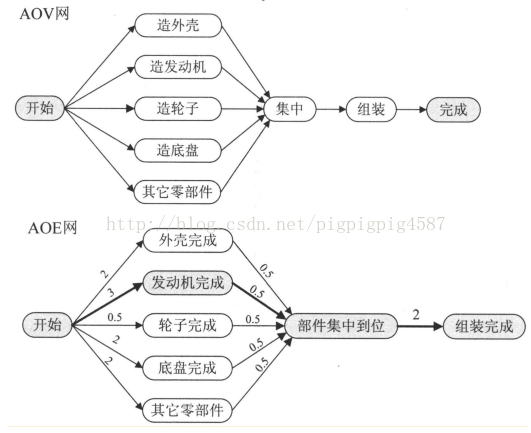
1. 相关概念



我们把路径上各个活动所持续的时间之和称为路径长度，从源点到汇点具有最大长度的路径叫关键路径，在关键路径上的活动叫关键活动。

最早开始时间与最晚开始时间不等，说明有空闲时间。也就是说，我们只需要找到所有活动的最早开始时间和最晚开始时间，并且比较它们，如果相等就意味着此活动是关键活动，活动间的路径为关键路径。如果不等就不是。为此有如下几个参数

1. 事件的最早发生时间etv(earliest time of vertex):即顶点vk的最早发生时间

2. 事件的最晚发生时间ltv(latest time of vertex):即顶点vk的最晚发生时间，超出此时间将会延误工期。

3. 活动的最早开工时间ete（earliest time of edge）：即弧ak的最早发生时间

1. 活动的最晚开工时间lte(latest time of edge):即弧ak的最晚发生时间，也就是不推迟工期的最晚开工时间。

/\*\*

\*

\* CriticalPath.c 关键路径算法

\*

\* @author Darbuly 970073804@QQ.com 2018-2019

\* @copyright 2018-2019

\*

\*

\*

\*/

#include <stdio.h>

#include <stdlib.h>

#define VRType int

#define InfoType int

#define VertexType char

#define MAX\_VERTEX\_NUM 20

#define OK 1

#define ERROR 0

typedef int Status;

typedef enum {DG,DN,AG,AN}GraphKind;

typedef struct ArcNode{

int adjvex;

struct ArcNode \*nextarc;

int weight;

InfoType \*info;

}ArcNode;

typedef struct VNode{

VertexType data;

int in;

ArcNode \*firstarc;

}VNode,AdjList[MAX\_VERTEX\_NUM];

typedef struct

{

AdjList vertices;

int vexnum,arcnum;

GraphKind kind;

}ALGraph;

int \*etv,\*ltv;

int \*stack2;

int top2;

void InsertNode(ALGraph \*G,int i,int j,int weight)

{

printf("\ni=%d,j=%d\n",i,j);

ArcNode \*ptrArcNode = (ArcNode \*)malloc(sizeof(ArcNode));

ptrArcNode->adjvex = j;

ptrArcNode->nextarc = G->vertices[i].firstarc;

ptrArcNode->weight = weight;

G->vertices[i].firstarc = ptrArcNode;

G->vertices[j].in++;

printf("%c-%c,weight:%d \n",G->vertices[i].data,G->vertices[j].data,ptrArcNode->weight);

}

void CreateALGraph(ALGraph \*G)

{

int i,j,k,weight;

if(NULL==G) G = (ALGraph\*)malloc(sizeof(ALGraph));

printf("Please input the vexnum and arcnum:\n");

scanf("%d%d",&G->vexnum,&G->arcnum);

getchar();

printf("Please input the Vertices:\n");

for(i=0;i<G->vexnum;i++)

{

printf("Vertex %d:",i+1);

scanf("%c",&G->vertices[i].data);

G->vertices[i].in = 0;

G->vertices[i].firstarc = NULL;

getchar();

}

for(k=0;k<G->arcnum;k++)

{

printf("Please input (vi,vj,weight) vertex:\n");

scanf("%d%d%d",&i,&j,&weight);

InsertNode(G,i,j,weight);

}

}

void ShowALGraph(ALGraph \*G)

{

ArcNode \*ptrArcNode;

int i;

printf("\nShowALGraph\n");

for(i=0;i<G->vexnum;i++)

{

printf("The Vertex %d is %c, in %d:",i+1,G->vertices[i].data,G->vertices[i].in);

ptrArcNode = G->vertices[i].firstarc;

while(1)

{

if(ptrArcNode==NULL)break;

printf("->%c",G->vertices[ptrArcNode->adjvex]);

if(ptrArcNode->nextarc==NULL)break;

ptrArcNode = ptrArcNode->nextarc;

}

printf("\n");

}

}

Status ToplogicalSort(ALGraph G)

{

ArcNode \*e;

int i,k,gettop;

int top = 0;

int count = 0;

int \*stack;

stack = (int\*)malloc(G.vexnum\*sizeof(int));

printf("\nToplogicalSort\n");

for(i=0;i<G.vexnum;i++)

{

if(0==G.vertices[i].in)

{

// printf("\ntop=%d\n",i);

stack[++top] = i;

}

}

top2 = 0;

etv = (int\*)malloc(G.vexnum\*sizeof(int));

for(i=0;i<G.vexnum;i++)

{

etv[i] = 0;

}

stack2 = (int\*)malloc(G.vexnum\*sizeof(int));

while(0!=top)

{

gettop = stack[top--];

stack2[++top2] = gettop;

count++;

for(e=G.vertices[gettop].firstarc;e;e=e->nextarc)

{

k = e->adjvex;

if(!(--G.vertices[k].in))

{

stack[++top]=k;

}

// printf("\n%d\n",etv[gettop]+e->weight);

if( (etv[gettop] + e->weight)>etv[k])

{

etv[k] = etv[gettop] + e->weight;

}

}

}

if(count < G.vexnum)

{

return ERROR;

}

else

{

printf("\netv :\n");

for(i=0;i<G.vexnum;i++)

{

printf("%d\t",etv[i]);

}

printf("\n");

return OK;

}

}

void CriticalPath(ALGraph G)

{

ArcNode \*e;

int i,gettop,k,j;

int ete,lte;

ToplogicalSort(G);

ltv = (int\*)malloc(G.vexnum\*sizeof(int));

for(i=0;i<G.vexnum;i++)

{

ltv[i] = etv[G.vexnum-1];

}

while(0 != top2)

{

gettop = stack2[top2--];

for(e=G.vertices[gettop].firstarc;e;e=e->nextarc)

{

k = e->adjvex;

if( (ltv[k] - e->weight) < ltv[gettop])

{

ltv[gettop] = ltv[k] - e->weight;

}

}

}

printf("\nltv======:\n");

for(i=0;i<G.vexnum;i++)

{

printf("%d\t",ltv[i]);

}

printf("\n");

for(j=0;j<G.vexnum;j++)

{

for(e=G.vertices[j].firstarc;e;e= e->nextarc)

{

k = e->adjvex;

ete = etv[j];

lte = ltv[k] - e->weight;

if(ete == lte)

{

printf("<v%c, v%c> length: %d , \n",G.vertices[j].data,G.vertices[k].data,e->weight);

}

}

}

}

int main()

{

printf("CriticalPath.c Test\n");

ALGraph G;

CreateALGraph(&G);

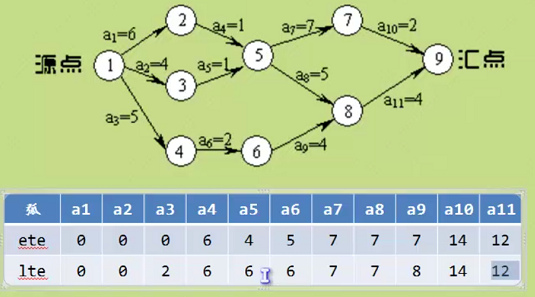
ShowALGraph(&G);

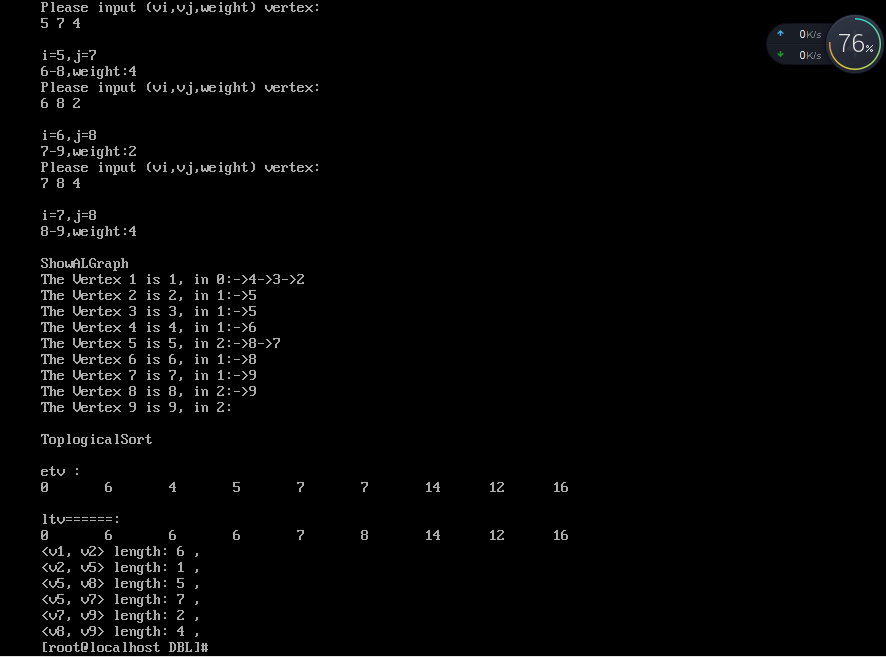
// ToplogicalSort(G);

CriticalPath(G);

return 0;

}





代码分析：

n个顶点，e条边，

这个算法来求关键路径，其实就是利用拓扑排序，首先求出，每个节点最早开始时间etv，再倒退求每个最晚开始的时间ltv。

从而算出活动最早开始的时间和最晚开始的时间，如果这两个时间相等，则为关键路径。

时间复杂度为O(n+e)