

자료구조 HW4 Supplementary Report

Mechanical Engineering

2019-15838 주기영

4. Search method 확장

4.1 데이터 특성에 따른 Q, M 정렬의 정렬시간

Quick sort는 항상 최적의 알고리즘이 아니다. Quick sort는 중복된 원소가 많을수록, 정렬 완성도가 높을수록 정렬시간이 느리다. 실제로 10^7 을 원소로 갖는 배열을 이용하여 배열의 sortedPair과 redundancy 값에 따른 Q의 정렬 시간을 plot해본 것이 Fig 1, Fig 2이다. M은 비교를 위해서 같이 plot하였다. $1 \sim 10^7$ 을 원소로 갖는 배열에서 배열의 아무 값을 임의로 섞는 shuffle, 난수의 상한 조절을 이용하여 적절한 배열을 얻어 실험을 진행하였다. sortedPair 값이 99.94% 이상, redundancy 값이 0.005% 이하에서는 Q에서 재귀에 의한 Stack overflow가 발생하는 것을 관찰할 수 있었고, 이를 제외했다.

Q가 M에 비해서 fluctuate가 매우 심한 것이 관찰되는데 이는 Q 정렬이 pivot 값 설정에 많은 영향을 받기 때문이다. 또한, merge sort의 경우에는 sortedPair 값이 높을 때(즉, 배열이 거의 정렬되어 있을 때), 정렬시간이 더 빠른 것을 관찰할 수 있는데 이는 거의 정렬된 배열에서 M 정렬을 수행하면 두 배열을 합치는 과정에서 비교를 하는 과정이 현저히 줄어들기 때문이다. 중복도가 높을 때에도 같은 이유로 M의 성능향상이 관찰된다. 두 그래프에서 배열의 정렬쌍 비율이 90퍼 이상일 때와 중복도가 현저히 높을 때(redundancy 값이 0.5% 이하일 때) Q의 정렬시간이 급격하게 상승하는 것을 확인할 수 있다. 이를 통해 Q는 데이터 특성의 영향을 많이 받음을 알 수 있고, Search method를 통해 데이터의 특성을 파악하여 적절한 정렬을 수행하는 것이 필요하다.

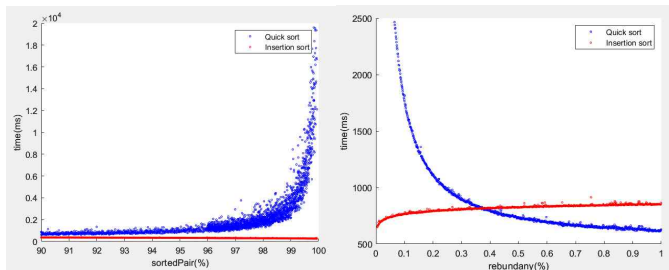


Fig 1. Time-Sortedpair curve Fig 2. Time-redundancy curve

4.2 Search method 확장의 필요성

과제 스펙에서의 Search method의 입력은 크기가 50000 이하의 입력까지만 고려하는 방식이었다. 물론 실제로 실생활에서 정렬이 필요할 때 작은 데이터 개수를 다루는 경우가 많다고도 할 수 있으나 50000개 이상의 데이터를 정렬해야 하는 경우도 물론 존재한다. 또한, 데이터가 작을 때(5만 개 이하) 항상 Quick sort를 적용하더라도 운이 안 좋아 거의 정렬된 배열이나 중복도가 매우 높은 배열을 정렬하더라도 항상 $O(n^2)$ 의 시간복잡도를 갖는다. 5만 개 이하에서도 물론 $\theta(n^2)$, $\theta(n \log n)$ 의 차이가 크다고 할 수 있으나 이는 배열의 크기가 클수록 차이가 더욱 더 극명해진다. 실제로 Fig 1, Fig 2를 보면 충분히 랜덤한 배열에 대해서 Q를 적용하면 1000ms 이하로 수행하지만 sortedPair가 99%를 넘어가거나 redundancy가 0.05% 이하일 때에는 30000ms의 정렬시간을 보여준다. 또한 Q의 재귀에 의한 Stack overflow도 배열 특성에 더욱 의존하게 된다. //50000개 이하에서의 퍼포먼스 설명을 생각하면 이는 매우 큰 차이이며 따라서 Search method를 확장해보려 한다. 확장할 때에는 5만 이하에서 구현했던 아이디어를 적용하면서 더 최적화하는 방법을 생각해보았다.

4.3 최대자리수

기존 Report에서 설명했듯이 R과 M의 최대자리수에 대한 HP는 배열의 크기에 의존한다. 이는 $\theta(kn)$, $\theta(n \log n)$ 의 경계에서 k가 $\log n$ 에 비례하는 함수로 나타나고, 이를 반영하면 HP가 n, k의 함수로 나타나는 것이다. 최대자리수 k에 대해서 k=1일 때에는 배열의 크기 5만부터 int 값의 최대값까지 모두 R이 Q/M보다 빠른 것을 관찰할 수 있다. 하지만 k=2일 때는 5만 이상에서는 항상 M/Q가 R보다 빠른 것을 관찰할 수 있었다. 하지만 배열 크기 5만 이상, k=1일 때조차 최대자리수를 구하는 동작시간과 R의 동작시간을 합했을 때 거의 M의 동작시간과 비슷하다. 따라서 5만 이상에서는 최대자리수의 HP를 고려하는 것이 더욱 비효율적이고, 이는 Search 알고리즘에서

제외하였다.

4.4 redundancy

Fig 2에서 redundancy 값이 0.5% 이하일 때 Q의 성능이 현저하게 떨어지는데 이는 같은 원소를 모두 pivot의 오른쪽으로 보내는 “partition” 메소드에서 결과적으로 pivot이 배열의 중간이 아닌 가장자리에 쏠리게 되면서 Q의 성능이 안좋아지는 것이다. 5만 개 이하에서의 Search에서는 M을 사용하는 것으로 해결하였다. 하지만 배열의 크기가 5만 이상일 때는 Q의 “partition” 메소드를 일부 수정하는 것이 더 좋다고 판단하였다. pivot과 같은 원소를 왼쪽과 오른쪽에 번갈아가며 넣을 수 있도록 Q를 수정했고, 수정부분을 Fig 3에 나타냈다. 하지만 이는 redundancy가 높을 때 사용한다면 오히려 시간복잡도의 상승으로 이어지므로, Fig 4의 실험을 진행하여 redundancy HP를 회귀분석을 이용하여 1.2%임을 결정하였다. 천만 이하 배열에 대해서 1.1~2% 근처에서 큰 차이가 없기 때문에 천만일 때의 하이퍼 파라미터를 적용하였다. 평소에는 본래의 Q를 사용하고, 하이퍼 파라미터보다 낮은 redundancy를 갖는 배열이 들어올 때 수정된 Q를 사용한다. Q와 수정된 Q를 같이 사용하면 M보다 항상 정렬시간이 짧게 만들 수 있고, Fig 5에서 관찰 가능하다.

```
private int partition(int p, int r, boolean isRevised) {
    int pivot_value = value[r];
    int i = p - 1, tmp;
    if(!isRevised){
        for (int j = p; j < r; j++) {
            if (value[j] < pivot_value) {
                i++;
                swap(i, j);
            }
        }
    } else {
        for (int j = p; j < r; j++) {
            if (value[j] < pivot_value || (value[j] == pivot_value && j%2 == 0)) {
                i++;
                swap(i, j);
            }
        }
    }
    swap(r, i+1);
    return i + 1;
}
```

Fig 3. Revised partition method

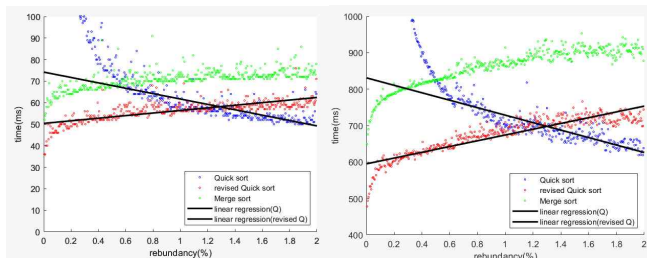


Fig 4. Q와 revisedQ의 비교(왼쪽: 백만, 오른쪽: 천만)

4.5 sortedPair

sortedPair값이 높을 때 Q보다 빠른건 M과 I이다. sortedPair값이 99% 이상일 때 M과 I를 비교해보았는데 배열의 크기가 백만일 때 99.95%, 천만일 때 99.95%를 초과하였을 때 I가 M보다 빨랐다. 실험 결과 $(100 - 5000 / (\text{double})n)\%$ 을 I와 M의 경계 HP로 지정하였다. 이를 적용하게 되면 10만, 100만일 때에도 적절한 경계이다. n은 배열의 크기이다.

sortedPair이 일정 값보다 높으면 Q는 pivot이 균등하게 나누지 못해 비효율적이며, M은 효율적이어서 M이 Q보다 정렬시간이 적게 나온다. 이에 대하여 배열의 크기가 십만, 백만, 천만, 억일 때 실험을 진행하였다. 실험 결과 각각 하이퍼 파라미터는 67.3%, 66.3%, 60.3%, 61.2%로 계산되어 천만 이하의 배열 크기에 대해서는 65%, 천만 이상의 배열에 대해서는 61%로 하이퍼 파라미터를 설정하였다. 또한, 역순의 배열에 대해 Q는 성능이 좋지 않기 때문에 sortedPair이 매우 낮을 때에도 고려를 해야한다. 이는 백만, 천만일 때의 결과를 통하여 하이퍼 파라미터를 계산하였다. 실험 결과 하이퍼 파라미터는 9.3%, 7.2%로 계산되었고, 천만 이하의 배열에서 9.3%, 천만 이상의 배열에서 7.2%로 HP 설정하였다.

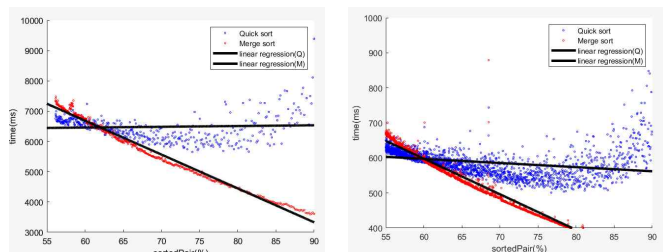


Fig 5. High sortedPair(왼쪽: 천만, 오른쪽: 억)

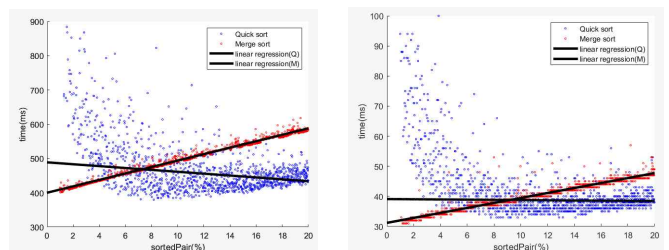


Fig 6. Low sortedPair(왼쪽: 백만, 오른쪽: 천만)

4.6 Search method

5만 이상에서 사용할 수 있는 Search method를 구현한 결과가 Fig 7이다. revised Q는 q로 나타냈고, 실제 java 파일에서 Q에 boolean parameter를 추가하여 Q와 revised Q를 선택하여 사용할 수 있도록 구현해두었다. Search method 두 개를 혼용해서 사용할 시에는 먼저 배열의 크기를 체크하고, 두 개의 method를 구분하여 사용하면 될 것이다. Table 1은 정렬쌍 검사와 관련된 search의 동작시간

과 각 정렬의 정렬시간으로 sortedPair이 매우 높은 경우 Q는 stack overflow가 발생하고, sortedPair가 50% 이하인 경우에는 I가 너무 오래걸려서 생략하였다. 각각의 경우에서 Search가 옳은 정렬을 반환하는 것을 확인할 수 있다. Table 2는 중복도 검사와 관련된 Search의 동작시간, 각 정렬의 정렬 시간을 나타낸 것이다. 1.2%보다 낮을 때 모두 revised Q가 Q보다 정렬시간이 짧고, 10% 가까이에서는 Q가 revised Q보다 정렬시간이 짧은 것을 확인할 수 있어서 제대로 된 HP를 설정하였다고 볼 수 있다.

```
private static char DoSearch(int[] value)
{
    int arrSize=value.length;

    //rebundancy
    double rebundancy=redundancy(value);

    //수정된 Q를 사용하는 경우
    if(rebundancy<=1.2) return 'q';

    //sortedPair
    double sortedPair=sortedPair(value);

    //I를 사용하는 경우
    if(sortedPair>100-5000/(double)arrSize) return 'I';

    //Q 대신 M을 사용하는 경우
    if(arrSize<10000000){
        if(sortedPair<7.2||sortedPair>61) return 'M';
    }
    else {
        if(sortedPair<9.3||sortedPair>65) return 'M';
    }

    return 'Q';
}
```

Fig 4. Search method(5만 이상)

N, rebundancy	S [ms]	Q [ms]	q [ms]	M [ms]
십만, 1.01	2.32, q	12.13	9.12	10.68
십만, 10.00	4.378, Q	11.01	12.35	11.89
백만, 1.04	21.35, q	77.13	71.42	80.21
백만, 9.99	39.72, Q	71.08	100.24	93.49
천만, 1.03	93.81, q	756.67	698.12	834.78
천만, 9.99	472.12, Q	688.58	914.18	976.47
억, 1.05	3859.22, q	7842.18	7012.34	9854.25
억, 9.99	1152.07, Q	7762.72	9814.13	9713.45

Table 1. Search, Q, q, M 동작시간(중복도 검사)

N, sortedPair	S [ms]	Q [ms]	M [ms]	I [ms]
십만, 99.98	8.62, I	stack overflow	6.47	3.82
십만, 50.12	11.86, Q	9.62	10.46	N/A
십만, 3.84	9.22, M	11.52	6.79	N/A
백만, 99.9951	62.22, I	stack overflow	29.4	14.5
백만, 49.98	97.73, Q	65.07	87.66	N/A
백만, 5.65	74.38, M	72.15	58.75	N/A
천만, 99.99954	606.7, I	stack overflow	217.43	167.34
천만, 50.91	1207.72, Q	642.88	932.76	N/A
천만, 9.06	796.82, M	427.24	629.05	N/A

Table 2. Search, Q, M, I 동작시간(정렬쌍 검사)