# *Optimalisasi Strategi Penyerangan dan Pemilihan Kartu dalam Clash Royale Menggunakan Algoritma Branch and Bound*

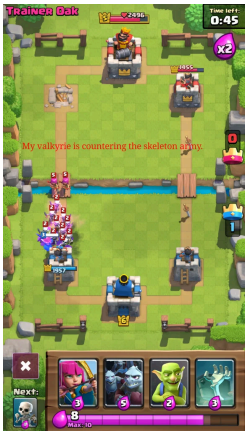
Marzuli Suhada M - 13522070
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13522070@std.stei.itb.ac.id

***Abstrak*—Makalah ini membahas optimalisasi strategi penyerangan dan pemilihan kartu dalam permainan Clash Royale menggunakan algoritma Branch and Bound. Clash Royale, sebuah permainan strategi real-time oleh Supercell, melibatkan pemain bertarung satu sama lain menggunakan berbagai kartu yang mengonsumsi elixir dan memiliki kemampuan yang berbeda-beda. Pemilihan kartu dan strategi yang optimal sangat penting untuk meraih kemenangan. Penelitian ini bertujuan untuk menerapkan algoritma Branch and Bound untuk mengidentifikasi kombinasi kartu optimal yang memaksimalkan kerusakan dalam batas elixir yang ditentukan. Studi ini membandingkan kinerja algoritma Branch and Bound dengan algoritma Greedy, menunjukkan keunggulan Branch and Bound dalam mencapai solusi optimal.**

***Kata Kunci—Clash Royale, Optimalisasi, Algoritma Branch and Bound, Algoritma Greedy***

## I. Pendahuluan

Clash Royale adalah sebuah permainan strategi *real-time* yang dikembangkan oleh Supercell, di mana pemain bertarung satu sama lain menggunakan kartu untuk menyerang menara musuh dan mempertahankan menara mereka sendiri. Setiap kartu memiliki biaya elixir tertentu dan kemampuan yang berbeda, sehingga pemilihan kartu dan strategi penyerangan menjadi sangat krusial untuk meraih kemenangan.




**Gambar 1.** Contoh pertarungan permainan Clash Royale

Source : https://www.instructables.com/Battle-Strategies-Clash-Royale/

Dalam konteks ini, optimalisasi strategi penyerangan dengan mempertimbangkan pemilihan kartu menjadi tantangan yang menarik. Pemain perlu menentukan kombinasi kartu yang tidak hanya efektif dalam menyerang tetapi juga efisien dalam penggunaan elixir. Algoritma Branch and Bound adalah salah satu metode yang dapat digunakan untuk mengatasi masalah ini dengan mencari solusi optimal dalam ruang kemungkinan yang sangat besar. Algoritma ini bekerja dengan membagi ruang solusi menjadi beberapa subruang yang lebih kecil dan mengevaluasi batas atas atau batas bawah dari solusi optimal di setiap subruang tersebut. Jika suatu subruang tidak dapat menghasilkan solusi yang lebih baik daripada solusi yang sudah ditemukan, maka subruang tersebut diabaikan.

Makalah ini bertujuan untuk menjelaskan penerapan algoritma Branch and Bound dalam optimalisasi strategi penyerangan dan pemilihan kartu pada permainan Clash Royale. Peneliti akan menjelaskan bagaimana algoritma ini dapat digunakan untuk memilih kombinasi kartu yang menghasilkan damage maksimal dengan biaya elixir yang terbatas. Peneliti juga akan membandingkan kinerja algoritma Branch and Bound dengan algoritma Greedy untuk menunjukkan keunggulan dalam mencapai solusi optimal.

## II. Landasan Teori

### A. *Algoritma Branch and Bound*

Algoritma Branch and Bound merupakan salah satu algoritma pencarian sistematis yang digunakan untuk menemukan solusi optimal dari berbagai masalah optimisasi, termasuk masalah Knapsack dan masalah TSP (Travelling Salesman Problem). Algoritma ini bekerja dengan membagi ruang solusi menjadi beberapa subruang yang lebih kecil (*branch*) dan mengevaluasi batas atas atau batas bawah dari solusi optimal di setiap subruang tersebut (*bound*). Jika suatu subruang tidak dapat menghasilkan solusi yang lebih baik daripada solusi yang sudah ditemukan, maka sub-ruang tersebut diabaikan. Definisi dari algoritma ini dituliskan dalam formula

$$min_{x \in S} f(x) \text{ atau } max_{x \in S} f(x)$$

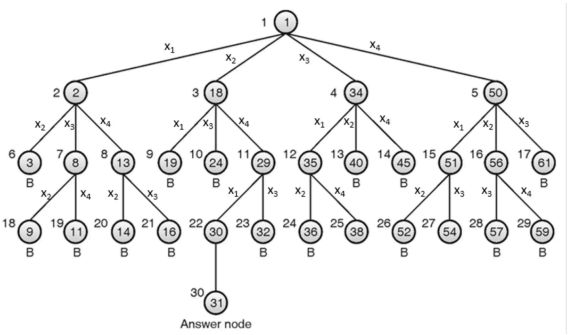
dimana $x$ adalah variabel $(x1, x2, \ldots, xn)$, $f$ adalah fungsi tujuan dan $S$ adalah ruang solusi yang memungkinkan.

Beberapa istilah teknis yang digunakan dalam Algoritma Branch and Bound adalah sebagai berikut

1. Pohon Ruang Status (*State Space Tree*)

   Pohon Ruang Status (*State Space Tree*) dalam algoritma Branch and Bound adalah struktur pohon digunakan untuk mewakili seluruh ruang pencarian dari solusi potensial dalam masalah optimisasi. Pohon ini dimulai dari simpul akar yang mewakili himpunan lengkap solusi yang memungkinkan. Setiap simpul dalam pohon mewakili keadaan atau konfigurasi yang mungkin dari variabel yang terlibat dalam masalah.

   Algoritma Branch and Bound menggunakan pohon ruang status untuk memecah masalah optimisasi menjadi submasalah yang lebih kecil dan mengorganisir pencarian secara hierarkis. Proses pencarian dilakukan dengan menjelajahi pohon ini menggunakan berbagai metode seperti *breadth-first*, *depth-first*, atau *best-first*.




**Gambar 2.** Contoh Pohon Ruang Status Menggunakan Algoritma Branch and Bound

Source :

https://www.instructables.com/Battle-Strategies-Clash-Royale/

2. *Best-Fit Search*

   *Best-Fit Search* merupakan metode yang paling umum digunakan berdasarkan perhitungan *bounding estimation cost*. Dalam metode ini, setiap simpul diberi *cost* sebesar ĉ(i) yang dihitung dengan pendekatan *bounding estimation* tertentu. Kemudian, simpul yang diperluas selanjutnya akan dipilih dari simpul dengan *cost* paling rendah (untuk masalah meminimalkan) atau *cost* paling tinggi (untuk masalah memaksimalkan).

3. *Fungsi Pembatas (Bounding Function)*

   Fungsi pembatas (*bounding functio*n) dalam algoritma Branch and Bound digunakan untuk mengevaluasi simpul-simpul dalam pohon ruang status. Tujuannya adalah untuk memutuskan apakah submasalah yang terkait dengan simpul tersebut perlu diekspansi atau tidak. Fungsi pembatas berperan penting untuk meminimalisir waktu pencarian karena fungsi ini akan memotong cabang-cabang pencarian yang tidak mungkin menghasilkan solusi optimal, sehingga meningkatkan efisiensi algoritma.

   Berikut adalah pseudocode untuk fungsi pembatas dalam algoritma Branch and Bound.

```
function BranchAndBound(problem):
    best_solution_found = infinity
    node_stack = empty_stack()
    root_node =
create_initial_node(problem)
    push(root_node, node_stack)
    while node_stack is not empty:
        current_node = pop(node_stack)
        if not should_prune(current_node,
best_solution_found):
            if is_leaf(current_node):
                current_solution =
evaluate(current_node)
                if current_solution <
best_solution_found:
                    best_solution_found =
current_solution
            else:
                child_nodes =
expand(current_node)
                for each child_node in
child_nodes:
                    push(child_node,
node_stack)

    return best_solution_found

function should_prune(node,
best_solution_found):
    return upper_bound(node) >=
best_solution_found

function is_leaf(node):
    return node is a leaf node

function evaluate(node):
    return evaluate_solution(node)

function expand(node):
    return generate_child_nodes(node)

function upper_bound(node):
    return calculate_upper_bound(node)

function evaluate_solution(node):
    return compute_solution(node)

function create_initial_node(problem):
    return initialize_node(problem)
```

## B. *Permainan Clash Royale*

Clash Royale merupakan permainan strategi di mana pemain menggunakan *deck* kartu yang terdiri dari berbagai jenis pasukan, bangunan, dan spell. Setiap kartu memiliki biaya elixir yang harus dikeluarkan untuk menggunakannya. Strategi dalam permainan ini melibatkan pemilihan kartu yang tepat dan pengaturan waktu penggunaan kartu untuk menyerang musuh atau bertahan dari serangan musuh. Optimalisasi strategi di Clash Royale melibatkan dua aspek utama:

1. Pemilihan Kartu (*Card Selection*)

   Memilih kartu yang akan dimasukkan ke dalam *deck* dengan mempertimbangkan biaya elixir dan potensi *damage*.


**Gambar 3.** Kartu Pertarungan pada Permainan Clash Royale

Source : https://www.ibtimes.co.uk/clash-royale-best-common-rare-epic-cards-mix-deck-1552804

2. Strategi Penyerangan (*Attack Strategy*)

   Mengatur urutan dan waktu penggunaan kartu selama pertandingan untuk memaksimalkan damage ke menara musuh dan meminimalkan kerusakan pada menara sendiri.

## C. *Integer Knapsack Problem*

*Integer Knapsack Problem* adalah salah satu persoalan yang terdapat pada algoritma Branch and Bound. Persoalan ini melibatkan pemilihan item yang memberikan nilai maksimum tanpa melebihi kapasitas tertentu. Dalam konteks Clash Royale, masalah ini dapat diterapkan dalam pemilihan kartu di mana nilai setiap kartu adalah *damage* yang dihasilkan dan kapasitas adalah total biaya elixir yang tersedia. Algoritma Branch and Bound digunakan untuk mencari kombinasi kartu yang menghasilkan *damage* maksimal tanpa melebihi batas elixir yang dimiliki oleh pemain. Formula matematisnya adalah sebagai berikut.

$$\text{Maksimasi } F = \sum_{i=1}^{n} p_i x_i$$

$$\text{dengan kendala (}constraint\text{) } \sum_{i=1}^{n} w_i x_i \leq K$$

$$\text{yang dalam hal ini, } x_i = 0 \text{ atau } 1,\ i = 1, 2, \ldots, n.$$

Dalam masalah maksimasi, nilai atau cost setiap simpul pada pohon ruang status digunakan sebagai *upper bound* dari solusi optimum yang membatasi pencarian agar hanya mengekspansi cabang yang memiliki potensi untuk memberikan solusi yang lebih baik. Untuk memastikan pencarian solusi lebih efisien, objek-objek yang ada pada persoalan diurutkan berdasarkan rasio keuntungan per beratnya yaitu $p_i / w_i$ secara menurun. Hal ini membuat algoritma menjadi lebih cepat dalam menemukan solusi yang optimal dengan mengurangi jumlah cabang yang perlu diekspansi, karena objek dengan rasio keuntungan per berat tertinggi akan diekspansi lebih dahulu.

Pada setiap simpul, terdapat dua kemungkinan cabang yaitu cabang kiri mewakili objek yang dipilih $x_i = 1$ dan cabang kanan mewakili objek yang tidak dipilih $x_i = 0$. Cabang-cabang ini membentuk himpunan bagian (subset) dari objek-objek yang dipilih dari $i$ objek pertama yang sudah diurutkan berdasarkan rasio keuntungan per beratnya.

Rumus untuk menghitung *upper bound* (batas atas) dari simpul pada algoritma Branch and Bound untuk persoalan 1/0 Knapsack adalah sebagai berikut:

$$\hat{c}(i) = F + (K - W)\, p_{i+1}/w_{i+1}$$

di mana $F$ adalah total keuntungan yang sudah dicapai pada simpul saat ini, $K$ adalah kapasitas maksimal knapsack, $W$ adalah total berat yang sudah digunakan pada simpul saat ini, $p_{i+1}$ adalah keuntungan dari objek berikutnya dalam urutan yang sudah diurutkan, dan $w_{i+1}$ adalah berat dari objek berikutnya dalam urutan yang sudah diurutkan.

# III. Implementasi

Implementasi algoritma Branch and Bound untuk menyelesaikan masalah strategi penyerangan dan pemilihan kartu dalam permainan *Clash Royale* dilakukan dalam bahasa pemrograman python dengan menambahkan GUI menggunakan tkinter untuk memperindah visualisasi. Berikut adalah implementasi kode yang dilakukan.

## A. *Fungsi Bound*

Fungsi *bound* digunakan untuk menghitung batas atas dari *damage* yang bisa dicapai dengan menggunakan sisa kartu yang belum dipilih, mulai dari indeks *i*. Fungsi ini digunakan untuk melakukan *pruning* dalam algoritma Branch and Bound, dengan cara mengevaluasi potensi *maximum damage* yang bisa dicapai pada *branch* tertentu.

```
def bound(W, F, i):
    cost = F
    weight = W
    while i < n and weight +
cards[i].elixir_cost <= max_elixir:
        weight += cards[i].elixir_cost
        cost += cards[i].damage
        i += 1
    if i < n:
        cost += (max_elixir - weight) *
(cards[i].damage / cards[i].elixir_cost)
    return cost
```

*B. Fungsi Explore*

Fungsi explore adalah fungsi rekursif yang mengeksplorasi setiap kemungkinan kombinasi kartu, sambil menggunakan fungsi bound untuk melakukan pruning pada *branch* yang tidak mengarahkan ke solusi optimal.

```
def explore(current_strategy, current_damage,
current_elixir, index):
    nonlocal best_strategy, best_damage

    if current_elixir > max_elixir:
        return

    if index == n:
        if current_damage > best_damage:
            best_strategy =
list(current_strategy)
            best_damage = current_damage
        return

    # Check if including current card is
feasible
    if current_elixir + cards[index].elixir_cost
<= max_elixir:
        current_strategy.append(cards[index])
        explore(current_strategy, current_damage
+ cards[index].damage, current_elixir +
cards[index].elixir_cost, index + 1)
        current_strategy.pop()

    # Calculate upper bound (cost)
    cost_with_current = bound(current_elixir,
current_damage, index + 1)
    if cost_with_current > best_damage:
        explore(current_strategy,
current_damage, current_elixir, index + 1)
```

*C. Fungsi Branch and Bound Knapsack*

Fungsi ini adalah fungsi utama yang menginisialisasi algoritma Branch and Bound dan memulai proses eksplorasi.

```
def branch_and_bound_knapsack(self, cards,
max_elixir):
    cards.sort(key=lambda x: x.damage /
x.elixir_cost, reverse=True)
    n = len(cards)
    best_strategy = []
    best_damage = 0

    explore([], 0, 0, 0)
    return best_strategy, best_damage
```

*D. Fungsi Optimize Strategi*

Fungsi *branch_and_bound_knapsack* dipanggil dalam metode *optimize_strategy* saat tombol "Optimize Strategy" ditekan. Hasil optimasi ditampilkan dalam result_text.

```
def optimize_strategy(self):
        try:
            max_elixir =
int(self.elixir_entry.get())
            if max_elixir <= 0:
                raise ValueError
        except ValueError:
            messagebox.showerror("Error", "Max
Elixir must be a positive integer.")
            return

        # Sort cards based on damage per elixir
cost ratio (Greedy strategy)
        self.cards.sort(key=lambda x: x.damage /
x.elixir_cost, reverse=True)

        current_elixir = 0
        best_strategy_greedy = []
        best_damage_greedy = 0

        for card in self.cards:
            if current_elixir + card.elixir_cost
<= max_elixir:

best_strategy_greedy.append(card)
                best_damage_greedy +=
card.damage
                current_elixir +=
card.elixir_cost

        result_text_greedy = f"Greedy Strategy
with max {max_elixir} elixir:\n"
        for card in best_strategy_greedy:
            result_text_greedy += f"{card.name}
(Elixir: {card.elixir_cost}, Damage:
{card.damage})\n"
        result_text_greedy += f"Total Damage:
{best_damage_greedy}\n"

        # Branch and Bound
        best_strategy_bb, best_damage_bb =
self.branch_and_bound_knapsack(self.cards,
max_elixir)

        result_text_bb = f"Branch and Bound
Strategy with max {max_elixir} elixir:\n"
        for card in best_strategy_bb:
```

```
            result_text_bb += f"{card.name}
(Elixir: {card.elixir_cost}, Damage:
{card.damage})\n"
        result_text_bb += f"Total Damage:
{best_damage_bb}\n"

        self.result_text.delete(1.0, tk.END)
        self.result_text.insert(tk.END,
result_text_greedy)
        self.result_text.insert(tk.END, "\n")
        self.result_text.insert(tk.END,
result_text_bb)
```

Algoritma ini akan secara iteratif mengekspansi simpul dalam *priority queue* sampai menemukan solusi optimal. Pada setiap iterasi, algoritma akan mengambil *node* dengan *cost* maksimum dari *priority queue* . Jika *node* terakhir yang diambil menunjukkan bahwa semua item telah dipertimbangkan, solusi yang ditemukan akan diberikan. Namun, jika tidak, algoritma akan menghasilkan *child node* untuk setiap kemungkinan pemilihan item pada level saat ini.

*Child node* dibuat dengan menggandakan array yang sudah dipertimbangkan dan menetapkan status pemilihan untuk item saat ini. *Cost* jalur *child node* diperbarui dengan menambahkan nilai item yang dipilih. Total biaya *child node* dihitung sebagai penjumlahan total keuntungan yang sudah dicapai ditambah dengan perkalian sisa kapasitas knapsack dengan rasio *damage* per *elixir cost* dari objek yang tersisa. *Child node* kemudian ditambahkan ke *priority queue* untuk eksplorasi lebih lanjut. Algoritma melanjutkan proses ini sampai *priority queue* kosong, yang menunjukkan bahwa semua kemungkinan pemilihan item telah dieksplorasi tanpa menemukan solusi yang lebih baik.

Menggunakan test case di bawah ini, konstruksi pohon ruang status serta perhitungan biaya *node* dan kondisi *priority queue* pada setiap iterasi ditunjukkan di bawah ini.

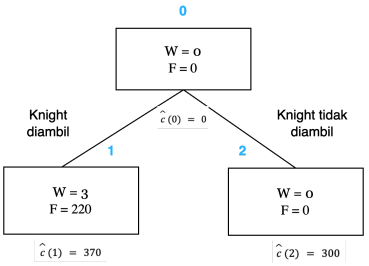
```
Test Case:
    1. Knight: Elixir 3, Damage 220
    2. Archers: Elixir 3, Damage 150
    3. Goblins: Elixir 2, Damage 50
    4. Minions: Elixir 3, Damage 90
Batas elixir maksimum: 6
```

Pencarian solusi dimulai dengan mengurutkan kartu berdasarkan rasio *damage* per *elixir cost* secara *descending*. Urutan yang diperoleh adalah:

1. Knight (220/3 = 73.33)
2. Archers (150/3 = 50)
3. Minions (90/3 = 30)
4. Goblins (50/2 = 25)

Dengan batas elixir maksimum 6, algoritma akan mulai dengan simpul *root* yang tidak memilih kartu apapun, memiliki total elixir 0, dan total damage 0. *Upper bound* pada simpul *root* adalah $(6 - 0) * 73.33 = 440$.

1. Iterasi 1




**Gambar 4.** Pohon Ruang Status untuk iterasi pertama dengan perhitungan *cost*

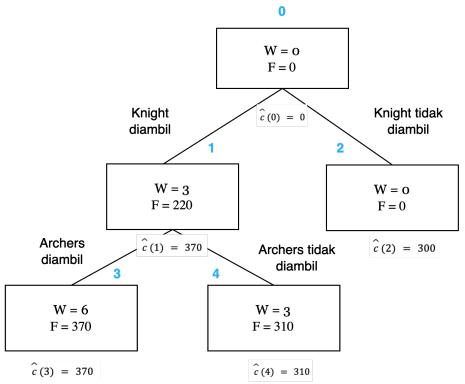
Pada simpul *root*, algoritma mengekspan dua simpul yaitu simpul 1 yang memilih kartu Knight, dan simpul 2 yang tidak memilih kartu Knight.

```
Simpul Kiri (1) :
ĉ(1) = 220 + (6 − 3) * 50 = 370

Simpul Kanan (2) :
ĉ(2) = 0 + (6 − 0) * 50 = 300
```

Berdasarkan perhitungan *cost* tersebut, simpul kiri memiliki *cost* yang lebih besar sehingga simpul kiri yang selanjutnya akan diekspansi.

2. Iterasi 2




**Gambar 5.** Pohon Ruang Status untuk iterasi kedua dengan perhitungan *cost*

Simpul 1 kemudian membangkitkan anak-anaknya yaitu simpul 3 dan simpul 4. Simpul 3 memilih Archers dan simpul 4 tidak memilih Archers.

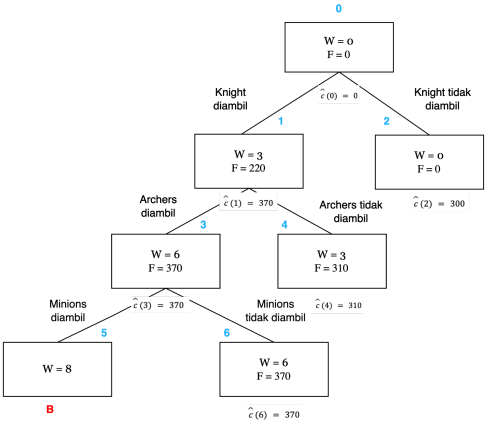
```
Simpul Kiri (3) :
ĉ(3) = 370 + (6 − 6) * 30 = 370
```

Simpul Kanan (4) :

$\hat{c}(4) = 220 + (6 - 3) * 30 = 310$

Berdasarkan perhitungan *cost* tersebut, simpul kiri memiliki *cost* yang lebih besar sehingga simpul kiri yang selanjutnya akan diekspansi.

3. Iterasi 3




**Gambar 6.** Pohon Ruang Status untuk iterasi ketiga dengan perhitungan *cost*

Simpul 3 kemudian membangkitkan anak-anaknya yaitu simpul 5 dan simpul 6. Simpul 5 memilih Minions dan simpul 6 tidak memilih Minions.

Simpul Kiri (5) :

$W = 6 + 3 = 9 > 6$

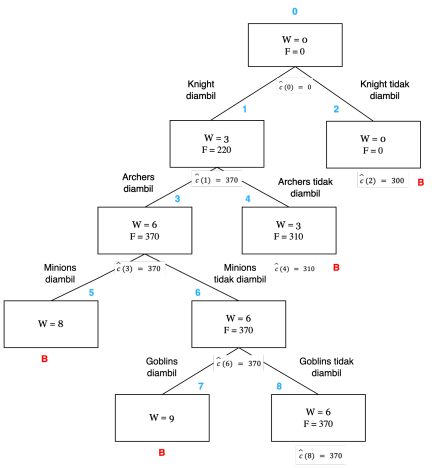
(Langsung dimatikan)

Simpul Kanan (6) :

$\hat{c}(6) = 370 + (6 - 6) * 25 = 370$

Berdasarkan perhitungan *cost* tersebut, karena simpul kiri dimatikan maka simpul kanan yang selanjutnya akan diekspansi.

4. Iterasi 4




**Gambar 7.** Pohon Ruang Status untuk iterasi keempat dengan perhitungan *cost*

Simpul 6 kemudian membangkitkan anak-anaknya yaitu simpul 7 dan simpul 8. Simpul 7 memilih Goblins dan simpul 8 tidak memilih Goblins.

Simpul Kiri (7) :

$W = 6 + 2 = 8 > 6$

(Langsung dimatikan)

Simpul Kanan (8) :

$\hat{c}(8) = 370 + (6 - 6) * 0 = 370$

Berdasarkan perhitungan *cost* tersebut, simpul 8 adalah simpul solusi (*goal node*) dan merupakan solusi optimal.

Setelah mengekspan seluruh simpul dan membandingkan bound dari setiap cabang, solusi optimal yang dihasilkan oleh Branch and Bound adalah memilih Knight dan Archers dengan total elixir 6 dan total damage 370 dan semua simpul hidup yang *cost*-nya lebih kecil dari 370 dibunuh (simpul 2 dan simpul 4 dibunuh).

## IV. Pengujian dan Analisis

Untuk mengevaluasi keoptimalan algoritma *branch and bound* yang diterapkan pada permasalahan pemilihan kartu dalam permainan *Clash Royale* ini, program diuji pada berbagai kasus dengan pembandingnya adalah algoritma *greedy*. Berikut hasil pengujiannya.

### *A. Analisis Kompleksitas Waktu*

Kompleksitas waktu dari algoritma *Branch and Bound* yang digunakan memiliki dua perhitungan.

1. Proses pengurutan kartu berdasarkan rasio *damage* terhadap *cost elixir*, yang memerlukan waktu $O(n \log n)$, di mana $n$ adalah jumlah kartu.

2. Eksplorasi pohon pencarian yang dapat memiliki hingga $2^n$ simpul dalam skenario terburuknya, karena setiap kartu bisa dipilih atau tidak dipilih, sehingga kompleksitas waktunya sebesar $O(n \cdot 2^n)$.

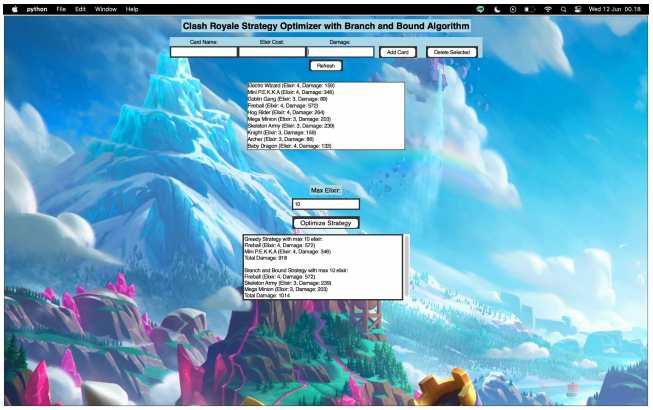
Namun, dalam realisasinya, banyak *branch* yang dipangkas lebih awal, yang dapat mengurangi waktu eksekusi sebenarnya. Meskipun algoritma Branch and Bound ini terkadang lebih lambat daripada algoritma Greedy, yang memiliki kompleksitas waktu $O(n \log n)$, algoritma Branch and Bound masih unggul dalam segi optimalitas. Branch and Bound akan selalu mendapatkan solusi yang optimal sedangkan algoritma greedy tidak selalu optimal.

## *B. Hasil Uji*

TABEL 1. HASIL UJI KE-1

```
TEST CASE :
CARDS = [
CARD("GIANT", 5, 126),
CARD("ELECTRO WIZARD", 4, 159),
CARD("MINI P.E.K.K.A", 4, 346),
CARD("GOBLIN GANG", 3, 89),
CARD("FIREBALL", 4, 572),
CARD("HOG RIDER", 4, 264),
CARD("MEGA MINION", 3, 203),
CARD("SKELETON ARMY", 3, 239),
CARD("KNIGHT", 3, 159),
CARD("ARCHER", 3, 86),
CARD("BABY DRAGON", 4, 133) ]

MAX_ELIXIR = 10
```




ALGORITMA BRANCH AND BOUND

```
BRANCH AND BOUND STRATEGY WITH MAX 10 ELIXIR:
FIREBALL (ELIXIR: 4, DAMAGE: 572)
SKELETON ARMY (ELIXIR: 3, DAMAGE: 239)
MEGA MINION (ELIXIR: 3, DAMAGE: 203)
TOTAL DAMAGE: 1014
```

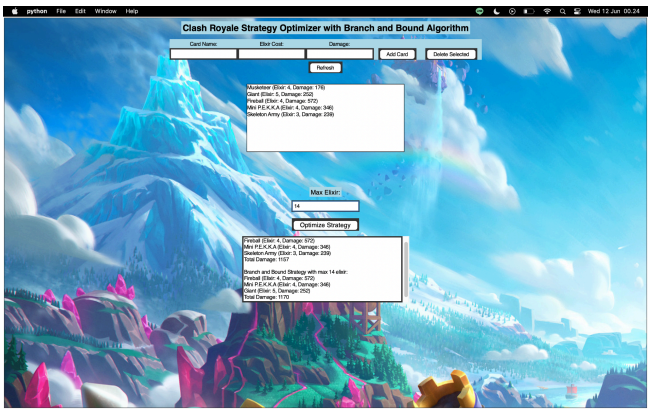
ALGORITMA GREEDY

```
GREEDY STRATEGY WITH MAX 10 ELIXIR:
FIREBALL (ELIXIR: 4, DAMAGE: 572)
MINI P.E.K.K.A (ELIXIR: 4, DAMAGE: 346)
TOTAL DAMAGE: 918
```

TABEL 2. HASIL UJI KE-2

```
TEST CASE :
CARDS = [
CARD("MUSKETEER", 4, 176),
CARD("GIANT", 5, 252),
CARD("FIREBALL", 4, 572),
CARD("MINI P.E.K.K.A", 4, 346),
CARD("SKELETON ARMY", 3, 239) ]

MAX_ELIXIR = 14
```




ALGORITMA BRANCH AND BOUND

```
BRANCH AND BOUND STRATEGY WITH MAX 14 ELIXIR:
FIREBALL (ELIXIR: 4, DAMAGE: 572)
MINI P.E.K.K.A (ELIXIR: 4, DAMAGE: 346)
GIANT (ELIXIR: 5, DAMAGE: 252)
TOTAL DAMAGE: 1170
```

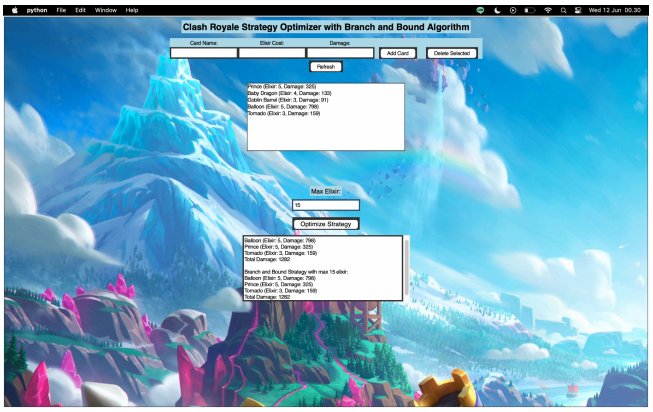
ALGORITMA GREEDY

```
GREEDY STRATEGY WITH MAX 14 ELIXIR:
FIREBALL (ELIXIR: 4, DAMAGE: 572)
MINI P.E.K.K.A (ELIXIR: 4, DAMAGE: 346)
SKELETON ARMY (ELIXIR: 3, DAMAGE: 239)
TOTAL DAMAGE: 1157
```

TABEL 3. HASIL UJI KE-3

```
TEST CASE :
CARDS = [
CARD("PRINCE", 5, 325),
CARD("BABY DRAGON", 4, 133),
CARD("GOBLIN BARREL", 3, 91),
CARD("BALLOON", 5, 798),
CARD("TORNADO", 3, 159) ]
```

```
MAX_ELIXIR = 15
```




```
ALGORITMA BRANCH AND BOUND

BRANCH AND BOUND STRATEGY WITH MAX 15 ELIXIR:
BALLOON (ELIXIR: 5, DAMAGE: 798)
PRINCE (ELIXIR: 5, DAMAGE: 325)
TORNADO (ELIXIR: 3, DAMAGE: 159)
TOTAL DAMAGE: 1282

ALGORITMA GREEDY

GREEDY STRATEGY WITH MAX 15 ELIXIR:
BALLOON (ELIXIR: 5, DAMAGE: 798)
PRINCE (ELIXIR: 5, DAMAGE: 325)
TORNADO (ELIXIR: 3, DAMAGE: 159)
TOTAL DAMAGE: 1282
```

Dari hasil test case yang telah diuji, dapat disimpulkan bahwa algoritma Branch and Bound dan Greedy menghasilkan solusi yang berbeda dalam beberapa kasus. Pada Test Case 1 dan 2, algoritma Greedy menghasilkan solusi yang berbeda dari solusi optimal yang dihasilkan oleh Branch and Bound. Sedangkan pada Test Case 3 kedua algoritma menghasilkan solusi yang sama, karena urutan kartu optimal yang dipilih oleh algoritma Greedy juga optimal secara keseluruhan.

Algoritma Branch and Bound selalu menghasilkan solusi yang optimal, yaitu solusi dengan total damage tertinggi yang mungkin dengan kapasitas elixir tertentu. Algoritma ini melakukan pencarian lebih dalam dengan mempertimbangkan semua kombinasi yang memungkinkan dari kartu-kartu yang tersedia. Di sisi lain, algoritma Greedy memilih solusi berdasarkan keputusan lokal terbaik pada setiap langkahnya, yaitu memilih kartu dengan rasio *damage* per *elixir cost* tertinggi. Karena algoritma Greedy tidak mempertimbangkan konsekuensi jangka panjang dari setiap pilihannya, solusi yang dihasilkan tidak selalu optimal.

Perbedaan dalam solusi antara algoritma Greedy dan Branch and Bound juga bergantung pada kombinasi kartu yang digunakan dan rasio *damage per elixir cost* dari setiap kartu. Dalam kasus di mana kartu-kartu memiliki rasio yang bervariasi atau elixir maksimum yang ketat, solusi dari kedua algoritma dapat berbeda secara signifikan. Branch and Bound cocok digunakan ketika solusi optimal diperlukan dan ada cukup waktu untuk menjalankan algoritma secara lengkap. Di sisi lain, Greedy dapat digunakan jika solusi yang cukup baik sudah cukup, atau jika waktu komputasi terbatas.

## V. KESIMPULAN

Dari hasil implementasi dan analisis, disimpulkan bahwa algoritma Branch and Bound adalah algoritma yang tepat untuk menyelesaikan permasalahan strategi penyerangan dan pemilihan kartu dalam permainan *Clash Royale*. Algoritma ini selalu berhasil menghasilkan solusi yang optimal yang secara umum lebih baik dari algoritma Greedy. Namun, kekurangannya hanya waktu eksekusinya yang lebih lama. Selain itu, terdapat juga batasan lain seperti kompleksitas ruang yang perlu dianalisis lebih lanjut. Faktor-faktor lainnya dalam permainan Clash Royale seperti strategi lawan, waktu pertarungan, dan lain sebagainya juga perlu dipertimbangkan dalam algoritma yang sesungguhnya.

## PRANALA VIDEO YOUTUBE

Demo penjelasan terkait makalah ini dapat dilihat pada video youtube berikut:

https://youtu.be/0xQoBM2pdZ8

## PRANALA GITHUB

*Source code* yang diimplementasikan dalam makalah ini dapat dilihat pada repositori GitHub berikut:

https://github.com/zultopia/13522070-Makalah-Stima.git

## UCAPAN TERIMA KASIH

Penulis mengucapkan rasa syukur kepada Allah SWT atas berkah dan rahmat-Nya, yang mengizinkan penulis untuk menyelesaikan makalah ini dalam jangka waktu yang ditentukan. Penulis ingin berterima kasih kepada Ibu Dr. Nur Ulfa Maulidevi, S.T, M.Sc., sebagai dosen mata kuliah IF2211 Strategi Algoritma pada semester kedua tahun akademik 2023/2024, Kelas 02, atas bimbingan dan pengajarannya sepanjang semester ini. Penulis juga ingin mengucapkan terima kasih kepada orang tua atas dukungan yang berkelanjutan selama perjalanan pendidikan, dan terakhir kepada semua teman yang telah memberikan banyak dukungan penyelesaian makalah ini.

## REFERENSI

[1] Clausen, Jens. 1999. Branch and Bound Algorithms - Principles and Examples. Diakses pada 11 Juni 2024 dari https://janders.eecg.toronto.edu/1387/readings/b_and_b.pdf

[2] Munir, R. 2021. Algoritma Branch and Bound (Bagian 1). Diakses pada 10 Juni 2024 dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Branch-and-Bound-2021-Bagian1.pdf

[3] Munir, R. 2021. Algoritma Branch and Bound (Bagian 2). Diakses pada 10 Juni 2024 dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Branchand-Bound-2021-Bagian2.pdf

[4] Munir, R. 2021. Algoritma Branch and Bound (Bagian 3). Diakses pada 10 Juni 2024 dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Branchand-Bound-2021-Bagian3.pdf

[5] Munir, R. 2021. Algoritma Branch and Bound (Bagian 4). Diakses pada 10 Juni 2024 dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Branchand-Bound-2022-Bagian4.pdf

[6] Zang, L and Luo, W. 2022. A User-based Collaborative Filtering System for Deck Recommendation in Game Clash Royale. Diakses pada 11 Juni 2024 dari https://ieeexplore.ieee.org/abstract/document/9730614/figures#figures

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

Marzuli Suhada M 13522070