[22.03.04] 메모리 할당 시스템

TS공간할당시스템

김 태 현

문 제

0~N-1 번호를 갖는 N개의 연속된 단위 공간 존재

할당되지 않은 연속된 단위 공간의 크기를 묶어 하나의 빈 공간으로 표현

아래 두가지 명령 수행

1. 빈 공간에서 요청된 크기의 공간을 할당
   - 수용 가능한 빈 공간 중, 가장 작은 빈 공간에 할당
   - 여러 개 인 경우, 가장 앞쪽 빈 공간에 할당
2. 할당된 공간의 맨 앞 번호가 주어지면 그 공간을 빈 공간으로 바꾸고, 인접한 빈 공간은 병합

`void init(int N)`

전체 공간의 크기 N (30 ~ 100,000,000)
모두 빈 공간으로 초기화

`int allocate(int size)`

size 크기의 공간 할당
할당 후, 할당된 공간의 시작 번호 반환
할당 할 수 없는 경우 -1 반환

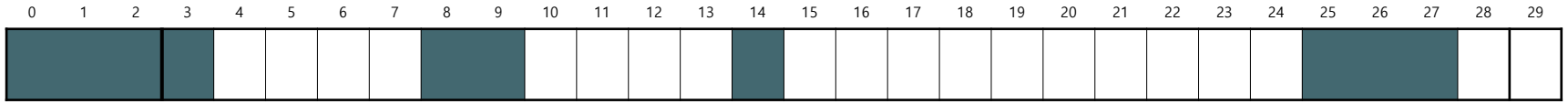
`int deallocate(int start)`

start 번호를 시작으로 하는 공간의 할당 해제
해제 후 해제한 공간 크기 반환
start 가 공간의 시작이 아니거나 이미 빈 공간이면 -1 반환

*** 함수 호출 총합 <= 30,000**

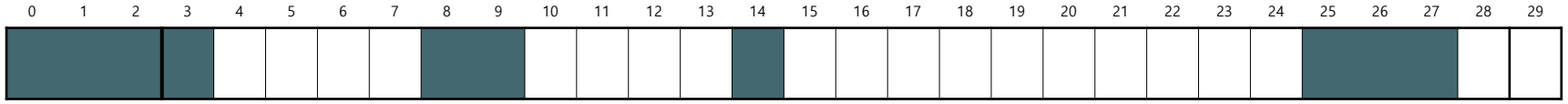
문제 분석

`int allocate(int size)`



size = 1 ~ 2 : [28:28+size)에 할당
size = 3 ~ 4 : [4:4+size)에 할당
size = 5 ~ 10 : [15:15+size)에 할당
size > 10 : 할당 불가

`int deallocate(int start)`



start = 0 : [0:3) 해제
start = 3 : [3:4) 해제 후, [3:8) 로 합병
start = 8 : [8:10) 해제 후, [4:14) 로 합병
start = 14 : [14:15) 해제 후, [10:25) 로 합병
start = 25 : [25:28) 해제 후, [15:30) 으로 합병

그 외, 할당 해제 실패

필요한 정보

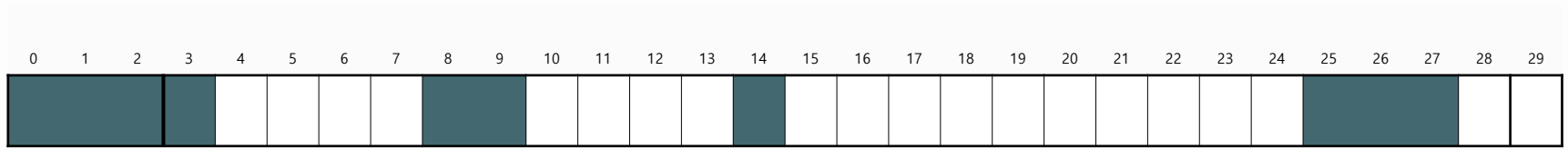
```
int allocate(int size)
```

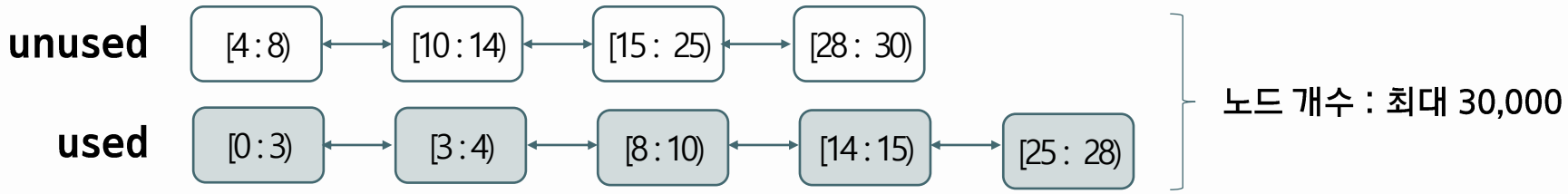
1. 크기가 size 이상인 빈 공간 중, 가장 작고 빠른 곳 구하기
2. 할당

```
int deallocate(int start)
```

1. start가 할당된 공간의 시작번호인지 파악
2. 할당 해제
3. 인접한 빈 공간 병합

1. linked list





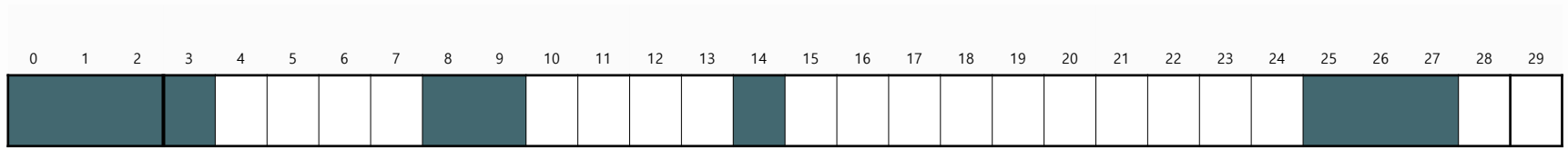

```
int allocate(int size)
```

1. 크기가 size 이상인 빈 공간 중, 가장 작고 빠른 곳 구하기 : unused linear search
2. 할당 : 1) used에 추가 , 2) unused 시작번호 변경(남은 공간 없을시 제거)

```
int deallocate(int start)
```

1. start가 할당된 공간의 시작번호인지 파악 : used linear search
2. 할당 해제 : used -> unused
3. 인접한 빈 공간 병합 : unused linear search

2. linked list + hash



unused [4 : 8) ↔ [10 : 14) ↔ [15 : 25) ↔ [28 : 30) : linked list

used {0, 3} {3, 4} {8, 10} {14, 15} {25, 28} : **hash** (key = start, value = end)

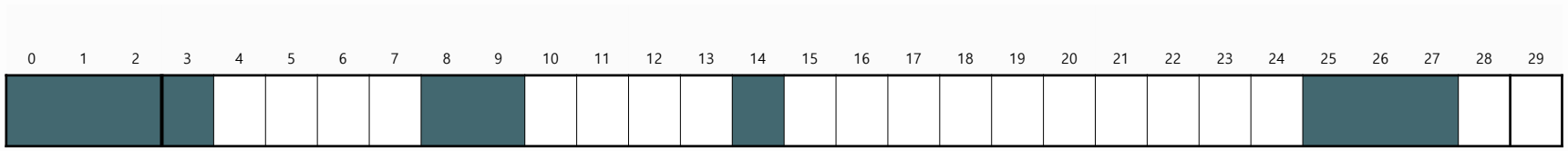
```
int allocate(int size)
```

1. 크기가 size 이상인 빈 공간 중, 가장 작고 빠른 곳 구하기 : unused linear search
2. 할당 : 1) used에 추가 , 2) unused 시작번호 변경(남은 공간 없을시 제거)

```
int deallocate(int start)
```

1. start가 할당된 공간의 시작번호인지 파악 : used에서 key값 start 검색하여 판별, [s : e) 구함
2. 할당 해제 : used -> unused
3. 인접한 빈 공간 병합 : unused linear search

3. set + hash



unused {2, 28} {4, 4} {4, 10} {10, 15} : **set** (first = size, second = start)

used {0, 3} {3, 4} {8, 10} {14, 15} {25, 28} : **hash** (key = start, value = end)

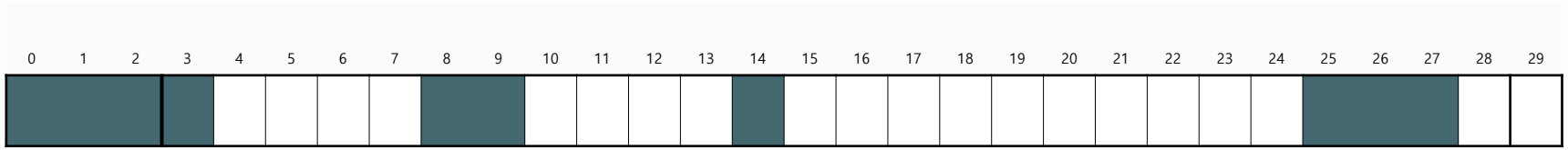
```
int allocate(int size)
```

1. 크기가 size 이상인 빈 공간 중, 가장 작고 빠른 곳 구하기 : unused lower_bound({size, 0})
2. 할당 : 1) used에 추가 , 2) unused 삭제 후, 남은 공간 있을시 추가

```
int deallocate(int start)
```

1. start가 할당된 공간의 시작번호인지 파악 : used에서 key값 start 검색하여 판별, [s : e) 구함
2. 할당 해제 : used 에서 삭제
3. 인접한 빈 공간 병합 unused linear search 로 병합할 [s: e) 구하여 unused 업데이트

4. set + set iterator + hash



sit[28], eit[30] sit[4], eit[8] sit[10], eit[14] sit[15], eit[25]

sit[start] = unused의 start로 시작하는 노드 iterator
eit[end] = unused의 end-1로 끝나는 노드 iterator

unused {2, 28} {4, 4} {4, 10} {10, 15} : **set** (first = size, second = start)

used {0, 3} {3, 4} {8, 10} {14, 15} {25, 28} : **hash** (**key** = start, **value** = end)

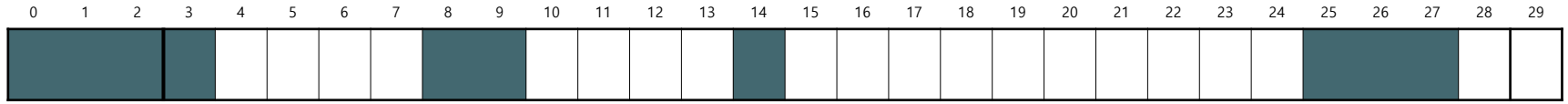
```
int allocate(int size)
```

1. 크기가 size 이상인 빈 공간 중, 가장 작고 빠른 곳 구하기 : unused lower_bound({size, 0})
2. 할당 : 1) used에 추가 , 2) unused 삭제 후, 남은 공간 있을시 추가

```
int deallocate(int start)
```

1. start가 할당된 공간의 시작번호인지 파악 : used에서 key값 start 검색하여 판별, [s : e) 구함
2. 할당 해제 : used 에서 삭제
3. 인접한 빈 공간 병합 : 왼쪽 확인 : eit[s] 가 존재하는지 , 오른쪽 확인 : sit[e] 가 존재하는지
   존재하면 병합 후 unused, sit, eit 업데이트

Example - allocate(3)

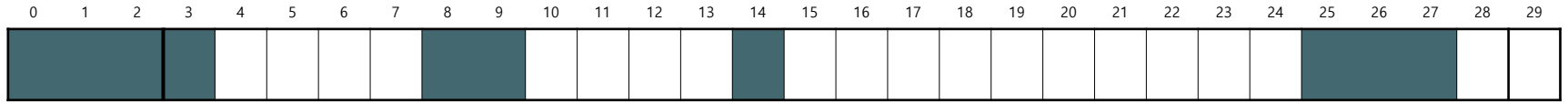


sit[28], eit[30] sit[4], eit[8] sit[10], eit[14] sit[15], eit[25]

unused {2, 28} {4, 4} {4, 10} {10, 15} : **set** (first = size, second = start)

used {0, 3} {3, 4} {8, 10} {14, 15} {25, 28} : **hash** (**key** = start, **value** = end)

Example - deallocate(3)



sit[28], eit[30] sit[4], eit[8] sit[10], eit[14] sit[15], eit[25]

unused {2, 28} {4, 4} {4, 10} {10, 15} : **set** (first = size, second = start)

used {0, 3} {3, 4} {8, 10} {14, 15} {25, 28} : **hash** (**key** = start, **value** = end)


감사합니다