

INTEGER ARRAY PACK / UNPACK

민경욱

1. 문제 분석

- 10,000개의 unsigned Int 형 배열에 1 ~ 1,000,000 의 숫자가 정렬되어 들어가 있다.
- 이 배열을 압축(Packed)하고 압축해제(UnPacked)를 독립적으로 수행 가능한 함수 Packed()와 Unpacked()를 작성하여야 한다.
- 힙, 스택에 데이터를 저장 후 복원은 안되며, 외부 라이브러리 함수는 사용이 불가능

2. 설계 과정 - PACK

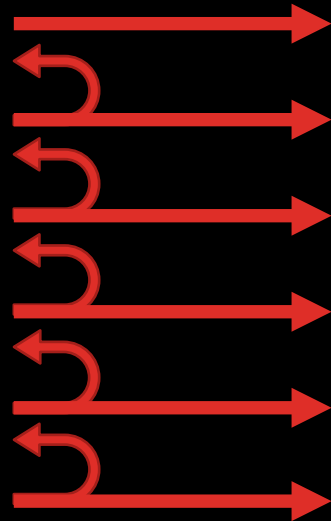
- Unsigned Int 배열은 각셀이 4 Byte = 32 Bit 이며 이 32bit의 셀에 최대한의 데이터를 모두 집어 넣어야함.
- 우선 10,000개의 정수형 배열에 들어있는 값들을 줄여서 사용하는 bit수를 줄인다.
- 그리고 배열 하나의 셀에 최대한 많은 데이터를 넣는다.

2. 설계 과정(계속)

- 각 셀의 데이터가 사용하는 비트를 줄이고, 원래대로 복구가 가능해야 하기 때문에 0번째 셀을 제외하고 $[i+1]$ 번째 셀에서 $[i]$ 번째 셀의 값을 뺀 뒤 차이값 저장.

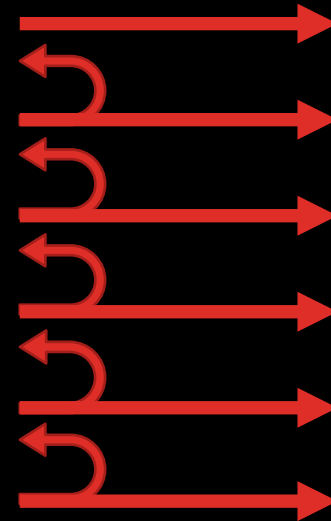
빼기를 해서 크기를 줄임

10
25
35
60
70
90



덧셈하여 원래 데이터로 복원

10
15
10
25
10
20



10
25
35
60
70
90

2. 설계 과정(계속)

1) 배열 하나에 두개 셀 데이터를 저장

0000 0000 0000 0000 0000 0000 0000 0000



2) 배열 하나에 세개 셀 데이터를 저장

case1 : 0000 0000 0000 0000 0000 0000 0000 0000



case2 : 0000 0000 0000 0000 0000 0000 0000 0000



case3 : 0000 0000 0000 0000 0000 0000 0000 0000



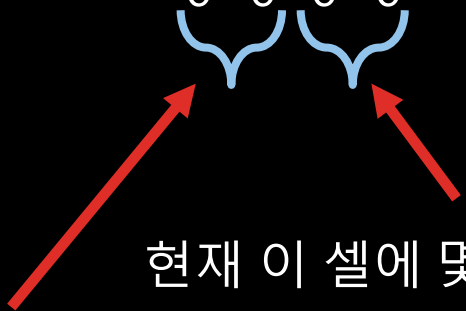
case4 : 0000 0000 0000 0000 0000 0000 0000 0000



2. 설계 과정(계속)

- 1) 마지막 4개의 비트에는 현재 이 셀에 몇 개 셀의 데이터가 있는지
그리고 만약 세개의 셀의 데이터가 들어있다면 몇번 케이스 인지 저장.

0 0 0 0



현재 이 셀에 몇개의 데이터가 압축 된 상태 인지

3개일 경우 case몇번인지
(0, 1, 2, 3)

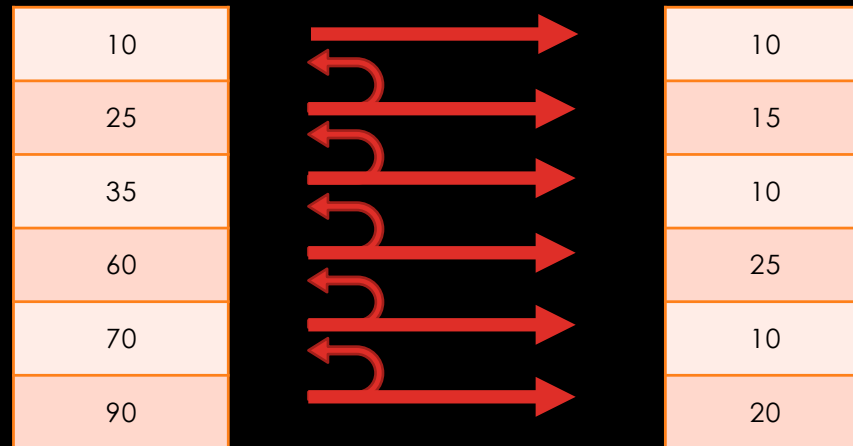
2. 설계 과정 - UNPACK

- 마지막 4비트를 먼저 확인해서 이 셀에 몇개의 데이터가 들어
가있는지 확인.
- 3개가 있다면 case몇번에 해당하는지 확인.
- Case에 따라 적절히 Shift연산을 통하여 데이터를 복원.

3. 알고리즘 - PACK

1. `card[1]`부터 `card[SIZE]`까지 i 번째에서 $i-1$ 번째 값을 뺀 데이터를 `temp[]`에 저장.

```
for(i = 1; i < SIZE; i++)  
{  
    temp[i] = card[i] - card[i-1];  
}
```



3. 알고리즘(계속)

2. 차이값을 저장한 배열에서 데이터를 압축하기 위해서 $i, i+1, i+2$ 까지의 데이터의 크기를 먼저 확인.

- 3개의 데이터가 모두 8bit이하거나 혹은 하나만 8bit를 넘는다면 3개씩 케이스별로 구분하여 데이터를 삽입.
- 나머지의 경우는 2개씩 삽입

3. 알고리즘(계속)

```
if(temp[i]<=255 && temp[i+1]<=255 && temp[i+2]<=255)
{
    temp[i] = temp[i] << 24;
    temp[i] = temp[i] + (temp[i+1]<<16);
    temp[i] = temp[i] + (temp[i+2]<<8);
    temp[i] += 3; //데이터 세개가 있다고 표시
    packedBinarySet[setIdx] = temp[i];
    setIdx++;
    i += 3;
}
```

1, i+1, i+2번째까지의 데이터가 모두 8bit 이하 숫자인 Case

0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

3. 알고리즘 (계속)

```
else if(temp[i]<=255 && temp[i+1]<=255 && temp[i+2]>255)
{
    temp[i] = temp[i] << 24;
    temp[i] = temp[i] + (temp[i+1]<<16);
    temp[i] = temp[i] + (temp[i+2]<<4);
    temp[i] += 3; //데이터 세개가 있다고 표시
    temp[i] += 4; //Case 1로 표시
    packedBinarySet[setIdx] = temp[i];
    setIdx++;
    i += 3;
}
```

1, i+1, i+2 번째까지의 데이터가 모두 8bit 이하 숫자인 Case

0000 0000 0000 0000 0000 0000 0000 0000
0110 0100 0100

3. 알고리즘 – UNPACK

1. 먼저 이 셀에 몇개의 데이터가 있는지 확인

```
cellCnt = temp[idx] << 30;  
cellCnt = cellCnt >> 30;
```

```
if(cellCnt==3)  
{  
    mode = temp[idx] << 28;  
    mode = mode >> 30;  
}
```

0011 0101 1110 0111 1100 1010 1010 0111

3. 알고리즘(계속)

```
else if(cellCnt==3 && mode == 1 )
{
    temp1 = temp[idx] >> 24;
    temp2 = temp[idx] << 8;
    temp2 = temp2 >> 24;
    temp3 = temp[idx] << 16;
    temp3 = temp3 >> 20;

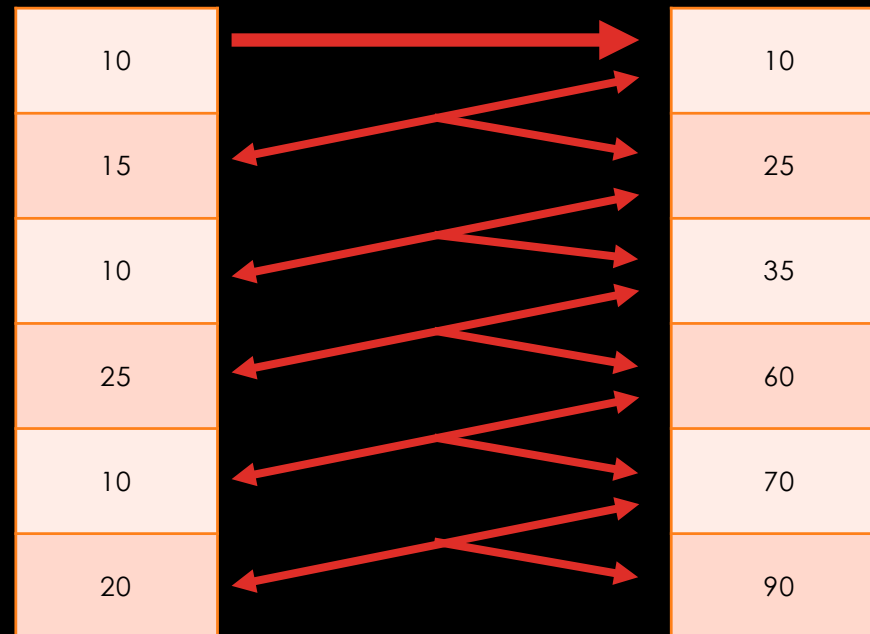
    tempBinary[(tmpIdx)] = temp1;
    tempBinary[(tmpIdx+1)] = temp2;
    tempBinary[(tmpIdx+2)] = temp3;
    tmpIdx+=3;
}
```


```
0011 0101 1110 0111 1100 1010 1010 0111
0011 0101 1110 0111 1100 1010 1010
```

3. 알고리즘

이전 값과 다시 더해서 원래의 데이터를 복원 시킨다.

```
for(i = 1; i<SIZE; i++)  
{  
    unpacked[i] = unpacked[i-1] + tempBinary[i];  
}
```





Q & A