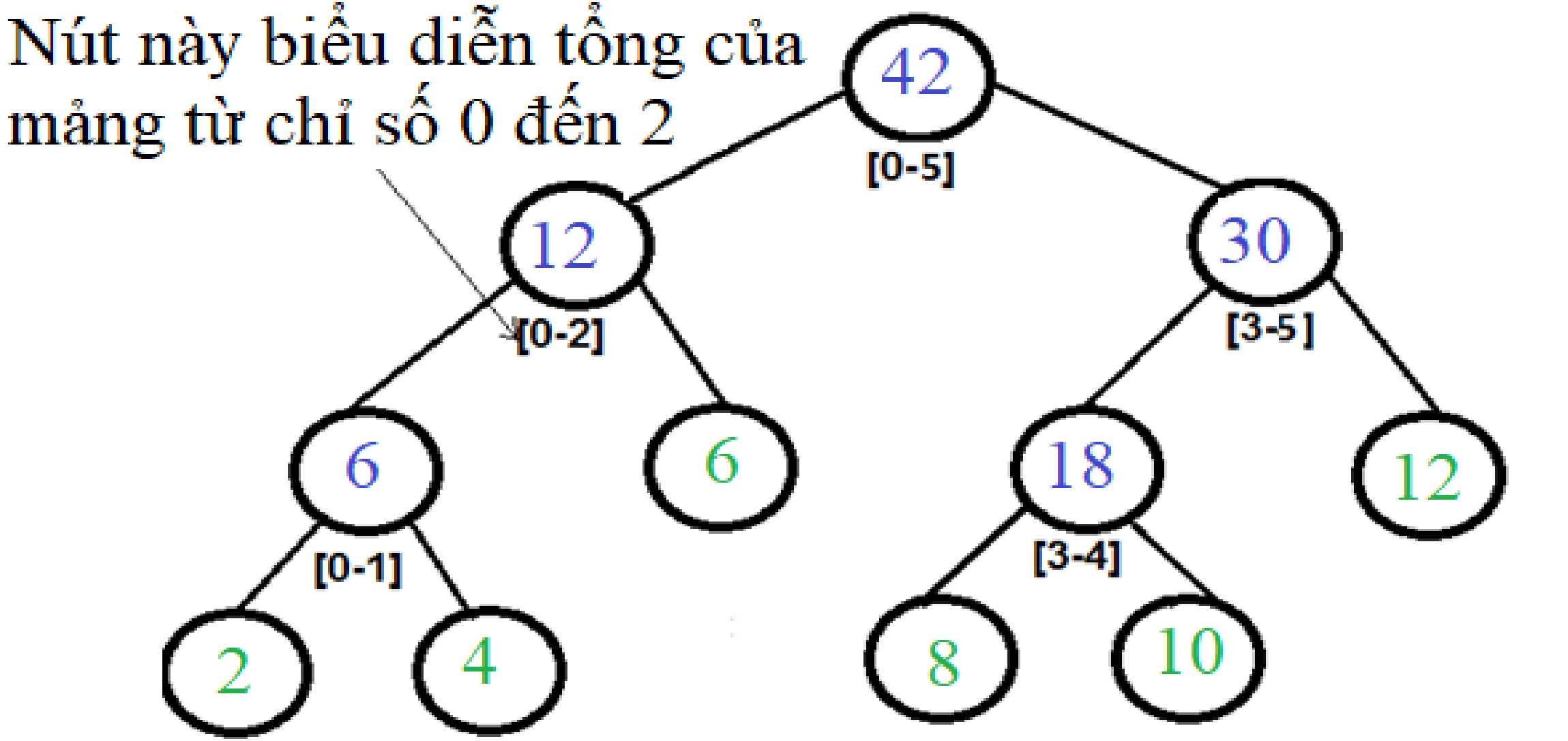
SEGMENT TREE

Tiến sĩ Đào Duy Nam
PTNK – ĐHQG TPHCM

- Segment tree – là cấu trúc dữ liệu cho phép thực hiện hiệu quả(với độ phức tạp $O(\log n)$) các công việc như sau: tìm tổng hoặc phần tử nhỏ nhất của mảng trong đoạn cho trước ($a[l…r]$, trong đó l và r là đầu vào của thuật toán), ở đây có thể có khả năng thêm vào sự thay đổi các phần tử của mảng: như là thay đổi một phần tử, cũng như là thay đổi cả một đoạn con của mảng (có nghĩa là cho phép gán tất cả các phần tử $a[l…r]$ một giá trị nào đó hoặc cộng thêm vào tất cả các phần tử của mảng một số nào đó).
- Điều đặc biệt của segment tree là nó chỉ cần bộ nhớ tuyến tính: một segment tree chuẩn cần $4n$ phần tử để làm việc với mảng kích thước n.

Segment tree trong trường hợp đơn giản

Để xem xét trường hợp đơn giản nhất của segment tree – segment tree cho tổng. Có mảng $a[0..n-1]$ và segment tree phải tìm tổng các phần tử từ l đến r (truy vấn tổng), cũng có thể thay đổi giá trị một phần tử nào đó của mảng như thực hiện phép gán $a[i]=x$ (đây là truy vấn hiệu chỉnh giá trị một phần tử của mảng). Segment tree phải thực hiện các truy vấn này với thời gian $O(\log n)$.



Segment tree cho mảng đầu vào { 2, 4, 6, 8, 10, 12 }

Làm sao biểu diễn segment tree?

1. Các nút lá là các phần tử của mảng ban đầu.

2. Mỗi nút trong biểu diễn một vài kết hợp của các nút lá. Sự kết hợp này có thể khác nhau cho các bài toán khác nhau. Ở đây kết hợp là tổng của các nút lá dưới một nút.

Một mảng biểu diễn cây được sử dụng để biểu diễn segment tree. Với mỗi nút ở chỉ số *i*, con trái ở chỉ số 2**i*+1, con phải ở chỉ số 2**i*+2 và cha ở $\lfloor (i-1)/2 \rfloor$

Segment tree nhìn như thế nào trong bộ nhớ?

Giống như Heap, segment tree cũng được biểu diễn như mảng. Khác ở chỗ, nó là một cây nhị phân đầy đủ mỗi nút là lá hoặc có đúng 2 nút con).

Dưới đây là biểu diễn bộ nhớ của segment tree cho mảng đầu vào

a{2, 4, 6, 8, 10, 12}

st[]={42, 12, 30, 6, 6, 18, 12, 2 , 4, D, D, 8, 10, D, D}

Các giá trị D là phần tử không bao giờ được truy xuất tới.

Tạo segment tree từ mảng cho trước

Bắt đầu với đoạn *a*[0..*n*-1] và mỗi lần chia đôi đoạn hiện tại thành hai nửa (nếu nó chưa trở thành đoạn có độ dài 1) và gọi chính thủ tục đó cho hai nửa và với mỗi đoạn này lưu lại tổng của các nút tương ứng.

Tất cả các tầng của segment tree được tạo ngoại trừ các nút lá. Cây là cây nhị phân đầy đủ vì luôn chia đôi đoạn ở mỗi tầng của cây. Cây được tạo ra luôn là cây nhị phân đầy đủ có *n* nút lá, có *n*-1 nút trong. Tổng các nút là 2**n*-1.

Độ cao của cây segment tree là $\lceil log_2 n \rceil$.

```
int n, st[4*MAXN];
void build (int a[], int v, int l, int r) {
   if (l == r)
      st[v] = a[l];
   else {
      int mid = (l + r) / 2;
      build (a, v*2 + 1, l, mid);
      build (a, v*2 + 2, mid+1, r);
      st[v] = st[v*2 + 1] + st[v*2 + 2];
   }
}
```

Truy vấn cho tổng trong phạm vi cho trước

Một khi cây được tạo, làm sao để lấy được tổng sử dụng segment tree. Sau đây là thuật toán để lấy tổng của các phần tử.

```
int getSum(node, l, r) {
  if nút trong phạm vi l và r
      return giá trị của nút
  else if phạm vi của nút hoàn toàn nằm ngoài l và r
          return 0
    else return getSum(nút con trái, l, r) + getSum(nút con phải, l, r)
}
```

```
int getSum(int v, int tl, int tr, int l, int r)
{
   if (l > r)
      return 0;
   if (l == tl && r == tr)
      return st[v];
   int mid = (tl + tr) / 2;
   return getSum (v*2 + 1, tl, mid, l, r)
      + getSum (v*2 + 2, mid+1, tr, l, r);
}
```

Cập nhật một giá trị

Giống như việc tạo cây và các lệnh truy vấn, việc cập nhật cũng được thực hiện đệ quy. Chúng ta được đưa một chỉ số cần để cập nhật. Cho **new_val** là giá trị cần gán cho phần tử ở chỉ số cần để cập nhật. Chúng ta đi từ gốc của segment tree và gán **new_val** vào tất cả các nút mà có chỉ số trong phạm vi của nó. Nếu một nút không có chỉ số cho trước đó trong phạm vi của nó thì chúng ta không thay đổi gì ở nút đó.

```
void update(int v,int tl,int tr,int pos,int new_val)
{
   if (tl == tr)
      st[v] = new_val;
   else {
      int mid = (tl + tr) / 2;
      if (pos <= mid)
         update (v*2 + 1, tl, mid, pos, new_val);
      else
         update (v*2 + 2, mid+1, tr, pos, new_val);
      st[v] = st[v*2 + 1] + t[v*2 + 2];
   }
}
```

TỔNG ĐOẠN

Cho một dãy số $a_0, a_1, a_2, \ldots, a_{n-2}, a_{n-1}$ là các số nguyên $|a_i| \leq 2.10^9$. Ban đầu tất cả các số có giá trị 0 và trên dãy số có thể thực hiện hai lệnh sau:

Lệnh cập nhật $S(i, k)$:

Gán giá trị k cho phần tử a_i ($0 \leq i \leq n-1; |k| \leq 2.10^9$).

Lệnh truy vấn $Q(i, j)$:

Cho biết tổng của các số $a_i, a_{i+1}, \ldots, a_{j-1}, a_j$ ($0 \leq i \leq j \leq n-1$).

Yêu cầu: Cho một dãy m lệnh thuộc một trong hai loại trên, hãy trả lời tất cả các lệnh truy vấn.

Dữ liệu: Vào từ file văn bản **SUM.INP**

Dòng 1 chứa hai số nguyên dương n, $m \leq 10^5$.

m dòng tiếp theo, mỗi dòng chứa thông tin về một lệnh, đầu tiên là một ký tự $\in \{S, Q\}$. Nếu ký tự đầu dòng là S, tiếp theo là hai số nguyên i, k cho biết đó là lệnh $S(i, k)$. Nếu ký tự đầu dòng là Q, tiếp theo là hai số nguyên i, j cho biết lệnh $Q(i , j)$.

Kết quả: Ghi ra file văn bản **SUM.OUT**

Tương ứng với mỗi lệnh truy vấn Q trong file dữ liệu, ghi ra trên một dòng một số nguyên là trả lời cho truy vấn đó.

Ví dụ:

SUM.INP	SUM.OUT
5 6 S 2 1 S 4 5 Q 2 4 S 3 6 S 2 7 Q 1 4	6 18

```
#include <iostream>
#include <stdio.h>
using namespace std;
const int N = 1e5 + 10;
long long f[4 * N];
char c;
int n, q;
```

```
void update(int ind, int l, int r, int x, int val)
{
    if (l > x || r < x) return;
    if (l == x && r == x)
    {
        f[ind] = val;
        return;
    }
    int mid = (l + r) / 2;
    update(ind * 2 + 1, l, mid, x, val);
    update(ind * 2 + 2, mid + 1, r, x, val);
    f[ind] = f[ind * 2 + 1] + f[ind * 2 + 2];
}
```

```
long long get(int ind, int l, int r, int x, int y)
{
    if (l > y || r < x) return 0;
    if (l >= x && r <= y) return f[ind];
    int mid = (l + r) / 2;
    return get(ind * 2 + 1, l, mid, x, y)
                + get(ind * 2 + 2, mid + 1, r, x, y);
}
```

```
int main()
{   freopen("Sum.inp", "r", stdin);
    freopen("Sum.out", "w", stdout); ios_base::sync_with_stdio(0);
    cin >> n >> q;
    while (q--)
    {   cin >> c;
        if (c == 'S')
        {   int x, val;
            cin >> x >> val;
            update(0, 0, n-1, x, val);
        }
```

```
        else
        {   int x, y;
            cin >> x >> y;
            cout << get(0, 0, n-1, x, y) << '\n';
        }
    }    return 0;
}
```