edge = edge->next) {
        int neighbor = edge->to;
        int e = earliest[i];
        int l = latest[neighbor] - edge->weight;
        if (e == l) {
            result.AddPath(i + 1, neighbor + 1);
        }
    }
}
if (result.paths != nullptr)
    result.success = true;
return result;
}

```

3.2.4 示例

示例同 3.1.4

3.3 异常处理功能

3.3.1 输入非法的异常处理

3.3.1.1 任务交接点数目与任务数量输入非法的异常处理

在输入任务交接点与任务数目时，需要确保输入的值均为大于 0 的正整数，其中任务交接点数目还应该在 2~100 范围之内；通过一个 while 无限循环来确保输入无误，循环中输入结束后，判断输入的数值是否正确且在规定范围之内（根据参数不同限定不同范围），若输入无误则退出循环，进行下一步操作，否则清除当前输入状态和缓冲区，继续执行循环。具体代码如下：

```
int Input1(const bool ret,const char* prompt)
{
    double temp;
    while (true) {
        std::cout << prompt;
        std::cin >> temp;
        if (std::cin.fail() ||temp != static_cast<int>(temp)||ret&& (temp <= 1 ||
temp > 100)||temp<=0) {
            std::cout << "输入非法，请重新输入！ ";
            Clear();
            continue;
        }
        Clear();
        break;
    }
    return static_cast<int>(temp);
}
```

3.3.1.2 完成任务所需要的时间输入非法的异常处理

完成任务所需时间需要输入三个参数：任务开始和完成设计的交接点编号以及完成该任务所需要的时间。由于三个参数均为正整数，设置一个三位数组，通过一个 for 循环来简化逻辑；

进入 for 循环，当用户输入的数合法且在一定范围时（交接点需要不大于 N），若输入无误则退出循环，输入下一个参数或进行下一步操作，否则清除当前输入状态和缓冲区，重新从第一个参数开始输入这三个参数，具体实现代码如下：

```
void Input2(const int N,int row,int&src,int&dst,int& value)
{
```

```

double temp[3];
for (int i=0;i<3;i++) {
    std::cin >> temp[i];
    if (std::cin.fail() || temp[i] <= 0 || temp[i] != static_cast<int>(temp[i]) ||
i != 2 && temp[i] > N) {
        std::cout << "第" << row + 1 << "行输入错误，请从第" << row +
1 << "行开始重新输入！";
        Clear();
        i=-1;
    }
}
src = static_cast<int>(temp[0]);
dst=static_cast<int>(temp[1]);
value=static_cast<int>(temp[2]);
}

```

3.3.2 动态内存申请失败的异常处理

在进行 KeyActivity()函数、Graph 类与 CriticalPath 类的动态内存申请时，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，对于内存分配失败，采用断言抛出异常，例如如下代码：

```

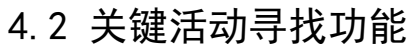
inline Graph::Graph(const int v) : vertices(v), top_count(0)
{
    adj_list = new(std::nothrow) Edge *[v];
    assert(adj_list != nullptr);
    in_degree = new(std::nothrow) int[v];
    assert(in_degree != nullptr);
    earliest = new(std::nothrow) int[v];
    assert(earliest != nullptr);
    latest = new(std::nothrow) int[v];
    assert(latest != nullptr);
    topo_order = new(std::nothrow) int[v];
    assert(topo_order != nullptr);
}

```

```
memset(adj_list, 0, sizeof(Edge *) * v);  
memset(in_degree, 0, sizeof(int) * v);  
memset(earliest, 0, sizeof(int) * v);  
memset(latest, 0x3f, sizeof(int) * v);  
}
```

4.1 输入功能测试测试

同时,若输入的完成设计的交接点序列(有序)编号重复,也需要重新输入。



17

```

+-----+
|           |
|  关键活动  |
|  Key Activity  |
|           |
+-----+
请输入任务交接点数目[2-100]:
7
请输入任务数量:
8
请输入任务开始和完成设计的交接点编号以及完成该任务所需要的时间:
1 2 4
1 3 3
2 4 5
3 4 3
4 5 2
4 6 6
5 7 5
6 7 2
关键路径总花费: 17
1->2
2->4
4->6
6->7

进程已结束, 退出代码为 0

```

```

+-----+
|           |
|  关键活动  |
|  Key Activity  |
|           |
+-----+
请输入任务交接点数目[2-100]:
9
请输入任务数量:
11
请输入任务开始和完成设计的交接点编号以及完成该任务所需要的时间:
1 2 6
1 3 4
1 4 5
2 5 1
3 5 1
4 6 2
5 7 9
5 8 7
6 8 4
7 9 2
8 9 4
关键路径总花费: 18
1->2
2->5
5->8
5->7
7->9
8->9

```

```

D:\Programme\DataStruct\key_activity\cmake-build-debug\key_activ
+-----+
|           |
|  关键活动  |
|  Key Activity  |
|           |
+-----+
请输入任务交接点数目[2-100]:
4
请输入任务数量:
5
请输入任务开始和完成设计的交接点编号以及完成该任务所需要的时间:
1 2 4
2 3 5
3 4 6
4 2 3
4 1 2
有向图中存在环路, 构建拓扑排序失败!
关键路径总花费: 0

```

第 5 章 相关说明

5.1 编程语言

本项目全部.cpp 文件以及.h 文件均使用 C++编译完成，使用 UTF-8 编码。

5.2 Windows 环境

Windows 系统：Windows 11 x64

Windows 集成开发环境：CLion 2024

工具集：MinGW 11.0 w64

5.3 Linux 环境

基于 Linux 内核的操作系统发行版：Ubuntu 24.04.1

Linux 命令编译过程为：

1. 定位项目所在文件夹，包括 .pp 与 .h 文件；具体命令为：cd /home/bruce/programe/ key_activity

2.编译项目，生成可执行文件；具体命令为: g++ -static -o key_activity_linux key_activity.cpp my_critical_path.h;

其中指令含义分别为：

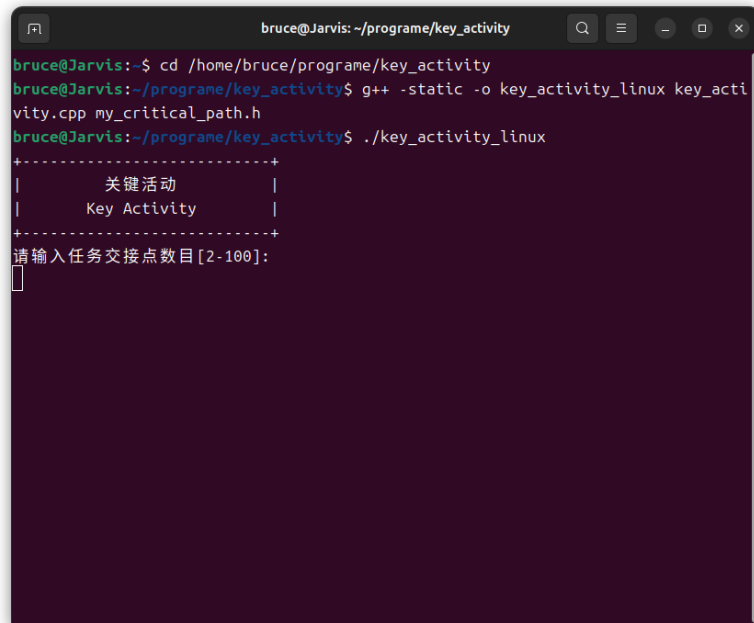
g++: 调用 GNU 的 C++编译器

-static: 使用静态链接而非动态链接，将所有依赖库直接嵌入到可执行文件，文件存储空间变大，但可以单独运行

-o key_activity_linux: -o 表示输出文件选项，key_activity_linux 为可执行文件名

key_activity.cpp my_critical_path.h:编译所需要的文件。

3.运行可执行文件；具体命令为 ./ key_activity_linux



```
bruce@Jarvis: ~/programe/key_activity
bruce@Jarvis:~/programe/key_activity$ g++ -static -o key_activity_linux key_acti
vity.cpp my_critical_path.h
bruce@Jarvis:~/programe/key_activity$ ./key_activity_linux
+-----+
|      关键活动      |
|      Key Activity  |
+-----+
请输入任务交接点数目[2-100]:
█
```