Radix Sort
(기수정렬)

㈜한컴에듀케이션 이주현

O(n) 정렬

❖ 비교정렬의 하한은 **O**($n\log n$)이지만 입력된 자료의 원소가 제한적인 성질을 만족하는 경우 **O**(n)정렬이 가능하다.

❖ 계수 정렬(counting sort)

- 원소의 범위가 k (–O(n) ~ O(n)) 범위의 정수인 경우

❖ 기수 정렬(radix sort)

- 원소의 자리수가 d 이하의 자리인 경우 (d : 상수로 간주될 크기)

1. 계수정렬(counting sort)

- ❖ 원소의 범위가 제한적일때 원소의 개수를 세어 정렬하는 방법이다.
- ❖ 개수를 셀 배열과 정렬된 결과가 저장될 배열이 추가로 필요하다.
- ❖ 일반적으로 정수 또는 문자를 정렬하는 경우에 많이 사용될 수 있다.
- ❖ 다양한 구현 방법이 있지만
  기수정렬(radix sort)과 연관된 버전으로 살펴본다.
- ❖ A[]배열에 아래와 같은 수들이 담겨있다고 하자.

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	3	2	2	4	3	5	3	1

계수정렬 예제

❖ A[]에 담긴 원소의 개수를 세어 count[]에 저장한다.

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	3	2	2	4	3	5	3	1

```
n = 10, k = 6;
for(i=0;i<k; ++k) count[i] = 0;          /// initialize count array
for(i=0;i<n;++i) count[A[i]] ++;          /// counting
```

❖ count[]에 담긴 결과이다.

index	0	1	2	3	4	5
count[i]	0	3	2	3	1	1

계수정렬 예제 cont.

❖ count[]에 담긴 결과이다.

index	0	1	2	3	4	5
count[i]	0	3	2	3	1	1

```
for(i=1;i<k;++i) count[i] += count[i-1]; /// accumulate
```

❖ count[]에 누적합을 구한다.

index	0	1	2	3	4	5
count[i]	0	3	5	8	9	10

계수정렬 예제 cont.

❖ count[]과 A[]를 이용하여 정렬된 결과 soredA[]를 구한다.

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	3	2	2	4	3	5	3	1

index	0	1	2	3	4	5
count[i]	0	3->2	5	8	9	10

```
for(i=n-1;i>=0;--i){ /// sorting
    sortedA[--count[A[9]]] = A[9];
}
```

index	0	1	2	3	4	5	6	7	8	9
sortedA[i]			1							

계수정렬 예제 cont.

❖ count[]과 A[]를 이용하여 정렬된 결과 soredA[]를 구한다.

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	3	2	2	4	3	5	3	1

index	0	1	2	3	4	5
count[i]	0	2	5	8->7	9	10

```
for(i=n-1 ~ 0){  /// sorting
      sortedA[--count[A[8]]] = A[8];
}
```

index	0	1	2	3	4	5	6	7	8	9
sortedA[i]			1					3		

계수정렬 예제 cont.

❖ count[]과 A[]를 이용하여 정렬된 결과 soredA[]를 구한다.

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	3	2	2	4	3	5	3	1

index	0	1	2	3	4	5
count[i]	0	2	5	7	9	10->9

```
for(i=n-1 ~ 0){  /// sorting
      sortedA[--count[A[7]]] = A[7];
}
```

index	0	1	2	3	4	5	6	7	8	9
sortedA[i]			1					3		5

계수정렬 예제 cont.

❖ count[]과 A[]를 이용하여 정렬된 결과 soredA[]를 구한다.

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	3	2	2	4	3	5	3	1

index	0	1	2	3	4	5
count[i]	0	2	5	7->6	9	9

```
for(i=n-1 ~ 0){  /// sorting
      sortedA[--count[A[6]]] = A[6];
}
```

index	0	1	2	3	4	5	6	7	8	9
sortedA[i]			1				3	3		5

계수정렬 예제 cont.

❖ count[]과 A[]를 이용하여 정렬된 결과 soredA[]를 구한다.

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	3	2	2	4	3	5	3	1

index	0	1	2	3	4	5
count[i]	0	2	5	6	9->8	9

```
for(i=n-1 ~ 0){  /// sorting
      sortedA[--count[A[5]]] = A[5];
}
```

index	0	1	2	3	4	5	6	7	8	9
sortedA[i]			1				3	3	4	5

계수정렬 예제 cont.

❖ count[]과 A[]를 이용하여 정렬된 결과 soredA[]를 구한다.

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	3	2	2	4	3	5	3	1

index	0	1	2	3	4	5
count[i]	0	2	5->4	6	8	9

```
for(i=n-1 ~ 0){  /// sorting
      sortedA[--count[A[4]]] = A[4];
}
```

index	0	1	2	3	4	5	6	7	8	9
sortedA[i]			1		2		3	3	4	5

계수정렬 예제 cont.

❖ count[]과 A[]를 이용하여 정렬된 결과 soredA[]를 구한다.

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	3	2	2	4	3	5	3	1

index	0	1	2	3	4	5
count[i]	0	2	4->3	6	8	9

```
for(i=n-1 ~ 0){  /// sorting
    sortedA[--count[A[3]]] = A[3];
}
```

index	0	1	2	3	4	5	6	7	8	9
sortedA[i]			1	2	2		3	3	4	5

계수정렬 예제 cont.

❖ count[]과 A[]를 이용하여 정렬된 결과 soredA[]를 구한다.

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	3	2	2	4	3	5	3	1

index	0	1	2	3	4	5
count[i]	0	2	3	6->5	8	9

```
for(i=n-1 ~ 0){  /// sorting
      sortedA[--count[A[2]]] = A[2];
}
```

index	0	1	2	3	4	5	6	7	8	9
sortedA[i]			1	2	2	3	3	3	4	5

계수정렬 예제 cont.

❖ count[]과 A[]를 이용하여 정렬된 결과 soredA[]를 구한다.

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	3	2	2	4	3	5	3	1

index	0	1	2	3	4	5
count[i]	0	2->1	3	5	8	9

```
for(i=n-1 ~ 0){  /// sorting
      sortedA[--count[A[1]]] = A[1];
}
```

index	0	1	2	3	4	5	6	7	8	9
sortedA[i]		1	1	2	2	3	3	3	4	5

계수정렬 예제 cont.

- count[]과 A[]를 이용하여 정렬된 결과 soredA[]를 구한다.

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	3	2	2	4	3	5	3	1

index	0	1	2	3	4	5
count[i]	0	1->0	3	5	8	9

```
for(i=n-1 ~ 0){ /// sorting
      sortedA[--count[A[0]]] = A[0];
}
```

index	0	1	2	3	4	5	6	7	8	9
sortedA[i]	1	1	1	2	2	3	3	3	4	5

계수정렬(counting sort) – code sample

```
/// 원소의 범위 k ( 0 ~ 5) 라고 가정
int A[10], sortedA[10], count[10], k = 6;
void countingSort(int A[], int n){
    int i ;
    for(i=0; i< k; ++i) count[i] = 0;              /// initialize count array
    for(i=0; i<n; ++i) count[A[i]] ++;             /// counting
    for(i=1; i< k; ++i) count[i] += count[i-1];   /// accumulate
    for(i=n-1; i>=0; --i){                         /// sorting
        sortedA[--C[ A[i] ] ] = A[i];
    }
}
```

2. 기수정렬(radix sort)

- ❖ 원소의 자리수가 d자리 이하인 N개의 수들로 이루어진 경우에 O(d * N) 의 시간복잡도를 갖는 정렬방법이다.
- ❖ 낮은 자리부터 정렬하고 정렬된 순서를 유지하면서 보다 높은자리를 정렬하는 과정에서 counting sort(계수 정렬)가 사용된다.
- ❖ 따라서 개수를 셀 배열과 정렬된 결과가 저장될 배열이 추가로 필요하다.
- ❖ 자리수별로 정렬할 때 몫과 나머지 연산을 사용하게 되는데 10진 기수법을 이용하면 효율성이 떨어진다. 이 때문에 진법수로 2의 제곱수를 선택하게 되는데 여기서는 256(2의 8제곱)진법으로 설명한다. 이 경우 d = 4가 된다.
- ❖ 정수라면 음수가 포함된 경우에도 정렬가능하다.
- ❖ 또한 IEEE754를 사용하는 시스템이라면 부동소수점 실수도 기수정렬을 사용할 수 있다.

기수정렬 예제 – 10진 기수를 예로

❖ 먼저 1의 자리를 기준으로 정렬한다.

3251
3
45
7918
397
469
6138
42

3251
42
3
45
397
7918
6138
469

기수정렬 예제 – 10진 기수를 예로

❖ 순서를 유지하면서 10의 자리를 기준으로 정렬한다.

3251	3251	3
3	42	7918
45	3	6138
7918	45	42
397	397	45
469	7918	3251
6138	6138	469
42	469	397

1의 자리 정렬 순서를
유지해야 한다.

기수정렬 예제 – 10진 기수를 예로

❖ 순서를 유지하면서 100의 자리를 기준으로 정렬한다.

3251	3251	3	3
3	42	7918	42
45	3	6138	45
7918	45	42	6138
397	397	45	3251
469	7918	3251	397
6138	6138	469	469
42	469	397	7918

기수정렬 예제 – 10진 기수를 예로

❖ 순서를 유지하면서 1000의 자리를 기준으로 정렬한다.

3251	3251	3	3	3
3	42	7918	42	42
45	3	6138	45	45
7918	45	42	6138	397
397	397	45	3251	469
469	7918	3251	397	**3**251
6138	6138	469	469	**6**138
42	469	397	7918	**7**918

기수정렬 예제 – 10진 기수를 예로 정렬한 결과

❖ 4자리수 이므로 O(4 * N) 시간복잡도이다.

3251
3
45
7918
397
469
6138
42

3
42
45
397
469
3251
6138
7918

기수정렬 코드 예 – 10진 기수

```
const int SIZE = 10;
const int DIGIT = 4;
int N, A[SIZE], B[SIZE], cnt[10];
void radixSort(){
    int i, j, deci = 1;
    for(i=0;i<DIGIT;++i, deci*=10){
        for(j=0;j<10;++j) cnt[j] = 0;
        for(j=0;j<N;++j) cnt[ A[j] / deci % 10 ] ++
        for(j=1;j<=10;++j) cnt[j] += cnt[j-1];
        for(j=N-1;j>=0;--j){
            B[--cnt[A[j] / deci % 10]] = A[j];
        }
        for(j=0;j<N;++j) A[j] = B[j];
    }
}
```

이부분은 계수정렬 코드 이다.
기수를 계수정렬하고 있다.

A[j]를 deci로 나눈 몫을
10으로 나눈 나머지이므로
0에서 9사이의 값이 나온다.

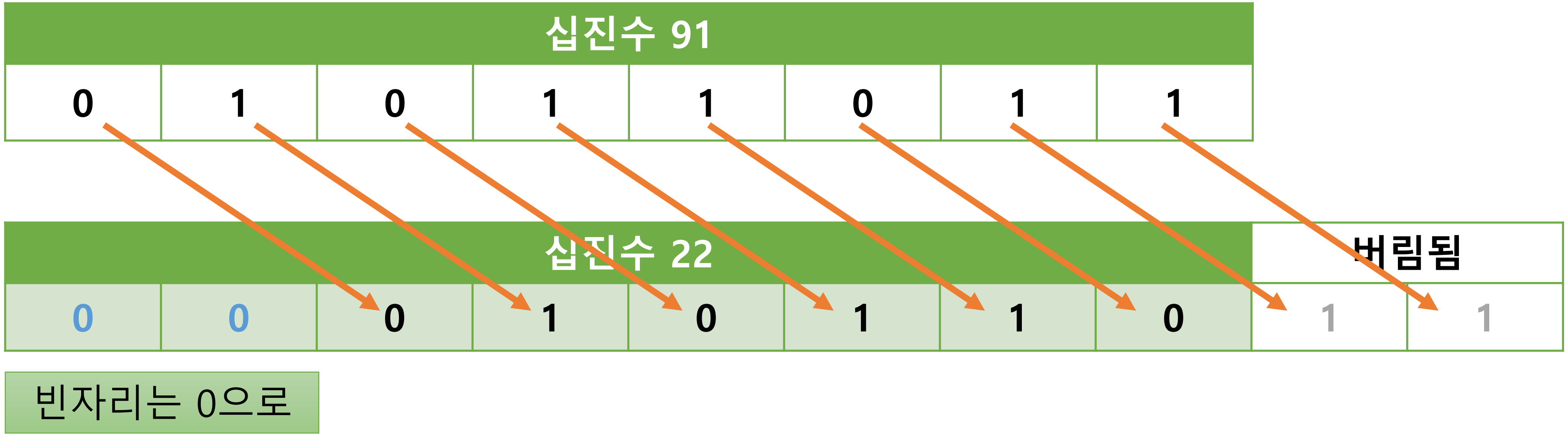
10진 기수정렬코드에 대한 고찰

- ❖ 위의 코드는 데이터의 개수가 많지 않은 경우에는 잘 수행된다. 하지만 데이터의 크기가 늘어나고 자리수가 9자리까지 늘어나면 시간이 많이 걸린다.
- ❖ 시간이 많이 걸리는 주요한 이유는 **A[j] / deci % 10** 부분에 있다.
  1. 나눗셈 연산('/')은 다른 연산에 비하여 시간이 많이 걸린다.
  2. 나머지 연산('%') 은 나눗셈 연산보다도 더 많이 걸린다.
     a % b 는 a – a / b * b 로 계산되기 때문이다.
- ❖ 이에 대한 해결방안으로 비트연산자를 생각할 수 있다.
  a / b 형식에서 b가 2의 제곱수인 경우 나누기는 '>>'를, 나머지는'&'를 이용하여 연산의 효율성을 높일 수 있다.
- ❖ 따라서 10진 기수가 아닌 16진, 256진, 65536진 기수 등 2의 제곱수 진법을 생각할 수 있는데 이 중 8비트를 사용하는 256진법으로 해결하고자 한다.
  실험적, 경험적으로 가장 효율이 좋았기 때문이다.

2의 제곱수로 나눈 몫

❖ a, b 가 unsigned int 정수라고 가정하자.
a >> b는 a를 2진수로 볼때 오른쪽으로 b칸 이동한다는 의미이다. 이동하여 최하위 비트 아래로 밀려난 수는 버림된다. 빈 자리가 되는 상위비트에는 0이 채워진다.
이말은 결국 a를 2의 b제곱으로 나눈 몫을 구한다는 의미가 된다.

❖ 예 : 91 >> 2 는 91 / 4 이므로 22가 된다. 아래 그림 참조



2의 제곱수로 나눈 나머지

❖ a를 4로 나눈 나머지를 생각해보자.
: a % 4가 될 수 있는 수들을 이진 비트로 표현 하면 00, 01, 10, 11 이다.
이들은 모두 최하위 2비트로서 그 이상의 비트들이 모두 제거된 그 나머지 값들이다.

❖ 비트연산자('&')를 이용하면 아주 간단히 이들을 얻을 수 있다.

- ✓ a % 2 => a & 1로 계산
- ✓ a % 4 => a & 3로 계산
- ✓ a % 8 => a & 7로 계산

  …

- ✓ 즉, a % (2의 b제곱수)는 a & ((2의 b제곱수) – 1)로 구할 수 있다는 것이다.
  이것은 아주 유용하게 사용 될 수 있다. (2의 b제곱수는 (1 << b)로 간단히 구해진다.)

❖ 이제 다음 슬라이드의 기수정렬코드중 가장 어려워 보이는 부분을 살펴보자.
코드에서 (ap[j] >> i) & MASK 는 MASK 가 (1<<8) – 1 이므로
ap[j]를 (2의 i제곱수)로 나누고 (2의 8제곱수)로 나눈 나머지를 구한다는 의미이다.

기수정렬 코드 예 – 256진 기수

```
const int SIZE = (int)16e6 + 5;
const int BASE = (1<<8);
const int MASK = BASE - 1;
int N, A[SIZE], B[SIZE], cnt[BASE];
void radixSort(){
    int i, j;
    int *ap = A, *bp = B, *tp;      /// swap 연산을 위하여
    for(i=0;i<32;i+=8){
        for(j=0;j<BASE;++j) cnt[j] = 0;
        for(j=0;j<N;++j) cnt[( ap[j] >> i) & MASK ]++;
        for(j=1;j<BASE;++j) cnt[j] += cnt[j-1];
        for(j=N-1;j>=0;--j){
            bp[ --cnt[ ( ap[j] >> i) & MASK ] ] = ap[j];
        }
        tp = ap, ap = bp, bp = tp; /// 배열 포인터의 swap 연산
    }
}
```

ap[j]를 (2의 i제곱수)로 나눈몫을 (2의 8제곱수)로 나눈 나머지이므로 0에서 255 사이의 값이 나온다.

기수정렬 관련문제

❖ jungol 3120 [기수정렬(Radix Sort)]
: 기수정렬 연습 문제이다.
: 음수가 포함된다는 점에 유의한다.
: 2의 보수표기법에 대한 이해가 필요하다.

❖ jungol 1357 [합이 0이 되는 4개의 숫자들]
: 더해서 0이 되는 문제를 같은 값을 찾는 문제로 바꿀수 있다.
: 해시를 이용한 해법도 가능하지만 기수정렬이
실행속도가 빠르다.

감사합니다.^^