Алгоритмы и структуры данных

Приближённый поиск подстрок

CS Center, Новосибирск

Приближённый поиск подстрок

- $P, T \in \Sigma^*$
  - $P \approx P', P' \text{ is\_sub } T?$

1. $P \approx P'$:
   - $|P| = |P'|$
   - $\text{distH}(P, P') \leq k$
     - $\text{distH}(P, P') = |\{ i, P[i] \neq P'[i] \}|$
2. $P \in (\Sigma \cup \{?\})^*, P \approx P'$:
   - $|P| = |P'|,$
   - $P'[i] \in \{ P[i], ? \}, i \in [0 : |P|)$
3. $P \approx P'$:
   - $\text{ED}(P, P') \leq k$

Приближённый поиск: $P \in (\Sigma \cup \{?\})^*$

- $P = ?^*a_0\, P_1\, ?^*a_1\, P_2\, ?^*a_2 \ldots ?^*_{k-1}\, P_k\, ?^*a_k$
  - Ахо-Корасик: $O(|T|+|P|+|A|)$ – неэффективно
  - Массив счётчиков cnt
    - cnt[i]=k => вхождение в позиции i
    - найден P_i в позиции j -> увеличить счётчик cnt[j-shift[i]]
    - вхождения находятся по порядку
      - достаточно хранить |P| счётчиков

#include FFT

- $a = (a_0, \ldots a_n)$
- $b = (b_0, \ldots b_m)$
- $c = (c_0, \ldots c_{n+m})$
  - $c_i = \Sigma_j a_j b_{i-j}$
- Есть алгоритм со сложностью $O((n+m) \log(n+m))$
- $b'_j = b_{m-j}$
  - $c_i = \Sigma_j a_j b'_{i-j} = \Sigma_j a_j b_{m-i+j}$
- $c'_s = c_{m-s}$
  - $c'_i = c_{m-i} = \Sigma_j a_j b_{i+j}$

Приближённый поиск: distH(P, P’) ≤ k

- $|\Sigma_{abc}| = C$
- C масок для строки α:
  - $mask_{\alpha x} \in \{0, 1\}$
  - $|mask_{\alpha x}| = |\alpha|$
  - $mask_{\alpha x}[i] = \alpha[i]=x \ ? \ 1 : 0$
- $match(i, x) = \Sigma_j mask_{Px}[j] \cdot mask_{Tx}[i+j]$
- $match(i) = \Sigma_{x \in \Sigma_{abc}} match(i, x)$
  - $match(i) \geq |P|-k$ => вхождение в позиции i
- N=|P|, M=|T|
  - Сложность: O(CMlogM)

- Делим текст на блоки длины 2N
  - Блоки перекрываются, чтобы не потерять вхождения
  - O(M/N) блоков
- В каждом блоке ищем шаблон независимо
  - сложность в блоке O(CNlogN)
- Общая сложность

  O(CMlogN)

Приближённый поиск: P ∈ (Σ ∪ {?})*

- Замена Σ на {0, 1} – работаем с бинарными строками
  - code: Σ -> {0, 1}*
    - |code(x)| = log|Σ|
    - code(?) = 00...0
  - bin(α) = code(α[0]) code(α[1]) ... code(α[|α|-1])
    - |bin(α)| = |α|log|Σ|
  - Свертка bin(P)*bin(T) -> res
    - res[i log|Σ|] = popcnt(bin(P)) => вхождение в позиции i
  - Сложность O(MlogC log(NlogC))

Приближённый поиск: $P \in (\Sigma \cup \{?\})^*$

- $R = \{ i, P_i \neq ? \}$
  - $\Sigma_{j \in R} (P_j - T_{i+j})^2 = 0$ => вхождение в позиции i
- $\Sigma_{j \in R} (P_j - T_{i+j})^2 = \Sigma_j (P_j - T_{i+j})^2 \cdot X_j = \Sigma_j P_j^2 \cdot X_j - 2\Sigma_j P_j \cdot T_{i+j} \cdot X_j + \Sigma_j T_{i+j}^2 \cdot X_j$
  - $\Sigma_j P_j^2 \cdot X_j$ не зависит от i
  - $\Sigma_j P_j \cdot T_{i+j} \cdot X_j$ – свертка T и PX
  - $\Sigma_j T_{i+j}^2 \cdot X_j$ – свертка T^2 и X
    - сложность O(MlogN), M=|T|, N=|P|

Приближённый поиск: ED(P, P’) ≤ k

Динамическое программирование

dp[i][j] = min {

dp[i-1][j] + (0<i<|T| ? 1 : 0),

dp[i][j-1] + 1,

dp[i-1][j-i] + (T[i]=P[j] ? 1 : 0)

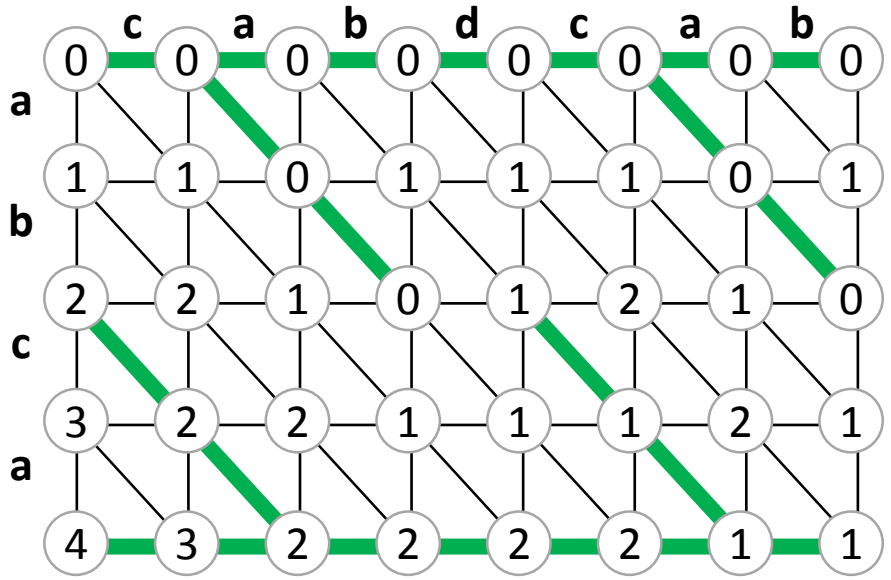
}

dp[|T|][|P|] ≤ k => есть вхождение

- Сложность O(|T||P|)

Пример:

P=abca, T=cabdcab



Приближённый поиск: ED(P, P’) ≤ k

- S – вершины в строке 0, T – вершины в строке |P|
- D_i = {v | dist(S, v) ≤ i}
- $D_k \cap T = \emptyset$?
- D_i' вместо D_i – на каждой диагонали оставляем только последнюю вершину
  - $|D_i'| \le |P| + |T|$
  - $D_k \cap T = \emptyset \Leftrightarrow D_k' \cap T = \emptyset$
- Нужно построить D_0' и научиться переходить D_i' –> D_{i+1}'
  - D_0' строится с помощью LCP(P, T[i :)) за линейное время
  - Переход D_i' –> D_{i+1}':
    - для каждой вершины 3 запроса LCP
    - на диагонали берём наиболее далёкую вершину
- Сложность O(k|T|)