Penerapan Algoritma Knapsack untuk Maksimasi Statistik Melalui "Build Item" di League of Legends

Darrel Adinarya Sunanda - 13523061[1,2]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13523061@std.stei.itb.ac.id, [2]darrelyanuar@gmail.com

Abstrak*—League of Legends (LoL) adalah permainan video multipemain yang kompleks di mana strategi pemilihan item secara signifikan memengaruhi hasil pertandingan. Pemain dihadapkan pada tantangan untuk mengalokasikan sumber daya terbatas berupa emas (gold) untuk membeli item yang dapat memaksimalkan statistik karakter mereka. Masalah optimasi ini memiliki kemiripan fundamental dengan masalah Knapsack klasik dalam ilmu komputer. Makalah ini membahas perancangan dan implementasi sebuah sistem yang menerapkan pendekatan algoritma Knapsack, khususnya varian Unbounded Conditional Knapsack, untuk menemukan kombinasi item yang memberikan nilai statistik tertinggi berdasarkan anggaran emas yang ditentukan. Proses implementasi meliputi beberapa tahapan, mulai dari pengumpulan data item mentah dalam format Lua, konversi ke JSON, pra-pemrosesan dan pembersihan data menggunakan skrip Python untuk menghasilkan dataset yang siap pakai dalam format CSV, hingga penerapan algoritma Program Dinamis untuk menyelesaikan masalah tersebut. Solusi ini juga berhasil menangani berbagai batasan unik dalam permainan seperti item legendaris dan pasif unik. Hasil dari program ini adalah rekomendasi build item yang optimal secara matematis untuk satu statistik tertentu, yang menunjukkan bagaimana pendekatan algoritmik dapat digunakan untuk memecahkan masalah optimasi dalam konteks permainan video modern, sekaligus menyoroti perbedaan antara optimalitas teoretis dan kepraktisan strategis.

Kata kunci*— Program Dinamis, Masalah Knapsack, Optimisasi, League of Legends, Strategi Algoritma, Build Item.

I. PENDAHULUAN

League of Legends (LoL) adalah salah satu permainan *Multiplayer Online Battle Arena* (MOBA) paling populer di dunia. Salah satu aspek strategis yang paling fundamental dalam permainan ini adalah pengembangan karakter, yang salah satunya dilakukan melalui pembelian item. Setiap item memiliki harga (biaya dalam *gold*) dan memberikan bonus statistik tertentu, seperti *Attack Damage* (AD), *Ability Power* (AP), atau *Health* (HP). Pemain mendapatkan *gold* secara bertahap selama permainan dan harus membuat keputusan strategis tentang item apa yang akan dibeli untuk memaksimalkan efektivitas karakter mereka.

Gambar I.1 *Item yang tersedia di League of Legends*
Sumber: [1]

Proses pemilihan item ini pada dasarnya adalah sebuah masalah optimasi. Dengan sumber daya (*gold*) yang terbatas, pemain harus memilih sekumpulan item dari ratusan pilihan yang tersedia untuk mencapai bonus statistik setinggi mungkin sesuai kebutuhannya. Masalah ini secara konseptual identik dengan "Masalah Knapsack" (*Knapsack Problem*), sebuah persoalan optimasi kombinatorial terkenal di mana seseorang harus memilih item dengan nilai dan berat tertentu untuk dimasukkan ke dalam ransel dengan kapasitas terbatas, dengan tujuan memaksimalkan total nilai.

Dalam konteks ini, permasalahan optimasi dalam LoL dapat dimodelkan ke “Masalah Knapsack” dengan kapasitas knapsack sebagai total *gold* yang dimiliki pemain, berat barang sebagai harga setiap item, dan nilai barang sebagai bonus statistik yang diberikan oleh setiap item. Dengan demikian, memecahkan masalah knapsack menjadi solusi sekaligus untuk optimisasi *build* dalam *League of Legends*.

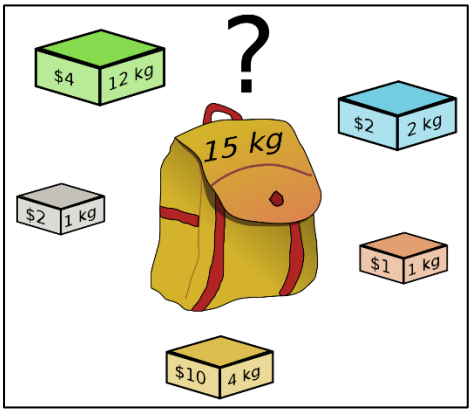
Makalah ini akan menjelaskan proses pengembangan sebuah program yang mampu menyelesaikan masalah ini menggunakan data item LoL yang sebenarnya dengan tujuan memaksimalkan total nilai, sembari mematuhi berbagai aturan dan batasan unik dalam permainan.

II. Dasar Teori

A. Masalah Knapsack (Knapsack Problem)

Masalah Knapsack adalah salah satu masalah optimasi paling fundamental dalam ilmu komputer. Terdapat beberapa varian utama:

1. 0/1 *Knapsack Problem*
   Varian paling umum. Terdapat N item, masing-masing dengan berat w_i dan nilai v_i. Tujuannya adalah memilih *subset* item untuk dimasukkan ke dalam ransel dengan kapasitas W sehingga total nilai $\sum v_i$ maksimal, tanpa melebihi total berat $\sum w_i \leq W$. Setiap item hanya bisa dipilih satu kali (diambil atau tidak).
2. *Unbounded Knapsack Problem* (UKP)
   Mirip dengan 0/1 Knapsack tetapi setiap item tersedia dalam jumlah tak terbatas. Pemain bisa mengambil item yang sama berkali-kali selama total berat tidak melebihi kapasitas.
3. *Conditional Knapsack Problem*
   Varian ini menambahkan batasan atau kondisi pada pemilihan item. Misalnya, "jika item A dipilih, maka item B tidak boleh dipilih".



Gambar II.1 *Ilustrasi permasalahan knapsack*
Sumber: [3]

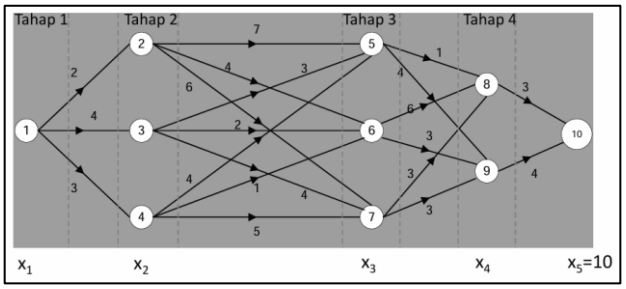
B. Program Dinamis (Dynamic Programming)

Program Dinamis atau *Dynamic Programming* (DP) adalah sebuah metode pemecahan masalah yang menguraikan solusi menjadi sekumpulan tahapan (*stage*). Dengan cara ini, solusi persoalan dapat dipandang sebagai serangkaian keputusan yang saling berkaitan.

DP pada umumnya digunakan untuk menyelesaikan persoalan-persoalan optimasi, baik maksimasi maupun minimisasi. Berbeda dengan algoritma *Greedy* yang hanya menghasilkan satu rangkaian keputusan, pada program dinamis, lebih dari satu rangkaian keputusan akan dipertimbangkan.

Langkah-langkah pengembangan algoritma program dinamis secara umum adalah sebagai berikut:

1. Karakterisasi struktur solusi optimal.
2. Definisikan nilai solusi optimal secara rekursif.
3. Hitung nilai solusi optimal dengan pendekatan maju atau mundur.
4. Konstruksi solusi optimal.



Gambar II.2 *Struktur tahapan dalam program dinamis*
Sumber: [3]

C. Sistem Item dalam League of Legends

Sistem item di LoL memiliki beberapa aturan yang membuat masalah optimasinya menjadi unik:

- Statistik Item
  Setiap item memberikan bonus pada satu atau lebih statistik seperti *Attack Damage* (AD), *Ability Power* (AP), *Armor*, *Magic Resist* (MR), dan lain-lain.
- Biaya Item
  Setiap item memiliki biaya pembelian dalam satuan *gold*.
- Batasan *Slot*
  Pemain hanya memiliki 6 *slot* untuk membawa item di dalam inventorinya.
- Batasan Kategori
  Terdapat batasan untuk item pada kategori tertentu. Pemain hanya bisa memiliki satu item dari kategori *'Boots'* dan tidak bisa memiliki duplikat dari item *'Legendary'*.
- Batasan Kelompok
  Beberapa item memberi kemampuan yang unik. Apabila kemampuan tersebut digandakan, pemain dapat menjadi terlalu kuat. Oleh karena itu, terdapat batasan untuk beberapa item yang diberikan kelompok yang sama. Pemain tidak bisa memiliki lebih dari satu item pada suatu kelompok yang sama.

Aturan-aturan ini harus dimodelkan sebagai batasan tambahan dalam algoritma optimasi.

III. Analisis

Untuk melakukan optimisasi *build* item dalam LoL, kita akan membutuhkan nilai-nilai dari setiap item yang tersedia. Data terpusat mengenai dokumentasi setiap item dalam permainan tidak tersedia secara langsung pada game-nya. Meskipun demikian, kita dapat menggunakan situs Wiki yang menyediakan dokumentasi lengkap yang dibuat oleh komunitas. Dokumen ini dapat ditemukan di https://wiki.leagueoflegends.com/en-us/Module:ItemData/data (24 Juni 2024).

```
...,
["Aether Wisp"] = {
    ["id"]                  = 3113,
    ["nickname"]            = {"spooky ghost"},
    ["tier"]                = 2,
    ["type"]                = {"Epic"},
    ["modes"] = {
        ["classic sr 5v5"]  = true,
        ["aram"]            = true,
        ["nb"]              = true,
        ["arena"]           = false,
    },
    ["menu"] = {
        ["mage"]            = true,
        ["movement"]        = true,
    },
    ["stats"] = {
        ["ap"]              = 30,
        ["ms"]              = 4,
    },
    ["recipe"]              = {"Amplifying Tome"},
    ["buy"]                 = 900,
},
...
```

Tabel III.1 *Cuplikan itemData/data*
Sumber: [2]

Data yang didapatkan sudah mencakup semua item yang ada. Namun, data disediakan dalam bentuk modul tabel untuk bahasa pemrograman Lua yang tidak mudah untuk diproses. Oleh karena itu, kita akan melakukan konversi format data menjadi bentuk JSON yang lebih umum terlebih dahulu.

```
...,
"Aether Wisp": {
    "id": 3113,
    "nickname": [
        "spooky ghost"
    ],
    "tier": 2,
    "type": [
        "Epic"
    ],
    "modes": {
        "classic sr 5v5": true,
        "aram": true,
        "nb": true,
        "arena": false
    },
    "menu": {
        "mage": true,
        "movement": true
    },
    "stats": {
        "ap": 30,
        "ms": 4
    },
    "recipe": [
        "Amplifying Tome"
    ],
    "buy": 900
},
...
```

Tabel III.2 *Cuplikan itemData.json*
Sumber: Dokumen Penulis

Dengan data yang sudah berbentuk JSON seperti ini, kita sudah siap untuk melakukan pemrosesan berikutnya. Data masih bersifat "kotor" dengan banyak item yang tidak seharusnya dipertimbangkan. Contohnya, terdapat beberapa item yang tidak dapat dimiliki oleh pemain. Sebaliknya, item tersebut (uniknya) diberikan untuk *minion* dan *turret*. Selain itu, terdapat beberapa item lainnya yang tidak dapat dibeli di *shop*. Item tersebut berasal dari mode permainan lainnya yang memberikan item tersebut ke pemainnya langsung secara berkala. Tidak hanya itu, beberapa tipe item tidak memberikan bonus statistik secara langsung ke pemain, melainkan memberikan keuntungan dalam bentuk *buff* sementara untuk karakter atau *map control* untuk tim.

Item-item seperti ini tidak akan diikutsertakan dalam solusi. Maka dari itu, kita akan melakukan pra-pemrosesan data berupa memfilter item yang tidak layak digunakan dalam perhitungan *build.* Dengan demikian, kita akan menentukan beberapa batasan untuk item yang akan diikutsertakan. Batasan tersebut adalah sebagai berikut:

- Mode Summoner's Rift
  Item yang diikutsertakan hanyalah item yang terdapat pada mode permainan klasik atau disebut juga dengan "Summoner's Rift 5v5". Item yang eksklusif pada mode permainan lainnya akan diabaikan. Atribut ini dapat dilihat pada entri "modes" di setiap item.
- Tipe Item
  Setiap item memiliki atribut tipe. Tipe item yang dapat dibeli pemain dan memberikan bonus statistik secara langsung meliputi "Starter", "Basic", "Epic", "Legendary", dan "Boots". Tipe selain ini mencakup "Potion" dan "Consumable", yang merupakan item sekali pakai, serta "Minion" dan "Turret", yang merupakan item untuk entitas bukan pemain masing-masing. Tipe selain yang memberikan bonus statistik secara langsung akan diabaikan.

Dengan menggunakan batasan tersebut, kita sudah mengurangi item yang dipertimbangkan dari berjumlah 319 item menjadi berjumlah 198 item. Pengurangan ini akan bermanfaat besar dalam komputasi karena mengurangi jumlah iterasi yang diperlukan secara signifikan.

Langkah berikutnya adalah menentukan atribut-atribut yang penting untuk dipertimbangkan dalam algoritma. Hal ini dapat ditentukan dengan memperhatikan setiap aturan sistem item yang sudah dibahas sebelumnya. Dengan memperhatikan setiap aturan tersebut, ditemukan atribut sebagai berikut:

- Statistik Item
  Terdapat pada atribut bersarang "stats".
- Biaya Item
  Terdapat pada atribut "buy".
- Batasan Kategori
  Terdapat pada atribut daftar "type".
- Batasan Kelompok
  Terdapat pada atribut opsional "itemlimit" dan "itemlimit2".

Dengan demikian, daftar atribut yang akan disimpan meliputi nama, tipe, [batasan kelompok], harga, dan [stats].

NAME	TYPE	ITEMLIMIT	COST	AH	HP	MR	ARMOR	AP	⋯
ABYSSAL MASK	Legendary	[None]	2650	15	350	45	0	0	⋯
AEGIS OF THE LEGION	Epic	[None]	1100	10	0	25	25	0	⋯
AETHER WISP	Epic	[None]	900	0	0	0	0	30	⋯
AMPLIFYING TOME	Basic	[None]	400	0	0	0	0	20	⋯
ARCHANGEL'S STAFF	Legendary	['Manaflow', 'Lifeline']	2900	25	0	0	0	70	⋯
ARDENT CENSER	Legendary	[None]	2200	0	0	0	0	45	⋯
ARMORED ADVANCE	Boots	[None]	1700	0	0	0	40	0	⋯
AXIOM ARC	Legendary	[None]	3000	20	0	0	0	0	⋯

Tabel III.3 *Cuplikan itemData.csv*
Sumber: Dokumen Penulis

Setelah data berhasil diproses dan disimpan dalam format CSV yang terstruktur, tahap selanjutnya adalah algoritma optimasi. Untuk kasus ini, model knapsack yang paling sesuai adalah *Unbounded Conditional Knapsack*. "*Unbounded*" karena pemain secara teoretis bisa membeli item komponen yang sama berulang kali, dan "*Conditional*" karena adanya batasan unik yang meliputi jumlah *slot*, kategori item, dan kelompok item.

Masalah *Unbounded Conditional Knapsack* ini diselesaikan dengan menggunakan metode Program Dinamis, sesuai dengan dasar teori yang telah dijelaskan. Pemodelan ini melibatkan pendefinisian *state* dan hubungan rekursif untuk mencapai solusi optimal.

1. Definisi State
   Setiap sub-masalah dalam optimasi ini dapat didefinisikan oleh sebuah *state* yang terdiri dari dua parameter utama:
   w : Sisa anggaran *gold* yang tersedia.
   k : Sisa *slot* item yang dapat diisi.
   Dengan demikian, kita dapat mendefinisikan sebuah fungsi tujuan $f(w,k)$ yang menyatakan nilai statistik maksimum yang bisa didapatkan dengan anggaran w dan k *slot* item.
2. Hubungan Rekursif
   Inti dari solusi program dinamis adalah hubungan rekursif yang membangun solusi dari sub-masalah yang lebih kecil. Untuk setiap *state* (w,k), keputusan yang harus diambil adalah "item mana (dari himpunan item I) yang harus dipilih untuk mengisi salah satu dari k slot?".
   Jika kita memilih item $i \in I$, maka kita akan mendapatkan nilai statistik sebesar $nilai(i)$ dan sisa anggaran akan menjadi $w - harga(i)$ dengan sisa *slot* $k-1$.
   Berdasarkan Prinsip Optimalitas, nilai total yang akan kita dapatkan dari keputusan ini adalah $nilai(i) + f(w - harga(i), k-1)$, dengan asumsi $f(w - harga(i), k-1)$ adalah solusi optimal untuk sub-masalah tersebut.
   Karena tujuan kita adalah maksimasi, kita akan memilih item i yang memberikan nilai total terbesar. Dengan demikian, hubungan rekursifnya dapat dirumuskan sebagai berikut:
   $$f(w,k) = \max_{i \in I}\{nilai(i) + f(w - harga(i), k-1)\}$$
   Rumusan ini berlaku untuk semua item i yang memenuhi syarat:
   - $harga(i) <= w$ (item dapat dibeli dengan anggaran yang ada).
   - Penambahan item i tidak melanggar batasan kondisional (pasangan terlarang).
3. Kasus Dasar (*Base Case*)
   Hubungan rekursif memerlukan sebuah kasus dasar untuk menghentikan rekursi:
   - $f(w,0) = 0$
     Jika tidak ada *slot* yang tersedia, maka tidak ada item yang bisa dibeli, sehingga total nilai statistiknya adalah 0.
   - $f(0,k) = 0$
     Jika tidak ada *gold* yang tersedia, maka tidak ada item yang bisa dibeli, sehingga total nilai statistiknya adalah 0.
4. Solusi Akhir
   Dengan menggunakan pendekatan *bottom-up*, program akan mengisi tabel $DP(dp[w][k])$ yang merepresentasikan fungsi $f(w,k)$ untuk setiap kemungkinan nilai w dan k. Solusi akhir dari masalah optimasi ini adalah nilai yang tersimpan pada:
   $$f(TotalGold, 6)$$
   Di mana *TotalGold* adalah anggaran awal yang diberikan pengguna dan 6 adalah jumlah *slot* item maksimal dalam *League of Legends*. Nilai inilah yang merepresentasikan statistik maksimum yang dapat dicapai. Kombinasi item yang menghasilkan nilai ini kemudian direkonstruksi dengan menelusuri kembali pilihan-pilihan optimal yang telah disimpan selama pengisian tabel.

IV. IMPLEMENTASI

A. Penanganan Batasan Item

Langkah pertama dalam implementasi algoritma adalah menangani batasan-batasan unik dari sistem *item League of Legends*. Algoritma tidak boleh menghasilkan kombinasi item yang "ilegal" menurut aturan permainan. Untuk itu, sebuah fungsi bernama $forbidden\_pair\_generator()$ dibuat.

Fungsi ini bertugas untuk menghasilkan sebuah struktur data (set) yang berisi semua pasangan item yang tidak boleh dibeli secara bersamaan.Aturan yang diterapkan adalah sebagai berikut:

- Batasan Kelompok (*itemlimit*)
  Item yang memiliki *itemlimit* (pasif unik) yang sama tidak dapat digabungkan. Misalnya, dua item dengan pasif unik *'Lifeline'* akan membentuk pasangan terlarang.
- Batasan Kategori (*'Boots'*)
  Pemain hanya boleh memiliki satu item dari kategori *'Boots'*. Fungsi ini akan menghasilkan pasangan terlarang untuk setiap kombinasi dua item *'Boots'*.

- Batasan Item Legendaris

  Pemain tidak boleh memiliki duplikat item *'Legendary'*. Maka, setiap item *'Legendary'* akan dipasangkan dengan dirinya sendiri sebagai pasangan terlarang.

Hasil dari fungsi ini adalah sebuah daftar lengkap "pasangan terlarang" yang akan menjadi input krusial bagi algoritma knapsack untuk memastikan validitas solusi.

B. Algoritma Unbounded Conditional Knapsack

Inti dari solusi adalah sebuah fungsi bernama $unbounded\_conditional\_knapsack()$. Fungsi ini mengimplementasikan pendekatan program dinamis untuk mengisi sebuah tabel DP dua dimensi, yaitu $dp[w][k]$.

- Definisi State

  $dp[w][k]$ merepresentasikan nilai statistik maksimum yang bisa dicapai dengan total anggaran gold sebesar w dan menggunakan tepat k buah item.
- Proses Iteratif

  Algoritma bekerja secara *bottom-up*, mengiterasi setiap kemungkinan anggaran w (dari 0 hingga kapasitas maksimum) dan setiap kemungkinan jumlah item k (dari 1 hingga 6).
- Pengecekan Kondisi

  Untuk setiap *state* (w, k), algoritma akan mencoba untuk menambahkan setiap item i yang ada. Sebelum menambahkan, algoritma akan melakukan pengecekan: Apakah penambahan item i ke dalam kombinasi item yang sudah ada akan menciptakan "pasangan terlarang" yang telah didefinisikan sebelumnya?
- Pembaruan Tabel DP

  Jika penambahan item i valid (tidak melanggar batasan) dan menghasilkan total statistik yang lebih tinggi dari yang tersimpan di dp[w][k], maka nilai pada tabel DP dan daftar kombinasi itemnya akan diperbarui.

Setelah seluruh tabel DP terisi, nilai maksimum yang ditemukan pada baris kapasitas *gold* yang ditentukan akan menjadi solusi optimalnya. Kombinasi item yang menghasilkan nilai tersebut dapat ditelusuri kembali (*backtrack*) dari tabel yang menyimpan jejak pilihan item.

C. Orkestrasi Solusi

Keseluruhan proses di atas diatur oleh sebuah fungsi utama bernama $solve(stat, gold)$. Fungsi ini bertindak sebagai orkestrator yang:

1. Menerima input dari pengguna berupa statistik yang ingin dimaksimalkan dan total *gold* yang tersedia.
2. Memuat data item yang sudah bersih dari itemData.csv.
3. Memanggil $forbidden\_pair\_generator()$ untuk menghasilkan batasan-batasan.
4. Menjalankan algoritma $unbounded\_conditional\_knapsack()$ dengan semua parameter yang relevan.
5. Menampilkan output akhir kepada pengguna berupa daftar item yang direkomendasikan, total statistik yang dicapai, dan total biaya yang dikeluarkan

```python
def unbounded_conditional_knapsack(weights, values, capacity, max_items, forbidden_pairs=None):
    n = len(weights)
    dp = [[0 for _ in range(max_items + 1)] for _ in range(capacity + 1)]
    item_choice = [[[] for _ in range(max_items + 1)] for _ in range(capacity + 1)]
    forbidden_pairs = set(forbidden_pairs or [])

    for w in range(capacity + 1):
        print(f"Progress: {w}/{capacity} ({(w / capacity) * 100:.1f}%)", end='\r', flush=True)
        for k in range(1, max_items + 1):
            for i in range(n):
                if weights[i] <= w:
                    prev_items = item_choice[w - weights[i]][k - 1]
                    # Check for forbidden pairs
                    if any((i, j) in forbidden_pairs or (j, i) in forbidden_pairs for j in prev_items):
                        continue
                    if dp[w - weights[i]][k - 1] + values[i] > dp[w][k]:
                        dp[w][k] = dp[w - weights[i]][k - 1] + values[i]
                        item_choice[w][k] = prev_items + [i]
    # noinspection DuplicatedCode
    print(f"Progress: {capacity}/{capacity} (100.0%)")

    # Find the best result
    max_value = 0
    best_items = []
    for k in range(1, max_items + 1):
        if dp[capacity][k] > max_value:
            max_value = dp[capacity][k]
            best_items = item_choice[capacity][k]
    return max_value, best_items

def forbidden_pair_generator(data):
    from itertools import combinations

    forbidden_pairs = set()
    # Only consider items with real limits (not [None])
    itemlimited = data[data['itemlimit'].apply(lambda x: x != [None])]

    # Map each limit to the indices of items that have it
    limit_to_indices = {}
    for idx, limits in itemlimited['itemlimit'].items():
        for limit in limits:
            limit_to_indices.setdefault(limit, []).append(idx)

    # For each limit, forbid all pairs of items sharing that limit
    for indices in limit_to_indices.values():
        for i in indices:
            forbidden_pairs.add((i, i))  # Add self-pair
        for i, j in combinations(indices, 2):
            forbidden_pairs.add((i, j))
            forbidden_pairs.add((j, i))  # If order matters

    # Add self-pairs for legendary items
    legendary_items = data[data['type'] == 'Legendary'].index
    for i in legendary_items:
        forbidden_pairs.add((i, i))

    # Add pairs of boots items
    boots_items = data[data['type'] == 'Boots'].index
    for i, j in combinations(boots_items, 2):
        forbidden_pairs.add((i, j))
        forbidden_pairs.add((j, i))
    for i in boots_items:
        forbidden_pairs.add((i, i))

    return forbidden_pairs

def solve(stat: str, gold: int):
    import pandas as pd
    import ast

    data = pd.read_csv('itemData.csv')
    data['itemlimit'] = data['itemlimit'].apply(ast.literal_eval)
    print(f"\nLoaded {len(data)} items.")

    # Filter out items with 0 value for the chosen stat
    data = data[data[stat] != 0]
    data = data.reset_index(drop=True)
    print(f"Filtered to {len(data)} items with nonzero {stat.upper()}.")

    weights = data['cost'].tolist()
    values = data[stat].tolist()
    capacity = gold
    max_items = 6
    forbidden_pairs = forbidden_pair_generator(data)

    print("\nStarting knapsack problem...")
    max_value, items = unbounded_conditional_knapsack(weights, values, capacity, max_items, forbidden_pairs)
    print("Knapsack problem completed.\n")

    print(f"Maximum {stat.upper()} in knapsack: {max_value}")
    print("Items included:")
    for i in items:
        item_name = data.iloc[i]['name']
        item_cost = data.iloc[i]['cost']
        print(f"- {item_name} (Cost: {item_cost}G, {stat.upper()}: {data.iloc[i][stat]})")
    print(f"Total cost: {sum(weights[i] for i in items)}G / {capacity}G")

def main():
    stats = {
        "ah": "Ability Haste",
        "hp": "Health",
        "mr": "Magic Resist",
        "armor": "Armor",
        "ap": "Ability Power",
        "ms": "Movement Speed",
        "mana": "Mana",
        "hsp": "Heal and Shield Power",
        "mp5": "Mana Regeneration",
        "msflat": "Flat Movement Speed",
        "crit": "Critical Strike Chance",
        "ad": "Attack Damage",
        "lethality": "Lethality",
        "as": "Attack Speed",
        "lifesteal": "Lifesteal",
        "mpen": "Magic Penetration",
        "hp5": "Health Regeneration",
        "gp10": "Gold per 10 seconds",
        "tenacity": "Tenacity",
        "hp5flat": "Flat Health Regeneration",
        "omnivamp": "Omnivamp",
        "critdamage": "Critical Strike Damage",
        "armpen": "Armor Penetration",
        "mpenflat": "Flat Magic Penetration"
    }

    import os
    os.system('cls' if os.name == 'nt' else 'clear')

    print(r"""
  /\/\   __ _ _ __ ___| |__   __ _| | |  /\/\   __ ___  __(_)_ __ ___ (_)_______ _ __
 /    \ / _` | '__/ __| '_ \ / _` | | | /    \ / _` \ \/ / | '_ ` _ \| |_  / _ \ '__|
/ /\/\ \ (_| | |  \__ \ | | | (_| | | |/ /\/\ \ (_| |>  <| | | | | | | |/ /  __/ |
\/    \/\__,_|_|  |___/_| |_|\__,_|_|_|\/    \/\__,_/_/\_\_|_| |_| |_|_/___\___|_|
    """)
    print("Your very own League of Legends item stat build maximizer!")
    print("\nAvailable stats:")
    for key, value in stats.items():
        print(f"{key.upper()}: {value}")
    print("\nThis program will help you maximize a specific stat in League of Legends items within a given gold budget.")

    stat = input("Enter the stat you want to maximize (e.g., 'AD', 'AP', etc.): ").strip().lower()
    if stat not in stats:
        print(f"\nInvalid stat '{stat}'. Please run the program again and choose a valid stat.\n")
        return

    gold = int(input("Enter the amount of gold available: "))

    solve(stat, gold)
    print("\nGood luck trying out your new build! :3\n")

if __name__ == "__main__":
    main()
```

Gambar IV.1 *Algoritma Utama*
Sumber: Dokumen Penulis

V. HASIL DAN PEMBAHASAN

Program diuji sebanyak lima kali dengan target statistik dan anggaran *gold* yang berbeda untuk melihat bagaimana algoritma berperilaku dalam berbagai kondisi. Statistik yang dipilih untuk dioptimalkan meliputi *Attack Damage* (AD), *Critical Strike Chance* (CRIT), *Health* (HP), *Attack Speed* (AS), dan *Lethality*.

Statistik yang Dioptimalkan	Anggaran Gold	Build Item yang Dihasilkan	Total Statistik	Total Biaya
AD	17	- Bloodthirster - Infinity Edge - Eclipse - Essence Reaver - Hubris - B. F. Sword	365	16.95
CRIT	10	- Navori Flickerblade - Phantom Dancer - Noonquiver - Noonquiver - Noonquiver - Cloak of Agility	125	9.8
HP	9	- Warmog's Armor - Heartsteel - Giant's Belt - Giant's Belt - Giant's Belt	2.95	8.8
AS	11	- Phantom Dancer - Nashor's Tooth - Wit's End - Scout's Slingshot - Gunmetal Greaves - Dagger	230	10.8
LETHALITY	13	- Serrated Dirk - Opportunity - Axiom Arc - Hubris - Profane Hydra	82	12.9

***Tabel V.1** Hasil percobaan*
Sumber: Dokumen Penulis

Berdasarkan hasil yang disajikan, beberapa analisis dan pembahasan dapat ditarik.

1. Validitas Solusi Optimal

   Secara konsisten, program berhasil menemukan kombinasi item yang memaksimalkan statistik target sesuai dengan anggaran yang diberikan. Algoritma program dinamis yang diimplementasikan pada knapsack.py terbukti mampu menavigasi ruang pencarian yang besar untuk menemukan solusi yang optimal secara matematis.

   Terlihat juga bahwa program secara cerdas mengisi sisa *gold* dengan item komponen yang paling efisien, seperti *B. F. Sword* pada uji coba AD dan *Serrated Dirk* pada uji coba *Lethality*.

2. Efektivitas Penanganan Batasan

   Uji coba ini juga memvalidasi keberhasilan fungsi $forbidden\_pair\_generator()$ dalam menerapkan aturan permainan:

   - Pada uji coba AD dan *Lethality*, meskipun banyak item *Legendary* yang dipilih, tidak ada satupun item yang diduplikasi.
   - Pada uji coba AS, program hanya memilih satu item *Boots* (*Gunmetal Greaves*), sesuai dengan aturan permainan.
   - Pada uji coba CRIT dan HP, program membeli beberapa item komponen (*Epic* atau Basic) yang sama, seperti "*Noonquiver*" dan "*Giant's Belt*". Hal ini valid karena aturan batasan duplikasi hanya berlaku untuk item *Legendary*. Ini menunjukkan bahwa algoritma secara akurat mematuhi batasan yang telah didefinisikan.

3. Analisis Praktikalitas *Build*

   Meskipun optimal secara matematis, jika dilihat dari kacamata seorang pemain, *build* yang dihasilkan sering kali tidak praktis.

   - *Build* AD dan *Lethality* menghasilkan output kerusakan yang sangat tinggi namun tidak memiliki *Attack Speed* dan sangat rentan karena tidak ada item defensif.
   - *Build* HP memberikan daya tahan yang luar biasa besar dari segi *health pool*, namun tanpa adanya *Armor* dan *Magic Resist*, karakter akan tetap mudah dikalahkan oleh lawan yang memiliki item penetrasi atau pemberi kerusakan berdasarkan persentase HP.
   - *Build* CRIT menghasilkan total 125% *Critical Strike Chance*, yang melebihi batas efektif 100% di dalam permainan. Ini menyoroti bahwa model tidak mengetahui semua mekanik permainan yang implisit, hanya data statistik mentah dari itemData.csv.

Kasus-kasus ini menegaskan kembali bahwa program ini berfungsi sebagai alat analisis teoretis yang kuat untuk memahami efisiensi item, namun tidak dapat menggantikan intuisi dan pertimbangan strategis seorang pemain yang harus menyeimbangkan berbagai faktor dalam sebuah pertandingan nyata.

VI. KESIMPULAN DAN SARAN

Secara keseluruhan, proyek ini berhasil memodelkan masalah optimasi item *League of Legends* sebagai *Unbounded Conditional Knapsack Problem* dan menyelesaikannya menggunakan Program Dinamis. Melalui pra-pemrosesan data yang sistematis, algoritma yang diimplementasikan terbukti mampu menghasilkan *build* yang optimal secara matematis untuk satu statistik tertentu sambil mematuhi batasan-batasan unik dalam permainan seperti item legendaris dan pasif unik. Hasil uji coba menunjukkan keberhasilan model secara teknis, sekaligus menyoroti adanya kesenjangan antara solusi teoretis yang fokus pada satu metrik dengan *build* praktis yang membutuhkan keseimbangan dan sinergi antar statistik dalam sebuah permainan nyata.

Untuk pengembangan selanjutnya, disarankan agar model diperluas untuk mendukung optimasi multi-objektif dengan fungsi nilai berbobot agar dapat menyeimbangkan beberapa statistik sekaligus. Selain itu, untuk meningkatkan relevansi praktis, model dapat diperkaya dengan memperhitungkan nilai dari kemampuan pasif dan aktif item (seperti efek *slow*, *shield*, atau *damage proc*), bukan hanya berfokus pada statistik mentahnya. Integrasi data *champion* yang spesifik seperti rasio *scaling* kemampuan juga menjadi langkah penting untuk membuat alat ini lebih kontekstual, kuat, dan mudah diakses oleh pemain.

VII. Lampiran

Program yang digunakan dalam implementasi makalah ini dapat diakses melalui repositori GitHub pada tautan: https://github.com/Darsua/LeagueBuildStatMaximizer

VIII. Ucapan Terima Kasih

Puji syukur penulis panjatkan ke hadirat Tuhan Yang Maha Esa atas limpahan rahmat dan karunia-Nya sehingga makalah yang berjudul "Penerapan Algoritma Knapsack untuk Maksimasi Statistik Melalui 'Build Item' di League of Legends" dapat diselesaikan dengan baik tanpa hambatan yang berarti.

Penulis juga mengucapkan terima kasih kepada dosen pengampu mata kuliah IF2211 Strategi Algoritma, terutama kepada Ibu Dr. Nur Ulfa Maulidevi, Bapak Dr. Rinaldi Munir, serta Bapak Menterico Adrian, S.T, M.T, atas ilmu, bimbingan, dan arahan yang telah diberikan selama perkuliahan. Segala dukungan yang diterima sangat berkontribusi dalam menyempurnakan isi dan kualitas makalah ini.

Referensi

[1] Item (League of Legends) | League of Legends Wiki | Fandom https://leagueoflegends.fandom.com/wiki/Item (League of Legends)

[2] Module:ItemData/data | League of Legends Wiki https://wiki.leagueoflegends.com/en-us/Module:ItemData/data

[3] Program Dinamis (Dynamic Programming) Bagian 1 https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/25-Program-Dinamis-(2025)-Bagian1.pdf

[4] Unbounded Knapsack (Repetition of items allowed) - GeeksforGeeks https://www.geeksforgeeks.org/dsa/unbounded-knapsack-repetition-items-allowed/

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, 24 Juni 2025

Darrel Adinarya Sunanda
13523061