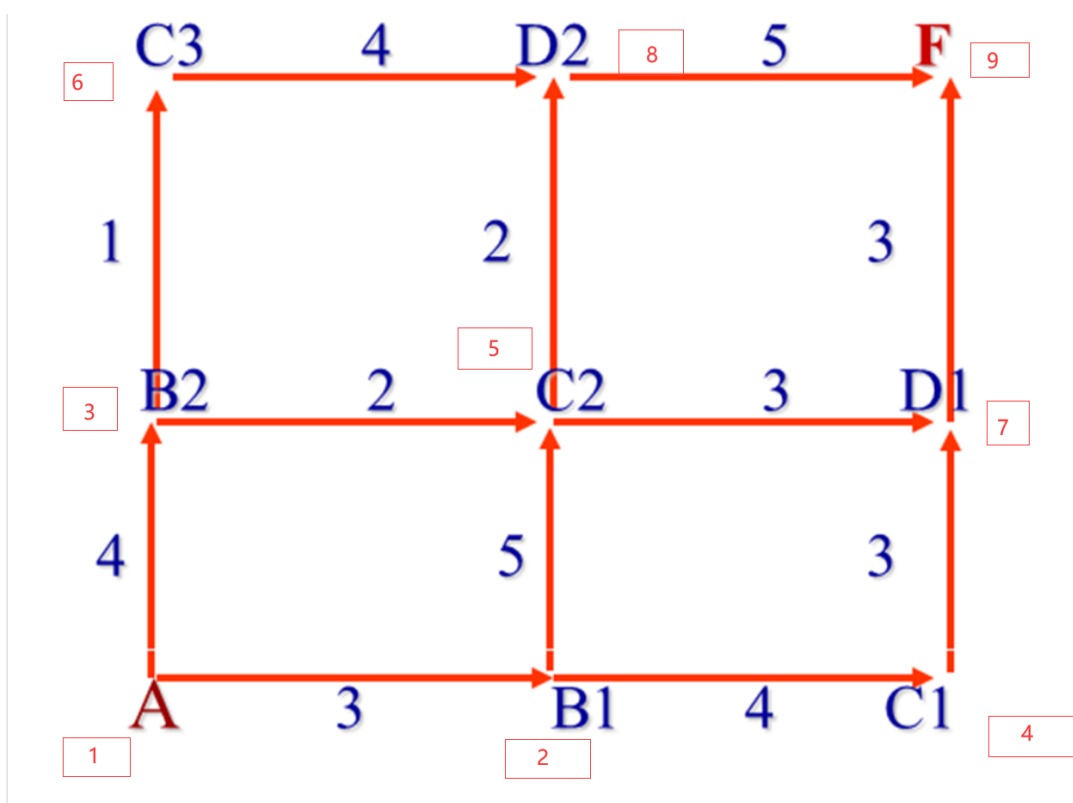


W先生每天驾车去公司上班。W先生的住所位于A，公司位于F，图中的直线段代表公路，交叉点代表路口，直线段上的数字代表两路口之间的平均行驶时间。现在W先生的问题是要确定一条最省时的上班路线。



## 问题分析

作为一个有向图的带权最短路径问题，我们可以使用迪杰斯特拉算法求得两个点之间的最短路径。

## 代码实现

```
#include <iostream>
#include <vector>
#include <queue>
#include <climits>
#include <stack>

using namespace std;

// 边的结构体
struct Edge {
    int number;
    int dest;
    int weight;
```

```

};

// 图的结构体
struct Graph {
    int numVertices;
    vector<vector<Edge>> adjList;
};

// Dijkstra算法求解最短路径
pair<int, stack<int>> dijkstra(const Graph& graph, int startVertex, int
endVertex) {
    // 初始化距离数组和路径数组
    vector<int> dist(graph.numVertices, INT_MAX);
    vector<int> prev(graph.numVertices, -1);
    dist[startVertex] = 0;

    // 创建最小堆，用于选择距离最小的顶点
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int,
int>>> minHeap;
    minHeap.push(make_pair(0, startVertex));

    while (!minHeap.empty()) {
        int currVertex = minHeap.top().second;
        int currDist = minHeap.top().first;
        minHeap.pop();

        // 如果到达结束顶点，则提前结束算法
        if (currVertex == endVertex) {
            break;
        }

        // 检查当前顶点的所有邻居
        for (const auto& edge : graph.adjList[currVertex]) {
            int neighbor = edge.dest;
            int weight = edge.weight;

            // 如果通过当前顶点到达邻居的路径更短，则更新距离和路径
            if (currDist + weight < dist[neighbor]) {
                dist[neighbor] = currDist + weight;
                prev[neighbor] = currVertex;
                minHeap.push(make_pair(dist[neighbor], neighbor));
            }
        }
    }

    // 构建最短路径
    stack<int> path;
    int currVertex = endVertex;
    while (currVertex != -1) {
        path.push(currVertex);
        currVertex = prev[currVertex];
    }

    // 返回最短路径距离和路径
    return make_pair(dist[endVertex], path);
}

int main() {

```

```

// 创建图的示例
Graph graph;
graph.numVertices = 10; // 顶点数量
cout << "请输入边的数量" << endl;
int numLine;
cin >> numLine;
// 初始化邻接表
graph.adjList.resize(graph.numVertices);
for (int i = 1; i < numLine+1; i++) {
    int number;
    int dest;
    int weight;
    cout << "请输入结点号, 指向, 权值" << endl;
    cin >> number >> dest >> weight;
    graph.adjList[number].push_back({ number, dest, weight });
}

int startVertex = 1; // 起始顶点
int endVertex = 0; // 结束顶点

cout << "请输入结束顶点: " << endl;
cin >> endVertex;

// 调用Dijkstra算法求解最短路径
pair<int, stack<int>> result = dijkstra(graph, startVertex, endVertex);
int shortestDistance = result.first;
stack<int>& shortestPath = result.second;

// 打印结果
cout << "最短路径距离: " << shortestDistance << endl;
cout << "最短路径: " << endl;
while (!shortestPath.empty()) {
    cout << shortestPath.top() << " ";
    shortestPath.pop();
}
cout << endl;

return 0;
}

```

## 结果展示

---

```
请输入边的数量
12
请输入结点号，指向，权值
1 2 3
请输入结点号，指向，权值
1 3 4
请输入结点号，指向，权值
2 4 4
请输入结点号，指向，权值
2 5 5
请输入结点号，指向，权值
3 5 2
请输入结点号，指向，权值
3 6 1
请输入结点号，指向，权值
4 7 3
请输入结点号，指向，权值
5 7 3
请输入结点号，指向，权值
5 8 2
请输入结点号，指向，权值
6 8 4
请输入结点号，指向，权值
7 9 3
请输入结点号，指向，权值
8 9 5
请输入结束顶点：
9
最短路径距离：12
最短路径：
1 3 5 7 9
```

**在芯片设计中，经常要考虑输入与输出端之间的连接关系，假设输入与输出端分别有若干端口，并且用一组数字进行标注（可能有重复），当且仅当输入端与输出端数字相等并且不与其他连接线相交的情况下，可以建立输入与输出之间的连接。设计算法策略，计算可能得到的最大连接数。**

## 问题分析

1. 创建一个二分图，其中左侧顶点表示输入端口，右侧顶点表示输出端口。每个顶点都用数字进行标注。
2. 对于每对输入和输出端口，如果它们的数字标注相等且它们之间没有相交的连接线，则在二分图中添加一条边连接它们。
3. 使用匈牙利算法找到二分图中的最大匹配。最大匹配即为可能得到的最大连接数。

我们首先构建了一个二分图，根据输入和输出端口的数字标注进行连接。然后，使用匈牙利算法求解二分图的最大匹配，最大匹配的数量即为可能得到的最大连接数。在示例中，我们给出了一组输入端口和输出端口的数字标注，通过调用函数来计算可能得到的最大连接数，并将结果打印输出。

## 代码实现

```
#include <iostream>
#include <vector>
#include <cstring>

using namespace std;

// 匈牙利算法求解最大匹配
bool dfs(int u, const vector<vector<int>>& graph, vector<int>& match,
vector<bool>& visited) {
    int n = graph.size();
    for (int v = 0; v < n; v++) {
        if (graph[u][v] && !visited[v]) {
            visited[v] = true;
            if (match[v] == -1 || dfs(match[v], graph, match, visited)) {
                match[v] = u;
                return true;
            }
        }
    }
    return false;
}

int maxMatching(const vector<vector<int>>& graph) {
    int n = graph.size();
    vector<int> match(n, -1);
    int count = 0;

    for (int u = 0; u < n; u++) {
        vector<bool> visited(n, false);
        if (dfs(u, graph, match, visited)) {
            count++;
        }
    }

    return count;
}

// 计算可能得到的最大连接数
int calculateMaxConnections(const vector<int>& inputs, const vector<int>&
outputs) {
    int n = inputs.size();
    int m = outputs.size();

    // 构建二分图
    vector<vector<int>> graph(n, vector<int>(m, 0));
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (inputs[i] == outputs[j]) {
                graph[i][j] = 1;
            }
        }
    }
}
```

```

    }
}

// 使用匈牙利算法求解最大匹配
int maxConnections = maxMatching(graph);

return maxConnections;
}

int main() {
    // 输入端口的数字标注
    vector<int> inputs = {1, 2, 3, 4, 5};

    // 输出端口的数字标注
    vector<int> outputs = {2, 3, 4, 5, 6};

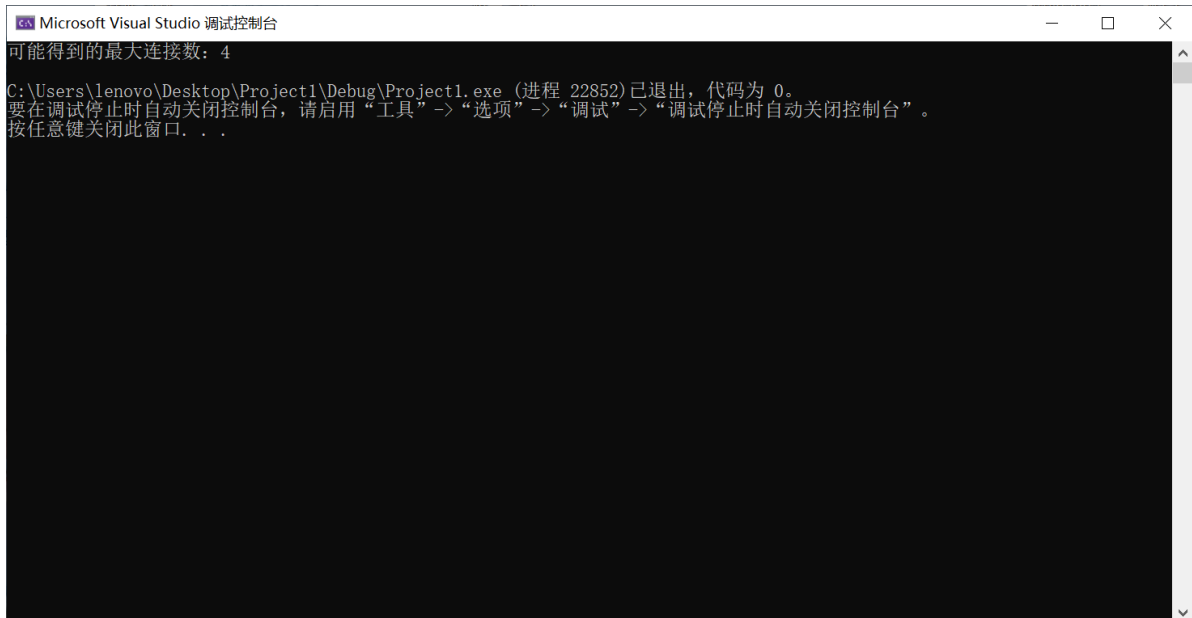
    // 计算可能得到的最大连接数
    int maxConnections = calculateMaxConnections(inputs, outputs);

    cout << "可能得到的最大连接数: " << maxConnections << endl;

    return 0;
}

```

## 结果展示



The screenshot shows the Microsoft Visual Studio Debug Console window. The title bar reads "Microsoft Visual Studio 调试控制台". The output text is as follows:

```

可能得到的最大连接数: 4
C:\Users\lenovo\Desktop\Project1\Debug\Project1.exe (进程 22852) 已退出, 代码为 0。
要在调试停止时自动关闭控制台, 请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口。 . . .

```