11.2 디스크를 이용한 정렬

11.2.1 디스크 정렬의 개요

디스크를 이용한 정렬에서 테이프의 경우와 같이 가장 널리 알려진 방법은 병합 정렬이다. 즉 입력 파일을 읽어 적당한 내부 정렬 방법을 적용시켜 정렬하여 몇 개의 서브 파일(run이라고 함)을 생성하여 디스크에 수록한 후, 이것들을 최종적으로 하나의 파일이 될 때까지 반복적으로 병합하는 것이다.

테이프 정렬에서는 k-way 병합을 하고자 할 때 k는 사용 가능한 테이프 장치에 영향을 받지만 디스크에서는 영향을 받지 않는다. 또 디스크 장치는 테이프와는 달리 임의 접근(random access)이 가능하므로 보다 효율적인 방법들이 제안되어 있다.

디스크 정렬의 효율성은 전체 시간과 밀접한 관련을 갖는데, 디스크의 처리 시간은 다음과 같은 3가지 요소가 영향을 미친다.

① 탐색 시간(seek time) : 읽기/쓰기 헤드가 해당 실린더(cylinder)를 찾는 데 걸리는 시간으로, 이것은 헤드가 움직여야 하는 실린더의 거리와 관계가 있다.
② 회전 지연 시간(rotation delay time) : 읽기/쓰기 헤드가 트랙 위의 해당 섹터(sector)에 올 때까지 걸리는 시간이다.
③ 전송 시간(transmission time) : 디스크와 주기억 장치간에 데이터가 전송되는 데 걸리는 시간이다.

검색 시간을 줄이기 위해서는 자료가 가능한 한 같은 실린더에 수록되어야 하는데, 파일이 클 경우에는 불가능하므로 이를 위하여 별도로 키(key)만 모아 키 정렬(key sort)을 한다. 또 전송 시간을 줄이기 위해서는 입출력 횟수를 줄여야 하는데, 이것은 내부 정렬을 통하여 가능한 한 서브 파일을 크게 하여 그 개수를 줄이거나 또는 k-way 병합에서 k의 값을 크게 하거나 또는 병합 순서를 적절히 조정하여 병합 횟수를 최소화하여야 한다.

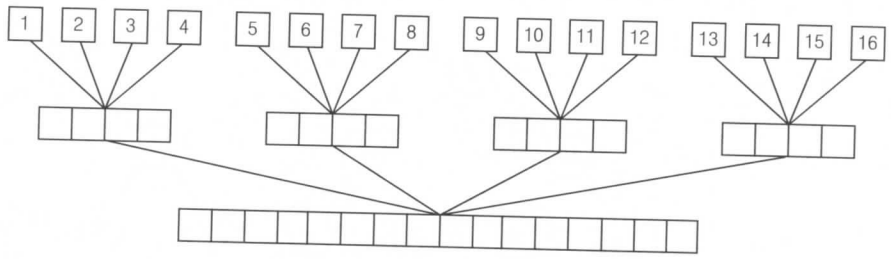
따라서 이 절에서는 k값이 큰 k-way 병합 방법, 서브 파일의 크기를 최대화하기 위한 방법 및 최적 병합 트리와 키 정렬 등에 대하여 살펴본다.

11.2.2 k-way 병합

병합 정렬에서 m개의 서브 파일이 있다면 2-way 병합에서의 병합 트리는 데이터 파일에 대하여 총 $\lceil \log_2 m \rceil$회의 처리를 해야 하므로 $\lceil \log_2 m \rceil + 1$의 레벨을 가지게 된다.

그러므로 자료의 처리 횟수는 $k \geq 2$인 k-way 병합을 통하여 차수를 높임으로써 줄일 수 있다.

예를 들어, 16개의 서브 파일이 있을 경우 2-way 병합을 하면 처리 횟수가 4회이지만 4-way 병합을 하면 〈그림 11.5〉와 같이 2회의 처리 횟수로 끝난다.



〈그림 11.5〉 16개의 서브 파일에 대한 4-way 병합

일반적으로 m개의 서브 파일에 대한 k-way 병합은 많아야 $\lceil \log_2 m \rceil$회의 자료 처리를 한다. 따라서 고차 병합을 사용하면 그 만큼 입출력 시간인 전송 시간을 줄인다.

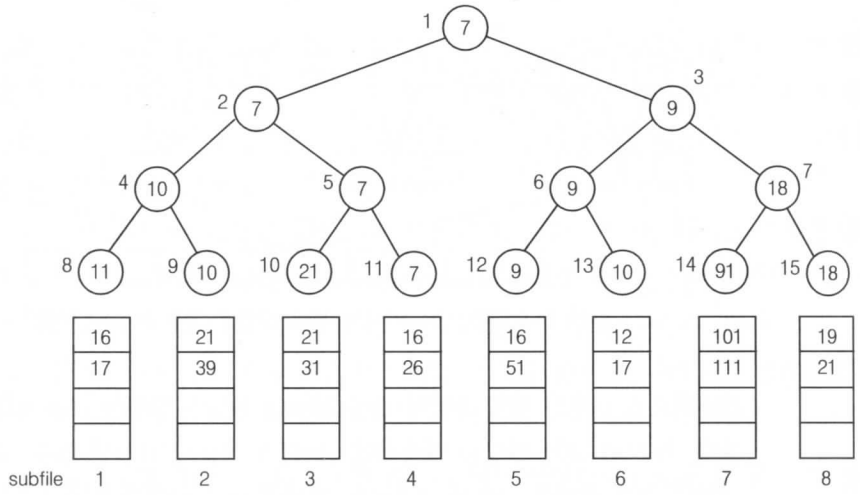
그러나 고차 병합을 사용할 경우에는 정렬에 다른 영향을 미친다. 즉, k개의 서브 파일을 내부적으로 병합할 때, k개의 자료를 비교하여 가장 작은 것을 선택하는 데 걸리는 시간이 길어진다. 그러므로 k가 증가할 때 입출력 시간은 감소하더라도 k-way 병합을 하는 데 필요한 시간 때문에 결과적으로 전체 시간은 증가한다. 그러나 k값이 클 경우, 다음 최소값의 데이터를 찾는 데 필요한 비교 횟수는 선택 트리 기법을 사용하여 어느 정도 줄일 수 있다.

선택 트리(selection tree)는 각 노드가 두 자노드보다 더 작은 값을 갖도록 구성된 이진 트리이다. 따라서 근노드는 그 트리에서 가장 작은 값이 된다. 〈그림 11.6〉은 8개 서브 파일의 8-way 병합에 대한 선택 트리를 나타내고 있다.

이 선택 트리의 구조는 더 작은 값을 가진 데이터가 승자(winner)가 되는 토너먼트 경기에 비유될 수 있으므로 이 트리에서 단노드(terminal node)가 아닌 것은 토너먼트의 승자를 뜻하고, 근노드는 전체 승자, 즉 가장 작은 값을 의미하게 된다. 단노드는 해당 서브 파일에서 첫 번째 데이터를 나타낸다. 여기에서 데이터를 해당 레코드의 키 또는 그 레코드의 포인터를 갖도록 하면 전체 레코드를 다루지 않아도 된다. 선택 트리는 이진 트리로 구성되고 각 노드의 옆에 쓰여진 번호는 순차적으로 표현해서 그 노드의 주소로 활용함으로써 알고리즘을 단순화시킬 수 있다.

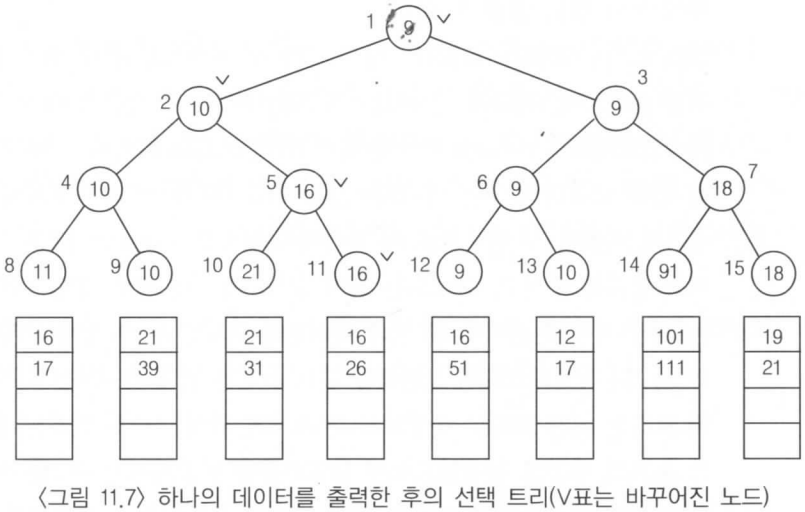
선택 트리를 이용하여 어떻게 정렬 작업이 행해지는지 살펴보자. 먼저 근노드에 가장 작은 값이 있으므로 이것을 출력하고, 이것은 서브 파일 4에서 올라온 것이므로 서브 파일 4에서 11번 노드에 넣는다. 여기에서 제노드인 10번 노드와 16을 비교하

면 21>16이 되므로 16이 5번 노드로 올라간다. 다시 4번 노드와 비교하면 10<16이므로 10이 2번 노드로 올라가며, 여기에서 3번 노드와 비교하면 10>9이므로 9가 근노드로 올라간다. 이것을 나타내면 <그림 11.7>과 같다.



<그림 11.6> 8-way 병합에 대한 선택 트리

<그림 11.7>의 선택 트리에서 다시 근노드의 9를 출력하고, 서브 파일 5에서 16을 노드 12로 입력하여 앞에서와 같은 방법으로 계속 반복하여 정렬을 해나간다.

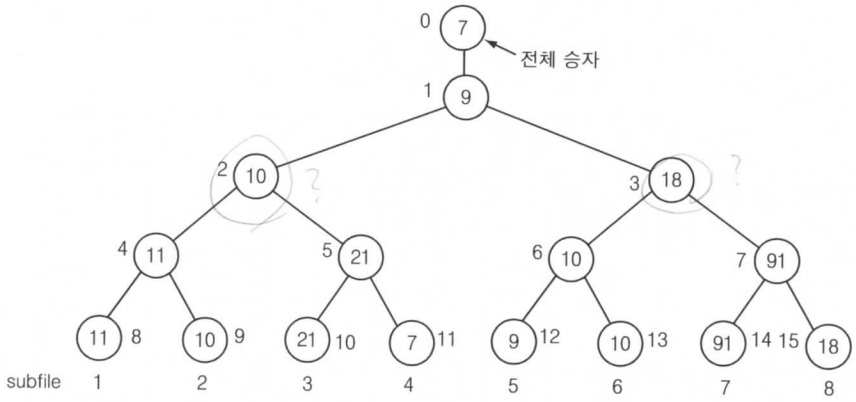


<그림 11.7> 하나의 데이터를 출력한 후의 선택 트리(V표는 바꾸어진 노드)

이 트리에서 레벨의 수는 $\lceil \log_2 k \rceil + 1$이므로 트리를 재구성하는 데 걸리는 시간이 $O(\log_2 k)$가 된다. 트리는 1개의 데이터가 출력될 때마다 한 번씩 재구성되므로 n개의

데이터를 병합하는 데 걸리는 시간은 $O(n\log_2 k)$이다. 처음에 선택 트리를 만드는 데 $O(k)$시간이 소요되고, 레벨 수는 $O(\log_k m)$이므로 내부 처리 시간은 k와는 관계없이 $O(n\log_2 k \cdot \log_k m)=O(n\log_2 m)$이 된다.

총 소요 시간을 다소 줄이려면 비 단노드들이 패자가 되는 패자 트리(tree of loser)를 사용하면 되는데, 〈그림 11.7〉의 선택 트리를 패자 트리로 나타내면 〈그림 11.8〉과 같다.



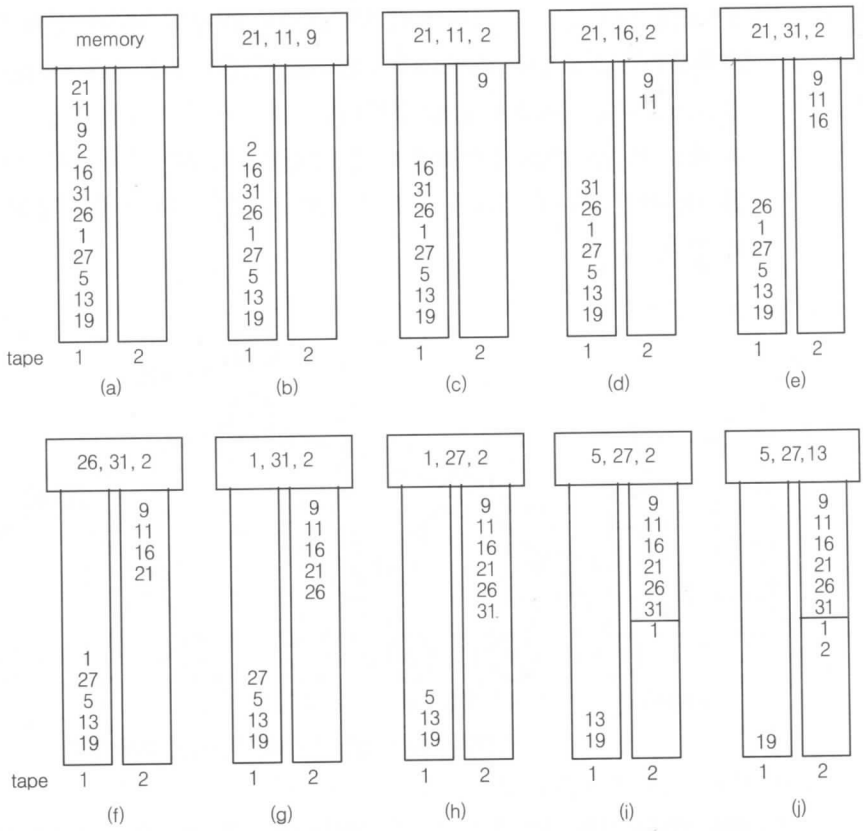
〈그림 11.8〉 〈그림 11.7〉에 대응하는 패자 트리

패자 트리에서는 추가로 노드 0을 설정하여 최종 승자를 나타내며 이것을 출력한 후 트리를 재구성한다.

11.2.3 서브 파일의 생성

내부 정렬을 통하여 생성되는 서브 파일의 크기는 사용하는 컴퓨터의 기억 용량과 관계가 있다. 예를 들어, 한 번에 기억시킬 수 있는 최대 레코드의 수가 b라면 생성되는 서브 파일의 크기는 b보다 클 수 없다. 그런데 대치 선택(replacement-selection) 알고리즘을 사용하면 평균적으로 약 2배 크기의 서브 파일을 생성할 수 있어서 전체적으로 서브 파일의 개수가 적어지므로 병합에 있어서 단계의 수를 줄여 정렬 작업의 총 시간을 감소시킬 수 있다.

우선 대치 선택에 의하여 어떻게 서브 파일이 생성되는지 〈그림 11.9〉를 통하여 살펴보자.



〈그림 11.9〉 2개의 서브 리스트가 생성되는 대치 선택

리스트는 〈그림 11.9〉 (a)와 같이 tape 1에 수록되어 있고, 주기억 장치의 기억 용량의 크기는 3이라고 가정한다.

먼저 tape 1에서 3개의 자료를 읽어 기억 장치에 기억시키면 〈그림 11.9〉의 (b)와 같은 상태가 된다. 이 때 기억 장치에 기억된 내부 리스트에서 가장 작은 값을 선택하여 tape 2에 출력하고, 이 위치에 tape 1로부터 다음 값을 읽어 대치하면 〈그림 11.9〉의 (c)와 같이 된다. 다시 기억 장치에서 출력된 값보다 크거나 같은 값 중 가장 작은 11을 선택하여 출력하고, tape 1에서 다음 값 16을 읽어 그 장소에 대치하면 〈그림 11.9〉의 (d)와 같이 된다.

이와 같은 방법을 계속하여 〈그림 11.9〉의 (h)와 같은 상태가 되면 이제는 출력된 31보다 작지 않은 값이 없으므로 일단 1개의 서브 파일의 생성을 끝내고, 기억 장치 내에서 가장 작은 1을 출력한 후 그 자리에 tape 1로부터 5를 읽어 대치하면 〈그림 11.9〉의 (i)상태가 된다. 이런 방법을 계속하여 두 번째의 서브 파일을 생성한다.

〈그림 11.9〉에서 보는 바와 같이 일반적인 서브 파일의 생성 방법으로는 12/3=4개의 서브 파일이 만들어지나 대치 선택 알고리즘을 사용하면 2개의 서브 파일로 끝나게

되므로 다음 단계의 병합 작업을 효율적으로 수행할 수 있다.
대치 선택 알고리즘을 개괄적으로 기술하면 다음과 같다.

```
replacement- selection

{T1은 입력 테이프이고, T2는 출력 테이프이다. 여기에서 n은 내부 기억 장치의 크기이
다.}

1. T1으로부터 n개의 자료를 읽어 LIST라 불리는 내부 기억 장치에 기억시킨다.
2. T1이 empty가 될 때까지 다음 step을 반복한다.
   (a) LIST로부터 smallest data를 선택하여 MINMAX로 한다.
   (b) T2에 MINMAX를 출력하고, T1으로부터 1개의 data를 읽어 그 곳에 replace
       한다. 그리고 SELECTION='SUCCESS'로 놓는다.
   (c) SELECTION='FAIL'일 때까지 다음 step을 반복한다.
       i) LIST 내에 MINMAX보다 크거나 같은 값 중 가장 작은 것이 있으면, 이것을
          MINMAX에 replace하고 T2에 출력하며, T1으로부터 1개의 data를 읽어
          그 자리에 replace한다.
       ii) 아니면 SELECTION='FAIL'로 set한다.
   (d) T2에 subfile mark를 출력한다:
3. LIST 내의 (n-1)개의 largest 자료를 순서화하여 T2에 출력한 후, T2에
   filemark를 한다.
```

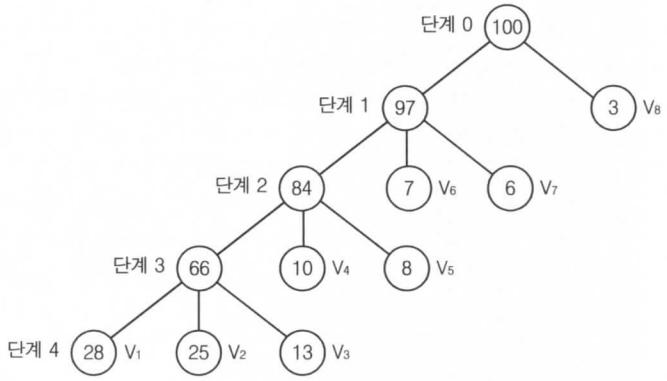
대치 선택 알고리즘은 각기 다른 크기의 순서화된 서브 파일을 만들어 내기 때문에 이것은 최소 병합 트리(minimal merge tree)를 형성하여 최적으로 병합 작업을 수행할 수 있으므로 전체 정렬 시간을 단축하게 된다.

11.2.4 최적 병합 트리

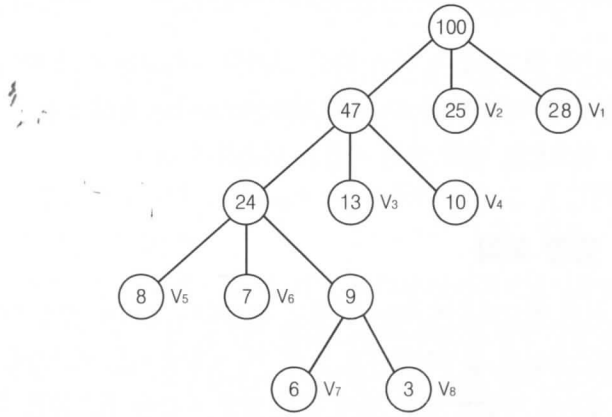
대치 선택 알고리즘에 의하여 서브 리스트의 길이가 각각 28, 25, 13, 10, 8, 7, 6, 3인 리스트들이 생성되었다고 하자. 이 8개의 서브 리스트를 3-way 병합을 한다면 어떤 것부터 병합을 하는 것이 전체 병합 시간을 최소화할 수 있는지에 대한 문제가 제기된다.

제일 먼저 서브 리스트의 길이가 가장 큰 3개를 병합한다고 하면 새로 만들어지는 서브 리스트의 길이는 28+25+13=66이 될 것이다. 그렇게 되면 다음 단계에서 병합해야 할 서브 리스트는 66, 10, 8, 7, 6, 3의 길이를 가진 6개가 된다. 여기서 다시 길이가 큰 3개의 서브 리스트를 병합하면 66+10+8=84의 길이가 되는 리스트가 만들어지고, 그 다음에는 84+7+6=97, 마지막으로 97+3=100의 길이를 가진 리스트가 만들어짐으로써 병합이 끝나게 된다.

이런 방법으로 3-way 병합을 할 경우 각 단계를 트리로 표현할 수 있는데, 이 트리는 초기의 서브 리스트를 단노드로서 표현한다. 단노드의 값은 서브 리스트의 길이를 나타내며, 간노드(internal node)는 병합된 리스트의 길이로서 그 자노드의 합계가 된다. 〈그림 11.10〉의 (a)는 앞에서 설명한 방법으로 병합하는 과정을 나타낸 병합 트리(merge tree)이다. 근노드는 계산의 편의상 레벨 0으로 하였다.



(a) 3-way 병합을 위한 병합 트리



(b) 3-way 병합을 위한 최적 병합 트리

〈그림 11.10〉 3-way 병합을 위한 최적 병합 트리

〈그림 11.10〉의 (b)와 같은 트리에 의하여 병합을 한다고 하자. 첫 단계의 병합은 가장 길이가 짧은 2개의 서브 리스트 V_7과 V_8을 대상으로 하여 길이가 9인 서브 리스트를 만들고, 다음 단계에서는 이것과 길이가 각각 8과 7인 서브 리스트 V_5, V_6과 함께 3-way 병합을 하여 길이가 24인 서브 리스트를 만든다. 그 다음에는 길이 24로 새로 만들어진 서브 리스트와 길이가 각각 13, 10인 V_3, V_4를 병합하여 길이가 47인 서브

리스트를 만들고 이것을 최종적으로 길이가 각각 25, 28인 서브 리스트 V_2, V_1을 병합하여 길이가 100인 순서화된 리스트를 얻음으로써 병합은 끝난다.

〈그림 11.10〉의 (b)의 병합 트리를 최적 병합 트리(optimal merge tree)또는 최소 병합 트리(minimal merge tree)라고 하는데, 그 이유는 병합 과정에서 필요로 하는 데이터의 이동 횟수(move operation)가 최소가 되도록 구성된 트리이기 때문이다.

그러면 주어진 트리의 총 병합 횟수는 어떻게 계산하고, 최적 병합 트리는 어떻게 만드는가에 대하여 알아보자.

병합에 따르는 데이터의 이동 횟수를 계산함에 있어서 병합 트리의 각 레벨은 한 번의 이동을 필요로 한다. 초기의 서브 리스트에 포함된 데이터가 최종 리스트까지 이동되기 위해서는 근노드에 포함될 때까지 반복해서 단노드를 병합해 나아가야 한다. 그러므로 초기 서브 리스트에서 하나의 데이터는 그 레벨부터 근노드까지 레벨 수만큼 이동되어야 한다.

즉 레벨 L에 있는 단노드는 근노드에 도달하기 위하여 L번 이동되어야 하므로 만일 단노드가 v_i의 값을 가지고 있다면 Lv_i는 초기 서브 리스트에서 최종 리스트까지 v_i개의 자료를 복사(copy)하는 데 필요한 이동 횟수가 된다.

여기에서 L_i를 단노드 i의 레벨 수라하고, v_i를 단노드 i의 서브 리스트의 길이라고 하면, r 레벨을 가진 트리의 총 이동 횟수 value는

$$\text{value} = \sum_{i=1}^{r} L_i v_i$$

가 된다.

최적 병합 트리는 병합 트리의 value가 가능한 한 최소가 되도록 단노드(길이가 v_i로 표현된 서브 리스트)를 배치한 트리이다. 〈그림 11.10〉에 보인 2개의 트리에 대한 총 이동 횟수의 계산 결과를 나타낸 것이 〈표 11.4〉이다.

〈표 11.4〉 2개의 병합 트리에 대한 총 이동 횟수 계산

level number	node	value	product
1	v_8	3	3
2	v_6, v_7	7 + 6	26
3	v_4, v_5	10 + 8	54
4	v_1, v_2, v_3	28 + 25 + 13	264
		value =	347

(a) 〈그림 11.10〉 (a)의 트리

level number	node	value	product
1	v_1, v_2	28 + 25	53
2	v_3, v_4	13 + 10	46
3	v_5, v_6	8 + 7	45
4	v_7, v_8	6 + 3	36
		value =	180

(b) 〈그림 11.10〉 (b)의 트리

〈표 11.4〉의 value는 각 레벨에 있는 노드의 값을 더한 것이고, product는 value에 level number를 곱한 것이다.

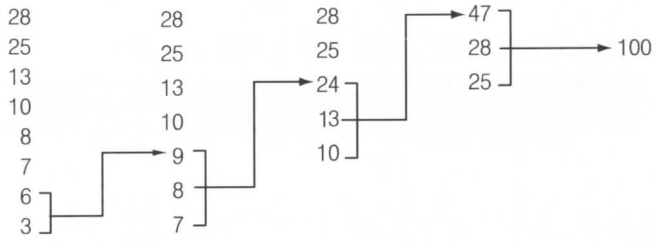
병합 트리의 value를 계산하는 공식은 주어진 트리가 최적인가를 판정하기 위하여 트리의 value를 검사하는 수단을 제공한다.

이번에는 어떻게 최적 병합 트리를 만드는지 그 방법에 대하여 살펴보자. 2-way 병합은 이진 트리를 형성하고, 3-way 병합은 3진 트리(ternary tree)를 형성한다.

일반적으로 C진 트리(C-nary tree)는 C-way 병합을 나타낸다. 최적 병합 C진 트리는 최소 값을 가진 C개의 노드를 함께 집단화함에 의하여 만들어지는데, 이것을 filial set라고 한다.

C개의 최소값들은 더해지고, 그 합계는 filial set의 부노드의 값이 된다. 다시 남아 있는 모든 노드들 중에서 최소값을 가진 C개의 노드를 선택하면 이것이 filial set이 된다. filial set의 값을 모두 더하여 이것을 다시 부노드로 해서 앞의 방법을 반복 적용하는 과정에서 하나의 노드만 남게 될 때 트리의 구성은 끝난다.

때로는 단노드의 수가 C의 배수가 아닐 경우가 있는데, 이런 경우에는 처음 filial set의 크기를 다음과 같이 결정하여야 한다.



〈그림 11.11〉 최적 병합 트리의 유도

만일 $(r-1)$ modulo $(C-1)=0$ 이면 최초 filial set은 C개의 노드를 포함하게 되고, 그렇지 않으면 $1+\lfloor(r-1) \text{ modulo } (C-1)\rfloor$ 개의 노드를 갖도록 한다.
〈그림 11.10〉의 (b)의 최적 병합 트리의 구성의 예를 나타낸 것이 〈그림 11.11〉이다. 여기에서 $r=4$이고 $C=3$이므로 최초 filial set은 $1+\lfloor(4-1) \text{ modulo } (3-1)\rfloor=2$이다.

11.2.5 키 정렬

디스크의 정렬에서 검색 시간(seek time)을 무시한다면 최적 병합 트리를 이용하는 것이 가장 적합한 병합 정렬 방법이 된다. 그러나 검색 시간은 디스크의 파일을 다루는 데 있어서 가장 큰 요인 중의 하나이기 때문에 이 시간을 최소화하는 노력이 필요하다.
잘 아는 바와 같이 디스크의 트랙은 트랙간의 이동 시간을 피하기 위하여 실린더로 구성되어 있다. 그러므로 다루려고 하는 레코드들이 1개의 실린더에 모두 포함된 트랙에 디스크 파일을 수록하면 헤드의 이동없이 레코드들을 다룰 수 있다.
만일 초기의 서브 파일이 한 실린더에 모두 수용할 수 있는 작은 것들이라면 정렬과 병합 알고리즘은 검색 시간을 최소화하여 수행할 수 있다.
그러나 대부분의 파일은 하나의 실린더에 수용할 수 없을 정도의 크기이기 때문에 각 레코드를 키 필드(key field)와 나머지 필드로 분할하여 정렬 대상이 되는 파일의 크기를 감소시키는 방법을 사용한다. 즉 키 필드와 그에 대응하는 레코드의 포인터로서 파일을 만들어 정렬한다.
이렇게 파일의 크기를 줄이기 위하여 키와 그 포인터만을 가지고 정렬하는 방법을 키 정렬(key sort) 또는 태그 정렬(tag sort)이라고 한다.
키 정렬의 단점은 정렬 후에 파일의 마스터 레코드를 키가 정렬된 후에 해당 위치로 재배치한다는 점이다.