

Banker's Algorithm

(은행원 알고리즘)

1. 은행원 알고리즘

- 교착 상태(DeadLock) 회피 기법
- “안정 상태 ” 검사

2. 계산 방식

- Available : 사용 가능한 자원의 수(1차원)
- Max : 각 프로세스 자원의 최대요청량($N \times M$, 2차원)
- Allocation: 현재 각 프로세스에 할당된 자원 수($N \times M$, 2차원)
- Need: 각 프로세스의 추가 요청량($N \times M$, 2차원)

2. 동작 방식 (i번째 프로세스(P_i) 가 자원 필요할 때)

1) $\text{Request}(i) \leq \text{Need}(i)$

2) [1번 성립할 때] $\text{Request}(i) \leq \text{Available}$

3) [2번 성립할 때] P_i 에게 자원 할당을 “가정”

$\text{Available} = \text{Available} - \text{Request}(i)$

$\text{Allocation}(i) = \text{Allocation}(i) + \text{Request}(i)$

$\text{Need}(i) = \text{Need}(i) - \text{Request}(i)$

4) 안정상태 검사

3. 안전 상태 검사

1) Work \leftarrow Available 선언

Finish \leftarrow [False] * N (1차원) 선언

2) 다음 조건을 만족하는 i값을 찾는다.

Finish[i] == False && (Need(i) \leq Work)

3) 다음을 수행하고 2단계로 이동

Work = Work + Allocation(i)

Finish[i] = True

4) 모든 i에 대해 Finish[i] == True이면 안정상태

4. 예시

Process	Allocated			Maximum			Available			Need(Maximum-allocated)		
	A	B	C	A	B	C	A	B	C	A	B	C
P1	0	0	0	7	5	3	3	3	2	7	5	3
P2	2	0	0	3	2	2				1	2	2
P3	4	0	1	9	0	4				5	0	3
P4	2	1	1	2	2	2				0	1	1

Request		
A	B	C
0	1	0

① $\text{Request}(i) \leq \text{Need}(i)$

② [1번 성립할 때] $\text{Request}(i) \leq \text{Available}$

③ [2번 성립할 때] P_i 에게 자원 할당 "가정"

$\text{Available} = \text{Available} - \text{Request}(i)$

$\text{Allocation}(i) = \text{Allocation}(i) + \text{Request}(i)$

$\text{Need}(i) = \text{Need}(i) - \text{Request}(i)$

④ 안정상태 검사하여 안정상태가 맞다면 실제로 할당

Pass!

4. 예시 (안정상태 검사 진행 중!)

Process	Allocated			Maximum			Available		
	A	B	C	A	B	C	A	B	C
P1	0	1	0	7	5	3	3	3	2
P2	2	0	0	3	2	2			
P3	4	0	1	9	0	4			
P4	2	1	1	2	2	2			

0) Need 계산

1) $Work \leftarrow Available$ 선언

$Finish \leftarrow [False] * N$ (1차원) 선언

2) 다음 조건을 만족하는 i 값을 찾는다.

$Finish[i] == False \ \&\& \ (Need(i) \leq Work)$

3) 다음을 수행하고 2단계로 이동

$Work = Work + Allocation(i)$

$Finish[i] = True$

4) 모든 i 에 대해 $Finish[i] == True$ 이면 안정상태

0) Need 계산 : Maximum - Allocated

Need(Maximum - allocated)		
A	B	C
7	4	3
1	2	2
5	0	3
0	1	1

4. 예시

Process	Allocated			Maximum			Available			Need(Maximum-allocated)		
	A	B	C	A	B	C	A	B	C	A	B	C
P1	0	1	0	7	5	3	3	3	2	7	4	3
P2	2	0	0	3	2	2				1	2	2
P3	4	0	1	9	0	4				5	0	3
P4	2	1	1	2	2	2				0	1	1

0) Need 계산

1) $Work \leftarrow Available$ 선언

$Finish \leftarrow [False] * N$ (1차원) 선언

2) 다음 조건을 만족하는 i값을 찾는다.

$Finish[i] == False \ \&\& \ (Need(i) \leq Work)$

3) 다음을 수행하고 2단계로 이동

$Work = Work + Allocation(i)$

$Finish[i] = True$

4) 모든 i에 대해 $Finish[i] == True$ 이면 안정상태

1) Work, Finish 선언

Work	Finish
(3, 3, 2)	False
	False
	False
	False

4. 예시

Process	Allocated			Maximum			Available			Need(Maximum-allocated)		
	A	B	C	A	B	C	A	B	C	A	B	C
P1	0	1	0	7	5	3	3	3	2	7	4	3
P2	2	0	0	3	2	2				1	2	2
P3	4	0	1	9	0	4				5	0	3
P4	2	1	1	2	2	2				0	1	1

0) Need 계산

1) Work \leftarrow Available 선언

Finish \leftarrow [False] * N (1차원) 선언

2) 다음 조건을 만족하는 i값을 찾는다.

Finish[i] == False && (Need(i) <= Work)

3) 다음을 수행하고 2단계로 이동

Work = Work + Allocation(i)

Finish[i] = True

4) 모든 i에 대해 Finish[i] == True이면 안정상태

1) Finish[i] == False && (Need(i) <= Work)

Work	Finish
(3, 3, 2)	False
	False
	False
	False

Fail

4. 예시

Process	Allocated			Maximum			Available			Need(Maximum-allocated)		
	A	B	C	A	B	C	A	B	C	A	B	C
P1	0	1	0	7	5	3	3	3	2	7	4	3
P2	2	0	0	3	2	2				1	2	2
P3	4	0	1	9	0	4				5	0	3
P4	2	1	1	2	2	2				0	1	1

①

Pass

1) $Finish[i] == False \ \&\& \ (Need(i) \leq Work)$

Work	Finish
(3, 3, 2)	False
	False
	False
	False

②

Work	Finish
(5, 3, 2)	False
	True
	False
	False

0) Need 계산

1) $Work \leftarrow Available$ 선언

$Finish \leftarrow [False] * N$ (1차원) 선언

2) 다음 조건을 만족하는 i값을 찾는다.

$Finish[i] == False \ \&\& \ (Need(i) \leq Work)$

3) 다음을 수행하고 2단계로 이동

$Work = Work + Allocation(i)$

$Finish[i] = True$

4) 모든 i에 대해 $Finish[i] == True$ 이면 안정상태

4. 예시

Process	Allocated			Maximum			Available			Need(Maximum-allocated)		
	A	B	C	A	B	C	A	B	C	A	B	C
P1	0	1	0	7	5	3	3	3	2	7	4	3
P2	2	0	0	2	2	2				1	2	2
P3	4	0	1	9	0	4				5	0	3
P4	2	1	1	2	2	2				0	1	1

0) Need 계산

1) Work \leftarrow Available 선언

Finish \leftarrow [False] * N (1차원) 선언

2) 다음 조건을 만족하는 i값을 찾는다.

Finish[i] == False && (Need(i) <= Work)

3) 다음을 수행하고 2단계로 이동

Work = Work + Allocation(i)

Finish[i] = True

4) 모든 i에 대해 Finish[i] == True이면 안정상태

①

Pass

1) Finish[i] == False && (Need(i) <= Work)

Work	Finish
(5, 3, 2)	False
	True
	False
	False

②

Work	Finish
(7, 4, 3)	False
	True
	False
	True

4. 예시

Process	Allocated			Maximum			Available			Need(Maximum-allocated)		
	A	B	C	A	B	C	A	B	C	A	B	C
P1	0	1	0	7	5	3	3	3	2	7	4	3
P2	2	0	0	2	2	2				1	2	2
P3	4	0	1	9	0	4				5	0	3
P4	2	1	1	2	2	2				0	1	1

0) Need 계산

1) Work \leftarrow Available 선언

Finish \leftarrow [False] * N (1차원) 선언

2) 다음 조건을 만족하는 i값을 찾는다.

Finish[i] == False && (Need(i) <= Work)

3) 다음을 수행하고 2단계로 이동

Work = Work + Allocation(i)

Finish[i] = True

4) 모든 i에 대해 Finish[i] == True이면 안정상태

①

Pass

Work	Finish
(5, 3, 2)	False
	True
	False
	False

②

Work	Finish
(7, 4, 3)	False
	True
	False
	True

4. 예시

Process	Allocated			Maximum			Available			Need(Maximum-allocated)
	A	B	C	A	B	C	A	B	C	
P1	0	1	0	7	5	3	3	3	2	7 4 3
P2	2	0	0	2	2	2				1 2 2
P3	4	0	1	9	0	4				5 0 3
P4	2	1	1	2	2	2				0 1 1

0) Need 계산

1) Work \leftarrow Available 선언

Finish \leftarrow [False] * N (1차원) 선언

2) 다음 조건을 만족하는 i값을 찾는다.

Finish[i] == False && (Need(i) <= Work)

3) 다음을 수행하고 2단계로 이동

Work = Work + Allocation(i)

Finish[i] = True

4) 모든 i에 대해 Finish[i] == True이면 안정상태

①

Pass

Work	Finish
(7, 4, 3)	False
	True
	False
	True

②

Work	Finish
(7, 5, 3)	True
	True
	False
	True

4. 예시

Process	Allocated			Maximum			Available			Need(Maximum allocated)		
	A	B	C	A	B	C	A	B	C	A	B	C
P1	0	1	0	7	5	3	3	3	2	7	4	3
P2	2	0	0	3	2	2				1	2	2
P3	4	0	1	9	0	4				5	0	3
P4	2	1	1	2	2	2				0	1	1

①

Pass

0) Need 계산

1) Work \leftarrow Available 선언

Finish \leftarrow [False] * N (1차원) 선언

2) 다음 조건을 만족하는 i값을 찾는다.

Finish[i] == False && (Need(i) <= Work)

3) 다음을 수행하고 2단계로 이동

Work = Work + Allocation(i)

Finish[i] = True

4) 모든 i에 대해 Finish[i] == True이면 안정상태

Work	Finish
(7, 5, 3)	False
	True
	False
	True

②

Work	Finish
(11, 5, 4)	True
	True
	True
	True

4. 예시

Work	Finish
(11, 5, 4)	True
	True
	True
	True



안정 상태!

0) Need 계산

1) Work \leftarrow Available 선언

Finish \leftarrow [False] * N (1차원) 선언

2) 다음 조건을 만족하는 i값을 찾는다.

Finish[i] == False && (Need(i) \leq Work)

3) 다음을 수행하고 2단계로 이동

Work = Work + Allocation(i)

Finish[i] = True

4) 모든 i에 대해 Finish[i] == True이면 안정상태

5. 단점

- 1) 현재 자원 수 파악 해야함
- 2) 사용하는 최대 자원 수 알고 있어야함.
- 3) 은행원 알고리즘 계산하는데 자원이 소모됨.
- 4) 불안정 상태라고 꼭 데드락이 발생하는 것이 아님.

6. 참고 사이트

- <https://dlsdn73.tistory.com/17>
- <https://hoyeonkim795.github.io/posts/bankers/>
- <https://www.youtube.com/watch?v=pyOJDBphHkI&t=87s>