자구알 과제

141019 김연우

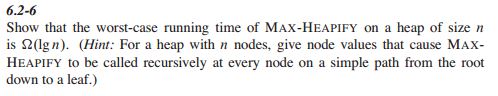


힙은 컴플리트 바이너리 트리이기 때문에 높이가 h인 힙의 노드 개수(n)는 최대 높이가 h인 풀 바이너리 트리의 개수고, 최소 높이가 h-1인 풀 바이너리 트리의 노드 개수 + 1개이다. 따라서 를 만족한다.



6.1-1의 증명에 의해서 이 성립한다. 이때 h로 정리하면 이므로 이다. 따라서 h = 이다.





Max Heapify는 왼쪽 자식과 오른쪽 자식 그리고 자기자신 중에서 가장 큰 값을 구하는 과정과, 가장 큰 값의 위치에서부터 다시 Max Heapify를 진행하는 작업으로 나눌 수 있다. 먼저 가장 큰 값을 구하는데 c(상수)만큼의 연산을 한다. 그리고 가장 큰 값의 위치에서부터 재귀적으로 자신을 호출하는데, 이때 가장 큰 값을 루트로 하는 서브 트리의 노드 개수가 다음 재귀함수의 연산시간에 결정적인 요소가 된다. 최악의 경우 왼쪽 서브 트리가 풀 바이너리 트리이고 오른쪽 서브 트리가 그보다 높이가 하나 더 낮은 풀 바이너리 트리이고, 가장 큰 값의 위치가 왼쪽 서브 트리인 경우이다. 이때 왼쪽 서브 트리의 개수는 전체 노드 개수 n에 대해서 2/3n개이다.

오른쪽 서브트리 개수n(r) + 왼쪽 서브트리 개수 n(l) + 1 = n

n(l) = n(r) + 왼쪽 서브트리의 루트 개수

왼쪽 서브트리는 풀바이너리 트리이므로 루트개수는 n(l)/2 + 1 개

n(r) = n(l)/2 – 1

2/3 \* n = n(l)

따라서 최악의 경우 전체 연산의 점화식을 세워보면 T(n) = T(2/3n) + c 가 성립한다. 이때 마스터 메소드를 적용해서 T(n)을 구해보면 f(n) = c = c \* 이므로 case 2에 해당하며, 따라서 T(n) = 이 성립한다.



높이 h 에 대한 induction

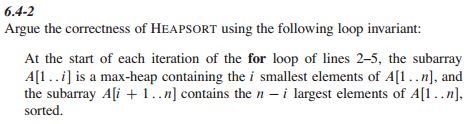
P(h) : 노드 개수가 n인 힙에서 높이가 h인 노드의 개수는 최대 개 이다.

Basis: 컴플리트 트리인 힙에서 h = 0인 노드들 즉 리프노드의 개수는 인터널 노드의 개수보다 1개 더 많거나 같다. 따라서 전체 노드의 개수가 n 일때 리프노드의 개수는 최대 개 이므로 P(0)이 성립된다.

Induction: P(k)가 성립한다고 가정하면, P(k+1)이 참임을 보인다.

높이가 k + 1 인 노드들은 높이가 k인 노드들의 부모 노드이다. 컴플리트 트리인 힙에서 부모 노드의 개수는 자식 노드의 개수/2 이거나 자식 노드의 개수 / 2 + 1 이다. 따라서 높이가 k+1 인 노드들의 개수는 최대 개이다. 따라서 P(k+1)이 성립한다.

수학적 귀납법에 의해서 h 인 정수 h에 대해 P(h)가 성립함을 보였다.



Initialization: I = Length 일 때 ,

1. 반복문의 시작점에서 A[1…length]는 원래 주어진 맥스 힙이므로 length개의 가장 작은 원소들로 구성된 맥스 힙이다.
2. 그리고 A[length + 1…length]는 원소가 하나도 없는 배열이므로 length – length 즉 0개의 가장 큰 원소가 정렬된 배열이다.

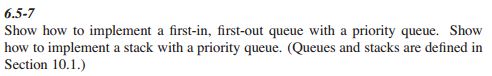
Maintenance: I = k일 때 Loop Invariant를 만족한다고 가정하자.

1. 가정에 의해서 A[1…k]은 전체 원소 중에서 k개의 가장 작은 원소로 구성된 맥스 힙이다.
2. A[k+1….n]은 n – k 개의 가장 큰 원소로 구성된 정렬된 배열이다.
3. 따라서 A[1]은 A[1…k]의 원소 중에서 가장 큰 숫자이면서 A[k+1….n]까지의 배열의 모든 원소보다 작은 숫자이다.
4. A[1]과 A[k]를 스왑하면 A[k….n]은 전체 배열에서 가장 큰 n – (k – 1) 개의 원소들의 정렬된 배열이다.
5. 그리고 A[1….k-1]까지의 배열은 A[1]을 제외하고 변하지 않았으므로 그대로 맥스 힙이다.
6. 따라서 힙의 사이즈를 하나 줄이면, 루트노드를 제외한 왼쪽 서브트리와 오른쪽 서브트리가 맥스 힙이므로 그상태에서 Max Heapify를 적용하는 것으로 A[1…k-1]을 맥스 힙으로 만들 수 있다.
7. 그리고 원래 A[1…k]에서 가장 큰 원소만 빠졌으므로 A[1…k-1]은 가장 작은 k-1 개의 원소를 갖는 맥스 힙이다.
8. 반복문이 하나의 루프를 끝마치고 I = k+1 이되면 루프의 시작지점에서 Loop Invariant가 성립한다.

Termination :

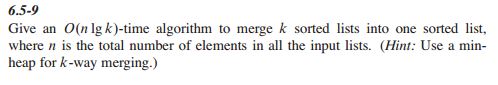
1. I = 1일 때, 수학적 귀납법에 의해서 Loop Invariant가 성립한다.
2. A[1]은 전체에서 가장 작은 1개의 원소의 맥스힙이다.
3. A[2….n]은 가장 큰 n - 1개의 정렬된 배열이다.
4. 따라서 전체 배열이 정렬되었음을 증명할 수 있다.





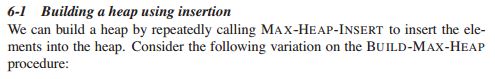
우선순위 큐에 삽입할 때 먼저 들어온 노드에 높은 우선 순위 값을 부여하고 나중 들어온 노드에더 낮은 우선순위 값을 부여해서 삽입한다. 그러면 반드시 먼저 들어온 노드가 먼저 나갈 수 밖에 없다. 이렇게 큐를 구성할 수 있다.

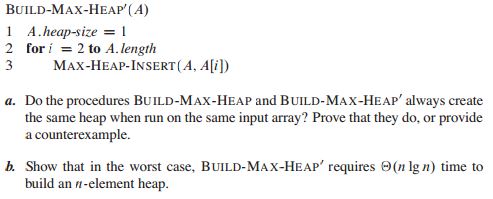
우선순위 큐에 삽입할 때 삽입할 때의 큐 사이즈 만큼의 우선순위를 부여하는 것으로 스택을 구현할 수 있다. 그러면 마지막으로 들어온 노드가 반드시 가장 높은 우선순위를 갖게 되기때문에 LIFO를 구현할 수 있다.



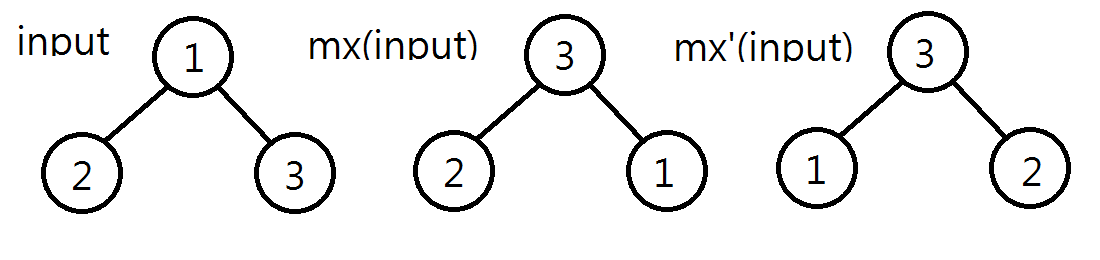
K개의 배열에서 가장 작은 숫자들을 minHeap에 지속적으로 유지시키면, minHeap의 루트에 있는 값은 전채 배열들에 있는 원소 중에 가장 작은 값이 된다. minHeap의 루트노드를 추출해서 순서대로 결과 배열에 집어 넣으면 소팅을 할 수 있다.

1. K개의 배열에서 맨 앞에 있는 원소들을 빼서 K개짜리 minHeap을 구성한다.
2. minHeap의 루트노드를 추출해서 결과배열에 집어 넣는다.
3. minHeap에서 추출된 데이터가 포함된 배열에서 한 개를 더 빼내서 전체 배열에서 가장작은 K개의 원소로 구성된 minHeap을 유지한다.
4. 만약 그 배열에 원소가 없으면 minHeap에서 한번 더 루트노드를 추출한뒤 결과배열에 넣는다.
5. minHeap에 아무 원소가 없을 때까지 1-3을 반복한다.



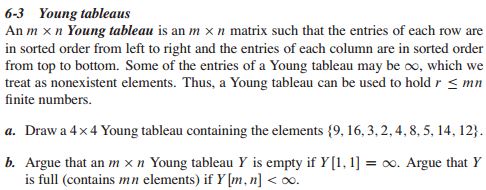


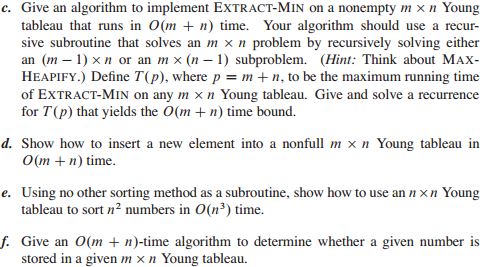
a.



b.

Max-Heap-Insert는 Heap-Increase-Key를 호출하는데 최악의 경우 모든 부모노드의 계열을 거슬러 올라가야하므로 log n의 연산시간이 걸린다.루프문의 모든 노드가 모두 최악의 경우이면, 즉 입력 배열이 오름차순으로 주어졌다면, 총 n log n의 연산시간이 걸리게 된다.

0



|  |  |  |  |
| --- | --- | --- | --- |
| 2 | 4 | 8 | 12 |
| 3 | 5 | 9 | 14 |
| 16 |  |  |  |
|  |  |  |  |

* 1. 증명
     + 1. Y[m, n](m, n 은 0보다 큰 임의의 정수)의 값이 c이면 모든 row가 왼쪽에서 오른쪽으로 오름차순 정렬되어 있으므로 x축 위치가 m보다 큰 모든 원소는 c보다 크다. 그리고 x축 위치가 m보다 작은 모든 원소는 c보다 작다.
       2. 그리고 모든 column이 위에서 아래로 오름차순 정렬되어 있으므로, y축 위치가 n보다 큰 모든 원소는 c보다 크다. 그리고 y축 위치가 n보다 작은 모든 원소는 c보다 작다.
       3. Y[1, 1]은 전체 Young Tableau에서 x축 값과 y축 값 모두 가장 작은 원소이므로 1과 2에 의해 가장 작은 원소임을 보장받는다. Y[1,1]이 무한대라면 다른 모든 원소는 Y[1,1]보다 크기 때문에 그들 역시 무한대일 수 밖에 없다. 값이 무한대인 원소는 비어있는 것으로 한다. 따라서 Y[1,1]이 무한대이면 전체 Young Tableau 는 비어있다.
       4. Y[m, n]이 무한대 보다 작다면, 1과 2에 의해서 x값이 m보다 작거나 y값이 n보다 작은 모든 원소들은 Y[m, n]보다 작다. 따라서 m\*n개의 원소들이 무한대보다 작은 값을 갖고 있으며, 이 Young Tableau는 꽉 차 있다.
  2. EXTRACT\_MIN( ELEM[Y yt, INT xIdx, INT yIdx ]) {
     + 1. EXTRACT Y[xIdx, yIdx];
       2. IF(Y[xIdx + 1, yIdx] == INFINITE && Y[xIdx , yIdx + 1] == INFINITE)
          1. QUIT;
       3. ELEM nextMin = Y[xIdx + 1, yIdx] < Y[xIdx , yIdx + 1] ?
          1. [yt, xIdx + 1, yIdx] : [yt, xIdx , yIdx + 1];
       4. EXTRACT\_MIN(nextMin);

}

* + 1. EXTRACT\_MIN( [yt, 1, 1] );
  1. INSERT\_YT(INT value, ELEM[Y yt, INT xIdx, INT yIdx]) {
     + 1. IF(xIdx == 1 && yIdx == 1)
          1. Y[1, 1] = value;
       2. IF(value < Y[xIdx – 1, yIdx])
          1. Y[xIdx, yIdx] = Y[xIdx – 1, yIdx];
          2. INSERT\_YT(value, [yt, xIdx – 1, yIdx]);
       3. ELSE IF(value < Y[xIdx, yIdx -1])
          1. Y[xIdx, yIdx] = Y[xIdx, yIdx – 1];
          2. INSERT\_YT(value, xIdx, yIdx -1);
       4. ELSE
          1. Y[xIdx, yIdx] = value;

}

* + 1. INSERT\_YT(inputValue, [yt, MAX\_X\_IDX, MAX\_Y\_IDX]);
       1. n^2짜리 배열을 Young Tableau에 INSERT\_YT하면 한번에 2n씩 n^2번 총 2n^3
       2. 이렇게 만들어진 Young Tableau 가 모두 비었을 때까지 EXTRACT\_MIN을 수행해서 새로운 결과 배열의 뒤에 삽입하면 2n\*n^2 총 2n^3
       3. 모두 합쳐서 c\*n^3번 연산해서 정렬할 수 있기 때문에 O(n^3)시간 만에 성공
  1. BOOL FIND\_KEY(INT key, INT xIdx, INT yIdx){
     + 1. IF(xIdx == 0 || yIdx == MAX\_Y\_IDX + 1)
          1. return FALSE;
       2. IF(Y[xIdx, yIdx] == key)
          1. return TRUE;
       3. ELSE IF(Y[xIdx, yIdx] > key)
          1. return FIND\_KEY(key, xIdx, yIdx + 1);
       4. ELSE
          1. Return FIND\_KET(key, xIdx – 1 , yIdx);
     1. }
     2. FIND\_KEY(key, MAX\_X\_IDX, 1);