Faster and Simpler Algorithm for Sorting Signed Permutations by Reversals

Introduction

어떠한 순열 $\pi = (\pi_1, \ldots, \pi_n)$ 에 대해서 순열의 몇 개 원소에 -1 을 곱해서 만들 수 있는 수열들을 *signed permutation* (부호 있는 순열) 이라고 하자. 이 수열에 우리는 *뒤집기* 라는 연산을 할 수 있는데, 뒤집기 연산 $r(i, j)$ 는 구간 $[i, j]$ 에 대해서 구간의 원소에 -1 을 곱하고 구간을 뒤집는 것을 뜻한다. 즉, π 에 뒤집기 연산 $r(i, j)$ 를 수행하면, 수열이 $\pi = (\pi_1, \ldots, -\pi_j, -\pi_{j-1}, \ldots, -\pi_i, \pi_{j+1}, \ldots, \pi_n)$ 으로 변환된다.

주어진 수열을 $(+1, +2, \ldots, +n)$ 으로 변환하는 것을 *signed permutation sorting* (부호 있는 순열 정렬하기) 라고 할 때, 우리는 signed permutation을 최소 횟수의 연산으로 정렬하는 문제를 해결하려고 한다.

이 문제는 언뜻 보면 효율적인 정렬 알고리즘과 관련이 있어 보이지만, 실제로는 분자생물학 분야에서 시작된 문제이다. 거대한 크기의 DNA 게놈을 분석하는 일반적인 방법은 유전자의 등장 순서를 비교하는 방법이다. 염색체 상의 유전자를 수직선 상의 점으로 모델링하자. 각 유전자는 종류와 방향을 가지는데, 이는 순열 상의 숫자와 부호로 표현할 수 있다. 위에서 설명한 *뒤집기* 연산은 어떠한 게놈에 가해진 진화적 변화를 모델링한다. 즉, 어떠한 게놈이 진화하는 과정은 직선 상의 유전자의 위치가 반전되고, 그 과정에서 각 유전자가 보는 방향이 뒤집히는 식으로 모델링이 가능하다는 것이다. 즉, 최소 횟수의 뒤집기 연산으로 어떠한 게놈 수열에서 다른 게놈 수열로 변화하는 과정을 찾으면, 두 게놈의 진화적 거리와 진화 과정을 짐작할 수 있다.

이러한 생물학적 중요성에 힘입어 이 문제는 오랜 기간동안 많은 연구의 대상이 되어 왔고, 1995년 Hannenhalli와 Pevzner는 이 문제가 다항 시간에 해결 가능한 문제임을 보이고 이를 $O(n^4)$ 에 해결하는 알고리즘을 제시하였다. 이후에도 이 문제를 해결하는 더 많은 개선된 방법이 고안되었다. 이 글에서는 1998년 Haim Kaplan, Ron Shamir, Robert E. Tarjan이 고안한 $O(n^2)$ 알고리즘을 소개한다. 이 알고리즘은 기존에 나온 모든 알고리즘들보다 빠르고, 또한 그 알고리즘과 증명이 훨씬 간단하다는 점에서 큰 의외가 있다. 또한, 이 알고리즘은 만약 실제 뒤집기 연산의 필요 횟수가 작을 경우 훨씬 빠르게 동작한다: 만약 필요한 횟수가 r 이라면, 알고리즘의 시간 복잡도는 $O(rn + n\alpha(n))$ 이다. ($\alpha(n)$ 은 Inverse Ackermann Function).

Preliminaries

일단 몇 가지 정의를 짚고 넘어간다.

1. 입력은 signed permutation $\pi = (\pi_1, \ldots, \pi_n)$ 이다.
2. $\pi_0 = 0, \pi_{n+1} = n + 1$ 이라고 하자. (길이 $n + 2$ 수열이라고 가정한다.)
3. $0 \le i \le n$ 에 대해 (π_i, π_{i+1}) 을 *gap* 이라고 부른다. 이 때 $abs(\pi_i - \pi_{i+1}) \neq 1$ 일 경우 이 *gap* 은 *breakpoint*. 그 외 경우 *adjacency* 라고 불린다.
4. $b(\pi)$ 를 순열 π 의 *breakpoint* 의 개수로 정의한다.
5. 뒤집기 연산 $r(i, j)$ 는 π 를 $(\pi_1, \ldots, -\pi_j, -\pi_{j-1}, \ldots, -\pi_i, \pi_{j+1}, \ldots, \pi_n)$ 로 변환한다. 이 때 $r(i, j)$ 는 *gap* (π_{i-1}, π_i), (π_j, π_{j+1}) 에 *작용했다* 고 한다.

Breakpoint Graph

어떠한 순열 $\pi = (\pi_1, \ldots, \pi_n)$ 의 *breakpoint graph* $B(\pi)$ 는 $n + 2$ 개의 정점 $\{\pi_0, \pi_1, \ldots, \pi_n, \pi_{n+1}) = \{0, 1, \ldots, n + 1\})$ 을 가진다. 정점 π_i, π_{i+1} 에 대해 (π_i, π_{i+1}) 이 π 에서 breakpoint 라면 두 정점간에 *검은 간선* 을 이어주고, 정점 i, j 에 대해 (i, j) 가 π^{-1} 에서 breakpoint 라면 (즉, $abs(\pi_i - \pi_j) = 1, abs(i - j) \neq 1$

) 두 정점 간에 *회색 간선* 을 이어준다. 여기서의 순열은 부호가 없어서, π^{-1} 의 정의는 우리가 일반적으로 아는 정의와 동일하다.

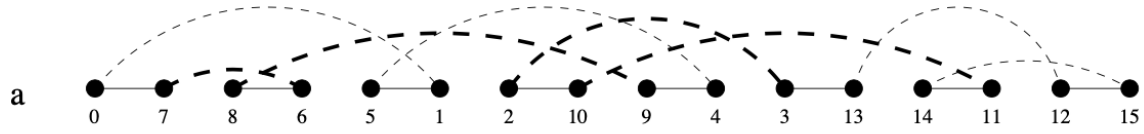
Breakpoint graph의 정의는 일반 순열에만 성립하니, Signed permutation에서도 성립하게 하려면 추가적인 작업이 필요하다. 길이 n 의 signed permutation π가 주어졌을 때, 원소 $x > 0$ 을 $2x-1, 2x$ 로 대체하고, 원소 $(-x) < 0$ 을 $2x, 2x-1$ 로 대체함으로써 길이 $2n$ 의 순열 $u(\pi)$ 를 만들 수 있다.

다음 사실을 관찰할 수 있다.

Observation 1. $B(u(\pi))$ 의 임의의 정점은 인접한 간선이 없거나, 검은 간선 하나와 회색 간선 하나에 인접한다.

Proof. 정점이 π_0, π_{2n+1} 인 경우에는 개별적으로 쉽게 보일 수 있다. 그 외 정점이 $2x$ 라고 하자. 인접한 $2x-1$의 존재로 인해 최대 하나의 검은 간선과 회색 간선에 인접할 수 있다. 검은 간선이 존재한다면, $2x+1$ 은 $2x$ 와 인접하지 않으니 회색 간선도 존재한다. 회색 간선이 존재한다면 동일한 이유로 검은 간선이 존재한다. $2x-1$ 에 대해서도 대칭적으로 같은 논리가 성립한다.

즉, $B(u(\pi))$ 를 그래프로 보았을 때 그래프는 차수가 0인 정점들과, 회색 / 검은색 간선이 교대로 등장하는 사이클들로 이루어져 있다. 이 사실은 $B(\{4, -3, 1, -5, -2, 7, 6\})$ 을 표현한 아래 Figure에 확인할 수 있다.



$u(\pi)$ 에 $r(2i+1, 2j)$ 연산을 가하는 것은 π 에 $r(i+1, j)$ 연산을 가하는 것과 동일하다고 볼 수 있다. 고로, 전체 문제는 $u(\pi)$ 에 $r(2i+1, 2j)$ 연산을 반복적으로 진행해서 $u(\pi)$ 를 정렬시키는 것으로 볼 수 있다. 여기서부터는 이 변형된 문제를 해결하고, Reversal 연산은 $r(2i+1, 2j)$ 와 같이 시작점이 홀수고 끝점이 짝수인 구간에서만 가능하다고 정의한다. 다시 원래 문제로 돌아오지 않을 것이니, 이 시점부터 $\pi = u(\pi)$ 를 뜻한다. $c(\pi)$ 를 $B(\pi)$ 상의 사이클의 개수라고 정의한다.

순열 π 상의 임의의 reversal r 에 대해, $\Delta b(\pi, r) = b(\pi r) - b(\pi), \Delta c(\pi, r) = c(\pi r) - c(\pi)$ 라고 정의한다. 문맥상 π, r 이 명확한 경우 Δb, Δc 로 줄여 표기한다.

Reversal $r(i, j)$ 가 작용하는 두 개의 gap에 따라서, Δb, Δc 의 값을 결정할 수 있다. 상태는 다음과 같은 케이스로 나눌 수 있다. 각 케이스에 대한 검증은 어렵지 않아 생략한다.

- Case 1. Adjacency 2개: $\Delta b = 2, \Delta c = 1$
- Case 2. Adjacency 1개, Breakpoint 1개: $\Delta b = 1, \Delta c = 0$
- Case 3. 다른 사이클에 속하는 Breakpoint 2개: $\Delta b = 0, \Delta c = -1$
- Case 4. 같은 사이클에 속하는 Breakpoint 2개:
- Case 4a. $(\pi_i, \pi_{j+1}), (\pi_{i-1}, \pi_j)$ 중 Gray edge 2개: $\Delta b = -2, \Delta c = -1$
- Case 4b. $(\pi_i, \pi_{j+1}), (\pi_{i-1}, \pi_j)$ 중 Gray edge 1개: $\Delta b = -1, \Delta c = 0$
- Case 4c. $(\pi_i, \pi_{j+1}), (\pi_{i-1}, \pi_j)$ 중 Gray edge 0개이며, Breakpoint가 속하는 공통 사이클 C에서 정점 i, j 를 기준으로 사이클을 잘랐을 때, $i-1, j+1$ 이 속하는 Path가 같음: $\Delta b = 0, \Delta c = 0$
- Case 4d. $(\pi_i, \pi_{j+1}), (\pi_{i-1}, \pi_j)$ 중 Gray edge 0개이며, Breakpoint가 속하는 공통 사이클 C에서 정점 i, j 를 기준으로 사이클을 잘랐을 때, $i-1, j+1$ 이 속하는 Path가 같음: $\Delta b = 0, \Delta c = 1$

이에 따라 몇 가지 정의를 한다.

6. $\Delta b - \Delta c = -1$ 일 경우 reversal을 *proper reversal* 이라고 정의하자. 위 케이스에서 그러한 경우는 4a, 4b, 4d가 있다.
7. Reversal r 이 어떠한 회색 간선 e 에 인접한 두 검은 간선이 표현하는 Breakpoint에 작용했다면, r 이 회색 간선 e에 *작용했다* 고 한다. (회색 간선 e의 양 끝점에 인접한 Breakpoint를 기준으로 Reversal했다고 이해하면 된다.)
8. 회색 간선 e에 작용하는 Reversal이 proper한 경우 e 를 *oriented*, 그렇지 않은 경우 *unoriented* 라고 한다. 이 때, 회색 간선 (π_k, π_l) 이 oriented일 조건은 $k + l$ 이 짝수일 조건과 동치이다.

위 Figure에서

- 굵게 표시된 점선들은 oriented 회색 간선이다.
- 그렇지 않은 점선들은 unoriented 회색 간선이다.
- 실선은 검은 간선이다.

Interval Overlap Graph

어떠한 그래프 G 가 *Interval Overlap Graph* (혹은 Circle Graph) 라는 것은, G 의 각 정점에 구간 $I_1, I_2, \ldots, I_n$ 을 배정할 수 있어서 두 정점 i, j 간에 간선이 있다는 것과, I_i, I_j 간에 겹치는 구간이 있으며 둘 중 하나가 다른 것을 포함하지 않음을 뜻한다. 모든 구간의 끝점이 $[M]$ 의 부분집합이라과 하면, 원의 둘레에 $1, 2, \ldots, M$ 의 자연수를 순서대로 쓴 후, 각 구간의 끝점을 선분으로 잇고, 두 선분이 끝점이 아닌 곳에서 교차할 때 간선을 이었다는 것과 동치라고 볼 수 있다.

이에 따라 몇 가지 정의를 한다.

9. 두 구간 간에 겹치는 구간이 있으며 둘 중 하나가 다른 것을 포함하지 않으면 두 구간이 교차한다고 (*overlap*) 하자.
10. 순열 π 에 대해서, $B(\pi)$ 의 회색 간선 (π_i, π_j) 에 대해 구간 $[i, j]$ 를 배정하였다고 하자. 이 때 π 의 Overlap graph $OV(\pi)$ 는 $B(\pi)$ 의 회색 간선을 정점으로 하는 Interval overlap graph 를 뜻한다. 두 회색 간선이 인접하지 않으니, 같은 끝점을 공유하는 구간은 없다.
11. $OV(\pi)$ 의 연결 컴포넌트에 oriented 간선이 하나라도 있으면 이 컴포넌트는 *oriented* 이고, 그렇지 않다면 이 컴포넌트는 *unoriented* 이다.
12. $B(\pi)$ 의 회색 간선의 집합을 X 라고 했을 때, $min(X)$ 를 $OV(\pi)$ 의 구간들 중 시작점 최소, $max(X)$ 를 $OV(\pi)$ 의 구간들 중 끝점 최대, $span(X) = [min(X), max(X)]$ 라고 정의하자.

이제 문제 해결에 필요한 몇 가지 Lemma를 소개한다.

Lemma 2. M 을 $OV(\pi)$ 의 어떤 연결 컴포넌트를 구성하는 회색 간선의 집합이라고 하면, $min(M)$ 은 짝수고, $max(M)$ 은 홀수이다.

Proof. $min(M)$ 이 홀수라고 가정하면, $span(M)$ 에 있는 모든 수에 대해서 $min(\pi_i) < \pi_{min(M)} < max(\pi_i)$ 가 성립한다. 만약 최솟값이라고 가정하면, $\pi_{min(M)} + 1 = \pi_{min(M)+1}$ 이 되기 때문에 M 이 공집합이 된다. 최댓값에 대해서도 비슷한 논리가 성립한다 (순열을 2차원 평면에 놓고 회색 간선을 그려보면 쉽게 이해할 수 있다). 고로 $span(M)$ 의 구간 최솟값 혹은 최댓값은 $min(M) < j < max(M)$ 에 있는 π_j 이다. 만약 이것이 최솟값이라면 π_j 는 홀수고 (짝수일 경우 구간 안에 더 작은 수가 들어가서 모순) 최댓값이라면 π_j 는 짝수다. 일반성을 잃지 않고 최솟값 케이스를 보면, $\pi_j - 1$ 은 $span(M)$ 밖에 존재하며, 고로 $(\pi_j - 1, pi_j)$ 는 회색 간선이다. 이는 M 이 연결 컴포넌트라는 가정에 모순이다. $max(M)$ 이 짝수임도 동일하게 보일 수 있다.

Lemma 3. $B(\pi)$ 의 한 연결 컴포넌트에 속하는 모든 회색 간선을 골랐을 때, 이들은 모두 $OV(\pi)$ 상에서 하나의 연결 컴포넌트에 속한다.

Proof. 모순을 가정하자. $B(\pi)$ 의 한 연결 컴포넌트 (사이클) C 를 고르자. 사이클 상에서 검은 간선을 무시했을 때, 연속하게 등장하며 다른 컴포넌트에 속하는 두 회색 간선 e_1, e_2 이 존재한다. e_1 이 속하는 컴포넌트를 M_1, e_2 가 속하는 컴포넌트

를 M_2 라고 하자. $OV(\pi)$ 의 서로 다른 연결 컴포넌트는 겹치지 않거나 하나가 다른 하나에 완전히 포함된다. 두 가지 경우를 고려하자.

- $span(M_2) \subseteq span(M_1)$. (반대 경우도 동일) e_1, e_2 는 다른 컴포넌트에 속하니 교차할 수 없다. 고로 e_2의 오른쪽 끝점이 $max(M_2)$ 이며 짝수이거나, e_2 의 왼쪽 끝점이 $min(M_2)$ 이고 홀수이다. 어떠한 경우여도 Lemma 2에 모순이다.
- $span(M_2) \cap span(M_1) = \emptyset$. 일반성을 잃지 않고 $max(M_1) < min(M_2)$ 라고 하면, e_1 의 오른쪽 끝점은 짝수이며 $max(M_1)$ 과 동일하고, 이는 Lemma 2에 모순이다.

Observation 4. $OV(\pi)$ 에는 차수가 0인 정점이 없다.

Proof. $B(\pi)$ 의 사이클 크기가 항상 4 이상이라 Lemma 3에 의해 $OV(\pi)$ 의 연결 컴포넌트의 크기가 항상 2 이상이다.

Hurdles

$\pi_{i_1}, \pi_{i_2}, \ldots \pi_{i_k} (i_1 < i_2 < \ldots < i_k)$ 를 $OV(\pi)$ 의 모든 Unoriented component에 속하는 구간의 양 끝점을 모두 모은 subsequence라고 하자. 이 subsequence를 원형으로 나열하고, 각각의 unoriented component M에 대해서 M 에 속하는 구간의 양 끝점을 모두 표시하자. 표시한 점들이 원 상에서 연속된 구간을 이루면, M 을 ***hurdle*** 이라고 부른다.

이에 따라 몇 가지 정의를 한다.

13. 어떠한 hurdle을 $OV(\pi)$ 에서 지웠을 때 다른 unoriented component가 새롭게 hurdle이 되지 않는다면 이를 ***simple hurdle*** 이라고 부른다.
14. ***simple hurdle*** 이 아닌 hurdle을 ***super hurdle*** 이라고 부른다.
15. 어떠한 순열의 모든 hurdle이 ***super hurdle*** 이며 hurdle이 홀수개면 이 순열은 ***fortress*** 이다.

Outline of the Algorithm

Hannenhalli-Pevzner은 아래와 같은 Theorem을 증명했다.

Theorem 5. 순열을 정렬하기 위해 필요한 뒤집기 연산의 최소 횟수는 π가 fortress가 아닐 경우 $d(\pi) = b(\pi) - c(\pi) + h(\pi)$ 이고, fortress일 경우 $d(\pi) = b(\pi) - c(\pi) + h(\pi) + 1$ 이다.

이것이 하한인 이유는 Hannenhalli-Pevzner의 원문을 참고하라. 우리의 알고리즘의 목표는 이 상한 안에 순열을 정렬하는 알고리즘을 만드는 것이다. 이를 위한 계획은 다음과 같다.

- **Hurdle 제거하기.** Hurdle이 있는 순열에 대해서, $t = \lceil h(\pi)/2 \rceil$ 번의 연산을 통해서 $h(\pi) = 0, d(\pi) := d(\pi) - t$ 가 되도록 변환할 수 있다. 이 방법은 이 논문의 기여가 아니며, Hannenhalli-Pevzner의 논문의 내용을 그대로 사용한다. 고로 이 부분에 대해서는 이 글에서 다루지 않는다.
- **Safe reversal 찾기.** Unoriented component가 존재하는 순열이면 무조건 Hurdle이 존재한다. 고로 위 변환 이후 모든 컴포넌트가 oriented라고 볼 수 있다. 어떠한 oriented gray edge e 에 대해서, e 에 작용하는 reversal이 새로운 Hurdle을 만들지 않는 e 가 항상 존재함을 보일 수 있다. 이러한 reversal을 ***safe reversal*** 이라고 한다. 이 reversal을 $O(n)$ 에 찾을 수 있음을 보인다. Safe reversal은 proper하니, 이를 반복하면 항상 $d(\pi)$ 가 1씩 감소한다.

이 safe한 것이 존재함을 증명하고, 또한 그러한 oriented vertex를 찾는 선형 시간 알고리즘을 소개한다. 여기서:

16. $N(e)$ 는 e 에 인접한 정점들의 집합을 뜻한다.
17. $UN(e)$ 는 $N(e)$ 중 unoriented된 정점들의 집합을 뜻한다.
18. $ON(e)$ 는 $N(e)$ 중 oriented된 정점들의 집합을 뜻한다.

19. $OV(\pi)$ 에서 orient된 정점들의 클리크를 C 라고 하자. C 가 ***happy*** 하다는 것은, 모든 orient된 정점 $e \notin C$ 와 그에 인접한 모든 정점 $f \in N(e) \cap C$ 에 대해, oriented된 정점 $g \notin C$ 가 존재하여 $(g, e) \in E(OV(\pi)), (g, f) \notin E(OV(\pi))$ 가 만족한다는 것이다.

이 때, happy clique에는 항상 safe reversal이 존재한다.

Theorem 6. C 를 Happy clique라고 하고, $e \in V(C)$ 를 $UN(e)$ 의 값을 최대화하는 정점이라고 하자. $r(e)$ 는 safe reversal이다.

만약 Happy clique를 찾았다고 하면, 여기서 $UN(e)$ 의 값을 최대화하는 정점을 찾는 것은 단순한 계산 문제이다. 논문에서는 이를 $O(n)$ 에 하는 방법을 서술하는데, Segment tree 등의 자료구조를 사용해서 $O(n \log n)$ 정도에 찾는 것은 어렵지 않으며 역시 충분히 빠를 것이다. $OV(\pi)$ 에 Happy clique는 항상 존재하며, 증명은 선형 시간에 작동하는 구성적 알고리즘이다.

Theorem 7. $OV(\pi)$ 에는 Happy clique가 항상 존재한다.

Proof. $OV(\pi)$ 상의 모든 oriented vertices들을 왼쪽 끝점이 증가하는 순으로 정렬하자. 이를 $e_1, e_2, \ldots, e_k$ 라고 했을 때, 알고리즘은 $e_1, e_2, \ldots, e_i$ 의 happy clique C_i 를 관리하고, 만약 C_i 의 모든 구간을 포함하는 구간이 존재한다면 이러한 구간 t_i 를 하나 관리한다. 이 때의 불변량은 다음과 같다.

- 모든 $e_x \notin C_i, i_1 < x \leq i$ 에 대해 x 는 t_i 와 인접해야 한다.
- 모든 $e_x \notin C_i, x < i_1$ 중 C_i 에 있는 정점과 인접한 정점은, t_i 와 인접하거나, $R(e_p) < L(e_{i_1})$ 를 만족하는 구간 p 와 인접하다.

t_i 가 정의되지 않을 수 있음에 유의하라. 초기에는, $C_1 = \{e_1\}$, 그리고 t_1 을 정의하지 않는다. 현재 C_i, t_i 가 있을 때, e_{i+1} 을 다음과 같이 처리한다. $C_i = \{i_1, i_2, \ldots, i_j\}$ 라고 할 때

- $R(e_{i_j}) < L(e_{i+1})$ 일 경우 이후 들어오는 모든 구간은 C 와 인접하지 않으니 알고리즘을 종료하면 된다.
- t_i 가 정의되어 있고 $R(t_i) < R(e_{i+1})$ 일 경우, $(C_{i+1}, t_{i+1}) = (C_i, t_i)$ 로 두고 넘어간다.
- 그렇지 않은 경우:
- $R(e_{i_j}) < R(e_{i+1}), L(e_{i+1}) \leq R(e_{i_1})$ 일 경우, $(C_{i+1}, t_{i+1}) = (C_i \cup \{e_{i+1}\}, t_i)$
- $R(e_{i_j}) < R(e_{i+1}), L(e_{i+1}) > R(e_{i_1})$ 일 경우, $(C_{i+1}, t_{i+1}) = (\{e_{i+1}\}, t_i)$
- $R(e_{i_j}) \geq R(e_{i+1})$ 일 경우, $(C_{i+1}, t_{i+1}) = (\{e_{i+1}\}, e_{i_j})$

이 각각의 경우에 대해, invariant가 깨지지 않음을 귀납적으로 확인할 수 있고, invariant가 보존됨을 가정할 때, C_i 가 항상 Happy clique를 유지함 역시 귀납적으로 확인할 수 있다.