開發平台：Windows7

開發環境：DEV-C++4.9.9.2

sollin 程式功能：

輸入檔案名稱〈含副檔名〉，印出 minimum spanning tree

```
C:\Users\yuron\Documents\資料結構\sollin.exe
Please input file name: Sample_input1.txt
(0,5) = 10
(2,3) = 12
(1,6) = 14
(1,2) = 16
(3,4) = 22
(4,5) = 25
請按任意鍵繼續 . . .
```

sollin 程式架構：

```
typedef struct{                 //use to record a edge
        int endpoint1;          //one endpoint of edge
        int endpoint2;          //another endpoint of edge
        int weight;             //weight of edge
}EDGE;
EDGE Find_min_cost_edge() :
```

```
EDGE Find_min_cost_edge(int **graph,int vertex,int **MST,int *flag,int cur_vertex)
{
    EDGE min;
    int i,next_vertex[vertex],front = 0,rear = 0;

    next_vertex[0] = -1;            //Queue to store set member
    min.weight = INT_MAX;
    while( cur_vertex != -1 )       //search the set for min. cost edge
    {
        for(i = 0 ; i < vertex ; i++)   //search min. cost edge for current vertex in graph
            if( graph[cur_vertex][i] && (graph[cur_vertex][i] < min.weight) )
            {
                min.weight = graph[cur_vertex][i];          //min is used to record the smallest cost edge
                min.endpoint1 = cur_vertex;
                min.endpoint2 = i;
            }
        for(i = 0 ; i < vertex ; i++)  // find other vertices in set
        {
            if( MST[cur_vertex][i] && !flag[i])
            {
                next_vertex[rear] = i;
                next_vertex[++rear] = -1;
            }
        }
        flag[cur_vertex] = 1;
        cur_vertex = next_vertex[front++]; //move to next vertex in set until no vertex in set
    }
    return min;
}
```

void Sollin_MST() :

```
void Sollin_MST(int **graph,int vertex,int **MST)
{
	int n = 0,i,j;
	int flag[vertex];
	EDGE temp;

	for(i = 0 ; i < vertex ; i++) flag[i] = 0; //initializing flag
	while(n < vertex-1)
	{
		for(i = 0 ; i < vertex ; i++)	//each stage start
		{
			if(flag[i]) continue;	//searched vertices will be ignored
			temp = Find_min_cost_edge(graph,vertex,MST,flag,i);
			MST[temp.endpoint1][temp.endpoint2] = temp.weight; //store min. cost edge of a tree to minimum spanning tree
		}
		for(i = 0 ; i < vertex ; i++) //reset flag for next stage and eliminate picked edge from graph
		{
			flag[i] = 0;		//reset
			for(j = i ; j < vertex ; j++)
			{
				if(MST[i][j] || MST[j][i])	//adjacency matrix of undirected graph is symmetric
				{				//store non-zero value to another side
					(MST[i][j] != 0) ? (MST[j][i] = MST[i][j]) : (MST[i][j] = MST[j][i]);
					graph[i][j] = graph[j][i] = 0;	//eliminate
					n++;
				}
			}
		}
	}
}
```

void List_MST_edge()：儲存 minimum spanning tree 的所有邊，並且由小到大排序〈使用 insertion sort〉，最後印出結果。

aoe_network 程式功能：

輸入檔案名稱〈含副檔名〉，印出邊的 early time, late time, critical activities 和 critical path

```
C:\Users\yuron\Documents\資料結構\aoe_network.exe
Please input file name: a.txt
activity        early time      late time       slack
<0,1>   :       0               0               0
<0,2>   :       0               2               2
<0,3>   :       0               3               3
<1,4>   :       6               6               0
<2,4>   :       4               6               2
<3,5>   :       5               8               3
<4,6>   :       7               7               0
<4,7>   :       7               7               0
<5,7>   :       7               10              3
<6,8>   :       16              16              0
<7,8>   :       14              14              0
critical path:
<0,1>
<1,4>
<4,6>,<4,7>
<7,8>
<6,8>
請按任意鍵繼續
```

aoe_network 程式架構：

void AOE()：

```
void AOE(int **graph,int *count,int vertex)                    //count record the number of predecessors
{
     int i,j,k,early[vertex],late[vertex];
     int e,l,slack,flag;

     modified_TopoSort(graph,count,early,vertex);              //calculate earliest start time by modified_TopoSort

     for(i = 0 ; i < vertex ; i++) late[i] = early[vertex-1]; //initialize latest time for vertex to finished time
     for(i = vertex -2 ; i >= 0; i--)                          //calculate latest time with inverse order of top. sort(count
           for(j = 0 ; j < vertex ; j++)
                 if( graph[count[i]][j] && (late[count[i]] > late[j] - graph[count[i]][j]))
                     late[count[i]] = late[j] - graph[count[i]][j];

     printf("activity\tearly time\tlate time\tslack\n");
     for(i = 0 ; i < vertex ; i++)
           for(j = 0 ; j < vertex ; j++)                       //calculate early and late of activities(edges)
                 if( graph[i][j])
                 {
                     e = early[i];
                     l = late[j] - graph[i][j];
                     slack = l - e;
                     printf("(%d,%d)\t:\t%d\t\t%d\t\t%d\n",i,j,e,l,slack);
                     if(slack) graph[i][j] = 0;          //delete noncritical activities
                 }

     printf("critical path:\n");
     for(i = 0 ; i < vertex ; i++)          //print all critical paths
```

void modified_TopoSort()：

```
void modified_TopoSort(int **graph,int *count,int *early,int vertex)
{
     int i,j,k,top = -1,order = 0;
     int temp[vertex];

     for(i = 0 ; i < vertex ; i++)                //push starting vertex to stack
           if( !count[i])
           {
               count[i] = top;
               top = i;
           }
     for(i = 0 ; i < vertex ; i++) early[i] = 0; //initializing earliest start time for vertex to zero
     for(i = 0 ; i < vertex ; i++)                //determine the topological order(refer to textbook) and earliest start time
     {
           if(top <= -1)
           {
                 printf("Infeasible network! Network has a cycle.");
                 system("pasue");
                 exit(EXIT_FAILURE);
           }
```

```
        else
        {
            k = top;                        //pop from stack
            top = count[k];                 //top move down
            count[k] = order--;             //record top. order in pop element
            for(j = 0 ; j < vertex ; j++)
                  if( graph[k][j])
                  {
                      count[j]--;
                      if( !count[j])
                      {
                          count[j] = top;
                          top = j;
                      }
                      if(early[j] < early[k] + graph[k][j]) early[j] = early[k] + graph[k][j]; //determine early
                  }
        }
    }
    for(i = 0 ; i < vertex ; i++) temp[i] = -count[i]; //count become top. ordered vertices in network
    for(i = 0 ; i < vertex ; i++) count[temp[i]] = i;
```