**Homework 2C report**

**2014313303 홍태하(개발환경 : Linux)**

1. **알고리즘**

인풋이 10자리 16진수로 들어오는데 2차원 배열을 선언하여 최대 20개의 인풋을 행마다 하나씩 저장한다. 각 행에는 10개의 열이 있고 각 열마다 한 자릿수의 값을 저장한다. A~F의 알파벳으로 들어오는 인풋은 10~15의 숫자로 바꾸어 저장한다.

그 후 각 열 별로 1의 자리부터 counting sort를 이용하여 정렬하는데 낮은 자릿수부터 큰 자릿수까지 정렬하므로 radix sort이다.

1. **코드 설명.**
2. **인풋 받기.**

input\_c라는 변수에 character형태로 인풋을 하나씩 받아서 int형으로 바꿔 input\_i에 저장한다. 이 input\_i는 hex배열에 저장된다. ‘ ‘가 나오면 하나의 숫자가 끝난 것이므로 hex배열의 행을 다음 행으로 바꾸어 저장하도록 행을 의미하는 r을 하나 늘려주고 열을 의미하는 c는 0으로 바꿔준다. ‘\n’가 나오면 인풋이 끝난 것이므로 인풋 받는 것을 멈춘다. length변수에는 인풋의 개수를 저장한다.

1. **PrintAll 함수**

2차원 배열의 모든 값을 print한다. 이 때 10~15의 값은 A~F의 값으로 바꾸어 출력한다.

1. **Counting\_Sort 함수**

16진수이므로 16칸짜리 count라는 배열을 만든다. 이 배열을 0으로 초기화하고 정렬할 값과 같은 count배열의 인덱스의 값을 하나 늘려준다. 그 후 배열의 앞에서부터 전 칸의 값과 자신의 값을 더한 값을 자신의 자리에 저장한다. 그럼으로써 정렬할 값들의 위치를 알 수 있게 된다. 그 후 정렬할 값들의 뒤에서부터(뒤에서부터 정렬하면 stable한 sort가 된다.) 그 값과 같은 count배열의 인덱스의 값에서 1을 뺀(배열은 첫 인덱스가 0이므로 1을 뺀다.) 인덱스의 자리에 정렬할 값을 저장한다. 그 후 count배열의 값을 1줄여주는데 그 이유는 같은 값이 2개 이상 있을 경우 그 수들에게 들어갈 인덱스를 모두 주기 위해서이다. 같은 과정을 반복하면 정렬이 완료된다.

1. **Radix\_Sort 함수**

2차원배열에 각 행마다 하나의 10자리 16진수가 저장되어 있으므로 같은 행은 16진수들의 같은

자릿수의 값이다. 따라서 낮은 자릿수부터 Counting\_Sort를 이용해서 정렬을 한다. arr배열의 값들을 정렬하여 res배열에 저장하는데 매번 정렬된 값들로 다시 한 단계 높은 자리의 값을 Counting\_Sort로 정렬해야 하므로 Counting\_Sort가 끝나면 res배열의 값을 arr배열로 옮겨준다. 이 과정을 가장 높은 자릿수까지 반복하면 정렬이 완료된다.

1. **예시**

**2014313303 305A486FD0 02951DC764 A5168472DF**

인풋이 이렇게 들어오면 먼저 hex배열에 인풋들을 저장한다. 저장하면 첫 번째 행에는 2 0 1 4 3 1 3 3 0 3이 저장되고 두 번째 행에는 3 0 5 10 4 8 6 15 13 0, 세 번째 행에는 0 2 9 5 1 13 12 7 6 4, 네번째 행에는 10 5 1 6 8 4 7 2 13 15가 저장된다. length는 4가 된다. Radix\_Sort함수를 실행시키면 1의 자릿수부터 Counting\_Sort를 한다. 먼저 count배열을 0으로 초기화 시키고, 1의 자릿수는 3 0 4 15이므로 count배열의 인덱스에 해당하는 값들을 1씩 증가시킨다. 그러면 count배열은

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| Value | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

이렇게 된다. 그 후 정렬할 값들의 위치를 알기 위해 전 칸의 값과 자기 자리의 값을 더하면

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| Value | 1 | 1 | 1 | 2 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 4 |

가 된다. 그 후 정렬할 값들의 뒤에서부터 자기 자리를 찾으면 위 예시의 경우 15가 가장 뒤이므로 count배열에서 index 15를 찾는다. 그 값이 4이므로 15의 자리는 거기에서 1을 뺀 3이다. 따라서 res배열의 행의 인덱스 3에 A5168472DF를 저장한다. 그 후 count[15]의 값은 하나 줄여준다. 같은 방식으로 4는 인덱스 2의 자리로 가고 0은 인덱스 0의자리로, 3은 인덱스 1의 자리로 간다. 따라서 res배열은 첫 번째 행에 305A486FD0이, 두 번째 행에 2014313303이, 세 번째 행에 02951DC764, 네 번째 행에 A5168472DF가 저장된다(실제로는 A~F는 10~15의 정수로 저장되어 있다.). 위와 같은 과정을 가장 높은 자릿수까지 반복하면 output은 이렇게 된다.

**305A486FD0 2014313303 02951DC764 A5168472DF**

**2014313303 02951DC764 305A486FD0 A5168472DF**

**A5168472DF 2014313303 02951DC764 305A486FD0**

**2014313303 305A486FD0 A5168472DF 02951DC764**

**2014313303 A5168472DF 305A486FD0 02951DC764**

**02951DC764 2014313303 305A486FD0 A5168472DF**

**2014313303 02951DC764 A5168472DF 305A486FD0**

**2014313303 A5168472DF 305A486FD0 02951DC764**

**2014313303 305A486FD0 02951DC764 A5168472DF**

**02951DC764 2014313303 305A486FD0 A5168472DF**