Visualisasi Algoritma *Divide and Conquer* Melalui Implementasi Permasalahan Papan Catur Defektif

Richard Rivaldo 13519185[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13519185@std.stei.itb.ac.id

***Abstrak*—Pemecahan suatu permasalahan yang cukup rumit akan membutuhkan sebuah algoritma tertentu untuk mendekati solusi permasalahan tersebut. Salah satu kelompok algoritma yang bisa digunakan adalah Algoritma *Divide and Conquer*. Contoh permasalahan yang bisa dipecahkan dengan algoritma ini adalah permasalahan Papan Catur Defektif. Namun, implementasi algoritma ini biasanya dilakukan secara rekursif sehingga algoritma ini tergolong cukup sulit untuk dibuat dan di-*debug* ketika diprogram. Oleh karena itu, selain menganalisis mengenai permasalahan tersebut, penulis juga akan mengimplementasikan visualisasi pemecahan permasalahan Papan Catur Defektif dalam bahasa pemrograman di dalam makalah ini.**

***Kata Kunci*—Algoritma, *Divide and Conquer*, Papan Catur Defektif, Visualisasi.**

I. PENDAHULUAN

Pada zaman dahulu, peperangan sudah menjadi sebuah penampakan yang sering terjadi. Tidak peduli apakah alasannya adalah memperebutkan suatu wilayah tertentu, menampilkan kekuatan dan dominasi militer, atau bahkan hanya sebatas konsekuensi atas saling singgung-menyinggung antara kedua belah pihak. Peperangan memberikan dampak yang sangat besar, tidak hanya kepada pihak yang kalah, tetapi juga tentunya terhadap pihak yang menang dan mengorbankan banyak jiwa untuk mendapatkan kejayaan tersebut.

Saat ini, peperangan telah berakhir dan dunia telah memasuki era baru yang ditopang oleh keberadaan teknologi. Namun demikian, masih banyak sekali orang yang adrenalinnya terpacu ketika mendengarkan kata `perang` hingga saat ini. Hal tersebut bisa terlihat dari banyaknya *game digital* yang mengambil tema peperangan, dan tentunya *game* tersebut laris manis di pasar. Tentunya, sejak dulu kala, para komandan militer dan strategis juga sangat membutuhkan sebuah cara untuk melakukan simulasi peperangan. Tidak mungkin bagi mereka memerintahkan prajurit mereka untuk melakukan *trial* peperangan secara nyata yang akan menghabiskan *resources* mereka, bukan? Oleh karena itu, dibutuhkan sebuah permainan yang bisa digunakan untuk menghidupkan imajinasi dan menyalurkan strategi para jenius pada saat itu.

Checkmate! Ya, salah satu permainan yang bisa digunakan untuk memenuhi kebutuhan simulasi tersebut adalah `Catur` atau yang kerap disebut juga dengan *Chess.* Sejak zaman India Kuno, banyak kekuatan militer yang menggunakan Catur sebagai media metafora untuk melaksanakan `peperangan` di atas sebuah papan dua warna tersebut. Tidak hanya itu, bahkan sampai saat ini Catur masih sering digunakan untuk melakukan pelatihan dan pembuatan strategi dalam menghadapi peperangan.

Sekarang, Catur masih menjadi permainan papan yang paling sering dimainkan oleh banyak orang di berbagai belahan dunia. Sudah banyak sekali perlombaan Catur internasional yang juga melibatkan banyak sekali *Grandmaster,* sebutan untuk pemain catur profesional dengan pengalaman yang mumpuni. Dalam masa pandemi seperti inipun, perlombaan Catur tetap dihelat secara *online* melalui berbagai *platform* Catur *digital* yang ada.

Dengan bantuan teknologi, Catur menjadi semakin terkenal dan bahkan banyak dimainkan di kalangan remaja pada saat ini. Dalam *Computer Science* sendiri, permainan Catur terlihat sangat berkembang, terutama ketika kita melihat perkembangan Kecerdasan Buatan seperti *Stockfish* dan *AlphaZero.* Uniknya, banyak sekali permasalahan Catur yang bisa dijadikan sebagai masalah komputasi dan matematika dalam dunia Informatika.

Salah satu permasalahan matematis yang menarik adalah Permasalahan Papan Catur Defektif atau *Defective Chessboard Problem*. Permasalahan yang juga sering disebut dengan *Tiling Chessboard Problem* ini merupakan permasalahan di mana salah satu kotak pada permainan Catur rusak dan tidak bisa dimainkan lagi. Papan Catur tersebut perlu dipasangkan kotak-kotak berukuran L sedemikian sehingga semua permukaannya tertutupi dengan sempurna, kecuali kotak yang hilang tadi.

Gambar 1. Tampilan Permainan Catur *Digital* pada Situs *Chess.com*
(Sumber: https://betacssjs.chesscomfiles.com/bundles/web/images/offline-play/standardboard.png)

Ternyata, permasalahan tersebut bisa diselesaikan dengan menggunakan salah satu kelompok algoritma yang ada di dalam *Computer Science*. Algoritma yang dimaksud adalah Algoritma *Divide and Conquer*. Proses pemecahan permasalahan tersebut bisa diabstraksikan dengan menggunakan pendekatan rekursif. Tentunya, pendekatan yang demikian cukup sukar untuk dimengerti sehingga pendekatan visual bisa menjadi salah satu pilihan yang cocok ketika memecahkan persoalan tersebut.

II. Landasan Teori

A. *Algoritma Divide and Conquer*

Berbicara mengenai *Divide and Conquer*, biasanya hal yang paling diingat adalah sejarah kelam dan menyakitkan yang dimiliki bangsa Indonesia karena penjajahan oleh Hindia Belanda selama kurang lebih 350 tahun. Pada mulanya, banyak kerajaan-kerajaan Indonesia yang dipecah-belah oleh strategi militer Belanda dengan mengadudomba kedua pihak. Akibatnya, Indonesia yang pada saat itu masih sangat kuat menjadi lemah seketika dan mudah diperalat oleh penjajah. Strategi yang demikian dinamakan *Devide et Impera*, yang artinya memecah untuk menguasai.

Gambar 2. Ilustrasi Strategi *Devide et Impera*
(*Sumber: https://ibtimes.id/wp-content/uploads/2020/06/IMG_20200609_141252.jpg*)

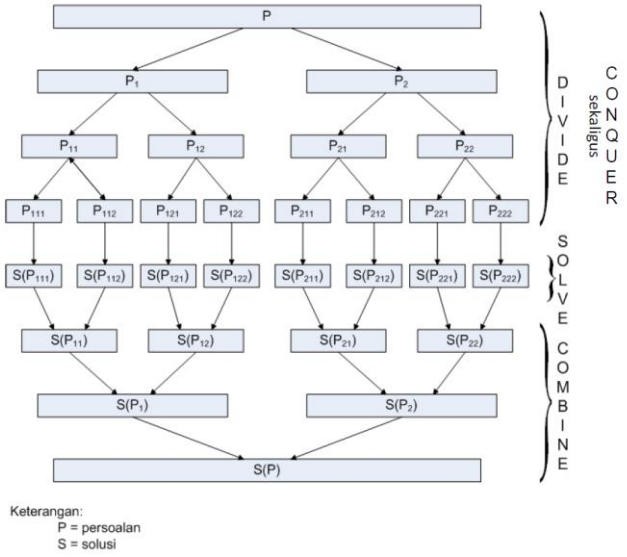
Konsep yang serupa juga digunakan oleh banyak penggagas dalam bidang Ilmu Komputer untuk menyelesaikan sebuah permasalahan yang sulit untuk dipecahkan secara langsung. Untuk itu, diciptakanlah sebuah strategi algoritma fundamental yang baru untuk melakukan pendekatan suatu permasalahan atas dorongan dari adanya strategi militer tersebut, yaitu algoritma *Divide and Conquer*.

Algoritma *Divide* dan *Conquer* merupakan algoritma pemecahan suatu permasalahan komputasi dengan cara membagi-bagi suatu persoalan yang besar ke dalam bagian-bagian yang kecil agar bisa dipecahkan dengan mudah. Lagi, bisa dilihat betapa miripnya strategi ini dengan strategi militer yang telah disebutkan sebelumnya.

Secara konseptual, kata *Divide* dalam nama algoritma ini merujuk pada tahap dalam membagi persoalan menjadi beberapa upa-persoalan. Sub-masalah ini memiliki kemiripan dengan persoalan semula namun dengan ukuran yang lebih kecil. Adapun kata *Conquer* di sini merujuk pada tahapan pemecahan masalah (*Solve*) yang berusaha menyelesaikan masing-masing upa-persoalan tersebut sesuai dengan ukuran yang dimilikinya.

Jika upa-persoalan masih berukuran cukup besar dan tidak bisa dipecahkan secara langsung, maka upa-persoalan akan dipecahkan secara rekursif, dan tentunya, berimplikasi pada pembagian upa-persoalan tersebut menjadi bagian-bagian yang lebih kecil lagi. Jika sebaliknya, maka upa-persoalan tersebut akan diselesaikan secara langsung karena ukurannya sudah dianggap cukup kecil.

Selain kedua tahap tersebut, ada tahap terakhir yang diperlukan dalam algoritma ini, yaitu tahap *Combine*. Ketika semua upa-persoalan telah berhasil diselesaikan, maka diperlukan adanya penggabungan solusi masing-masing upa-persoalan kembali. Dengan demikian, didapatkan solusi persoalan semula secara utuh.



Gambar 3. Ilustrasi Konsep Algoritma *Divide and Conquer*
(*Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian1.pdf*)

Objek persoalan algoritma ini yang bisa dibagi misalnya tabel atau larik, eksponen, polinom, dan matriks yang merupakan kumpulan tabel. Dalam hal ini, tiap upa-persoalan yang dibagi memiliki karakteristik yang sama dengan karakteristik persoalan semula. Dengan demikian, maka algoritma ini lebih natural dan cocok diungkapkan dalam skema rekursif.

```
procedure DIVIDEandCONQUER(input P : problem, n : integer)
{ Menyelesaikan persoalan P dengan algoritma divide and conquer
  Masukan: masukan persoalan P berukuran n
  Luaran: solusi dari persoalan semula  }
Deklarasi
   r : integer

Algoritma
  if n ≤ n0 then  {ukuran persoalan P sudah cukup kecil }
     SOLVE  persoalan P yang berukuran n ini
  else
     DIVIDE  menjadi r upa-persoalan, P1, P2, ..., Pr , yang masing-masing berukuran n1, n2, ..., nr
     for  masing-masing P1, P2, ..., Pr , do
          DIVIDEandCONQUER(Pi,  ni)
     endfor
     COMBINE  solusi dari P1, P2, ..., Pr  menjadi solusi persoalan semula
  endif
```

Gambar 4. Skema Rekursif Umum *Divide and Conquer*
(*Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian1.pdf*)

Untuk skema yang demikian, maka kompleksitas waktu algoritma *Divide and Conquer* yang demikian dapat dituliskan sebagai berikut.

$$T(n) = \begin{cases} g(n), & n \le n_0 \\ T(n_1) + T(n_2) \dots + T(n_r) + f(n), & n > n_0 \end{cases}$$

Dalam kompleksitas waktu algoritma *T(n)*, *g(n)* merupakan kompleksitas waktu yang diperlukan jika ternyata persoalan yang diberikan merupakan basis rekursi dan bisa dipecahkan secara langsung (*solve*). Adapun $T(n_i)$ merupakan kompleksitas waktu yang dimiliki oleh setiap upa-persoalan *i* dan *f(n)* merupakan kompleksitas waktu penggabungan semua solusi upa-persoalan yang ada.

Jika semua upa-persoalan dibagi menjadi dua bagian yang berukuran sama, maka skema rekursif yang dimiliki oleh algoritma ini akan berubah, begitu juga dengan pembagian ke dalam *n* bagian yang sama. Pembagian yang demikian akan menghasilkan kompleksitas waktu dan skema rekursif sebagai berikut.

```
procedure DIVIDEandCONQUER(input P : problem, n : integer)
{ Menyelesaikan persoalan dengan algoritma divide and conquer
  Masukan: masukan yang berukuran n
  Luaran: solusi dari persoalan semula
}
Deklarasi
   r : integer

Algoritma
 if n ≤ n0 then {ukuran persoalan sudah cukup kecil }
     SOLVE persoalan P yang berukuran n ini
 else
     DIVIDE menjadi 2 upa-persoalan, P1 dan P2, masing-masing berukuran n/2
     DIVIDEandCONQUER(P1, n/2)
     DIVIDEandCONQUER(P2, n/2)
     COMBINE solusi dari P1 dan P2
 endif
```

Gambar 5. Skema Rekursif Hasil Modifikasi
(*Sumber:* https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian1.pdf)

$$T(n) = \begin{cases} g(n), & n \le n_0 \\ 2T(n/2) + f(n), & n > n_0 \end{cases}$$

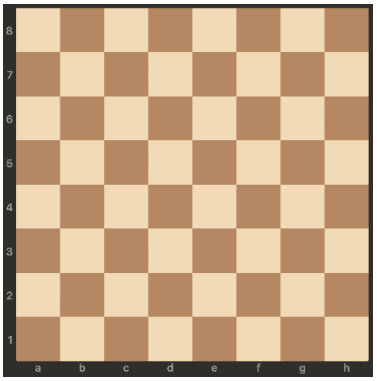
Beberapa permasalahan yang bisa dipecahkan dengan menggunakan algoritma *Divide and Conquer* antara lain adalah persoalan mencari nilai maksimum dan minimum, menghitung perpangkatan, algoritma pengurutan seperti *Quicksort* dan *Mergesort*, *Closest Pair Problem*, *Convex Hull*, perkalian matriks, bilangan bulat besar, atau perkalian dua buah polinom.

Selain algoritma *Divide and Conquer*, ada algoritma lain dengan konsep yang mirip, yaitu Algoritma *Decrease and Conquer*. *Decrease and Conquer* adalah metode perancangan algoritma yang dilakukan dengan mereduksi persoalan menjadi dua upa-persoalan yang lebih kecil, tetapi selanjutnya hanya akan melakukan pemrosesan terhadap sebuah sub-persoalan saja. Dengan kemiripan ini, algoritma ini sering digabungkan dan kemudian dipisahkan karena perbedaan mendasar tersebut.

B. *Papan Catur*

Permainan Catur dimainkan pada sebuah papan yang memiliki dua buah warna. Dahulu kala, hanya ada dua buah warna utama yang digunakan pada papan catur, yaitu hitam dan putih. Karena terlihat monoton, maka pada saat ini sudah bisa dilihat banyak sekali pasangan warna papan catur.

Papan catur biasanya berukuran 8 × 8, dan setiap bloknya ditandai dengan pasangan huruf dan angka, dari A sampai dengan H, serta dari 1 sampai dengan 8. Dengan demikian, ada 64 blok yang ada di papan catur normal.



Gambar 6. Ilustrasi Papan Catur
(*Sumber:* https://i.stack.imgur.com/s8XND.png)

C. *Triomino*

Triomino dalam permainan legendaris Tetris, adalah sebutan untuk blok yang berukuran tiga kotak, bukan empat seperti yang dimiliki oleh *Tetromino*. Dalam hal ini, *Triomino* memiliki banyak bentuk dan orientasi, misalnya ada *Triomino* yang berbentuk seperti huruf I, ataupun ada *Triomino* yang berbentuk seperti huruf L, dan terkadang juga disebut sebagai *Triomino* V atau J.

Gambar 7. *Triomino* L dengan Berbagai Orientasi
(*Sumber:* https://i.stack.imgur.com/s8XND.png)

D. *Teorema Master*

Teorema Master merupakan sebuah teorema yang bisa digunakan untuk menentukan notasi asimptotik dari algoritma yang kompleksitas waktunya berada dalam bentuk rekurens. Dalam hal ini, Teorema Master memudahkan penyelesaian tersebut, karena dengan Teorema Master tidak diperlukan lagi penyelesaian secara iteratif. Isi dari Teorema Master adalah sebagai berikut.

Misalkan bahwa kompleksitas waktu sebuah algoritma, yaitu $T(n)$, adalah sebuah fungsi rekurens monoton menaik yang jika digambarkan sebagai relasi rekurens berbentuk seperti persamaan berikut.

$$T(n) = aT\left(\frac{n}{b}\right) + cn^d$$

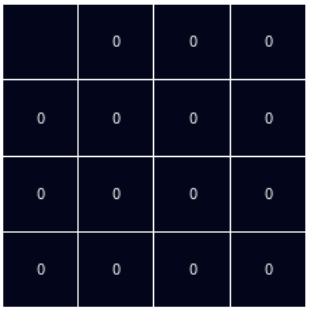
Dalam persamaan tersebut, $n = b^k$, dengan $k = 1, 2, ...,$ dan $a \geq 1,\ b \geq 2,\ c \geq 0,$ serta $d \geq 0$. Dengan demikian, maka kompleksitas waktu $T(n)$ bisa diterjemahkan ke dalam notasi asimptotik melalui persamaan berikut.

$$T(n) = \begin{cases} O(n^d), & a < b^d \\ O(n^d log n), & a = b^d \\ O\left(n^{\log_b a}\right), & a > b^d \end{cases}$$

III. Persoalan Papan Catur Defektif

A. Gambaran Permasalahan

Persoalan Papan Catur Defektif atau *Defective Chessboard Problem* memang tidak terlalu terkenal dibandingkan dengan persoalan catur lainnya, seperti persoalan N-Ratu atau N-*Rook.* Namun, melalui persoalan ini, dapat digambarkan secara intuitif mekanisme kerja yang dimiliki oleh algoritma penyelesaiannya, yaitu Algoritma *Divide and Conquer*. Dimisalkan ada sebuah papan catur, namun dengan ukuran yang dapat digambarkan sebagai papan berukuran $n \times n$ dengan n adalah 2^k, dan $k \in \mathbb{N}$. Namun, dari papan tersebut, salah satu kotaknya secara acak kemudian hilang atau rusak, sehingga total kotak yang ada pada papan tersebut adalah $n^2 - 1$. Berikut merupakan ilustrasi instans persoalan dengan $n = 4$.



Gambar 8 Papan Catur Rusak

Dengan menggunakan kotak-kotak Triomino, dapatkah kita menutupi semua permukaan catur yang masih bagus bisa tertutupi dengan sempurna? Bagaimanakah cara yang harus ditempuh untuk mencapai hal tersebut? Orientasi Triomino yang digunakan adalah bebas dan tidak ada batasan mengenai jumlah Triomino yang digunakan.

B. Analisis Awal Persoalan

Dari domain permasalahan di atas, dapat diketahui bahwa jumlah kotak yang ada di dalam papan tersebut adalah $n^2 - 1$ atau jika dijabarkan lebih lanjut $2^{2k} - 1$. Dapatkah dibuktikan bahwa untuk setiap variasi papan yang ada, maka akan selalu ada susunan Triomino yang bisa menutupi keseluruhan papan?

Hal tersebut bisa dibuktikan melalui induksi matematika sederhana yang telah dipelajari pada Matematika Diskrit. Misalkan bahwa dugaan awal bahwa $2^{2k} - 1$ bisa membagi habis 3 untuk $k = 1$. Maka, jika diperiksa untuk $k = 1$:

$$2^{(2 \cdot 1)} - 1 = 4 - 1 = 3$$

yang akan habis membagi 3. Dengan demikian, maka untuk $k = 1$ hipotesis tersebut adalah benar. Selanjutnya, dengan asumsi bahwa untuk x bagian dari bilangan asli berlaku dugaan $2^{2x} - 1 = 3q$, akan dibuktikan bahwa dugaan tersebut juga benar untuk masukan $x + 1$.

$$2^{2 \cdot (x+1)} - 1 = 2^{2x+2} - 1$$
$$(2^{2x}.4) - 1 = 2^{2x} + 3.2^{2x} - 1$$
$$3.2^{2x} + 3q = 3(2^{2k} + q)$$

Dari hasil di atas, jelas bahwa sesuatu yang dikalikan dengan 3 juga akan habis membagi 3. Dengan demikian, terbukti bahwa setiap varian papan catur yang ada haruslah memiliki paling tidak sebuah solusi, tidak peduli bagaimana susunan Triomino tersebut diletakkan.

Selain itu, karena Triomino memiliki 3 buah kotak, dan pada papan catur tersebut tersisa $2^{2k} - 1$ kotak, maka jumlah Triomino yang digunakan adalah $\frac{2^{2k}-1}{3}$. Sebagai contoh, untuk papan berukuran 4×4 di atas dengan $k = 2$, maka akan digunakan Triomino sebanyak:

$$\frac{2^{2 \cdot 2} - 1}{3} = \frac{2^4 - 1}{3} = \frac{15}{3} = 5$$

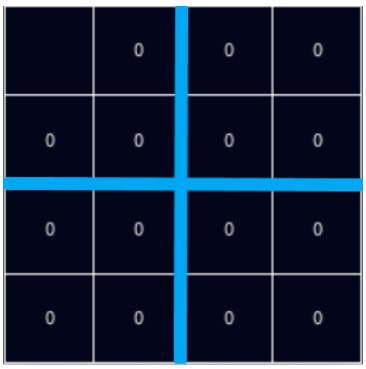
C. Pendekatan Algoritma

Penyelesaian permasalahan ini akan dipecahkan dengan menggunakan algoritma rekursif melalui implementasi *Divide and Conquer*. Dengan demikian, maka hal pertama yang harus dilakukan adalah menentukan basis dari tahapan rekursif yang ada dalam algoritma program.

Basis rekursif yang akan dipakai adalah ketika ukuran dari papan tersebut adalah 2×2. Hal ini dikarenakan, dengan ukuran ini, papan tersebut hanya memerlukan sebuah Triomino untuk diisikan ke dalamnya sehingga semua permukaannya tertutupi. Dengan demikian, tidak perlu adanya rekursi sama sekali dalam kasus tersebut.

Gambar 9. Basis Rekursi Algoritma

Pendekatan rekursif yang akan dilakukan adalah sebagai berikut. Pertama, papan catur akan dibagi-bagi ke dalam empat bagian sama rata. Pembagian ini berkoresponden dengan tahap *Divide* sebagai intrinsik *Divide and Conquer*. Dengan pembagian ini, maka akan dihasilkan empat buah kuadran pada papan catur tersebut.

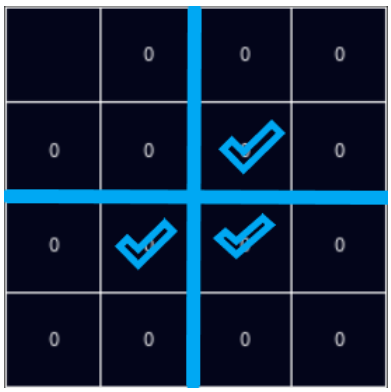


Gambar 10. Pembagian Papan Catur ke dalam 4 Bagian

Sampai tahap ini, dapat diketahui bahwa salah satu dari keempat kuadran merupakan tempat beradanya kotak rusak papan catur awal. Dari ilustrasi di atas, dapat diketahui bahwa kotak rusak terdapat pada kuadran kedua dari papan tersebut. Dalam kasus tersebut, meskipun keempat papan catur hasil pembagian tersebut sudah berukuran 2×2 dan mencapai basis rekursif, algoritma penyelesaian yang dimiliki tidak akan bisa melanjutkan penyelesaiannya.

Algoritma yang sama tidak bisa diterapkan pada semua papan catur tersebut, karena ada sebuah papan yang memiliki kotak rusak, dan ada tiga buah papan tanpa kotak rusak. Algoritma *Divide and Conquer* yang dibangun membutuhkan karakteristik persoalan yang sama dengan persoalan awal, yaitu di sebuah papan catur harus terdapat sebuah kotak yang rusak atau hilang.

Masalah ini bisa diatasi dengan cara menambahkan kotak rusak secara sengaja pada masing-masing papan yang belum memilikinya. Agar pola penambahan kotak ini teratur, maka cara termudah menambahkan kotak tersebut adalah dengan merusak (selalu) tiga kotak yang terdapat pada pusat papan. Bagian yang dimaksud dapat dilihat pada ilustrasi berikut.



Gambar 11. Perusakan Papan secara Artifisial

Dengan perusakan tersebut, algoritma bisa melanjutkan penyelesaiannya dan pemrogram hanya perlu membuat generalisasi algoritma penyelesaiannya. Hal ini akan dilakukan secara rekursif sampai status dari rekursi yang dilakukan telah mencapai basis, yaitu secara lengkap adalah papan catur berukuran 2×2 dengan satu kotak yang rusak.

Ketika papan telah mencapai basis rekursif, maka tahap Solve akan terjadi. Setiap papan akan dipasangkan masing-masing satu Triomino yang orientasinya disesuaikan dengan letak dari kotak rusak pada papan tersebut. Karena adanya generalisasi algoritma, maka tahap penyelesaian setiap sub-papan akan dijalankan secara teratur menurut pola tertentu, misalnya dari kuadran kedua terlebih dahulu, dilanjutkan ke kuadran pertama, ketiga, dan keempat.

Namun, tahap *Solve* sampai di sini masih belum selesai. Setelah setiap kuadran dipasangkan dengan sebuah Triomino, maka kotak tengah yang sebelumnya tadi dirusak secara sengaja juga harus diperbaiki lagi. Setelah itu, pada area ini juga akan dipasangkan sebuah Triomino lagi sesuai dengan orientasinya. Dengan demikian, lengkaplah sudah semua tahap *Solve* yang ada di dalam algoritma ini.

Tahap *Combine* sendiri sebenarnya sudah tercakup ke dalam tahap *Solve*. Secara tidak langsung, dengan menyelesaikan satu sub-papan sedikit demi sedikit, maka nantinya semua permukaan papan akan tertutupi secara sempurna sehingga kombinasi susunan Triomino tersebut sudah dikombinasikan secara implisit.

E. Kompleksitas Waktu

Dalam uraian algoritma di atas, papan catur dibagi ke dalam empat sub-permasalahan yang sama besar. Dalam hal ini, pembagian ini dilakukan terhadap setengah ukuran awal dari papan catur. Selain operasi pembagian sub-persoalan, juga ada operasi-operasi lain seperti operasi penambahan Triomino atau operasi penambahan kotak rusak. Tentunya, operasi-operasi ini jauh memiliki kompleksitas waktu yang jauh lebih kecil dibandingkan operasi pembagian tadi, yaitu dalam waktu yang konstan. Dengan demikian, maka kompleksitas waktu algoritma penyelesaian Papan Catur Defektif, yaitu $T(n)$, dapat dinyatakan dalam persamaan berikut ini.

$$T(n) = 4T(n/2) + c$$

Konstanta c merupakan konstanta operasi-operasi lain yang telah disebutkan sebelumnya. Koefisien 4 dan masukan $n/2$ berasal dari uraian pembagian papan catur dan masukan yang telah dijelaskan sebelumnya. Jika diterjemahkan menurut relasi rekurens yang ada pada Teorema Master, maka akan dihasilkan beberapa konstanta berikut.

$$a = 4$$
$$b = 2$$
$$d = 0$$

Semua konstanta di atas sudah memenuhi aturan penggunaan Teorema Master, yaitu $a \geq 1$, $b \geq 2$, $c \geq 0$, dan $d \geq 0$. Fungsi kompleksitas waktu tersebut juga sudah pasti monoton menaik. Dengan demikian, maka Teorema Master bisa diaplikasikan ke dalam relasi rekurens ini untuk mencari notasi asimptotik dari algoritma penyelesaian Papan Catur Defektif.

Untuk mencapai hal tersebut, pertama harus dibandingkan terlebih dahulu nilai dari a dan $b\hat{}d$, apakah lebih besar, lebih kecil, atau sama. Dengan memasukkan semua konstanta yang dibutuhkan, maka:

$$a \; ? \; b^d$$
$$4 \; ? \; 2^0$$
$$4 \; ? \; 1$$
$$4 > 1$$
$$\rightarrow a > b^d$$

Artinya, relasi rekurens tersebut masuk ke dalam kategori ketiga Teorema Master dengan syarat $a > b^d$. Dengan demikian, maka penerjemahan relasi rekurens tersebut ke dalam notasi asimptotik adalah sebagai berikut.

$$T(n) = O\left(n^{\log_b a}\right)$$
$$T(n) = O\left(n^{\log_2 4}\right)$$
$$T(n) = O(n^2)$$

Artinya, algoritma penyelesaian Papan Catur Defektif memiliki kompleksitas waktu asimptotik $O(n^2)$. Hal ini menandakan kompleksitas waktu untuk algoritma tersebut masih tidak terlalu baik, namun tentunya pendekatan dengan *Divide and Conquer* ini jauh lebih baik daripada mencoba semua kemungkinan yang mungkin dengan pendekatan secara *Brute Force*.

F. Struktur Data Papan Catur

Papan catur dalam persoalan ini memiliki bentuk dan ukuran yang teratur. Oleh karena itu, maka struktur data yang paling sederhana yang bisa digunakan untuk menggambarkan papan catur ini adalah matriks. Representasi papan catur dengan menggunakan matriks merupakan salah satu yang paling umum. Dalam hal ini, matriks ini merupakan matriks persegi yang simetris karena ukuran panjang dan lebar yang sama.

Elemen penyusun matriks ini adalah sebuah *integer*. *Integer* ini akan menjadi representasi status dari setiap kotak yang ada pada papan catur tersebut. Sebagai permulaan, papan catur akan diinisialisasi dengan matriks yang semua elemennya bernilai 0. Untuk membedakan kotak yang rusak dengan kotak yang bagus, maka elemen matriks untuk posisi yang bersesuaian akan diganti dengan elemen lain, misalnya -1 untuk memudahkan pengecekannya.

Dalam bahasa pemrograman Python, matriks bisa digambarkan dengan menggunakan representasi *built-in* struktur data *list*. Di dalam *list* ini akan diisikan *list* lagi secara *nested* atau bersarang. Ukuran dari *nested list* ini akan merepresentasikan jumlah kolom yang terdapat pada matriks tersebut. Adapun jumlah *nested list* ini akan merepresentasikan jumlah baris pada matriks tersebut.

```
[[1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1.]]
```

Gambar 12. Representasi Matriks Identitas pada Python

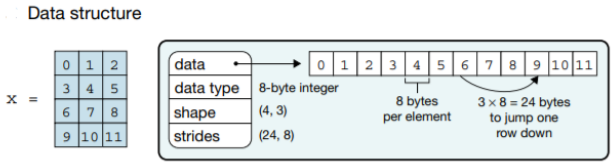
Dalam implementasinya, penulis menggunakan *package Numpy* untuk struktur datanya. Numpy merupakan pustaka utama dalam pemrograman *Array* untuk bahasa pemrograman Python. Di kalangan pengembang Python, Numpy memang sudah sangat terkenal dengan versatilitas yang dimilikinya. Dalam hal ini, struktur data yang paling utama dan sering digunakan adalah *array* dan matriks.

Pemrograman *Array* dengan menggunakan Numpy memberikan berbagai operasi data yang lebih kuat, *compact*, dan dengan sintaks yang lebih ekspresif. Dalam Python biasa, inisialisasi matriks dan larik memiliki sintaks yang tidak *clean* dan bersifat sangat manual.

Dengan menggunakan Numpy, inisialisasi struktur data bisa menjadi lebih mudah, karena inisialisasi matriks bisa dilakukan dengan menggunakan fungsionalitas yang ada pada Numpy, dan pemrogram hanya perlu mendefinisikan jenis data untuk elemen matriks dan ukuran dari matriks tersebut.

Operasi-operasi lain seperti pengaksesan elemen maupun manipulasi data menjadi lebih cepat. Numpy dibangun dengan mengintegrasikan berbagai bahasa pemrograman lain yang lebih cepat dibandingkan Python, seperti C, C++, dan Fortran. Selain itu, Numpy juga melakukan *breakdown* pada kumpulan *task* yang akan dieksekusi terhadap struktur datanya ke dalam fragmen-fragmen kecil. Fragmen ini kemudian diproses secara parelel sehingga kecepatan eksekusinya jauh lebih cepat.

Untuk *array* sendiri (atau matriks yang tersusun dari *array*), elemen larik merupakan elemen dengan tipe data yang homogen. Elemen-elemen ini juga disimpan pada memori yang kontigu. Hal ini berbanding terbalik dengan elemen pada *list* Python yang memungkinkan penyimpanan data bertipe heterogen. Elemen juga disimpan pada memori yang tidak kontigu sehingga iterasi elemen akan menjadi lebih lambat.



Gambar 13. Larik pada Numpy
(*Sumber: https://www.nature.com/articles/s41586-020-2649-2.pdf*)

G. Visualisasi Papan Catur

Berbicara mengenai visualisasi program, banyak sekali cara yang bisa ditempuh untuk melakukannya. Salah satu pendekatan yang mungkin digunakan adalah dengan menggunakan *Graphical User Interface Programming*. Pendekatannya adalah sebagai berikut.

Pendekatan GUI dilakukan dengan menggambarkan papan catur pada layar GUI. Agar setiap langkah penyelesaian bisa tervisualisasikan dengan jelas, setiap *tile* pada papan catur ini bisa dikaitkan dengan elemen dari matriks representasinya. Dengan demikian, maka perubahan yang ada pada setiap *tile* matriks bisa dideteksi dengan mudah dan memberikan visual yang tampak lebih dinamis.

Namun, pada karya tulis ini pemrogram menggunakan bantuan dari pustaka visualisasi *chart* dan *graph* (bukan graf biasa) seperti *Matplotlib* dan *Seaborn*. Pustaka *Seaborn* sendiri dibangun menggunakan *Matplotlib* sebagai basis utamanya. Kedua pustaka saling bekerja sama untuk memberikan visualisasi terbaik untuk sekumpulan datanya.

Karena struktur data yang digunakan pada penyelesaian ini adalah matriks, maka struktur data ini cocok dan kompatibel dengan struktur data yang dibutuhkan oleh *Seaborn* dan *Matplotlib*. Dengan demikian, tidak dibutuhkan lagi pengolahan data apapun sehingga menghemat banyak *resources* dan waktu.

Pustaka Seaborn menyediakan tampilan *interface* dengan abstraksi pengolahan data yang cukup tinggi. Dalam hal ini, tampilan yang dihasilkan merupakan tampilan atraktif dan informatif untuk memberikan kesan visualisasi statistikal yang lebih mendetail dan menarik. Benar, baik Matplotlib maupun Seaborn merupakan pustaka visualisasi yang paling sering digunakan oleh *Data Scientist* yang berurusan dengan Python.

Selain memiliki sintaks yang cukup sederhana dan mudah dimengerti, Seaborn melakukan operasi *plotting* terhadap *dataframe* ataupun larik yang menjadi inputnya. Dalam hal ini, Seaborn akan melakukan beberapa pemetaan semantik dan agregasi statistikal sebagaimana diperlukan dalam memberikan visualisasi tersebut.

Namun, pengolahan ini dilakukan tanpa mengubah larik masukan yang diberikan ke dalamnya sehingga pemrogram tidak perlu khawatir terhadap hal tersebut. Selain itu, penggunaan pustaka Seaborn memberikan kemudahan kepada pemrogram dan lebih berfokus pada apa yang ingin divisualisasikan, bukan bagaimana cara memvisualisasikannya. Dengan demikian, Seaborn dapat dikatakan memiliki API yang bersifat deklaratif dan *data oriented*.

Dari pustaka tersebut, akan digunakan visualisasi berupa *heatmap* yang sering digunakan untuk menggambarkan distribusi data. Dalam hal ini, *heatmap* cocok digunakan karena tidak seperti kebanyakan *chart* pada *Data Science*, *heatmap* memiliki bentuk yang paling mendekati papan catur.

Untuk membuat *heatmap* benar-benar menyerupai papan catur, maka pada *heatmap* dalam pustaka Seaborn ada atribut yang dinamakan *square*. Atribut ini merupakan atribut yang bertipe *Boolean* dan jika ditetapkan menjadi *True*, maka semua *tile* yang ada pada *heatmap* akan memiliki ukuran yang simetris. Dengan demikian, *heatmap* sudah menjadi papan catur dan hanya perlu disesuaikan ukurannya dengan ukuran masukan program.

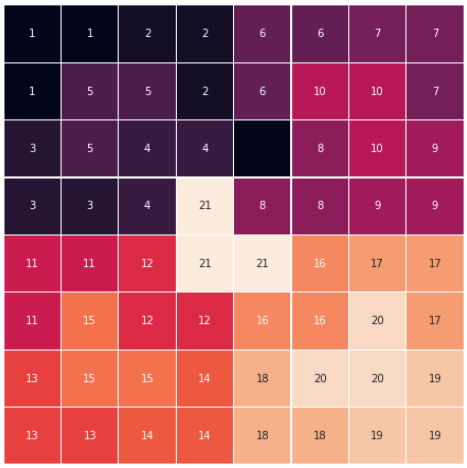
Pendekatan yang dilakukan untuk memvisualisasikan proses *tiling* pada penyelesaian ini adalah sebagai berikut. *Heatmap* pada visualisasi akan dibagi menjadi dua buah *layer* yang saling tumpang-tindih. *Layer* pertama merupakan fondasi utama papan catur. Pada lapisan ini, *heatmap* tidak akan menggambarkan *label* atau anotasi apapun.

Selanjutnya, lapisan kedua merupakan lapisan yang digunakan untuk memperlihatkan perubahan yang terjadi selama proses *tiling* terjadi. Pada lapisan ini, akan ditampilkan anotasi pada setiap *tile* yang ada. Anotasi yang diberikan berupa urutan kotak-kotak tersebut dipasangkan Triomino. Dalam hal ini, semua kotak pada papan tersebut awalnya akan diberikan anotasi 0 karena belum ada proses yang terjadi.

Tidak hanya itu, juga diperlukan sebuah cara bagi penggunanya untuk membedakan kotak yang masih bagus dengan kotak yang sudah rusak. Tujuan utama dari dibuatnya lapisan kedua ini adalah untuk mencapai hal tersebut. Fungsionalitas Seaborn yang digunakan adalah fungsionalitas *masking* terhadap data masukan visualisasi tersebut.

Seperti yang diketahui, pada representasi matriks papan catur tersebut, kotak yang rusak akan diberikan representasi bilangan bulat -1 sebagai elemennya. Oleh karena itu, *masking* terhadap data masukan bisa dilakukan dengan menyediakan kondisi data yang ingin ditutupi. Dalam kasus ini, maka kondisi *masking* untuk kotak yang defektif adalah ketika elemen pada posisi yang bersesuaian bernilai kurang dari 0. Ketika divisualisasikan, maka nantinya kotak yang memenuhi kondisi ini akan menjadi kotak kosong tanpa anotasi apapun.

Visualisasi ini dibuat ke dalam bentuk fungsi, sehingga setiap tahapan rekursi yang dilakukan nantinya akan memanggil fungsi ini untuk melakukan visualisasi status papan catur pada saat itu. Dengan demikian, opsionalitas terakhir yang bisa dilakukan adalah dengan mempercantik visualisasi tersebut, misalnya dengan menggunakan *color pallete* tertentu, atau menghilangkan *color bar*, yaitu *bar* yang menyatakan saturasi warna pada *heatmap*.



Gambar 14. Ilustrasi Visualisasi Papan Catur Rusak yang Telah Dipasangkan 21 Buah Triomino

V. Lain-Lain

Dalam implementasi algoritma tersebut, penulis masih menggunakan paradigma prosedural. Dalam hal ini, paradigma lain yang bisa digunakan tentunya adalah paradigma berorientasi objek.

Dalam pemrograman berorientasi objek, implikasi terhadap program adalah adanya pembagian objek-objek berdasarkan tanggungjawab masing-masing objek tersebut. Dalam persoalan Papan Catur Defektif ini, maka objek-objek yang mungkin muncul adalah *tile* papan catur dan papan catur itu sendiri.

Dalam hal ini, *tile* papan catur akan memberikan layanan berupa pengecekan status internalnya pada saat tertentu. Dengan demikian, maka kotak rusak akan dideteksi dengan mudah melalui pengecekan tersebut. Perubahan-perubahan yang diperlukan terhadap setiap *tile* juga bisa diatur secara mandiri oleh setiap *tile*.

Adapun tentunya papan catur akan menerapkan prinsip komposisi terhadap instansiasi objek dari kelas *tile*. Dalam hal ini, jika *tile* yang dibuat telah mengandung informasi mengenai letak dari setiap *tile*, maka pemrogram tinggal memasukkan kumpulan objek *tile* ke dalam sebuah larik. Nantinya, algoritma *tiling* dan visualisasi penyelesaian persoalan akan ditangani oleh kelas papan catur ini.

Selain itu, papan catur juga bisa melakukan *tracking* jumlah Triomino yang telah digunakan di dalam program. Hal ini bisa dilakukan dengan menjadikan jumlah Triomino ini sebagai sebuah status internal untuk papan catur tersebut. Dengan demikian, jumlah Triomino yang pada paradigma prosedural diimplementasikan dengan variabel global dapat digunakan dengan mudah dalam pemrograman objek ini.

Keuntungan dari penggunaan paradigma berorientasi objek ini adalah dihasilkannya kode program yang lebih rapi dan teratur. Dengan paradigma ini, maka setiap entitas yang ada di dalam persoalan dapat dikelompokkan dan didelegasikan tugasnya dengan jelas dan tepat. Selain itu, dengan tidak adanya penggunaan variabel global, maka program menjadi lebih tahan dari *bug* yang mungkin muncul akibat variabel tersebut. Kelemahan yang ada pada pendekatan ini terletak pada tahap visualisasi nantinya, meskipun hal ini bisa diatasi, lagi-lagi dengan menggunakan matriks sebagai struktur data penyimpanan *tile* papan catur tersebut.

VI. Tautan Relevan

Berikut merupakan beberapa tautan relevan yang bisa dilihat terkait dengan karya tulis ini.

YouTube:
https://www.youtube.com/watch?v=l6CkEuuqXh8&t=9s

Source Code:
https://colab.research.google.com/drive/1wfbQ6V6y0A7FXnY3GsqN3bFXWCva6zEP?usp=sharing

VII. Simpulan

Catur merupakan salah satu permainan papan yang hingga saat ini masih dimainkan di berbagai belahan dunia. Bahkan, dengan bantuan teknologi, catur berkembang pesat dan menjadi salah satu cabang *esport* virtual. Dalam *Computer Science* sendiri, banyak sekali persoalan-persoalan yang berkaitan dengan catur, baik terhadap komponen permainan seperti *Queen* atau *Rook* maupun papan catur itu sendiri. Salah satu persoalan tersebut adalah persoalan Papan Catur Defektif atau *Defective Chessboard Problem* yang meminta pemrogram untuk menutupi permukaan papan rusak ini dengan Triomino.

Algoritma yang bisa digunakan untuk memecahkan permasalahan ini adalah *Divide and Conquer*. Pendekatannya adalah dengan membagi papan catur ke dalam papan catur yang lebih kecil lagi secara rekursif hingga hanya diperlukan satu buah Triomino untuk mentuupinya. Selain memecahkan persoalan ini, visualisasi papan catur selama proses penyelesaian juga menjadi hal yang penting. Adapun karena merupakan algoritma yang cukup kompleks dan memakan *resources* yang cukup banyak, maka diperlukan berbagai optimisasi untuk membuat program menjadi lebih baik.

Optimisasi tersebut dilakukan dengan menggunakan pustaka Numpy sehingga struktur data yang digunakan di dalam program menjadi lebih cepat dan efektif untuk dimanipulasi. Untuk visualisasinya sendiri, digunakan pustaka Seaborn dan Matplotlib dengan penggambaran *heatmap*. Dengan demikian, setiap tahap rekursif yang dilakukan pada proses penyelesaian program bisa dipahami secara lebih intuitif melalui visualisasi algoritma tersebut.

VII. Ucapan Terima Kasih

Penulis mengucapkan puji syukur kepada Tuhan Yang Maha Esa atas segala bantuan lahir batin yang diberikan kepada penulis dalam menyelesaikan penyusunan karya tulis ini serta perkuliahan yang ada. Selain itu, penulis juga berterima kasih kepada keluarga yang telah senantiasa mendukung dan memberikan penulis semangat untuk menjadi pribadi yang lebih baik. Tidak lupa rasa hormat dan terima kasih yang sangat dalam dari penulis untuk semua dosen pengampu mata kuliah Strategi Algoritma IF2211 Semester 2 2020/2021 yang telah mengupayakan pembelajaran dalam masa pandemi ini. Selain itu juga, penulis mengucapkan terima kasih untuk teman-teman yang telah membantu dan menyemangati penulis.

Daftar Pustaka

[1] Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. Nature 585, 357–362 (2020). *Array Programming with Numpy*. Diakses melalui https://www.nature.com/articles/s41586-020-2649-2.pdf pada 1 Mei 2021 pukul 15.31 WIB.

[2] Hedge, Chetana. *Divide and Conquer CSE*. Diakses melalui http://www.chetanahegde.in/wp-content/uploads/2016/02/Divide_Conquer_CSE.pdf pada 1 April 2021 pukul 10.21 WIB.

[3] Karajgi, Aniruddha. *Visualizing the Defective Chessboard Problem*. Diakses melalui https://polaris000.medium.com/visualizing-the-defective-chessboard-problem-aa5fc38b6e5e pada 30 April 2021 pukul 17.21 WIB.

[4] Karajgi, Aniruddha. *Visualizing the Defective Chessboard Problem*. Diakses melalui https://polaris000.github.io/blog/defective_chessboard#the-code pada 30 April 2021 pukul 18.42 WIB.

[5] Munir, Rinaldi. *Divide and Conquer (Bagian 1)*. Diakses melalui https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian1.pdf pada 30 April 2021 pukul 20.21 WIB.

[6] Munir, Rinaldi. *Divide and Conquer (Bagian 2)*. Diakses melalui https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian2.pdf pada 30 April 2021 pukul 21.31WIB.

[7] Munir, Rinaldi. *Divide and Conquer (Bagian 3)*. Diakses melalui https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian3.pdf pada 30 April 2021 pukul 21.38 WIB.

[8] Munir, Rinaldi. *Divide and Conquer (Bagian 4)*. Diakses melalui https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian4.pdf pada 30 April 2021 pukul 22.22 WIB.

[9] Poojary, Manjula. *Note For Design And Analysis Of Algorithm*. Diakses melalui https://lecturenotes.in/notes/27800-note-for-design-and-analysis-of-algorithm-daa-by-manjula-poojary?reading=true&continue=16&materialId=27800-note-for-design-and-analysis-of-algorithm-daa-by-manjula-poojary pada 30 April 2021 pukul 19.02 WIB.

[10] Seaborn. *Seaborn: User Guide and Tutorial*. Diakses melalui https://seaborn.pydata.org/tutorial.html pada 1 Mei 2021 pukul 12.48 WIB.

[11] UF - CISE. *Divide and Conquer Lecture 34 UF - CISE*. Diakses melalui https://www.cise.ufl.edu/~sahni/cop3530/slides/lec346.pdf pada 30 April 2021 pukul 19.01 WIB.

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.

Palembang, 6 Mei 2021

Richard Rivaldo 13519185