구간 최장 증가 부분 수열 쿼리 (Part 2)

Chapter 4. ⊡ 연산자의 빠른 구현

현재 우리의 알고리즘이 $O(N^5)$ 인 이유는 다음과 같다:

- ⊡ 연산자가 $O(N^3)$ 에 구현됨
- ⊡ 연산자를 $O(N^2)$ 번 호출함

잠시 ⊡ 연산자의 실제 이름을 짚고 넘어가자면, 논문에서는 위 연산자가 *unit-Monge matrix-matrix distance multiplication* 라는 이름으로 소개되었다. Part 1에서는 괜히 글이 어렵다는 인상을 줄 것 같아서 의도적으로 언급하지 않은 이름이다. 이제부터는 **(순열의) unit-Monge 곱** 이라고 부르거나 그냥 앞과 같이 ⊡ 이라고 부른다.

⊡ 연산자를 어떻게 $O(N \log N)$ 에 계산하는지 살펴보자. 행렬 $\Sigma(A), \Sigma(B)$ 에 대해 다음과 같은 분할 $\Sigma(A) = [\Sigma(A)_{lo}, \Sigma(A)_{hi}], \Sigma(B) = \begin{bmatrix}\Sigma(B)_{lo} \\ \Sigma(B)_{hi}\end{bmatrix}$을 생각해 보자. lo 는 앞 $N/2+1$ 개 항, hi 는 뒤 $N/2$ 개 항을 나타내며. 설명의 편의를 위해 N은 짝수라고 가정한다.

$\Sigma(A) \odot \Sigma(B)$ 는 $\Sigma(A)_{lo} \odot \Sigma(B)_{lo}$ 와 $\Sigma(A)_{hi} \odot \Sigma(B)_{hi}$의 element-wise minimum일 것이다. 또한, $\Sigma(\{A, B\})_{lo,hi}$ 는 *대략* $\Sigma(\{A, B\}_{lo,hi})$ 와 유사한 꼴을 띌 것인데, 이 때

- A_{lo} 는 A 의 부분 순열로, 값이 $[1, N/2]$ 범위에 있는 원소로 구성
- A_{hi} 는 A 의 부분 순열로, 값이 $[N/2+1, N]$ 범위에 있는 원소로 구성
- B_{lo} 는 B 의 부분 순열로, 인덱스가 $[1, N/2]$ 범위에 있는 원소로 구성
- B_{hi} 는 B 의 부분 순열로, 인덱스가 $[N/2+1, N]$ 범위에 있는 원소로 구성
- $M_{lo}(i,k) = \min_{j=1}^{N/2+1}(\Sigma(A)(i,j) + \Sigma(B)(j,k))$
- $M_{hi}(i,k) = \min_{j=N/2+2}^{N+1}(\Sigma(A)(i,j) + \Sigma(B)(j,k))$

라고 하면, 위 관찰대로 $\Sigma(C)(i,k) = \min(M_{lo}(i,k), M_{hi}(i,k))$ 이다.

$A_{lo} \boxdot B_{lo}$, $A_{hi} \boxdot B_{hi}$ 를 재귀적으로 계산한 뒤 이 결과를 잘 합쳐 $C = A \boxdot B$ 를 만들어 보는 게 목표이다. 이를 위해서는 M_{lo} 와 M_{hi} 를 $A_{lo} \boxdot B_{lo}$ 와 $A_{hi} \boxdot B_{hi}$ 로 표현하면 좋겠지만, 이 둘은 같지 않다. 기본적으로, M_{lo} 는 $(N+1) \times (N+1)$ 인데 $\Sigma(A_{lo} \boxdot B_{lo})$ 는 $(N/2+1) \times (N/2+1)$ 이기에 그렇다.

M_{lo}, M_{hi} 를 $C_{lo} = A_{lo} \boxdot B_{lo}$ 와 $C_{hi} = A_{hi} \boxdot B_{hi}$ 로 표현하기

우리는

- $\Sigma(A_{lo})$ 를 행/열 인덱스가 $[1, N+1] \times [1, N/2+1]$ 인 $(N+1) \times (N/2+1)$ 행렬로 정의한다.
- $\Sigma(A_{hi})$ 를 행/열 인덱스가 $(N+1) \times (N/2+1)$ 인 $[1, N+1] \times [N/2+1, N+1]$ 행렬로 정의한다.

- $\Sigma(B_{lo})$ 를 행/열 인덱스가 $(N/2+1)\times(N+1)$ 인 $[1, N/2+1]\times[1,N+1]$ 행렬로 정의한다.
- $\Sigma(B_{hi})$ 를 행/열 인덱스가 $[N/2+1, N+1]\times[1,N+1]$ 인 $(N/2+1)\times(N+1)$ 행렬로 정의한다.

여기서 $N/2+1$ 개의 행(과 열) 은 그 아래 (혹은 왼쪽) 의 행(과 열)의 값을 복사해서 $N+1$ 개의 행(과 열)로 확장된다. 이를 수식으로 풀면 다음과 같다:

- $\Sigma(A)_{lo}(i,j) = \Sigma(A_{lo})(i,j)$
- $\Sigma(A)_{hi}(i,j) = \Sigma(A_{hi})(i,j) + \Sigma(A_{lo})(i, N/2+1)$
- $\Sigma(B)_{hi}(i,j) = \Sigma(B_{hi})(i,j)$
- $\Sigma(B)_{lo}(i,j) = \Sigma(B_{lo})(i,j) + \Sigma(B_{hi})(N/2+1, j)$

더 전개하자:

- $M_{lo}(i,k) = \min_{j=1}^{N/2+1}(\Sigma(A_{lo})(i,j) + \Sigma(B_{lo})(j,k) + \Sigma(B_{hi})(N/2+1,k))$
- $M_{hi}(i,k) = \min_{j=1}^{N/2+1}(\Sigma(A_{hi})(i,j) + \Sigma(A_{lo})(i,N/2+1) + \Sigma(B_{hi})(j,k))$

$C_{lo} = A_{lo} \boxdot B_{lo}$ 이고 $C_{hi} = A_{hi} \boxdot B_{hi}$ 이다. 이걸로 표현을 대체해 보자.

- $M_{lo}(i,k) = \Sigma(C_{lo})(i,k) + \Sigma(B_{hi})(N/2+1,k)$
- $M_{hi}(i,k) = \Sigma(C_{hi})(i,k) + \Sigma(A_{lo})(i,N/2+1)$
  (C_{lo}, C_{hi} 를 $(N+1)\times(N+1)$ 행렬로 간주한다.)

변홧값을 관찰하면 다음과 같은 등식을 유도할 수 있다.

- $M_{lo}(i,k) = \Sigma(C_{lo})(i,k) + \Sigma(C_{hi})(1,k)$
- $M_{hi}(i,k) = \Sigma(C_{hi})(i,k) + \Sigma(C_{lo})(i,N+1)$

최종적으로 M_{lo} 와 M_{hi} 를 C_{lo} 와 C_{hi} 로 표현하는 법을 찾았다.

C_{lo}, C_{hi} 에서 C 유도하기

$\Sigma(C)(i,k) = \min(M_{lo}(i,k), M_{hi}(i,k))$ 라는 식으로 C 를 유도하려면, $M_{lo}(i,k) - M_{hi}(i,k) \geq 0$ 인 위치의 특성을 관찰하는 것이 도움이 될 것이다. 이 값을 $\delta(i,k) = M_{lo}(i,k) - M_{hi}(i,k)$라 하자. 다음 사실을 관찰할 수 있다.

- $\delta(i,k) = \Sigma(C_{lo})(i,k) - \Sigma(C_{lo})(i,N+1) + \Sigma(C_{hi})(1,k) - \Sigma(C_{hi})(i,k)$
- $\delta(i,k) = |\{x | 1 \leq x \leq i-1, 1 \leq C_{hi}[x] \leq k-1\}| - |\{x | i \leq x \leq N, k \leq C_{lo}[x] \leq N\}|$

위 함수가 i,k 에 대해 모두 비감소한다. 구체적으로, $C_{hi} \cup C_{lo}$ 의 값이 모두 다르니 다음이 성립한다.

- $0 \leq \delta(i,k+1) - \delta(i,k) \leq 1$
- $0 \leq \delta(i+1,k) - \delta(i,k) \leq 1$

$\delta(i,k) < 0$ 인 영역과 $\delta(i,k) \geq 0$ 인 영역을 양분하는 경계선을 그려보면, 이 경계선은 좌하단 모서리 $(N+1,1)$ 에서 시작해서 우상단 모서리 $(1,N+1)$ 로 향한다. 차잇값 $\delta(i,k+1) - \delta(i,k)$ 와 $\delta(i-1,k) - \delta(i,k)$ 는 $O(1)$ 시간에 계산할 수 있으니, two pointers를 사용하면 이 경계선을 $O(N)$ 시간에 계산할 수 있다.

목표는, $\Sigma(C)(i,j+1) - \Sigma(C)(i,j) - \Sigma(C)(i+1,j+1) + \Sigma(C)(i+1,j) = 1$ 을 만족하는 모든 점 (i,j) 를 찾는 것이다. 만약 C_{lo} 와 C_{hi} 에 있는 이러한 점이 경계선 근처에 있지 않다면, $\Sigma(C) = \Sigma(C_{lo})$ (혹은 hi) 이니 그대로 쓰면 된다. 하지만 만약에 경계선 근처에 있다면 그대로 쓰지 못할 수도 있다. 식을 써서 이 조건들을 구체적으로 풀어보자.

- Case 1. $\delta(i+1,j+1) \leq 0$: 이 경우 모든 모서리가 M_{lo} 에 있기에 C_{lo} 에 있는 점이 보존된다: 만약 $(i,j) \in C_{lo}$ 라면 그대로 쓰면 된다.
- Case 2. $\delta(i,j) \geq 0$: 이 경우 모든 모서리가 M_{hi} 에 있기에 C_{hi} 에 있는 점이 보존된다: 만약 $(i,j) \in C_{hi}$ 라면 그대로 쓰면 된다.
- Case 3. 둘 다 아님: 이 경우 $\delta(i,j) = -1, \delta(i,j+1) = \delta(i+1,j) = 0, \delta(i+1,j+1) = 1$ 이 성립한다. 이 경우 $(i,j) \in C$ 가 항상 만족함을 보일 수 있다 (증명 생략).

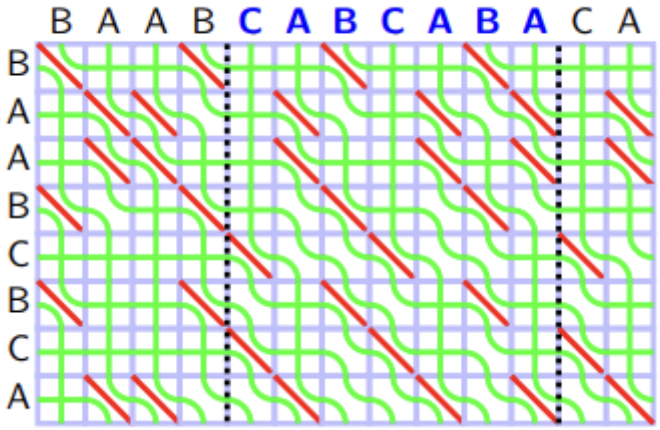
C_{lo} 와 C_{hi} 에 있는 점들의 x-좌표와 y-좌표는 서로 다르기 때문에, C 를 C_{lo}, C_{hi} 의 합집합으로 취하고 Case 3가 성립하는 지점에 대해서만 값을 바꿔주면 된다. 이는 경계선을 따라 이동하면서 Case 3 조건이 만족할 때마다 값을 변경해 주는 식으로 구현할 수 있다.

최종적으로, 우리는 $A \boxdot B$ 을 $A_{lo} \boxdot B_{lo}$ 과 $A_{hi} \boxdot B_{hi}$ 를 사용해서 계산하는 알고리즘을 얻었다. 이를 사용해 분할 정복하면 $T(N) = O(N) + 2T(N/2)$, 즉 $T(N) = O(N \log N)$ 시간에 $\boxdot$ 연산자를 구현할 수 있다.

이 알고리즘의 구현은 그렇게 어렵지 않았고, 내가 구현했을 때 나름대로 짧은 코드 로 구현할 수 있었다. 하지만 실제로 Seaweed matrix를 얻으려고 했을 때 인터넷에 있는 다른 코드들보다 5배 정도 느려서 문제가 되었다. 내 생각에 속도 차이는 메모리 관리 문제에서 발생하는 것 같다. 나는 재귀 함수에서 많은 vector를 새로 생성하고, 빠른 코드들은 그냥 $O(n)$ 메모리 풀을 만든 후 해당 메모리 풀에서 계산을 진행한다. 빠른 코드를 보면 다시 직접 짜고 싶다는 생각이 들지 않아서 그냥 그 코드를 복사 붙여넣기 해서 문제들을 해결했다. 구현은 LibreOJ. 单位蒙日矩阵乘法 문제에서 테스트 할 수 있으며 나의 최종 제출 도 다음과 같다.

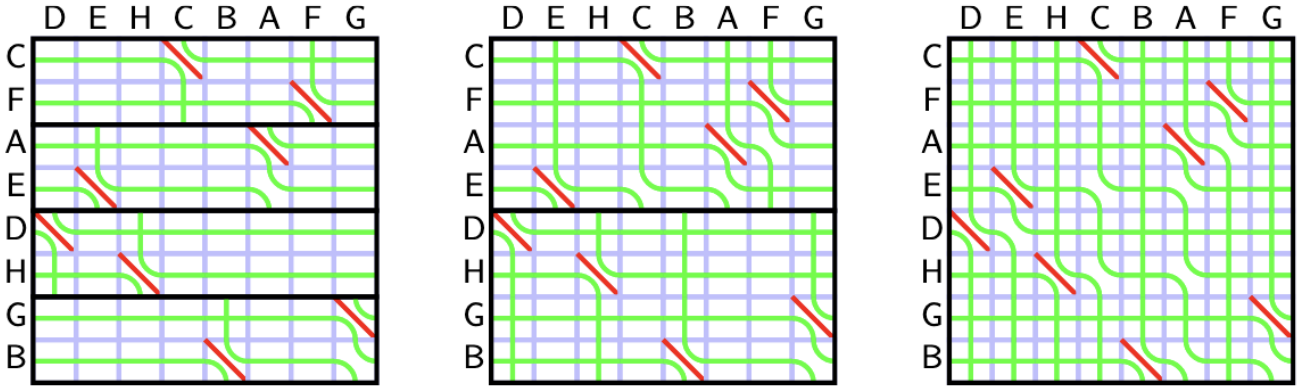
Chapter 5. $\boxdot$ 연산자로 Seaweed matrix 얻기

이제 $\boxdot$ 연산자를 구현하는 법을 알았고, Range LIS 문제를 $\boxdot$ 연산을 $O(N^2)$ 번 적용해서 푸는 방법도 알았으니, 현재 우리의 알고리즘은 $O(N^3 \log N + Q \log N)$ 이다. 대단히 비효율적이지만, 사실 여기서 $O(N)$ 의 최적화를 더하는 건 꽤 쉽다. Seaweed matrix의 각 *원소* 에 대해서 행렬을 정의할 필요가 없이, 각 *행* 에 대해서 행렬을 정의하면 되기 때문이다:



이렇게 $O(N^2 \log N + Q \log N)$ 알고리즘을 얻을 수 있는데, 아직 부족하다. 다행이도 이 부분은 앞 단계만큼 어렵지는 않다.

다시 한번 분할 정복을 사용한다. 순열 A 에 대한 Seaweed matrix의 결과를 반환하는 함수를 $f(A)$ 라고 하자. A_{lo} 를 $[1, N/2]$ 구간의 값을 가지는 원소들로만 이루어진 부분순열, A_{hi} 를 $[N/2+1, N]$ 구간의 값을 가지는 원소들로만 이루어진 부분순열 이라고 하자. 우리의 전략은 $f(A_{lo}), f(A_{hi})$ 의 seaweed matrix를 재귀적으로 계산한 후 이 둘을 합치는 것이다. 합치는 것 자체는 $\boxdot$ 연산으로 할 수 있지만, 이 작은 seaweed matrix 들에서 빠진 열이 있다는 것이 문제가 된다.



Seaweed의 규칙을 상기해 보자: 만약 두 seaweed가 만난 적이 없을 때는 둘이 교차한다. 이를 생각해 보면 빠진 열에 대해서는 그냥 seaweed가 아래로 감을 관찰할 수 있다. 고로, A_{lo}, A_{hi} 의 seaweed matrix를 원래 행렬에 맞게 키우는 것은 그냥 빠진 열과 빠진 행 (여기서는 그냥 들어온 데로 간다) 에 대한 값을 이렇게 채워주기만 하면 되고, 그 이후는 $\boxdot$ 연산이다. 크기를 키우는 데 $O(N)$ 시간, $\boxdot$ 연산에 $O(N \log N)$ 시간이니, $T(N) = 2T(N/2) + O(N \log N) = O(N \log^2 N)$ 이 성립한다. 드디어 우리는 Range LIS 를 $O(N \log^2 N + Q \log N)$에 해결할 수 있다! 이 코드 에서 위에 언급한 모든 내용의 구현체를 볼 수 있다.

Chapter 6. Seaweed를 사용한 문제 해결

Seaweed matrix를 $O(N \log^2 N)$ 에 구했으니 이를 사용하여 행렬의 Range LIS를 구하는 것은 자명하다.

Problem: Range LIS. 순열 A 와 Q 개의 쿼리 (l, r) 가 주어졌을 때 $A[l], A[l+1], \ldots, A[r]$ 의 LIS를 계산하라.

Solution. A 의 Seaweed matrix를 $O(N \log^2 N)$에 계산하자. Chapter 1 에서 관찰했듯이, LIS의 길이는 인덱스가 최대 $l + N - 1$ 인 Seaweed 중 $[l, r]$ 에 도착한 것의 개수이다. $[l, r]$ 에 도착할 수 있는 Seaweed의 수는 최대 $r + N$ 이니, 대신 $[l + N, r + N]$ 에서 출발해서 $[l, r]$ 에 도착하는 Seaweed의 수를 세고 이를 $r - l + 1$ 에서 빼주자.
다시 말해, LIS의 크기는 $r - l + 1$ 에서 점선으로 표기된 박스의 위쪽 변에서 아래쪽 변으로 가는 Seaweed의 수를 빼줌으로서 계산할 수 있다. 2D 쿼리이니, 스위핑 + 펜윅 트리로 계산할 수 있다.

위 풀이에서 약간의 통찰이 있었다. 이를 사용하면 다른 비자명한 값들 역시 계산할 수 있다.

Problem: Prefix-Suffix LIS. 순열 A 와 Q 개의 쿼리 (l, r) 가 주어졌을 때 $A[1], \ldots, A[l]$ 에서 r 이상의 값들만으로 이루어진 LIS를 계산하라.
Solution. 우리는 $[r, N] \times [1, l]$ 상자의 위쪽 변에서 아래쪽 변으로 가는 Seaweed의 개수를 세어주고 싶다 (상자는 좌하단 위치에 있을 것이다). 위쪽 변을 거치는 Seaweed는 $[N - r + 1, N + l]$ 구간에서 시작하니, 앞 문제와 비슷한 2D 쿼리가 되고 스위핑 + 펜윅 트리로 계산할 수 있다. 같은 전략을 Suffix-Prefix LIS에 대해서도 적용할 수 있다 (Prefix-Prefix, Suffix-Suffix 에 대해서 되는지는 잘 모르겠는데, 이 값들은 이런 테크닉 없이도 계산이 자명하다.)

두 번째 문제를 사용하면 잘 알려진 문제를 효율적으로 해결할 수 있다.

Problem: Maximum Clique in a circle graph. 원 위에 $2n$ 개의 점이 $1, 2, \ldots, 2n$ 순서대로 적혀있으며, 이들을 잇는 n 개의 현이 존재한다. 각 현의 끝점은 서로 다르다. 서로 교차하는 현들의 최대 부분집합을 계산하라.

Solution. 각 현의 *왼쪽 끝점* 을 번호가 작은 점, *오른쪽 끝점* 을 번호가 큰 점 이라고 하자. 최적해를 이루는 현 중 왼쪽 끝점이 가장 작은 현을 고정하자. 이걸 $c = (s, e)$ 라고 하면, 우리가 골라야 할 모든 현들은 c 와 교차해야 하며, 현들끼리도 서로 교차해야 한다. 다시 말해, 두 현 $p = (l_1, r_1), q = (l_2, r_2)$ 에 대해서 $l_1 < l_2$ 면 $r_1 < r_2$ 이다.

$A[x]$ 를 x 를 끝점으로 가지는 현의 반대쪽 끝점 번호라고 하자. 위 관찰을 요약하면 다음과 같다: 모든 $x < A[x]$ 에 대해서, $A[x], A[x + 1], \ldots, A[A[x] - 1]$ 중 모든 원소의 값이 $A[x]$ 이상인 LIS를 구하라. 이렇게 보면 어렵지만, 사실 이는 $A[1], A[2], \ldots, A[A[x] - 1]$ 중 모든 원소의 값이 $A[x]$ 이상인 LIS를 구하는 것과 동일하다: 이렇게 해서 얻은 현의 집합도 결국 서로 교차하기 때문이다. 이제 문제는 정확히 Prefix-Suffix LIS이기 때문에 $O(N \log^2 N)$ 시간에 해결된다. Naive한 알고리즘은 $O(N^2 \log N)$ 를 사용하는 것에 대비된다.

연습 문제

- LibreOJ. 单位蒙日矩阵乘法
- Yosupo Judge. Static Range LIS Query
- Ptz Winter 2014. Circle Clique ($O(n \log^2 n)$ 에 해결해 보자.)
- Ptz Summer 2018. Form the Maximal Set ($O(n \log^2 n)$ 에 해결해 보자.)
- BOJ 26164. 싱싱미역