TALLER PROGCOMP: TRACK EDD

SQRT DECOMPOSITION

Gabriel Carmona Tabja

Universidad Técnica Federico Santa María,
Università di Pisa

April 15, 2024

Part I

RANGE QUERY CON UPDATE

Prefix Sum con update

Problema

Dado un arreglo de n números y q queries, existen dos tipos de queries:

- Determinar la suma entre el rango i y j ($1 \leq i \leq j \leq n$)
- Actualizar el valor en la posición i por k

PREFIX SUM CON UPDATE

Problema

Dado un arreglo de n números y q queries, existen dos tipos de queries:

- Determinar la suma entre el rango i y j ($1 \leq i \leq j \leq n$)
- Actualizar el valor en la posición i por k

Usando Prefix Sum:

- Primera query $O(1)$
- Segunda query ????:

PREFIX SUM CON UPDATE

Problema

Dado un arreglo de n números y q queries, existen dos tipos de queries:

- Determinar la suma entre el rango i y j ($1 \leq i \leq j \leq n$)
- Actualizar el valor en la posición i por k

Usando Prefix Sum:

- Primera query $O(1)$
- Segunda query ????:
  - $O(n)$, por cada update deberiamos precomputar el prefix sum desde la posición i hasta n.
- Complejidad Peor Caso: $O(q \cdot n)$.

Prefix Sum con update

Problema

Dado un arreglo de n números y q queries, existen dos tipos de queries:

- Determinar la suma entre el rango i y j ($1 \leq i \leq j \leq n$)
- Actualizar el valor en la posición i por k

Usando Prefix Sum:

- Primera query $O(1)$
- Segunda query ????:
  - $O(n)$, por cada update deberiamos precomputar el prefix sum desde la posición i hasta n.
- Complejidad Peor Caso: $O(q \cdot n)$.

¿Podremos hacerlo mejor?

Part II

SQRT Decomposition

Idea

Dividamos el arreglo en bloques de tamaño $\sqrt{n}$ y precomputemos la respuesta de cada bloque

Arreglo	7	3	6	2	4	1	10	5	3
Bloques	16			7			18		

Idea

Dividamos el arreglo en bloques de tamaño $\sqrt{n}$ y precomputemos la respuesta de cada bloque

Arreglo	7	3	6	2	4	1	10	5	3
Bloques	16			7			18		

¿Cómo calculamos la respuesta en un rango i y j?

Idea

Dividamos el arreglo en bloques de tamaño $\sqrt{n}$ y precomputemos la respuesta de cada bloque

Arreglo	7	3	6	2	4	1	10	5	3
Bloques	16			7			18		

¿Cómo calculamos la respuesta en un rango i y j?

Teorema

Cualquier rango i y j, esta compuesto por la unión de bloques completos y a los más dos bloques parciales (uno a cada extremo)

Ejemplo

Determinar la suma entre $2, 7$:

Arreglo	7	3	6	2	4	1	10	5	3
Bloques	16			7			18		

Complejidad de una query

- Bloques tamaño $\sqrt{n}$
- Arreglo dividido en $\frac{n}{\sqrt{n}} = \sqrt{n}$ bloques

Complejidad de una query

- Bloques tamaño $\sqrt{n}$
- Arreglo dividido en $\frac{n}{\sqrt{n}} = \sqrt{n}$ bloques

Para determinar la suma debemos:

- Sumar todos los bloques completos: peor caso $O(\sqrt{n})$
- Sumar a los más dos bloques parciales: peor caso recorremos casi todo el bloque $O(\sqrt{n})$
- Peor caso final: $O(\sqrt{n})$

ACTUALIZAR

¿Es necesario precomputar todo de nuevo?

ACTUALIZAR

¿Es necesario precomputar todo de nuevo? No! Solo el bloque en que el indice se encuentra.

ACTUALIZAR

¿Es necesario precomputar todo de nuevo? No! Solo el bloque en que el indice se encuentra.
Actualicemos la posición 5, por el valor 6

Arreglo	7	3	6	2	6	1	10	5	3
Bloques	16			9			18		

¿Cuál sería la complejidad?

ACTUALIZAR

¿Es necesario precomputar todo de nuevo? No! Solo el bloque en que el indice se encuentra.
Actualicemos la posición 5, por el valor 6

Arreglo	7	3	6	2	6	1	10	5	3
Bloques	16			9			18		

¿Cuál sería la complejidad?

- Actualizar la posición en el arreglo: $O(1)$
- Calcular nuevamente el bloque: $O(\sqrt{n})$
- Peor caso final: $O(\sqrt{n})$

Código

```cpp
struct SqrtDecomp {
  int n;
  int block_size
  vector< int > a;
  vector< int > blocks;
  int block(int i) { return i / block_size; }
  SqrtDecomp(vector< int > &v, int _block_size = -1) {
    n = v.size();
    a = v;
    block_size = (_block_size == -1 ? ceil(sqrt(v.size())) : _block_size);
    blocks.assign(block_size);
    for(int i = 0; i < n; i++) blocks[block(i)] += a[i];
  }
  int query(int l, int r) {
    int sum = 0;
    int i = l;
    while(i <= r) {
      if(i % block_size == 0 && i + block_size - 1 <= r) {
        sum += blocks[block(i)];
        i += block_size;
      } else {
        sum += a[i];
        i++;
      }
    }
    return sum;
  }
  void update(int i, int k) {// siguiente diapositiva }
};
```

Código

```
struct SqrtDecomp {
  void update(int i, int k) {
    a[i] = k;
    int begin_block = i - (i % block_size);
    int end_block = begin_block + block_size;
    blocks[block(i)] = 0
    for(int i = begin_block; i < end_block; i++)
      blocks[block(i)] += a[i];
  }
};
```

REFERENCES I