Taller ProgComp: Track EDD

Sparse Table

Gabriel Carmona Tabja

Universidad Técnica Federico Santa María,
Università di Pisa

June 3, 2024

Part I

Range Query

Range Minimum Query

Definición

Dado un arreglo a de tamaño n, determinar cuál es el valor mínimo en un rango i y j.

Range Minimum Query

Definición

Dado un arreglo *a* de tamaño *n*, determinar cuál es el valor mínimo en un rango *i* y *j*.

Opciones

- Sqrt Decomp: query $O(\sqrt{n})$
- Segment Tree: query $O(\log n)$
- Fenwick Tree: query $O(\log n)$

Part II

Sparse Table

Sparse Table

Características

- Estructura que permite responder range queries

SPARSE TABLE

Características

- Estructura que permite responder range queries
- Mayoría de tipos queries en $O(\log n)$

Sparse Table

Características

- Estructura que permite responder range queries
- Mayoría de tipos queries en $O(\log n)$
- Pero, mínimo/máximo en $O(1)$

SPARSE TABLE

Características

- Estructura que permite responder range queries
- Mayoría de tipos queries en $O(\log n)$
- Pero, mínimo/máximo en $O(1)$

Intuición

Todo número x no negativo, se puede escribir como suma de potencias de dos.

- Ejemplo: $13 = 1101_2 = 8 + 4 + 1$

Sparse Table

Características

- Estructura que permite responder range queries
- Mayoría de tipos queries en $O(\log n)$
- Pero, mínimo/máximo en $O(1)$

Intuición

Todo número x no negativo, se puede escribir como suma de potencias de dos.

- Ejemplo: $13 = 1101_2 = 8 + 4 + 1$

Ahora, todo intervalo se puede separar en la unión de intervalo con largos que decresca en potencias de dos.

- Ejemplo: $[2, 14] = [2, 9] \cup [10, 13] \cup [14, 14]$

Construcción

Idea

Usar un arreglo 2D, llamado `table` de tamaño $K \times MAXN$:

- K, tal que $K \geq \log_2 MAXN$
- En `table[i][j]` se almacenará la respuesta en el rango $[j, j + 2^i - 1]$

Construcción

Idea

Usar un arreglo 2D, llamado `table` de tamaño $K \times MAXN$:

- K, tal que $K \geq \log_2 MAXN$
- En `table[i][j]` se almacenará la respuesta en el rango $[j, j + 2^i - 1]$

```
for (int i = 1; i <= K; i++)
    for (int j = 0; j + (1 << i) <= N; j++)
        table[i][j] = f(table[i - 1][j], table[i - 1][j + (1 << (i - 1))]);
```

Construcción

Idea

Usar un arreglo 2D, llamado `table` de tamaño $K \times MAXN$:

- K, tal que $K \geq \log_2 MAXN$
- En `table[i][j]` se almacenará la respuesta en el rango $[j, j + 2^i - 1]$

```
for (int i = 1; i <= K; i++)
    for (int j = 0; j + (1 << i) <= N; j++)
        table[i][j] = f(table[i - 1][j], table[i - 1][j + (1 << (i - 1))]);
```

¿Cuál es la complejidad?

Construcción

Idea

Usar un arreglo 2D, llamado `table` de tamaño $K \times MAXN$:

- K, tal que $K \geq \log_2 MAXN$
- En `table[i][j]` se almacenará la respuesta en el rango $[j, j + 2^i - 1]$

```
for (int i = 1; i <= K; i++)
    for (int j = 0; j + (1 << i) <= N; j++)
        table[i][j] = f(table[i - 1][j], table[i - 1][j + (1 << (i - 1))]);
```

¿Cuál es la complejidad? $O(K \log n \cdot f)$

Range Sum Queries

```
// Construccion

for (int i = 1; i <= K; i++)
    for (int j = 0; j + (1 << i) <= N; j++)
        table[i][j] = table[i - 1][j] + table[i - 1][j + (1 << (i - 1))]);

// Resolver query
long long sum = 0;
for (int i = K; i >= 0; i--) {
    if ((1 << i) <= R - L + 1) {
        sum += st[i][L];
        L += 1 << i;
    }
}
```

¿Cuál es la complejidad?

Range Sum Queries

```
// Construccion

for (int i = 1; i <= K; i++)
    for (int j = 0; j + (1 << i) <= N; j++)
        table[i][j] = table[i - 1][j] + table[i - 1][j + (1 << (i - 1))]);

// Resolver query
long long sum = 0;
for (int i = K; i >= 0; i--) {
    if ((1 << i) <= R - L + 1) {
        sum += st[i][L];
        L += 1 << i;
    }
}
```

¿Cuál es la complejidad? $O(\log n)$

Range Min Queries

Truco

Ahora dividimos el rango en 2 partes en las cuales se produce overlap.

- Ejemplo: $[1, 6]$ lo dividimos en $[1, 4]$ y $[3, 6]$

Range Min Queries

Truco

Ahora dividimos el rango en 2 partes en las cuales se produce overlap.

- Ejemplo: $[1, 6]$ lo dividimos en $[1, 4]$ y $[3, 6]$

```
// Construccion
for (int i = 1; i <= K; i++)
    for (int j = 0; j + (1 << i) <= N; j++)
        table[i][j] = min(table[i - 1][j], table[i - 1][j + (1 << (i - 1))]);

// Query
int i = lg[R - L + 1];
int minimum = min(table[i][L], table[i][R - (1 << i) + 1]);
```

¿Cuál es la complejidad?

Truco

Ahora dividimos el rango en 2 partes en las cuales se produce overlap.

- Ejemplo: $[1, 6]$ lo dividimos en $[1, 4]$ y $[3, 6]$

```
// Construccion
for (int i = 1; i <= K; i++)
    for (int j = 0; j + (1 << i) <= N; j++)
        table[i][j] = min(table[i - 1][j], table[i - 1][j + (1 << (i - 1))]);

// Query
int i = lg[R - L + 1];
int minimum = min(table[i][L], table[i][R - (1 << i) + 1]);
```

¿Cuál es la complejidad? $O(1)$

References I