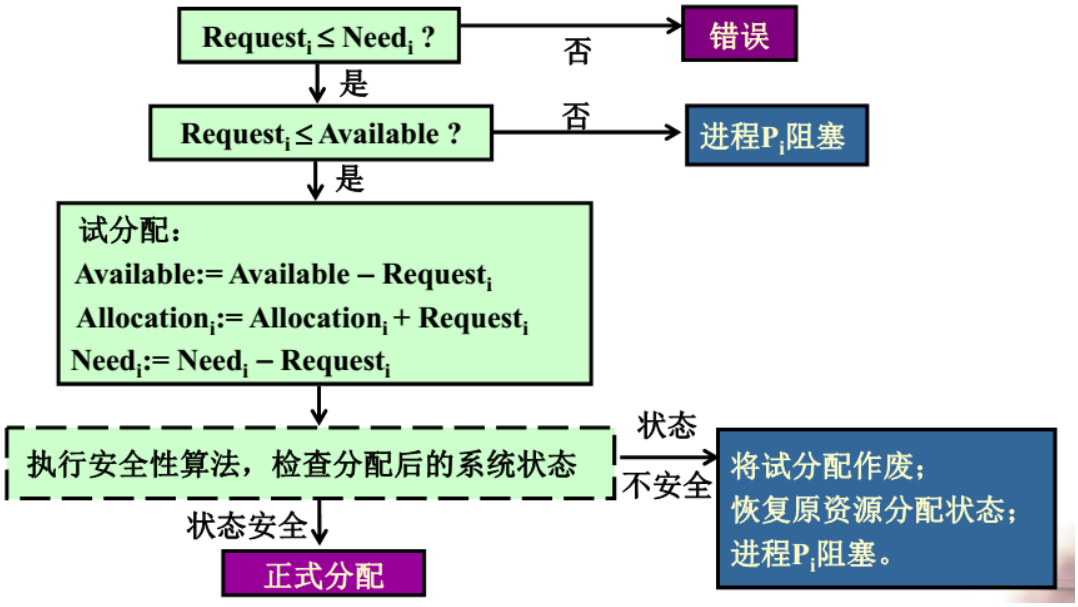
第二章　实验五

银行家+安全性算法

- 银行家算法流程图：



银行家算法自然语言描述：

设 Request$_i$ 是进程 P$_i$ 的请求向量，如果 Request$_i$［j］=*K*，表示进程 P$_i$ 需要 *K* 个 R$_j$ 类型的资源。当 P$_i$ 发出资源请求后，系统按下述步骤进行检查：

(1) 如果 Requesti［j］≤Need［i,j］，便转向步骤 2；否则认为出错，因为它所需要的资源数已超过它所宣布的最大值。

(2) 如果 Requesti［j］≤Available［j］，便转向步骤(3)；否则， 表示尚无足够资源，Pi 须等待。

(3) 系统试探着把资源分配给进程 P$_i$，并修改下面数据结构中的数值：

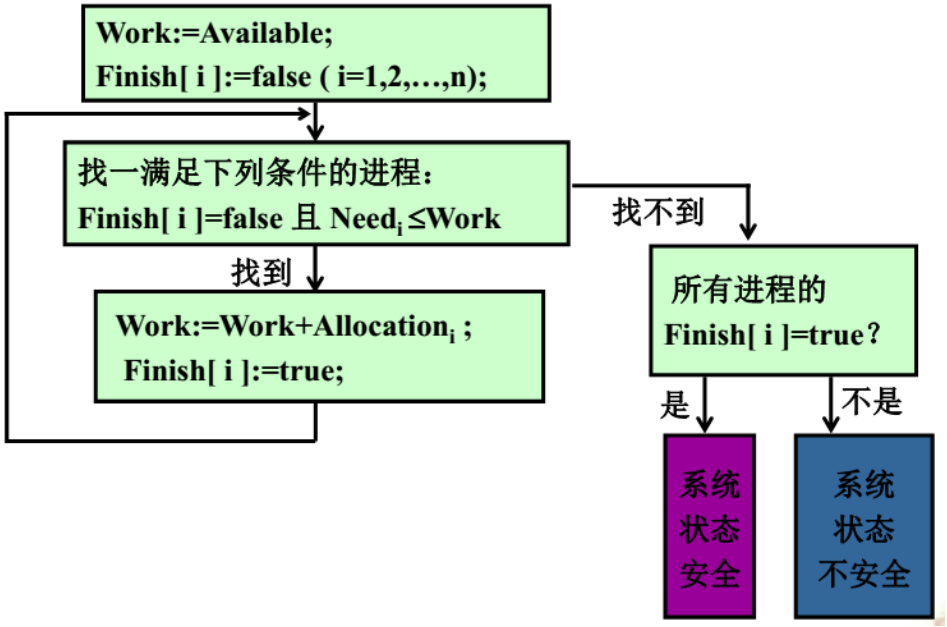
Available［j］=Available［j］-Request$_i$［j］；

Allocation［i,j］=Allocation［i,j］+Request$_i$［j］；

Need［i,j］=Need［i,j］-Request$_i$［j］；

(4) 系统执行安全性算法，检查此次资源分配后，系统是否处于安全状态。若安全，才正式将资源分配给进程 P$_i$，以完成本次分配；否则， 将本次的试探分配作废，恢复原来的资源分配状态，让进程 P$_i$ 等待。

- 安全性算法流程图：



安全性算法自然语言描述：

(1) 设置两个向量：① **工作向量 Work:** 它表示系统可提供给进程继续运行所需的各类资源数目，它含有 m 个元素，在执行安全算法开始时，Work：=Available; ② **Finish: 它表示系统是否有足够的资源分配给进程，使之运行完成。**开始时先做 Finish［i］：=false; 当有足够资源分配给进程时， 再令 Finish［i］：=true。

(2) 从进程集合中找到一个能满足下述条件的进程：
① Finish［i］=false; ② Need［i,j］≤Work［j］; 若找到， 执行步骤(3)， 否则，执行步骤(4)。

(3) 当进程 Pi 获得资源后，可顺利执行，直至完成，并释放出分配给它的资源，故应执行：
Work［j］：= Work［i］+Allocation［i,j］;
Finish［i］：= true;
go to step (2);

(4) 如果所有进程的 Finish［i］=true 都满足， 则表示系统处于安全状态；否则，系统处于不安全状态。

- **实例：** 假定系统中有五个进程{P_0, P_1, P_2, P_3, P_4}和三类资源{A, B, C}，各种资源的数量分别为10、5、7，在T_0时刻的资源分配情况下图所示。

进程 \ 资源情况	Max A B C	Allocation A B C	Need A B C	Available A B C
P_0	7 5 3	0 1 0	7 4 3	3 3 2 (2 3 0)
P_1	3 2 2	2 0 0 (3 0 2)	1 2 2 (0 2 0)	
P_2	9 0 2	3 0 2	6 0 0	
P_3	2 2 2	2 1 1	0 1 1	
P_4	4 3 3	0 0 2	4 3 1	

```
Resource--输入M种资源的总数量:
10 5 7
Max--输入N个进程分别对M种资源的最大需求量:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Allocation--输入N个进程获得M种资源的数量:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
```

输入 M 资源总数量、Max 矩阵和 Allocation 矩阵

```
状态表:
!Process     !Max          !Allocation   !Need         !Available    !
!P0          !   7   5   3!   0   1   0!   7   4   3!   3   3   2!
!P1          !   3   2   2!   2   0   0!   1   2   2!              !
!P2          !   9   0   2!   3   0   2!   6   0   0!              !
!P3          !   2   2   2!   2   1   1!   0   1   1!              !
!P4          !   4   3   3!   0   0   2!   4   3   1!              !
```

显示初始状态表

(1) 判断T_0时刻是否安全？

```
Initial state is safe!

安全序列表如下:
!Process     !Work         !Need         !Allocation   !Work+Alloc   !Finish      !
!P1          !   3   3   2!   1   2   2!   2   0   0!   5   3   2!true         !
!P3          !   5   3   2!   0   1   1!   2   1   1!   7   4   3!true         !
!P0          !   7   4   3!   7   4   3!   0   1   0!   7   5   3!true         !
!P2          !   7   5   3!   6   0   0!   3   0   2!  10   5   5!true         !
!P4          !  10   5   5!   4   3   1!   0   0   2!  10   5   7!true         !
```

存在一个安全序列<P1,P3,P0,P2,P4>

(2) P1 请求资源：P1 发出请求向量 Request1(1，0，2)，调用银行家算法检查是否能够分配?

```
Input the id of request process:1
Input request resources:1 0 2
```

输入

```
是安全状态，分配成功!

安全序列表如下:
!Process     !Work        !Need        !Allocation  !Work+Alloc  !Finish      !
!P1          !   2   3   0!   0   2   0!   3   0   2!   5   3   2!true        !
!P3          !   5   3   2!   0   1   1!   2   1   1!   7   4   3!true        !
!P0          !   7   4   3!   7   4   3!   0   1   0!   7   5   3!true        !
!P2          !   7   5   3!   6   0   0!   3   0   2!  10   5   5!true        !
!P4          !  10   5   5!   4   3   1!   0   0   2!  10   5   7!true        !


状态表:
!Process     !Max         !Allocation  !Need        !Available   !
!P0          !   7   5   3!   0   1   0!   7   4   3!   2   3   0!
!P1          !   3   2   2!   3   0   2!   0   2   0!            !
!P2          !   9   0   2!   3   0   2!   6   0   0!            !
!P3          !   2   2   2!   2   1   1!   0   1   1!            !
!P4          !   4   3   3!   0   0   2!   4   3   1!            !
```

存在一个安全序列<P1,P3,P4,P2,P0>，显示新的状态表

(3) P_4 请求资源：P_4 发出请求向量 $Request_4$(3，3，0)，系统按银行家算法进行检查：

```
Input the id of request process:4
Input request resources:3 3 0
```

输入

```
尚无足够资源，第4个进程堵塞。

状态表:
!Process     !Max         !Allocation  !Need        !Available   !
!P0          !   7   5   3!   0   1   0!   7   4   3!   2   3   0!
!P1          !   3   2   2!   3   0   2!   0   2   0!            !
!P2          !   9   0   2!   3   0   2!   6   0   0!            !
!P3          !   2   2   2!   2   1   1!   0   1   1!            !
!P4          !   4   3   3!   0   0   2!   4   3   1!            !
```

① Request4(3, 3, 0)≤Need4(4, 3, 1);

② Request4(3, 3, 0) >Available(2, 3, 0)，让 P4 堵塞等待。 状态表没有变化

(4) P_0 请求资源：P_0 发出请求向量 $Requst_0$(0，2，0)，系统按银行家算法进行检查：

```
Input the id of request process:0
Input request resources:0 2 0
```

输入

① Request0(0, 2, 0)≤Need0(7, 4, 3);② Request0(0, 2, 0)≤Available(2, 3, 0);

系统暂时先假定可为 P0 分配资源，并修改有关数据，如下图所示。

进程 \ 资源情况	Allocation A B C	Need A B C	Available A B C
P_0	0 3 0	7 2 3	2 1 0
P_1	3 0 2	0 2 0	
P_2	3 0 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

可用资源 Available(2,1,0)不能满足任何进程的需求，进入不安全状态。此时系统不分配资源给 P0。

```
不存在安全序列，不是安全状态。

状态表：
!Process      !Max          !Allocation   !Need         !Available    !
!P0           !   7   5   3!   0   1   0!   7   4   3!   2   3   0!
!P1           !   3   2   2!   3   0   2!   0   2   0!             !
!P2           !   9   0   2!   3   0   2!   6   0   0!             !
!P3           !   2   2   2!   2   1   1!   0   1   1!             !
!P4           !   4   3   3!   0   0   2!   4   3   1!             !
```

输出：找不到安全序列，状态表没有变化

(5) 若 P0 发出请求向量 Requst0(0，1，0)，系统是否将资源分配给它？

```
Input the id of request process:0
Input request resources:0 1 0
```

输入

```
安全序列表如下：
!Process      !Work         !Need         !Allocation   !Work+Alloc   !Finish       !
!P1           !   2   2   0!   0   2   0!   3   0   2!   5   2   2!true         !
!P3           !   5   2   2!   0   1   1!   2   1   1!   7   3   3!true         !
!P0           !   7   3   3!   7   3   3!   0   2   0!   7   5   3!true         !
!P2           !   7   5   3!   6   0   0!   3   0   2!  10   5   5!true         !
!P4           !  10   5   5!   4   3   1!   0   0   2!  10   5   7!true         !

状态表：
!Process      !Max          !Allocation   !Need         !Available    !
!P0           !   7   5   3!   0   2   0!   7   3   3!   2   2   0!
!P1           !   3   2   2!   3   0   2!   0   2   0!             !
!P2           !   9   0   2!   3   0   2!   6   0   0!             !
!P3           !   2   2   2!   2   1   1!   0   1   1!             !
!P4           !   4   3   3!   0   0   2!   4   3   1!             !
```

存在一个安全序列<P0,P1,P2,P3,P4>，显示新的状态表

- 使用 Micrsoft Visual Studio C++ 6.0 或 CodeBlocks 编程：程序 2_10_banksafety.cpp。完善如下程序代码：

```
#include <malloc.h>
#include <stdio.h>
#include <string.h>
#define M 3
#define N 5
```

```
int Resource[M];
int Max[N][M];
int Allocation[N][M];
int Need[N][M];
int Available[M];
int Work[M];
bool Finish[N];
int List[N];     //存放安全序列的下标序列

void initial()
//创建初始状态：先输入 Resource、Max 和 Allocation，再计算出 Need、Available。
  { 填补程序 }

void printState()
//输出当前的状态表|Process      |Max            |Allocation   |Need           |Available    |
{ 填补程序 }

int isfinish()
//返回同时满足两个条件{①Finish[i]=false;  ②Need[i][j]≤Work[j]}的进程下标  i(修改
Finish[i]=true)，否则返回-1。
{ 填补程序 }

bool issafe()
 //判定当前状态是否为安全状态（返回 true 或 false），把安全序列的下标放入 List[N]数组。
{ 填补程序 }

void printList( )
//输出安全序列表|Process |Work    |Need     |Allocation    |Work+Alloc    |Finish    |
{ 填补程序 }

void reqresource(int i, int request[M])
//表示第 i 个进程请求 M 类资源 request[M]
{
      bool flag;
      int j;
      //Step1: 判断条件 Request[j]≤Need[i][j]
      //填补程序
      //Step2: 判断条件 Request[j]≤Available[j]
      //填补程序
      //Step3: 预先分配
      //填补程序
      //Step4:检测是否为安全状态
      //填补程序
}
```

```
void main()
{
	int reqid=-1,j,req[M];
	initial();
	printState();
	if(issafe()==false)
	{
		printf("Initial state is unsafe!\n");
	}
	else
	{
		printf("\nInitial state is safe!\n");
		printList();
		printf("Input the id of request process:");
		scanf("%d",&reqid);
		while(reqid>=0 && reqid<N)   //输入进程 id 是否合法
		{
			printf("Input request resources:");
			for(j=0;j<M;j++)
			{
				scanf("%d",&req[j]);
			}
			reqresource(reqid, req);
			printState();
			printf("Input the id of request process:");
			scanf("%d",&reqid);
		}
	}
}
```