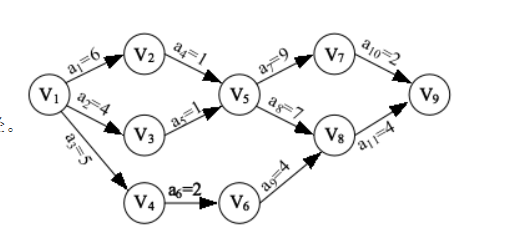
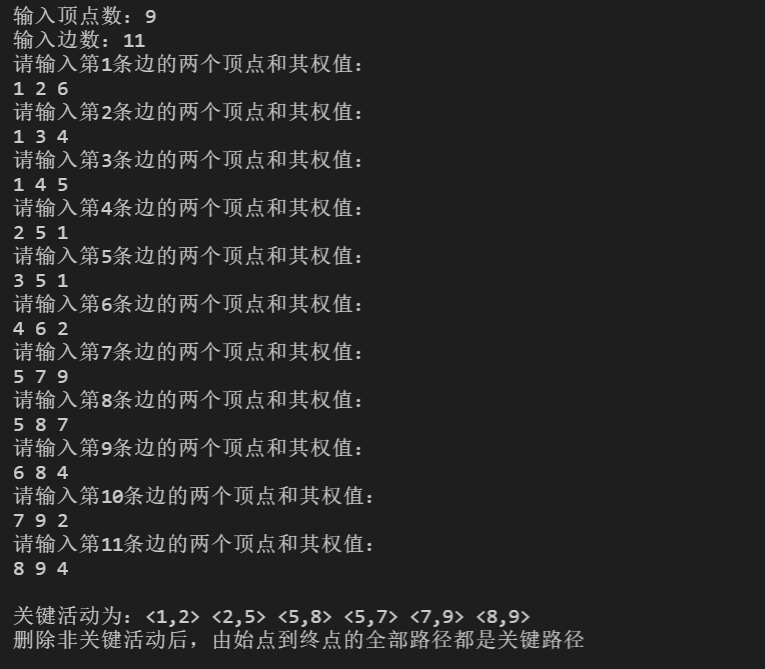
**测试用例**





**算法和数据结构设计**

1. 建立网。利用邻接表，每次输入边相邻的两个结点和其权值，hnode记录当前结点的信息，node记录和这个结点邻接的所有结点的信息
2. 寻找关键路径。vi 表示能够发生的最早时间，e(i)表示活动 ai 的最早开始时间，l(i)表示最迟开始时间，关键活动为e(i) = l(i)的活动 。

先计算e(i)与l(i)，Ve(j)表示事件 j 的最早发生时间；Vl(j)表示事件 j 的最迟发生时间，因为e(i)=Ve(j)，所以先计算Ve(j)，又因为l(i) = Vl(k) - dut(<j , k>)，所以先计算Vl，即先算Ve(j)和Vl。

1. 对Ve的计算。从Ve(1)=0开始向前递推求Ve(j)，Ve初始均置为0，利用拓扑排序求出Ve，在建立了拓扑正向的栈同时建立逆向的栈，当拓扑出栈时，逆拓扑进栈
2. 对Vl的计算。从Vl(n)=Ve(n)起向后递推求Vl(i)（即从最后开始求）。Vl初始均置为Ve(n)，通过逆向栈向前。
3. 计算e(i)与l(i)。通过得到的Ve和Vl，e = Ve，l(i) = Vl(k) – dut(<j , k>)。
4. 对e和l比较，若两者相同，输出这两个结点间的边。删除这些后，剩下的为关键路径

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define LEN sizeof(node)

typedef struct Node{

    int vex;           //邻接结点

    int dut;           //边的权值

    struct Node \*next; //指向其他邻接结点

}node;

typedef struct Hnode{

    int index;         //入度

    int data;          //顶点

    struct Node \*link;  //指向所有邻接结点

}hnode;

void create(hnode h[], int n, int e); //创建有向图

void CMP(hnode h[], int n);           //求关键路径

int main() {

    int n, e;  //n 为顶点数, e 为总边数

    hnode h[20];

    printf("输入顶点数：");

    scanf("%d", &n);

    printf("输入边数：");

    scanf("%d", &e);

    create(h, n, e);

    CMP(h, n);

    return 0;

}

void create(hnode h[], int n, int e) {   //邻接表存储

    int i, v1, v2, l;

    node \*p;

    for (i = 1; i <= n; i++) {

        h[i].data = i;

        h[i].index = 0;

        h[i].link = NULL;

    }

    for (i = 1; i <= e; i++) {

        printf("请输入第%d条边的两个顶点和其权值：\n", i);

        scanf("%d %d %d", &v1, &v2, &l);

        p = (node\*)malloc(sizeof(node));

        p->vex = v2;

        p->dut = l;

        p->next = h[v1].link;

        h[v1].link = p;

        h[v2].index++;

    }

  }

void CMP(hnode h[], int n) {

    int i, j, m, k;

    int endnode, btop = 0, ftop = 0; // btop为逆拓扑栈，ftop为拓扑栈

    int Ve[n], Vl[n];   // Ve[n] 时间最早发生时间, Vl[n] 是最迟发生时间

    node \*p;

    for (i = 1; i <= n; i++) {   //寻找入度为0结点

        if (h[i].index == 0) {

            h[i].index = ftop;

            ftop = i;

        }

        Ve[i] = 0; //初始化，置零

    }

    m = 0; //拓扑排序输出顶点计数器

    while (ftop != 0) {

        j = ftop;

        ftop = h[ftop].index; //拓扑退栈

        h[j].index = btop;    //逆拓扑入栈

        btop = j;

        m = m + 1;

        p = h[j].link;

        while (p != NULL) { //修改拓扑出栈后相关顶点的入度

            k = p->vex;

            h[k].index--;

            if (h[k].index == 0) {

                h[k].index = ftop;

                ftop = k;

            }

            if (Ve[k] < Ve[j] + p->dut) {

                Ve[k] = Ve[j] + p->dut;

            }

            p = p->next;

        }

    }

    if (m < n) {

        printf("工程图有环\n");

        exit(0);

    }

    endnode = btop;  // endnode为终点元素编号

    Vl[endnode] = Ve[endnode];

    while (btop != 0) { //求Vl

        j = btop;

        btop = h[btop].index;

        Vl[j] = Ve[endnode]; //初始化最大值

        p = h[j].link;

        while (p != NULL) {

            k = p->vex;

            if (Vl[j] > Vl[k] - (p->dut)) {

                Vl[j] = Vl[k] - (p->dut);

            }

            p = p->next;

        }

    }

    printf("\n关键活动为：");

    for (k = 1; k <= n; k++) {   //e[i]即Ve[i], l[i] = Vl[k] - dut<j , k>

        p = h[k].link;

        while (p != NULL) {

            if (Ve[k] == Vl[p->vex] - (p->dut)) { // e[i] = Ve[k],l[i]=Vl[p]-dut<k,p>

                printf("<%d,%d> ", k, p->vex);

            }

            p = p->next;

        }

    }

    printf("\n删除非关键活动后，由始点到终点的全部路径都是关键路径\n");

}