Contents

1 Basic

1.1 Disjoint Set

```
/*
 * Easy disjoint set implmentation
 * Author: roy4801
 * Team: FJU_ElPsyCongroo
 * ver 0.0.1
 */
#define SIZE 1000005
int p[SIZE];

/*
 * void init()
 * Description: Initialize the disjoint set
 */
void init()
{
    for(int i = 0; i < SIZE; i++)
        p[i] = i;
}
/*
 * int find(const int x)
 * Description: Find the team leader of idx x
 */
int find(const int x)
{
    return x==p[x] ? x : find(p[x]);
}
/*
 * void uni(const int a, const int b)
 * Description: Make a and b same group
 */
void uni(const int a, const int b)
{
    p[find(a)] = p[find(b)];
}
/*
 * bool equ(const int a, const int b)
 * Description: If a and b are in the same group
 */
bool equ(const int a, const int b)
{
    return find(a) == find(b);
}
```

1.2 int128

```
/*
 * __int128 print and scan function implmentation
 * Author: roy4801
 * Team: FJU_ElPsyCongroo
 * ver 0.0.1
 */
#include <iostream>
#include <assert.h>

/*
 * int print_i128(_int128 i128)
 * Description: Print a __int128 to stdout
 */
static int print_i128(__int128 i128)
{
    char ch128[40], *now = ch128, *head = ch128;
    int len = 0;

    if(i128 < 0)
    {
        putchar('-');
        i128 = -i128;
    }
    // Turn __int28 into char[] from lowest digit
    while(i128 > 9)
    {
        *now++ = i128 % 10 + '0';
        i128 /= 10;
    }
    *now = i128 + '0';

    // Print
    while(now >= head)
    {
        putchar(*now--);
    }

    return 1;
}
/*
 * int scan_i128(__int128 *n)
 * Description: Reads a __int128 to the passed in __int128 *
 */
static int scan_i128(__int128 *n)
{
    #ifdef DBG
    assert(n != NULL);
    #endif
    char num[40], *now = num;
    bool minus = false;
    *n = 0; // reset n

    int ret = scanf("%s", num);
    if(ret == EOF) // scanf fails
        return EOF;
    // Judge if minus
    if(*now == '-')
    {
        minus = true;
        now++; // skip '-'
    }

    // Add the digit and multiply it by 10 one after another
    while(*now)
    {
        *n += *now - '0';
        now++;
        if(*now) // check if now touches '\0'
            *n *= 10;
    }

    *n = minus ? -(*n) : *n;

    return 1;
}
```

1.3 sieve

```
/*
 * Sieve of Eratosthenes Implementation
 * Author: roy4801
 * Team: FJU_ElPsyCongroo
 */
```

```cpp
#include <iostream>

/*
 * Sieve of Eratosthenes
 *
 * from 2 to n , begining at 2 and delete all of its multiples and do it over and over again
 * until all multiples are deleted in [2, n]
 */

#define TABLE_SIZE 100000

bool prime[TABLE_SIZE];

void buildPrimeTable()
{
    prime[0] = prime[1] = false;
    for(int i = 2; i < TABLE_SIZE; i++)
        prime[i] = true;

    for(int i = 2; i < TABLE_SIZE; i++)
    {
        if(prime[i])
            for(size_t a = i*i; a < TABLE_SIZE; a += i)
                prime[a] = false;
    }
}
```

2 Sequence

2.1 RMQ

2.1.1 seg-tree

```cpp
void buildSegTree(int segTree[], int val[], int p, const int L, const int R)
{
    // If it touches leafs
    if(L == R)
        segTree[p] = val[L];
    else
    {
        int mid = (L+R) / 2, lCh = p*2, rCh = lCh+1;

        buildSegTree(segTree, val, lCh, L, mid);    // Build left subtree [L, mid]
        buildSegTree(segTree, val, rCh, mid+1, R);  // Build right subtree [mid+1, R]

        segTree[p] = max(segTree[lCh], segTree[rCh]);
    }
}
void createSegTree(int segTree[], const int size, int val[])
{
    memset(segTree, -1, 4 * size * sizeof(int)); // clean
    buildSegTree(segTree, val, 1, 0, size-1);
}
int querySegTree(int segTree[], int p, int L, int R, int quL, int quR)
{
    int mid = (L+R)/2, ans = INT_MIN;

    if(L >= quL && R <= quR) // L, R are wrapped by quL, qyR
        return segTree[p];

    if(quL <= mid) // Left subtree
    {
        int tmp = querySegTree(segTree, 2*p, L, mid, quL, quR);
        ans = max(ans, tmp);
    }

    if(quR > mid) // Right subtree
    {
        int tmp = querySegTree(segTree, 2*p+1, mid+1, R, quL, quR);
        ans = max(ans, tmp);
    }

    return ans;
}
```

2.1.2 sparse table

```cpp
// Sparse Table (1-index)
int N = 14, logN = __lg(N), spI = logN+1;
int sp[spI][N] = {0};

void buildST()
{
    // Build the Sparse Table
    for(int i = 0; i < N; i++) // first row (only one in a group)
        sp[0][i] = value[i];
    for(int i = 1; i < spI; i++) // number of elements in a group = 2^i
    {
        for(int j = 0; j < N - ((1 << i) - 1); j++) // j < N - (2^i - 1)
        {
            // Current row overlapped two upper groups in (i-1) row
            sp[i][j] = max(sp[i-1][j], sp[i-1][j+(1 << (i-1))]);
        }
    }
}

// Query
int query(int l, int r)
{
    l--, r--;

    int distance = r - l + 1;
    int targetIdx = l != r ? __lg(distance)-1 : 0;

    return max(sp[targetIdx][l], sp[targetIdx][r - (1<< targetIdx - 1)]);
}
```

3 Ad-hoc

3.1 n 皇后

```cpp
int Queen[37000][14];
int Tmp[14];
int total=0;
int Row[14]={0},Left[27]={0},Right[27]={0};

void N_Queen(int k,int Number){
    int i,j;
    if(k==Number){
        for(j=0;j<Number;j=j+1){
            Queen[total][j]=Tmp[j];
        }
        total=total+1;
        return;
    }
    for(i=0;i<Number;i=i+1){
        int right= k+i;
        int left= k-i+Number-1;
        if( !Row[i] && !Left[left] && !Right[right] ){
            Row[i]=1;
            Left[left]=1;
            Right[right]=1;

            Tmp[k]=i;

            N_Queen(k+1,Number);

            Row[i]=0;
            Left[left]=0;
            Right[right]=0;

        }
    }
}

// 用法
N_Queen(0, num);
```