TALLER PROGCOMP: TRACK EDD

SEGMENT TREE

Gabriel Carmona Tabja

Universidad Técnica Federico Santa María,
Università di Pisa

April 29, 2024

Part I

Range Query con Update

RMQ - Range Minimum Query

Contextualización

Tenemos un arreglo de tamaño N, este arreglo posee números. Dentro de este programa existen dos tipos de consultas:

1. Determinar el valor mínimo dentro de un rango i y j, donde $i <= j$
2. Actualizar el valor que se encuentra en la posición i

Intento trivial

RMQ - Range Minimum Query

Contextualización

Tenemos un arreglo de tamaño N, este arreglo posee números. Dentro de este programa existen dos tipos de consultas:

1. Determinar el valor mínimo dentro de un rango i y j, donde $i <= j$
2. Actualizar el valor que se encuentra en la posición i

Intento trivial

- Por cada consulta tipo 1: recorremos el arreglo desde i hasta j, $O(n)$
- Por cada consulta tipo 2: actualizamos accediendo al arreglo directamente, o sea $O(1)$

¿Cómo resolverlo?

- SQRT Decomposition: $O(q \cdot \sqrt{n})$

¿Cómo resolverlo?

- SQRT Decomposition: $O(q \cdot \sqrt{n})$

¿Se podrá mejorar?

¿Cómo resolverlo?

- SQRT Decomposition: $O(q \cdot \sqrt{n})$

¿Se podrá mejorar?
¡Sí, con el Segment Tree!

SEGMENT TREE

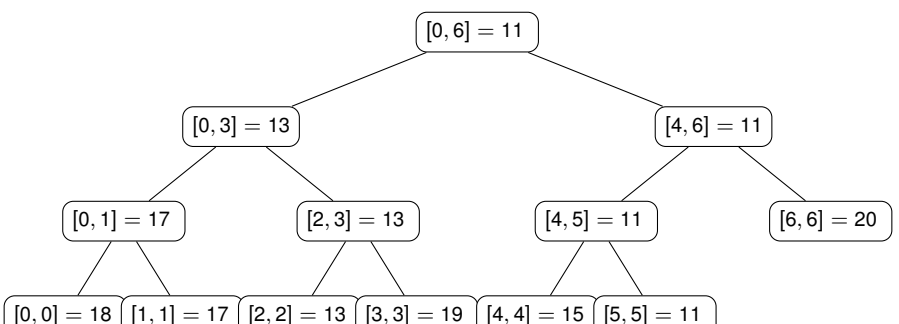
Definición

Estructura de datos basada en árboles binarios:

- Un nodo representará la solución para el intervalo del arreglo i hasta j.
- El hijo izquierdo de ese nodo tendrá la solución para el intervalo i hasta $(j + i)/2$
- El hijo derecho de ese nodo tendrá la solución para el intervalo $(j + i)/2 + 1$ hasta j.

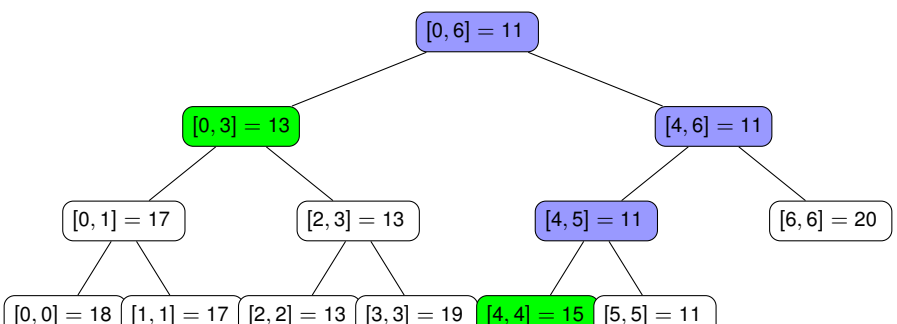
SEGMENT TREE - CÓMO FUNCIONA

Digamos el siguiente arreglo, $A = \{18, 17, 13, 19, 15, 11, 20\}$ y queremos resolver RMQ.



SEGMENT TREE - SOLUCIÓN DE BÚSQUEDA

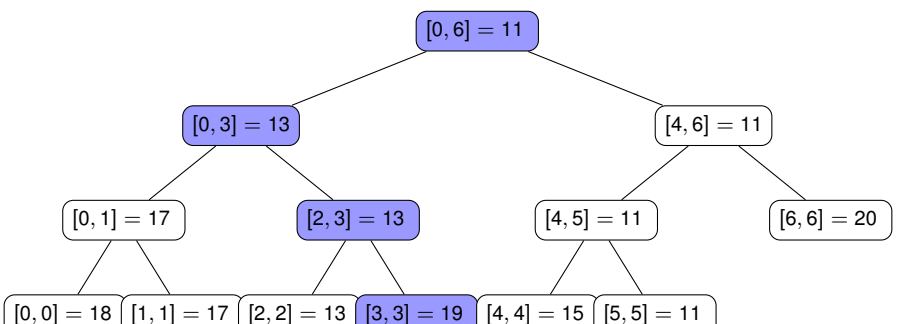
Digamos el siguiente arreglo, $A = \{18, 17, 13, 19, 15, 11, 20\}$ y queremos resolver RMQ. Digamos que preguntamos por el mínimo valor en el intervalo [0, 4].



Luego, se debe comparar entre los nodos verdes para saber la solución final.

Segment Tree - Solución de Actualización

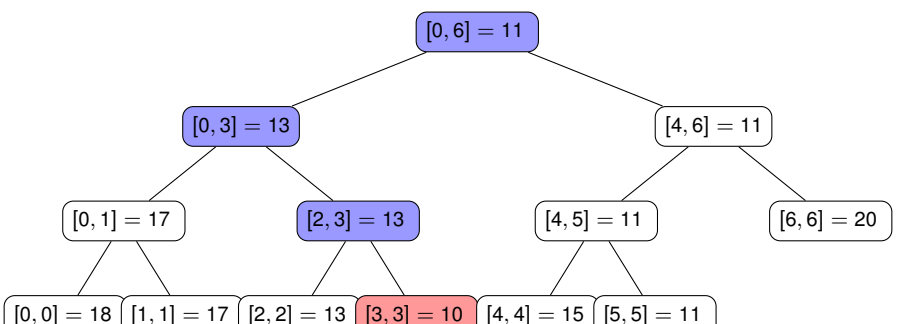
Digamos el siguiente arreglo, $A = \{18, 17, 13, 19, 15, 11, 20\}$ y queremos resolver RMQ. Digamos que actualizamos el valor de la posición 3 por 10.



Al encontrar al nodo correspondiente, ahora debemos actualizarlo y actualizar a sus padres.

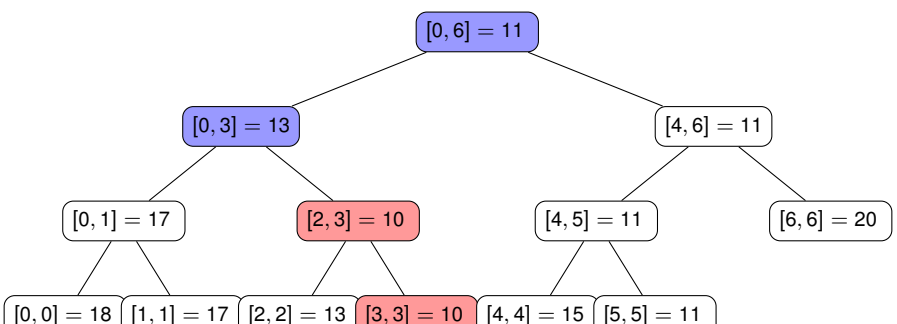
Segment Tree - Solución de Actualización

Digamos el siguiente arreglo, $A = \{18, 17, 13, 19, 15, 11, 20\}$ y queremos resolver RMQ. Digamos que actualizamos el valor de la posición 3 por 10.



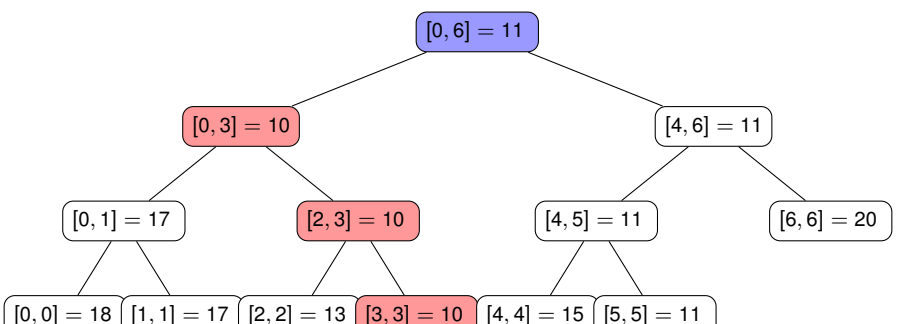
Segment Tree - Solución de Actualización

Digamos el siguiente arreglo, $A = \{18, 17, 13, 19, 15, 11, 20\}$ y queremos resolver RMQ. Digamos que actualizamos el valor de la posición 3 por 10.



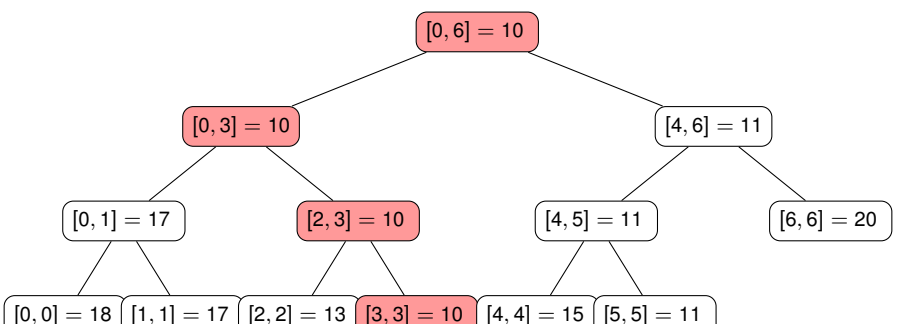
Segment Tree - Solución de Actualización

Digamos el siguiente arreglo, $A = \{18, 17, 13, 19, 15, 11, 20\}$ y queremos resolver RMQ. Digamos que actualizamos el valor de la posición 3 por 10.



Segment Tree - Solución de Actualización

Digamos el siguiente arreglo, $A = \{18, 17, 13, 19, 15, 11, 20\}$ y queremos resolver RMQ. Digamos que actualizamos el valor de la posición 3 por 10.



Segment Tree - Código Struct

```
struct SegmentTree {
  int n; // cantidad de elementos en el arreglo original
  vector<long long> ST; // vector que nos ayudara a crear la estructura
  // puntero funcion que ayuda a resolver el problema
  long long (*merge)(long long, long long);

  void build(int index, int l, int r, vector<long long> &values){
    // funcion que construye el segment tree
  }
  SegmentTree(vector<long long> &values, long long (*merge_)(long long a, long long b)){
    // constructor del segment tree
  }
  long long query(int i, int j){
    // recibe un i y j correspondientes al intervalo consultado
  }
  long long query(int l, int r, int index, int i, int j){
    // la real funcion que responde la consulta
  }
  void update(int pos, long long val){
    // recibe una posicion y actualiza el valor en esa posicion
  }
  void update(int l, int r, int index, int pos, long long val){
    // la real funcion que actualiza el valor dada una posicion
  }
};
```

Segment Tree - Función Build y Constructor

```cpp
void build(int index, int l, int r, vector<long long> &values){
  // funcion que construye el segment tree
  if(l == r){
    ST[index] = values[l];
  } else {
    // hijo izquierdo
    build(index * 2, l, (r + l) / 2, values);
    // build derecho
    build(index * 2 + 1, (r + l) / 2 + 1, r, values);
    // padre = hijo izq + hijo der
    ST[index] = merge(ST[index * 2],ST[index * 2 + 1]);
  }
}

SegmentTree(vector<long long> &values, long long (*merge_)(long long a, long long b)){
  // constructor del segment tree
  merge = merge_;
  n = values.size();
  ST.resize(4 * n + 3);
  build(1, 0, n - 1, values);
}
```

Segment Tree - Función Query

```
long long query(int i, int j){
  // recibe un i y j correspondientes al intervalo consultado
  return query(0, n - 1, 1, i, j);
}

long long query(int l, int r, int index, int i, int j){
  // la real funcion que responde la consulta
  if(l >= i and r <= j) return ST[index];

  int mid = (r + l) / 2;
  // solo consulta al lado derecho
  if(mid < i){
    return query(mid + 1, r, index * 2 + 1, i, j);
  // solo consulta al lado izquierdo
  } else if(mid >= j){
    return query(l, mid, index * 2, i, j);
  } else {
  // consulta hacia ambos hijos
    return merge(query(l, mid, index * 2, i, j),
                 query(mid + 1, r, index * 2 + 1, i, j));
  }
}
```

Segment Tree - Función Update

```
// update
void update(int pos, long long val){
  // recibe una posicion y actualiza el valor en esa posicion
  update(0, n - 1, 1, pos, val);
}

void update(int l, int r, int index, int pos, long long val){
  // la real funcion que actualiza el valor dada una posicion
  if(r < pos || pos < l) return;
  if(l == r){
    ST[index] = val;
  } else {
    int mid = (r + l) / 2;
    // buscar posicion en arbol izquierdo
    update(l, mid, index * 2, pos, val);
    // buscar posicion en arbol derecho
    update(mid + 1, r, index * 2 + 1, pos, val);

    ST[index] = merge(ST[index * 2], ST[index * 2 + 1]);
  }
}
```

SEGMENT TREE - CÓMO UTILIZARLO

```cpp
// asumiendo que esta todo el codigo del Segment Tree aqui

// la funcion que vamos a utilizar para responder nuestra pregunta
long long mininum(long long a, long long b){ return min(a,b); }

int main(){
  int n; cin >> n;
  vector<ll> nums(n);
  for (int i = 0; i < n; ++i) cin >> nums[i];

  SegmentTree arbol(nums, minimum);

  int q; cin >> q;
  while(q--){
    int tipo; cin >> tipo;
    if(tipo == 1){ //query
      int i, j; cin >> i >> j;
      cout << arbol.query(i, j) << "\n";
    } else if(tipo == 2){ //update
      int pos, val; cin >> pos >> val;
      arbol.update(pos, val);
    }
  }
  return 0;
}
```

REFERENCES I