Tabela Esparsa

Wladimir Araújo Tavares [1]

[1]Universidade Federal do Ceará - Campus de Quixadá

4 de junho de 2018

1 Tabela Esparsa

Tabela Esparsa

Problema:

- Entrada: Dado um vetor A e conjunto de queries (L, R)
- Saída: Para cada query, compute o valor de $F(A[L], \ldots, A[R])$
- Pré-processamento: $O(N \times log(N))$
- Query : $O(log(N))$

Premissas:

- O vetor A é imutável.
- F é associativa, ou seja, $F(a, b, c) = F(F(a, b), c) = F(a, F(b, c))$
- Funções associativas: min, max, sum, gcd.

Intuição

- Um intervalo (L, R) pode ser representado como a união de intervalo cujo tamanho é uma potências de 2.
- $[2, 14] = [2, 9] \cup [10, 13] \cup [14, 14]$
- O número de intervalos será igual $\lceil log(R - L + 1) \rceil$
- $R - L + 1 = 13 = (1101)_2 = 8 + 4 + 1$
- $[2, 14] = [L, L + 2^3 - 1] = [2, 9]$
- $[10, 14] = [L, L + 2^2 - 1] = [10, 13]$
- $[14, 14]$
- Complexidade query : $O(logN)$

Pré-computação

- Calcule o valor $st[i][j]$ armazena a resposta para a seguinte cálculo $F(A[i], \ldots, F[i, i + 2^j - 1])$ de tamanho 2^j.
- Tamanho da tabela $O(MAXN \times K)$, onde $MAXN$ tamanho do vetor A e $K \geq \lfloor log_2 MAXN \lfloor +1$
- Para $MAXN \leq 10^7$, $K = 25$ é suficiente.

Pré-computação

$$st[i][j] = \begin{cases} F(A[i]) & j = 0 \\ F(st[i][j-1], [i + 2^{j-1}][j-1]) & j > 0 \end{cases} \tag{1}$$

Range Sum Query

```
// build Sparse Table
for(int i = 0; i < n; i++)
  st[i][0] = Arr[i];

for(int j = 1; j <= K; j++) {
  for(int i = 0; i <= n - (1 << j); i++)
    st[i][j]=st[i][j - 1]+st[i + (1 << (j - 1))][j - 1]);
}
```

Range Sum Query

```
long long answer = ~(1<<31);
for(int j = K; j >= 0; j--)
    if(   R-L+1 >= (1 << j) ) {
        answer = answer + st[L][j];
        L += 1 << j;
    }
}
```

Range Minimum Query

- Algumas query podem ser respondidas sem precisar dividir em intervalos disjuntos.
- Por exemplo, o menor valor no intervalo [1,6] pode ser resolvido calculando o menor valor do intervalo [1,4] e [3,6].

$$rmq(L, R) = min(st[L][j], st[R - 2^j + 1][j]), \text{onde } j = log_2(R - L + 1)$$

Range Minimum Query

```
log[1] = 0;
for (int i = 2; i <= MAXN; i++)
      log[i] = log[i/2] + 1;

for(int i = 0; i < n; i++)
  st[i][0] = Arr[i];

for(int j = 1; j <= K; j++) {
  for(int i = 0; i <= n - (1 << j); i++)
    st[i][j] = min(st[i][j - 1],st[i + (1 << (j - 1))][j - 1]);
}
```

Range Minimum Query

```
int minimo(int L, int R){
  if( R >= L){
        int j = logtable[R - L + 1];
        int m = min(stMin[L][j], stMin[R - (1 << j) + 1][j]);
  }else{
        return INT_MAX;
  }
}
```

Problemas

- http://www.spoj.com/problems/RMQSQ/
- http://www.spoj.com/problems/THRBL/
- http://www.spoj.com/problems/RPLN/
- http://www.spoj.com/problems/DIFERENC/
- http://www.spoj.com/problems/CITY2/

CGCDSQ - Couting GCD Segment Query

- Problema: Dado um vetor A de tamanho N. Conte todos os intervalos (L, R) tal que $0 \leq L \leq R < N$ e $gcd(A[L], \ldots, A[R])$

Observações:

- Força-Bruta: $O(N^2 LogN)$
- $rgcd(L, L) \geq rgcd(L, L+1) \geq rgcd(L, L+2) \geq \ldots rgcd(L, N-1)$
- Busca Binária: $O(NlogNlogN)$, $O(logN)$ busca binária

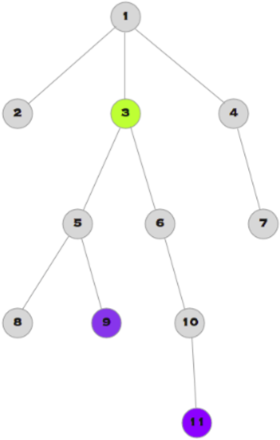
CGCDSQ - Couting GCD Segment Query

- Para cada L fixo, vamos encontrar o menor R tal que $rgcd(L, R) = 1$
- Todos os intervalos $rgcd(L, R') = 1$ para todo $R \geq R' \geq N - 1$. Logo, existem $N - R$ intervalos com a propriedade acima.
- Observe que o intervalo (L, R) também pode ser decomposto em intervalos cujos tamanho são potências de 2.
- Comece com $R = L$, para cada $i = k, \ldots 0$
- Se $gcd(A[L], \ldots, A[R + 2^i]) > 1$ então $R = R + 2^i$.

CGCDSQ - Couting GCD Segment Query

```
long long answer = 0;
for(int i = 0; i < n; i++){
    int R, g;
    R = i;
    g = 0;
    for(int j = K; j >= 0; j--){
        if( R + (1<<j)-1 >= n) continue;
        if( gcd(g, st[R][j]) > 1 ){
            g = gcd(g, st[R][j]);
            R += 1<<j;
        }
    }
    answer += n-R;
}
cout << answer << endl;
```

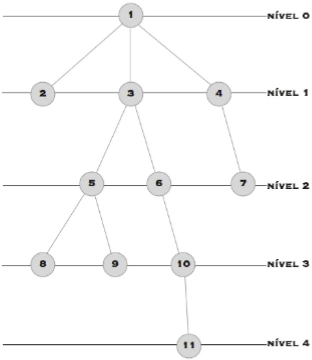
LCA



LCA_DFS

```
int pai[MAXN];
int nivel[MAXN];
int ancestral[MAXN][MAXL];
vector<int> lista[MAXN];
void dfs(int u){
  for(int i = 0;i < (int)lista[u].size();i++){
    int v = lista[u][i];
    if(nivel[v] == -1){
        pai[v] = u;
        nivel[v] = nivel[u]+1;
        dfs(v);
    }
  }
}
```

LCA_DFS



LCA_DFS

```
int LCA(int a, int b){
    while(a != b){
            if(nivel[a] > nivel[b]) a = pai[a];
            else b = pai[b];
    }
    return a;
}
```

intuição

- Não precisamos ficar subindo um nível por vez.
- Vamos subir uma potência de 2.
- $ancestral[i][j]$ armazene 2^j -ésimo ancestral de i.

$$ancestral[i][j] = \begin{cases} pai[i] & j = 0 \\ ancestral[ancestral[i][j-1]][j-1] & j > 0 \end{cases} \tag{2}$$

tabela de ancestral

```
// declarar a tabela
ancestral[MAXN][MAXL]; // MAXL representa log N (com uma margem
// primeiro, inicializamos tudo para -1
for(int i = 0;i < MAXN;i++)
  for(int j = 0;j < MAXL;j++)
    ancestral[i][j] = -1;
// definimos os pais de carda vértice
for(int i = 1;i <= N;i++)
  ancestral[i][0] = pai[i];
// montamos o restante da tabela com programação dinâmica
for(int j = 1;j < MAXL;j++)
  for(int i = 1;i <=   N;i++)
    if(ancestral[i][j-1] != -1)
      ancestral[i][j] = ancestral[ ancestral[i][j-1] ][j-1];
```

tabela de ancestral

```
int LCA(int u, int v){
  if(nivel[u] < nivel[v]) swap(u, v); // isto é para definir u c
  // vamos agora fazer nivel[u] ser
  // igual nivel[v], subindo pelos ancestrais de u
  for(int i = MAXL-1;i >= 0;i--)
    if(nivel[u] - (1<<i) >= nivel[v])
      u = ancestral[u][i];
  // agora, u e v estão no mesmo nível
  if(u == v) return u; // se eles forem o mesmo nó já achamos nc
  // subimos o máximo possível de forma
  // que os dois NÃO passem a ser iguais
  for(int i = MAXL-1;i >= 0;i--)
    if(ancestral[u][i] != -1 && ancestral[u][i] != ancestral[v][
      u = ancestral[u][i];
      v = ancestral[v][i]
    }
  // como subimos o máximo possível
  // sabemos que u != v e que pai[u] == pai[v]
  // logo, LCA(u, v) == pai[u] == pai[v]
  return ancestral[u][0];
}
```