离散第三次上机作业

## 一、读取数据:getchar()

需要包含头文件<stdio.h>

getchar()是在输入缓冲区顺序读入一个字符(包括空格、回车和Tab)

getchar()函数可以直接用：getchar();

这种是吸收了一个缓冲区最前面的字符（包括空格，回车，tab）比如： （注：putchar( )是打印字符的函数）

   #include <iostream>

#include<stdio.h>

using namespace std;

int main() {

 int n,m;

    cin>>n>>m;

    char c;

    c=getchar();

    putchar(c);

}

input:30 100a

Output:a

Input:30 100 a

Output: （空格）

这里打印出来的c就是前面cin n和m的最后打出来的回车键，所以你输入：1 2（回车）

，后会直接return 0

而如果加入一个getchar()语句，这个语句会吸收掉前面的回车。输入台可以继续输入。

比如这里：输入1 2 a 会输出a。

#include <iostream>

#include<stdio.h>

using namespace std;

int main() {

int n,m;

    cin>>n>>m;

    char c;

    getchar();

    c=getchar();

    putchar(c);

}

Input：1 2 a

Output：a

Input：1 2

A（回车后+a）

Output：无

吸收了回车后return 0；

getchar()也可以用来赋值，返回类型就是一个char：

比如char c; c=getchar();这里的getchar在吸收一个缓冲区的字符后，将该字符赋值给c。这样做就保存下来了我们吸收的字符。

getchar()也可以作为一个表达式（右值）直接进行操作，比如：if (getchar()!=’/n’)是合法的表达

## 二、带注释的代码

#include <iostream>

#include <vector>

#include <queue>

#include <stdio.h>

#include <algorithm>

**using** **namespace** std;

**struct** Vnode{//节点struct

**int** V;//工序编号

**int** time;

**int** count=0;//记录顶点入度

**int** diff;//可拖延时间

    vector<**int**>prec;//记录所有前驱节点

    vector<**int**>succ;//记录所有后继节点

};

Vnode node[103];

vector<**int**>seq;//记录拓扑排序结果，seq[i]表示排序后的第i个节点应该是哪个原编号节点

**int** Pi[103]={0};//记录该工序开始前需要花费的最短时间（不包含该工序完成时间）（也就是最早启动时间）

**int** T[103];//记录每个工序的最晚启动时间

vector<**int**>Pre[103];//记录每个工序前需要等时间最久的那道工序

**int** a[103];//a[i]用来记录原编号是i的节点现在是第几个节点，这是seq[i]的反函数

**void** TopSort(Vnode adj[],**int** n){//拓扑排序，得到的结果是seq[]

    queue<**int**>q;

**for** (**int** i=0;i<n;i++){

**if** (adj[i].count==0){q.push(i);}

    }

**while** (!q.empty()){

**int** p=q.front();

        q.pop();

        seq.push\_back(p);

**for** (**int** i=0;i<adj[p].succ.size();i++){

**int** y=adj[p].succ[i];

            adj[y].count--;

**if** (adj[y].count==0){

                q.push(y);

            }

        }

    }

}

vector<**int**>path;

**void** dfs(**int** n){//用深度优先搜索找到节点node[n]的每一条关键路径

**if** (node[n].prec.size()==0){//递归的终结条件：如果这个点是起始点，输出整条path

        cout<<"[";

**for** (**int** k=path.size()-1;k>=0;k--){

            cout<<node[path[k]].V<<"->";

        }//这里也可以用stack，但是用vector也没差，只不过stack因为是先进后出，所以可以顺序输出，而vector需要反向输出

        cout<<"end]"<<endl;

**return**;

    }

**for** (**int** i=0;i<Pre[n].size();i++){

        path.push\_back(Pre[n][i]);//将某一个前驱放到path里头

        dfs(Pre[n][i]);//递归，深度优先搜索

        path.pop\_back();//删除path的最后一个元素，这是回溯

    }

}

**void** printdiff(**int** n){

**for** (**int** i=0;i<n-1;i++){

        cout<<node[a[i]].diff<<endl;

    }

}//输出可拖延时间（要按照原来的顺序输出！所以是a[i]

**int** main() {

**int** n;

    cin>>n;

**for** (**int** i=0;i<n;i++){

        cin>>node[i].V>>node[i].time;

**char** c;

**if** ((c=getchar())!=' '){**continue**;}//讨论如果第三个数据为空的情况

**int** num=0;

**bool** noPre=**true**;

**while**((c = getchar()) >' '){//这里本来应该是判断c!=’\n’,但是由于“不同系统中的输入格式有区别，比如\n可能和\r之类奇怪的东西混在一起“——志愿者语，改成了c>’ ‘的判断方式。当还没有输入换行符的时候，用getchar函数处理输入的东西，并转化成int型

            noPre=**false**;

**if** (c!=','){//如果接受到的字符不是逗号

**int** tmp=c-'0';

                num=num\*10+tmp;

            }

**else**{//接收到逗号，此时num正好是先驱节点的编号

                node[i].prec.push\_back(num);

                node[num].succ.push\_back(i);

                node[i].count++;

                num=0;

            }//将i的信息加入其前驱节点的后继节点中，同时将num加入点i的先驱节点中。

        }//对于每一行的最后一个点，从while循环里出来后num恰好是最后一个先驱节点的编号

**if** (noPre){**continue**;}

**else**{

            node[i].prec.push\_back(num);

            node[num].succ.push\_back(i);

            node[i].count++;}

    //设置虚拟终止点

    vector<**int**>end;

**for** (**int** i=0;i<n;i++){

**if** (node[i].succ.size()==0){

            end.push\_back(i);

            node[i].succ.push\_back(n);//将虚拟终止点设成所有原来的终止点的后继

        }

    }

    node[n].V=n;

    node[n].time=0;

    node[n].count=end.size();    //中止虚拟点，序号是n。

    node[n].prec=end;

    n=n+1;

    Vnode temp[103];

**for** (**int** i=0;i<n;i++){

        temp[i]=node[i];

    }

    TopSort(node,n);

**for** (**int** i=0;i<n;i++){

        node[i]=temp[seq[i]];//拓扑排序完成

    }

**for** (**int** i=0;i<n;i++){

        a[seq[i]]=i;

    }

**for** (**int** i=1;i<n;i++){//找从起点到每一个点的最长距离

**int** tmp=0;

**for** (**int** j=0;j<node[i].prec.size();j++){//第i个点的第j个前驱

**if** (tmp<Pi[a[node[i].prec[j]]]+node[a[node[i].prec[j]]].time){

                tmp=Pi[a[node[i].prec[j]]]+node[a[node[i].prec[j]]].time;

                Pre[i].clear();

                Pre[i].push\_back(a[node[i].prec[j]]);//记录前一个点；

            }

**else** {**if** (tmp==Pi[a[node[i].prec[j]]]+node[a[node[i].prec[j]]].time){//else!else!else!

                Pre[i].push\_back(a[node[i].prec[j]]);

            }

        }

    }

        Pi[i]=tmp;//如果这个点没有前驱，tmp就是0.

    }//找到了每个点的Pi值，同时记录到这个点的关键路径前的所有点

**int** Time=Pi[n-1];

    cout<<Time<<endl;

    dfs(n-1);

    T[n-1]=Time;

**for** (**int** i=n-2;i>=0;i--){

**int** tmp=Time;

**if** (node[i].succ.size()==0){

            T[i]=Time-node[i].time;

**continue**;

        }

**for** (**int** j=0;j<node[i].succ.size();j++){

**if** (tmp>T[a[node[i].succ[j]]]-node[i].time){

                tmp=T[a[node[i].succ[j]]]-node[i].time;

            }

        }

        T[i]=tmp;

    }

**for** (**int** i=0;i<n;i++){

        node[i].diff=T[i]-Pi[i];

    }

    printdiff(n);

**return** 0;

}