Deteksi Plagiarisme Gambar menggunakan Algoritma Pencocokan Pola Rabin-Karp

Fadhil Imam Kurnia - 13515146
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
fadhilimamk@students.itb.ac.id

Absraksi – Plagiarisme dalam dunia akademik di Indonesia masih rawan dilakukan oleh mahasiswa dan dosen. Aksi plagiarisme di era digital tidak hanya terjadi dalam dokumen teks saja, namun aksi plagiarisme juga dapat ditemukan dalam video dan gambar. Tentu saja aksi plagiarisme dalam bentuk teks, gambar, ataupun video tidak dapat dibenarkan. Salah satu bentuk plagiarisme yang muncul di era digital adalah plagiarisme gambar atau foto. Dalam penulisan karya ilmiah misalnya, seseorang dapat mengambil gambar orang lain tanpa mencantumkan nama pemiliknya. Kasus ini juga dapat kita temui pada media sosial, seorang pengguna media sosial dapat mengunggah foto milik orang lain tanpa mencantumkan nama pemiliknya. Seorang plagiator dapat mengambil sebagian (*crop*) dari gambar orang lain untuk menghilangkan kesan plagiarisme. Untuk mendeteksi plagiarisme pada gambar digital kita dapat menggunakan algoritma Rabin-Karp dengan fungsi hash tertentu. Makalah ini membahas implementasi Algoritma Rabin-Karp untuk mendeteksi plagiarisme pada gambar digital. Gambar digital akan direpresentasikan sebagai matriks integer. Menggunakan matriks integer tersebut kita dapat mendeteksi apakah terdapat plagiarisme dalam suatu gambar.

Kata Kunci—plagiarisme, gambar, pattern matching, deteksi

I. PENDAHULUAN

Jika ditinjau dari segi bahasa, kata plagiarisme muncul pertama kali dalam bahasa Latin *plagiari(us)* yang berarti penculik, selain itu juga terdapat kata *plagi(um)* yang berarti menculik. Pada sekitar abad pertama masehi, seorang penyair Romawi, Marcus Valerius Martialis memperkenalkan kata tersebut kepada publik. Saat itu dia mengeluhkan puisi lain yang memiliki konten yang sama dengan puisi yang telah dibuatnya. Kemudian pada tahun 1601, kata Latin tersebut dimasukkan dalam bahasa Inggris oleh Ben Johnson menjadi *plagiarism*[5]. Kata tersebut kemudian diserap ke bahasa Indonesia menjadi plagiarisme yang dalam Kamus Besar Bahasa Indonesia[6] memiliki arti "penjiplakan yang melanggar hak cipta". Tindakan melakukan plagiarisme disebut plagiat, dan orang yang melakukannya disebuat plagiator.

Dalam dunia akademik di Indonesia, plagiarisme rawan dilakukan oleh mahasiswa dan dosen[4]. Beberapa waktu belakangan ini kita masih menjumpai beberapa berita terkait plagiarisme yang melibatkan mahasiswa bahkan dosen di beberapa perguruan tinggi terkemuka Indonesia. Dalam kesehariannya, seorang mahasiswa seringkali diharuskan untuk menulis lembar tugas secara berkelompok. Mahasiswa dapat mencari sumber literatur yang sesuai dengan tugasnya sebagai rujukan. Kerawanan plagiarisme terjadi karena banyak kelompok dengan tugas yang kurang lebih sama. Dapat terjadi tindakan saling *copy-paste* antar kelompok mahasiswa, bahkan yang lebih ekstrem adalah dengan mengganti nama mahasiswa saja tanpa sedikitpun mengubah kontennya. Hal tersebut tentu saja menyalahi integritas yang harus dijunjung tinggi oleh mahasiswa.

Pada era digital sekarang ini, aksi plagiarisme menjadi lebih rawan berkat adanya internet. Seseorang dapat dengan mudah menjiplak karya orang lain yang didapatnya dari internet tanpa mencantumkan nama pemilik karya tersebut. Sekarang ini aksi plagiarisme tidak hanya terjadi pada tulisan saja. Plagiarisme juga dapat muncul dalam gambar, musik hingga video. Tentu saja aksi plagiarisme dalam bentuk teks, gambar, ataupun video tidak dapat dibenarkan.

Gambar 1. Contoh gambar plagiarisme
(sumber: dokumen pribadi)

Salah satu bentuk plagiarisme yang muncul di era digital adalah plagiarisme gambar atau foto. Dalam penulisan karya ilmiah misalnya, seseorang dapat mengambil gambar orang lain tanpa mencantumkan nama pemiliknya. Kasus ini juga dapat kita temui pada media sosial, seorang pengguna media sosial dapat mengunggah foto milik orang lain tanpa mencantumkan nama pemiliknya. Seorang plagiator dapat mengambil sebagian (*crop*) dari gambar orang lain untuk menghilangkan kesan plagiarisme, contohnya dapat dilihat pada Gambar 1. Sekilas jika kita memperhatikan 2 gambar pada Gambar 1 keduanya tampak berbeda, namun jika diperhatikan secara seksama kita dapat menemukan bahwa gambar bawah merupakan bagian dari gambar atas. Hal ini dapat dilihat pada Gambar 2.

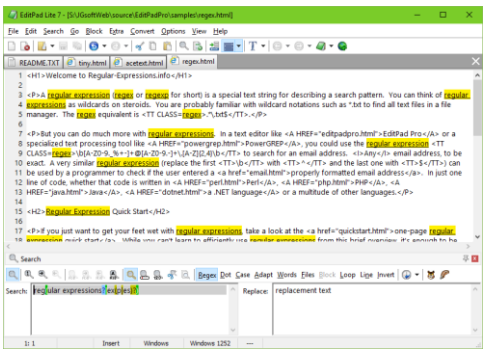
Gambar 2. Pengamatan terhadap Gambar 1, didapat kedua gambar pada Gambar 1 merupakan gambar yang sama
(sumber: dokumen pribadi)

Untungnya, kita dapat menggunakan bantuan komputer untuk mendeteksi gambar hasil plagiat. Kita dapat mendeteksi apakah suatu gambar terdapat dalam gambar lainnya dengan menggunakan algoritma pencocokan pola (*pattern matching*). Dalam makalah ini, penulis akan membahas permasalahan ini menggunakan algoritma Rabin-Karp yang memanfaatkan fungsi hash untuk mendeteksi apakah terdapat plagiarisme gambar dengan menemukan suatu gambar didalam gambar lainnya.

II. DASAR TEORI

A. Pencocokan Pola (*Pattern Matching*)

Pencocokan pola adalah sebuah metode untuk memeriksa dan menemukan pola tertentu dalam sebuah data yang lebih besar. Metode pencocokan pola ini berusaha menemukan urutan data yang sama persis dengan pola yang dimaksud. Dalam dunia nyata, metode ini banyak digunakan untuk melakukan pencarian kata atau kalimat (*string matching*), contohnya adalah dalam pencarian berita di internet, atau pencarian kalimat tertentu dalam editor teks. Namun dalam perkembangannya konsep pencocokan pola dan pencocokan *string* dapat juga digunakan untuk pencarian pola dalam audio, video, dan gambar. Contoh penggunaan pencocokan pola pada audio adalah untuk melakukan *speech recognition* yang dapat mengonversi suara menjadi teks.



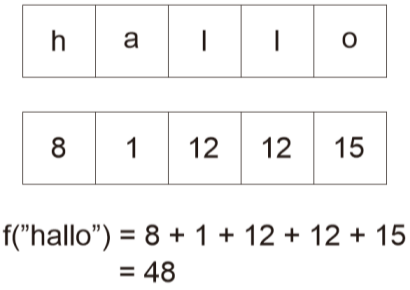
Gambar 3. Penggunaan pencocokan pola untuk mencari kalimat dalam editor teks (sumber: www. editpadlite.com)

Beberapa algortima yang dikenal untuk melakukan pencocokan *string* diantaranya adalah:

- Algortima *Brute Force*
- Algoritma Knuth-Morris-Pratt (KMP)
- Algoritma Boyer-Moore
- Algoritma Rabin-Karp
- Algoritma Z
- Algoritma Aho-Corasick

B. Fungsi Hash

Fungsi hash merupakan fungsi yang digunakan untuk memetakan sebuah masukan *string* atau angka yang panjangnya sembarang menjadi angka yang dapat digunakan untuk membedakan satu data dengan data lainnya. Contoh fungsi hash sederhana adalah fungsi yang menerima masukan string dan menghasilkan keluaran jumlah dari tiap karakter dalam string tersebut. Setiap karakter dalam string memiliki representasi angka khusus, contohnya a = 1, b =2, c=3, dan seterusnya. Misalnya kita memiliki string “hallo”, maka dengan fungsi hash tersebut akan dihasilkan 48.



Gambar 4. Ilustrasi penggunaan fungsi hash pada string “hallo”
(sumber: dokumen pribadi)

Pemanfaatan fungsi hash ini dapat dijumpai pada sistem keamanan web, dengan adanya fungsi hash sebuah pesan dapat dienkripsi sehingga hanya pengirim pesan dan penerima pesan

saja yang dapat mengetahui isi pesan tersebut. Selain itu fungsi hash juga digunakan dalam basis data untuk mempercepat proses pencarian data, fungsi hash digunakan untuk mengolah suatu masukan *string* dan menghasilkan keluaran indeks. Contohnya pada basis data yang menyimpan nama pengguna, kita dapat membuat fungsi hash yang menerima masukan nama dan memberikan keluaran berupa indeks sehingga proses pencarian data nama bisa menjadi lebih cepat.

C. Algoritma Rabin-Karp

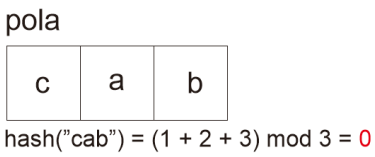
Algoritma Rabin-Karp dikemukakan oleh Michael O. Rabin dan Richard M. Karp sesuai dengan namanya. Secara umum algoritma ini digunakan untuk melakukan pencocokan pola, namun algoritma ini juga dapat digeneralisasi untuk melakukan pencocokan pola 2 dimensi. Algoritma ini menggunakan fungsi hash untuk menemukan pola tertentu dalam suatu data.

Pada pencocokan *string* algoritma ini memiliki kompleksitas O(n+m) untuk mencari *substring* dengan panjang m dalam *string* dengan panjang n, namun algoritma ini memiliki kasus terburuk dengan kompleksitas O(nm). Algoritma ini menggunakan fungsi hash sebagai pembanding antara pola *string* yang dicari dengan *substring* pada teks, sehingga kecepatan eksekusi algoritma ini juga ditentukan oleh fungsi hash yang digunakan.

Misalkan sebuah pola *string* dengan panjang m karakter akan dicari dalam teks dengan panjang n karakter. Apabila *substring* dan pola *string* memiliki hasil hash yang sama, maka akan dilakukan pengecekan lebih lanjut untuk setiap karakter. Namun, jika hasil keduanya tidak sama, maka substring akan digeser ke kanan. Pergeseran tersebut dapat dilakukan hingga sebanyak (n-m) kali. Perhitungan nilai hash yang efisien pada saat pergeseran akan mempengaruhi performa dari algoritma ini.

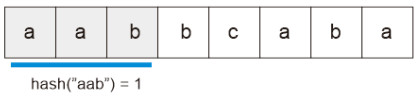
Contohnya kita akan mencari substring “cab” dalam teks “aabbcaba”. Fungsi hash yang digunakan misalnya akan menambahkan setiap karakter dalam alfabet dan melakukan modulo dengan 3. Karakter a = 1, b = 2, dan seterusnya. Langkah langkah pencariannya adalah sebagai berikut:

1. Tentukan nilai dari pola yang akan dicari



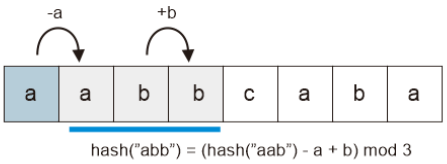
Didapatkan nilai dari pola yang akan dicari adalah 0.

2. Lakukan pengecekan hash pada *substring* pertama

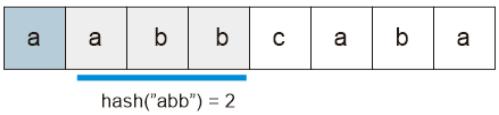


Didapatkan nilai dari *substring* awal adalah 1 yang tidak sama dengan nilai dari pola, maka *substring* digeser ke kanan sebanyak satu karakter.

3. Lakukan penghitungan hash untuk 3 karakter berikutnya

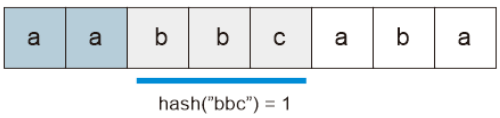


Untuk menghitung nilai hash saat ini, kita tidak perlu menghitung satu persatu lagi. Kita dapat menggunakan nilai hash 3 karakter sebelumnya untuk menghitung nilai hash saat ini. Caranya adalah dengan mengurangi nilai hash lama dengan karakter awal *substring* sebelumnya, kemudian dijumlahkan dengan karakter terakhir dari *substring* saat ini.



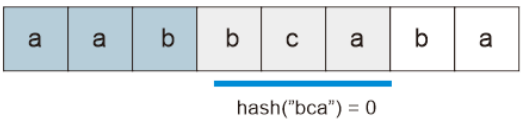
Menggunakan metode tersebut didapatkan nilai hash saat ini adalah 2. Karena nilai hashnya berbeda dengan pola maka geser substring ke kanan sebanyak 1 karakter.

4. Lakukan penghitungan hash untuk 3 karakter berikutnya

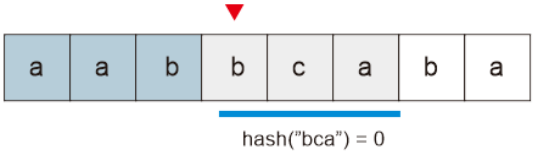


Didapatkan nilai dari *substring* saat ini adalah 1 yang tidak sama dengan nilai dari pola, maka *substring* digeser ke kanan sebanyak satu karakter.

5. Lakukan penghitungan hash untuk 3 karakter berikutnya

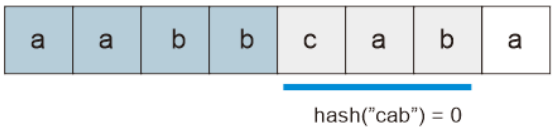


Didapatkan nilai dari *substring* saat ini adalah 0 dan sama dengan nilai dari pola, maka perlu dilakukan pengecekan satu per satu untuk tiap karakter.

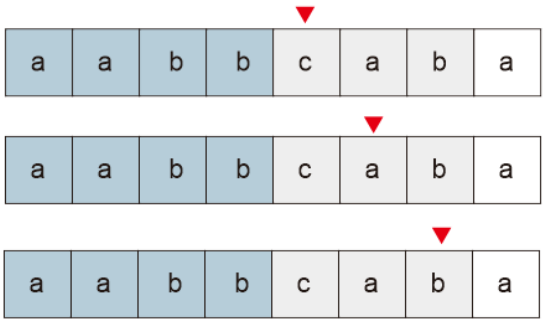


Terjadi *mismatch* pada karakter pertama, maka substring ini tidak sama dengan pola string yang dicari. Lanjutkan ke *substring* berikutnya, geser satu karakter ke kanan.

6. Lakukan penghitungan hash untuk 3 karakter berikutnya



Didapatkan nilai dari *substring* saat ini adalah 0 dan sama dengan nilai dari pola, maka perlu dilakukan pengecekan satu per satu untuk tiap karakter.



Setelah dilakukan pengecekan setiap karakter ternyata tidak ada *mismatch*, maka pola string yang dicari sudah ditemukan pada indeks ke-5.

7. Proses pencarian selesai, dan pola string ditemukan pada indeks 5.

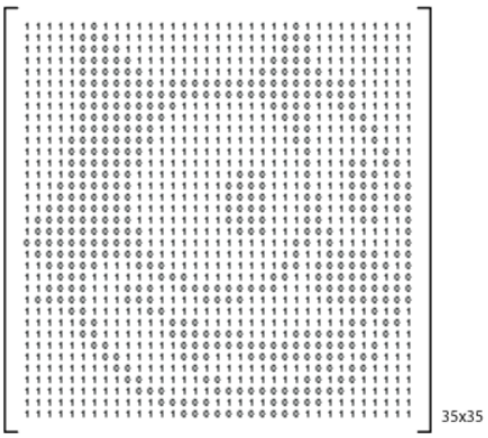
Seperti terlihat dalam contoh kasus, algoritma ini sangat bergantung pada fungsi hash yang digunakan. Pada contoh kasus sebelumnya fungsi hash yang digunakan tidak mengharuskan pengecekan satu persatu saat berganti substring, hal tersebut dapat membuat pencarian menjadi lebih cepat. Hal ini tentu saja berbeda dengan metode *brute force* yang mengharuskan pengecekan satu persatu untuk setiap substring.

D. Implementasi Gambar pada Komputer

Gambar yang biasa kita lihat di layar monitor dan foto yang kita ambil dengan *smartphone* adalah gambar digital. Dalam komputer, gambar dapat direpresentasikan menggunakan matriks. Implementasi paling sederhana dapat kita temukan pada *binary image* atau *boolean image* yang hanya terdiri dari warna hitam dan putih saja. Contoh gambar jenis *binary image* dapat dilihat pada Gambar 4.

Gambar 4. Contoh gambar *binary image* yang hanya terdiri dari warna hitam dan putih
(sumber: www.uff.br)

Gambar kepala kucing pada Gambar 4 tersebut dapat kita representasikan menggunakan matriks 0/1 yang berukuran 35 x 35. Angka 0 merepresentasikan warna hitam dan angka 1 merepresentasikan warna putih. Gambar dibawah ini menunjukan representasi Gambar 4 menggunakan matriks 0/1.

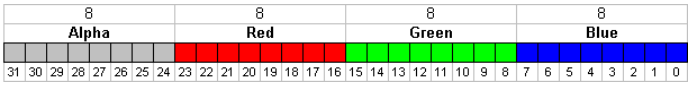


Gambar 5. Implementasi *binary image* dengan matriks 0/1
(sumber: www.uff.br)

Representasi gambar berwarna dalam komputer juga dapat direpresentasikan dengan matriks. Namun elemen dari matriks gambar berwarna ini bukan angka 0 dan 1 seperti pada *binary image* melainkan angka integer yang merepresentasikan pixel dalam gambar. Pixel merupakan satuan terkecil dalam gambar yang merepresentasikan warna tertentu, biasanya komponen dari pixel tersebut dibagi lagi menjadi 3 komponen yaitu Red, Green, dan Blue (RGB) yang merupakan warna dasar. Seiring

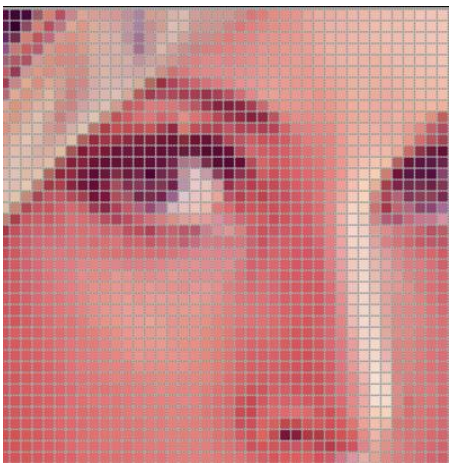
dengan berkembangnya teknologi gambar, sekarang ini pixel dapat dibagi menjadi 3 komponen yaitu Red, Green, Blue, dan Alpha (RGBA). Komponen Alpha dalam RGBA menandakan tingkat transparansi dari suatu warna dalam pixel.

Dalam komputer 32 bit kita dapat merepresentasikan RGBA dalam integer 32 bit. Jumlah bit dalam integer akan dibagi menjadi 4 bagian untuk menyimpan masing-masing komponen RGBA, sehingga untuk setiap komponen memiliki ukuran 8 bit. Oleh karena itu setiap komponen dalam RGBA memiliki batasan antara 0 – 225. Misalkan sebuah pixel dengan warna merah tidak transparan dapat direpresentasikan dengan R = 225, G = 0, B = 0, A = 0. Implementasi pixel RGBA dalam integer 32 bit dapat dilihat pada Gambar 6.



Gambar 6. Representasi warna gambar dalam integer 32 bit
(sumber: www.googglet.com)

Dengan adanya representasi warna menggunakan integer 32 bit kita dapat merepresentasikan gambar digital dengan menggunakan matriks yang elemennya adalah integer 32 bit. Contoh implementasi gambar digital berwarna dapat dilihat pada Gambar 7. Semakin besar ukuran dari suatu gambar maka ukuran matriks yang dibutuhkan juga menjadi lebih besar, mata manusia akan menanggap suatu gambar menjadi komtinu jika ukuran gambar tersebut besar



Gambar 7. Gambar yang direpresentasikan menggunakan matriks warna
(sumber: www.ashleymills.com)

III. Penggunaan Algortima Rabin-Karp pada Gambar

Algortima Rabin-Karp akan digunakan untuk menemukan pola pada gambar untuk mendeteksi plagiarisme. Gambar dan pola yang akan digunakan dalam algoritma ini dapat direpresentasikan sebagai matriks, sehingga untuk menyelesaikan permasalahan ini kita perlu menemukan sub-matriks di dalam matriks gambar. Jika terdapat sub-matriks dalam matriks gambar yang sama dengan matriks pola maka dapat diindikasikan terdapat plagiarisme.

Sebelumnya kita perlu menyesuaikan algortima Rabin-Karp yang digunakan agar dapat diimplementasikan pada matriks gambar. Beberapa hal yang perlu disesuaikan adalah fungsi hash yang digunakan dan teknik penggeseran sub-matriks.

A. *Fungi Hash untuk matriks*

Misalkan kita memiliki pola yang direpresentasikan dengan matriks berikut:

12	4	10
1	5	9
3	6	4

Fungsi hash sederhana yang dapat digunakan adalah dengan menjumlahkan nilai semua elemen lalu mencari modulonya dengan bilangan tertentu. Misalkan kita menjumlahkan semua elemen dan mencari modulonya dengan angka 100, maka untuk matriks diatas akan didapatkan nilai:

$f(M) = (12 + 4 + 10 + 1 + 5 + 9 + 3 + 6 + 4) \bmod 100$

$= \mathbf{54}$

B. *Pergeseran Submatriks*

Karena kita bekerja dalam 2 dimensi maka pergeseran matriks dapat dilakukan ke 4 arah, yaitu ke atas, bawah, kanan, dan kiri. Oleh karena itu untuk setiap arah pergeseran kita memerlukan metode penghitungan nilai hash yang efisien agar tidak perlu dilakukan penghitungan setiap elemen berkali-kali.

Misalkan kita memiliki submatriks M_1 seperti contoh sebelumnya, dan matriks gambar yang lebih besar seperti pada ilustrasi berikut:

12	4	10	32	84	12	61	6	30	79	46	44	39	36	38	18
1	5	9	11	36	60	80	86	65	78	36	71	41	86	41	94
3	6	4	55	7	74	58	22	66	25	48	98	39	49	88	2
4	50	58	92	30	1	8	13	66	92	34	10	55	54	54	0
33	24	94	73	7	48	15	19	84	10	89	88	14	36	20	51
1	89	32	0	41	35	67	29	92	89	47	22	31	47	21	36
68	65	39	72	62	45	93	19	25	53	79	81	74	34	35	2
92	54	19	61	6	10	21	74	77	79	2	80	28	12	6	59
4	74	16	79	32	43	73	96	90	86	89	59	7	33	16	95
29	79	42	46	2	72	39	9	68	28	50	99	63	89	5	24
37	47	82	86	35	64	64	22	15	91	25	62	15	65	92	51
76	28	69	72	20	74	93	29	13	78	25	48	89	35	47	56
27	62	93	23	97	45	98	95	28	86	58	97	35	31	16	83
12	74	80	43	93	72	75	66	94	32	89	30	82	60	95	22
6	90	39	88	58	21	15	60	29	71	73	82	28	48	6	22

Nilai hash submatriks saat ini adalah :

$f(M_1) = (12 + 4 + 10 + 1 + 5 + 9 + 3 + 6 + 4) \bmod 100$

$= \mathbf{54}$

Posisi submatriks saat ini sedang berada di ujung kiri atas. Jika submatriks tersebut digeser ke kanan sebanyak satu

kolom, kita dapat menghitung nilai hash submatriks yang baru dengan cara mengurangi nilai hash submatriks sebelumnya dengan seluruh eleman pada kolom pertama submatriks sebelumnya dan menjumlahkannya dengan seluruh elemen pada kolom terakhir dari submatriks yang baru. Untuk lebih jelasnya perhatikan ilustrasi berikut:

12	4	10	32	84	12	61	6	30	79	46	44	39	36	38	18
1	5	9	11	36	60	80	86	65	78	36	71	41	86	41	94
3	6	4	55	7	74	58	22	66	25	48	98	39	49	88	2
4	50	58	92	30	1	8	13	66	92	34	10	55	54	54	0
33	24	94	73	7	48	15	19	84	10	89	88	14	36	20	51
1	89	32	0	41	35	67	29	92	89	47	22	31	47	21	36
68	65	39	72	62	45	93	19	25	53	79	81	74	34	35	2
92	54	19	61	6	10	21	74	77	79	2	80	28	12	6	59
4	74	16	79	32	43	73	96	90	86	89	59	7	33	16	95
29	79	42	46	2	72	39	9	68	28	50	99	63	89	5	24
37	47	82	86	35	64	64	22	15	91	25	62	15	65	92	51
76	28	69	72	20	74	93	29	13	78	25	48	89	35	47	56
27	62	93	23	97	45	98	95	28	86	58	97	35	31	16	83
12	74	80	43	93	72	75	66	94	32	89	30	82	60	95	22
6	90	39	88	58	21	15	60	29	71	73	82	28	48	6	22

Misalkan M_1 digeser ke kanan sebanyak satu kolom menjadi M_2. Setelah submatriks digeser ke kanan maka kolom pertama yang berwarna biru bukan menjadi bagian dari submatriks, dan kolom ke-4 yang berwarna hijau kini menjadi bagian dari submatriks. Kita dapat menghitung nilai hash dari submatriks yang baru ini dengan memanfaatkan nilai hash submatriks sebelumnya.

$f(M_2) = (f(M_1) - \Sigma kolom\_sebelum + \Sigma kolom\_sesudah) \bmod 100$

$= (54 – (12 + 1 + 3) + (71 + 42 + 85)) \bmod 100$

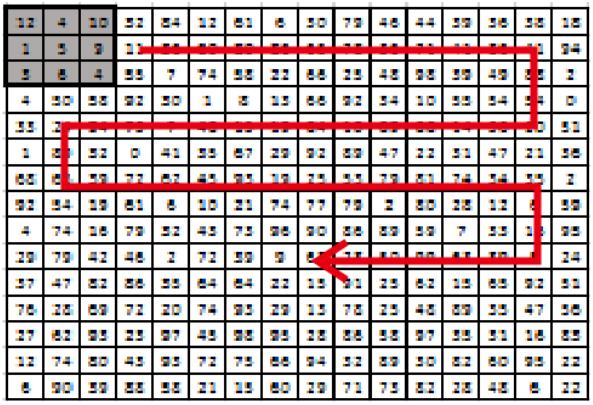
$= (54 – 16 + 98) \bmod 100$

$= 120 \bmod 100$

$= \mathbf{20}$

Metode ini dapat digunakan untuk pergeseran ke kanan, kiri, atas, dan bawah.

Dengan menggunakan metode tersebut kita dapat menghitung nilai hash submatriks yang baru tanpa harus menghitung ulang semua elemen dalam submatriks. Kemudian agar proses pencarian menjadi lebih efisien alur pergerakan submatriks juga perlu disesuaikan. Hal tersebut supaya untuk setiap pergeseran nilai hash yang baru dapat ditentukan dari nilai hash sebelumnya. Maka alur yang dapat digunakan adalah sebagai berikut:



Gambar 8. Alur jalannya proses pencarian dalam matriks menggunakan algoritma Rabin-Karp
(sumber: dokumen pribadi)

Alur bolak – balik seperti pada Gambar 8 membuat setiap pencarian nilai submatriks baru dapat menggunakan nilai submatriks sebelumnya. Submatriks yang digeser ke kanan atau kiri dapat menghitung nilai hash menggunakan nilai hash submatriks sebelumnya, lalu submatriks yang digeser ke bawah juga dapat menggunakan nilai hash submatriks sebelumnya (submatriks diatasnya) untuk menghitung nilai hash-nya.

Oleh karena itu kita dapat menentukan apakah suatu gambar terdapat didalam gambar lainnya dengan langkah sebagai berikut:

1. Konversi gambar pola yang akan dicari menjadi matriks integer
2. Konversi gambar tempat pencarian menjadi matriks integer
3. Tentukan nilai hash gambar pola
4. Hitung nilai hash submatriks
5. Jika nilai hash submatriks berbeda dengan nilai hash pola maka geser submatriks:
    a. Jika berada di kolom paling ujung pada gambar dan bukan pergeseran pertama maka geser ke bawah
    b. Jika berada dalam baris genap maka geser ke kanan
    c. Jika berada dalam baris ganjil maka geser ke kiri

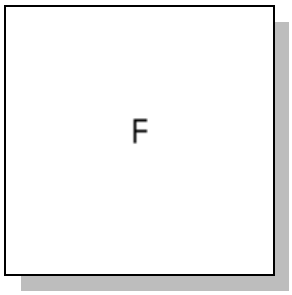
    Lalu kembali ke langkah 4.
6. Jika nilai hash submatriks sama dengan nilai hash pola maka lakukan pengecekan setiap elemen pada submatriks.
    a. Jika ada elemen yang berbeda maka geser submatriks sama dengan langkah 5.
    b. Jika semua elemen sama maka proses pencarian selesai

Menggunakan metode ini untuk mengecek gambar berukuran k x k dalam gambar N x N melibatkan k^2 proses untuk pertama kali mgnhitung nilai hash pola dan submatriks pertama. Kemudian untuk menghitung nilai hash yang baru terdapat 2k proses, penghitungan ini dilakukan sebanyak $(N-k)^2$ kali. Lalu jika ditemukan submatriks dengan nilai hash sama maka dioerlukan k^2 proses untuk memastikannya. Sehingga secara keseluruhan terdapat sekitar $3k^2 + 2k(N-k)^2$ proses. Oleh karena itu untuk gambar pola yang jauh lebih kecil dibandingkan dengan gambar yang ingin diperiksa, maka kompleksitas algoritma ini adalah $O(N^2)$.

Untuk membuktikan konsep ini penulis juga telah membuat program sederhana menggunakan bahasa Java. *Siurce code* dapat diakses melalui https://github.com/fadhilimamk/matcher Hasil eksekusi program tersebut adalah sebagai berikut:

Kasus Uji 1

Gambar uji :



Gambar yang dicurigai :

Hasil :

Menemukan pola pada :42 45

Kasus Uji 2

Gambar uji :

Gambar yang dicurigai :

Hasil :

Menemukan pola pada :47 934

Kasus Uji 3

Gambar uji :

Gambar yang dicurigai :

Hasil :

Menemukan pola pada : 354 1767

IV. SIMPULAN

Algoritma pencocokan pola (*pattern matching*) Rabin-Karp dapat digunakan untuk mendeteksi plagiarisme dalam gambar digital. Gambar digital perlu direpresentaskan menjadi matriks dengan elemen integer untuk dapat diproses dengan algoritma ini. Elemen integer dalam matriks tersebut merepresentasikan warna RGBA yang dimiliki oleh pixel dalam gambar. Elemen integer dalam matriks akan digunakan oleh fungsi hash untuk menghitung nilai setiap submatriks. Fungsi hash yang digunakan juga perlu disesuaikan agar penghitungan nilai hash untuk submatriks berikutnya dapat menjadi efisien.

Penerapan algoritma ini sangat berguna untuk mendeteksi plagiarisme dalam gambar digital. Kedepannya algoritma ini dapat dikembangkan lebih lanjut, misalkan dengan melakukan pemrosesan gambar terlebih dahulu sebelum melakukan *pattern matching*, hal tersebut dapat memperluas jangkauan pendeteksian. Selain itu algoritma ini juga dapat dikembangkan agar dapat mendeteksi plagiarisme parsial.

V. UCAPAN TERIMA KASIH

Penulis mengucapkan syukur kepada Tuhan Yang Maha Esa karena dengan rahmatNya penulis dapat menyelesaikan makalah ini dengan tuntas. Penulis juga berterimakasih kepada orang tua penulis yang selalu mendukung kegiatan perkuliahan penulis hingga semester ini usai, baik dengan dukungan doa maupun dengan dukungan materi. Penulis juga mengucapkan banyak terimakasih kepada Ibu Nur Ulfa Maulidevi, sebagai dosen Mata Kuliah Startegi Algoritma yang telah mengajarkan berbagai pengetahuan kepada penulis, termasuk didalamnya mengenai *pattern matching* yang penulis gunakan dalam makalah ini. Penulis juga mengucapkan terimakasih kepada teman-teman penulis yang mendukung dan memberi saran dalam penyusunan makalah ini.

REFERENSI

[1] Munir, Rinaldi, Diktat Kuliah IF2211 Strategi Algoritma. Bandung: Program Studi Teknik Informatika Intitut Teknologi Bandung, 2009

[2] Puntambekar, A.A. Analysis of Algorithm and Design. Pune : Technical Publication, 2007

[3] https://visualgo.net/ . Diakses pada 13 Mei 2017.

[4] Plagiarisme, Runtuhnya Tembok Kejujuran Akademik Herqutanto Departemen Ilmu Kedokteran Komunitas Fakultas Kedokteran Universitas Indonesia, Jakarta

[5] Suryono IAS. Plagiarisme dalam penulisan ilmiah. Majalah Kedokteran Indonesia. 2011;61(5):1

[6] Kamus Besar Bahasa Indonesia

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 15 Mei 2017

Fadhil Imam Kurnia
13515146