자료구조 5강. 힙(Heap)

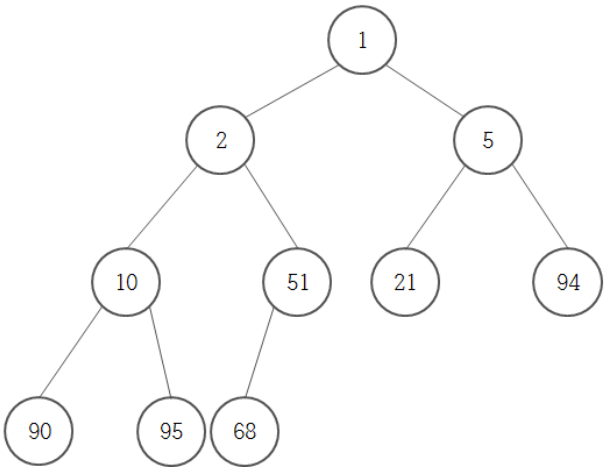
2013.05.31 20:25 | 수학/알고리즘/자료구조

[자료구조 강좌]

특별한 트리를 기본으로 하는 자료구조!

힙(Heap)

오늘은 '힙(Heap)'이란 자료구조에 대해서 알아보려고 합니다. 이 힙(Heap)이란 자료구조는 위키백과에 따르면 '특별한 트리를 기본으로 하는 자료구조이다.'라고 설명되어 있습니다. 여기서 특별한 트리란 우리가 전에 배운 완전 이진 트리를 말하며, 힙 자료구조는 최대 힙(Max Heep)과 최소 힙(Min Heep)으로 나뉘며 이러한 힙은 최대값 또는 최소값을 짧은 시간내에 찾기 위해서 만들어진 자료구조입니다. 최대 힙이란 부모 노드의 값이 항상 자식 노드의 값보다 크다는 것이며, 최소 힙은 부모 노드의 값이 항상 자식 노드의 값보다 작다는 것입니다. 예를 들어, 부모 노드의 값이 항상 자식 노드의 값보다 작은 최소 힙(Min Heap)은 아래와 같은 구조를 지닙니다.

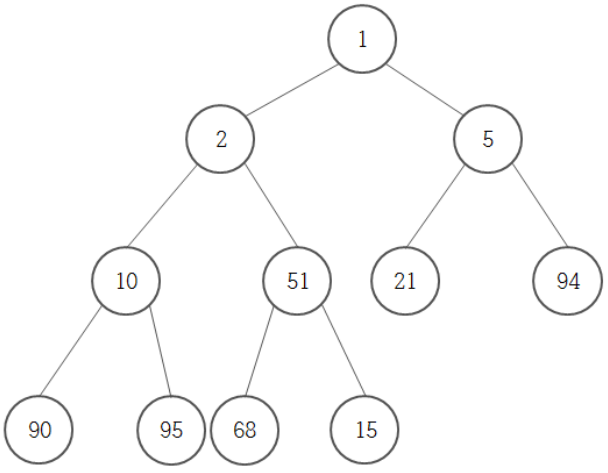


<최소 힙(Min Heap)>

위의 최소 힙을 살펴보시면 모든 자식 노드가 부모 노드보다 크다는 것을 알 수 있습니다. 반대로 모든 부모 노드는 자식 노드보다 값이 작다는 것을 알 수 있습니다.

최소 힙(Min Heap)에서의 삽입 과정

이제 저 위의 힙에서 새 노드 15를 삽입하려면 어떻게 해야 할까요? 우선은 완전 이진 트리를 유지하기 위해 51의 오른쪽 자식 노드로 추가합니다. 이는 힙에서 노드가 추가될때 최고 깊이에서 가장 오른쪽에 노드가 추가되는 것입니다.



그리고 추가된 노드 15를 부모 노드와 비교합니다. 이것은 최소 힙이므로, 모든 자식 노드가 부모 노드보다

ABOUT WRITER

아무리 어려운 내용이라도 쉽게 설명할 수 있을 때 비로소 아는 것이라 굳게 믿고 있습니다. 최근에는 점점 바빠지기 시작하여 제때 글을 올리질 못하고 있지만, 최대한 시간을 내서 블로그를 다시 전과 같이 관리할 수 있도록 노력하겠습니다. (_ _) (카카오톡: su6net, 메일: su6net@gmail.com)

지금까지 진행한 프로그래밍 강좌..

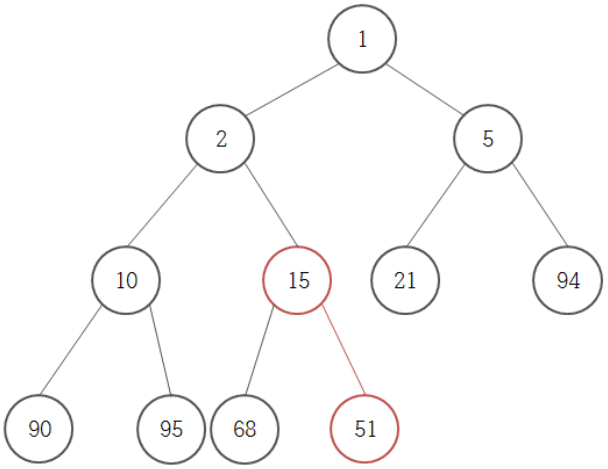
태그로그 위치로그 미디어로그 방명록

CATEGORY

분류 전체보기 (290)
잡담 (43)
프로그래밍 관련 (137)
C/C++ (31)
C++/STL (2)
C# (24)
HTML/CSS (7)
Python (20)
자바/자바 스크립트 (26)
정규 표현식 (9)
비주얼 베이직 (9)
게임메이커 (2)
MFC/API (6)
서버/네트워크 (12)
윈도우즈 (4)
프로그램 (17)
AI/VR (0)
소스 관련 (37)
워게임 (7)
수학 (13)
알고리즘/자료구조 (12)
스터디 (16)
Direct 3D (8)
어셈블리 (3)
리버스 엔지니어링 (5)
문제 풀이 (2)

RECENT POST

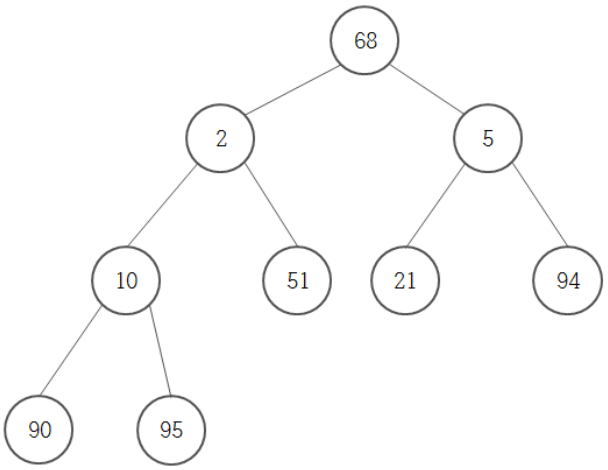
크도록 하기 위해 15와 51을 서로 교환합니다.



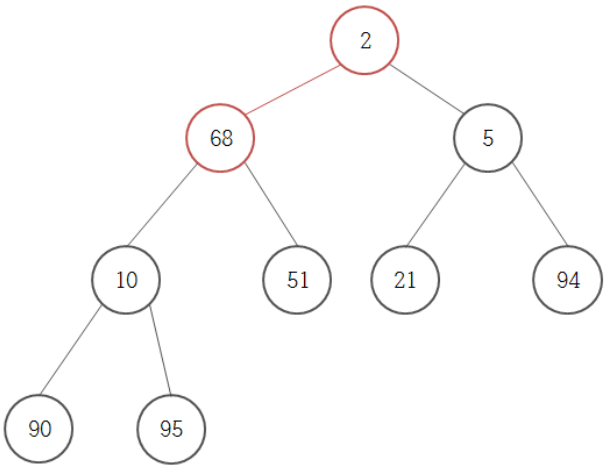
그리고 다시 부모 노드인 2와 자식 노드인 15를 비교하게 됩니다. 여기서는 자식 노드 15가 부모 노드인 2보다 크기 때문에 삽입 과정은 여기서 마무리 됩니다.

최소 힙(Min Heap)에서의 최소값 삭제(Extract Min) 과정

최소 힙에서 최소값을 뽑아낸다는건 루트 노드를 뽑아낸다는 것과 같습니다. 최소값 삭제가 이루어지면, 부모 노드가 자식 노드보다 작도록 하기 위해 복원 과정을 거칩니다. 우선, 맨 위에서 예를 든 힙에서 최소값이 제거된다면 힙의 최고 깊이에 있는 가장 오른쪽의 노드를 루트 노드로 옮깁니다.



그리고 루트 노드인 68은 자식 노드인 2와 5 중에 가장 작은 자식 노드인 2와 교환을 합니다.



다시 부모 노드인 68과 자식 노드인 10과 51 중에 가장 작은 자식 노드인 10과 교환을 합니다.

RECENT COMMENT

이해잘되게 예시 잘써주셔서 감사합니다!..
호 at 09.14

알겟습니닷!
ddd at 09.10

설명을 쉽게하셔서 이해가 잘 가네요 감..
초보 at 09.09

좋은 강의 감사합니다. 예제도 해주시고..
학생 at 09.07

메소드가 C언어의 함수 개념과 비슷한것..
테오 at 09.06

C:₩Study>javac JavaTutorial1.java 'ja..
ㅇㅅㅇ at 09.02

감사합니다! window 10 여러분들은 -..
ㅇㅅㅇ at 09.02

AP 컴퓨터 싸이언스를 듣기 시작하는 학..
ㅇㅅㅇ at 09.01

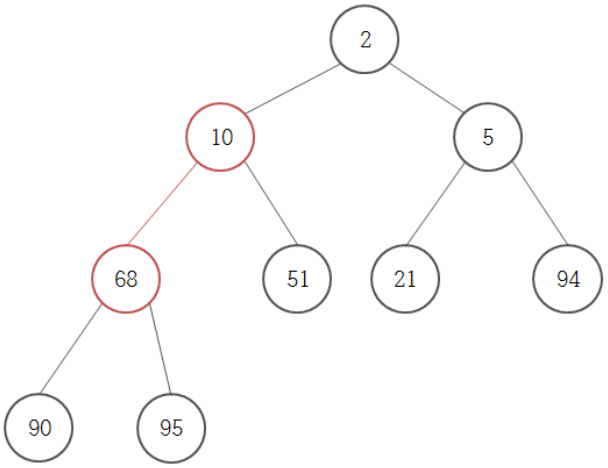
각 path마다 ; (세미콜론)으로 구분을..
ldk at 08.30

좋은 설명 감사합니다~
qwd at 08.25

RECENT TRACKBACK

왜 가상함수(Virtual Function)을 쓰는가?
PATHOS at 2013

로그인,쪽지,메일
관리자 at 2012



다시 부모 노드인 68과 자식 노드인 90과 95를 비교하게 되는데, 두 노드 모두 부모 노드인 68보다 크기 때문에 더이상 교환이 일어나지 않습니다. 여기서 최소값 삭제 과정은 끝이 납니다.

최소 힙(Min Heap)의 구현

최소 힙을 구현할때는 배열이 가장 힙을 표현하기가 좋으므로, 배열로 최소 힙을 구현할 것이며 배열로 힙을 구현하게 된다면 힙의 마지막 노드의 위치를 빠르게 알 수 있고, 완전 이진 트리이므로 따로 링크를 보관하지 않아도 된다는 등의 장점을 지니고 있습니다. 완전 이진 트리를 배열로 나타낼 때, 깊이 n의 노드는 배열의 2^n-1 ~ 2^(n+1)-2번 요소에 저장을 하게 됩니다. 물론 깊이 0의 루트 노드는 배열의 0번째 요소로 들어갑니다. 먼저 최소 힙(Min Heap)을 구현하면서 차차 살펴보도록 합시다. 제일 첫번째로, Heap 클래스를 보도록 합시다.

```
class Heap
{
public:
    Heap(int Capacity);
    ~Heap();
    void Insert(int data);
    void extractMin();
    void Swap(int* a, int* b);
    void Output();
private:
    int* heap;
    int capacity;
    int usedsize;
};
```

Heap 클래스에는 용량을 전달받는 생성자와 소멸자, 삽입 연산을 위한 메서드, 최소값 제거 연산을 위한 메서드, 교환을 위한 메서드, 출력을 위한 메서드와 힙 포인터, 용량, 사용 용량 변수가 존재합니다. 먼저 생성자와 소멸자 부분을 보도록 하겠습니다.

```
Heap::Heap(int Capacity) : capacity(Capacity)
{
    heap = new int[Heap::capacity];
    usedsize = 0;
}

Heap::~Heap()
{
    delete[] heap;
}
```

위의 Heap 생성자에서는 넘어온 Capacity 값으로 멤버 capacity를 초기화합니다. 그리고 new 연산을 통해 capacity개의 int형 데이터를 저장할 공간을 마련하였습니다. 사용 용량은 Heap에 아무런 데이터가 없으므로 0으로 값을 초기화해줍니다. 소멸자에서는 new 연산으로 할당한 메모리 공간을 delete 연산으로 해제합니다. 다음으로 Insert 메서드를 보도록 하겠습니다.

```
void Heap::Insert(int data)
{
    int usedPos = usedsize;
    // 여기서 usedPos번 인덱스에 위치한 노드의 부모 노드가 위치한 인덱스는 (usedPos - 1)
    // 만약 5번 인덱스에 위치한 노드의 부모 노드가 위치한 인덱스를 구하고 싶다면, (5 - 1)
    int parentPos = (int) ((usedPos - 1) / 2);
    // 사용 용량과 최대 용량이 똑같을 경우 힙이 가득 찼다는 것이므로 함수를 끝냅니다.
    if (usedsize == capacity)
    {
        cout << "힙이 가득 찼습니다." << endl;
        return;
    }

    // 배열의 마지막에 데이터를 집어 넣습니다.
    heap[usedsize] = data;
    // 부모 노드의 값이 자식 노드의 값보다 클 경우
    while (usedPos > 0 && heap[usedPos] < heap[parentPos])
    {
        // 부모 노드와 자식 노드를 서로 교환합니다.
        Swap(&heap[usedPos], &heap[parentPos]);
        // 그리고 현재 위치는 부모 노드의 위치입니다.
        usedPos = parentPos;
        // 부모 노드의 위치를 다시 구합니다.
        parentPos = (int) ((usedPos - 1) / 2);
    }
    // 사용 용량을 1 증가시킵니다.
    usedsize++;
}
```

이 Insert 메서드는 위에서 말한 최소 힙에서의 삽입 과정과 똑같습니다. 배열의 마지막에 노드가 추가되고, 마지막 노드의 부모 노드와 값을 비교하여 부모 노드가 크면 교환이 일어나고, 작으면 교환이 일어나지 않고 반복문이 종료됩니다. 간단하죠? 그럼, 다음으로 extractMin 메서드를 보도록 합시다.

```
void Heap::extractMin()
{
    // 사용 용량이 0인 것은 힙이 비어있다는 것으로, 함수를 종료합니다.
    if (usedsize == 0) return;
    // 루트 노드의 인덱스는 0, 왼쪽 자식 노드의 인덱스는 1, 오른쪽 자식 노드의 인덱스는 2
    int parentPos = 0, leftPos = 1, rightPos = 2;

    // 루트 노드를 비웁니다.
    heap[0] = NULL;
    // 사용 용량이 줄어들었으므로 1을 감소시킵니다.
    usedsize--;
    // 루트 노드가 있던 자리에 마지막에 있는 노드를 이동시킵니다.
    Swap(&heap[0], &heap[usedsize]);
    // 무한 루프에 빠집니다.
    while (true)
    {
        // 선택된 자식 노드의 인덱스를 0으로 초기화 시킨다.
        int selectChild = 0;

        // 왼쪽 자식 노드의 인덱스가 사용 용량과 같거나 크다면 루프를 빠져나온다.
        if (leftPos >= usedsize) break;
        // 오른쪽 자식 노드의 인덱스가 사용 용량과 같거나 크다면 선택된 자식 노드의 인덱스
        if (rightPos >= usedsize) selectChild = leftPos;
        // 오른쪽 자식 노드의 인덱스가 사용 용량보다 작을 경우
        else {
            // 왼쪽 자식 노드의 값이 오른쪽 자식 노드의 값보다 크다면 선택된 자식 노드의
            if (heap[leftPos] > heap[rightPos]) selectChild = rightPos;
            // 그 반대의 경우는 선택된 자식 노드의 인덱스가 왼쪽 노드의 인덱스다.
            else selectChild = leftPos;
        }

        // 부모 노드가 선택된 자식 노드보다 클 경우
        if (heap[selectChild] < heap[parentPos])
        {
            // 부모 노드와 선택된 자식 노드를 서로 교환한다.
            Swap(&heap[parentPos], &heap[selectChild]);
            // 부모 노드의 인덱스는 선택된 자식 노드의 인덱스이다.
            parentPos = selectChild;
        }
        // 부모 노드가 선택된 자식 노드보다 작을 경우 루프를 탈출한다.
        else break;

        // 왼쪽 자식 노드의 인덱스를 구해온다.
        leftPos = 2 * parentPos + 1;
        // 오른쪽 자식 노드의 인덱스는 왼쪽 자식 노드의 인덱스에 1을 더한것과 같다.
        rightPos = leftPos + 1;
    }
}
```

위의 extractMin 메서드는 위에서 설명드렸던 최소값 제거 과정과 똑같습니다. 루트 노드가 최소값을 지니므로, 루트 노드를 제거하고 루트 노드가 있던 자리에 최고 깊이의 가장 우측 노드, 즉 마지막 노드가 옵니다. 그리고 '모든 자식 노드는 부모 노드보다 크다'를 만족하기 위해 복원 과정을 거칩니다. 이 복원 과정은 루트 노드에 새롭게 온 데이터를 가지고 좌측, 우측 자식 노드의 값보다 크다면 교환이 일어납니다. 이 과정을 계속 반복하여 새롭게 온 데이터는 자기 자리를 찾아갑니다. 한번 전체 예제를 보도록 합시다.

```
#include <iostream>
#include <cstdlib>

using namespace std;

class Heap
{
public:
    Heap(int Capacity);
```

```
    ~Heap();
    void Insert(int data);
    void extractMin();
    void Swap(int* a, int* b);
    void Output();
private:
    int* heap;
    int capacity;
    int usedsize;
};

Heap::Heap(int Capacity) : capacity(Capacity)
{
    heap = new int[Heap::capacity];
    usedsize = 0;
}

Heap::~Heap()
{
    delete[] heap;
}

void Heap::Swap(int* a, int* b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

void Heap::Insert(int data)
{
    int usedPos = usedsize;
    int parentPos = (int) ((usedPos - 1) / 2);
    if (usedsize == capacity)
    {
        cout << "힙이 가득 찼습니다." << endl;
        return;
    }

    heap[usedsize] = data;
    while (usedPos > 0 && heap[usedPos] < heap[parentPos])
    {
        Swap(&heap[usedPos], &heap[parentPos]);
        usedPos = parentPos;
        parentPos = (int) ((usedPos - 1) / 2);
    }
    usedsize++;
}

void Heap::extractMin()
{
    if (usedsize == 0) return;
    int parentPos = 0, leftPos = 1, rightPos = 2;

    heap[0] = NULL;
    usedsize--;
    Swap(&heap[0], &heap[usedsize]);
    while (true)
    {
        int selectChild = 0;

        if (leftPos >= usedsize) break;
        if (rightPos >= usedsize) selectChild = leftPos;
        else {
            if (heap[leftPos] > heap[rightPos]) selectChild = rightPos;
            else selectChild = leftPos;
        }

        if (heap[selectChild] < heap[parentPos])
        {
            Swap(&heap[parentPos], &heap[selectChild]);
            parentPos = selectChild;
        }
        else break;

        leftPos = 2 * parentPos + 1;
        rightPos = leftPos + 1;
    }
}

void Heap::Output()
{
    for (int i = 0; i < usedsize; i++)
        cout << heap[i] << " ";
    cout << endl;
}

int main()
{
    int maxSize = 6;
    int input;
    Heap heap(maxSize);

    for(int i = 0; i < maxSize; i++)
    {
        cin >> input;
        heap.Insert(input);
    }
    heap.Output();

    heap.extractMin();
    heap.Output();
    heap.extractMin();
    heap.Output();
    heap.extractMin();
```

```
        heap.ExtractMin();
        heap.Output();

        return 0;
    }
```

결과:

```
1 7 10 9 5 6
1 5 6 9 7 10
5 7 6 9 10
6 7 10 9
7 9 10
계속하려면 아무 키나 누르십시오 . . .
```

위 결과에서 두번째 줄을 보시면 1 5 6 9 7 10인데, 이는 깊이 0에 있는 노드가 1, 깊이 1에 있는 노드가 5와 6, 깊이 2에 있는 노드가 9와 7 그리고 10가 있다는 것을 의미하는 것입니다. 그 아래는 최소값이 제거되면서 최소 힙이 어떻게 바뀌는지 보여주는 것 입니다. 대충 힙에대한 개념이 잡히시나요? 이런 최소 힙 또는 최대 힙을 통해서 정렬을 하는 힙 정렬(Heap Sort)도 존재한다는 것 아시나요? 나중에 시간이 되시면 힙 정렬에 대해서도 한번 찾아보세요! 오늘은 힙에 대한 설명은 여기까지 마무리 짓고 다음 강좌부터 우선순위 큐에 대해 간단히 알아보도록 하겠습니다. 수고하셨습니다.

'수학 > 알고리즘/자료구조' 카테고리의 다른 글

- 알고리즘 4-1강. 깊이 우선 탐색(Depth First Search) (49)
- 알고리즘 3강. 탐욕 알고리즘(Greedy Algorithm) (4)
- 자료구조 5강. 힙(Heap) (4)
- 알고리즘 2-3강. 탐색 알고리즘 - 이진 탐색 트리(Binary Search Tree) (9)
- 알고리즘 2-2강. 탐색 알고리즘 - 이진 탐색(Binary Search) (7)
- 알고리즘 2-1강. 탐색 알고리즘 - 순차 탐색(Sequential Search) (2)

COMMENT 4 / 트랙백 0

코딩꿈나무 at 2016.05.12 22:27 신고 [edit/del] Reply

여기가 설명 제일 잘되있는거같네요.. 진짜 감사합니다!

지나가다 at 2016.11.07 02:52 신고 [edit/del] Reply

Heap에서 추가랑 삭제가 잘 이해가 안됐었는데 깔끔한 설명 덕분에 완벽하게 이해했습니다! 좋은 글 감사합니다!
자료구조 다음 글도 기대하겠습니다!

wingu at 2016.12.11 22:11 신고 [edit/del] Reply

좋은글 감사합니다.

ㅂㅈㄷㄱ at 2017.02.15 18:05 신고 [edit/del] Reply

항상 감사드립니다. 덕분에 도움 많이 받습니다!!

Name * Password * Link

http://

Comments

Secret

Submit

Prev 1 ⋯ 54 55 56 57 58 59 60 61 62 ⋯ 290 Next