```
//
//  Created by GyeungUkMin on 10/6/15.
//  Copyright © 2015 GyeungUckMin. 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)
         **/
    }
```

```
    }
```