(1) Result screenshot

```
Jamei@DESKTOP-PGELV1B MINGW64 ~/Desktop
$ ./hw<input0_windows.txt>output.txt

Jamei@DESKTOP-PGELV1B MINGW64 ~/Desktop
$ diff -c output.txt output0_windows.txt

Jamei@DESKTOP-PGELV1B MINGW64 ~/Desktop
$
```

Screenshot of command line

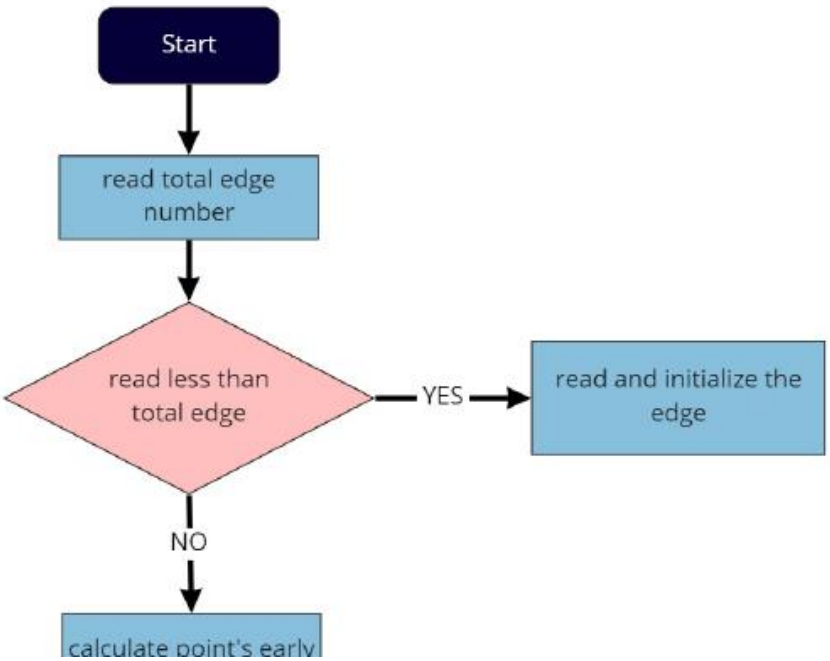
output.txt - 記事本

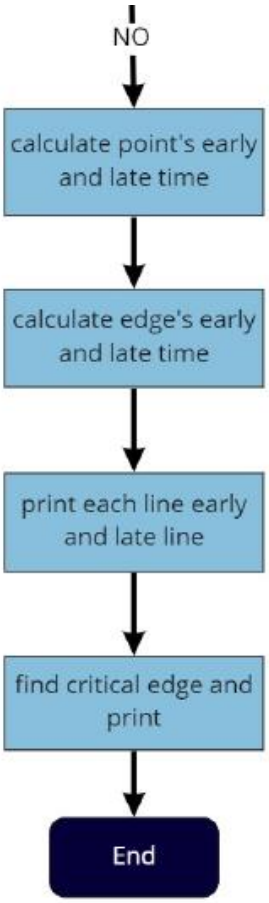
檔案(F) 編輯(E) 格式(O) 檢視(V) 說明

```
0 0 0
1 0 2
2 0 3
3 6 6
4 4 6
5 5 8
6 7 7
7 7 7
8 7 10
9 16 16
10 14 14
0 3 6 7 9 10
```

“output.txt” screenshot

(2) Program Architecture





(3) Program function

```
#define MAX 100//因為題目說 0<=I,j,k,w<100，所以將 MAX 訂 100

typedef struct node{//struct of vertex
    int e,l,num;
}node;
typedef struct edge{//struct of activity
    node *start,*end;
    int weight,name;
    struct edge *nextlink;
}edge;
typedef edge * eptr;
typedef struct list{//struct of adjacency list's element
    edge *link;
    int count;
}list;
```

Parameters

struct node

e-該節點的 early time，最早甚麼時候可以走

l-該節點的 late time，最晚甚麼時候可以走

num-該節點名稱

struct edge

start-指向這個邊的起點所連的 node

end-指向這個邊的終點所連的 node

weight-這個邊的 weight

name-這個邊的代號

nextlist-在 adjacency list 裡，一個 node 可能不只連接一個邊，如果他同時連接多個邊，那他會指向其中一個他連的邊，在由這個去指向其他的邊

struct list

link-連向這個 node 名稱會指的邊

count-前面有多少個邊連著它

```
int end_p,flag=0;
```

Parameters

end_p-計算有多個的節點

flag-控制結尾不空格的變數

Function

get_edge-產生新的 edge，負責在 adjancy lists 被連接及連接其他同起點的 edge

```
eptr get_edge(int e_name,int s,int f,int w,node v[])
{
    eptr tptr=malloc(sizeof(edge));
    tptr->weight=w;
    tptr->start=&v[s];//node 會用一個 array 方便依序紀錄，因此把邊的起始點指到具有相同 index 的 node array element
    tptr->end=&v[f]; //把邊的終點指到具有相同 index 的 node array element
    tptr->nextlink=NULL;//先初始化下一個可能會指到的邊，此時沒指任何東西
    tptr->name=e_name;
    return tptr;
}
```

Parameters

e_name-邊的名字

tptr-一個暫存的指標，指向 malloc 出來的空間

s-開始的 node 名稱

f-結束的 node 名稱

w-邊的 weight

v[]-存放 node 的 array

Return value

該程式會把產生完的 tptr 會回傳給 function read_data 使用

read_data-讀取邊並產生 early 與 late 的 adjacency list

```
void read_data(int e_num, list early[],list late[],node v[]) {
    for (int i = 0; i < MAX; i++) {//initialize some element
        early[i].count =late[i].count= v[i].e = 0;
        early[i].link = late[i].link = NULL;
        v[i].num = i;
    }
    for (int i = 0; i<e_num; i++) {//依序輸入邊，看第一行得到多少總邊數就讀幾組輸入
        int edg, s, f, w;
        scanf("%d%d%d%d", &edg, &s, &f, &w);//讀入邊的資料
        {//將讀入的邊的資料和 early adjancy list 做連接
            eptr tptr = get_edge(edg, s, f, w, v);//產生邊準備和 list 連接
            early[f].count++;//count 數的是那個點前面有幾個 node，所以要加在終點的點上
            if (early[s].link == NULL)//如果還沒有連接的邊，直接連接
                early[s].link = tptr;
            else {//如果有連接的邊，順著邊找直到還有沒連接邊的(排成 linked list)
                eptr now = early[s].link;
                while (now->nextlink != NULL) now = now->nextlink;
                now->nextlink = tptr;
            }
        }
        {//將讀入的邊的資料和 late adjancy list 做連接
            eptr tptr = get_edge(edg, f, s, w, v); //產生邊準備和 list 連接，因為是從尾端指回去，所以 f 和 s 的位置要交換
            late[s].count++;//count 數的是那個邊連那個 node，所以要加在起點的點上
            if (late[f].link == NULL)
                late[f].link = tptr;
            else {
```

```
                eptr now = late[f].link;
                while (now->nextlink != NULL) now = now->nextlink;
                now->nextlink = tptr;
            }
        }
        end_p = f;//calculate maximum point, 因為它會照順序，可以確定最大的一定是最後一個邊的結束點
    }
}
```

Parameters

e_num-最大的 edge 數量，甚麼時候結束輸出的依據

early[]-early adjacency list，依序紀錄連了哪些邊跟有多少邊連到該 node

late[]-late adjacency list，依序紀錄連了哪些邊跟有多少邊從該 node 出去

v[]-node 的資料，這邊會有 edge 連過來

edg-edge 的名字（編號）

s- start point of the edge

f- end point of the edge

w- weight of the edge

tptr-暫時指向 edge 的指標

now-在把 tptr 和 adjacency list 做連接時產生的指標，檢查哪裡是連接的尾端並讓 tptr 插入

Return value

function 沒有 return value

calculate_early-計算 node 的 early time

```
void calculate_early(list early[],int s[],int top,node v[])
{
    s[top]=0,top++;//從 0 開始算，所以先把 0 push 進 stack
    while(top>0)
    {
        int now=s[top-1];//pop out a number
        top--;
        eptr nptr=early[now].link;//從 early list 看那個點連了哪個邊
        while(nptr!=NULL)//如果還沒指向空，代表那個 node 還有連東西
        {
            int point=nptr->end->num;//point 是那個邊的終點
            if(nptr->end->e>(nptr->weight)+(v[now].e));//計算終點的 early time 如果原本的比較大，就不要動
            else nptr->end->e=(nptr->weight)+(v[now].e);//不然換新的值
```

```
            if(early[point].count>1) early[point].count--;//如果指向的邊的 start  point 的 count 大於 1，還不能 push，但 count-1
            else s[top]=point,top++;
            nptr=nptr->nextlink;
        }
    }
}
```

Parameters

early[]-early adjacency list

s[]-array 實作的 stack，要放 node 的 index

top-stack 的 top(可以從哪個開始放)，基本上開始用前和用完後 stack 都是空的，都是 0

v[]-node 的資料，要計算裡面的 e

now-現在 pop 出來的數，用來指引後面

nptr-指向現在在計算的邊

point-指向現在在計算 early time 的 node

Return value

這個 function 沒有 return value

calculate_late-計算 node 的 late time，方法和 early 差不多

```
void calculate_late(list late[],int s[],int top,node v[])
{
    for(int i=0;i<=end_p;i++) v[i].l=v[end_p].e;//先把每個都複製成 end_point 的 early  time 再來減
    s[top]=end_p,top++;//從尾端開始算，所以先把 0  push 進 stack
    while(top>0)
    {
        int now=s[top-1];
        top--;
        eptr nptr=late[now].link;
        while(nptr!=NULL)
        {
            int point=nptr->end->num;
            if(nptr->end->l<(v[now].l)-(nptr->weight)); //計算終點的 late  time 如果原本的比較小，就不要動
            else nptr->end->l=(v[now].l)-(nptr->weight);
            if(late[point].count>1) late[point].count--;
            else s[top]=point,top++;
            nptr=nptr->nextlink;
```

```
        }
    }
}
```

Parameters

late[]-late adjacency list

s[]-array 實作的 stack，要放 node 的 index

top-stack 的 top(可以從哪個開始放)，基本上開始用前和用完後 stack 都是空的，都是 0

v[]-node 的資料，要計算裡面的 l

now-現在 pop 出來的數，用來指引後面

nptr-指向現在在計算的邊

point-指向現在在計算 late time 的 node

Return value

這個 function 沒有 return value

critical_e-算每個 edge 的 early 和 late time，並按順序列出 critical edge

```
void critical_e(list early[],int e_num)
{
    int c_e[MAX]={0};
    edge all_edge[MAX];
    for(int i=0;i<end_p;i++)//利用 early adjacency list 來將 edge 按順序排入 edge 的 array all_edge[]
    {
        eptr nptr=early[i].link;
        while(nptr!=NULL)
        {
            int index=nptr->name;
            all_edge[index]=*nptr;
            nptr=nptr->nextlink;
        }
    }
    for(int i=0;i<e_num;i++)//依序印出每個 edge 的名稱以及 early time, late time
     {
        int early,late;
        early=all_edge[i].start->e;//邊的 early time 即為起點的 early time
        late=(all_edge[i].end->l)-(all_edge[i].weight);//邊的 late time 於課堂中有公式
```

```
        printf("%d ",i);
        printf("%d ",early);
        printf("%d",late);
        puts("");
        if(early==late) c_e[i]=1;//如果發現 early time=late time，即為 critical edge，將它記錄在 c_e 中
        else c_e[i]=0;
    }
   for(int i=0;i<e_num;i++)//依順序印出 critical edge
   {
       if(c_e[i])
       {
           if(flag)
               printf(" %d",i);
           else
           {
               flag=1;
               printf("%d",i);
           }
       }
   }
}
```

Parameters

early[]-early adjacency list

e_num-邊的數目

c_e-紀錄 activity 是否為 critical edge

all_edge[]-存放所有的邊的 edge array

nptr-現在準備紀錄的邊

early-紀錄邊的 early time

late-紀錄邊的 late time

Return Value

Function 沒有 return value

```
int main() {
    int e_num;
    list early[MAX],late[MAX];
    node v[MAX];
    scanf("%d",&e_num);
```

```
    read_data(e_num,early,late,v);
    int stack[MAX]={-1},top=0;
    calculate_early(early,stack,top,v);
    calculate_late(late,stack,top,v);
    critical_e(early,e_num);
    return 0;
}
```

Parameters

e_num-最大邊數，輸入第一行會告訴我們

early[]-early adjacency list，使用 array 是因為有已知最大邊數會小於 100，於是直接設定 100，以下同理

late[]-late adjacency list

v[]-array used to save node information

Return Value

若程式順利結束會回傳 0