동국대학교
dongguk university

실습: Week 10

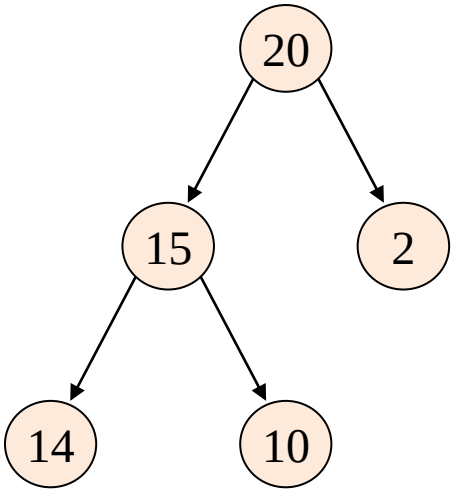
Data Structures

Contents

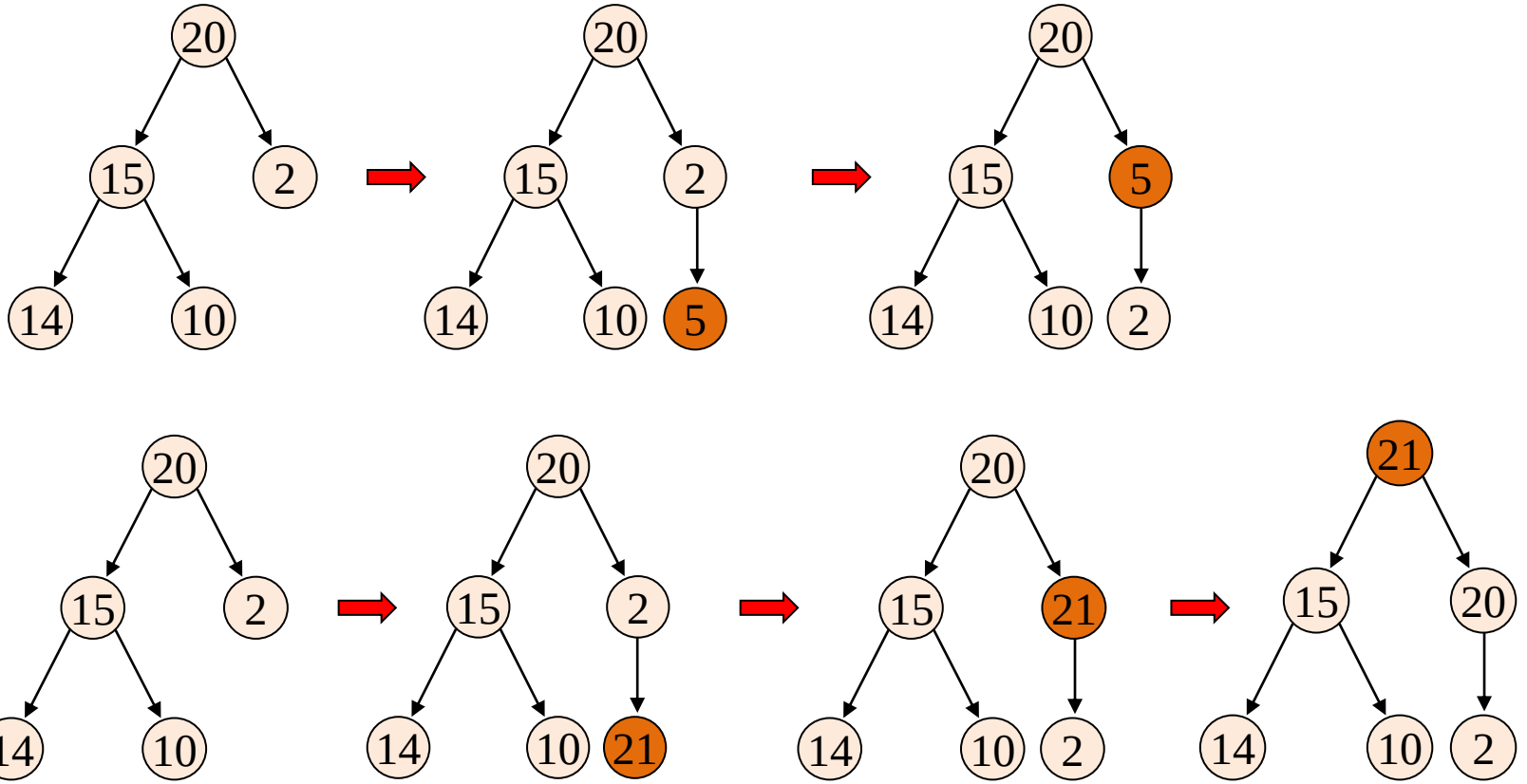
- Heap
  - Max_Heap 구현
- 실습
  - 실습 10-1. Max Heap
  - 실습 10-2. Priority Queue Simulation

Heap

- 최대히프(max heap)
  - 각 노드의 키 값이 그 자식의 키 값보다 작지 않은 완전 이진 트리

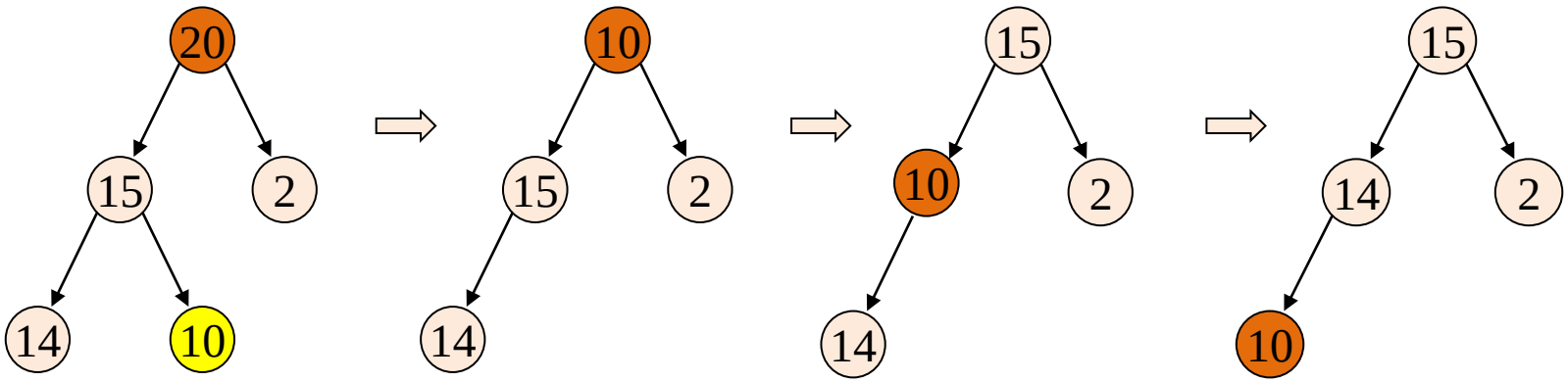


Insertion in Max Heap



Deletion in Max Heap

- 히프에서 최대값을 갖는 원소를 삭제하고 반환



실습 10-1. Max heap

- Heap : key와 data를 가진 max heap 구현
- Heap 삽입, 삭제 함수 구현
- 명령어
  - I: Insert data - <key, data>
  - D: Delete max data
  - P : Print heap
  - Q : Quit

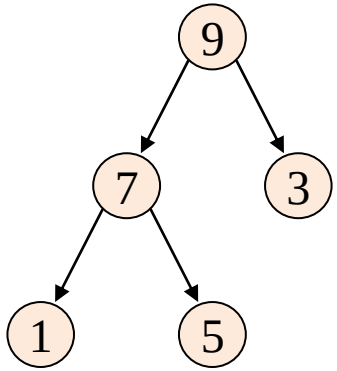
자료구조 및 함수

```
typedef struct {
        int             key;
        char            data;
} Element;
```

key	data

```
Element heap[MAX_DATA];
int heap_size = 0;
```

- void insert_max_heap(Element item);
  - 히프에 item(key, data) 삽입
  - 예 : (3,B) 삽입
    (1,A) 삽입
    (7,D) 삽입
    (9,E) 삽입
    (5,C) 삽입



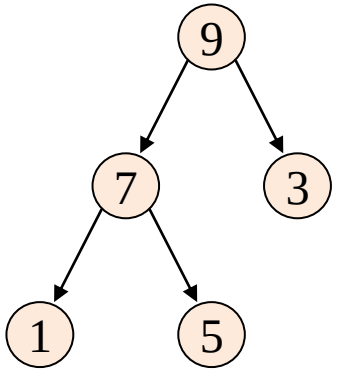
0	-	-
1	9	E
2	7	D
3	3	B
4	1	A
5	5	C
6	-	

자료구조 및 함수

- Element delete_max_heap();
  - 히프에서 max item (루트) 삭제 및 반환
- void max_heap_show();
  - 히프의 자료들을 차례로 출력
- boolean is_heap_empty();

실습 10-1. 실행 예

```
************* Command ************
I: Insert data, D: Delete max data
P: Print heap, Q: Quit
**********************************

Command> i
 key and data: 3 B

Command> i
 key and data: 1 A

Command> i
 key and data: 7 D

Command> i
 key and data: 9 E

Command> i
 key and data: 5 C

Command> p
 9  E
 7  D
 3  B
 1  A
 5  C
```



실습 10-1. 실행 예

```
Command> p
  9  E
  7  D
  3  B
  1  A
  5  C

Command>
Command> d
 Max: key 9, data E

Command> d
 Max: key 7, data D

Command> d
 Max: key 5, data C

Command> d
 Max: key 3, data B

Command> d
 Max: key 1, data A

Command> d
Heap is empty
```

max_heap.h

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#define    MAX_SIZE 100
#define    boolean     int
#define    true        1
#define    false       0

typedef struct Element {
            int         key;
            char        data;
} Element;

// Global heap
Element heap[MAX_SIZE];
int heap_size = 0;
// 히프에 item(key, data) 삽입
void insert_max_heap(Element item);
// 히프에서 max item (루트) 삭제 및 반환
Element delete_max_heap();
// 히프의 자료들을 차례로 출력
void max_heap_show();
boolean is_heap_empty();
```

max_heap.c - main()함수

```
#include "max_heap.h"

void main() {
        char    c, data;
        int     key;
        Element item;

        printf("************** Command ************ \n");
        printf("I: Insert data, D: Delete max data \n");
        printf("P: Print heap, Q: Quit            \n");
        printf("************************************ \n");

        while (1) {
                printf("\nCommand> ");
                c = _getche();
                c = toupper(c);
                switch (c) {
```

max_heap.c - main()함수

```
case 'I':
                        printf("\n key and data: ");
                        scanf("%d %c", &key, &data);
                        item.key = key;
                        item.data = data;
                        insert_max_heap(item);
                        break;
                case 'D':
                        if (is_heap_empty())
                                printf("\nHeap is empty\n");
                        else {
                                item = delete_max_heap();
                                printf("\n Max: key %d, data %c \n", item.key,
item.data);
                        }
                        break;
```

max_heap.c - main()함수

```
                case 'P':
                        printf("\n");
                        max_heap_show();
                        break;
                case 'Q':
                        printf("\n");
                        exit(1);
                default: break;
                }
        }
}
```

실습 10-2. Priority queue simulation

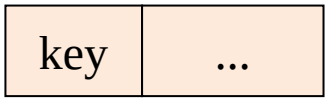
- Priority queue simulation
  - Priority queue를 이용한 프린터 시뮬레이션
- Min heap
  - key값(duration)이 작을수록 우선 순위가 높음
- 시뮬레이션 방식
  - current_time을 증가시키면서 매 시각 가상의 프린트 job을 처리

```
while(current_time < MAX_SIMUL_TIME) {
        ...    ++current_time;
}
```

  - 시뮬레이션 종료 후 프린트 job 들의 평균 지연 시간 출력

자료구조 및 함수

```
typedef  struct {
    int    key;      // Priority queue의 키 값 (duration)
    ...
} Job;
typedef  Job Element;

Element  PQ[MAX_PQ_SIZE];  // min heap
int  PQ_size = 0;
```



자료구조 및 함수

- void insert_PQ(Element item)
  - PQ에 job 삽입
- Element delete_PQ()
  - PQ에서 min item (루트) 삭제 및 반환
- void PQ_show()
  - PQ의 job들의 key와 id를 차례로 출력
- boolean is_PQ_empty()

실습 10-2. 실행 예

```
----- time 0 -----
 새 jop <1>이 들어 왔습니다. key(출력 시간) = 6 입니다.
 프린트를 시작합니다 - jop <1>...
현재 프린트 큐(key,id) : [ ]

----- time 1 -----
 새 jop <2>이 들어 왔습니다. key(출력 시간) = 9 입니다.
 아직 Jop <1>을 프린트하고 있습니다 ...남은 시간 : 5
현재 프린트 큐(key,id) : [ (9  2) ]

----- time 2 -----
 아직 Jop <1>을 프린트하고 있습니다 ...남은 시간 : 4
현재 프린트 큐(key,id) : [ (9  2) ]

----- time 3 -----
 새 jop <3>이 들어 왔습니다. key(출력 시간) = 4 입니다.
 아직 Jop <1>을 프린트하고 있습니다 ...남은 시간 : 3
현재 프린트 큐(key,id) : [ (4  3) (9  2) ]

----- time 4 -----
 아직 Jop <1>을 프린트하고 있습니다 ...남은 시간 : 2
현재 프린트 큐(key,id) : [ (4  3) (9  2) ]

----- time 5 -----
 아직 Jop <1>을 프린트하고 있습니다 ...남은 시간 : 1
현재 프린트 큐(key,id) : [ (4  3) (9  2) ]

----- time 6 -----
 프린트를 시작합니다 - jop <3>...
현재 프린트 큐(key,id) : [ (9  2) ]

----- time 7 -----
 새 jop <4>이 들어 왔습니다. key(출력 시간) = 9 입니다.
 아직 Jop <3>을 프린트하고 있습니다 ...남은 시간 : 3
현재 프린트 큐(key,id) : [ (9  2) (9  4) ]
```

실습 10-2. 실행 예

```
----- time 13 -----
 아직 Jop <6>을 프린트하고 있습니다 ...남은 시간 : 2
현재 프린트 큐(key,id) : [ (2  7) (9  4) (9  2) ]

----- time 14 -----
 새 jop <8>이 들어 왔습니다. key(출력 시간) = 2 입니다.
 아직 Jop <6>을 프린트하고 있습니다 ...남은 시간 : 1
현재 프린트 큐(key,id) : [ (2  7) (2  8) (9  2) (9  4) ]

----- time 15 -----
 새 jop <9>이 들어 왔습니다. key(출력 시간) = 1 입니다.
 프린트를 시작합니다 - jop <9>...
현재 프린트 큐(key,id) : [ (2  8) (2  7) (9  2) (9  4) ]

----- time 16 -----
 새 jop <10>이 들어 왔습니다. key(출력 시간) = 6 입니다.
 프린트를 시작합니다 - jop <8>...
현재 프린트 큐(key,id) : [ (2  7) (6  10) (9  2) (9  4) ]

----- time 17 -----
 아직 Jop <8>을 프린트하고 있습니다 ...남은 시간 : 1
현재 프린트 큐(key,id) : [ (2  7) (6  10) (9  2) (9  4) ]

----- time 18 -----
 프린트를 시작합니다 - jop <7>...
현재 프린트 큐(key,id) : [ (6  10) (9  4) (9  2) ]

----- time 19 -----
 아직 Jop <7>을 프린트하고 있습니다 ...남은 시간 : 1
현재 프린트 큐(key,id) : [ (6  10) (9  4) (9  2) ]

완료된 프린트 job = 7 개
평균 지연 시간    = 1.571429 단위시간
```

priority_queue_simulation.h

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// 시뮬레이션 설정 상수
#define   MAX_SIMUL_TIME  20                    // 시뮬레이션 진행 시간
#define   MAX_PRINTING_TIME       10            // 각 Job의 가능한 최대 프린트 시간
#define   JOB_ARRIVAL_PROB        0.5           // 매 시각 새로운 Job의 도착 확률
#define   boolean     int
#define   true        1
#define   false       0
#define   MAX_PQ_SIZE 1000                      // Priority queue size

// 시뮬레이션을 위한 global variables
int current_time = 0;             // 현재 시각
int new_job_id = 0;               // 새로운 Job의 ID
int current_job_id;               // 현재 프린트하고 있는 Job의 ID
int remaining_time;               // 현재 프린트하고 있는 Job의 남은 프린트 시간. 매 시각 1씩 감소
int total_wait_time;              // 프린트를 시작한 모든 Job의 대기시간의 합
int num_printed_jobs;             // 시뮬레이션이 끝날 때까지 프린트가 시작된 Job의 총 수
```

priority_queue_simulation.h

```
// Job
typedef struct {
        int     key;            // Priority queue의 키 값 (duration을 키로 설정)
        int     id;             // Job ID
        int     arrival_time;   // Job이 요청된(도착한) 시간
        int     duration;       // Job의 프린트 시간
} Job;
typedef Job Element;

// Global PQ (priority queue): min heap
// key값(duration)이 작을 수록 우선순위가 높음
Element PQ[MAX_PQ_SIZE];
int PQ_size = 0;

// ID가 id, 요청시간이 arrival_time, 프린트 시간이 duration인 Job을 큐에 삽입
void insert_job(int id, int arrival_time, int duration);
// 다음 job을 큐에서 꺼내 수행(현재 job id, remaining time 등 설정)
void process_next_job();
```

priority_queue_simulation.h

```
// 랜덤하게 true 혹은 false를 return. True일 확률은 JOB_ARRIVAL_PROB
boolean is_job_arrived();
// 프린터가 놀고 있으면(현재 job의 remaining time <= 0) true
boolean is_printer_idle();

double random();                    // 0.0 - 1.0 사이의 랜덤 값을 반환
int get_random_duration();          // 1 - MAX_PRINTING_TIME+1 사이의 랜덤 값을 반환

// 구현할 함수

// PQ에 job 삽입
void insert_PQ(Element item);
// PQ에서 min item (루트) 삭제 및 반환
Element delete_PQ();
// PQ의 job들의 key와 id를 차례로 출력
void PQ_show();
boolean is_PQ_empty();
```

priority_queue_simulation.c

```
#include "priority_queue_simulation.h"

void main ()
{
        int duration;

//      srand(time(NULL));

        while(current_time < MAX_SIMUL_TIME) {
                printf("\n----- time %d ----- \n", current_time);

                // 새 job이 들어오면 큐에 삽입
                if (is_job_arrived()) {
                        ++new_job_id;
                        duration = get_random_duration();
                        insert_job(new_job_id, current_time, duration);
                }
```

priority_queue_simulation.c

```
                // 프린터가 놀고 있으면 다음 job을 수행
                if (is_printer_idle()) {
                        if (!is_PQ_empty()) process_next_job();
                }
                // 아직 프린트 중이면, 현재 job의 remaining time을 하나 줄임
                else {
                   printf(" 아직 Jop <%d>을 프린트하고 있습니다 ...남은 시간 : %d \n",
                           current_job_id, remaining_time);
                   --remaining_time;
                }

                // 현재 큐의 상태를 보여줌
                PQ_show();
                ++current_time;
        }

        // MAX_TIME이 지난 후 통계 자료 출력
        printf("\n완료된 프린트 job = %d 개 \n", num_printed_jobs);
        printf("평균 지연 시간    = %f 단위시간 \n\n", (double) total_wait_time /
        num_printed_jobs);
}
```

그 외 함수

```
// ID가 id, 요청시간이 arrival_time, 프린트 시간이 duration인 Job을 큐에 삽입
void insert_job(int id, int arrival_time, int duration)
{
        Job p;

        p.key = duration;
//      p.key = 1;
        p.id = id;
        p.arrival_time = arrival_time;
        p.duration = duration;

        insert_PQ(p);
        printf(" 새 jop <%d>이 들어 왔습니다. key(출력 시간) = %d 입니다. \n", id, duration);
}
```

그 외 함수

```
// 다음 job을 큐에서 꺼내 수행(현재 job id, remaining time 등 설정)
void process_next_job()
{
        Job p;

        p = delete_PQ();

        // Set global variables
        current_job_id = p.id;
        remaining_time = p.duration - 1;
        total_wait_time += current_time - p.arrival_time;
        ++num_printed_jobs;

        printf(" 프린트를 시작합니다 - jop <%d>... \n", current_job_id);
}
```

그 외 함수

```
// 랜덤하게 true 혹은 false. True일 확률은 ARRIVAL_PROB
boolean is_job_arrived()
{
        if(random() < JOB_ARRIVAL_PROB)
                return true;
        else
                return false;
}


// 프린터가 놀고 있으면(현재 job의 remaining time <= 0) true
boolean is_printer_idle()
{
        if(remaining_time <= 0)
                return true;
        else
                return false;
}
```

그 외 함수

```
// 0.0 - 1.0 사이의 랜덤 값을 반환
double random()
{
        return rand()/(double)RAND_MAX;
}


// 1 - MAX_PRINTING_TIME+1 사이의 랜덤 값을 반환
int get_random_duration()
{
        return (int)(MAX_PRINTING_TIME * random()) + 1;
}
```