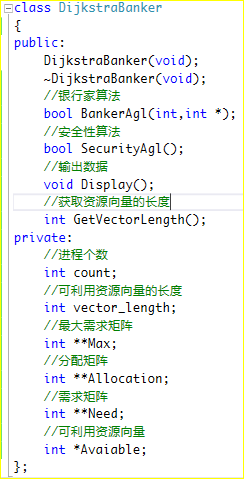
Dijkstra银行家算法

银行家算法的演示主要涉及到三个方面：银行家算法的数据结构、银行家算法以及安全性检测算法。演示功能的实现主要依赖于这三个方面的设计。

数据结构设计以及算法的实现均封装到DijkstraBanker类中，类的构造如下图所示（具体细节部分请看附件）。



1.数据结构

**①**可利用资源向量Available。初始值为系统所配置的该类资源的数目，其数值随该类资源的分配和回收动态地改变。

**②**最大需求矩阵Max。系统中的进程对每一类资源的最大需求。

**③**分配矩阵Allocation。定义了当前系统中的每一类资源已经分配给对应进程的资源数目。但不会大于最大需求。

**④**需求矩阵Need。用以表示每一个进程尚需的各类的资源数目。

矩阵间的关系：

2.银行家算法

设是进程的请求向量，如果，表示进程需要个类型的资源。当发出资源请求后，系统按下述步骤进行检查：

**①**如果，便转向步骤**②**；否则，提示出现错误：请求的资源大于其需求资源。

**②**如果，便转向步骤**③**；否则，提示错误：尚无足够的资源去提供，需等待。

**③**系统试着把资源分配给进程，并修改下面的数据结构中的数值：

**④**执行安全性算法。若通过安全性检测算法，则真正地将资源分配给进程；否则，“试探性”地分配作废，恢复原来的状态。

3.安全性算法

**⑴**设置两个向量：

①工作向量Work，表示可以提供给进程继续运行所需的各类资源数目；初始化值为：。

②Finish，表示系统有足够的资源分配给进程，使之运行完成。初始值均为false；当有足够的资源分配给进程时，再令。

**⑵**在进程集合中找到一个满足下述条件的进程：

①；

②；若找到，执行步骤**⑶**；否则，执行步骤**⑷**。

**⑶**当进程获取资源之后，执行如下的操作：

for(k=0;k<this->vector\_length;++k)

work[k]+=this->Allocation[j][k];

finish[j]=true;

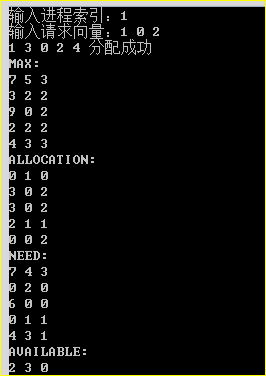
然后，返回到步骤**⑵**继续。

**⑷**如果所有进程的满足，则表示进程处于安全状态；否则，系统处于不安全状态。

程序演示的相关截图：



安全性检测算法执行



分配资源测试

实习总结：

这次的实习内容比较简单的。困难的地方在于初始化数据。我就直接使用课本上的数据来演示的。首先，我写了一个DijkstraBanker类，将所有的操作都封装在其中。初始进程和资源分配状态的数据在data.txt文本文件中，然后读取该文件数据以完成初始化。

然后，执行安全性检测算法后，可以给指定的进程分配资源。采取的方法自然是银行家算法。一开始利用银行家算法尝试分配，然后利用安全性检测算法进行检测。检测通过，即实现真正的分配；否则，进行数据还原操作，分配失败。

因为这次实习不像上次的进程同步问题，不涉及到与系统相关的API函数，主要还是数据结构的设计。两个算法的伪代码课本也提供了，因此还是很轻松。开始看银行家算法来避免死锁还是有点迷惑的，这次编程实习演示，确实是深刻理解了。

<https://github.com/solome/OS/tree/master/banker_algorithm>