

# 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

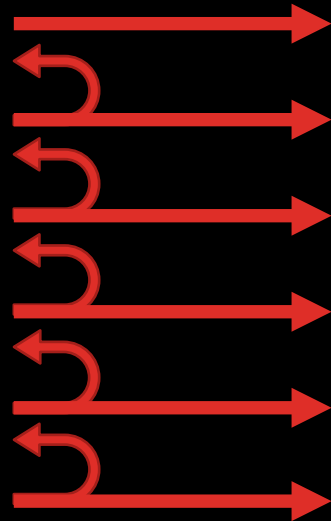
- Unsigned Int 배열은 각셀이 4 Byte = 32 Bit 이며 이 32bit의 셀에 최대한의 데이터를 모두 집어 넣어야함.
- 우선 10,000개의 정수형 배열에 들어있는 값들을 줄여서 사용하는 bit수를 줄인다.
- 그리고 배열 하나의 셀에 최대한 많은 데이터를 넣는다.

## 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.

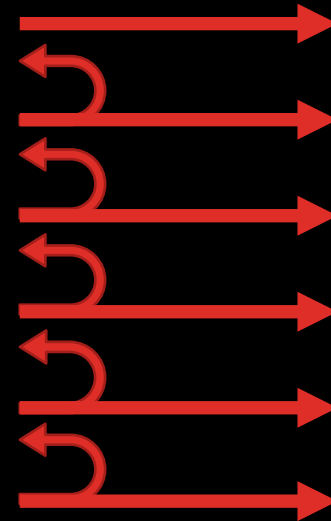
빼기를 해서 크기를 줄임

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



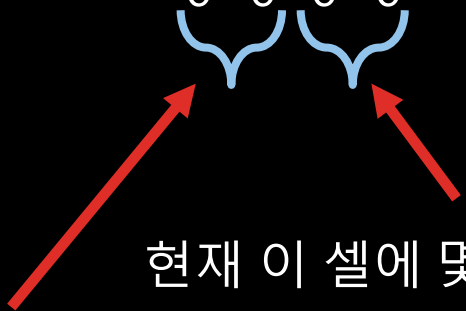
case4 : 0000 0000 0000 0000 0000 0000 0000 0000



## 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)

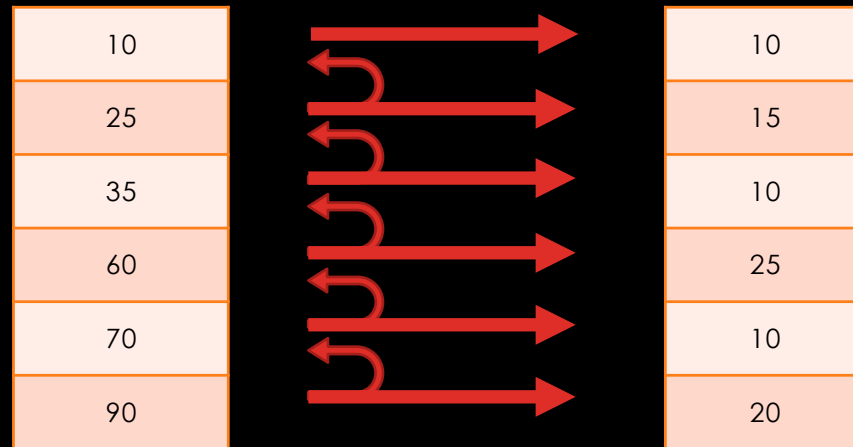
## 2. 설계 과정 - UNPACK

- 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 i번째에서 i-1번째 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+2번째까지의 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;

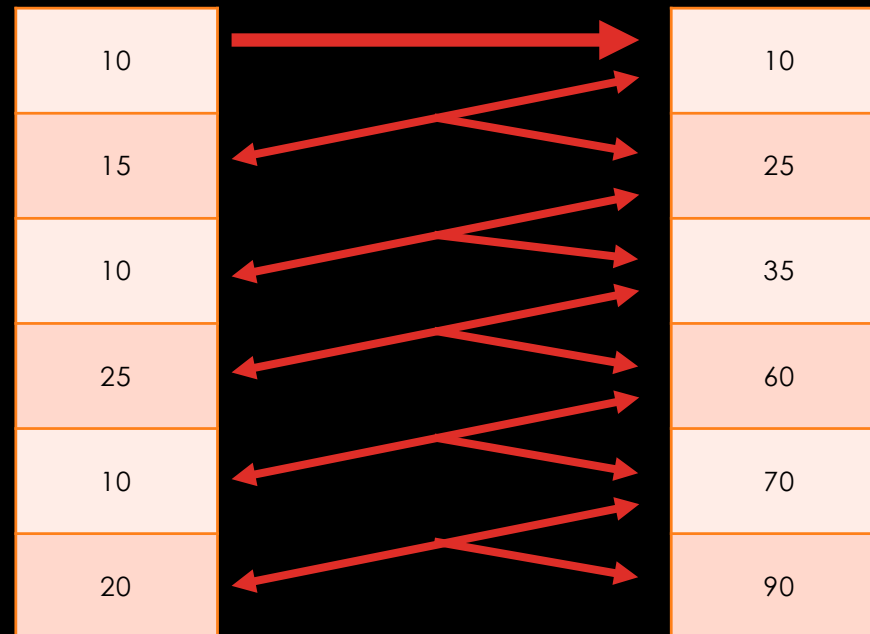
    tempBinary[(tmpIdx)] = temp1;
    tempBinary[(tmpIdx+1)] = temp2;
    tempBinary[(tmpIdx+2)] = temp3;
    tmpIdx+=3;
}
```


```
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

```
//
// Created by GyeongUkMin on 10/6/15.
// Copyright © 2015 GyeongUckMin. All rights reserved.
//
// This code is to pack and unpack integer array that has 10,000 integers.

#define SIZE 10000

unsigned int temp[SIZE]={0};
unsigned int packedBinarySet[SIZE]={0}; //packed data will be saved here.
unsigned int tempBinary[SIZE] = {0};
int packedArrSize = 0;

int pack(unsigned int packed[SIZE], const unsigned int card[SIZE])
//This function is to pack integer array.
{
    int setIdx = 0, i = 0, cnt=0, cnt1=0;

    temp[0] = card[0];

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

        //Subtract card[i-1]'s int data from card[i]'s int data and repeat it
        //until end of cards.
        //Then all cards have a gap of each card's int data.
        //Ex) 10 - 25 - 35 - 60 - 70 - 90
        //     10 - 15 - 10 - 25 - 10 - 20
    }

    /**
    I make some cases.

    First : One int data type has two datas.

    Second : One int data type has three datas.
    For second case, I set some specific cases.
    Second-first : 8 + 8 + 8 + 8(add for mode bit) bit = 32 bit
    Second-second : 8 + 8 + 12 + 4(for mode bit) bit = 32 bit
    Second-third : 8 + 12 + 8 + 4(for mode bit) bit = 32 bit
    Second-four : 12 + 8 + 8 + 4(for mode bit) bit = 32 bit

    In mode bit (4 bit), last 2 bit use for how many data stored in this int,
    And first 2 bit use for recognize case.
    */

    for(i =0; i<SIZE-2; )
    {
        //If forward 3 cards's number are less than 256.
        //Operate these commands.

        if(temp[i]<=255 && temp[i+1]<=255 && temp[i+2]<=255)
        {
            // If number is less than 256, then I can store it by using just
            // 8bit.
            // This case is 8 + 8 + 8 + 8, Second-first case.
            // And Integer is 32 bit. Therefore, I can store 3 cards's data in
```



```

        one int data type.

temp[i] = temp[i] << 24;
temp[i] = temp[i] + (temp[i+1]<<16);
temp[i] = temp[i] + (temp[i+2]<<8);

temp[i] += 3;
// After store 3 cards's data, add mode bit at end of int type.
// add just 3(decimal), then mode bit will be 0 0 1 1.
// it means that this int has 3 cards's data and Second-first
    case.

packedBinarySet[setIdx] = temp[i];
setIdx++;
i += 3;
}

else if(temp[i]<=255 && temp[i+1]<=255 && temp[i+2]>255)
{
    // This case is 8 + 8 + 12 + 4, Second-second case.

temp[i] = temp[i] << 24;
temp[i] = temp[i] + (temp[i+1]<<16);
temp[i] = temp[i] + (temp[i+2]<<4);

temp[i] += 3;
temp[i] += 4;
// Add 3(dec) and 4(dec), then mode bit will be 0 1 1 1.
// it means that this int has 3 cards's data and Second-second
    case.

packedBinarySet[setIdx] = temp[i];
setIdx++;
i += 3;
}

else if(temp[i]<=255 && temp[i+1]>255 && temp[i+2]<=255)
{
    // This case is 8 + 12 + 8 + 4, Second-thrid case.

temp[i] = temp[i] << 24;
temp[i] = temp[i] + (temp[i+1]<<12);
temp[i] = temp[i] + (temp[i+2]<<4);

temp[i] += 3;
temp[i] += 8;
// Add 3(dec) and 8(dec), then mode bit will be 1 0 1 1.
// it means that this int has 3 cards's data and Second-third
    case.

packedBinarySet[setIdx] = temp[i];
setIdx++;
i += 3;
}

else if(temp[i]>255 && temp[i+1]<=255 && temp[i+2]<=255)
{
    // This case is 12 + 8 + 8 + 4, Second-four case.

temp[i] = temp[i] << 20;

```

```

    temp[i] = temp[i] + (temp[i+1]<<12);
    temp[i] = temp[i] + (temp[i+2]<<4);

    temp[i] += 3;
    temp[i] += 12;
    // Add 3(dec) and 12(dec), then mode bit will be 1 1 1 1.
    // it means that this int has 3 cards's data and Second-four case.

    packedBinarySet[setIdx] = temp[i];
    setIdx++;
    i += 3;
}
else if(i==9999)
{
    //If it reach end of array, just add data to result array.
    packedBinarySet[setIdx] = temp[i];
    packedBinarySet[setIdx] = packedBinarySet[setIdx] << 8;
    packedBinarySet[setIdx]+=1;
    setIdx++;
    i++;
}

else
{
    //This case is First case that can store just two datas.
    //12 + 12 + 8(for mode bit) bit = 32 bit.
    temp[i] = temp[i] << 20;
    temp[i] = temp[i] + (temp[i+1]<<8);

    temp[i] += 2;
    // Add 2(dec), then mode bit will be 0 0 1 0.
    // it means that this int has 2 cards's data.

    packedBinarySet[setIdx] = temp[i];
    setIdx++;
    i+=2;
}
}

for(;i<10000;i++)
{
    packedBinarySet[setIdx] = temp[i] << 8;
    packedBinarySet[setIdx] +=1;
    setIdx++;
}

for(i = 0; i<SIZE; i++)
{
    if(packedBinarySet[i]>0)
    {
        packed[i] = packedBinarySet[i];
    }
}

packedArrSize = setIdx;

return packedArrSize; // return packed array size.

```

```

}
void unpack(unsigned int unpacked[SIZE], const unsigned int packed[SIZE], int
    size)
{
    int idx = 0, tmpIdx = 0, i = 0;
    unsigned int temp1=0, temp2=0, temp3=0;
    unsigned int cellCnt = 0, mode = 0;

    for(i =0; i<SIZE; i++)
    {
        temp[i] = packed[i];
        //copy all packed array data to temp array.
    }
    for(idx = 0; idx<size; idx++)
    {
        cellCnt = temp[idx] << 30;
        cellCnt = cellCnt >> 30;
        //from mode bit(last 4 bit), get information about how many number
        //stored in here.
        //ex) 3 == has three data, 2 == has two data.

        if(cellCnt==3)
        {
            mode = temp[idx] << 28;
            mode = mode >> 30;
            //if cellCnt is 3, get information about case...(Second-first,
            //Second-second...).
        }

        if(cellCnt==1)
        {
            tempBinary[tmpIdx] = temp[idx] >> 8;
            tmpIdx++;
            //this case is end of packed array data.
            //so just unpack by using simple 8 right shift operation.
        }

        else if(cellCnt==3 && mode ==0)
        {
            /**
             * this case is Second-first case, so get 3 int datas from it by
             * using proper shift operations.
             */
            temp1 = temp[idx] >> 24;
            temp2 = temp[idx] << 8;
            temp2 = temp2 >> 24;
            temp3 = temp[idx] << 16;
            temp3 = temp3 >> 24;

            tempBinary[(tmpIdx)] = temp1;
            tempBinary[(tmpIdx+1)] = temp2;
            tempBinary[(tmpIdx+2)] = temp3;
            tmpIdx+=3;
        }
        //below ifs are same, just has difference of cases.
        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;
    }
    else if(cellCnt==3 && mode == 2)
    {
        temp1 = temp[idx] >> 24;
        temp2 = temp[idx] << 8;
        temp2 = temp2 >> 20;
        temp3 = temp[idx] << 20;
        temp3 = temp3 >> 24;

        tempBinary[(tmpIdx)] = temp1;
        tempBinary[(tmpIdx+1)] = temp2;
        tempBinary[(tmpIdx+2)] = temp3;
        tmpIdx+=3;
    }
    else if(cellCnt==3 && mode ==3)
    {
        temp1 = temp[idx] >> 20;
        temp2 = temp[idx] << 12;
        temp2 = temp2 >> 24;
        temp3 = temp[idx] << 20;
        temp3 = temp3 >> 24;

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

    else {
        //this case is to unpack First case(has 2 data)
        temp1 = temp[idx] >> 20;
        temp2 = temp[idx] << 12;
        temp2 = temp2 >> 20;

        tempBinary[tmpIdx] = temp1;
        tempBinary[tmpIdx+1] = temp2;
        tmpIdx+=2;
    }
}

unpacked[0] = tempBinary[0];

for(i = 1; i<SIZE; i++)
{
    unpacked[i] = unpacked[i-1] + tempBinary[i];
    /**
    After unpack all datas, we need to calculate originally data by using
    add with unpacked[i-1] and unpacked[i].
    Ex) 10 - 15 - 10 - 25 - 10 - 20 (just unpacked data)
        10 - 25 - 35 - 60 - 70 - 90 (restore originally data)
    */
}

```

```
}
```