MO640 – Biologia Computacional

Carla Negri Lintzmayer

Prof. Dr. Zanoni Dias

Instituto de Computação – Unicamp

Segundo Semestre de 2014

Roteiro

1. Ordenação por Reversões Ponderadas
2. Limitando $P(n)$
3. Aproximando $p(\pi)$

Introdução

- A análise tradicional assume que o custo de cada reversão é unitário e independe do comprimento do segmento que está sendo invertido.
- No entanto, uma reversão genômica mais longa muito provavelmente vai perturbar mais o organismo, fazendo com que seja menos provável que ele sobreviva à mutação.
- Assim, é razoável que as probabilidades das reversões sejam dependentes do comprimento do segmento.

Definições

- Seja $\pi = (\pi_1 \ \pi_2 \ \dots \ \pi_n)$ uma permutação tal que $\pi_i \in \{1, 2, \dots, n\}$ e $\pi_i \neq \pi_j$ para $i \neq j$.
- Seja $\iota_n = (1 \ 2 \ \dots \ n)$ a permutação identidade.
- Seja $\rho(i, j)$, com $1 \leq i < j \leq n$, uma reversão:
  - $\pi \cdot \rho(i,j) = (\pi_1 \ \dots \ \pi_{i-1} \ \underline{\pi_j \ \dots \ \pi_i} \ \pi_{j+1} \ \dots \ \pi_n)$.
- O problema de Ordenação por Reversões Ponderadas consiste em, dada uma permutação π qualquer, transformá-la em ι_n utilizando uma sequência de reversões tal que o custo dessa sequência seja mínimo.

Definições

- O comprimento de uma reversão $\rho(i,j)$ é $|\rho(i,j)| = j - i + 1$ (número de elementos afetados).
- Definimos o custo de uma reversão $\rho(i,j)$ como sendo seu comprimento.
- Note que, tradicionalmente, o custo de qualquer reversão é 1.
- Se a sequência $\rho_1 \cdot \rho_2 \cdot \ldots \cdot \rho_k$ ordena π, então o custo dessa sequência é $|\rho_1| + |\rho_2| + \ldots + |\rho_k|$.
- Vamos denotar por $p(\pi)$ o custo mínimo necessário para ordenar uma dada permutação π.
- Vamos denotar por $P(n)$ o custo suficiente para ordenar qualquer permutação de n elementos.

Exemplos

$$\pi = (6\ 3\ 7\ 1\ 5\ 2\ 4),\ d_\rho(\pi) = 4$$

$$\begin{aligned}
&(6\ 3\ 7\ 1\ \underline{5\ 2}\ 4) \rightarrow |\rho(5,6)| = 2\\
&(6\ 3\ \underline{7\ 1\ 2\ 5\ 4}) \rightarrow |\rho(3,7)| = 5\\
&(6\ \underline{3\ 4\ 5}\ 2\ 1\ 7) \rightarrow |\rho(2,4)| = 3\\
&(\underline{6\ 5\ 4\ 3\ 2\ 1}\ 7) \rightarrow |\rho(1,6)| = 6\\
&(1\ 2\ 3\ 4\ 5\ 6\ 7) \rightarrow custo = 16
\end{aligned}$$

$$\begin{aligned}
&(6\ 3\ \underline{7\ 1\ 5\ 2}\ 4) \rightarrow |\rho(3,6)| = 4\\
&(6\ 3\ 2\ 5\ 1\ \underline{7\ 4}) \rightarrow |\rho(6,7)| = 2\\
&(6\ 3\ 2\ \underline{5\ 1}\ 4\ 7) \rightarrow |\rho(4,5)| = 2\\
&(\underline{6\ 3\ 2\ 1}\ 5\ 4\ 7) \rightarrow |\rho(1,4)| = 4\\
&(1\ 2\ 3\ \underline{6\ 5\ 4}\ 7) \rightarrow |\rho(4,6)| = 3\\
&(1\ 2\ 3\ 4\ 5\ 6\ 7) \rightarrow custo = 15
\end{aligned}$$

De fato, $p(\pi) = 15$.

Limitando $P(n)$

- Uma versão do *Selection sort* baseada em reversões faz no máximo $n-1$ reversões para ordenar qualquer permutação de tamanho n, sendo que uma fração delas tem comprimento potencialmente $\theta(n)$.
  - Portanto, ele garante que $P(n) \leq O(n^2)$.
- *Bubble sort* e *Insertion sort* fazem trocas de elementos vizinhos (cada uma com custo 2), uma para cada inversão na permutação de entrada.
  - Portanto, eles garantem que $P(n) \leq O(n^2)$.
- Em 2002, Ron Pinter e Steven Skiena mostraram o algoritmo `ReversalSort` para Ordenação por Reversões Ponderadas, que tem como subrotina chave o algoritmo `MedianEject`.
  - Esse algoritmo garante que $P(n) \leq O(n \lg^2 n)$.

Algoritmo `MedianEject`

- Dizemos que um elemento π_i de π está do lado errado da mediana $\lceil \frac{n}{2} \rceil$ se $\pi_i \leq \lceil \frac{n}{2} \rceil$ e $i > \lceil \frac{n}{2} \rceil$ ou se $\pi_i > \lceil \frac{n}{2} \rceil$ e $i \leq \lceil \frac{n}{2} \rceil$.
  - Exemplo: $\pi = (3\ 7\ 4\ 10\ 9\ 12 \mid 5\ 1\ 11\ 2\ 6\ 8)$ tem 8 elementos do lado errado.
- O `MedianEject` é aplicado em um intervalo da permutação, que vai da posição ini até a posição fim, $1 \leq ini \leq fim \leq n$.
- Uma execução dele vai consertar todos os elementos que estão do lado errado no intervalo dado (levando-os para o lado correto), considerando, portanto, a mediana $ini - 1 + \lceil \frac{fim-ini+1}{2} \rceil$.
- Vamos chamar de *run* um segmento maximal que contém apenas elementos de lado errado.
  - Exemplo: $\pi = (3\ \underline{7}\ 4\ \underline{10\ 9\ 12} \mid \underline{5\ 1}\ 11\ \underline{2\ 6}\ 8)$ tem 4 *runs*.

Eliminado elementos que estão do lado errado

$$\pi = (4\ 16\ 12\ 7\ 5\ 20\ 1\ 15\ 17\ 10\ 9\ 13\ 2\ 6\ 14\ 3\ 18\ 8\ 21\ 11\ 19)$$

$$(4\ \underline{16\ 12}\ 7\ 5\ \underline{20}\ 1\ \underline{15\ 17}\ 10\ 9 \mid 13\ \underline{2\ 6}\ 14\ \underline{3}\ 18\ \underline{8}\ 21\ \underline{11}\ 19)\ [runs = 7]$$

$$(4\ 5\ 7\ \underline{12\ 16\ 20}\ 1\ \underline{15\ 17}\ 10\ 9 \mid 13\ 14\ \underline{6\ 2\ 3}\ 18\ 21\ \underline{8\ 11}\ 19)\ [runs = 4]$$

$$(4\ 5\ 7\ 1\ \underline{20\ 16\ 12\ 15\ 17}\ 10\ 9 \mid 13\ 14\ 21\ 18\ \underline{3\ 2\ 6\ 8\ 11}\ 19)\ [runs = 2]$$

$$(4\ 5\ 7\ 1\ 9\ 10\ \underline{17\ 15\ 12\ 16\ 20} \mid \underline{11\ 8\ 6\ 2\ 3}\ 18\ 21\ 14\ 13\ 19)\ [runs = 2]$$

$$(4\ 5\ 7\ 1\ 9\ 10\ 3\ 2\ 6\ 8\ 11 \mid 20\ 16\ 12\ 15\ 17\ 8\ 21\ 14\ 13\ 19)\ [runs = 0]$$

Eliminado elementos que estão do lado errado

Algorithm 1: MedianEject

Input: π, ini, fim
Output: custo utilizado para eliminar elementos do lado errado no intervalo
if $ini = fim$ **then**
    return 0

$c \leftarrow 0$
$M \leftarrow ini - 1 + \left\lceil \frac{fim - ini + 1}{2} \right\rceil$
Seja r a quantidade de *runs* considerando a mediana M
$r' \leftarrow r$
for $k \leftarrow 1$ **to** $\lfloor \lg r \rfloor$ **do**
    Sejam s_j, e_j, para todo $1 \leq j \leq r'$, as posições inicial e final, respectivamente, da j-ésima *run*
    Seja x a quantidade de *runs* que existem antes da mediana M
    $m \leftarrow x$
    if $x \mod 2 = 1$ **then**
        $m \leftarrow x - 1$
    for $i \in \{1, 3, \ldots, m - 1\}$ **do**
        $\pi \leftarrow \pi \cdot \rho(s_i, s_{i+1} - 1)$
        $c \leftarrow c + s_{i+1} - s_i$
    $m \leftarrow r'$
    if $(r' - x) \mod 2 = 1$ **then**
        $m \leftarrow r' - 1$
    for $i \in \{x + 1, x + 3, \ldots, m - 1\}$ **do**
        $\pi \leftarrow \pi \cdot \rho(s_i, s_{i+1} - 1)$
        $c \leftarrow c + s_{i+1} - s_i$
    $r' \leftarrow \left\lceil \frac{x}{2} \right\rceil + \left\lceil \frac{r' - x}{2} \right\rceil$

Eliminado elementos que estão do lado errado

Algorithm 1: MedianEject (continuação)

Sejam s_1 e e_1 as posições inicial e final da primeira *run* restante, s_2 e e_2 as posições inicial e final da segunda e t o tamanho de cada uma delas

if $e_1 \neq M$ **then**
- $\pi \leftarrow \pi \cdot \rho(s_1, M)$
- $c \leftarrow c + M - s_1 + 1$

if $s_2 \neq M + 1$ **then**
- $\pi \leftarrow \pi \cdot \rho(M + 1, e_2)$
- $c \leftarrow c + e_2 - M$

$\pi \leftarrow \pi \cdot \rho(M - t + 1, M + t)$

$c \leftarrow c + 2t$

return c

Análise do custo utilizado por `MedianEject`

- O laço principal claramente executa $O(\lg r)$ vezes.
- No início de uma iteração qualquer, r' é a quantidade de *runs* da permutação atual e essa iteração realiza reversões que não se sobrepõem com a intenção de diminuir r' por cerca da metade.
  - Então, no máximo $\left\lceil \frac{r'}{2} \right\rceil$ reversões são realizadas.
  - Na k-ésima iteração, portanto, temos que $\left\lceil \frac{r'}{2} \right\rceil \leq \left\lceil \frac{r}{2^k} \right\rceil$.
  - A j-ésima dessas reversões tem um tamanho t_j e ela não se sobrepõe com nenhuma outra reversão.
  - Portanto, o custo de todas as reversões na k-ésima iteração é:

$$\sum_{j=1}^{\frac{r}{2^k}} t_j \leq fim - ini + 1.$$

- Ao fim do laço principal, três reversões extras de tamanho no máximo $fim - ini + 1$ cada uma colocam os elementos que estão do lado errado no lado certo.

Análise do custo utilizado por `MedianEject`

- `MedianEject` utiliza, portanto, custo de no máximo $O((fim - ini + 1) \lg r)$ para eliminar elementos que estão do lado errado no intervalo de ini até fim em uma permutação π qualquer.
- No pior caso, $ini = 1$, $fim = n$ e $r = O(n)$, de forma que `MedianEject` utiliza custo de no máximo $O(n \lg n)$.
- Complexidade de tempo: $O(n \lg n)$.

Ordenando qualquer permutação

Algorithm 2: ReversalSort

Input: π, ini, fim
Output: custo utilizado para ordenar π
if $ini = fim$ **then**
  return 0;
$c \leftarrow \texttt{MedianEject}(\pi, ini, fim)$
$M \leftarrow ini - 1 + \left\lceil \frac{fim-ini+1}{2} \right\rceil$
$c \leftarrow c + \texttt{ReversalSort}(\pi, ini, M)$
$c \leftarrow c + \texttt{ReversalSort}(\pi, M+1, fim)$
return c

Ordenando qualquer permutação: `ReversalSort`

(4 5 7 1 9 11 | 2 3 6 8 10)
(4 5 1 7 9 11 | 2 3 6 8 10)
(4 5 1 2 3 6 | 11 9 7 8 10)
(4 5 1 | 2 3 6 11 9 7 8 10)
(1 5 4 | 2 3 6 11 9 7 8 10)
(1 3 2 | 4 5 6 11 9 7 8 10)
(1 3 | 2 4 5 6 11 9 7 8 10)
(1 2 | 3 4 5 6 11 9 7 8 10)
(1 | 2 3 4 5 6 11 9 7 8 10)
(1 2 3 4 5 6 11 9 7 8 10)
(1 2 3 4 5 | 6 11 9 7 8 10)
(1 2 3 4 | 5 6 11 9 7 8 10)
(1 2 3 4 5 6 11 9 7 8 10)

(1 2 3 4 5 6 | 11 9 7 8 10)
(1 2 3 4 5 6 11 9 7 | 8 10)
(1 2 3 4 5 6 7 9 11 | 8 10)
(1 2 3 4 5 6 7 9 8 | 11 10)
(1 2 3 4 5 6 7 9 | 8 11 10)
(1 2 3 4 5 6 7 8 | 9 11 10)
(1 2 3 4 5 6 7 | 8 9 11 10)
(1 2 3 4 5 6 7 8 9 11 10)
(1 2 3 4 5 6 7 8 9 11 | 10)
(1 2 3 4 5 6 7 8 9 10 | 11)
(1 2 3 4 5 6 7 8 9 10 11)
(1 2 3 4 5 6 7 8 9 10 11)
(1 2 3 4 5 6 7 8 9 10 11)

Análise do custo utilizado por ReversalSort

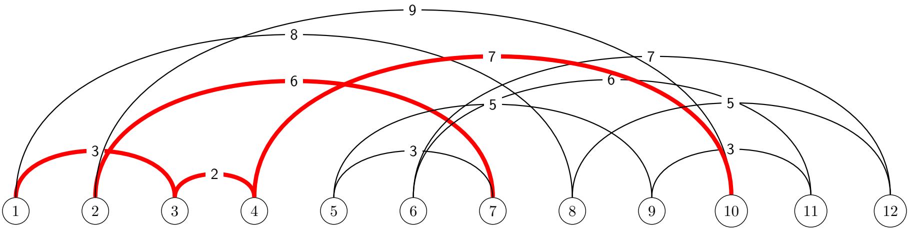
- Seja $c(n)$ o custo que `ReversalSort` utiliza para ordenar a permutação π dada, que tem tamanho n.
- Então $P(n) \leq c(n) = 2c\left(\frac{n}{2}\right) + O(n \lg n) = O(n \lg^2 n)$.
- Complexidade de tempo: $T(n) = 2T\left(\frac{n}{2}\right) + O(n \lg n) = O(n \lg^2 n)$.

Grafo de pesos

- Construímos $G_p(\pi)$, o grafo de pesos de uma permutação π da seguinte forma:
  - $V(G_p) = \{1, 2, \ldots, n\}$
  - $E(G_p) = \{(i, \pi_i)$ para $1 \leq i \leq n\}$
  - $w(i, \pi_i) = |i - \pi_i + 1|$ para todo $(i, \pi_i) \in E(G_p)$ tal que $i \neq \pi_i$
  - $w(i, \pi_i) = 0$ para todo $(i, \pi_i) \in E(G_p)$ tal que $i = \pi_i$
- Por convenção, ao lidarmos com uma aresta qualquer $(i, j) \in E(G_p)$, consideraremos que $i \leq j$.

Grafo de pesos

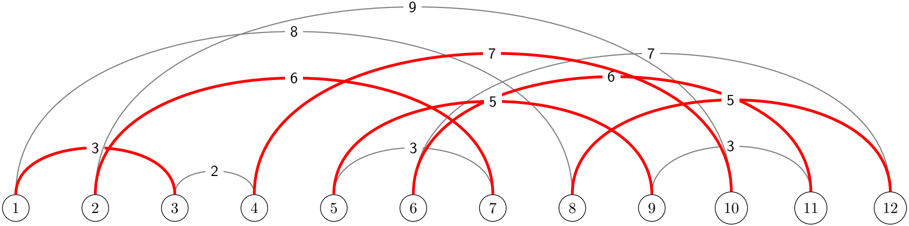
$$\pi = ({\color{red}3\ 7\ 4\ 10}\ 9\ 12\ 5\ 1\ 11\ 2\ 6\ 8)$$



Emparelhamento

- Um emparelhamento em um grafo é um subconjunto M das arestas do grafo tal que se $(u, v) \in M$, então $(u, x) \notin M$ e $(x, v) \notin M$ para qualquer outro vértice x.
- O custo de um emparelhamento M é a soma dos pesos de suas arestas, isto é, $c(M) = \sum_{e \in M} w(e)$.

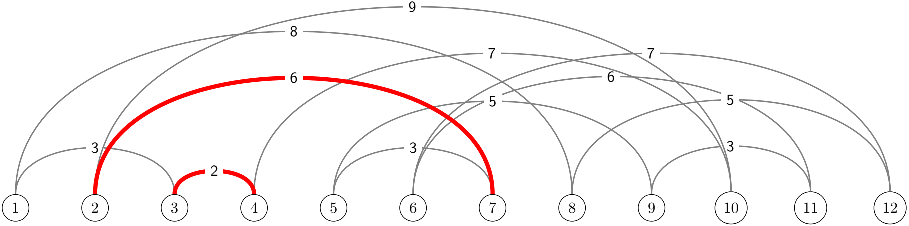
Emparelhamento



$$M = \{(1,3),(2,7),(4,10),(5,9),(6,11),(8,12)\}$$

$$c(M) = 32$$

Emparelhamento

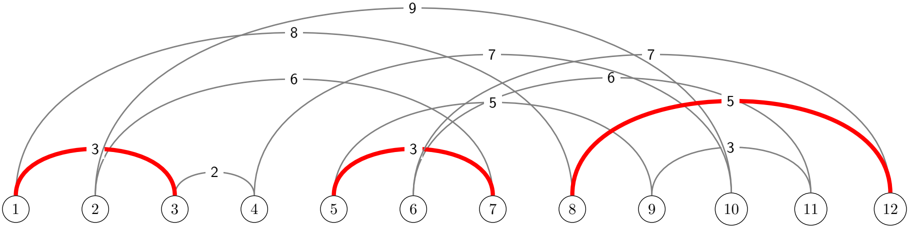


$$M = \{(2,7),(3,4)\}$$

$$c(M) = 8$$

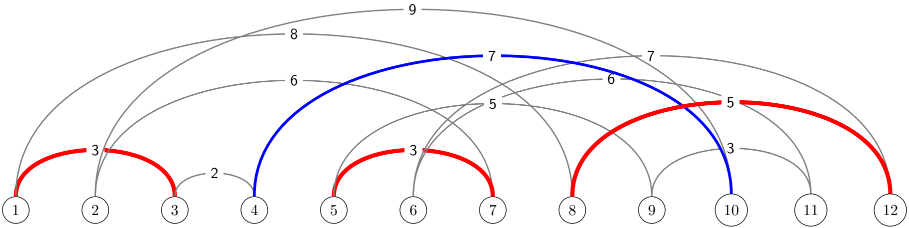
Emparelhamento sem cruzamento

- Um emparelhamento M em G_p é sem cruzamento se para todo par $(i,j),(k,\ell) \in M$, não existe x tal que $i \leq x \leq j$ e $k \leq x \leq \ell$.



$$M = \{(1,3),(5,7),(8,12)\}$$

Emparelhamento com cruzamento

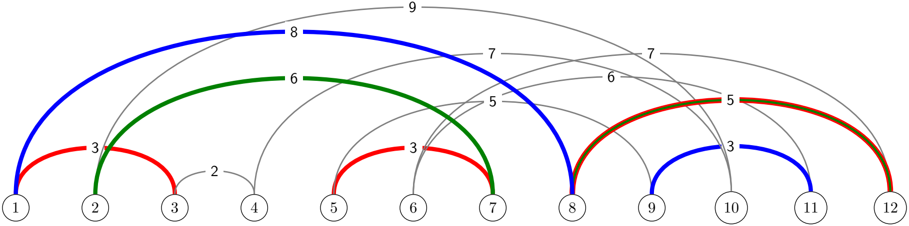


$$M = \{(1,3),(4,10),(5,7),(8,12)\}$$

M é emparelhamento mas não é sem cruzamento pois existe x tal que $4 \leq x \leq 10$ e $5 \leq x \leq 7$ (tome $x = 5$, por exemplo), assim como existe x' tal que $4 \leq x' \leq 10$ e $8 \leq x' \leq 12$ (tome $x' = 9$, por exemplo).

Emparelhamento sem cruzamento mais pesado

- Vamos chamar de M_p um emparelhamento sem cruzamento de G_p tal que M_p tem o maior custo dentre todos os emparelhamentos de G_p (também chamado de "mais pesado").



$M_p = \{(1,3),(5,7),(8,12)\}$
$M_p = \{(1,8),(9,11)\}$
$M_p = \{(2,7),(8,12)\}$

$c(M_p) = 11$

Aproximando $p(\pi)$

Lema 1

$p(\pi) \geq c(M_p) = \sum_{e \in M} w(e)$.

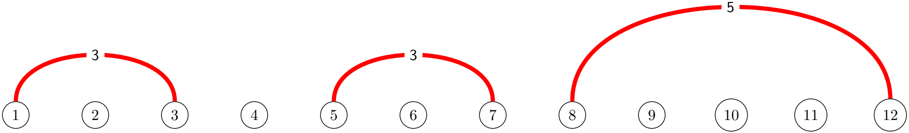
Demonstração.

Se $e = (i, j) \in M_p$, então a reversão $\rho(i, j)$ coloca pelo menos um elemento no lugar correto e tem custo $w(e) = j - i + 1$. Ao se fazer isso para todas as arestas de M_p, usaremos custo total $c(M_p)$, pois as reversões definidas pelas arestas de M_p não se sobrepõem, uma vez que tais arestas não se cruzam.

Note que as arestas (de peso não nulo) indicam elementos que estão fora do lugar. Suponha que $\rho(i, j)$ coloca j no lugar. Uma sequência de reversões de comprimento menor que traz j para a posição correta vai agir sobre segmentos que se sobrepõem por pelo menos um elemento (o próprio j), então seu custo total não pode ser menor do que $j - i + 1$. □

Aproximando $p(\pi)$

$$\pi = (3\ 7\ 4\ 10\ 9\ 12\ 5\ 1\ 11\ 2\ 6\ 8)$$



- Tome $M_p = \{(1,3),(5,7),(8,12)\}$.
- Esse emparelhamento define $\rho(1,3)$, $\rho(5,7)$ e $\rho(8,12)$.
- E a aplicação dessas reversões, pelo Lema 1, deve colocar pelo menos 3 elementos no lugar correto:

$$(\underline{3\ 7\ 4}\ 10\ \underline{9\ 12\ 5}\ \underline{1\ 11\ 2\ 6\ 8})$$

$$(4\ 7\ 3\ 10\ 5\ 12\ 9\ 8\ 6\ 2\ 11\ 1)$$

Aproximando $p(\pi)$

Lema 2

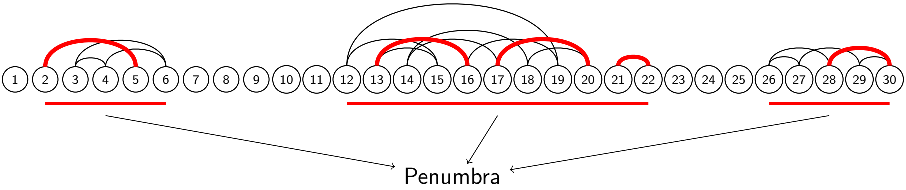
O número de elementos fora de posição em π é no máximo $3c(M_p)$.

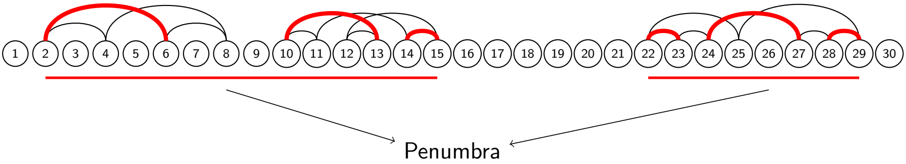
Ideia da demonstração:
Se uma aresta existe em $E(G_p)$, então pelo menos dois elementos de π estão fora de lugar (os extremos da aresta).
Precisamos observar, portanto, como estão distribuídas as arestas de $E(G_p)$ com base nas arestas de M_p (que são "mais bem comportadas").

Penumbra de M_p

- Vamos definir a penumbra de M_p como sendo o conjunto de intervalos onde elementos fora de posição podem estar, da seguinte forma:
  - Seja uma sequência maximal de arestas $e_i, \ldots, e_j \in M_p$ tal que os intervalos das arestas são consecutivos e o intervalo entre e_k e e_{k+1} é menor do que $w(e_k) + w(e_{k+1}) - 1$.
  - Considere o elemento mais à esquerda (resp. direita) de e_i (e_j) tal que existe uma aresta com um extremo ali e que cruza (os intervalos se intersectam) uma ou mais arestas dessa sequência. Chame sua posição de ℓ (r).
  - Os elementos do intervalo $[\ell, \ldots, r]$ fazem parte da penumbra de M_p.
- Note então que a penumbra de M_p consiste em vários intervalos disjuntos e que elementos que não fazem parte dela já estão nas posições corretas (pois se houvesse arestas entre dois intervalos da penumbra, M_p não seria máximo).

Penumbra de M_p





Penumbra de M_p e a mediana

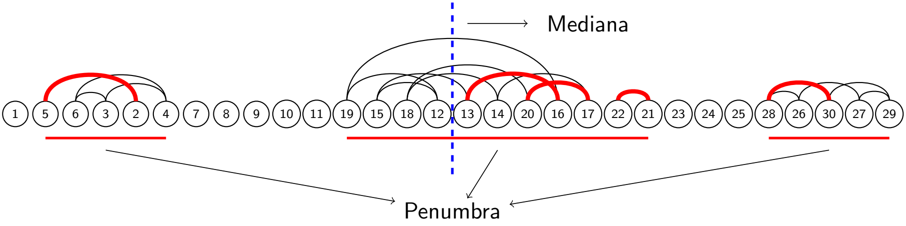
Lema 3

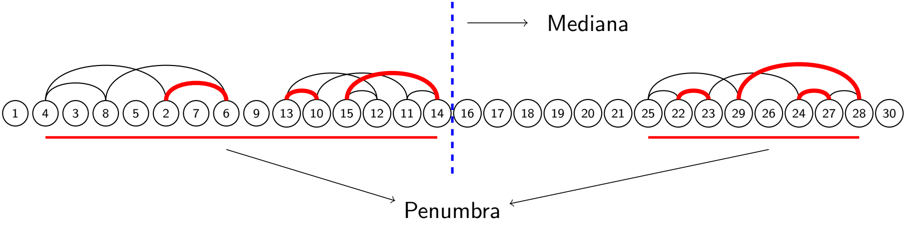
Nenhum elemento fora da penumbra se move durante a execução do `MedianEject`.

Demonstração.

Como elementos fora da penumbra estão nas posições corretas, `MedianEject` não vai aplicar reversões sobre tais posições.
Mais do que isso, se a penumbra é intersectada pela mediana, então os únicos elementos que serão movidos estão dentro do intervalo que é intersectado e nenhum outro elemento será afetado. $\square$

Penumbra de M_p e a mediana





Aproximando $p(\pi)$

- Como vimos, o custo das reversões aplicadas por uma iteração do laço principal do `MedianEject` não ultrapassa o tamanho do intervalo dado.
- O Lema 3 nos dá um resultado mais justo: esse custo não ultrapassa a quantidade de elementos que estão na penumbra (ou fora de posição).
- E pelo Lema 2, não existem mais do que $3c(M_p)$ elemento fora de posição.
- Então, cada iteração do laço principal do `MedianEject` faz reversões que não se cruzam e vão ter custo total de no máximo $3c(M_p)$.
- Logo, o custo total que `MedianEject` usa para consertar elementos que estão do lado errado vai ser de $O(c(M_p) \lg n)$.
- E o custo total que `ReversalSort` usa para ordenar π vai ser no máximo $O(c(M_p) \lg^2 n)$.

Aproximando $p(\pi)$

- Temos então que $c(M_p) \leq p(\pi) \leq O(c(M_p)\lg^2 n)$.
- `ReversalSort` é, portanto, um algoritmo de aproximação com fator $O(\lg^2 n)$ para Ordenação por Reversões Ponderadas.
- Em 2008, Michael A. Bender, Dongdong Ge, Simai He, Haodong Hu, Ron Y. Pinter, Steven S. Skiena e Firas Swidan apresentaram um algoritmo $O(\lg n)$-aproximado para este problema.
- Em 2004, Firas Swidan, Michael Bender, Dongdong Ge, Simai He, Haodong Hu e Ron Y. Pinter apresentaram um algoritmo $O(\lg n)$-aproximado para a versão com sinais.