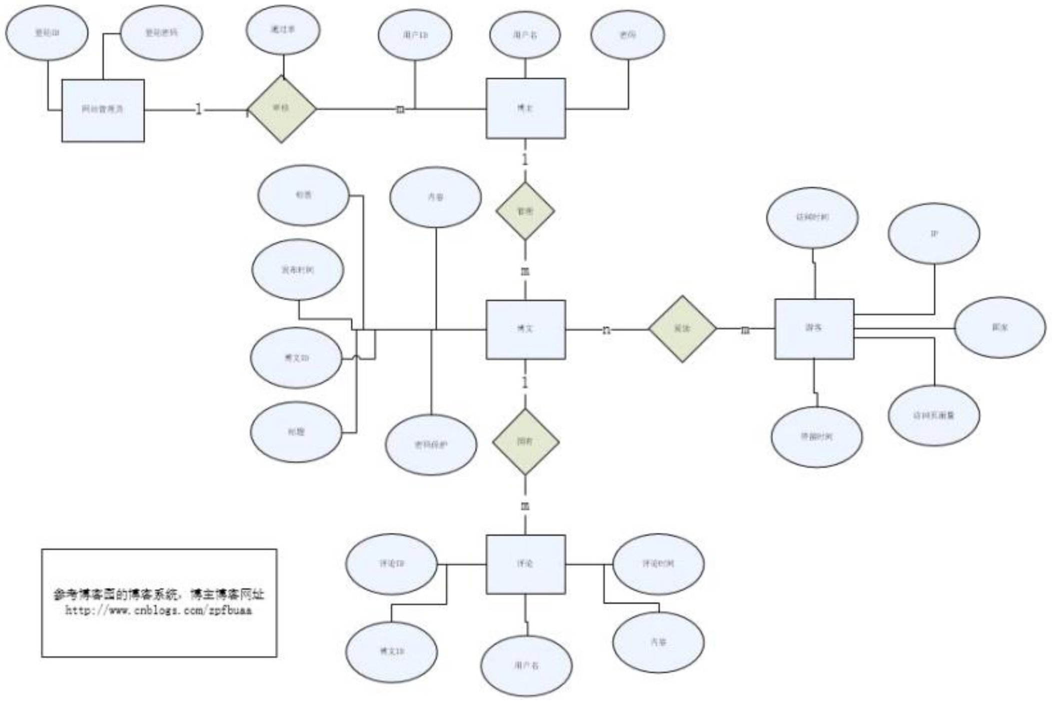
软工绘图

2016年5月6日 22:35

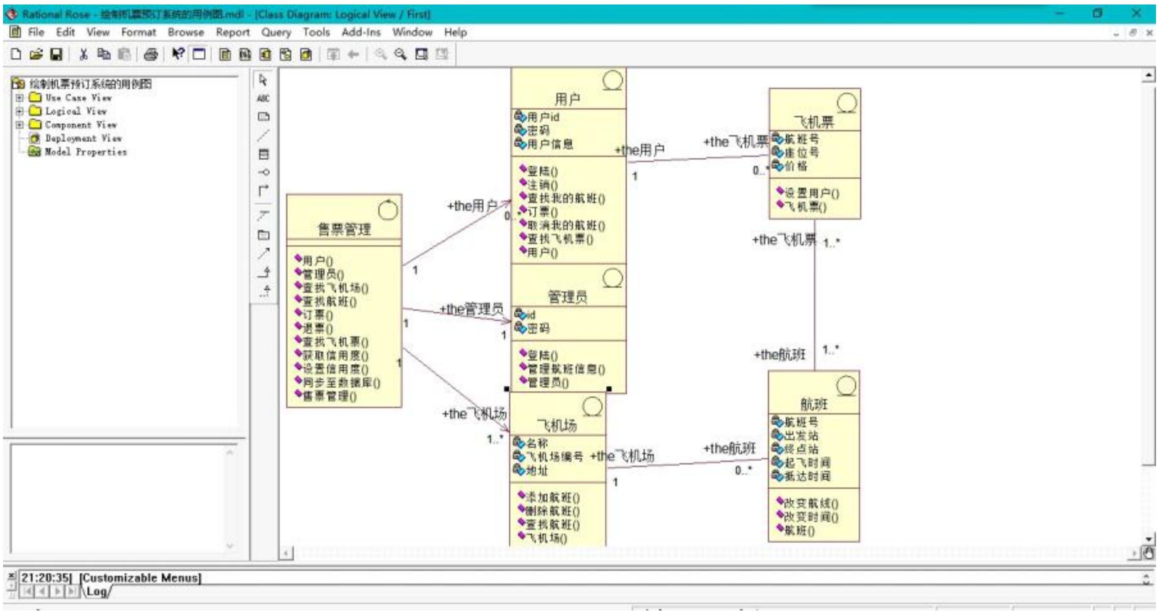
[由于是截图，清晰度可能不够，这里有源文件：http://pan.baidu.com/s/1i4XC6HJ](http://pan.baidu.com/s/1i4XC6HJ)

个人博客ER图：



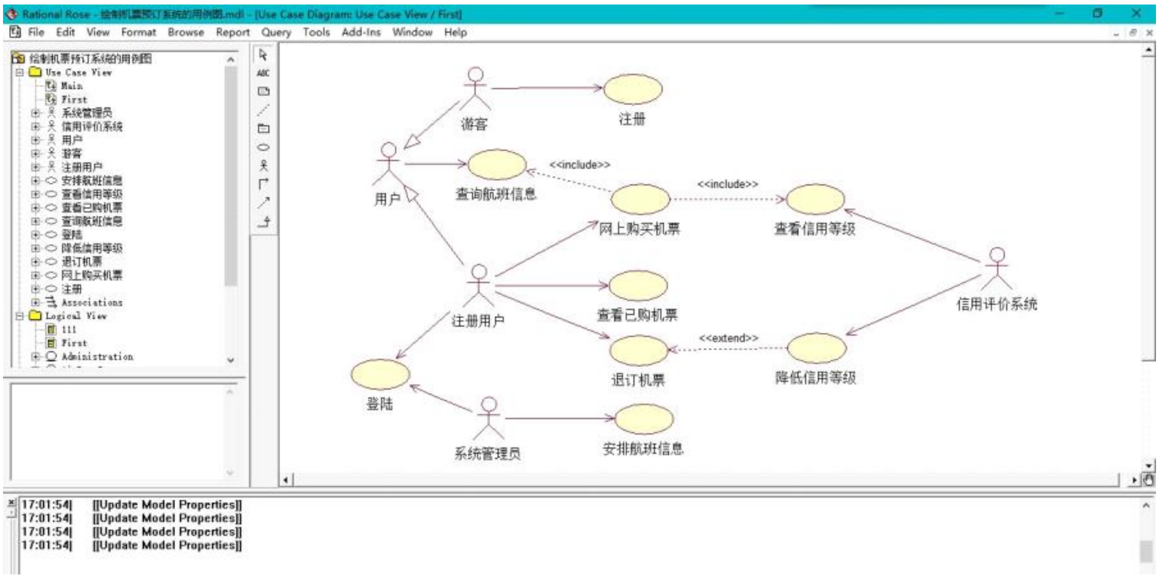
参考博客园的博客系统，博主博客网址
http://www.cnblogs.com/zpfbuaa

绘制机票预订系统的类图：


Rational Rose - 绘制机票预订系统的用例图.mdl - [Class Diagram: Logical View / First]

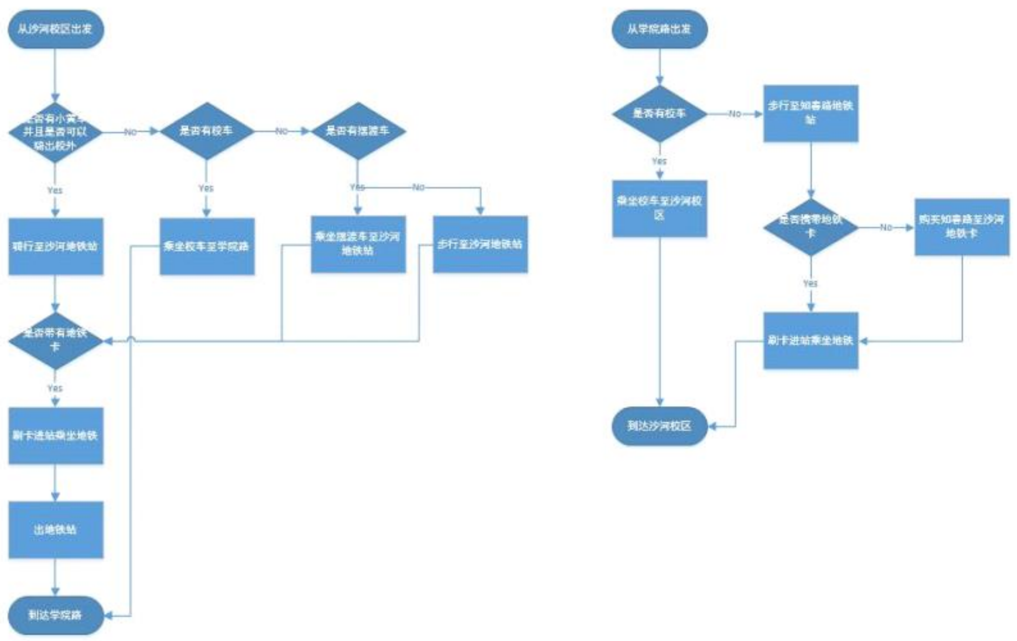
```
21:20:35]  [Customizable Menus]
```

绘制机票预订系统的用例图

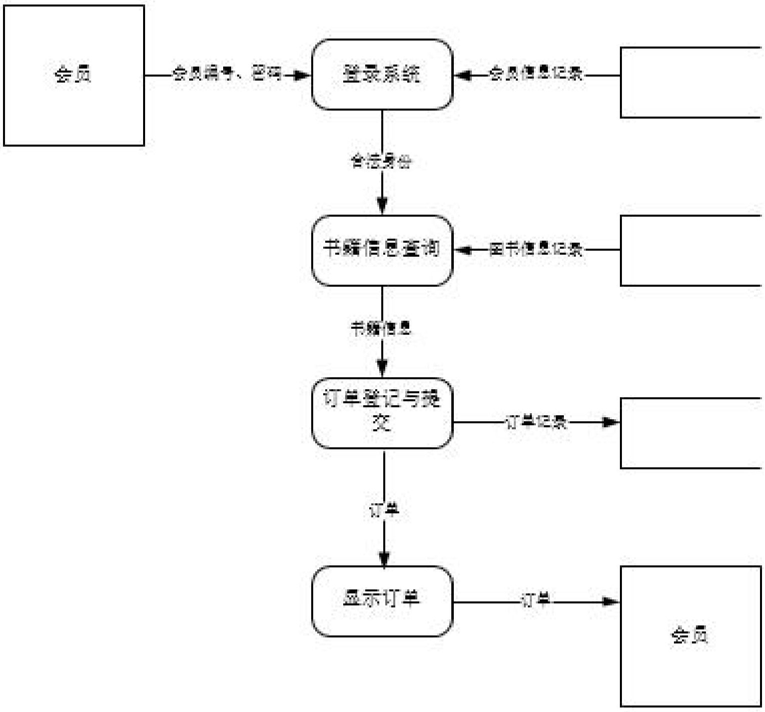


```
17:01:54|     [[Update Model Properties]]
17:01:54|     [[Update Model Properties]]
17:01:54|     [[Update Model Properties]]
17:01:54|     [[Update Model Properties]]
```

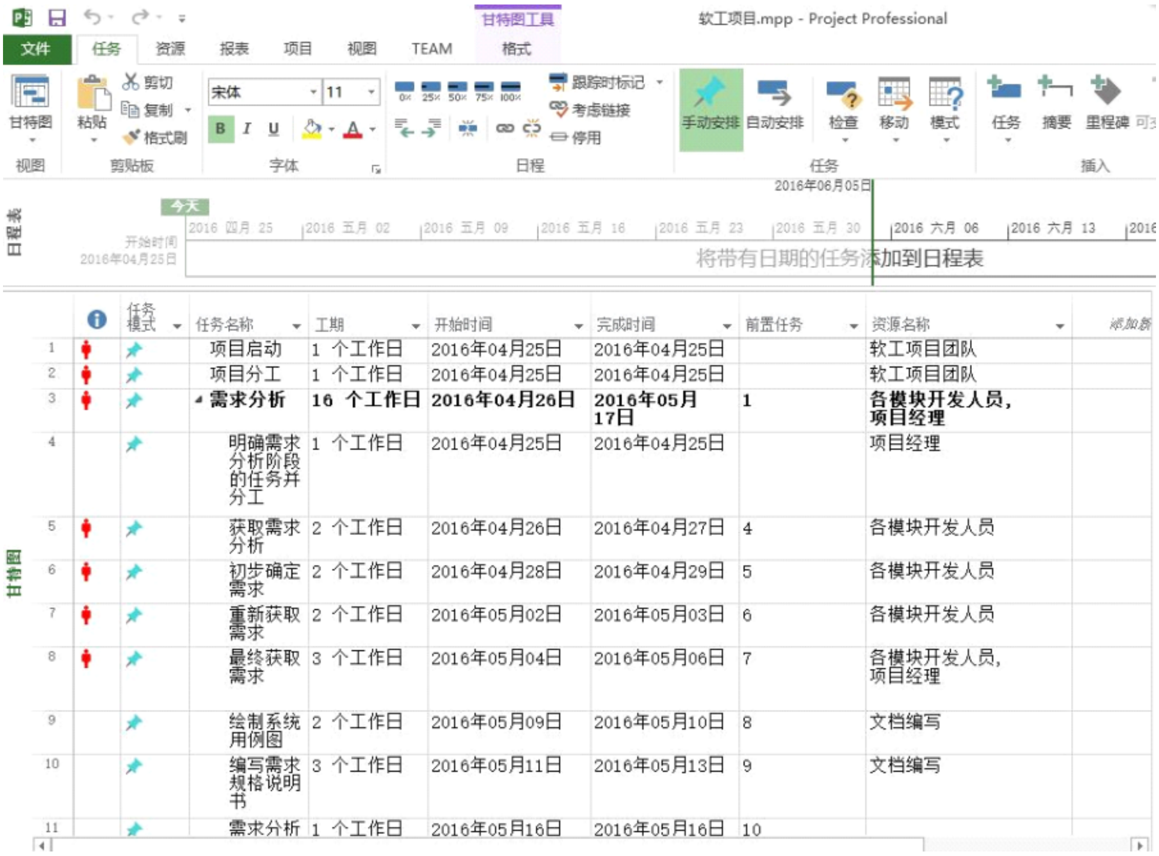
沙河<->本部：



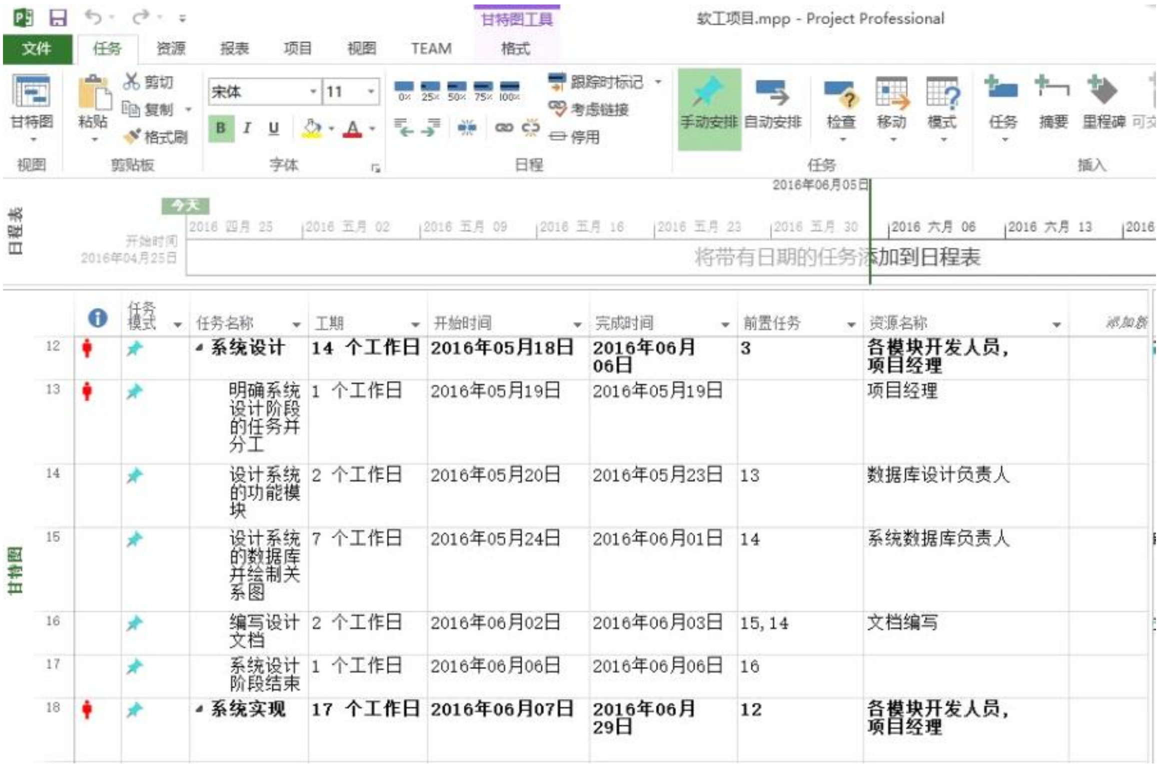
网上购书流程图：



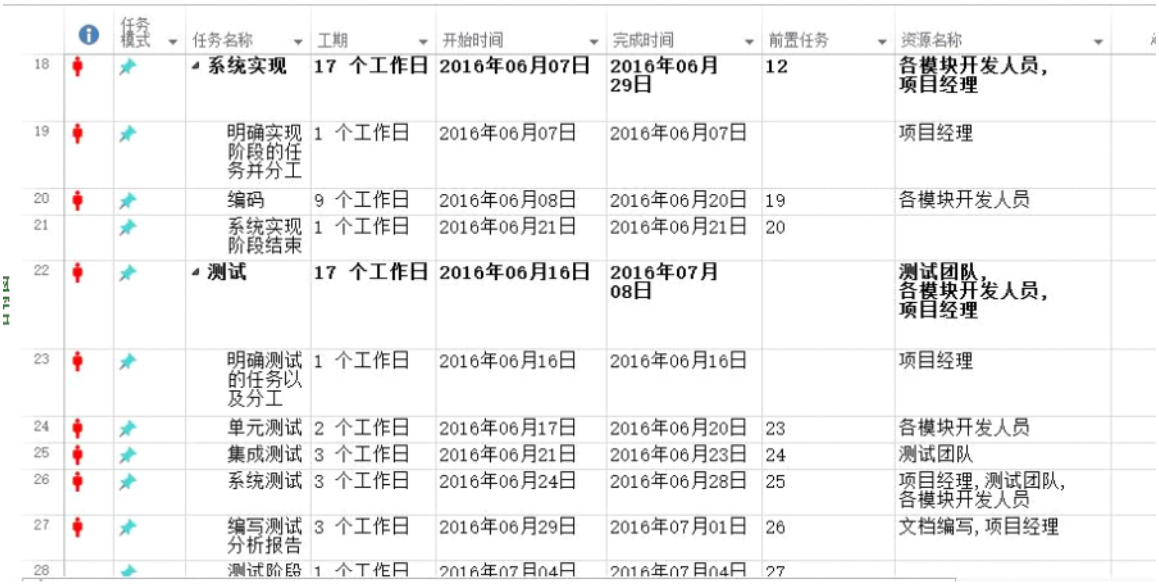
软工项目甘特图：



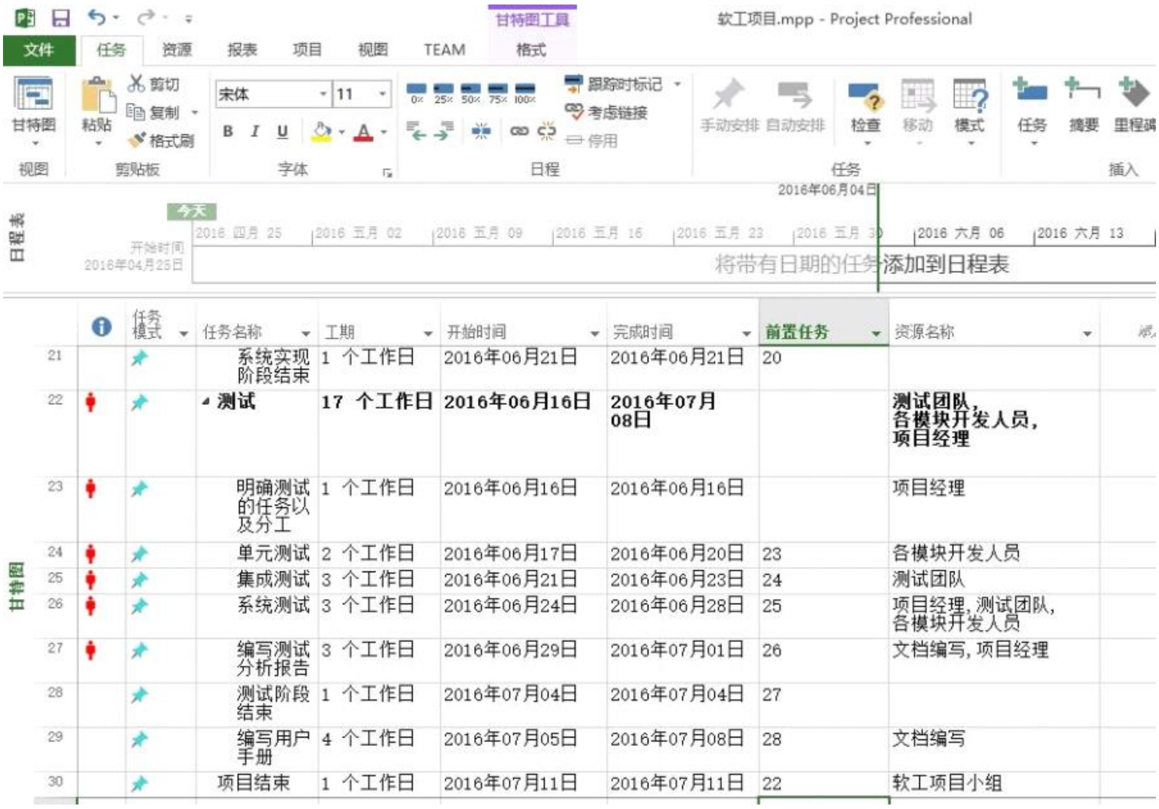
	任务模式	任务名称	工期	开始时间	完成时间	前置任务	资源名称
1		项目启动	1 个工作日	2016年04月25日	2016年04月25日		软工项目团队
2		项目分工	1 个工作日	2016年04月25日	2016年04月25日		软工项目团队
3		**需求分析**	**16 个工作日**	**2016年04月26日**	**2016年05月17日**	**1**	**各模块开发人员，项目经理**
4		明确需求分析阶段的任务并分工	1 个工作日	2016年04月25日	2016年04月25日		项目经理
5		获取需求分析	2 个工作日	2016年04月26日	2016年04月27日	4	各模块开发人员
6		初步确定需求	2 个工作日	2016年04月28日	2016年04月29日	5	各模块开发人员
7		重新获取需求	2 个工作日	2016年05月02日	2016年05月03日	6	各模块开发人员
8		最终获取需求	3 个工作日	2016年05月04日	2016年05月06日	7	各模块开发人员，项目经理
9		绘制系统用例图	2 个工作日	2016年05月09日	2016年05月10日	8	文档编写
10		编写需求规格说明书	3 个工作日	2016年05月11日	2016年05月13日	9	文档编写
11		需求分析	1 个工作日	2016年05月16日	2016年05月16日	10	



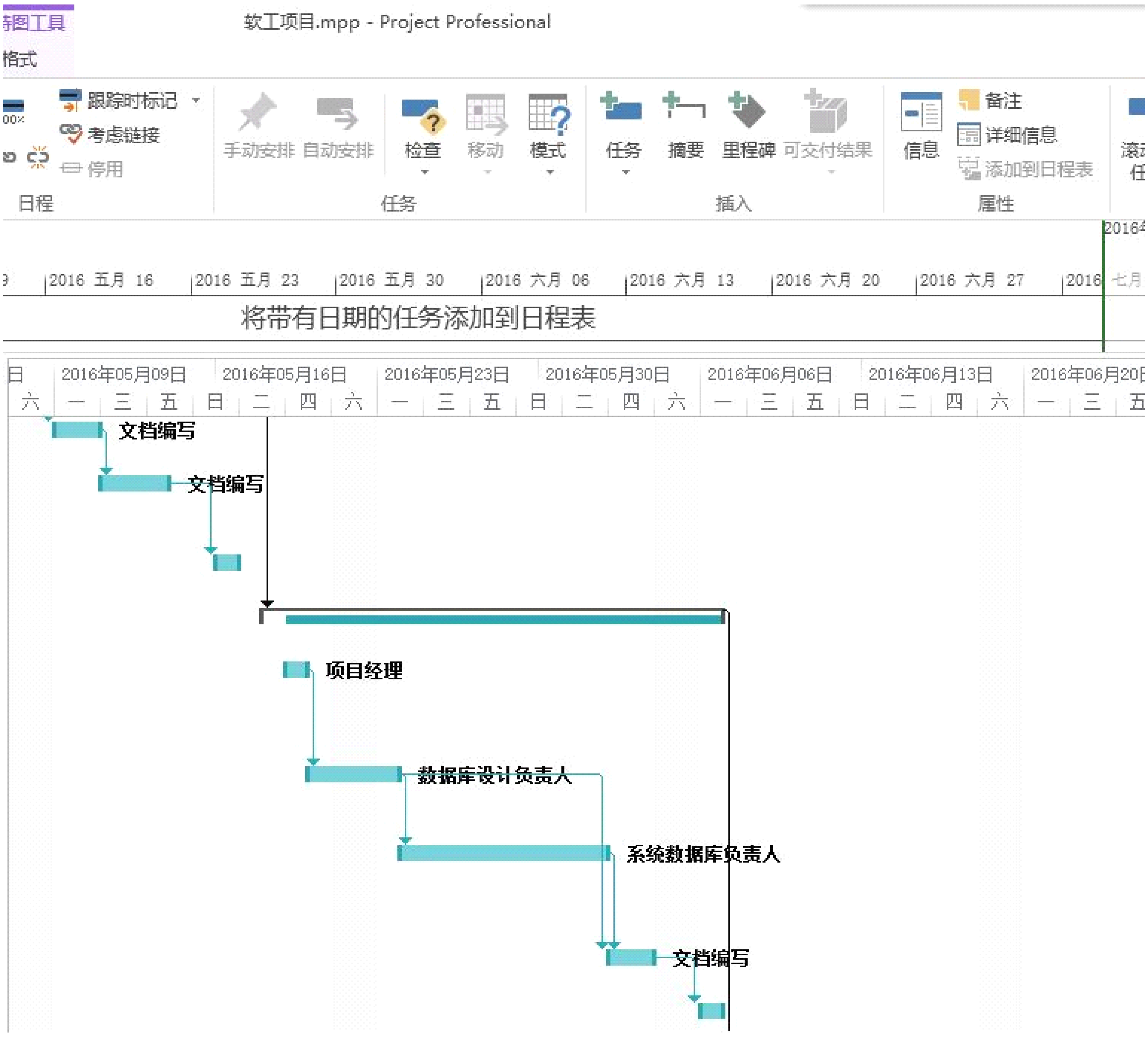
	ⓘ	任务模式	任务名称	工期	开始时间	完成时间	前置任务	资源名称	添加新
12			**◢ 系统设计**	**14 个工作日**	**2016年05月18日**	**2016年06月06日**	**3**	**各模块开发人员，项目经理**	
13			明确系统设计阶段的任务并分工	1 个工作日	2016年05月19日	2016年05月19日		项目经理	
14			设计系统的功能模块	2 个工作日	2016年05月20日	2016年05月23日	13	数据库设计负责人	
15			设计系统的数据库并绘制关系图	7 个工作日	2016年05月24日	2016年06月01日	14	系统数据库员责人	
16			编写设计文档	2 个工作日	2016年06月02日	2016年06月03日	15，14	文档编写	
17			系统设计阶段结束	1 个工作日	2016年06月06日	2016年06月06日	16		
18			**◢ 系统实现**	**17 个工作日**	**2016年06月07日**	**2016年06月29日**	**12**	**各模块开发人员，项目经理**	

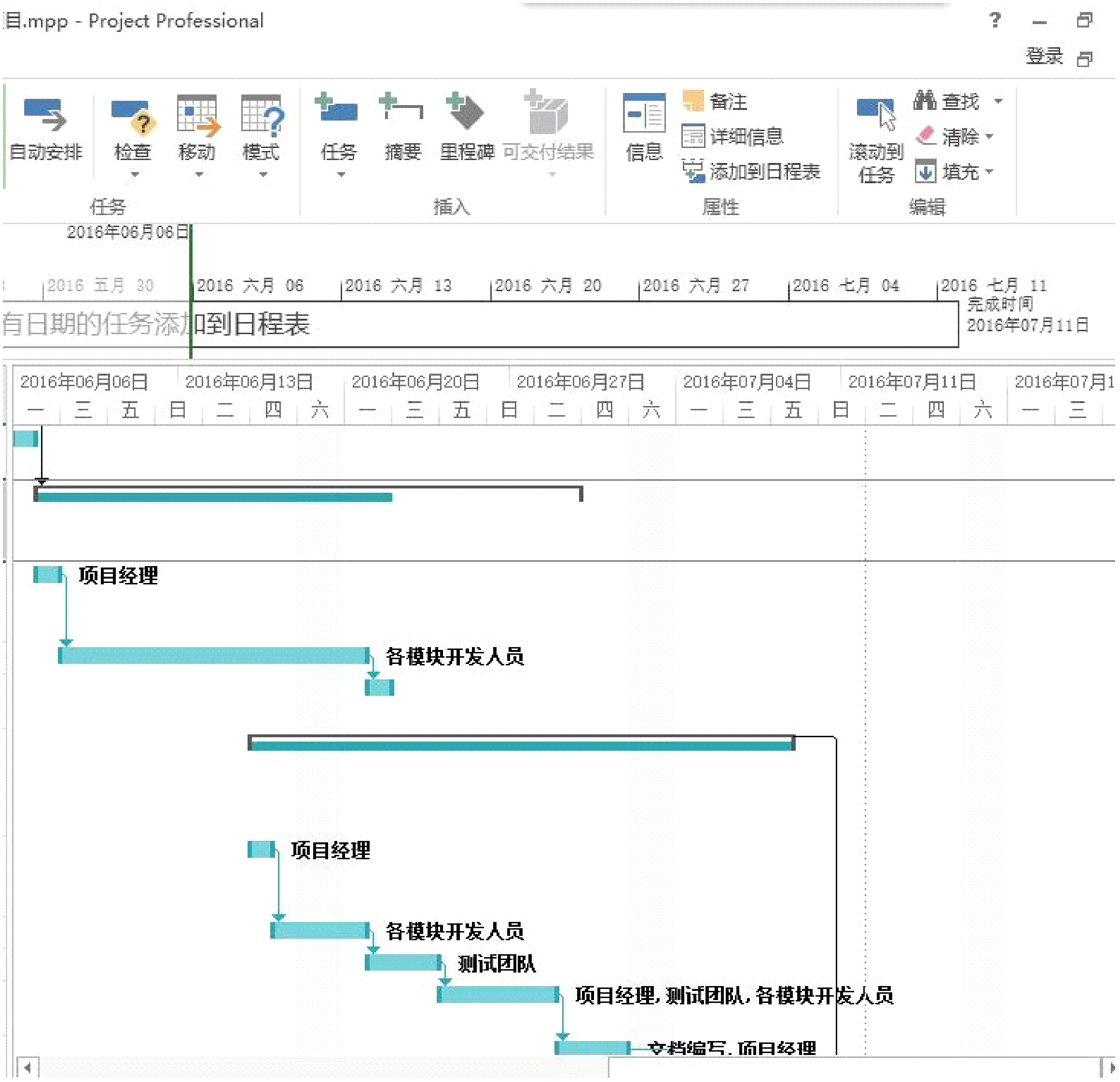


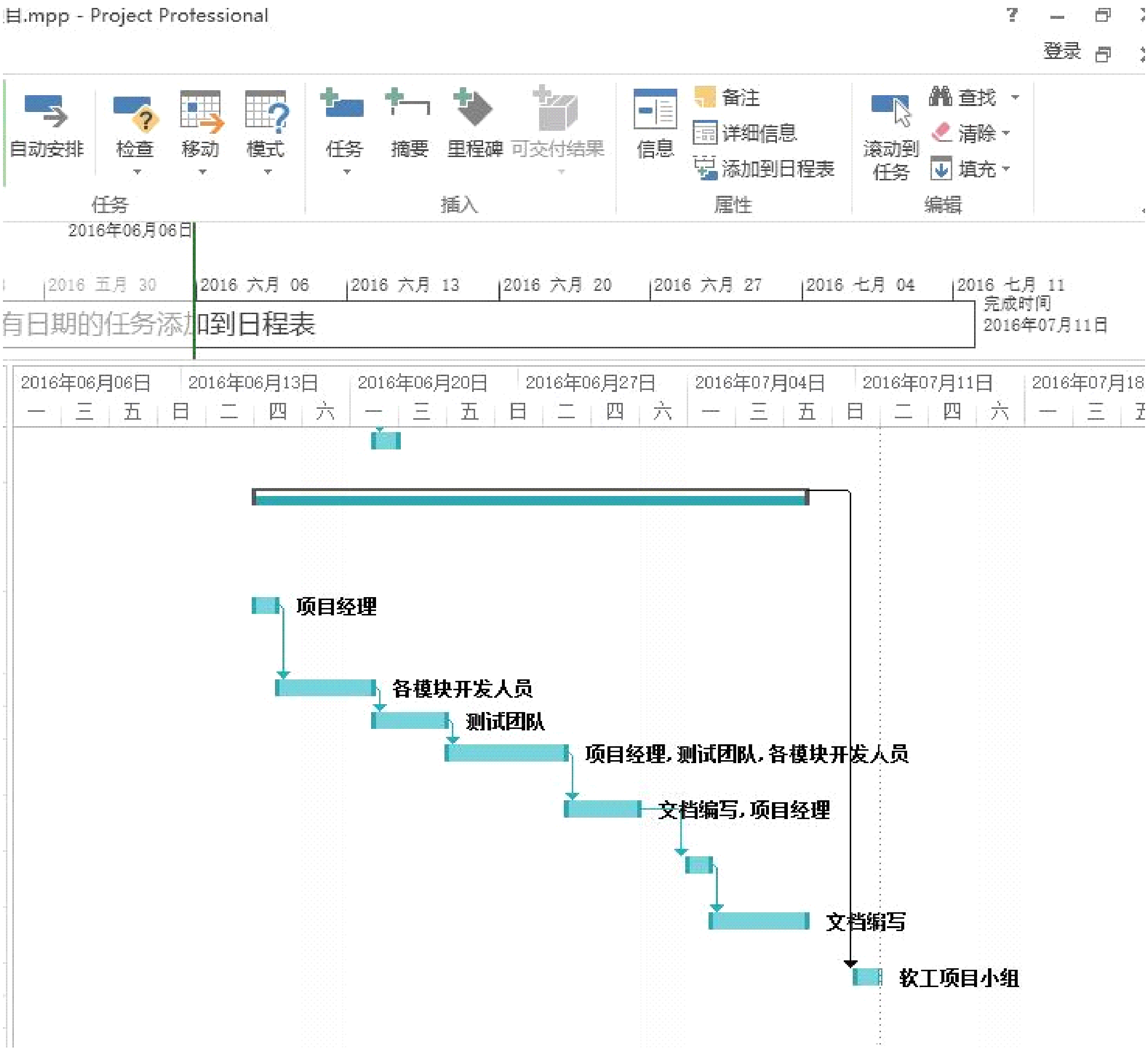
	ⓘ	任务模式	任务名称	工期	开始时间	完成时间	前置任务	资源名称
18			**◢ 系统实现**	**17 个工作日**	**2016年06月07日**	**2016年06月29日**	**12**	**各模块开发人员，项目经理**
19			明确实现阶段的任务并分工	1 个工作日	2016年06月07日	2016年06月07日		项目经理
20			编码	9 个工作日	2016年06月08日	2016年06月20日	19	各模块开发人员
21			系统实现阶段结束	1 个工作日	2016年06月21日	2016年06月21日	20	
22			**◢ 测试**	**17 个工作日**	**2016年06月16日**	**2016年07月08日**		**测试团队，各模块开发人员，项目经理**
23			明确测试的任务以及分工	1 个工作日	2016年06月16日	2016年06月16日		项目经理
24			单元测试	2 个工作日	2016年06月17日	2016年06月20日	23	各模块开发人员
25			集成测试	3 个工作日	2016年06月21日	2016年06月23日	24	测试团队
26			系统测试	3 个工作日	2016年06月24日	2016年06月28日	25	项目经理，测试团队，各模块开发人员
27			编写测试分析报告	3 个工作日	2016年06月29日	2016年07月01日	26	文档编写，项目经理
28			测试阶段	1 个工作日	2016年07月04日	2016年07月04日	27	



	任务模式	任务名称	工期	开始时间	完成时间	前置任务	资源名称
21		系统实现阶段结束	1 个工作日	2016年06月21日	2016年06月21日	20	
22		**◢ 测试**	**17 个工作日**	**2016年06月16日**	**2016年07月08日**		**测试团队，各模块开发人员，项目经理**
23		明确测试的任务以及分工	1 个工作日	2016年06月16日	2016年06月16日		项目经理
24		单元测试	2 个工作日	2016年06月17日	2016年06月20日	23	各模块开发人员
25		集成测试	3 个工作日	2016年06月21日	2016年06月23日	24	测试团队
26		系统测试	3 个工作日	2016年06月24日	2016年06月28日	25	项目经理，测试团队，各模块开发人员
27		编写测试分析报告	3 个工作日	2016年06月29日	2016年07月01日	26	文档编写，项目经理
28		测试阶段结束	1 个工作日	2016年07月04日	2016年07月04日	27	
29		编写用户手册	4 个工作日	2016年07月05日	2016年07月08日	28	文档编写
30		项目结束	1 个工作日	2016年07月11日	2016年07月11日	22	软工项目小组




目.mpp - Project Professional
?
登录
自动安排
检查
移动
模式
任务
任务
摘要
里程碑
可交付结果
插入
信息
备注
详细信息
添加到日程表
属性
滚动到任务
查找
清除
填充
编辑
2016年06月06日
2016 五月 30
2016 六月 06
2016 六月 13
2016 六月 20
2016 六月 27
2016 七月 04
2016 七月 11
完成时间
2016年07月11日
有日期的任务添加到日程表
2016年06月06日
2016年06月13日
2016年06月20日
2016年06月27日
2016年07月04日
2016年07月11日
2016年07月1
一
三
五
日
二
四
六
项目经理
各模块开发人员
项目经理
各模块开发人员
测试团队
项目经理,测试团队,各模块开发人员
文档编写,项目经理


目.mpp - Project Professional
登录
自动安排
检查
移动
模式
任务
任务
摘要
里程碑
可交付结果
插入
信息
备注
详细信息
添加到日程表
属性
滚动到任务
查找
清除
填充
编辑
2016年06月06日
2016 五月 30
2016 六月 06
2016 六月 13
2016 六月 20
2016 六月 27
2016 七月 04
2016 七月 11
完成时间
2016年07月11日
有日期的任务添加到日程表
2016年06月06日
2016年06月13日
2016年06月20日
2016年06月27日
2016年07月04日
2016年07月11日
2016年07月18
一 三 五 日 二 四 六
项目经理
各模块开发人员
测试团队
项目经理,测试团队,各模块开发人员
文档编写,项目经理
文档编写
软工项目小组

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5433203>

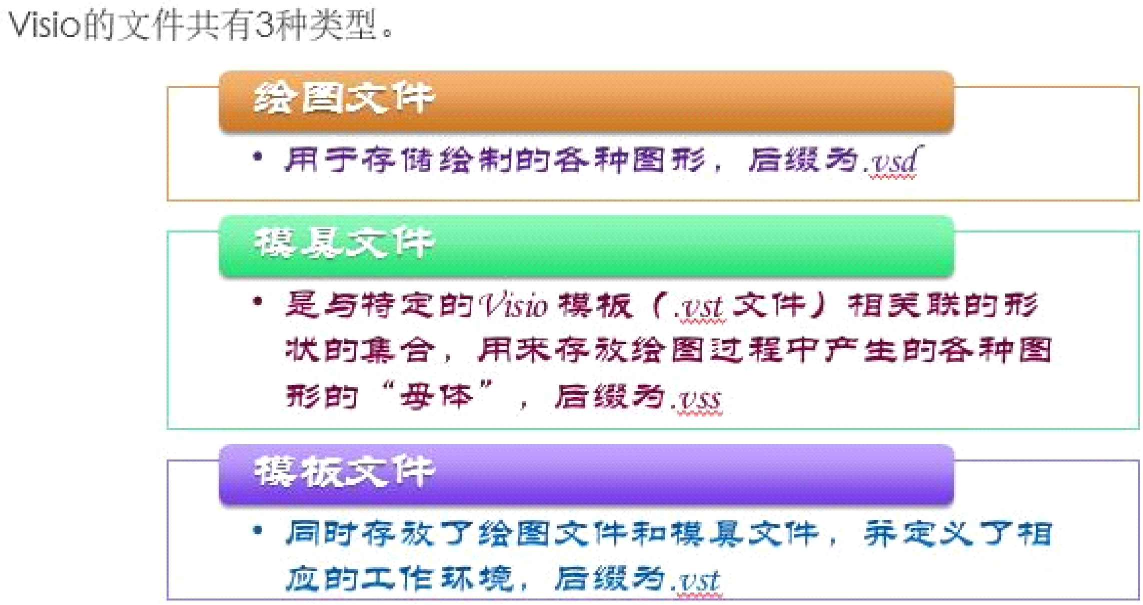
Visio作图

2016年5月6日　22:36

1.Microsoft Visio介绍

Visio是一款便于**IT**和商务专业人员就复杂信息、系统和流程进行可视化处理、分析和交流的软件，也是**Microsoft Office**办公软件家族中的一个绘图工具软件。

2.Visio的基本使用



Visio的文件共有3种类型。

绘图文件

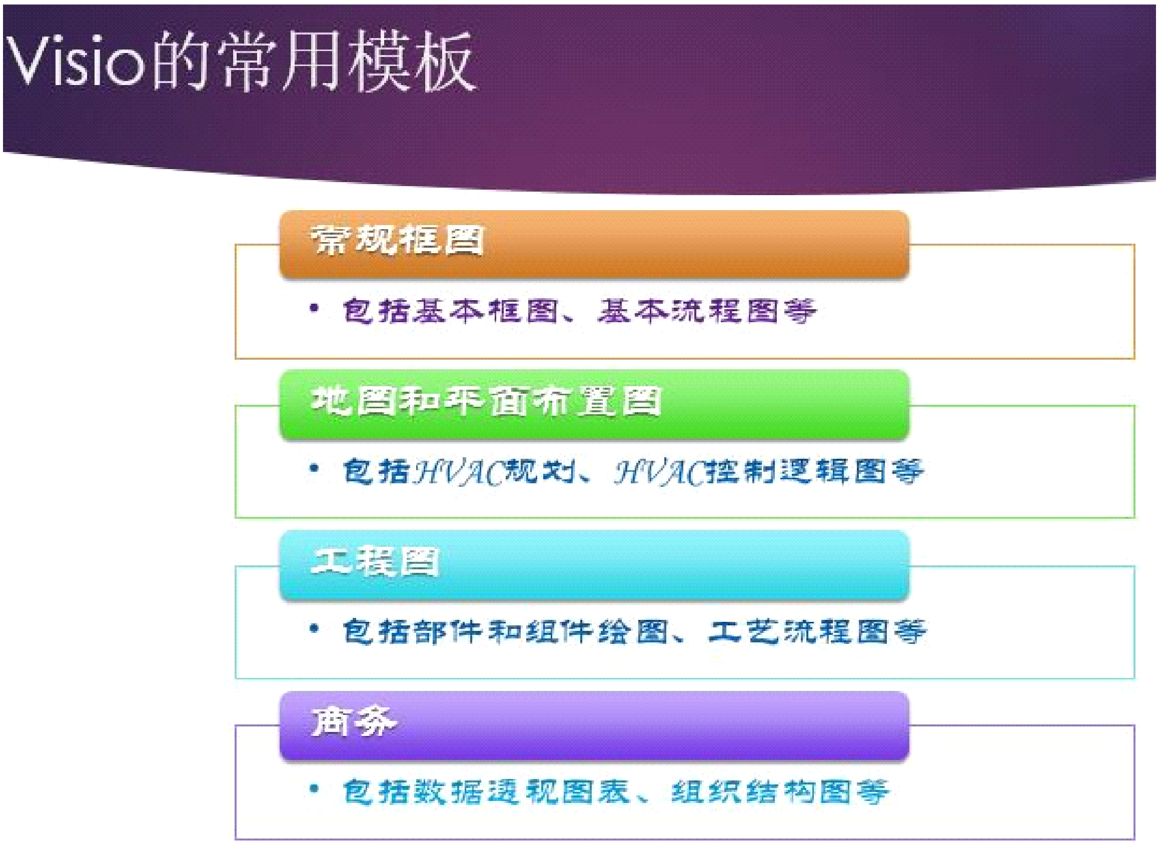
- 用于存储绘制的各种图形，后缀为.vsd

模具文件

- 是与特定的Visio模板（.vst文件）相关联的形状的集合，用来存放绘图过程中产生的各种图形的“母体”，后缀为.vss

模板文件

- 同时存放了绘图文件和模具文件，并定义了相应的工作环境，后缀为.vst



Visio的常用模板

常规框图

- 包括基本框图、基本流程图等

地图和平面布置图

- 包括HVAC规划、HVAC控制逻辑图等

工程图

- 包括部件和组件绘图、工艺流程图等

商务

- 包括数据透视图表、组织结构图等

Visio的常用模板

流程图

- 包括工作流程图、基本流程图、跨职能流程图、数据流图表、SDL图、IDEF0图表

日程安排

- 包括PERT图表、甘特图等

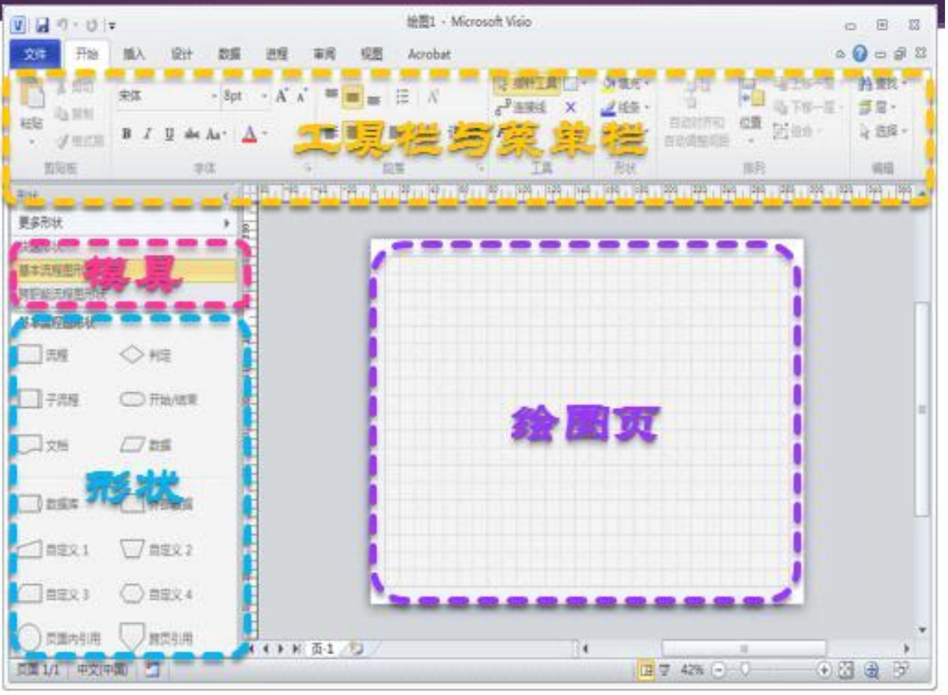
软件和数据库

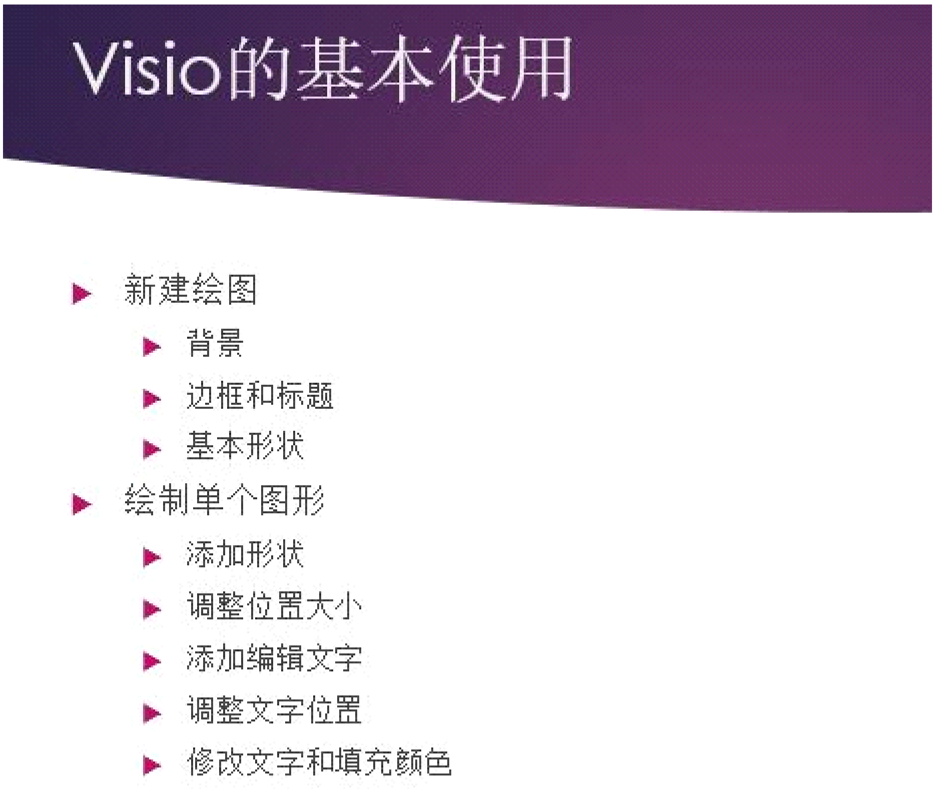
- 包括UML模型图、数据库模型图、Jason图、ORM图表、程序结构、数据流模型图、网站图、网站总体设计

网络

- 包括基本网络图、网站图、详细网络图、机架图等

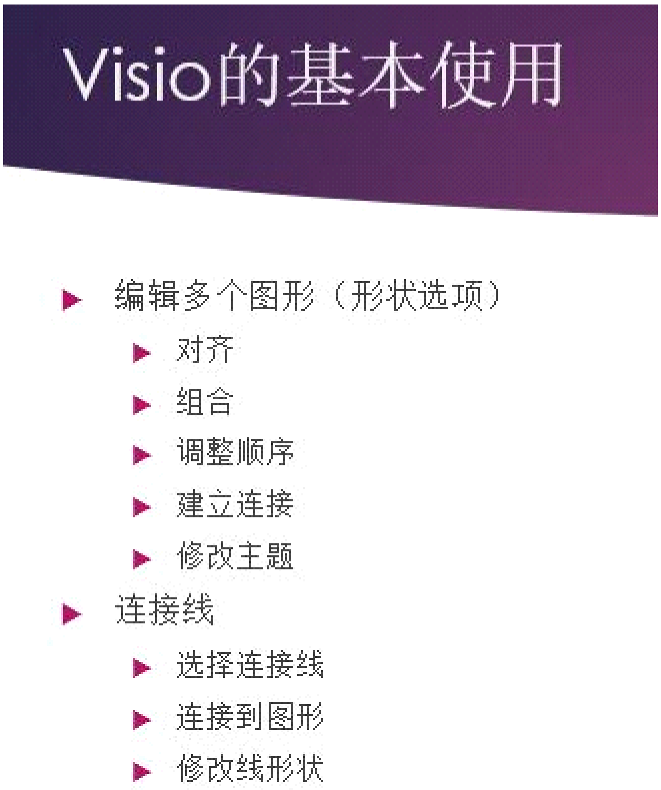
Visio的基本使用





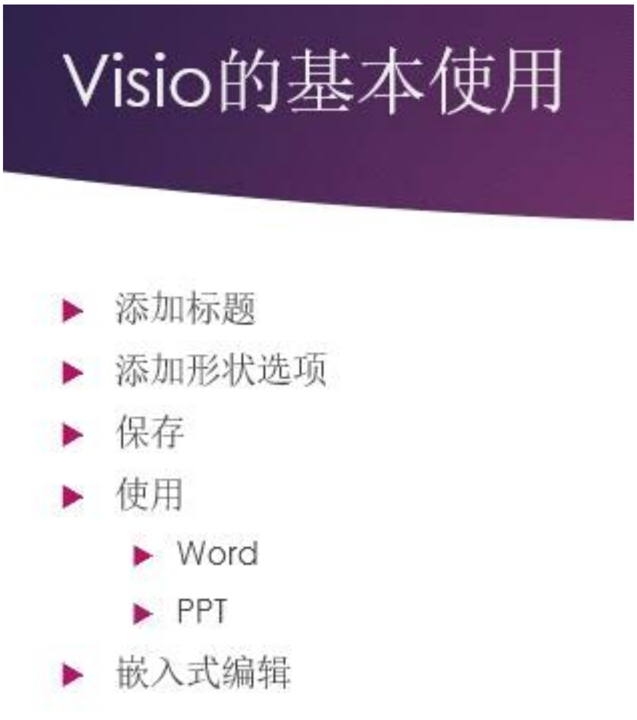
Visio的基本使用

- 新建绘图
  - 背景
  - 边框和标题
  - 基本形状
- 绘制单个图形
  - 添加形状
  - 调整位置大小
  - 添加编辑文字
  - 调整文字位置
  - 修改文字和填充颜色



Visio的基本使用

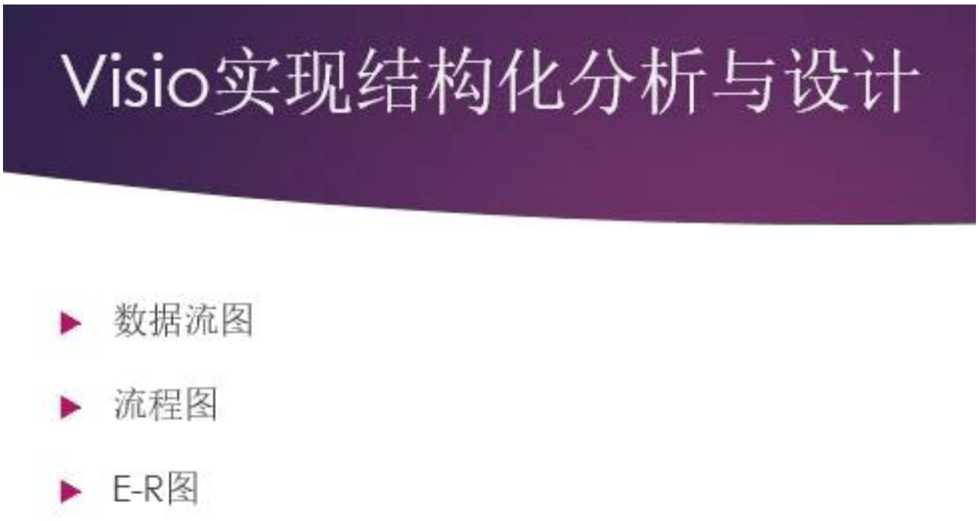
- 编辑多个图形（形状选项）
  - 对齐
  - 组合
  - 调整顺序
  - 建立连接
  - 修改主题
- 连接线
  - 选择连接线
  - 连接到图形
  - 修改线形状



Visio的基本使用

- 添加标题
- 添加形状选项
- 保存
- 使用
  - Word
  - PPT
- 嵌入式编辑

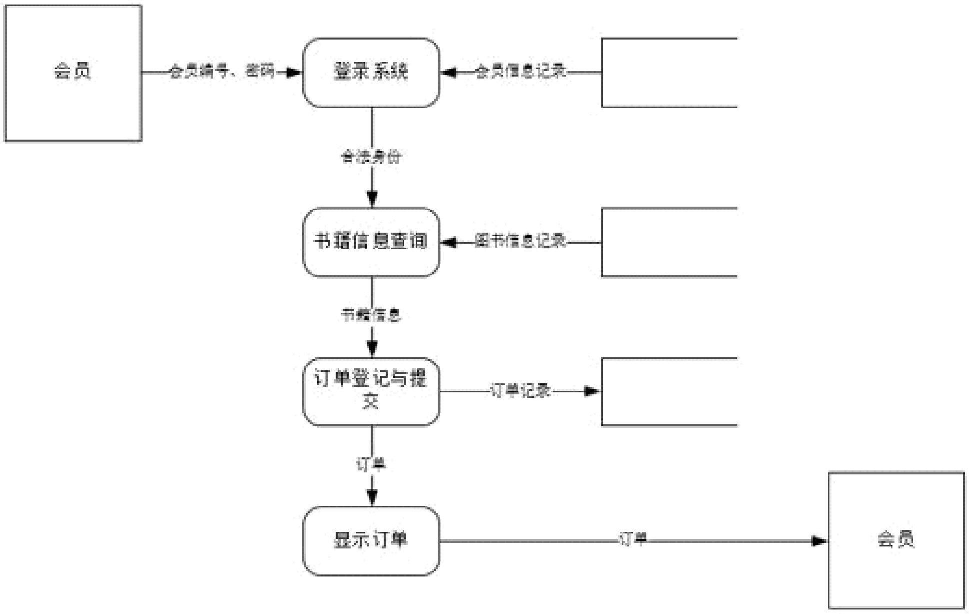
3.Visio实现结构化分析与设计



Visio实现结构化分析与设计

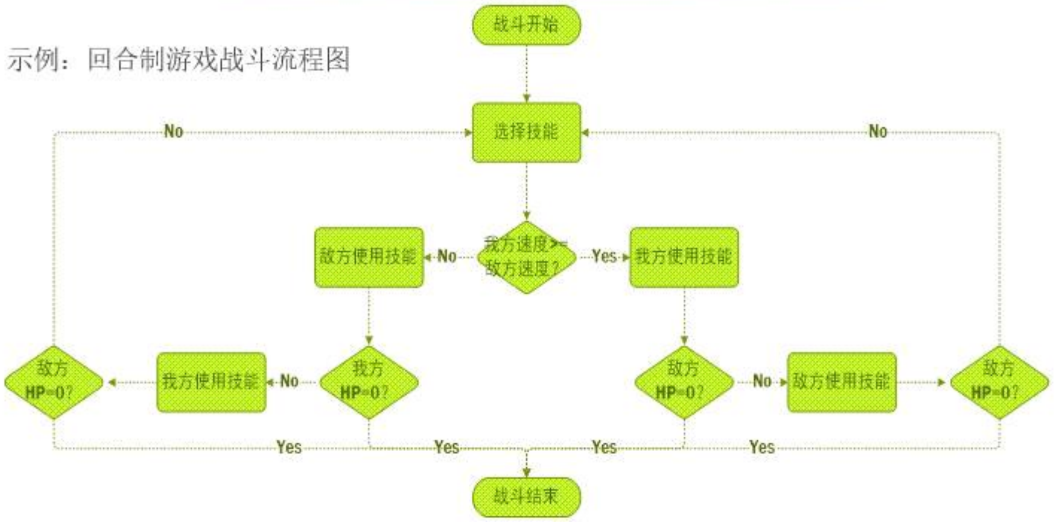
- 数据流图
- 流程图
- E-R图

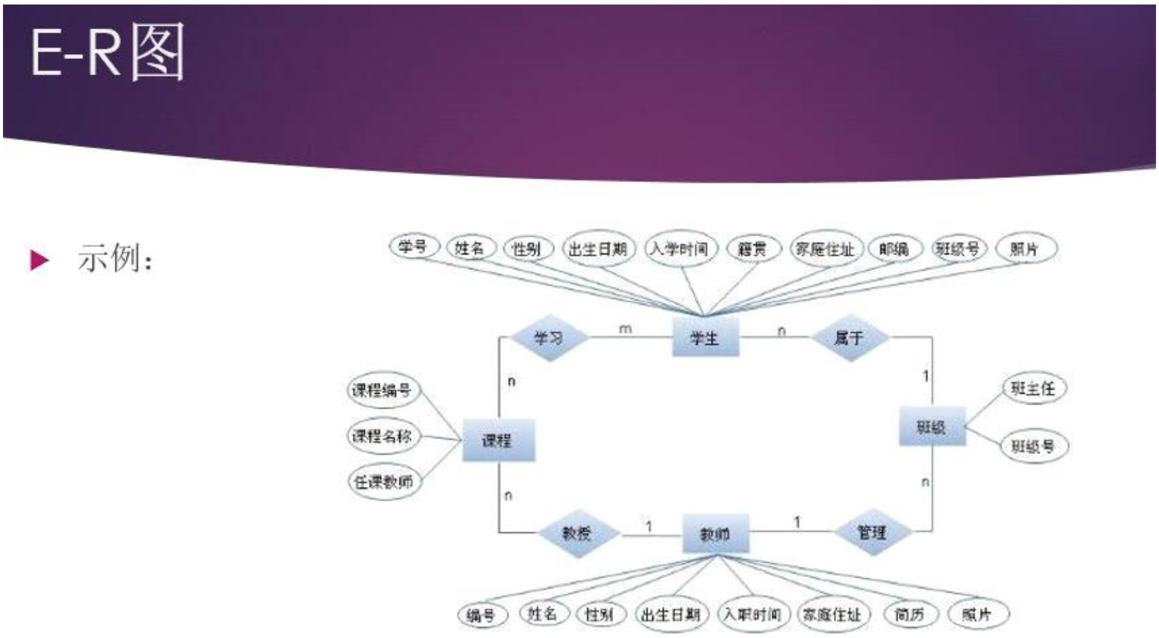
数据流图



流程图

- 示例：回合制游戏战斗流程图

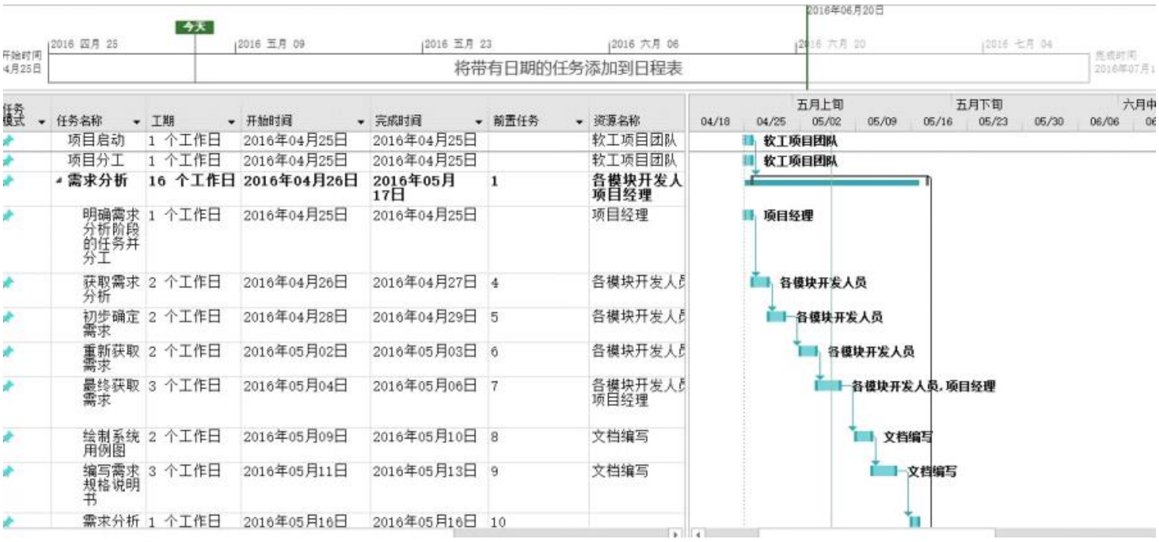




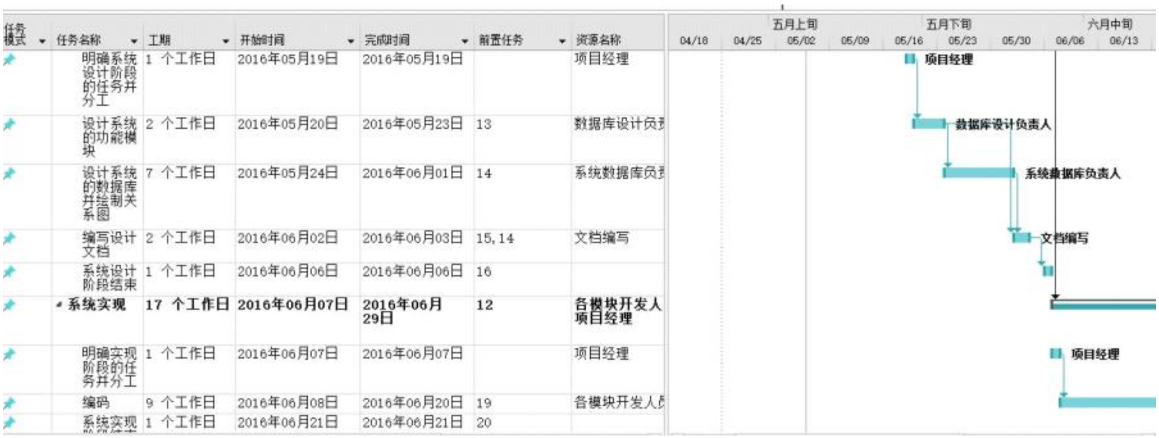
E-R图

- 示例：

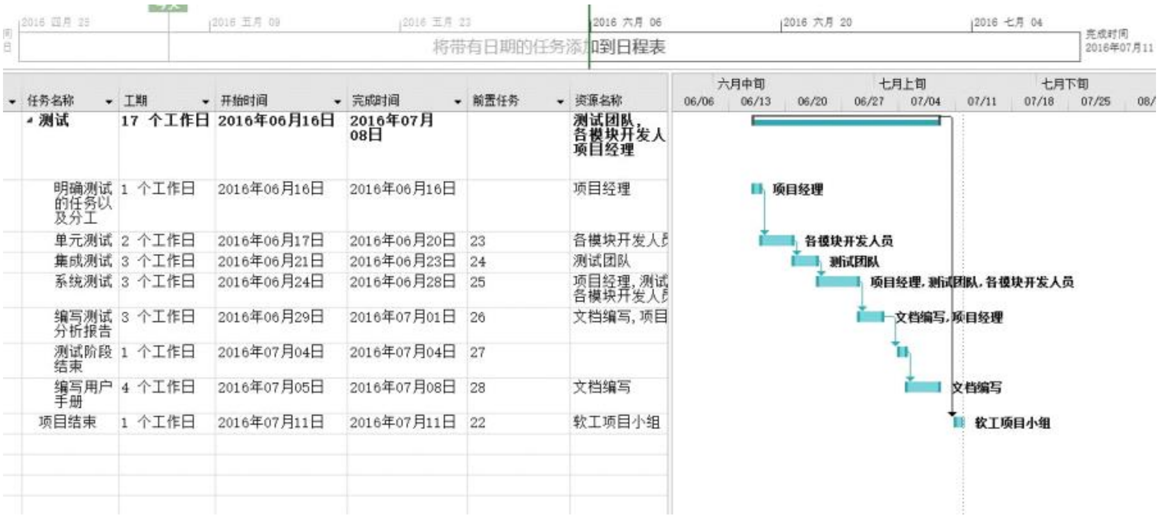
甘特图：



任务模式	任务名称	工期	开始时间	完成时间	前置任务	资源名称
📌	项目启动	1 个工作日	2016年04月25日	2016年04月25日		软工项目团队
📌	项目分工	1 个工作日	2016年04月25日	2016年04月25日		软工项目团队
📌	**需求分析**	**16 个工作日**	**2016年04月26日**	**2016年05月17日**	**1**	**各模块开发人 项目经理**
📌	明确需求分析阶段的任务并分工	1 个工作日	2016年04月25日	2016年04月25日		项目经理
📌	获取需求分析	2 个工作日	2016年04月26日	2016年04月27日	4	各模块开发人员
📌	初步确定需求	2 个工作日	2016年04月28日	2016年04月29日	5	各模块开发人员
📌	重新获取需求	2 个工作日	2016年05月02日	2016年05月03日	6	各模块开发人员
📌	最终获取需求	3 个工作日	2016年05月04日	2016年05月06日	7	各模块开发人员 项目经理
📌	绘制系统用例图	2 个工作日	2016年05月09日	2016年05月10日	8	文档编写
📌	编写需求规格说明书	3 个工作日	2016年05月11日	2016年05月13日	9	文档编写
📌	需求分析	1 个工作日	2016年05月16日	2016年05月16日	10	

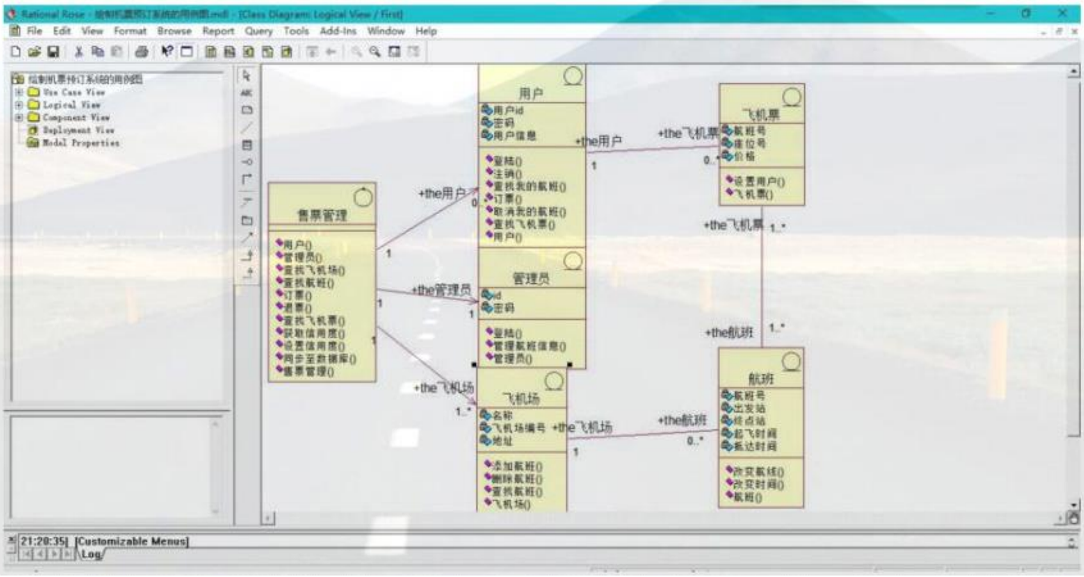


任务模式	任务名称	工期	开始时间	完成时间	前置任务	资源名称
📌	明确系统设计阶段的任务并分工	1 个工作日	2016年05月19日	2016年05月19日		项目经理
📌	设计系统的功能模块	2 个工作日	2016年05月20日	2016年05月23日	13	数据库设计负责
📌	设计系统的数据库并绘制关系图	7 个工作日	2016年05月24日	2016年06月01日	14	系统数据库负责
📌	编写设计文档	2 个工作日	2016年06月02日	2016年06月03日	15,14	文档编写
📌	系统设计阶段结束	1 个工作日	2016年06月06日	2016年06月06日	16	
📌	**系统实现**	**17 个工作日**	**2016年06月07日**	**2016年06月29日**	**12**	**各模块开发人 项目经理**
📌	明确实现阶段的任务并分工	1 个工作日	2016年06月07日	2016年06月07日		项目经理
📌	编码	9 个工作日	2016年06月08日	2016年06月20日	19	各模块开发人员
📌	系统实现	1 个工作日	2016年06月21日	2016年06月21日	20	



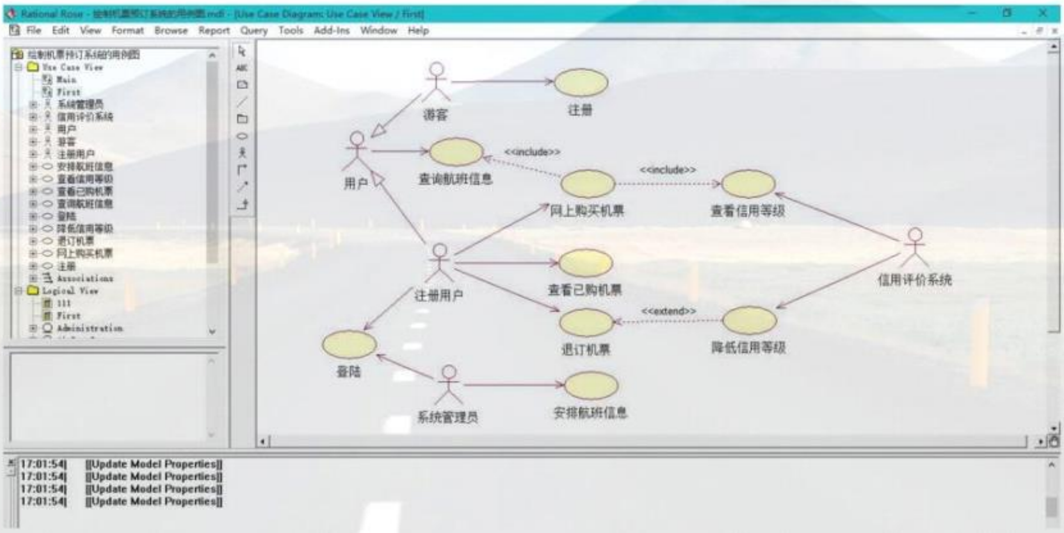
任务名称	工期	开始时间	完成时间	前置任务	资源名称
测试	**17 个工作日**	**2016年06月16日**	**2016年07月08日**		**测试团队, 各模块开发人项目经理**
明确测试的任务以及分工	1 个工作日	2016年06月16日	2016年06月16日		项目经理
单元测试	2 个工作日	2016年06月17日	2016年06月20日	23	各模块开发人员
集成测试	3 个工作日	2016年06月21日	2016年06月23日	24	测试团队
系统测试	3 个工作日	2016年06月24日	2016年06月28日	25	项目经理, 测试各模块开发人员
编写测试分析报告	3 个工作日	2016年06月29日	2016年07月01日	26	文档编写, 项目
测试阶段结束	1 个工作日	2016年07月04日	2016年07月04日	27	
编写用户手册	4 个工作日	2016年07月05日	2016年07月08日	28	文档编写
项目结束	1 个工作日	2016年07月11日	2016年07月11日	22	软工项目小组

绘制机票预订系统的类图：



```
21:20:35| [Customizable Menus]
```

绘制机票预订系统的用例图

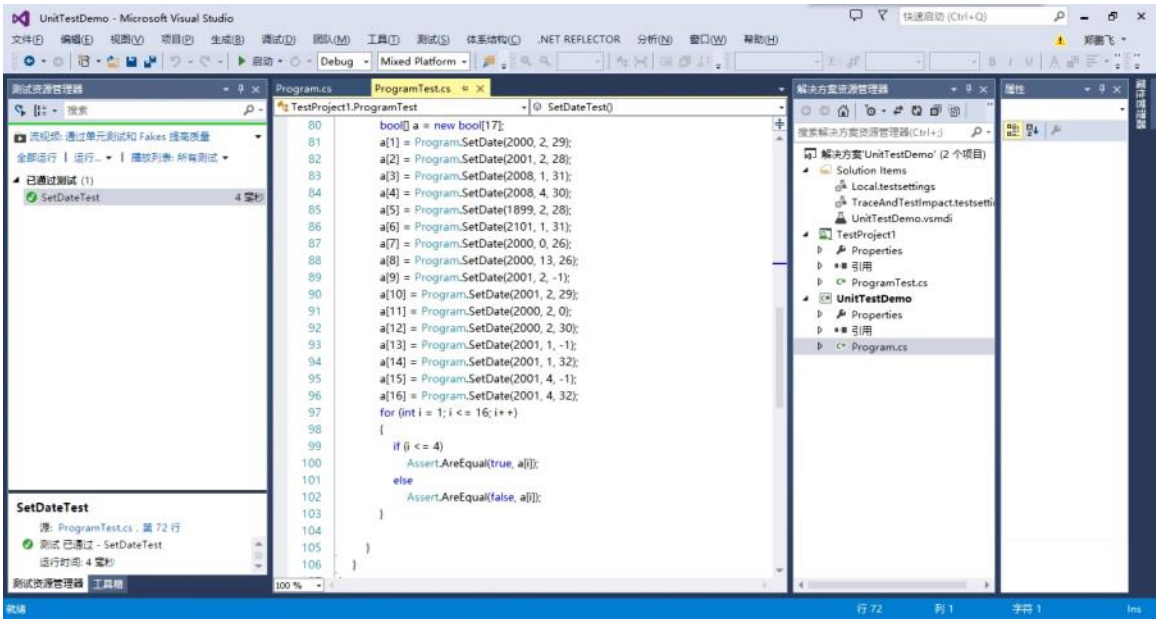


```
17:01:54|   [Update Model Properties]
17:01:54|   [Update Model Properties]
17:01:54|   [Update Model Properties]
17:01:54|   [Update Model Properties]
```

等价类划分法测试

输入及条件	有效等价类	等价类编号	无效等价类	等价类编号
year	1900<=year<=2100	1	year<1900	7
			Year>2100	8
month	1<=month<=12	2	month<1	9
			month>12	10
非闰年的 2 月	1<=day<=28	3	day<1	11
			day>28	12
闰年的 2 月	1<=day<=29	4	day<1	13
			day>29	14
month ϵ {1,3,5,7,8,10,12}	1<=day<=31	5	day<1	15
			day>31	16
month ϵ {4,6,9,11}	1<=day<=30	6	day<1	17
			day>30	18

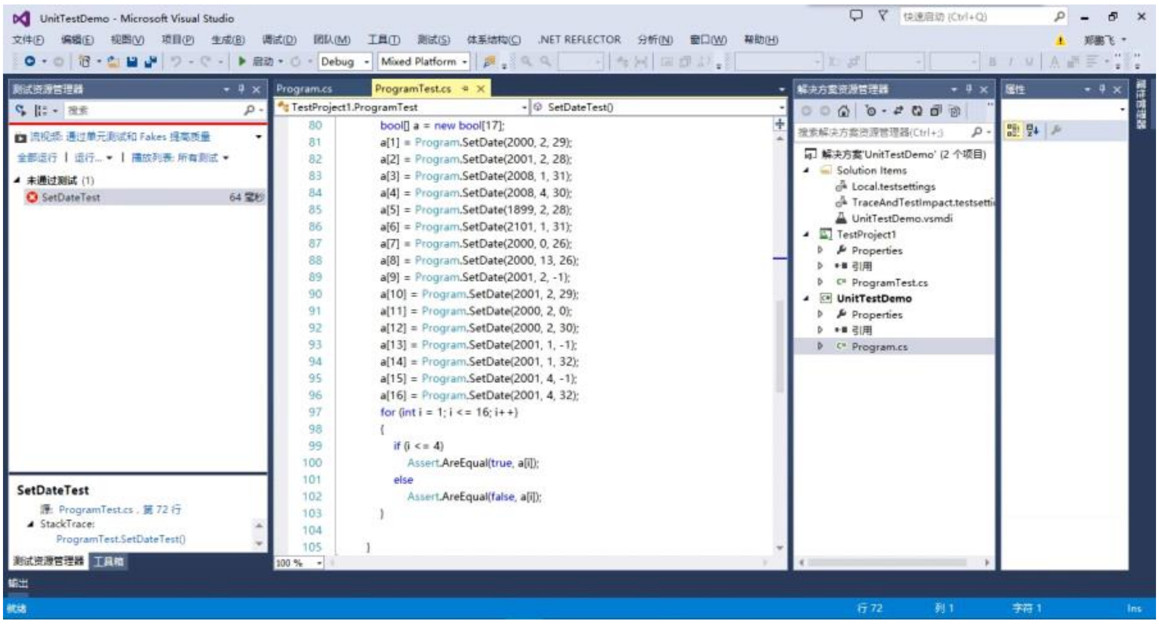
测试序号	输入数据			预期输出	覆盖范围
	year	month	day		
1	2000	2	29	True	1,2,4
2	2001	2	28	True	1,2,3
3	2008	1	31	Ture	1,2,5
4	2008	4	30	True	1,2,6
5	1899	2	28	False	7
6	2101	1	31	False	8
7	2000	0	26	False	9
8	2000	13	26	False	10
9	2001	2	-1	False	11
10	2001	2	29	False	12
11	2000	2	0	False	13
12	2000	2	30	False	14
13	2001	1	-1	False	15
14	2001	1	32	False	16
15	2001	4	-1	False	17
16	2001	4	32	False	18



SetDateTest
源: ProgramTest.cs，第 72 行
测试 已通过 - SetDateTest
运行时间: 4 毫秒

```
            bool[] a = new bool[17];
            a[1] = Program.SetDate(2000, 2, 29);
            a[2] = Program.SetDate(2001, 2, 28);
            a[3] = Program.SetDate(2008, 1, 31);
            a[4] = Program.SetDate(2008, 4, 30);
            a[5] = Program.SetDate(1899, 2, 28);
            a[6] = Program.SetDate(2101, 1, 31);
            a[7] = Program.SetDate(2000, 0, 26);
            a[8] = Program.SetDate(2000, 13, 26);
            a[9] = Program.SetDate(2001, 2, -1);
            a[10] = Program.SetDate(2001, 2, 29);
            a[11] = Program.SetDate(2000, 2, 0);
            a[12] = Program.SetDate(2000, 2, 30);
            a[13] = Program.SetDate(2001, 1, -1);
            a[14] = Program.SetDate(2001, 1, 32);
            a[15] = Program.SetDate(2001, 4, -1);
            a[16] = Program.SetDate(2001, 4, 32);
            for (int i = 1; i <= 16; i++)
            {
                if (i <= 4)
                    Assert.AreEqual(true, a[i]);
                else
                    Assert.AreEqual(false, a[i]);
            }

        }
    }
```


UnitTestDemo - Microsoft Visual Studio

未通过测试 (1)

SetDateTest 64 毫秒

SetDateTest

源: ProgramTest.cs，第 72 行

StackTrace:

ProgramTest.SetDateTest()

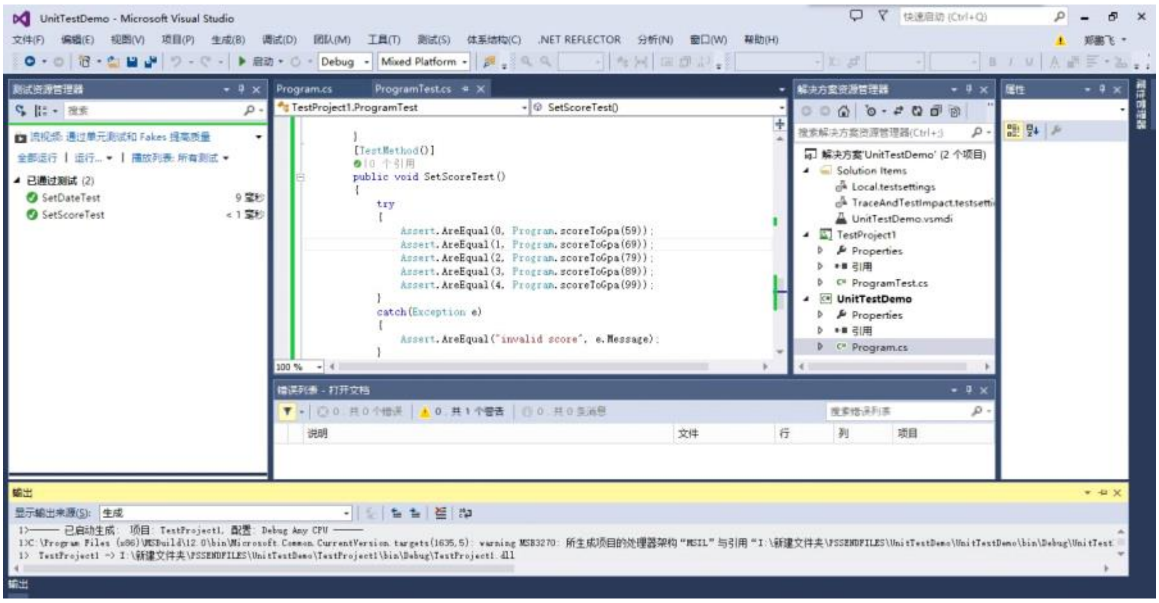
TestProject1.ProgramTest — SetDateTest()

```
bool[] a = new bool[17];
a[1] = Program.SetDate(2000, 2, 29);
a[2] = Program.SetDate(2001, 2, 28);
a[3] = Program.SetDate(2008, 1, 31);
a[4] = Program.SetDate(2008, 4, 30);
a[5] = Program.SetDate(1899, 2, 28);
a[6] = Program.SetDate(2101, 1, 31);
a[7] = Program.SetDate(2000, 0, 26);
a[8] = Program.SetDate(2000, 13, 26);
a[9] = Program.SetDate(2001, 2, -1);
a[10] = Program.SetDate(2001, 2, 29);
a[11] = Program.SetDate(2000, 2, 0);
a[12] = Program.SetDate(2000, 2, 30);
a[13] = Program.SetDate(2001, 1, -1);
a[14] = Program.SetDate(2001, 1, 32);
a[15] = Program.SetDate(2001, 4, -1);
a[16] = Program.SetDate(2001, 4, 32);
for (int i = 1; i <= 16; i++)
{
    if (i <= 4)
        Assert.AreEqual(true, a[i]);
    else
        Assert.AreEqual(false, a[i]);
}

}
```

分支覆盖法测试

测试序号	输入	覆盖分支
1	59	1
2	69	2,3
3	79	2,4,5
4	89	2,4,6,7
5	99	2,4,6,8


UnitTestDemo - Microsoft Visual Studio

已通过测试 (2)

SetDateTest 9 毫秒

SetScoreTest < 1 毫秒

TestProject1.ProgramTest — SetScoreTest()

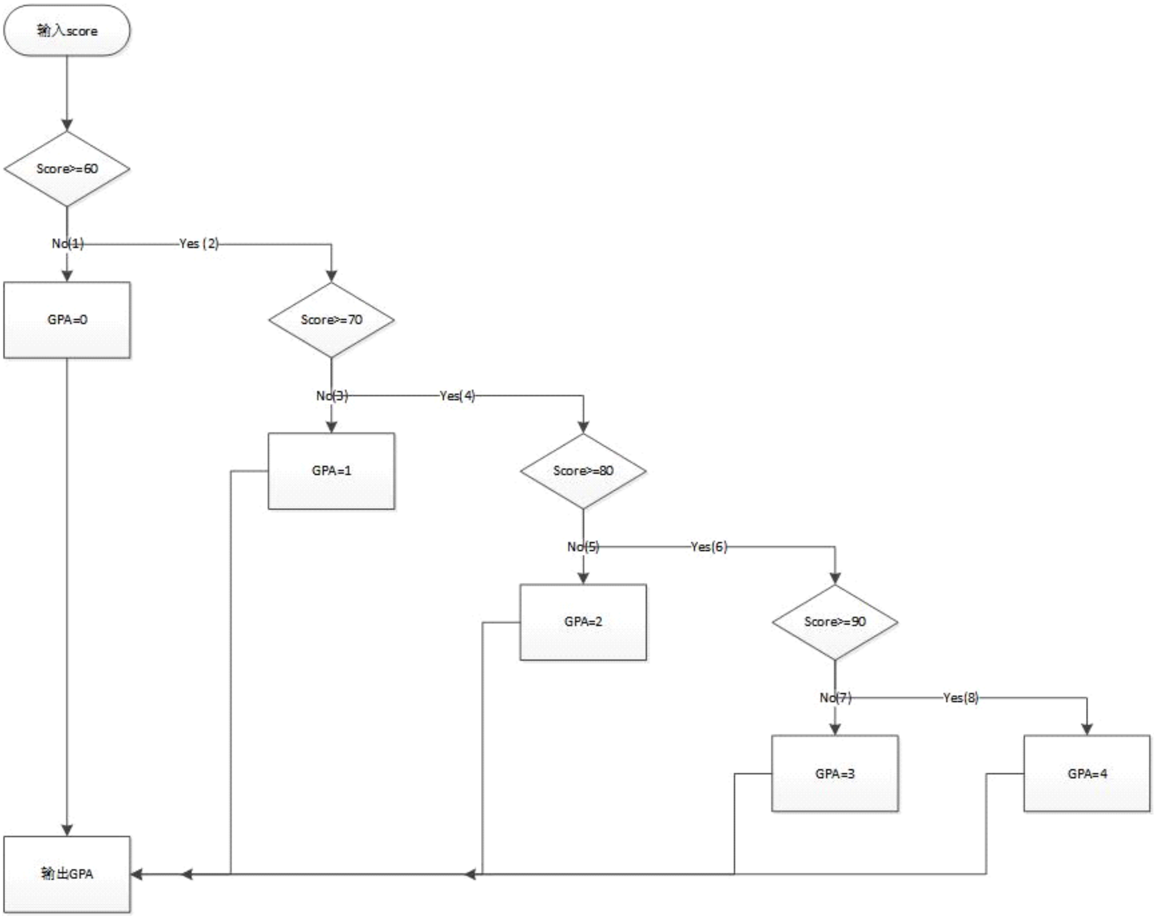
```
}
[TestMethod()]
0 个引用
public void SetScoreTest()
{
    try
    {
        Assert.AreEqual(0, Program.scoreToGpa(59));
        Assert.AreEqual(1, Program.scoreToGpa(69));
        Assert.AreEqual(2, Program.scoreToGpa(79));
        Assert.AreEqual(3, Program.scoreToGpa(89));
        Assert.AreEqual(4, Program.scoreToGpa(99));
    }
    catch(Exception e)
    {
        Assert.AreEqual("invalid score", e.Message);
    }
```

错误列表 - 打开文档

0 共 0 个错误 | 0 共 1 个警告 | 0 共 0 条消息

输出 — 显示输出来源(S): 生成

```
1>------ 已启动生成: 项目: TestProject1, 配置: Debug Any CPU ------
1>C:\Program Files (x86)\MSBuild\12.0\bin\Microsoft.Common.CurrentVersion.targets(1635,5): warning MSB3270: 所生成项目的处理器架构“MSIL”与引用“I:\新建文件夹\PSSENDFILES\UnitTestDemo\UnitTestDemo\bin\Debug\UnitTest
1>  TestProject1 -> I:\新建文件夹\PSSENDFILES\UnitTestDemo\TestProject1\bin\Debug\TestProject1.dll
```


输入score
Score>=60
No(1)
Yes (2)
GPA=0
Score>=70
No(3)
Yes(4)
GPA=1
Score>=80
No(5)
Yes(6)
GPA=2
Score>=90
No(7)
Yes(8)
GPA=3
GPA=4
输出GPA

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5372917>

UVa 108 - Maximum Sum（最大连续子序列）

2016年5月6日 22:36

题目来源：https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=3&page=show_problem&problem=44

Maximum Sum

Background

A problem that is simple to solve in one dimension is often much more difficult to solve in more than one dimension. Consider satisfying a boolean expression in conjunctive normal form in which each conjunct consists of exactly 3 disjuncts. This problem (3-SAT) is NP-complete. The problem 2-SAT is solved quite efficiently, however. In contrast, some problems belong to the same complexity class regardless of the dimensionality of the problem.

The Problem

Given a 2-dimensional array of positive and negative integers, find the sub-rectangle with the largest sum. The sum of a rectangle is the sum of all the elements in that rectangle. In this problem the sub-rectangle with the largest sum is referred to as the *maximal sub-rectangle*. A sub-rectangle is any contiguous sub-array of size

1×1

or greater located within the whole array. As an example, the maximal sub-rectangle of the array:

$$\begin{array}{rrrr} 0 & -2 & -7 & 0 \\ 9 & 2 & -6 & 2 \\ -4 & 1 & -4 & 1 \\ -1 & 8 & 0 & -2 \end{array}$$

is in the lower-left-hand corner:

$$\begin{matrix} 9 & 2 \\ -4 & 1 \\ -1 & 8 \end{matrix}$$

and has the sum of 15.

Input and Output

The input consists of an $N \times N$ array of integers. The input begins with a single positive integer *N* on a line by itself indicating the size of the square two dimensional array. This is followed by N^2 integers separated by white-space (newlines and spaces). These N^2 integers make up the array in row-major order (i.e., all numbers on the first row, left-to-right, then all numbers on the second row, left-to-right, etc.). *N* may be as large as 100. The numbers in the array will be in the range [-127, 127].

The output is the sum of the maximal sub-rectangle.

Sample Input

```
4
0 -2 -7  0 9  2 -6  2
-4  1 -4  1 -1
8  0 -2
```

Sample Output

```
15
```

解题思路：

题意：给出n*n的矩阵，求出里面子矩阵的和的最大值。

最大连续子序列的应用，序列是一维的，矩阵是二维的，所以我们可以把矩阵转换为一维的来算。

也就是枚举矩阵的连续几行的合并，这样就转换为一维的了，再用最大子序列的算法去求，更新最大值就可以了。

代码：

```cpp
#include <bits/stdc++.h>

using namespace std;

int table[100][100];
int sum[100];
int N;

int max_continuous_sum()
{
    int maxs=0,s=0;
    for(int i=0; i<N; i++)
    {
        if(s>=0) s+=sum[i];
        else s=sum[i];
        maxs = maxs>s ? maxs : s;
    }
    return maxs;
}
int main()
{
    cin >> N;
    int maxsum=0;
    int tmp;
    for(int i=0; i<N; i++)
    {
        for(int j=0; j<N; j++)
        {
            cin >> table[i][j];
            sum[j]=table[i][j];
        }
        tmp = max_continuous_sum();
        maxsum = maxsum>tmp ? maxsum : tmp;
        for(int j=i-1; j>=0; j--)
        {
            for(int k=0; k<N; k++)
                sum[k]+=table[j][k];
            tmp = max_continuous_sum();
            maxsum = maxsum>tmp ? maxsum : tmp;
        }
    }
    cout << maxsum << endl;
    return 0;
}
```

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5049457>

UVa 107 - The Cat in the Hat （找规律,注意精度）

2016年5月6日 22:37

题目来源：https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=3&page=show_problem&problem=43

The Cat in the Hat

Background

(An homage to Theodore Seuss Geisel)

The Cat in the Hat is a nasty creature,
But the striped hat he is wearing has a rather nifty feature.

With one flick of his wrist he pops his top off.

Do you know what's inside that Cat's hat?
A bunch of small cats, each with its own striped hat.

Each little cat does the same as line three,
All except the littlest ones, who just say ``Why me?''

Because the littlest cats have to clean all the grime,
And they're tired of doing it time after time!

The Problem

A clever cat walks into a messy room which he needs to clean. Instead of doing the work alone, it decides to have its helper cats do the work. It keeps its (smaller) helper cats inside its hat. Each helper cat also has helper cats in its own hat, and so on. Eventually, the cats reach a smallest size. These smallest cats have no additional cats in their hats. These unfortunate smallest cats have to do the cleaning.

The number of cats inside each (non-smallest) cat's hat is a constant, *N*. The height of these cats-in-a-hat is

$\frac{1}{N+1}$

times the height of the cat whose hat they are in.

The smallest cats are of height one;

these are the cats that get the work done.

All heights are positive integers.

Given the height of the initial cat and the number of worker cats (of height one), find the number of cats that are not doing any work (cats of height greater than one) and also determine the sum of all the cats' heights (the height of a stack of all cats standing one on top of another).

The Input

The input consists of a sequence of cat-in-hat specifications. Each specification is a single line consisting of two positive integers, separated by white space. The first integer is the height of the initial cat, and the second integer is the number of worker cats.

A pair of 0's on a line indicates the end of input.

The Output

For each input line (cat-in-hat specification), print the number of cats that are not working, followed by a space, followed by the height of the stack of cats. There should be one output line for each input line other than the ``0 0'' that terminates input.

Sample Input

```
216 125
5764801 1679616
0 0
```

Sample Output

```
31 671
335923 30275911
```

解题思路：

帽子里的猫，

此题给的条件是第一支猫的高度h和最后的working cats的数量w.

下面分析猫的高度和猫的数量之间的关系：

设开始的猫的高度是 h , N 是每个帽子里面猫的数量，设一个中间变量s。

猫的高度	h	h / (N + 1)	$h / (N + 1)^2$	…………	$h / (N + 1)^s$
猫的数量	1	N	N * N	…………	N^s

当最后工作的猫的个数为w时：

lg（n）/lg(n+1)=lg(num)/lg(hgiht)

是用二分查找算出N即可，条件就写成了fabs（lg（n）*lg(height)-lg(num)/lg(n+1)<0.00001)

下面贴代码：

```
#include <bits/stdc++.h>

using namespace std;

int main()
{
    int h,num;
    while(~scanf("%d%d",&h,&num))
    {
        if(0==h&&0==num)
            break;
        int left=1,right=10000000,mid;
        while(left)
        {
            mid=(left+right)/2;
            if(fabs(log(mid)*log(h)-log(mid+1)*log(num))<=0.000001) break;
            if(log(mid)*log(h)-log(mid+1)*log(num)>0)
                right=mid;
            else
                left=mid;
        }
        int x=0,y=h;
        left=1;
        while(h>1)
        {
            h/=(1+mid);
            left*=mid;
            x+=left;
            y+=h*left;
        }
        printf("%d %d\n",x-num+1,y);
    }
}
```

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5049429>

UVa 106 - Fermat vs Pythagoras（数论题目）

2016年5月6日 22:37

题目来源：https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=3&page=show_problem&problem=42

Fermat vs. Pythagoras

Background

Computer generated and assisted proofs and verification occupy a small niche in the realm of Computer Science. The first proof of the four-color problem was completed with the assistance of a computer program and current efforts in verification have succeeded in verifying the translation of high-level code down to the chip level.

This problem deals with computing quantities relating to part of Fermat's Last Theorem: that there are no integer solutions of

$a^n + b^n = c^n$

for $n > 2$.

The Problem

Given a positive integer N, you are to write a program that computes two quantities regarding the solution of

$$x^2 + y^2 = z^2$$

where x, y, and z are constrained to be positive integers less than or equal to N. You are to compute the number of triples (x,y,z) such that $x<y< z$, and they are relatively prime, i.e., have no common divisor larger than 1. You are also to compute the number of values

$0 < p \leq N$

such that *p* is not part of any triple (not just relatively prime triples).

The Input

The input consists of a sequence of positive integers, one per line. Each integer in the input file will be less than or equal to 1,000,000. Input is terminated by end-of-file.

The Output

For each integer *N* in the input file print two integers separated by a space. The first integer is the number of relatively prime triples (such that each component of the triple is

$\leq N$

). The second number is the number of positive integers

$\leq N$

that are not part of any triple whose components are all

$\leq N$

. There should be one output line for each input line.

Sample Input

```
10
25
100
```

Sample Output

```
1 4
4 9
16 27
```

解题思路：

这是一道数论题，用数学的语言描述就是：$x, y, z \in N$，给定一个数n，找出所有的$x, y, z \leq n$，使得$x^2 + y^2 = z^2$成立。如果要穷举所有的x, y, z的话，按照题目所给的数据量，肯定是无法在限定时间内完成的。考虑利用毕达哥拉斯数的性质生成所有的x, y, z来解决，数学推导简要介绍如下：

先假定x, y, z两两互质，由于x, y互质，故x, y中至少有1个是奇数。下面用反证法证明x和y中有且只有1个奇数。假定x, y都为奇数，设：

- $x = 2a + 1$
- $y = 2b + 1$
- $x^2 + y^2 = (2a + 1)^2 + (2b + 1)^2$
$= 4(a^2 + b^2 + a + b) + 2$

又因为x^2和y^2是奇数，则z^2是偶数，且必能被4整除，与上式矛盾，因此x, y中只有一个奇数。

假设x为奇数，y为偶数，则z为奇数，2z与2x的最大公因数为2，2z和2x可分别写作

- $2z = (z + x) + (z - x)$
- $2x = (z + x) - (z - x)$

那么跟据最大公因数性质，$z + x$和$z - x$的最大公因数也为2，又因为：

- $(z + x)(z - x) = y^2$，两边同除以4得：
$((z + x) / 2)((z - x) / 2) = (y / 2)^2$

故可令：

- $z + x = 2m^2, z - x = 2n^2$
其中$z = m + n, x = m - n$（m与n互质）

则有：

- $y^2 = z^2 - x^2 = 2m^2 2n^2 = 4m^2 n^2$
即$y = 2mn$。

综上所述，可得到下式：

- $x = m^2 - n^2, y = 2mn, z = m^2 + n^2$. (m, n为任意自然数)

这里还有一个问题：题目要求统计(x, y, z)三元组的数量时只统计x，y和z

两两互质的的情况，这个问题用上面的算法就可以解决了。但对于统计p的数量，题目并不限定三元组是两两互质的。但是上式不能生成所有x, y, z并不是两两互质的情况。然而假设x与y最大公因数w不为1，则z也必能被w整除，因此w为x, y, z三个数的公因数。归纳总结可知，所有非两两互质的x_0, y_0, z_0都可由一组互质的x, y, z乘以系数得到。根据以上理论就可以快速的求解了。

参考代码:

```
#include <cstdio>
#include <cmath>
#include <cstring>
#define N 1000010
bool used[N];

long long gcd(long long a , long long b)
{ return b==0 ? a: gcd(b,a%b); }

int main()
{
    long long n,a,b,c;
    long long count1,count2;
    while(scanf("%lld",&n)!=EOF)
    {
        count1=count2=0;
        memset(used,0,sizeof(used));
        long long m=(long long)sqrt(n+0.5);
        for(long long t=1; t<=m; t+=2)
            for(long long s=t+2; s*t<=n; s+=2)
                if(gcd(s,t)==1)  //s>t>=1且s与t互质
                {
                    a=s*t;          //奇数
                    b=(s*s-t*t)/2;  //偶数
                    c=(s*s+t*t)/2;  //奇数
                    if(c<=n)        //在n范围内的PPT
                    {
                        count1++;
                        //printf("本原勾股数组: %lld %lld %lld\n",a,b,c);
                        if(!used[a]) { count2++; used[a]=1; }
                        if(!used[b]) { count2++; used[b]=1; }
                        if(!used[c]) { count2++; used[c]=1; }

                        for(int j=2; c*j<=n; j++)  //j是倍数
                        {
                            if(!used[a*j]) { count2++; used[a*j]=1; }
                            if(!used[b*j]) { count2++; used[b*j]=1; }
                            if(!used[c*j]) { count2++; used[c*j]=1; }
                        }
                    }
                }
            printf("%lld %lld\n",count1,n-count2);
    }
    return 0;
}
```

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5049407>

UVa 111 - History Grading (by 最长公共子序列)

2016年5月6日　　22:37

History Grading

Background

Many problems in Computer Science involve maximizing some measure according to constraints.

Consider a history exam in which students are asked to put several historical events into chronological order. Students who order all the events correctly will receive full credit, but how should partial credit be awarded to students who incorrectly rank one or more of the historical events?

Some possibilities for partial credit include:

1. 1 point for each event whose rank matches its correct rank
2. 1 point for each event in the longest (not necessarily contiguous) sequence of events which are in the correct order relative to each other.

For example, if four events are correctly ordered 1 2 3 4 then the order 1 3 2 4 would receive a score of 2 using the first method (events 1 and 4 are correctly ranked) and a score of 3 using the second method (event sequences 1 2 4 and 1 3 4 are both in the correct order relative to each other).

In this problem you are asked to write a program to score such questions using the second method.

The Problem

Given the correct chronological order of *n* events

$1, 2, \ldots, n$

as

$c_1, c_2, \ldots c_n$

where

$1 \leq c_i \leq n$

denotes the ranking of event *i* in the correct chronological order and a sequence of student responses

$r_1, r_2, \ldots r_n$

where

$1 \leq r_i \leq n$

denotes the chronological rank given by the student to event *i*; determine the length of the longest (not necessarily contiguous) sequence of events in the student responses that are in the correct chronological order relative to each other.

The Input

The first line of the input will consist of one integer *n* indicating the number of events with

$$2 \le n \le 20$$

. The second line will contain*n* integers, indicating the correct chronological order of *n* events. The remaining lines will each consist of *n* integers with each line representing a student's chronological ordering of the n events. All lines will contain *n* numbers in the range

$$[1 \ldots n]$$

, with each number appearing exactly once per line, and with each number separated from other numbers on the same line by one or more spaces.

The Output

For each student ranking of events your program should print the score for that ranking. There should be one line of output for each student ranking.

Sample Input 1

```
4
4 2 3 1
1 3 2 4
3 2 1 4
2 3 4 1
```

Sample Output 1

```
1
2
3
```

Sample Input 2

```
10
3 1 2 4 9 5 10 6 8 7
1 2 3 4 5 6 7 8 9 10
4 7 2 3 10 6 9 1 5 8
3 1 2 4 9 5 10 6 8 7
2 10 1 3 8 4 9 5 7 6
```

Sample Output 2

```
6
5
```

10
9

解题思路:

最长公共子序列的应用.之前已经有过一些相对应的练习,不过这道题关于历史时间的时间排序.其中输入的序列并不一定代表着时间的顺序，而是代表着事件i发生的位置在哪里.

注意，给出的序列和一般理解上的出现次序是不同的。比如4 2 3 1，一般理解是第4个事件在第1位，第2个事件在第2位，第1个事件在第4位。但在这道题目，序列的含义是第1个事件在第4位，第2个事件在第2位，第4个事件在第一位。为方便计算，需要对输入的序列进行转换，使数字对号入座。比如一个正确的序列是：

	正确答案	学生答案	含义
输入的序列	3 1 2 4 9 5 10 6 8 7	2 10 1 3 8 4 9 5 7 6	历史事件发生在第几位
转换后的序列	2 3 1 4 6 8 10 9 5 7	3 1 4 6 8 10 9 5 7 2	发生了第几个历史事件

接下来就用转换后的序列进行计算。为了方便的看出学生答案的顺序是否正确，应该按照正确答案与1 2 ... 10的对应关系将学生答案再做一次转换。正确答案中第2个历史件事发生在第1位，那么学生答案中的2应转换为1；即3转换为2，以此类推。学生答案就转换为最终的序列为：

编号	1 2 3 4 5 6 7 8 9 10
序列	2 3 4 5 6 7 8 9 10 1

显而易见，这个序列的最长有序子串（非连序）的长度为9。要将输入的正确答案序列和学生答案序列都依次做上述转换就显得非常麻烦了，其实正确答案序列是无需转换的。假设正确答案序列为A，学生答案序列为B，则发生在第B_i位的历史事件的最终位置就应该是A_i。这样就可以一次完成全部转换。

接下来就是要找出最长有序子串。因为正确的顺序是由小至大，所以从最后一位开始遍例。依次找出每个数之后（含自身）的最长有序子串长度M_i，然后找最M_i中的最大值即为解。观查下面的序列：

编号	1 2 3 4 5 6 7 8 9 10
序列	5 8 3 7 6 1 9 2 10 4

因此在进行传入的参数应该是变换之后的数组。

对于如何将编号转化为序列号.很简单的操作：

```
for(int i=1; i<=n; i++)
{
    cin>>loc;//事件i发生的位置loc
    x[loc]=i;//
}
```

完整代码：

```
#include <bits/stdc++.h>
const int MAX=21;
int x[MAX];
int y[MAX];
int DP[MAX][MAX];
int b[MAX][MAX];
int n,i,j,loc;

using namespace std;

void work(int x[],int y[],int n)
{
    memset(DP,0,sizeof(DP));
    for(i=1; i<=n; i++)
    {
        for(j=1; j<=n; j++)
        {
            if(x[i]==y[j])
            {
                DP[i][j]=DP[i-1][j-1]+1;
            }
            else
                DP[i][j]=max(DP[i-1][j],DP[i][j-1]);
        }
    }
    cout<<DP[n][n]<<endl;
}

int main()
{
    scanf("%d",&n);
    for(int i=1; i<=n; i++)
    {
        cin>>loc;
        x[loc]=i;
    }
    while(~scanf("%d",&loc))//avoid TLE
    {
        y[loc]=1;
        for(int j=2; j<=n; j++)
        {
            cin>>loc;
            y[loc]=j;
        }
        work(x,y,n);
        memset(y,0,sizeof(y));
    }
    return 0;
}
```

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5049124>

UVa 105 - The Skyline Problem（利用判断，在于想法）

2016年5月6日　　22:37

题目来源：https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=3&page=show_problem&problem=41

解题思路：

当看到题目时,仔细想想感觉用暴力方法挺复杂的.需要判断建筑是否重叠,找到高度的变化位置,找出对应的左值,找出对应的右值.能这么想就很接近我要讲的解题方法了.

我们是需要找到高度转折点,找到最大的右值.

高度转折点的寻找：判断高度是否和前面的高度不同.但是这个建筑物是二维图形,如何去找建筑的分界点有些很复杂.由于题目中所给的限定,我们可以把建筑分成不连续的正整数点所围成.这样利用**for**循环,每次横坐标的值加**1**.直到高度变化之后,说明这里需要输出.(前提是我们要把这些不连续的正整数横坐标确定之后找出他们的最大的高度,利用**for**循环,横坐标每次加**1**,不断修改某一点的高度值)

下面给出代码：

```
#include<bits/stdc++.h>
#define MAX 10010

using namespace std;

int height[MAX];

int main()
{
    int i,l,h,r,max_right=-1;
    while (~scanf("%d%d%d", &l, &h, &r))
    {
        memset(height,0,sizeof(height));
        for (i=l; i<r; i++)
            height[i]=max(height[i], h); //注意右端点处不记录高度
        max_right=max(max_right,r);
    }
    for (i=1; i<max_right; i++)
        if (height[i]!=height[i-1])
            printf("%d %d ",i,height[i]);
    printf("%d 0\n", max_right);
    return 0;
}
```

推荐博客链接：

http://www.cnblogs.com/devymex/archive/2010/08/07/1794533.html

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5041502>

UVa 104 - Arbitrage（Floyd动态规划）

2016年5月6日 22:38

题目来源：https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=3&page=show_problem&problem=40

Arbitrage

Background

The use of computers in the finance industry has been marked with controversy lately as programmed trading -- designed to take advantage of extremely small fluctuations in prices -- has been outlawed at many Wall Street firms. The ethics of computer programming is a fledgling field with many thorny issues.

The Problem

Arbitrage is the trading of one currency for another with the hopes of taking advantage of small differences in conversion rates among several currencies in order to achieve a profit. For example, if $1.00 in U.S. currency buys 0.7 British pounds currency, £1 in British currency buys 9.5 French francs, and 1 French franc buys 0.16 in U.S. dollars, then an arbitrage trader can start with $1.00 and earn

$$1 \times 0.7 \times 9.5 \times 0.16 = 1.064$$

dollars thus earning a profit of 6.4 percent.

You will write a program that determines whether a sequence of currency exchanges can yield a profit as described above.

To result in successful arbitrage, a sequence of exchanges must begin and end with the same currency, but any starting currency may be considered.

The Input

The input file consists of one or more conversion tables. You must solve the arbitrage problem for each of the tables in the input file.

Each table is preceded by an integer *n* on a line by itself giving the dimensions of the table. The maximum dimension is 20; the minimum dimension is 2.

The table then follows in row major order but with the diagonal elements of the table missing (these are assumed to have value 1.0). Thus the first row of the table represents the conversion rates between country 1 and *n*-1 other countries, i.e., the amount of currency of country *i* (

$2 \leq i \leq n$

) that can be purchased with one unit of the currency of country 1.

Thus each table consists of *n*+1 lines in the input file: 1 line containing *n* and *n* lines representing the conversion table.

The Output

For each table in the input file you must determine whether a sequence of exchanges exists that results in a profit of more than 1 percent (0.01). If a sequence exists you must print the sequence of exchanges that results in a profit. If there is more than one sequence that results in a profit of more than 1 percent you must print a sequence of minimal length, i.e., one of the sequences that uses the fewest exchanges of currencies to yield a profit.

Because the IRS (United States Internal Revenue Service) notices lengthy transaction sequences, all profiting sequences must consist of *n* or fewer transactions where *n* is the dimension of the table giving conversion rates. The sequence 1 2 1 represents two conversions.

If a profiting sequence exists you must print the sequence of exchanges that results in a profit. The sequence is printed as a sequence of integers with the integer *i* representing the i^{th} line of the conversion table (country *i*). The first integer in the sequence is the country from which the profiting sequence starts. This integer also ends the sequence.

If no profiting sequence of *n* or fewer transactions exists, then the line

```
no arbitrage sequence exists
```

should be printed.

Sample Input

```
3
1.2 .89
.88 5.1
1.1 0.15
4
3.1    0.0023    0.35
0.21   0.00353   8.13
200    180.559   10.339
2.11   0.089     0.06111
2
2.0
0.45
```

Sample Output

```
1 2 1
1 2 4 1
no arbitrage sequence exists
```

解题思路：

给出n种国家的货币汇率，一定金额的某种货币经过一系列汇率变换后再换成原来货币，金额增加了，求出这样的一个变换，要求变换步数最少。

Floyd变形，关于Floyd动态规划的理解。

状态转移方程：

f[k][i][j]=min(f[k-1][i][j],f[k-1][i][k]+f[k-1][k][j])

f[k][i][j]表示只经过前k个点(包括k)，从i到j的最小值。当k从1到n时，就是从i到j的最小值。我们熟悉的用二维数组的写法实际上是对空间的一种压缩。

解释一下：

计算只经过前k个点，从i到j的最小值时，有两种情况需要考虑：经过第k个点和不经过第k个点。经过第k个点则距离应是从i到k的最小值和从k到j的最小值，两个最小值的路径都必须只经过前k-1个点（为什么是k-1而不是k，事实上他们两数值相同，因为起点和终点已经有第k个点，只是在dp的过程中先产生k-1，f[k][i][k]和f[k][k][j]有可能比f[k][i][j]的值晚计算出，就不能在计算f[k][i][j]时用到这两个值）。不经过k的点则距离与只经过前k-1个点时一样。

参考代码：

```
#include <cstdio>
#include <cstring>
#define N 25
double dp[N][N][N],p[N][N][N];
int n;

void print_path(int i ,int j , int s)
{
    if(s==0)
    {
        printf("%d",i);
        return ;
    }
    print_path(i, p[i][j][s] ,s-1);
    printf(" %d",j);
    return ;
}
void DP()
{
    int s,m;
    for(s=2; s<=n; s++)
    {
        for(int k=1; k<=n; k++)
            for(int i=1; i<=n; i++)
                for(int j=1; j<=n; j++)
                    if( dp[i][j][s] < dp[i][k][s-1]*dp[k][j][1])
                    {
                        dp[i][j][s]=dp[i][k][s-1]*dp[k][j][1];
                        p[i][j][s]=k;
                    }
        int i;
        for(i=1; i<=n; i++)
            if(dp[i][i][s]>1.01)
            {
                m=i ;
                break;
            }
        if(i<=n)
            break;
    }
    if(s>n)
        printf("no arbitrage sequence exists");
    else
        print_path(m , m , s);

    printf("\n");
}
int main()
{
    while(scanf("%d",&n)!=EOF)
    {
        memset(dp,0,sizeof(dp));
        memset(p,0,sizeof(p));
        for(int i=1; i<=n; i++)
            for(int j=1; j<=n; j++)
            {
                if(i==j) dp[i][j][1]=1;
                else
                    scanf("%lf",&dp[i][j][1]);
                p[i][j][1]=j;
            }
        DP();
    }
    return 0;
}
```

具体分析推荐博客：**http://www.cnblogs.com/scau20110726/archive/2012/12/26/2834674.html**

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5041465>

UVa 103 - Stacking Boxes（dp求解）

2016年5月6日 22:38

题目来源：

https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=3&page=show_problem&problem=39

Stacking Boxes

Background

Some concepts in Mathematics and Computer Science are simple in one or two dimensions but become more complex when extended to arbitrary dimensions. Consider solving differential equations in several dimensions and analyzing the topology of an *n*-dimensional hypercube. The former is much more complicated than its one dimensional relative while the latter bears a remarkable resemblance to its ``lower-class'' cousin.

The Problem

Consider an *n*-dimensional ``box'' given by its dimensions. In two dimensions the box (2,3) might represent a box with length 2 units and width 3 units. In three dimensions the box (4,8,9) can represent a box

$4 \times 8 \times 9$

(length, width, and height). In 6 dimensions it is, perhaps, unclear what the box (4,5,6,7,8,9) represents; but we can analyze properties of the box such as the sum of its dimensions.

In this problem you will analyze a property of a group of *n*-dimensional boxes. You are to determine the longest *nesting string* of boxes, that is a sequence of boxes

$b_1, b_2, \ldots, b_k$ such that each box b_i nests in box b_{i+1} ($1 \leq i < k)$.

A box D = ($d_1, d_2, \ldots, d_n$) nests in a box E = ($e_1, e_2, \ldots, e_n$) if there is some rearrangement of the d_i such that when rearranged each dimension is less than the corresponding dimension in box E. This loosely corresponds to turning box D to see if it will fit in box E. However, since any rearrangement suffices, box D can be contorted, not just turned (see examples below).

For example, the box D = (2,6) nests in the box E = (7,3) since D can be rearranged as (6,2) so that each dimension is less than the corresponding dimension in E. The box D = (9,5,7,3) does NOT nest in the box E = (2,10,6,8) since no rearrangement of D results in a box that satisfies the nesting property, but F = (9,5,7,1) does nest in box E since F can be rearranged as (1,9,5,7) which nests in E.

Formally, we define nesting as follows: box D = ($d_1, d_2, \ldots, d_n$) *nests* in box E = ($e_1, e_2, \ldots, e_n$) if there is a permutation π of $1 \ldots n$ such that ($d_{\pi(1)}, d_{\pi(2)}, \ldots, d_{\pi(n)}$

) ``fits'' in (

$e_1, e_2, ..., e_n$

) i.e., if

$d_{\pi(i)} < e_i$

for all

$1 \leq i \leq n$

.

The Input

The input consists of a series of box sequences. Each box sequence begins with a line consisting of the the number of boxes *k* in the sequence followed by the dimensionality of the boxes, *n* (on the same line.)

This line is followed by *k* lines, one line per box with the *n* measurements of each box on one line separated by one or more spaces. The

i^{th}

line in the sequence (

$1 \leq i \leq k$

) gives the measurements for the

i^{th}

box.

There may be several box sequences in the input file. Your program should process all of them and determine, for each sequence, which of the *k* boxes determine the longest nesting string and the length of that nesting string (the number of boxes in the string).

In this problem the maximum dimensionality is 10 and the minimum dimensionality is 1. The maximum number of boxes in a sequence is 30.

The Output

For each box sequence in the input file, output the length of the longest nesting string on one line followed on the next line by a list of the boxes that comprise this string in order. The ``smallest'' or ``innermost'' box of the nesting string should be listed first, the next box (if there is one) should be listed second, etc.

The boxes should be numbered according to the order in which they appeared in the input file (first box is box 1, etc.).

If there is more than one longest nesting string then any one of them can be output.

Sample Input

```
5 2
3 7
8 10
5 2
9 11
21 18
8 6
5 2 20 1 30 10
23 15 7 9 11 3
40 50 34 24 14 4
9 10 11 12 13 14
31 4 18 8 27 17
44 32 13 19 41 19
1 2 3 4 5 6
80 37 47 18 21 9
```

Sample Output

```
5
3 1 2 4 5
4
7 2 5 6
```

解题思路：

题目意思：给定n个m维的矩形，问我们能够嵌套的矩形最多有几个，输出个数和嵌套的矩形编号。

代码：

```
#include<bits/stdc++.h>
```

```
#define inf 0x7fffffff
using namespace std;
typedef long long LL;

int k,n;
int dp[33],pre[33];
struct node
{
    int an[13];
    int id;
    friend bool operator < (node a,node b)
    {
        for (int i=0 ;i<n ;i++)
        {
            if (a.an[i] != b.an[i]) return a.an[i] <  b.an[i];
        }
    }
}arr[33];

void printOut(int u)
{
    if (pre[u]!=-1) printOut(pre[u]);
    if (pre[u]==-1) printf("%d",arr[u].id+1 );
    else printf(" %d",arr[u].id+1 );
}

int main()
{
    while (scanf("%d%d",&k,&n)!=EOF)
    {
        memset(dp,0,sizeof(dp));
        memset(pre,-1,sizeof(pre));
        for (int i=0 ;i<k ;i++)
        {
            for (int j=0 ;j<n ;j++)
                scanf("%d",&arr[i].an[j]);
            arr[i].id=i;
            sort(arr[i].an,arr[i].an+n);
        }
        sort(arr,arr+k);
        for (int i=0 ;i<k ;i++)
        {
            int temp=0;
            for (int j=0 ;j<i ;j++)
            {
                int flag=0;
                for (int u=0 ;u<n ;u++)
                    if (arr[i].an[u]<=arr[j].an[u]) {flag=1;break; }
                if (!flag && dp[j]>temp)
                {
                    temp=dp[j];
                    pre[i]=j;
                }
            }
            dp[i]=temp+1;
        }
        int maxlen=-1,num=0;
        for (int i=0 ;i<k ;i++)
        {
            if (dp[i]>maxlen)
            {
                maxlen=dp[i];
                num=i;
            }
        }
        printf("%d\n",maxlen);
        printOut(num);
        printf("\n");
    }
    return 0;
}
```

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5041458>

UVa 102 - Ecological Bin Packing（规律,统计）

2016年5月6日　　22:38

题目来源：
https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=3&page=show_problem&problem=38

Ecological Bin Packing

Background

Bin packing, or the placement of objects of certain weights into different bins subject to certain constraints, is an historically interesting problem. Some bin packing problems are NP-complete but are amenable to dynamic programming solutions or to approximately optimal heuristic solutions.

In this problem you will be solving a bin packing problem that deals with recycling glass.

The Problem

Recycling glass requires that the glass be separated by color into one of three categories: brown glass, green glass, and clear glass. In this problem you will be given three recycling bins, each containing a specified number of brown, green and clear bottles. In order to be recycled, the bottles will need to be moved so that each bin contains bottles of only one color.

The problem is to minimize the number of bottles that are moved. You may assume that the only problem is to minimize the number of movements between boxes.

For the purposes of this problem, each bin has infinite capacity and the only constraint is moving the bottles so that each bin contains bottles of a single color. The total number of bottles will never exceed 2^31.

The Input

The input consists of a series of lines with each line containing 9 integers. The first three integers on a line represent the number of brown, green, and clear bottles (respectively) in bin number 1, the second three represent the number of brown, green and clear bottles (respectively) in bin number 2, and the last three integers represent the number of brown, green, and clear bottles (respectively) in bin number 3. For example, the line 10 15 20 30 12 8 15 8 31

indicates that there are 20 clear bottles in bin 1, 12 green bottles in bin 2, and 15 brown bottles in bin 3.

Integers on a line will be separated by one or more spaces. Your program should process all lines in the input file.

The Output

For each line of input there will be one line of output indicating what color bottles go in what bin to minimize the number of bottle movements. You should also print the minimum number of bottle movements.

The output should consist of a string of the three upper case characters 'G', 'B', 'C' (representing the colors green, brown, and clear) representing the color associated with each bin.

The first character of the string represents the color associated with the first bin, the second character of the string represents the color associated with the second bin, and the third character represents the color associated with the third bin.

The integer indicating the minimum number of bottle movements should follow the string.

If more than one order of brown, green, and clear bins yields the minimum number of movements then the alphabetically first string representing a minimal configuration should be printed.

Sample Input

```
1 2 3 4 5 6 7 8 9
5 10 5 20 10 5 10 20 10
```

Sample Output

```
BCG 30
CBG 50
```

题目翻译：

背景

装箱问题，即在一定的约束条件下如何将不同重量的物体放入不同的箱子，是一个历史悠久的有趣问题。某些装箱问题是NP完全的，但可以通过动态规划法或近似最优的启发式解法来解决。在这个问题中，你要解决一个回收玻璃瓶的装箱问题

问题

玻璃瓶的回收需要将其按颜色分为三类：棕、绿和无色。在这个问题中你会得到三个回收废品箱，每个都包括给定数量的棕、绿或无色玻璃瓶。为了便于回收，瓶子需要进行分捡和移动，使得每个废品箱都只有一个颜色的瓶子。

问题就是要使移动的瓶子数最小化。你可以假定唯一的问题就是使箱子间的移动次数最小化。

对于这一问题的目的，每个废品箱的容量是无限的，并且唯一的约束条件就是要通过移动瓶子使每个废品箱中都只有一种颜色的瓶子。瓶子的总量永不会超过2^{31}。

输入

输入由多行数据构成，每行数据中有9个整数。一行的前3个整数表示在1号废品箱中棕、绿和无色瓶子的数量各是多少，中间3个整数表示在2号废品箱中棕、绿和无色瓶子的数量各是多少，后3个

整数表示在3号废品箱中棕、绿和无色瓶子的数量各是多少。比如下面的一行：

10 15 20 30 12 8 15 8 31

表示共有1号废品箱中有20个无色瓶子，2号废品箱有12个绿色瓶子，3号废品箱中有15个棕色瓶子。

每行的各整数间有一个或多个空格。你的程序要处理输入数据中的所有行。

输出

对于每行输入要有对应的一行输出，给出哪个颜色的瓶子装入哪个废品箱才能使瓶子的移动最小化。你还应打印出瓶子移动的最少次数。

输出应用3个大写字母‘G’、‘B’、‘C’（分别表示绿色、棕色和无色）构成的字符串来表示各废品箱中瓶子的颜色。

字符串的第1个字母为1号废品箱的颜色，第2个字母为2号废品箱的颜色，第3个字母为3号废品箱的颜色。在字符串之后应用整数输出移动瓶子次数的最小值。如果有多于一种次序的棕、绿和无色废品箱都满足同一个最少的移动次数，则按照字母表的顺序输入第一个字符串最小的一种。

输入示例

1 2 3 4 5 6 7 8 9

5 10 5 20 10 5 10 20 10

输出示例

BCG 30

CBG 50

解题思路：

首先明白题目中的玻璃瓶只有三种颜色，因此三种颜色的全排列3*2*1= 6.题目要求按照字典序最小的输出。将这六种的移动次数全部求出，找出最小的次数，同时输出对应的颜色方案》

代码：

```cpp
#include <bits/stdc++.h>

using namespace std;

string s[6]={"BCG","BGC","CBG","CGB","GBC","GCB"};
int n[3][3];
int num[6];
int main()
{
    while(~scanf("%d%d%d%d%d%d%d%d%d",&n[0][0],&n[0][1],&n[0][2],&n[1][0],&n[1][1],&n[1][2],&n[2][0],&n[2][1],&n[2][2]))
    {
        int ans=0,i,j;
        num[0]=n[1][0] + n[2][0] + n[0][2] + n[2][2] + n[0][1] + n[1][1];
        num[1]=n[1][0] + n[2][0] + n[0][1] + n[2][1] + n[0][2] + n[1][2];
        num[2]=n[1][2] + n[2][2] + n[0][0] + n[2][0] + n[0][1] + n[1][1];
        num[3]=n[1][2] + n[2][2] + n[0][1] + n[2][1] + n[0][0] + n[1][0];
        num[4]=n[1][1] + n[2][1] + n[0][0] + n[2][0] + n[0][2] + n[1][2];
        num[5]=n[1][1] + n[2][1] + n[0][2] + n[2][2] + n[0][0] + n[1][0];
        for (j=i=0; ++i<6; j=num[i]<num[j] ? i : j);
        cout<<s[j]<<" "<<num[j]<<endl;
    }
}
```

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5041435>

UVa 101 - The Blocks Problem（积木问题,指令操作）

2016年5月6日 22:38

题目来源：https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=3&page=show_problem&problem=37

The Blocks Problem

Background

Many areas of Computer Science use simple, abstract domains for both analytical and empirical studies. For example, an early AI study of planning and robotics (STRIPS) used a block world in which a robot arm performed tasks involving the manipulation of blocks.

In this problem you will model a simple block world under certain rules and constraints. Rather than determine how to achieve a specified state, you will ``program'' a robotic arm to respond to a limited set of commands.

The Problem

The problem is to parse a series of commands that instruct a robot arm in how to manipulate blocks that lie on a flat table. Initially there are *n* blocks on the table (numbered from 0 to *n*-1) with block b_i adjacent to block b_{i+1} for all

$0 \le i < n - 1$

as shown in the diagram below:

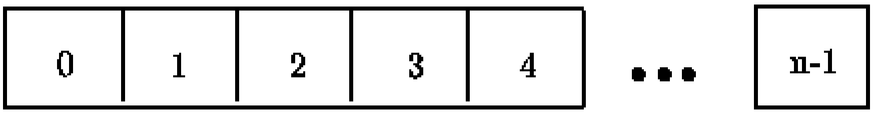


Figure: Initial Blocks World

The valid commands for the robot arm that manipulates blocks are:

- move *a* onto *b*
  where *a* and *b* are block numbers, puts block *a* onto block *b* after returning any blocks that are stacked on top of blocks *a* and *b* to their initial positions.

- move *a* over *b*
  where *a* and *b* are block numbers, puts block *a* onto the top of the stack containing block *b*, after returning any blocks that are stacked on top of block *a* to their initial positions.

- pile *a* onto *b*
  where *a* and *b* are block numbers, moves the pile of blocks consisting of block *a*, and any blocks that are stacked above block *a*, onto block *b*. All blocks on top of block *b* are moved to their initial positions prior to the pile taking place. The blocks stacked above block *a* retain their order when moved.

- pile *a* over *b*
  where *a* and *b* are block numbers, puts the pile of blocks consisting of block *a*, and any blocks that are stacked above block *a*, onto the top of the stack containing block *b*. The blocks stacked above block *a*retain their original order when moved.

- quit
  terminates manipulations in the block world.

  Any command in which *a* = *b* or in which *a* and *b* are in the same stack of blocks is an illegal command. All illegal commands should be ignored and should have no affect on the configuration of blocks.

The Input

The input begins with an integer *n* on a line by itself representing the number of blocks in the block world. You may assume that 0 < *n* < 25.

The number of blocks is followed by a sequence of block commands, one command per line. Your program should process all commands until the quit command is encountered.

You may assume that all commands will be of the form specified above. There will be no syntactically incorrect commands.

The Output

The output should consist of the final state of the blocks world. Each original block position numbered *i* (

$$0 \leq i < n$$

where *n* is the number of blocks) should appear followed immediately by a colon. If there is at least a block on it, the colon must be followed by one space, followed by a list of blocks that appear stacked in that position with each block number separated from other block numbers by a space. Don't put any trailing spaces on a line.

There should be one line of output for each block position (i.e., *n* lines of output where *n* is the integer on the first line of input).

Sample Input

```
10
move 9 onto 1
move 8 over 1
move 7 over 1
move 6 over 1
pile 8 over 6
pile 8 over 5
move 2 over 1
move 4 over 9
quit
```

Sample Output

```
 0: 0
 1: 1 9 2 4
 2:
 3: 3
 4:
 5: 5 8 7 6
 6:
 7:
 8:
 9:
```

感觉有必要解释一下题目的大致意思：

背景

在计算机科学中的很多地方都会使用简单，抽象的方法来做分析和实验验究。比如在早期的规划学和机器人学的人工智能研究就利用一个积木世界，让机械臂执行操作积木的任务。

在这个问题中，你将在确定的规则和约束条件下构建一个简单的积木世界。这不是让你来研究怎样达到某种状态，而是编写一个“机械臂程序”来响应有限的命令集。

问题

问题就是分析一系列的命令，告诉机械臂如何操纵放在一个平台上的积木。最初平台上有n个积木（编号由0到n - 1），对于任意的$0 \le i < n - 1$，积木b_i都与b_{i+1}相临，图示如下：

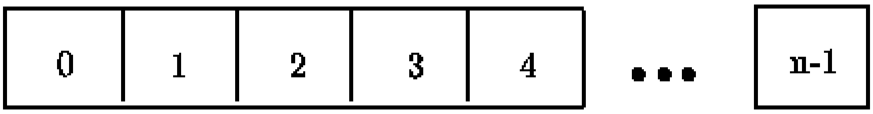


Figure: Initial Blocks World

图：积木世界的初始状态

机械臂操作积木的有效指令列举如下：

move a onto b

- a和b都是积木的编号，先将a和b上面所有的积木都放回原处，再将a放在b上。
move a over b
- a和b都是积木的编号，先将a上面所有的积木放回原处，再将a放在b上。（b上原有积木不动）
pile a onto b
- a和b都是积木的编号，将a和其上面所有的积极组成的一摞整体移动到b上。在移动前要先将b上面所有的积极都放回原处。移动的一摞积木要保持原来的顺序不变。
pile a over b
- a和b都是积木的编号，将a和其上面所有的积极组成的一摞整体移动到b所在一摞积木的最上面一个积木上。移动的一摞积木要保持原来的顺序不变。
quit
- 结束积木世界的操纵。

当a = b或a和b处在同一摞时，任何企图操作a和b的命令都是非法的。所有非法的命令都要忽略，且不能对当前积木的状态产生作用。

输入

输入由1个整数n开始开始，该整数独占一行，表示积木世界中的积木数量。你可以假定$0 < n < 25$。

从积木数量值的下一行开始是一系列的命令，每条命令独占一行。你的程序要处理所有的命令直到输入退出命令。你可以假定所有的命令都按上文所示的格式给出。不会出现语法错误的命令。

输出

以积木世界的最终状态作为输出。每一个原始积木的位置i（0 ≤ i < n，n为积木数量）后面都要紧跟一个冒号。如果至少有一个积木在该位置上，冒号后面都要紧跟一个空格，然后是该位置上所有积木编号的序列。每2个积木的编号之间以一个空格隔开。行尾不能出现多余的空格。

每个积木位置独占一行（即第一行输入的n，对应输出n行数据）。

输入示例

```
10
move 9 onto 1
move 8 over 1
move 7 over 1
move 6 over 1
pile 8 over 6
pile 8 over 5
move 2 over 1
move 4 over 9
quit
```

输出示例

```
0: 0
1: 1 9 2 4
2:
3: 3
4:
5: 5 8 7 6
6:
7:
8:
9:
```

解题分析：

再明白积木世界的操作规律之后，现在就是要思考一下怎样来解决这个问题。

当我看到输出结果的时候，我会想到二维数组。为什么呢？因为每次的输出都要把相对应位置的积木编号输出，总共的行数已知也就是输入时的积木个数（同时积木编号是从0开始，也是方便了二维数组的使用，不必再其中修改序号），另外的原因就是，这四种操作有着一定的规律。

规律：无需考虑将积木放回原位时原位被占的情况，因为没有任何一个操作可以把积木放在空的位置上，因此也不

可能出现在第i个位置上，第i个积木的下面有其它积木的情况。可将操作分为两类，一类需要清空上面的积木，一类不需要。

这样就更容易理解题目和操作。

但是在我开始向大家讲我使用二维数组的时候，希望大家自己思考使用不同的解题方法。（不得不承认二维数组实在是太麻烦了）。

那么你在有自己的想法之后，我的二维数组就是一个对过程很具体很细致的描述。你会感觉到二维数组看似很合适其实很麻烦。

下面我就开始讲一下二维数组怎么来解决这个问题。

我们倒着来想，我的结果输出需要序号同时也需要把相对应的积木编号输出来。这样积木中一摞就相当于二维数组的一行，这一点还是很清楚的。所以我们的目的就是进过一次操作之后就把二维数组的每一行进行更新。（这里不是全部行都要更新，只是更新操作中所涉及到的那些摞积木所对应的行）。一定要明白目的是什么，不然下面的分析不是很好懂。

首先我们需要把二维数组进行初始化。二维数组中所存储的是积木的编号。由于题目中给出积木的编号在25之内，所以二维数组的初始化见下图：

0	INF	INF	INF	INF	…
1	INF	INF	INF	INF	…
2	INF	INF	INF	INF	…
3	INF	INF	INF	INF	…
4	INF	INF	INF	INF	…
…	…	…	…	…	…

其中的INF可以设置为大于25的数字或者负数，关键是能够起到标记该位置没有积木的作用就可以。

然后就要进行输入操作指令。声明string类型变量s1（move 或pile）,s2（onto或over）.声明int类型start,to分别表示移动积木的编号。

进行指令的判断。

在进行操作之前，我们还需要一个很重要并且很简单很无脑的函数。遍历二维数组找到元素所在位置的行下标和列下标。（自己写find_pos函数来实现找位置）

【每次指令都会调用这个find_pos函数，但是由于题目数据量不是很大，所以速度还是可以的】

（1）如果是move start onto to 类型的指令：

首先需要把start和to上的所有积木全部放回原来的位置。（根据前面的分析，放回原来的位置不需要考虑有没有积木占着这个位置，因为没有积木可

以移动到不是自己原来位置的空位上）。然后把start放到to上。利用之前的find_pos函数来找到start的位置，然后使用for循环把这一摞的积木都按照编号放回原来的位置，对于to也是这样的操作。最后只需要把编号为start积木放到编号为to的积木上。（注意在操作过程中一定要把移动的积木后位置初始化为INF）

(2)如果是move start over to 类型指令:

按照上面的操作方法，只需要把start上面的所有积木都放回原来的位置（所谓原来的位置就是编号的积木编号一样的位置）.这时候不需要移动to上面的积木。最后把start积木移动到to积木上。初始化原来start积木位置为INF.

(3)如果是pile start onto to类型指令：

按照上面的操作方法，把to积木上面的积木都初始化，然后这些位置也相应的初始化为INF。然后把start以及start上的所有积木都拿到to积木上，不能改变积木顺序。利用嵌套的for循环加上一定的判断语句很容易实现

(4)如果是pile start over to 类型指令：

需要把start以及start上的积木都拿到to积木着一摞的上面。同样需要在操作中把没有积木的位置初始化为INF。移动一摞积木到另一摞积木的方法类似上面的操作，使用嵌套的for循环加上判断语句。

【注意1】：在移动积木的过程中一定要保持没有积木的位置的值为INF.

【注意2】：由于题目要求如果操作的积木处于同一摞则忽视该操作，同时该操作对积木世界没有任何影响。所以在判定每个指令之后首先要做的就是判断start积木和to积木是否位于同一摞。方法还是调用之前的无脑函数find_pos

利用二维数据求解的参考代码：

```
#include <bits/stdc++.h>
#define MAX 30
#define INF 10000000
using namespace std;
int MAP[MAX][MAX];
void find_pos(int MAP[MAX][MAX],int n,int temp,int &posx,int &posy)
{
    bool flag=false;
    for(int i=0; i<n; i++)
    {
        for(int j=0; j<n; j++)
        {
            if(MAP[i][j]==temp)
            {
                posx=i;
                posy=j;
                flag=true;
                break;
            }
            if(flag==true)
                break;
        }
    }
}
int main()
{
    int n,i,j;
    while(~scanf("%d",&n))
    {
        string s1,s2;
        int start,to;
        int posx,posy;
        for(i=0; i<n; i++)
        {
            for(j=0; j<n; j++)
                if(j==0)
                    MAP[i][j]=i;
                else
                    MAP[i][j]=INF;
        }
        while(cin>>s1)
        {
            if(s1=="quit")
                break;
            else if(s1=="move")
            {
                cin>>start>>s2>>to;
                if(s2=="onto")
                {//判断是否在同一摞
                    int x1,x2;
                    find_pos(MAP,n,start,posx,posy);
                    x1=posx;
                    find_pos(MAP,n,to,posx,posy);
                    x2=posx;
                    if(x1==x2)
                        continue;
                    find_pos(MAP,n,start,posx,posy);
                    for(j=posy+1;j<n;j++)
                    {
                        if(MAP[posx][j]!=INF)
                        {
                            MAP[MAP[posx][j]][0]=MAP[posx][j];
                            MAP[posx][j]=INF;
                        }
                    }
                    MAP[posx][posy]=INF;
                    find_pos(MAP,n,to,posx,posy);
                    for(j=posy+1;j<n;j++)
                    {
                        if(MAP[posx][j]!=INF)
                        {
                            MAP[MAP[posx][j]][0]=MAP[posx][j];
                            MAP[posx][j]=INF;
                        }
                    }
                    MAP[posx][posy+1]=start;
                }
                else if(s2=="over")
                {
                    int x1,x2;
                    find_pos(MAP,n,start,posx,posy);
                    x1=posx;
                    find_pos(MAP,n,to,posx,posy);
                    x2=posx;
                    if(x1==x2)
                        continue;
                    find_pos(MAP,n,start,posx,posy);
                    for(j=posy+1;j<n;j++)
                    {
                        if(MAP[posx][j]!=INF)
                        {
                            MAP[MAP[posx][j]][0]=MAP[posx][j];
                            MAP[posx][j]=INF;
                        }
                    }
                    MAP[posx][posy]=INF;
                    find_pos(MAP,n,to,posx,posy);
                    for(j=posy+1;j<n;j++)
                    {
                        if(MAP[posx][j]==INF)
                        {
                            MAP[posx][j]=start;
                            break;
                        }
                    }
                }
            }
            else if(s1=="pile")
            {
                cin>>start>>s2>>to;
                if(s2=="onto")
                {
                    int x1,x2;
                    find_pos(MAP,n,start,posx,posy);
                    x1=posx;
                    find_pos(MAP,n,to,posx,posy);
                    x2=posx;
                    if(x1==x2)
                        continue;
                    find_pos(MAP,n,to,posx,posy);
                    for(j=posy+1;j<n;j++)
                    {
                        if(MAP[posx][j]!=INF)
                        {
                             MAP[MAP[posx][j]][0]=MAP[posx][j];
                             MAP[posx][j]=INF;
                        }
                    }
                    int tempx=posx,tempy=posy;
                    find_pos(MAP,n,start,posx,posy);
                    for(j=tempy+1;j<n;j++)
                    {
                        for(int j2=posy;j2<n;j2++)
                        {
                            if(MAP[posx][j2]!=INF)
                            {
                                MAP[tempx][j]=MAP[posx][j2];
                                MAP[posx][j2]=INF;
                                break;
                            }
                        }
                    }
                }
                else if(s2=="over")
                {
                    int x1,x2;
                    find_pos(MAP,n,start,posx,posy);
                    x1=posx;
                    find_pos(MAP,n,to,posx,posy);
                    x2=posx;
                    if(x1==x2)
                        continue;
                    find_pos(MAP,n,to,posx,posy);
                    int tempx=posx,tempy=posy;
                    for(j=posy;j<n;j++)
                    {
                        if(MAP[tempx][j]==INF)
                        {
                            tempy=j;
                            break;
                        }
                    }
                    find_pos(MAP,n,start,posx,posy);
                    for(j=tempy;j<n;j++)
                    {
                        for(int j2=posy;j2<n;j2++)
                        {
                            if(MAP[posx][j2]!=INF)
                            {
                                MAP[tempx][j]=MAP[posx][j2];
                                MAP[posx][j2]=INF;
                                break;
                            }
                        }
                    }
                }
            }
        }
        for(i=0;i<n;i++)
        {
            printf("%d:",i);
            for(j=0;j<n;j++)
            {
                if(MAP[i][j]!=INF)
                {
                    printf(" %d",MAP[i][j]);
                }
            }
            printf("\n");
        }
    }
}
```

其他解题思路博客推荐1：

http://www.cnblogs.com/devymex/archive/2010/08/04/1792128.html

其他解题思路博客推荐1：

http://blog.csdn.net/mobius_strip/article/details/12765319

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5039598>

UVa 100 - The 3n + 1 problem（函数循环长度）

2016年5月6日 22:39

题目来源：https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=3&page=show_problem&problem=36

The 3*n* + 1 problem

Background

Problems in Computer Science are often classified as belonging to a certain class of problems (e.g., NP, Unsolvable, Recursive). In this problem you will be analyzing a property of an algorithm whose classification is not known for all possible inputs.

The Problem

Consider the following algorithm:

1. input *n*

2. print *n*

3. if *n* = 1 then STOP

4. if *n* is odd then
$n \longleftarrow 3n+1$

```
5. else
$n \longleftarrow n/2$

6. GOTO 2
```

Given the input 22, the following sequence of numbers will be printed 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

It is conjectured that the algorithm above will terminate (when a 1 is printed) for any integral input value. Despite the simplicity of the algorithm, it is unknown whether this conjecture is true. It has been verified, however, for all integers *n* such that 0 < *n* < 1,000,000 (and, in fact, for many more numbers than this.)

Given an input *n*, it is possible to determine the number of numbers printed (including the 1). For a given *n*this is called the *cycle-length* of *n*. In the example above, the cycle length of 22 is 16.

For any two numbers *i* and *j* you are to determine the maximum cycle length over all numbers between *i* and *j*.

The Input

The input will consist of a series of pairs of integers *i* and *j*, one pair of integers per line. All integers will be less than 1,000,000 and greater than 0.

You should process all pairs of integers and for each pair determine the maximum cycle length over all integers between and including *i* and *j*.

You can assume that no operation overflows a 32-bit integer.

The Output

For each pair of input integers *i* and *j* you should output *i*, *j*, and the maximum cycle length for integers between and including *i* and *j*. These three numbers should be separated by at least one space with all three numbers on one line and with one line of output for each line of input. The integers *i* and *j* must appear in the output in the same order in which they appeared in the input and should be followed by the maximum cycle length (on the same line).

Sample Input

```
1 10
100 200
201 210
900 1000
```

Sample Output

```
1 10 20
100 200 125
201 210 89
900 1000 174
```

解题分析：根据题目描述需要在给定的区间里面找出循环长度最长的。

首先明确这个函数是什么？

函数f(n)为分段函数：当n为奇数时f(n)=3n+1;当n为偶数时f(n)=n/2;

然后需要明白什么是循环长度？

就是当f(n)进行多少次内部运算之后才能得到 1。

最后就是代码：

（1）写出函数计算循环长度;

（2）由于题目中没有说明i，j的大小关系，同时题目要求输出的i,j顺序和输入的i,j顺序要相同，所以可以有两种方法：

第一种：输入i,j之后直接输出，然后计算最长循环长度;

第二种：输入i，j之后，声明新的变量来使用，计算过程中完全不影响i,j的值。最后输出时i,j直接输出即可。

已过代码：

```
#include <bits/stdc++.h>

using namespace std;


int length(int n) {
    int len=1;
    while(n!=1)
    {
        if(n%2==1)
            n=3*n+1;
        else
            n=n/2;
        len=len+1;
    }
    return len;
}

int main(void) {
    int start,over;
    int s,o;
    int ans;
    while(~scanf("%d%d",&start,&over))
    {
        ans=1;
        s=start, o=over ;
        if(s>o) swap(s,o);
        for(int i=s;i<=o;i++)
        {
            int len=length(i);
            ans=max(ans,len);
        }
        printf("%d %d %d\n",start,over,ans);
    }
}
```

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5039499>

就决定是你了！

2016年5月6日　　22:39

题目来源：https://biancheng.love/contest-ng/index.html#/34/problems

题目描述

Nova君来到神奇宝贝中心，准备从存储机中挑选若干个Pockmon去挑战最后的四天王和联盟冠军。Nova君现在有N只可挑选的Pockmon，对于第 i 只Pockmon，他的体总为 wi ，战斗力为 vi 。现在联盟规定，只能携带总重量为M的Pockmon进行挑战。那么，Nova君要怎么挑选才能使战斗力最强呢？大家肯定会说，这不是个简单的背包问题嘛？嗯，大概。但是现在有个问题，由于某些Pockmon在冒险的途中，和存储机内的其他Pockmon产生了羁绊，有可能对另一只Pokmon产生依赖，非得被依赖的那只被选中才能发挥战斗力，否则只能是个战斗力为0的卖萌生物而已，好在每只Pockmon都很专情，只会依赖一只（或者不依赖）。

PS：

(1) 依赖关系不对称，即，若A依赖B，B不一定依赖A，俗称beitai；

(2) 可能出现这样的情况：A依赖B，B依赖C，C依赖A，俗称love triangle，当然也有可能是多边形（悲伤脸）

(3) 保证不会出现自己依赖自己的情况

输入

多组测试数据（组数不超过10），对于每组数据，输入4行，第一行是两个正整数N，M，表示Pockmon总数和联盟允许的最大重量，第二行包含N个正整数w1，w2...wn，表示第 i 个Pockmon的重量，第三行包含N个正整数v1，v2...vn，表示第 i 个Pockmon的战斗力，第四行包含N个正整数D1，D2...Dn，表示第 i 个Pockmon依赖的Pockmon的序号，如果没有依赖对象，则 Di=0。

PS：输入数据都在INT范围内

输出

对于每组数据，输出一行，表示战斗力的最大值

输入样例

```
3 10
4 5 6
2 3 4
3 1 2
3 10
4 5 6
2 3 4
0 1 1
```

输出样例

```
0
6
```

代码：

```
#include<bits/stdc++.h>
using namespace std;

const int e=505;
int n,m,tmpw=0,tmpn;
int w[e]= {0},v[e]= {0},b[e]= {0},c[e]= {0},f[e][5*e]= {0},d[e]= {0};
bool mapp[e][e]= {0};

void floride()
```

```
{
    for(int i=1; i<=n; i++) //弗洛里德判断是否有环;
        for(int j=1; j<=n; j++)
            for(int k=1; k<=n; k++)
                if(mapp[k][i]==1 && mapp[i][j]==1)
                    mapp[k][j]=1;
}


void merge()//合点
{
    tmpn=n;
    for(int i=1; i<=tmpn; i++)
        for(int j=1; j<=tmpn; j++)
        {
            if(mapp[i][j]==1 && mapp[j][i]==1 && i!=j && w[i]>0 && w[j]>0)//如果是新环;
            {
                tmpn++;
                v[tmpn]=v[i]+v[j];
                w[tmpn]=w[i]+w[j];
                tmpw--;
                w[i]=tmpw;
                w[j]=tmpw;    //tmpw+tmpn永远等于最开始的n
            }

            //如果j依赖的点被合并(是旧环),且j在环里
            if(w[d[j]]<0 && w[j]>0 && mapp[j][d[j]]==1 && mapp[j][d[j]]==1)
            {
                w[n-w[d[j]]]+=w[j];
                v[n-w[d[j]]]+=v[j];
                w[j]=w[d[j]];
            }

            //如果j依赖的点在环里，但是j不在环里
            if(w[d[j]]<0 && w[j]>0)
                if((mapp[j][d[j]]==1 && mapp[d[j]][j]==0) || (mapp[j][d[j]]==0 && mapp[d[j]][j]==1))
                    d[j]=n-w[d[j]];
        }
}

int  dfs(int x,int k)
{
    if(f[x][k]>0)    return(f[x][k]);
    if(x==0 || k<=0)    return(0);
    //不取x
    f[b[x]][k]=dfs(b[x],k);
    f[x][k]=f[b[x]][k];
    int y=k-w[x];
    for(int i=0; i<=y; i++)
    {
        f[c[x]][y-i]=dfs(c[x],y-i);
        f[b[x]][i]=dfs(b[x],i);
        f[x][k]=max(f[x][k],v[x]+f[c[x]][y-i]+f[b[x]][i]);
    }
    return(f[x][k]);
}



int main()
{
    while(cin>>n>>m)
    {
        tmpw=0;
        memset(w,0,sizeof(w));
        memset(v,0,sizeof(v));
        memset(c,0,sizeof(c));
        memset(b,0,sizeof(b));
        memset(d,0,sizeof(d));
        memset(f,0,sizeof(f));
        memset(mapp,0,sizeof(mapp));

        for(int i=1; i<=n; i++)
            scanf("%d",&w[i]);
        for(int i=1; i<=n; i++)
            scanf("%d",&v[i]);
        for(int i=1; i<=n; i++)
        {
            int a;
            scanf("%d",&a);
            d[i]=a;
            mapp[a][i]=1;
        }

        floride();
        merge();

        //多叉转二叉
        for(int i=1; i<=tmpn; i++)
            if(w[i]>0)
            {
                b[i]=c[d[i]];
                c[d[i]]=i;
            }
        cout<<dfs(c[0],m)<<endl;
    }
}
```

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5037570>

阿姆斯特朗回旋加速喷气式阿姆斯特朗炮

2016年5月6日　　22:39

题目来源：https://biancheng.love/contest-ng/index.html#/34/problems

题目描述

时代在进步，在Nova君的改进下，阿姆斯特朗回旋加速喷气式阿姆斯特朗炮终于进化成为了先进的电磁轨道炮，不仅能够在直筒中发射直线型电磁炮，还能利用正负极磁场，在两座炮台间形成弧形电磁炮，大大加强了御敌能力。某一天，天人（外星人）入侵了地球，现在要用阿姆斯特朗回旋加速喷气式阿姆斯特朗炮进行轰击，简化图如下：![]
(http://ww1.sinaimg.cn/mw690/006jMFyngw1eyupfrs57wj30kg0drgm3.jpg)

现在抽象问题，假设外星人都在X轴上方（不含X轴本身）出现，现在有三门阿姆斯特朗回旋加速喷气式阿姆斯特朗炮，中间的A炮，可以发射以原点O为起点的任意射线的电磁炮，其方向向量和X轴正方向夹角为Θ，可以消灭位于射线的所有敌人，位于两翼的炮B和炮C形成电磁回路，两炮口间有半圆形的弧形电磁炮，可以消灭位于弧形线上的所有敌人，设半圆的半径为R。每当发现敌人，A炮可以计算出敌人与O点连线和X轴正方向的夹角Θi，B炮和C炮可以计算出敌人所处弧形炮的半径Ri。现在有N个敌人，至少需要发射几次炮弹才能清理所有的外星人？（B、C联动只算发射一次）

PS：

(1) 假设Θi为角度制，范围1~179（单位：度）

(2) 假设敌人出现的最大半径为1000

输入

多组测试数据（组数不超过10），对于每组数据，输入N+1行，第一行输入一个正整数N，代表外星人的个数。接下来的N行，每行输入两个正整数，为对应的A炮计算出的Θi 和 B、C炮计算出的Ri。

输出

对于每组数据，输出一行，代表发射的最少次数。

输入样例

```
3
1 2
2 2
3 4
```

输出样例

```
2
```

代码：

```
#include<iostream>
#include<cstdio>
#include<list>
#include<algorithm>
#include<cstring>
#include<string>
#include<queue>
#include<stack>
#include<map>
#include<vector>
#include<cmath>
#include<memory.h>
#include<set>
```

```

long long TNF=99999999;
#define LL __int64
#define eps 1e-8


using namespace std;

#define M 400000100

#define inf 0xfffffff


vector<int>G[1212];

char tempmp[1212];
int mp[1212][1212];
int lmarry[1212],rmarry[1212];
bool visl[1212],visr[1212];

int dis[2][4]={0,-1,0,1,1,0,-1,0};

int n=1000,m=1000,k;

void clear()
{
    memset(lmarry,-1,sizeof(lmarry));
    memset(rmarry,-1,sizeof(rmarry));
    memset(visl,false,sizeof(visl));
    memset(visr,false,sizeof(visr));
    memset(mp,0,sizeof(mp));
    for(int i=0;i<1212;i++)
        G[i].clear();
}

bool dfs(int x)
{
    visl[x]=true;
    for(int i=0;i<G[x].size();i++)
    {
        int v=G[x][i];
        if(!visr[v])
        {
            visr[v]=true;
            if(lmarry[v]==-1 || dfs(lmarry[v]))
            {
                lmarry[v]=x;
                rmarry[x]=v;
                return 1;
            }
        }
    }
    return 0;
}


int main(void)
{
    while(~scanf("%d",&k))
    {
        clear();
        int u,v;
        for(int i=0;i<k;i++)
        {
            scanf("%d %d",&u,&v);
            G[u].push_back(v);
        }
        int ans=0;
        for(int i=1;i<=n;i++)
        {
            memset(visr,false,sizeof(visr));
            if(dfs(i))
                ans++;
        }
        printf("%d",ans);
        memset(visl,false,sizeof(visl));
        memset(visr,false,sizeof(visr));
        printf("\n");
    }
}
```

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5037565>

零崎的悠哉日常III

2016年5月6日　　22:39

题目来源：https://biancheng.love/contest-ng/index.html#/34/problems

题目描述

零崎在空闲时间很多的时候，就喜欢玩一些非常耗时间的游戏，比如可以死上几千次的I wanna，又比如一不小心就要重头开始的多米诺骨牌。

在摆放多米诺骨牌时，如果中途碰倒一块，就会产生雪崩般的影响。比如说**11__1x11_11**这种形状，零崎这么作死的人当然会在中间x位置放一枚骨牌……这种作死的做法有*pl*的概率会倒向左边并把左边的1个骨牌碰倒，或者有*pr*的概率会倒向右边并把右边的2个都碰倒。（骨牌不是量子态不会既左倒又右倒……）

现在零崎准备把手里的N枚多米诺骨牌摆成一条直线，那么他摆放骨牌次数的期望是多少？

输入

多组输入数据。

每组数据为三个数，第一个为整数N<10000，接下来两个浮点数pl，pr，0<pl+pr<1

输出

对于每组数据，输出一行，为采取最佳摆放方式时的次数期望，保留两位小数

输入样例

```
2 0.1 0.1
10 0.2 0.3
```

输出样例

```
2.66
44.03
```

Hint

几何分布的期望Ex=1/p

O(n^2)可过

代码：

```
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
#define INF 0x3f3f3f3f
const int N = 1000005;
int n;
double p, pl, pr, dp[N];
double cal(int l, int r) {
  return dp[l] + dp[r] + (pl * dp[l] + pr * dp[r] + 1) / p;
}
double solve() {
  p = 1 - pl - pr;
  dp[0] = 0; dp[1] = 1 / p;
  int pre = 0;
  for (int i = 2; i <= n; i++) {
  dp[i] = cal(pre, i - pre - 1);
  for (int j = pre + 1; j < i; j++) {
    int l = j, r = i - 1 - j;
    double tmp = cal(l, r);
    if (dp[i] >= tmp) {
    dp[i] = tmp;
    pre = j;
    }
    else break;
  }
  }
  return dp[n];
}
int main() {
  while (~scanf("%d", &n) && n) {
  scanf("%lf%lf", &pl, &pr);
  printf("%.2lf\n", solve());
  }
  return 0;
}
```

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5037560>

零崎的悠哉日常 II

2016年5月6日 22:40

题目来源：https://biancheng.love/contest-ng/index.html#/34/problems

题目描述

零崎闲下来的时候很喜欢去看书，特别是在沙河的时候，经常和社团的小伙伴一起去自习室看小说。教学楼跑的次数多了，自然对路也比较熟，比如教三大教室旁边的楼梯间里或者教学楼连通走廊的制图教室外经常有妹子读英语什么的……

说起来教室之间有各种各样不同的路可以走，不同的路的容纳量也不同，那么这么多教室这么多路，如果从一间教室前往另一间教室上课，最多可以有多少人一起走呢？零崎虽然对路和每条路的容量很熟，不过教室这么多，路太多了怎么算得清呢？

输入

多组输入数据。每组数据N+1+T行。

第一行为三个整数，教室数V，路的数量N和查询数量T。

接下来N行每行三个整数s,t,c为一条教室s到t容量为c的路。

最后T行每行两个整数x，y为查询的教室编号。

输出

每组样例输出T行，每次查询一行，为x,y间通路的最大流量。

输入样例

```
5 7 2
1 2 10
1 4 2
2 4 6
2 3 6
3 4 3
3 5 8
4 5 5
1 5
4 2
```

输出样例

```
11
0
```

解题思路：

最大流问题的处理。

代码：

```
#include <bits/stdc++.h>
using namespace std;

const int MAX_V=1010;
const int INF=99999999;
const double eps = 1e-8;

//用于表示边的结构体（终点、容量、反向边）
struct edge
{
    int to,cap,rev;
};

vector<edge> G[MAX_V];//图的邻接表表示,注意是NMAX个数组
```

```
bool used[MAX_V];      //DFS中用到的访问标记

//向图中增加一条从s到t容量为cap的边
void add_edge(int from,int to,int cap)
{
    edge a = {to,cap,G[to].size()};
    edge b = {to,cap,G[to].size()};
    G[from].push_back(a);
    G[to].push_back(b);
}

//通过DFS寻找增广路
int dfs(int v,int t,int f)
{
    if(v==t)return f;
    used[v]=true;
    for(int i=0; i<G[v].size(); i++)
    {
        edge &e=G[v][i];
        if(!used[e.to]&&e.cap>0)
        {
            int d=dfs(e.to,t,min(f,e.cap));
            if(d > 0)
            {
                e.cap -= d;
                G[e.to][e.rev].cap += d;
                return d;
            }
        }
    }
    return 0;
}

//求解从s到t的最大流
int max_flow(int s,int t)
{
    int flow=0;
    while(1)
    {
        memset(used,0,sizeof(used));
        int f=dfs(s,t,INF);
        if(f==0)
            return flow;
        flow += f;
    }
}

int main()
{
    int s,t,c,v,n,T,res;
    while(scanf("%d%d%d",&v,&n,&T)==3)
    {
        for(int i=0;i<MAX_V;i++)
            G[i].clear();
        while(n--)
        {
            scanf("%d%d%d",&s,&t,&c);
            add_edge(s,t,c);
        }
        while(T--)
        {
            scanf("%d%d",&s,&t);
            res=max_flow(s,t);
            printf("%d\n",res);
        }
    }
    return 0;
}
```

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5037554>