Penerapan Algoritma A* dalam Penyelesaian Permainan Hue Color Sorting

Bagas Sambega Rosyada - 13522071
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): bagassambega@gmail.com

***Abstrak*—Algoritma penentuan rute digunakan untuk menentukan rute terbaik untuk mencapai suatu tujuan atau *state* tertentu. Algoritma penentuan rute dapat digunakan untuk menentukan langkah optimal yang dibutuhkan untuk mengurutkan sebuah matriks warna pada permainan Hue Color Sorting. Salah satu algoritma penentuan rute yang sering digunakan adalah algoritma A*. Makalah ini menyajikan prinsip dasar dan implementasi algoritma A* dalam menyelesaikan permainan Hue Color Sorting secara optimal. Dengan adanya makalah ini diharapkan dapat membantu penulis dan pembaca memahami implementasi algoritma A* dalam pengurutan sebuah matriks warna pada permainan Hue Color Sorting.**

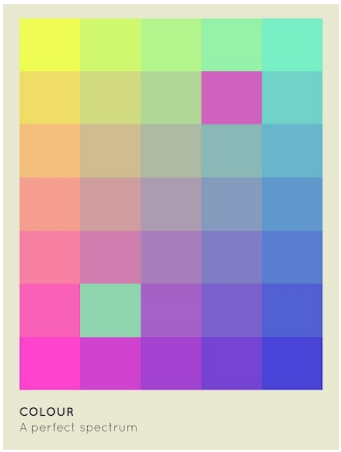
Kata kunci—gradasi warna, graf, matriks, optimasi, warna RGB

I. PENDAHULUAN

Permainan Hue Color Sorting adalah permainan pengurutan warna pada sebuah matriks dengan memindahkan atau menukar dua buah kotak warna hingga terbentuk sebuah gradasi warna. Permainan dimulai dengan diberikan sebuah matriks berukuran *n x m* dengan berisi beberapa kotak warna teracak, dengan beberapa kotak warna diposisikan *fixed* atau tidak dapat diubah/ditukar posisinya. Kotak-kotak warna laiinnya yang dapat ditukar posisinya harus disusun sedemikian rupa sehingga menyusun gradasi warna yang bersesuaian dengan kotak-kotak yang tidak dapat diubah posisinya (*fixed*) tersebut.

Penentuan kotak warna mana yang akan ditukar dapat menentukan bentuk matriks kotak warna yang baru. Bentuk matriks yang baru akan merepresentasikan suatu status yang baru dan untuk seluruh susunan matriks yang ada, akan terbentuk suatu pohon ruang status yang merupakan salah satu bentuk graf. Graf tersebut dapat membentuk sebuah rute untuk menyelesaikan permainan ini dengan mencari langkah penukaran kotak yang dilakukan dari simpul awal ke simpul tujuan. Representasi pencarian langkah yang diambil dalam penentuan rute penyelesaian permainan ini dapat menggunakan algoritma penentuan rute.

Salah satu algoritma penentuan rute yang menjamin rute yang dihasilkan minimum global adalah algoritma A*. Algoritma ini mengkombinasikan nilai heuristik dan juga biaya yang diperlukan untuk mencapai simpul tujuan, sehingga memperkecil lingkup pencarian rute menuju simpul tujuan.

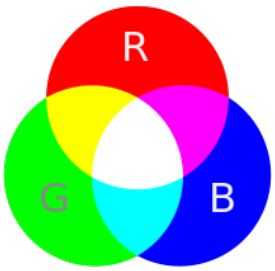


Gambar 1. Salah satu permainan Hue Color Sorting di Google Play Store, I Love Hue. Diambil dari https://play.google.com/store/apps/details?id=com.zutgames.ilovehue

II. LANDASAN TEORI

A. Model Warna RGB

Sebuah warna dapat direpresentasikan dalam sebuah kombinasi nilai *Red, Green,* dan *Blue* atau biasa disingkat RGB. Nilai RGB merupakan sebuah *tuple* yang berisi kombinasi tiga warna merah, hijau, dan biru, dan nilai untuk setiap warna berada pada rentang 0-255 (00 – FF). Sebagai contoh, warna merah menyala bernilai RGB(255, 0, 0) dan warna ungu bernilai RGB(160, 32, 240).



Gambar 2. Model warna RGB. Diambil dari https://en.wikipedia.org/wiki/RGB_color_model

Model warna RGB banyak digunakan pada perangkat elektronik untuk video/gambar *output*, di mana setiap piksel (penyusun terkecil dari *output* gambar/video) tersusun dari sebuah nilai RGB. Sebuah gambar dapat direpresentasikan sebagai sebuah matriks warna RGB, dengan satu sel matriks menyatakan sebuah piksel. [1]

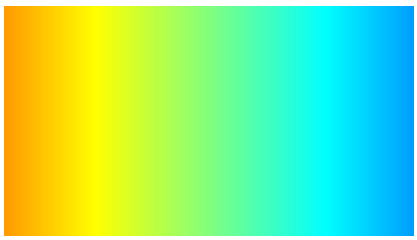
B. Gradasi Warna

Gradasi warna adalah tingkatan, susunan, derajat atau peningkatan, peralihan suatu warna menuju warna lain. Gradasi juga sering disebut sebagai gradien warna, karena menunjukkan rasio perubahan antara sebuah warna ke warna lainnya secara bertahap [2]. Gradasi warna antara dua warna dapat dibentuk dengan membuat beberapa kombinasi warna yang berada pada jangkauan warna asal dengan warna tujuan. Semakin besar jarak dari warna tujuan ke warna asal, semakin banyak juga kombinasi gradasi warna yang dapat dibentuk.

Untuk menentukan jarak dari sebuah warna ke warna lainnya, dapat digunakan rumus Euclidean:

$$\|C_1 - C_2\| = \sqrt{(C_{1,R} - C_{2,R})^2 + (C_{1,G} - C_{2,G})^2 + (C_{1,B} - C_{2,B})^2} \quad (1)$$

dengan C_1 merupakan warna asal dan C_2 merupakan warna tujuan [3].



Gambar 3. Contoh gradasi warna merah ke biru. Diambil dari https://dev.to/ndesmic/linear-color-gradients-from-scratch-1a0e

Sebuah warna termasuk ke dalam gradasi di antara warna C_1 dan C_2 jika seluruh nilai R, G, dan B masuk ke jangkauan kedua nilai R, G, dan B di antara C_1 dan C_2. Sebagai contoh diketahui C_1 = RGB(240, 10, 0) dan C_2 = RGB(120, 50, 80), maka C_3 = RGB(180, 15, 70) termasuk ke dalam jangkauan gradasi $C_1 - C_2$, dan C_4 = RGB(255, 10, 70) tidak termasuk ke dalam jangkauan gradasi $C_1 - C_2$.

C. Graf dan Penentuan Rute

Graf adalah representasi dari objek diskrit dan hubungan antarobjek tersebut [4]. Graf tersusun atas simpul sebagai objek diskrit dan sisi sebagai hubungan antarobjek. Salah satu permasalahan yang dapat direpresentasikan dalam bentuk graf adalah penentuan rute dari suatu simpul bercabang ke simpul tujuan.

Graf secara umum terbagi menjadi 2 berdasarkan bentuk saat pencarian rute berlangsung:

1. Graf statis, graf yang sudah terbentuk sejak awal sebelum proses pencarian dimulai
2. Graf dinamis, graf yang terus dibangun selama proses pencarian solusi berlangsung. Graf berbentuk pohon ruang status yang akan terus berkembang seiring proses pencarian berlangsung, merepresentasikan kemungkinan-kemungkinan status saat ini yang sudah dilalui untuk mencapai solusi.

Penentuan rute dengan biaya dan langkah minimum memerlukan suatu algoritma penentuan rute. Beberapa algoritma yang sering digunakan dalam pencarian rute adalah algoritma *Breadth-First Search* (BFS), *Depth-First Search, Iterative Deepening Search* (IDS), *Uniform Cost Search* (UCS), *Greedy Best First-Search*, dan A*.

D. Algoritma A*

Algoritma A* merupakan salah satu algoritma penentuan rute yang menggabungkan pendekatan berdasarkan nilai heuristik dan juga biaya (*cost*) untuk mencapai *state* atau kondisi saat itu [4]. Algoritma A* disebut sebagai *informed search*, karena menggunakan sebuah informasi mengenai masalah atau bagaimana cara menuju ke tujuan penyelesaian, dalam hal ini adalah nilai fungsi heuristik itu sendiri.

Tujuan utama dari penggunaan algoritma A* adalah menghindari sebuah simpul atau rute yang sudah memiliki biaya yang mahal, dan diperkirakan lebih jauh untuk mencapai tujuan. Algoritma ini juga bisa disebut sebagai gabungan algoritma *Uniform Cost Search* dengan prioritas *cost* terendah (*g(n)*) dan algoritma *Greedy Best First Search* dengan prioritas nilai heuristik terendah (*h(n)*). Kombinasi dari nilai biaya dan nilai heuristik akan menentukan simpul prioritas yang akan dibangkitkan selanjutnya, dengan nilai prioritas

$$f(n) = g(n) + h(n). \quad (2)$$

Semakin kecil nilai *f(n)* pada simpul, maka prioritas simpul tersebut untuk dituju semakin besar.

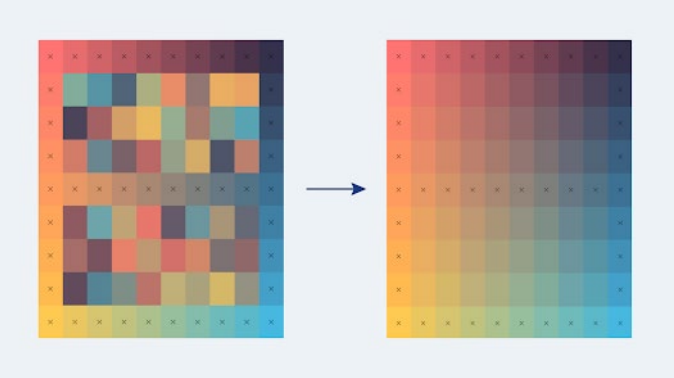
Algoritma penyelesaian penentuan rute pada algoritma A* adalah sebagai berikut [5],

1. Inisialisasi sebuah *priority queue* kosong, dan kunjungi simpul awal.
2. Bangkitkan seluruh simpul kemungkinan yang dapat dituju dari simpul awal.
3. Masukkan simpul-simpul tersebut ke sebuah *priority queue* berdasarkan prioritas nilai $f(n) = g(n) + h(n)$ terkecil, di mana *g(n)* adalah *cost* dan *h(n)* adalah nilai heuristik.
4. Selama *priority queue* tidak kosong atau solusi belum ditemukan, kunjungi satu per satu simpul di *priority queue* dan hapus simpul yang sudah dikunjungi dari *priority queue*. Bangkitkan simpul-simpul yang mungkin dituju dari simpul tersebut dan masukkan ke *priority queue* berdasarkan *f(n)* terkecil.
5. Jika solusi ditemukan, hentikan proses pencarian dan simpan rute dari simpul awal ke simpul akhir.

E. Representasi Hue Color Sorting pada Graf

Permainan Hue Color Sorting adalah permainan pengurutan warna pada matriks berukuran *m x n* sehingga menciptakan gradasi warna pada matriks tersebut. Matriks warna yang ada pada permainan terdiri dari sel/kotak warna yang dapat ditukar posisinya, dan sel/kotak warna yang tidak dapat dipindah

posisinya (*fixed*) dan menjadi patokan gradasi yang akan dibentuk. Matriks warna yang diberikan tidak berurutan pada awalnya, dan pemain perlu menyusun setiap sel-sel warna tersebut dengan cara menukar posisi antara 2 kotak warna yang dapat dipindahkan sehingga menyusun gradasi warma di antara kotak-kotak *fixed* tersebut.



Gambar 4. Ilustrasi penyusunan matriks warna acak menjadi matriks warna yang tersusun berdasarkan gradasi. Diambil dari https://play.google.com/store/apps/details?id=com.color.puzzle.i.love.hue.blendoku.game&hl=en_US

Pada implementasi penyelesaian permainan Hue Color Sorting ini, diberikan batasan matriks warna yang digunakan akan berukuran *n x n*, dan seluruh kotak *fixed* akan berada di pojok kiri-kanan atas-bawah dari matriks, sehingga total kotak yang tidak bisa dimodifikasi berjumlah 4, yaitu pada indeks *(0, n-1), (0,0), (n-1, 0),* dan *(n-1, n-1).* Seluruh kotak lainnya akan menjadi *movable boxes* atau kotak yang bisa ditukar posisinya dengan *movable boxes* lainnya.

Setiap sel/kotak warna pada matriks merupakan sebuah nilai *tuple* RGB. Setiap terjadi penukaran antara dua kotak, sebuah matriks baru akan terbentuk dengan susunan kotak yang berbeda dengan sebelumnya. Susunan-susunan yang berbeda tersebut dapat direpresentasikan sebagai sebuah pohon ruang status, dengan simpul-simpulnya adalah susunan matriks warna yang berbeda. Simpul yang bersisian dengan simpul lainnya adalah matriks yang telah dilakukan satu kali penukaran kotak warna pada matriks tersebut. Pencarian simpul selanjutnya yang akan dituju menggunakan sebuah fungsi pembangkitan simpul. Algoritma pembangkitan simpul pada terdapat pada *pseudocode* berikut,

```
def generate_successors(node:Node)->list[Node]:
      successors <- []
      n <- panjang matriks pada Node
      for I in range(n):
         for j in range(n):
              jika (I, j) merupakan kotak fixes:
                  skip
              for k in range(n):
                  for l in range(n):
                      jika (k, l) merupakan kotak fixes:
skip
                      new <- copy matriks of Node
                      swap(new[i][j], new[k],[l])
                      append new to successors
return successors
```

III. IMPLEMENTASI ALGORITMA

Program penyelesaian permainan Hue Color Sort menjadi sebuah matriks warna RGB terurut memerlukan beberapa tahap,

1. Konversi Gambar ke Matriks Warna RGB
2. Pendefinisian Fungsi *Cost* dan Heuristik
3. Implementasi Algoritma dan Struktur Data
4. Konversi Matriks Warna RGB ke Gambar Hasil

A. Tahap 1: Konversi Gambar ke Matriks Warna RGB

Tahap penyelesaian dimulai dengan merepresentasikan sebuah gambar matriks warna menjadi struktur data matriks yang berisi *tuple* nilai RGB. Dengan menggunakan *library* Image pada Python, sebuah gambar dapat direpresentasikan dalam bentuk matriks RGB untuk setiap pikselnya. Gambar matriks warna RGB yang digunakan akan terdiri dari *n x n* macam warna, dan setiap kotak/sel warna akan berukuran *p x p*. Algoritma pembentukan matriks warna RGB menggunakan bahasa Python adalah sebagai berikut,

```
image = Image.open(file)
image = image.convert('RGB')
n = image.width // size
rgb_matrix = [[image.getpixel((j * p, I * p)) for j in
range(n)] for I in range(n)]
return rgb_matrix
```

B. Tahap 2: Pendefinisian Fungsi Cost dan Heuristik

Algoritma A* memerlukan fungsi heuristik dan *cost* untuk menentukan prioritas pembangkitan simpul selanjutnya. Fungsi *cost* pada pencarian permainan Hue Color Sort adalah banyaknya langkah yang sudah dilakukan untuk mencapai ruang status saat ini atau bentuk susunan matriks saat ini.

Sebuah matriks warna akan semakin dekat dengan matriks tujuan jika memenuhi 2 kriteria berikut,

1. Jarak Euclidean I antara sel warna *I, j* terhadap salah satu pojok (*fixed box*) lebih kecil dibandingkan jarak dari salah satu pojok ke pojok lain yang berada di baris/kolom yang sama dengan sel warna *I, j*.

$$\begin{cases} \|C_{n,j} - C_{0,j}\| > \|C_{n,j} - C_{i,j}\|, pada\ satu\ baris\ yang\ sama \\ \|C_{i,n} - C_{i,0}\| > \|C_{i,n} - C_{i,j}\|, pada\ satu\ kolom\ yang\ sama \end{cases} \quad (3)$$

2. Sel warna *I, j* memiliki pola keterurutan yang sama dengan pola baris dan/atau kolomnya. Sebagai contoh pada baris ke-*I*, dari kolom paling kiri ke kolom paling kanan, warna merah semakin pekat (nilai R semakin besar ke kanan), maka jika di baris tersebut ada suatu nilai R yang tidak terurut membesar atau lebih kecil dari sel di kirinya, maka kotak tersebut harus ditukar dengan kotak lainnya.

Penghitungan nilai heuristik yaitu berupa banyaknya sel yang melanggar 2 ketentuan di atas pada matriks. Jika seluruh sel/kotak yang ada pada matriks memenuhi 2 kriteria di atas, maka nilai heuristiknya adalah 0 dan menjadi matriks tujuan. Berdasarkan kalimat sebelumnya, dapat diimplikasikan bahwa matriks tujuan sebenarnya tidak diketahui sejak awal dan

matriks tujuan hanya didefinisikan jika nilai heuristik matriks adalah 0. Hal ini untuk merealisasikan kondisi sebenarnya dari permainan yang belum selesai, yaitu mencari solusi permainan dengan kondisi akhir permainan belum diketahui. Semakin kecil nilai heuristik, semakin dekat matriks ke penyelesaian solusi. Algoritma penghitungan nilai heuristik untuk setiap simpul dilampirkan pada *pseudocode* berikut,

```
def calculate_heuristic(node)->int:
v_pattern <- generate vertical pattern
h_pattern <- generate horizontal pattern
invalid_pos <- 0
for item in row:
 if kriteria1(item) not valid then increment invalid_pos
 else {kriteria1 still valid}:
  if kriteria2(item, v_pattern) not valid then increment
invalid_pos
for item in col:
 if kriteria1(item) not valid then increment invalid_pos
 else {kriteria1 still valid}:
  if kriteria2(item, h_pattern) not valid then increment
invalid_pos
-> invalid_pos
```

Penggunaan kriteria pertama tersebut sebagai fungsi heuristik didasarkan pada batasan yang telah ditentukan bahwa setiap pojok matriks merupakan kotak yang tidak dapat dipindahkan dan menjadi patokan urutan gradasi warna secara vertikal dan horizontal. Namun terdapat suatu kondisi di mana kriteria pertama tetap bernilai benar, namun matriks yang dihasilkan salah. Hal ini karena di antara setiap pojokan, tidak selalu terdapat kondisi di mana sel warna yang ada hanya 1 buah saja atau hanya 1 buah sel saja yang tidak valid. Sebagai contoh di sebuah kolom ke-0 dengan 4 sel, berurutan terdapat RGB(0,0,0), RGB(40,0,0), RGB(20,0,0), dan RGB(60,0,0). Jarak Euclidean dari $C_1 - C_4$ dan $C_2 - C_4$ memang benar kurang dari $C_0 - C_4$, namun urutan dari kedua sel tersebut salah, karena adanya inkonsistensi keterurutan nilai R sehingga tidak dihasilkan gradasi yang diinginkan. Oleh karenanya digunakan kriteria 2 untuk melengkapi kekurangan pada kriteria 1.

Sementara itu kriteria 2 memerlukan kriteria 1 untuk memastikan bahwa setiap ujung dari matriks dapat terurut dengan benar, karena kriteria 2 memerlukan keterurutan nilai-nilai pada sisi-sisi ujung matriks terlebih dahulu agar dapat ditemukan pola menaik/menurun dari kolom/baris. Pola menaik/menurun dari suatu kolom/baris ditentukan dengan menghitung selisih antara ujung atas-ujung bawah dan ujung kiri-ujung kanan, sehingga jika urutan di ujung matriks masih salah, pola akan salah.

Gambar	Representasi Matriks Warna
	[[(232, 150, 139), (180, 162, 150), (140, 131, 160), (94, 122, 170)], [(229, 171, 134), (186, 140, 150), (133, 153, 162), (84, 145, 176)], [(225, 193, 132), (175, 185, 148), (124, 176, 164), (74, 167, 182)], [(222, 216, 128), (169, 207, 148), (116, 199, 169), (64, 190, 187)]]

Sebagai contoh pada figur di atas, terdapat sebuah matriks warna yang tidak memiliki keterurutan pada $kotak_{1,1}$ dan pada $kotak_{0,1}$. Pada $kotak_{0,2}$ terdapat ketidakurutan pada nilai Green pada baris 0 (horizontal), yaitu pola nilai Green yang seharusnya menurun (Pojok kiri atas RGB(232, 150, 139) – pojok kanan atas RGB(94, 122, 170)), namun nilai Green pada $kotak_{0,1}$ malah lebih menaik dari $kotak_{0,0}$. Selain itu, terdapat pula ketidaksesuaian pada $kotak_{1,1}$ di mana keterurutan dan jarak Euclidean pada sisi vertikal dan horizontalnya tidak sesuai, yaitu pada baris ke-1 (horizontal) nilai Green lebih tinggi dari sel di kirinya, padahal pola Green adalah menurun, dan pada kolomnya (vertikal) nilai Green seharusnya menaik, namun nilainya menurun. Dengan ketidaksesuaian pada 2 sel ini, maka nilai heuristiknya adalah 2.

C. Tahap 3: Implementasi Algoritma dan Struktur Data

Setelah fungsi heuristik dan *cost* terdefinisi, program perlu menyimpan nilai heuristik dan *cost* untuk setiap matriks yang ada pada pohon ruang status. Struktur data yang dibentuk adalah sebuah kelas Node dengan atribut:

- data: Matriks pada simpul
- cost: Nilai *cost* matriks data
- heuristic: Nilai heuristik matriks data
- parent: Simpul *parent* dari Node saat ini
- fn: *cost* + *heuristic*

Langkah-langkah penyelesaian permainan memerlukan suatu *priority queue* untuk menyimpan simpul yang dibangkitkan dan diurutkan berdasarkan nilai *fn*, dan suatu *set* yang menyimpan seluruh matriks yang sudah pernah dikunjungi sebelumnya untuk menghindari redundansi dan *loop* tidak berhingga. Langkah-langkah penyelesaian permainan menggunakan algoritma A* sebagai berikut,

1. Inisialisasi sebuah *priority queue* kosong dan sebuah *set* kosong, dan inisialisasi sebuah Node berisi matriks warna RGB yang sudah dikonversi dari gambar, dengan nilai *cost* = 0 dan *parent* = null, dan hitung nilai *heuristic*-nya.
2. Masukkan Node pertama ke *priority queue* dan periksa apakah *heuristic* = 0, jika tidak bangkitkan simpul selanjutnya dan hapus Node saat ini dari *priority queue*, dan masukkan Node saat ini ke *set*.
3. Urutkan Node yang sudah dibangkitkan sebelumnya dan masukkan ke *priority queue* secara terurut berdasarkan nilai *fn* terkecil.
4. Selama *priority queue* tidak kosong atau solusi belum ditemukan, ambil Node pertama dari *priority queue* dan jadikan sebagai Node saat ini (current_node) dan hapus Node dari *priority queue*.
5. Periksa apakah Node saat ini merupakan matriks solusi (*heuristic* = 0). Jika Node merupakan matriks solusi, kembalikan matriks solusi dan keluar dari program. Jika tidak, bangkitkan kembali simpul selanjutnya dan masukkan Node saat ini ke *set*. Ulangi langkah 3.
6. Jika matriks solusi ditemukan, lakukan *backtrack* pada Node dan simpan pertukaran-pertukaran kotak yang dilakukan pada rute solusi.

Di Python, ketiadaan *priority queue* dan *set* dapat diakali dengan membuat sebuah *array of Node* biasa dan mengurutkan seluruh isi dari *array* berdasarkan nilai *fn* setiap Node-nya, dan *set* dengan menambahkan matriks yang pernah diperiksa ke dalam *set*. Implementasi algoritma A* pada program Python terlampir sebagai berikut,

```
def solve(matrix, fixed_position):
    pq = Node(matrix, 0, None, [])
    print(pq.heuristic)
    if pq.heuristic == 0:
        return pq.data, []
    priority_queue:list[Node] = [pq]

    while len(priority_queue) > 0:
        priority_queue = sort_by_cost(priority_queue)
        current_node = priority_queue.pop(0)
        current_state = current_node.data

        if current_node.heuristic == 0:
            changes = []
            while current_node.parent is not None:
                changes += current_node.changes
                current_node = current_node.parent
            changes.reverse()
            return current_state, changes

        if current_state in visited:
            continue
        visited.append(current_state)

        successors = generate_successors(current_node,
fixed_position)
        for successor in successors:
            if successor.data not in visited:
                priority_queue.append(successor)

    return None, None
```

D. Tahap 4: Konversi Matriks Warna RGB ke Gambar Hasil

Setelah matriks solusi ditemukan dan sudah terurut menjadi gradasi warna yang sesuai, matriks warna perlu dikonversi kembali menjadi gambar hasil. *Library* Image di Python menyediakan fungsi untuk melakukan konversi matriks RGB ke gambar, diimplementasikan dalam fungsi berikut,

```
n = len(rgb_matrix)
if os.path.exists(output_path):
    os.remove(output_path)
image_size = n * box_size
image = Image.new('RGB', (image_size, image_size),
'white')
draw = ImageDraw.Draw(image)
for I in range(n):
    for j in range(n):
      color = rgb_matrix[i][j]
      top_left = (j * box_size, I * box_size)
      bottom_right = ((j + 1) * box_size, (I + 1) *
box_size)
draw.rectangle([top_left,bottom_right],fill=color)
image.save("../test/" + output_path)
```

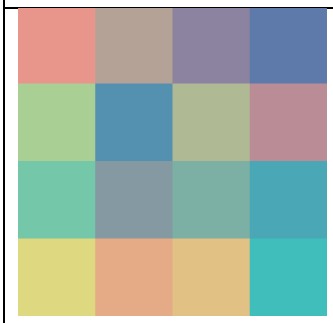
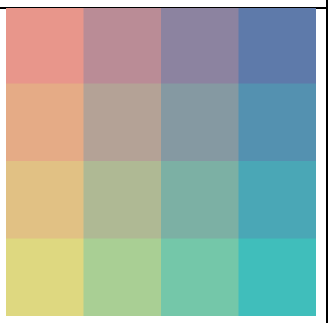
IV. Pengujian Dan Analisis

A. Pengujian

Pengujian dilakukan dengan menyiapkan sebuah matriks warna yang sudah terurut dan melakukan pertukaran kolom sebanyak *n* (perubahan asli) kali. Setelah menjadi matriks warna acak, matriks akan diuji menggunakan algoritma untuk membentuk matriks yang sama dengan matriks terurut awal.

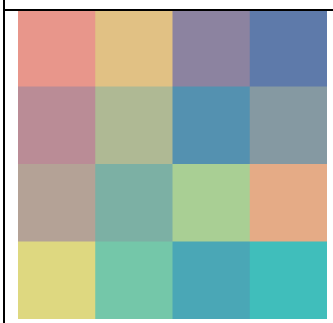
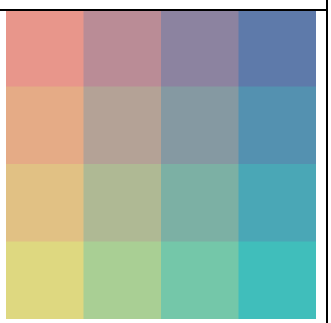
Program akan menghitung banyak langkah yang diperlukan untuk mencapai matriks tujuan, banyaknya simpul yang diperiksa, dan pertukaran kotak-kotak warna yang dilakukan. Asumsi yang digunakan dalam proses pengujian adalah seluruh warna pada matriks warna selalu valid dan dapat dibentuk menjadi matriks gradasi warna yang valid, matriks selalu berukuran *n x n*, dan kolom di posisi *fixed* memiliki nilai RGB yang valid dan menjadi patokan arah gradasi warna.

1. Pengujian 1

Kondisi Awal	Kondisi Target/Akhir
	

Statistik	Pertukaran simpul
Banyak perubahan asli dilakukan: 7 Ukuran matriks: 4 x 4 Banyak langkah: 7 Banyak simpul diperiksa: 9 Waktu pemrosesan: 0.06857514381408691 sekon	(3, 1) – (2, 0) (1, 3) – (1, 1) (3, 2) – (1, 0) (1, 1) – (0, 1) (2, 0) – (1, 0) (2, 1) – (1, 2) (3, 2) – (3, 1)

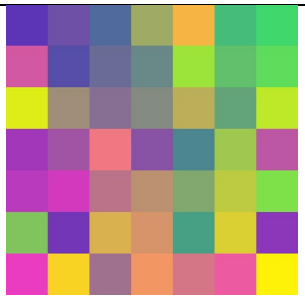
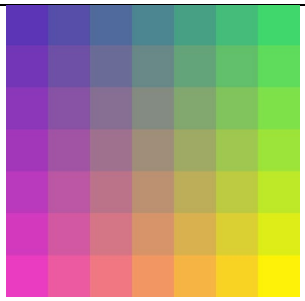
2. Pengujian 2

Kondisi Awal	Kondisi Target/Akhir
	

Statistik	Pertukaran Kotak
Banyak perubahan asli dilakukan: 9 Ukuran matriks: 4 x 4 Banyak langkah: 9	(1, 0) – (0, 1) (2, 0) – (1, 1) (1, 3) – (1, 2) (3, 2) – (2, 3) (3, 2) – (1, 0) (2, 2) – (2, 0)

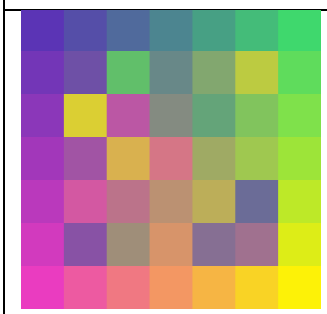
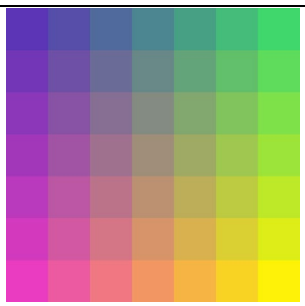
Banyak simpul diperiksa: 37 Waktu pemrosesan: 0.36939406394958496 sekon	(3, 2) – (2, 0) (2, 2) – (2, 1) (3, 2) – (3, 1)

3. Pengujian 3

Kondisi Awal	Kondisi Target/Akhir
	

Statistik	Pertukaran Kotak
Banyak perubahan asli dilakukan: 22 Ukuran matriks : 7 x 7 Banyak langkah: 22 Banyak simpul diperiksa: 28 Waktu pemrosesan: 9.93862771987915 sekon	(5, 6) – (2, 0) (3, 4) – (0, 3) (4, 5) – (3, 6) (5, 1) – (1, 0) (6, 4) – (3, 1) (1, 1) – (0, 1) (2, 5) – (1, 4) (6, 2) – (2, 1) (5, 0) – (4, 1) (4, 5) – (4, 1) (6, 5) – (6, 1) (6, 4) – (0, 4) (5, 4) – (0, 4) (5, 4) – (5, 2) (6, 2) – (3, 2) (3, 3) – (3, 2) (5, 2) – (3, 1) (3, 6) – (2, 5) (3, 2) – (2, 1) (4, 6) – (2, 6) (4, 4) – (2, 4) (4, 5) – (2, 5)

4. Pengujian 4

Kondisi Awal	Kondisi Target/Akhir
	

Ukuran matriks : 7 x 7 Banyak langkah: 10 Banyak simpul diperiksa: 10 Waktu pemrosesan: 2.458953380584717 sekon	(5, 2) – (3, 3) (2, 4) – (1, 4) (1, 5) – (1, 2) (4, 5) – (1, 2) (5, 1) – (2, 1) (5, 4) – (3, 2) (3, 2) – (2, 2) (5, 5) – (3, 2) (5, 5) – (5, 1) (5, 1) – (4, 1)

B. Analisis dan Pembahasan

Berdasarkan 4 pengujian di atas terhadap pola yang berbeda untuk matriks berukuran 4 x 4 dan 7 x 7, keseluruhan kasus dapat diselesaikan dengan menggunakan algoritma A* dengan fungsi heuristik dan fungsi *cost* yang telah didefinisikan sebelumnya. Seluruh kasus juga memiliki hasil yang optimal secara global karena jumlah langkah yang dibutuhkan untuk mencapai matriks warna terurut adalah sama dengan jumlah perubahan pengacakan kolom warna.

Jumlah simpul yang dibangkitkan untuk setiap kunjungan ke Node adalah sebanyak

$$v = n \; x \; (n - 1) \qquad (4)$$

dengan n adalah banyak kotak yang dapat dipindahkan/ditukar posisinya. Namun karena *priority queue* sebagai wadah penyimpanan Node untuk dikunjungi selanjutnya sudah terurut berdasarkan nilai $f(n) = g(n) + h(n)$ (heuristik + *cost*), maka kemungkinan jumlah kunjungan ke simpul-simpul mencapai nilai v menjadi kecil karena simpul yang memiliki nilai heuristik sangat besar atau *cost* sangat besar akan diprioritaskan di paling akhir.

Setiap pembangkitan simpul dilakukan, dilakukan pertukaran untuk seluruh simpul di baris I dan kolom j, terhadap simpul di baris k dan kolom l, sehingga dengan total 4 *loop* dan pada setiap *loop* dibentuk sebuah matriks baru ($O(n^2)$), maka kompleksitas algoritma pembangkitan matriks adalah $O(n^6)$. Penghitungan nilai heuristik untuk setiap Node memiliki operasi perbandingan terhadap pola pada sumbu horizontal dan vertikal masing-masing dengan kompleksitas $O(3n^2)$ karena masing-masing nilai R, G, dan B juga perlu dibandingkan, dan penghitungan jarak Euclidean untuk setiap kotak dengan kompleksitas $O(n^2)$.

Kompleksitas algoritma A* secara umum untuk kasus terburuk adalah $O(b^d)$, dengan b adalah *branching factor* dan d adalah kedalaman dari pohon status ruang. Kompleksitas untuk setiap operasi yang tinggi inilah yang menyebabkan program berjalan dalam waktu yang lama, seperti pada kasus 4 dengan waktu pemrosesan hampir mencapai 10 detik, sementara banyak simpul yang diperiksa hanyalah 28 simpul.

V. KESIMPULAN

Algoritma A* dapat digunakan untuk menyelesaikan permainan Hue Color Sorting dengan langkah minimum. Algoritma A* yang digunakan mengkombinasikan nilai *cost* yaitu banyak langkah yang diperlukan untuk mencapai kondisi matriks saat itu dihitung dari kondisi matriks awal, dan nilai heuristik berdasarkan dua kriteria, yaitu jarak Euclidean suatu

kotak warna ke pojok matriks, dan keterurutan kotak warna berdasarkan pola vertikal-horizontal setiap kolom dan baris matriks. Meskipun menghasilkan solusi yang optimal, namun kompleksitas algoritma masih cukup tinggi karena banyaknya operasi perbandingan dan kalkulasi untuk setiap sel di dalam matriks warna.

LAMPIRAN

- Link YouTube: https://youtu.be/vsTmvzyGSaw
- Link Github untuk *source code*: https://github.com/bagassambega/HueColorSorting

UCAPAN TERIMA KASIH

Rasa syukur penulis haturkan ke hadirat Allah Swt., atas ridha-Nya penulis dapat menyelesaikan makalah ini dalam waktu yang sudah ditentukan. Penulis juga mengucapkan terima kasih kepada Dr. Ir. Rinaldi Munir, M.T. selaku dosen Strategi Algoritma K-01 tahun ajaran 2023/2024. Penulis juga mengucapkan terima kasih kepada keluarga, teman-teman, dan pihak-pihak lain atas seluruh kontribusi yang membantu penulis menyelesaikan makalah ini.

DAFTAR PUSTAKA

[1] "RGB colour model | Description, Development, Uses, Science, & Facts | Britannica." Diakses: 12 Juni 2024. [Daring]. Tersedia pada: https://www.britannica.com/science/RGB-colour-model

[2] "Gradient Logos: Why You Need These Controversial Colors - Tailor Brands." Diakses: 12 Juni 2024. [Daring]. Tersedia pada: https://www.tailorbrands.com/blog/gradient-logos

[3] "Colour metric." Diakses: 12 Juni 2024. [Daring]. Tersedia pada: https://www.compuphase.com/cmetric.htm

[4] "Website Rinaldi Munir." Diakses: 12 Juni 2024. [Daring]. Tersedia pada: https://informatika.stei.itb.ac.id/~rinaldi.munir/

[5] "A* Search Algorithm - GeeksforGeeks." Diakses: 12 Juni 2024. [Daring]. Tersedia pada: https://www.geeksforgeeks.org/a-search-algorithm/

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, 12 Juni 2024

Bagas Sambega Rosyada - 13522071