String Matching

Pelatnas 2 TOKI 2015

William Gozali

Outline

- Motivasi
- Rabin-Karp
- Knuth-Morris-Pratt
- Latihan

Motivasi

- Diberikan string S dan T
- Cari semua kemunculan S pada T!

Motivasi

Contoh:

- S = “aaaba”
- T = “aabaaabaaaba”

Kemunculan:

- aabaaabaaaba
- aabaaabaaaba

Solusi 1: brute force

```
aabaaaabaaaba
aabaa
  abaaa
    baaab
      aaaba
        aabaa
          abaaa
            baaab
              aaaba
```

- Untuk setiap posisi yang mungkin, cek apakah substring dari T sama dengan S
- Kompleksitas: O(|S|*|T|)
- Tidak efisien karena banyak perbandingan yang diulang

Solusi 2: hashing

- Masalah pada solusi brute force: membandingkan string membutuhkan O(|S|)
- Bagaimana jika string di-hash menjadi angka?
- Perbandingan bisa menjadi O(1)!

Solusi 2: hashing

Misalkan hash(“aaaba”) = 871

```
aabaaaabaaaba     nilai hash:
aabaa             =>  123
 abaaa            =>   71
  baaab           =>   18
   aaaba          =>  871
    aabaa         =>  123
     abaaa        =>   71
      baaab       =>   18
       aaaba      =>  871
```

Solusi 2: hashing

- Pertanyaan selanjutnya: bagaimana melakukan hashing?

Formulasi hash

- Anggap string sebagai bilangan basis 26:
- aabc = $0*26^3 + 0*26^2 + 1*26^1 + 2*26^0$
- Membutuhkan O(|S|) untuk melakukan hash
- Jika ada |T| kemungkinan substring, kompleksitas untuk melakukan string matching tetap O(|S|*|T|)...

Rolling Hash

- Beruntungnya, kita bisa melakukan rolling hash!

  Perhatikan hubungan berikut:
- babcb = $1*26^3 + 0*26^2 + 1*26^1 + 2*26^0$
- babcb = $0*26^3 + 1*26^2 + 2*26^1 + 1*26^0$

  Misalkan S=”babcb”
- hash(S[0..3]) = X
- Maka hash(S[1..4]) = $26*(X - S[0]*26^3) + S[4]*26^0$

- Intinya: buang depan, tambah belakang!

Solusi 2: hashing => Rabin-Karp

- Inilah yang menjadi konsep Rabin-Karp
- Total kompleksitas menjadi O(|S| + |T|)

Isu yang perlu dipikirkan:

- Apakah nilai hash perlu di-modulo?
- Bagaimana jika ada hash collision?

Modulo atau tanpa-modulo

- Bila hanya butuh informasi apakah nilai hash-nya sama, tidak perlu di-modulo; biarkan saja overflow
- Meskipun overflow, nilai hashnya tetap sama jika kedua string memang sama
- Lagipula, modulo merupakan operasi yang berat. Menghindari modulo berarti melakukan constant optimization
- Jika perlu modulo, gunakan bilangan prima yang besar seperti: 10^9+7

Collision?

- Bila Anda paranoid, gunakan beberapa hash yang berbeda. Jika kedua string memiliki nilai yang sama untuk setiap nilai hash, kemungkinan kedua string itu sama sangat besar.
- Contoh fungsi hash lain yang bisa digunakan: anggap string sebagai bilangan basis 29, 31, atau 37, dsb.
- Sepengalaman, gunakan dua nilai hash saja sudah sangat aman.

Solusi 3: Knuth-Morris-Pratt

- Bermula dari ide brute force
- KMP menyadari bahwa banyak perbandingan yang tidak perlu
- Contoh: mencari string S="ababac" pada T="abababaca"

- abababaca
- ababac ← "aba" dicek
- ababac
- ababac ← "aba" kembali dicek lagi :(

Ide KMP

- KMP menghindari pengecekan berkali-kali dengan “teleport”

- abababaca
- ababac ← gagal pada karakter 'c'
- ~~ababac~~ ← dilewati
- ababac ← langsung “teleport” ke sini dan membandingkan b

  “aba” sudah dianggap match (dan memang benar)

Apa itu teleport??

Teleport...

- Untuk setiap karakter ke-i pada S, kita cari tahu border terbesar dari S[0..i-1]
- Border adalah proper prefix dan proper suffix dari suatu string yang sama
- Proper prefix: prefix dari suatu string S yang tidak sama dengan S
- Proper suffix: suffix dari suatu string S yang tidak sama dengan S

- Contoh:
- abacaba memiliki border terbesar sebanyak 3: aba**c**aba

Teleport...

- Dengan informasi border terbesar, kita bisa melewati perbandingan string yang sudah pernah dilakukan
- Contoh: S=”abacabad” dan T=”abacababaca”

```
abacababaca
abacabad          ← gagal pada karakter 'd'
   abacabad       ← langsung lompat & bandingkan 'c'
      abacabad    ← masih gagal, lompat & bandingkan 'b'
      abacabad    ← 'b' cocok! Lanjut bandingkan 'a'
```

Visualisasi

```
ababadababacababacababa
ababacababa
^
```

Visualisasi

```
ababadababacababacababa
ababacababa
  ^
```

*ketika karakter yang dibandingkan sama, penunjuk '^' akan maju. Kita namakan kejadian ini sebagai “maju”

Visualisasi

```
abababadababacababacababa
ababacababa
   ^
```

Visualisasi

```
ababadababacababacababa
ababacababa
    ^
```

Visualisasi

```
ababadababacababacababa
abababacababa
      ^
```

Visualisasi

```
ababadababacababacababa
ababacababa
      ^
```

Visualisasi

```
ababadababacababacababa
   ababacababa
        ^
```

*ketika karakter yang dibandingkan tidak sama, penunjuk '^' akan mundur (relatif terhadap string S).
Kita namakan kejadian ini sebagai “mundur”

Visualisasi

```
abababadababacababacababa
      ababacababa
        ^
```

Visualisasi

```
abababadababacababacababa
      ababacababa
      ^
```

Visualisasi

```
abababadababacababacababa
        abababacababa
        ^
```

Visualisasi

```
abababadababacababacababa
        ababacababa
          ^
```

Visualisasi

```
ababadababacababacababa
      ababacababa
      ^
```

Visualisasi

ababadababacababacababa

ababacababa

^

Visualisasi

```
abababadababacababacababa
    abababacababa
    ^
```

Visualisasi

ababadababacababacababa

ab**aba**cababa

^

Visualisasi

```
ababadababacababacababa
      ababacababa
      ^
```

Visualisasi

abababadababacababacababa

<u>a</u>bababa**a**baba

^

Visualisasi

ababadababacababacababa

<u>ab</u>ababac**ab**aba

^

Visualisasi

```
abababadababacababacababa
   ababacaba
          ^
```

Visualisasi

ababadababacababacababa

ababac**abab**a

^

Visualisasi

abababadababacababacababa

ababacababa

^

Match!

Visualisasi

```
ababadababacababacababa
                ab<u>a</u>b<b>a</b>cababa
                   ^
```

Visualisasi

```
ababadababacababacababa
                ababacababa
                      ^
```

Visualisasi

ababadababacababacababa

<u>a</u>ababac**a**baba

^

Visualisasi

```
abababadababacababacababa
                  ababacababa
                        ^
```

Visualisasi

```
abababadababacababacababa
                ababacabaaba
                        ^
```

Visualisasi

abababababacababacababa

ababac**ababa**

^

Visualisasi

abababababacababacababa

ababac**ababa**

^

Match!

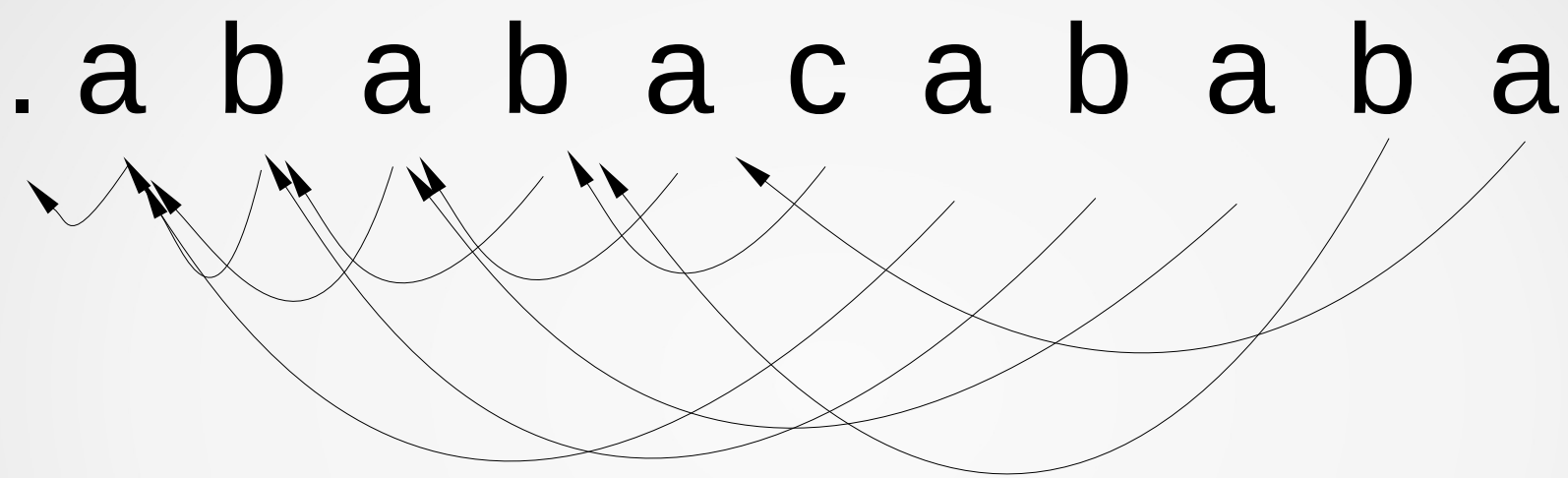
Teleport... (lanj.)

- Kita butuh informasi border terbesar untuk setiap prefix dari S

S=”ababacababa”

a
ab
aba
abab
ababa
ababac
ababaca
abababac**ab**
ababac**aba**
abababac**abab**
ababac**ababa**

Failure Function



- Bisa juga dinyatakan dalam diagam “failure”
- Panah dari karakter ke-i menuju karakter ke-j menyatakan “jika terjadi mismatch pada karakter ke-i dari S dengan suatu karakter ke-x dari T, mundur dan coba bandingkan karakter ke-j dari S dengan suatu karakter ke-x dari T”
- Pengecualian pada karakter pertama. Bila terjadi mismatch pada karakter pertama dari S dengan karakter ke-x dari T, mau tidak mau lanjut ke membandingkan karakter pertama dengan karakter ke-(x+1) dari T.

Kompleksitas

- Membangun tabel failure function dilakukan dalam O(|S|)
- Melakukan string matching dilakukan dalam O(|T|)
- Kompleksitas akhir: O(|S|+|T|)

- Mengapa string matching bisa dilakukan dalam O(|T|)?

Penjelasan sederhana

- Ada dua kemungkinan kejadian untuk setiap karakter pada T:
  1. Dibandingkan hanya satu kali dengan suatu karakter pada S. Kasus ini terjadi jika karakter pada T dan karakter pada S pada saat itu sama
  2. Dibandingkan beberapa kali dengan beberapa karakter pada S. Kasus ini terjadi jika karakter pada T dan karakter pada S saat itu berbeda

Penjelasan sederhana

- Total kejadian membandingkan pada kasus kedua untuk seluruh karakter pada T dibatasi oleh O(|T|)
- Banyaknya perbandingan yang dilakukan = total banyaknya mundur
- Total banyaknya mundur maksimal = banyaknya maju maksimal
- Banyaknya maju maksimal = O(|T|)
- Jadi total banyaknya perbandingan yang dilakukan

  = O(|T|)

Implementasi

```
char S[MAXN];
char T[MAXN];
int faf[MAXN];
int N, M;

void build(){
  int i,j;

  i = 0;
  j = -1;
  faf[0] = -1;
  while (i < N){
    while ((j >= 0) && (S[i] != S[j])) j = faf[j];

    i++;
    j++;
    faf[i] = j;
  }
}
```

Implementasi

```
int match(){
  int ret = 0;
  int i,j;

  i = 0;
  j = 0;
  while (i < M){
    while ((j >= 0) && (T[i] != S[j])) j = faf[j];

    i++;
    j++;
    if (j == N){
      printf("ditemukan di posisi %d\n", i - N);
      j = faf[j];
      ret++;
    }
  }
  return ret;
}
```

Implementasi

```
int main(){
  scanf("%s", T);
  scanf("%s", S);

  M = strlen(T);
  N = strlen(S);

  build();
  printf("%d\n", match());

  return 0;
}
```

Latihan 1

- Diberikan sebuah string S
- Cari substring terpanjang dari S yang muncul lebih dari 1 kali!

- Contoh: S=”banana”
- Jawaban: “ana”

Solusi

- Jika ada substring sepanjang L muncul lebih dari 1 kali, maka ada substring sepanjang L-1 pasti muncul lebih dari 1 kali juga
- Binary search panjang stringnya, lakukan rolling hash sambil mencatat nilai hash yang selama ini sudah ditemui
- Begitu ditemukan ada lebih dari 1 substring sama yang muncul lebih dari 1 kali, naikkan jawaban
- Jika tidak ditemukan substring sama yang muncul lebih dari 1 kali, turunkan jawaban
- Kompleksitas O(|S| log |S|)

Latihan 2

- Diberikan string S dan string T
- S dan T akan dikonkatenasi, tetapi beberapa karakter di suffix S dan prefix T yang sama boleh disatukan
- Tentukan banyaknya string hasil konkatenasi yang bisa dihasilkan!

- Contoh: S=”banana” T=”ananas”
- Jawaban: 4
  - b**anana**s
  - ban**ana**nas
  - banan**a**nanas
  - banana**ananas**

Solusi

- Gunakan KMP untuk melakukan string matching T ke S
- Setelah mencocokkan karakter pada T dengan karakter terakhir dari S, hitung banyaknya “mundur” yang bisa dilakukan dengan failure function sampai habis