2020牛客暑期多校训练营（第一场）

郭晓絮
shyhaimao@twitter

B–Suffix Array

C 1 2 0 1 0
aabaa
B 0 1 0 2 1

- Let C_i = min_{j > i and s_j = s_i} {j – i}
- The B–Suffix Array is equivalent to the suffix array of C_1 C_2 ... C_n

- Detailed proof can be found in “Parameterized Suffix Arrays for Binary Strings
- ” http://www.stringology.org/event/2008/p08.html

Infinite Tree

$m = 5000$

m^2

$$\sum_{1}^{k} \exp(\text{prime}[k], n)$$

fixed p

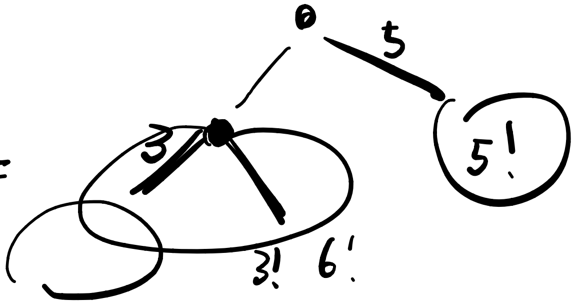
$p \quad 2p \quad 3p \quad \cdots \quad kp \quad \cdots$

$\frac{n}{p}$

- First, compute the “virual tree” of {1!, 2!, ..., n!}
- Second, to compute the actual cost, use Segment Tree or Fenwick Tree.
- O(m log^2 m)

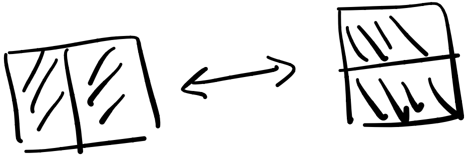
~~mlog~~

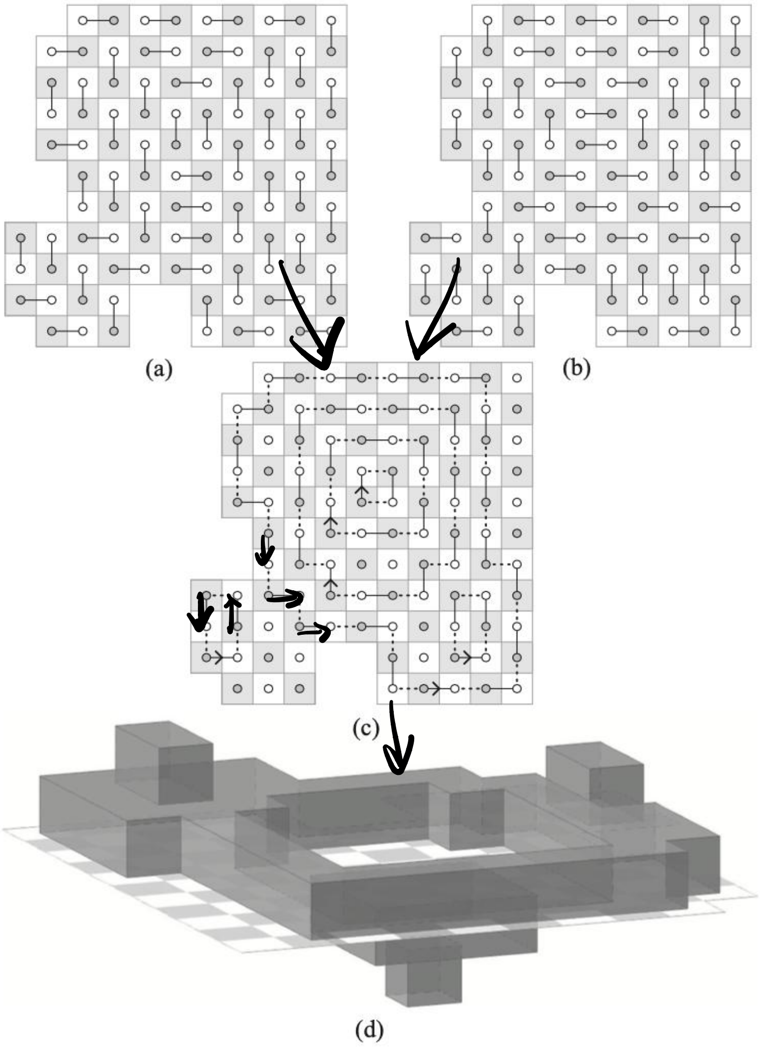
1!	2!	3!	4!	5!	6!
1	2	3	2	5	3



Domino

- See "Distances in Domino Flip Graphs"





Quadratic Form

$$\sqrt{b^T A^{-1} \cdot b}^{\,2}.$$

- The answer is b^T A^{-1} b, which can be proved by Lagrange Duality.
- All we need is to compute the inverse matrix of the matrix A.

$$\max \quad b^T \cdot x$$

$$\text{sub.} \quad x^T A x \le 1$$

$$\mathcal{L}(x, \lambda) = \underline{b^T x} - \lambda(x^T A x)$$

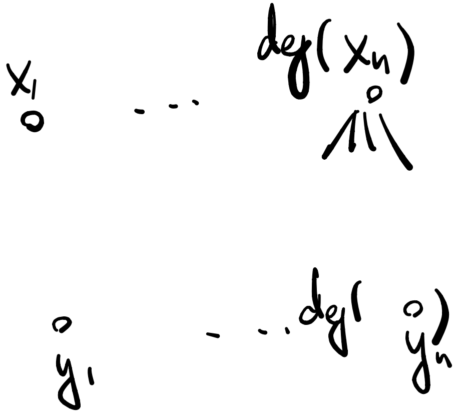
$$\frac{\partial \mathcal{L}}{\partial x} = b - 2\lambda A x = 0 \Rightarrow x = A^{-1} \frac{b}{2\lambda}$$

$$g(\lambda) = \frac{b^T A^{-1} b}{4\lambda} + \lambda \ge \sqrt{b^T A^{-1} b}$$

牛客竞赛 AC.NOWCODER.COM

Counting Spanning Trees

- The number of spanning trees is prod_{i >= 2} deg(x_i) deg(y_i)
- Detailed proof can be found in "Enumerative properties of Ferrers graphs" https://arxiv.org/pdf/0706.2918.pdf



$$= \frac{\prod deg(\ )\, deg(\ )}{\min(deg(x_i))\, \min(deg(y_i))}$$

Infinite String Comparision

自行车往哪边开

- Compare the string a^{infty} and b^{infty} directly n
- By (the Periodicity Lemma), if there is no mismatches in the first a + b – gcd(a, b) characters, the two string are identical

$$a^{\infty} = \overset{999\cdots 9}{aaa\cdots a}$$

$$10^5$$

$$lcm(\cancel{|a|}, |b|)$$

$$p, q \qquad \geqslant p+q-\gcd(p,q)$$

$$\gcd(p,q)$$

BaXianGuoHai, GeXianShenTong

$v_1 \ \dots \ v_n \qquad e_1 \ \dots \ e_n$

$\prod v_i^{e_i} = \sum e_i \cdot v_i$

- For simplicity, we denote the multiplication as +, and exponentiation as *
- Precompute B_{i, j} = 2^ {W * j} * v_i

  $B(i,j) = v_i^{(2^{w \cdot j})}$ $\qquad w = 16$
- To compute sum_{i, j} (sum e'_{i, j} * 2^{W * j}) v_i
  - = sum_x x sum_{i, j} [e'_{i, j} = x] B_{i, j} = sum_x x Q_x

  $< 2^{16}$

  $= x$

  $\prod v_i^{(\sum_j e_{i,j} 2^{w \cdot j})}$

  $= \prod_x \left(\prod \underbrace{v_i^{\sum [e_{i,j} = x] \cdot 2^{w \cdot j}}}_{B_{i,j}} \right)^x$

  $= \prod Q_x^x$
- To compute sum_x x Q_x
  - = sum_x (sum_{y >= x} Q_y)
- The overall complexity is O(nm / W + 2^W)

  $\frac{n \cdot m}{16} + 2^{16}$
- Taking W = 16 yields a fast enough solution

$Q_1 \cdot Q_2^2 \, Q_3^3 = $

Q_3

$Q_2 \cdot Q_3$

$Q_1 \, Q_2 \, Q_3$

Minimum-cost Flow

- We denote the cost in a network with capacity c and flow f as cost(c, f).
- $= \frac{u_i}{v_i}$
- cost(c, 1) = cost(c * 1/c, 1 / c) * c = cost(1, 1 / c) * c
- For a network with unitary capacity, its cost grows linearly with the flow f, with at most O(m) pieces.
- Thus, we can compute O(m) pieces first, and query in O(log m) time.

cap = 1

$cost_i$

q queries

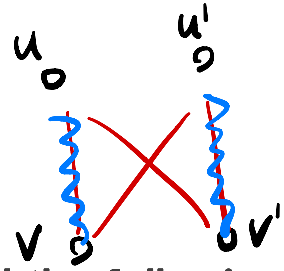
0 1

0 2

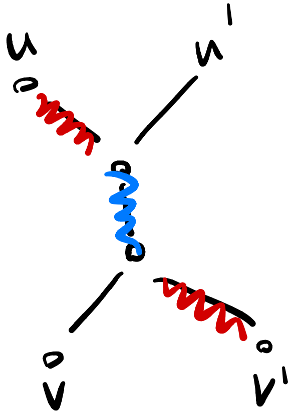
$\frac{u_i}{v_i}$

1 or 2

$d_i = 1$



- For an edge e=(x, y) where d_x = d_y = 2, add the following edges:
  - (x, e) (x', e)
  - (y, e') (y', e')
  - (e, e')
- The problems is turned into to find a perfect matching in a general graph, which can be solved with Edmond's Algorithm.



Easy Integration

$$\frac{(n!)^2}{(2n+1)!} \qquad \frac{\Gamma(n+\frac{1}{2})^2}{\Gamma(2n+1+\frac{1}{2})}$$

- The value is (n!)^2 / (2n+1)!
- Detailed proof can be found in “Wallis' integrals”. https://en.wikipedia.org/wiki/Wallis%27_integrals

Thanks