GUÍA DE APRENDIZAJE PARA CONCURSANTES ICPC Y IOI: TABLA DISPERSA (*SPARSE TABLE*)

1. Introducción

Supongamos que tenemos un arreglo de N elementos ($1 \leq N \leq 10^5$) $m_0, m_1, \ldots, m_{N-1}$ que no se puede modificar y una operación asociativa ϕ y queremos responder preguntas de la forma: ¿Cuál es el valor de $\Theta[i,j] = m_i \phi m_{i+1} \phi \ldots \phi m_j$?

Para resolver esta problemática se puede emplear una estructura de datos llamada Tabla dispersa (*Sparse Table*) a la cual le dedicaremos esta guía de aprendizaje

2. Conocimientos previos

2.1. Operación asociativa

La operación asociativa es una propiedad de las operaciones matemáticas que establece que el resultado de la operación no cambia, independientemente de cómo se agrupen los elementos. En otras palabras, si se tienen tres elementos a, b y c, la operación asociativa establece que $(a \phi b) \phi c = a \phi (b \phi c)$, donde ϕ representa cualquier operación matemática como la suma, la resta, la multiplicación o la división. Esta propiedad es fundamental en matemáticas y es utilizada en numerosos contextos para simplificar cálculos y demostrar teoremas.

2.2. Funciones idempotentes

Las funciones idempotentes son aquellas que, al aplicarse múltiples veces a un mismo valor de entrada, producen el mismo resultado que si se aplicaran una sola vez. En otras palabras, una función es idempotente si $f(f(x)) = f(x)$ para todo x en el dominio de la función. Este concepto es común en matemáticas, informática y otros campos técnicos.

3. Desarrollo

El algoritmo ingenuo recorre todos los elementos en $[i,j]$, y requiere O(N) en el peor caso. Ahora si nos hicieran M preguntas tendríamos un algoritmo con una complejidad de O(MN) lo cual no sería factible si la cantidad de preguntas ronda las 10^5. Ahora hagamos un breve análisis:

Si los elementos del arreglo no pueden cambiar de valor, podemos preprocesar los datos para acelerar el cálculo de las respuestas. Hay N^2 preguntas posibles, y podemos precalcular todas sus respuestas en N^2 usando programación dinámica. No hace falta guardar todas las respuestas: basta con precalcular una cantidad suficiente como para poder responder cualquier pregunta.

Todo número entero no negativo puede ser representado únicamente como una suma de decrecientes potencias de 2. Esto es precisamente la representación binaria de un número. Ej. $13 = (1101)_2 = 8 + 4 + 1$. Para un número x habrá cuando más $\lceil \log_2 x \rceil$ sumandos.

Por este mismo razonamiento cualquier intervalo puede ser únicamente representado como una unión de intervalos de longitud igual a decrecientes potencias de 2. Por ejemplo $[2, 14] = [2, 9] \cup [10, 13] \cup [14, 14]$ donde la longitud del intervalo completo es 13 y donde los intervalos individuales tienen de longitud 8, 4 y 1 respectivamente, coincidiendo con la representación binaria

de 13 que es la longitud del intervalo completo. Aquí también el intervalo principal podrá ser representado como máximo por $\lceil \log_2(\text{longitud del intervalo}) \rceil$ intervalos individuales.

La idea detrás de la Tabla dispersa es precalcular todas las respuestas para consultas de intervalos con longitud igual a una potencia de 2. Luego para responder a la consulta de un intervalo de longitud cualquiera se divide éste en la unión de intervalos de longitud igual a potencias de 2, donde conociendo las respuestas para estos últimos (pues lo precalculamos inicialmente), se combinan dichas respuestas y se obtiene el valor correspondiente al intervalo principal.

3.1. Precálculo

Usaremos una matriz bidimensional para almacenar las respuestas a las consultas precalculadas st$[i][j]$ almacenará la respuesta para el rango $[j, j+2^i-1]$ de longitud 2^i. El tamaño de la matriz bidimensional será $(K+1) \times MAXN$, dónde $MAXN$ es la mayor longitud de matriz posible. K tiene que satisfacer K $\geq \lfloor \log_2 \text{MAXN} \rfloor$ porque $2^{\lfloor \log_2 \text{MAXN} \rfloor}$ es la potencia más grande de dos rangos, que tenemos que soportar. Para arreglos con una longitud razonable ($\leq 10^7$ elementos), $K = 25$ es un buen valor. El MAXN es la segunda dimensión para permitir accesos consecutivos a la memoria (compatibles con la caché).

Porque el rango $[j, j+2^i-1]$ de longitud 2^i se divide muy bien en los rangos $[j, j+2^{i-1}-1]$ y $[j+2^{i-1}, j+2^i-1]$, ambos de longitud 2^{i-1}, podemos generar la tabla de manera eficiente usando programación dinámica.

3.2. Consultas de suma en rangos

Para este tipo de consultas, nosotros necesitamos hallar la suma de todos los valores en un rango. La operación asociativa empleada sería la suma. Para este tipo de consultas, queremos encontrar la suma de todos los valores en un rango. Por lo tanto la definición natural de la función f es $f(x, y) = x + y$.

Para responder a la consulta de suma para el rango $[L, R]$, iteramos sobre todas las potencias de dos, comenzando por la más grande. Tan pronto como una potencia de dos 2^i es menor o igual a la longitud del rango ($I = R - L + 1$) , procesamos la primera parte del rango $[L, L + 2^i - 1]$, y y continuar con el rango restante $[L + 2^i, R]$. Nosotros simplemente hallamos la longitud del intervalo $I = R - L + 1$ y descomponemos dicho intervalo en intervalos de longitud igual a una potencia de 2, que como vimos al inicio coincide con los bits activos de la representación binaria del número I.

Donde I en su representación binaria tenga el bit k activo, el subintervalo a tener en cuenta es $st[k][L]$. Luego aumentamos el extremo derecho de L en 2^k para continuar el procesamiento ahora en el intervalo $[L + 2^k, R]$. Continuar el procesamiento significa hallar los restantes bits activos de I y desplazar el extremo derecho del intervalo en 2^k posiciones, siendo k el bit activo que se encontró en I. El resultado a devolver en el método, la variable r, se inicializa con el elemento neutro de la operación asociativa empleada, pues cada vez que se encuentre un bit activo en I, r se operará con $st[k][L]$. Para chequear si I tiene el bit k activo se emplea la operación de bits $I^{\&}(1 \acute{} k)$.

Para desplazar el extremo izquierdo del intervalo en 2^k posiciones, se emplea la operación de bits $L+ = (1\acute{}k)$.

3.3. Consultas de mínimo en rangos

Al calcular el mínimo de un rango, no importa si procesamos un valor en el rango una o dos veces. Por lo tanto, en lugar de dividir un rango en varios rangos, también podemos dividir el rango en solo dos rangos superpuestos con una potencia de dos longitudes. Por ejemplo, podemos dividir el rango $[1, 6]$ en los rangos $[1, 4]$ y $[3, 6]$. El rango mínimo de $[1, 6]$ es claramente el mismo que el mínimo del rango mínimo de $[1, 4]$ y el rango mínimo de $[3, 6]$. Entonces podemos calcular el mínimo del rango $[L, R]$ con:

$$\text{mín}(\text{st}[i][L], \text{st}[i][R - 2^i + 1]) \quad \text{donde } i = \log_2(R - L + 1)$$

Esto requiere que seamos capaces de calcular $\log_2(R - L + 1)$ rápido. Puedes lograrlo precalculando todos los logaritmos o alternativamente, se puede calcular sobre la marcha en espacio y tiempo constantes.

4. Implementación

4.1. C++

```
int st[K + 1][MAX_N],arrays [MAX_N],N;

//Se descomenta en funcion de la operacion
int  f(int a, int b){
  //return a+b;
  //return min(a,b);
}

void init(){
   copy(arrays, arrays+MAX_N, st[0]);
   for (int i = 1; i <= K; i++)
      for (int j = 0; j + (1 << i) <= N; j++)
         st[i][j] = f(st[i - 1][j], st[i - 1][j + (1 << (i - 1))]);
}

int querySum(int L,int R){
   int sum = 0;
   for (int i = K; i >= 0; i--) {
      if ((1 << i) <= R - L + 1) {
         sum += st[i][L];
         L += 1 << i;
      }
   }
   return sum;
```

```
}

int log2_floor(int i) {
   return i ? __builtin_clzll(1) - __builtin_clzll(i) : -1;
}

int queryMin(int L,int R){
   int i = log2_floor(R - L + 1);
   int minimum = min(st[i][L], st[i][R - (1 << i) + 1]);
   return minimum;
}
```

4.2. Java

```
private class SparseTable{
   private long [][] st ;
   private long [] array;
   private int [] lg;
   private final int K = 25;
   private final int MAX_N = 1000001;
   private int N;

   public SparseTable(int N, long [] v){
      st = new long [K][MAX_N];
      array = Arrays.copyOf(v,N);
      lg = new int [MAX_N];
      this.N = N;
      this.init();
   }

   //Se descomenta segun el problema
   private long f(long a,long b){
      //return a + b;
      //return Math.min(a,b);
   }

   public void init(){
      lg[1] = 0;
      for (int i = 2; i < MAX_N; i++) lg[i] = lg[i/2] + 1;
      for(int i=0;i<N;i++) st[0][i] = array[i];
      for (int i = 1; i <= K; i++)
             for (int j = 0; j + (1 << i) <= N; j++)
            st[i][j] = f(st[i - 1][j], st[i - 1][j + (1 << (i - 1))]);
   }

   public int querySum(int L,int R){
      int sum = 0;
      for (int i = K; i >= 0; i--) {
```

```
            if ((1 << i) <= R - L + 1) {
                sum += st[i][L];
                L += 1 << i;
            }
        }
        return sum;
    }

    public long queryMin(int L,int R){
        int i = lg[R - L + 1];
        long minimum = Math.min(st[i][L], st[i][R - (1 << i) + 1]);
        return minimum;
    }
}
```

5. Aplicaciones

Entre las aplicaciones de la tabla dispersa podemos citar:

1. **Consultas de mínimo/máximo en un rango de datos estáticos:** La tabla dispersa puede utilizarse para encontrar rápidamente el mínimo o máximo en un rango específico de datos estáticos, lo que es útil en problemas de algoritmos competitivos y en aplicaciones de procesamiento de datos.
2. **Consultas de suma en un rango de datos estáticos:** También se puede utilizar la tabla dispersa para realizar consultas de suma en un rango de datos estáticos, lo que es útil en problemas que requieren el cálculo rápido de sumas acumulativas en un rango.
3. **Consultas de operaciones binarias en un rango de datos estáticos:** La tabla dispersa puede ser utilizada para realizar consultas de operaciones binarias, como operaciones AND, OR, XOR, etc., en un rango específico de datos estáticos.
4. **Búsqueda de puntos de inflexión o cambios en un rango de datos:** La tabla dispersa puede ser utilizada para encontrar rápidamente puntos de inflexión o cambios en un rango específico de datos estáticos, lo que es útil en problemas que requieren la identificación de cambios significativos en los datos.
5. **Consultas de funciones matemáticas en un rango de datos estáticos:** También se puede utilizar la tabla dispersa para realizar consultas de funciones matemáticas, como la mediana, la media, la desviación estándar, etc., en un rango específico de datos estáticos.

En resumen, la tabla dispersa es una estructura de datos útil para realizar consultas eficientes en rangos de datos estáticos y se utiliza en una variedad de aplicaciones, incluyendo algoritmos competitivos, procesamiento de datos, análisis de datos y más. El único inconveniente de esta estructura de datos es que sólo se puede utilizar en matrices inmutables . Esto significa que el arreglo no se puede cambiar entre dos consultas. Si algún elemento de del arreglo cambia, se debe volver a calcular la estructura de datos completa.

6. Complejidad

La tabla dispersa es una estructura de datos que permite responder consultas de rango. Puede responder a la mayoría de las consultas de rango en O($\log N$) , pero su verdadero poder es responder consultas de rango mínimo (o consultas de rango máximo equivalente). Para esas consultas puede calcular la respuesta en O(1) tiempo. Mientras tanto su operación de precalculo tiene una complejidad de O(N $\log$ N). En el caso de la operación de suma en el rango la complejidad es $O(K) = O(\log \text{MAXN})$

Una de las principales debilidades del O(1) de la consulta del mínimo es que este enfoque solo admite consultas de funciones idempotentes . Es decir, funciona muy bien para consultas de rango mínimo o máximo, pero no es posible responder consultas de suma de rango utilizando este enfoque.

Existen estructuras de datos similares que pueden manejar cualquier tipo de funciones asociativas y responder consultas de rango en O(1) . Uno de ellos se llama *Disjoint Sparse Table* . Otro sería el *Sqrt Tree* .

7. Ejercicios

A continuación una lista de ejercicios que se pueden resolver aplicando esta estructura de datos:

- CSES - Static Range Minimum Queries
- CSES - Static Range Sum Queries
- CSES - Range Xor Queries
- SPOJ - THRBL
- SPOJ - RMQSQ
- SPOJ - A Famous City
- SPOJ - Negative Score
- SPOJ - Postering
- SPOJ - Miraculous
- SPOJ - Diferencija
- Codeforces - CGCDSSQ
- Codeforces - R2D2 and Droid Army
- Codeforces - Animals and Puzzles
- Codeforces - Trains and Statistics
- Codeforces - Turn off the TV

- Codeforces - Map
- Codeforces - Awards for Contestants
- Codeforces - Longest Regular Bracket Sequence
- Codeforces - Array Stabilization (GCD version)
- Codechef - MSTICK
- Codechef - SEAD
- DevSkill - Multiplication Interval (archived)