【Adding Reversed Numbers】

```
#include <iostream>         // Preprocessor Directive
#include <cstdio>
#include <cstring>
#include <string>
using namespace std;         // Using C + +  Standard Libeary
int Num[3][1000];
void Read(int Ord)         // If Ord==0, input the first addend ; If Ord==1, input the second
addend
{
    int flag=0;
    string Tmp;
    cin>>Tmp;                                   // Input the string represent the integer
    for(int i=Tmp.length( )-1;i>=0;i--)    // Analyze each charchter from right to left
    {
        if(Tmp[i] > '0')                      // Store the integer into Num[Ord]
            flag = 1;
        if(flag)
            Num[Ord][++Num[Ord][0]] = Tmp[i] - '0';
    }
    for(int i=Num[Ord][0],j=1;i>j;i--,j++)            //   Get reversed number Num[Ord]
    {
        flag = Num[Ord][i];
        Num[Ord][i] = Num[Ord][j];
        Num[Ord][j] = flag;
    }
}
void Add( )
{
    Num[2][0] = max(Num[0][0],Num[1][0]);   // the number of additions is the maximum
length of two addends
    for(int i=1;i<=Num[2][0];i++)                  // Bitwise addition
        Num[2][i] = Num[0][i] + Num[1][i];
    for(int i=1;i<=Num[2][0];i++)                  // Carry
    {
        Num[2][i+1] += Num[2][i]/10;
        Num[2][i] %= 10;
    }
    if(Num[2][Num[2][0]+1] > 0)               // Carry
        Num[2][0] ++;
    int flag = 0;
    for(int i=1;i<=Num[2][0];i++)         // Output the reversed sum of two reversed numbers
    {
        if(Num[2][i] > 0)
            flag = 1;
        if(flag)
            printf("%d",Num[2][i]);
    }
    printf("\n");
}
int main( )                                   // Main function
{
    int N;                                    // The number of test cases
    cin>>N;
    for(N;N;N--)                              // Input and process for each test case
    {
        memset(Num,0,sizeof(Num));    // Initialize arrays of high precision numbers 0
        Read(0);                             // The first addend
        Read(1);                             // The second addend
        Add( );                  // Add two reversed numbers, and output their reversed sum
    }
    return 0;
}
```

【Fence Repair】

```
#include <iostream>
using namespace std;
const long maxn = 20000 + 10;    //size of the heap
long n, len;                      //n: the number of planks, len: the length of the heap
long long p[maxn];                //heap
void heap_insert(long long k)
{                                 //insert k into the min heap, maintain the heap property
        long t = ++len;
        p[t] = k;
        while (t > 1)
                if (p[t/2]>p[t]) {
                        swap(p[t], p[t / 2]);
                        t /= 2;
                } else
                        break;
}
void heap_pop(void)              // Delete the root of the min heap, maintain the heap property
{
        long t = 1;
        p[t] = p[len--];
        while (t * 2 <= len) {
                long k = t * 2;
                if (k < len && p[k] > p[k + 1])
                        ++k;
                if (p[t]>p[k]){
                        swap(p[t], p[k]);
                        t = k;
                } else
                        break;
        }
}
int main(void)
{
        cin >> n;
        for (long i = 1; i <= n; i++)          //lengths of n planks
                cin >> p[i];
```

```
    len = 0;
    for (long i = 1; i <= n; i++)           //a min heap is constructed with n planks
        heap_insert(p[i]);
    long long ans = 0;
    while (len > 1) {                       //construct a Huffman tree
        long long a, b;
        a = p[1];     //delete the root of heap (weight a), maintain the heap property
        heap_pop();
        b = p[1];     // delete the root of heap (weight b), maintain the heap property
        heap_pop();
        ans += a + b;                     // the cost ans increases (a+b)
        heap_insert(a + b);         // A new node (a+b) is inserted into the min heap
    }
    cout << ans << endl;                  //Output the minimal cost
}
```

【Prime Path】

```
#include<iostream>
#include<string>
using namespace std;
struct node{
        int k, step;          // current prime number k, the length of the path (the number of changed
digits ) step
};
node h[100000];                         //Queue
bool p[11000];                          // Sieve
int x, y, tot, s[11000];                // Initial prime x, goal prime y, the number of remainder test
cases tot, the current shortest path s[x] for prime x
void make(int n){                       // Get primes in [2..n] by sieve method
        memset(p, 0, sizeof(p));
        p[0]=1;
        p[1]=1;
        for (int i=2; i<=n; i++) if (!p[i])
        for (int j=i*i; j<=n; j+=i) p[j]=1;
}
int change(int x, int i, int j){      // change the i-th digit of x into j
        if (i==1) return (x/10)*10+j; else
        if (i==2) return (x/100)*100+x%10+j*10; else
        if (i==3) return (x/1000)*1000+x%100+j*100; else
        if (i==4) return (x%1000)+j*1000;
}
int main(){
        make(9999);                             // Get primes in [2..9999]
        cin>>tot;                               // the number of test cases
        while (tot--){
                cin>>x>>y;                      // initial prime x and goal prime y
                h[1].k=x;                       // initial prime x is pushed into the queue
                h[1].step=0;
                int l=1,r=1;                    // Initialize pointers of the queue
                memset(s, 100, sizeof(s));       //Initialize the length of the path
                int ans=-1;                     // Initialize the minimal cost
                while (1){
                        if (h[l].k==y) {        // goal prime y is gotten
                ans=h[l].step;
                break;
            }
            int tk,ts;
            for (int i=1; i<=4; i++)    // every digit of the front for the queue is changed
              for (int j=0; j<=9; j++) if (!((j==0)&&(i==4))){//Enumerate
                tk=change(h[l].k, i, j);
                if (p[tk]) continue;        // If tk isn’t a prime
                ts=h[l].step+1;                // the length of the path to tk
                if (ts>=s[tk]) continue;
                if (tk==y){          // If tk is the goal prime
                    ans=ts;
                    break;
                }
                s[tk]=ts;                  // the length of the path to tk
                r++;
                h[r].k=tk;              // Prime tk and its length of the path is pushed
                h[r].step=ts;
            }
            if (l==r||ans>=0) break; // If the queue is empty or the goal prime is arrived
            l++;
        }
        if (ans>=0) cout<<ans<<endl; else cout<<"Impossible"<<endl;    // Output the result
    }
}
```