Penerapan Algoritma Breadth-First Search dalam Penyelesaian Permainan Water Sort

Ahmad Farid Mudrika - 13522008[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]faridm0741@gmail.com

Abstrak—Algoritma Breadth-First Search (BFS) adalah salah satu algoritma fundamental dalam ilmu komputer yang digunakan untuk penelusuran graf. Makalah ini membahas penerapan algoritma BFS dalam penyelesaian permainan Water Sort, sebuah permainan teka-teki yang menantang pemain untuk mengurutkan air berwarna ke dalam tabung yang sesuai. Implementasi ini memanfaatkan BFS untuk menjelajahi semua kemungkinan langkah secara sistematis sehingga dapat menemukan solusi optimal. Makalah ini menguraikan proses penerapan algoritma BFS, termasuk representasi graf permainan, strategi penelusuran, dan efisiensi algoritma. Selain itu, dibahas juga tantangan dan solusi yang muncul selama implementasi.

Kata Kunci — Algoritma Breadth-First Search, Penelusuran Graf, Permainan Water Sort, Teka-teki, Optimalisasi.

I. PENGENALAN

Permainan Water Sort adalah sebuah permainan teka-teki populer yang menguji kemampuan pemain dalam mengurutkan air berwarna ke dalam tabung-tabung yang sesuai. Setiap tabung dapat berisi air dengan berbagai warna, dan tugas pemain adalah memindahkan air dari satu tabung ke tabung lain dengan aturan tertentu hingga setiap tabung hanya berisi satu warna air. Permainan ini menantang keterampilan logika dan strategi pemain, karena setiap langkah yang salah dapat membuat penyelesaian semakin rumit.

Algoritma Breadth-First Search (BFS) adalah salah satu algoritma pencarian graf yang terkenal dan banyak digunakan dalam berbagai aplikasi komputasi. BFS bekerja dengan menjelajahi semua node pada level yang sama sebelum melanjutkan ke level berikutnya, sehingga memastikan bahwa solusi yang ditemukan adalah yang paling pendek atau optimal dalam hal jumlah langkah. Dalam konteks permainan Water Sort, BFS dapat digunakan untuk menjelajahi semua kemungkinan urutan langkah yang dapat diambil untuk mencapai solusi yang diinginkan.

Penerapan algoritma BFS dalam permainan Water Sort menawarkan pendekatan sistematis untuk menjelajahi semua kemungkinan langkah secara efisien. Dengan BFS, kita dapat memastikan bahwa setiap kemungkinan urutan langkah dijelajahi secara menyeluruh, dan solusi yang ditemukan adalah solusi optimal dalam jumlah langkah paling sedikit. Hal ini sangat berguna dalam permainan Water Sort, di mana jumlah langkah yang diperlukan untuk mencapai solusi dapat dengan cepat bertambah besar jika pemain tidak berhati-hati.

Namun, penerapan BFS dalam permainan ini juga menghadirkan sejumlah tantangan. Salah satu tantangan utama adalah banyaknya kemungkinan state yang dapat terbentuk selama permainan, yang memerlukan penanganan yang efisien untuk menghindari eksplorasi berulang. Selain itu, mengingat semua state yang telah dijelajahi memerlukan memori yang cukup besar, terutama untuk permainan dengan banyak tabung dan warna air. Oleh karena itu, penting untuk mengimplementasikan BFS dengan strategi optimasi memori yang baik.

Selain itu, permainan Water Sort memiliki aturan dan batasan yang harus dipatuhi dalam setiap langkah, yang menambah kompleksitas penerapan BFS. Setiap kali air dipindahkan dari satu tabung ke tabung lain, algoritma harus memastikan bahwa langkah tersebut valid sesuai dengan aturan permainan. Langkah-langkah yang tidak valid atau tidak efisien harus dihindari untuk memastikan bahwa solusi yang ditemukan benar-benar optimal.

Dalam makalah ini, kita akan membahas secara mendalam penerapan algoritma BFS dalam penyelesaian permainan Water Sort. Pembahasan akan mencakup penjelasan teori dasar BFS, representasi permainan Water Sort sebagai graf, tantangan yang dihadapi dalam penerapan algoritma ini, serta evaluasi kinerja BFS dalam menyelesaikan permainan. Dengan memahami dan mengimplementasikan algoritma BFS secara tepat, diharapkan solusi optimal untuk permainan Water Sort dapat ditemukan dengan efisien, memberikan pemahaman yang lebih baik tentang bagaimana algoritma pencarian graf dapat diterapkan dalam konteks permainan teka-teki yang menantang.

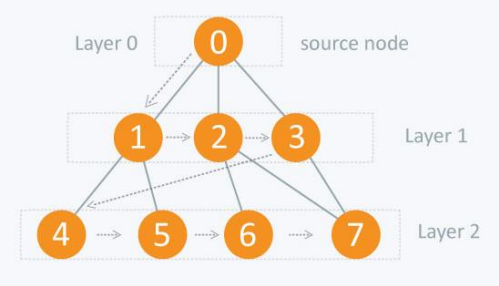
II. DASAR TEORI

A. Algoritma Breadth-First Search (BFS)

Algoritma Breadth-First Search (BFS) atau pelebaran terlebih dahulu adalah metode penelusuran graf yang digunakan untuk menjelajahi semua node pada level yang sama sebelum bergerak ke level berikutnya. BFS menggunakan struktur data antrian (queue) untuk

melacak node yang akan dijelajahi berikutnya. Berikut adalah langkah -langkah dasar dari algoritma BFS:

1. Mulai dari node awal (root) dan tambahkan ke dalam antrian.
2. Selama antrian tidak kosong, lakukan langkah berikut:
    - Ambil node terdepan dari antrian.
    - Kunjungi node tersebut dan tambahkan semua node tetangga yang belum dikunjungi ke dalam antrian.
3. Ulangi proses hingga semua node pada level yang sama telah dikunjungi sebelum berpindah ke level berikutnya.



Gambar 2.1 Pohon Pencarian BFS
https://www.hackerearth.com/practice/algorithms/graphs/breadth-first-search/tutorial/

B. Aturan Permainan Water Sort

Permainan Water Sort adalah sebuah teka-teki logika yang menantang pemain untuk mengurutkan air berwarna ke dalam tabung-tabung yang sesuai. Berikut adalah aturan dasar permainan Water Sort:

1. Tujuan Permainan:
   Pemain harus mengurutkan air berwarna ke dalam tabung-tabung sehingga setiap tabung hanya berisi satu warna air.

2. Set Up Awal:
   Pemain memulai dengan beberapa tabung yang berisi campuran air berwarna. Setiap tabung dapat memuat beberapa lapisan air dengan warna berbeda.
   Biasanya ada satu atau lebih tabung kosong yang dapat digunakan untuk membantu pemindahan air.

3. Langkah Permainan:
   Pemain dapat memindahkan lapisan atas air dari satu tabung ke tabung lainnya dengan beberapa aturan berikut:
   a. Hanya lapisan teratas dari tabung yang dapat dipindahkan.
   b. Pemindahan air hanya bisa dilakukan jika tabung tujuan memiliki cukup ruang untuk menerima lapisan air tersebut.
   c. Pemindahan hanya bisa dilakukan jika lapisan atas air di tabung tujuan memiliki warna yang sama atau jika tabung tujuan kosong.

4. Aturan Validasi:
   a. Pemain tidak diperbolehkan menuangkan air ke dalam tabung yang sudah penuh.
   b. Pemain tidak diperbolehkan menuangkan air yang warnanya berbeda ke dalam tabung yang sudah memiliki lapisan atas dengan warna berbeda.
   c. Pemain hanya bisa menuangkan air ke tabung yang memiliki ruang cukup untuk menerima lapisan air tanpa meluap.

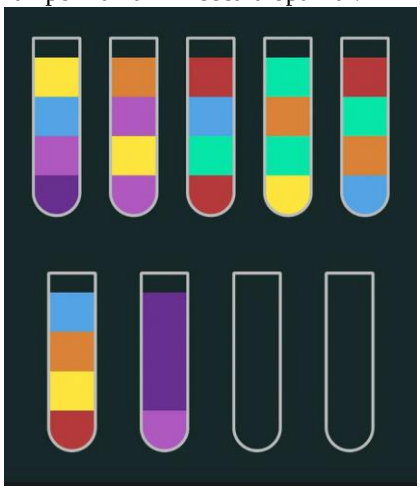
5. Kondisi Menang:
   Pemain memenangkan permainan ketika semua tabung berisi air yang terurut dengan setiap tabung hanya memiliki satu warna air atau tabung kosong.

C. Representasi Graf dalam Permainan Water Sort

Permainan Water Sort dapat direpresentasikan sebagai graf yang memiliki setiap state (keadaan) permainan sebagai node, dan setiap langkah yang memindahkan air dari satu tabung ke tabung lainnya sebagai edge (sisi) yang menghubungkan dua node. Dalam penerapan BFS, setiap state permainan akan dianalisis untuk menentukan langkah-langkah yang mungkin dan memindahkan air sesuai aturan permainan. Node tujuan adalah node yang semua botol di dalamnya kosong atau mengandung tepat 1 warna. Dari sini, permainan dapat diselesaikan dengan intuisi.

Misalkan ada beberapa tabung dengan berbagai warna air yang harus diurutkan. Setiap kombinasi distribusi air dalam tabung-tabung tersebut adalah satu node. Edge antara dua node muncul jika satu node dapat berubah menjadi node lainnya dengan satu langkah yang valid (memindahkan air dari satu tabung ke tabung lain sesuai aturan permainan).

Graf ini cenderung sangat besar dan kompleks, terutama jika jumlah tabung dan variasi warna air bertambah banyak. Oleh karena itu, representasi dan penelusuran yang efisien sangat penting untuk memecahkan permainan ini secara optimal.

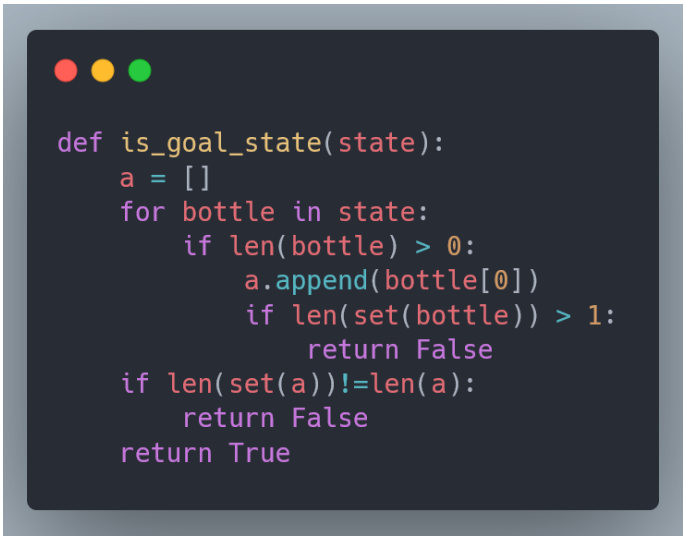


Gambar 2.2 Contoh State Permainan Water Sort
https://jerryan.medium.com/mastering-color-water-sort-puzzle-games-e7dcafec9a41

III. IMPLEMENTASI PROGRAM

Saya menggunakan bahasa Python dengan Python versi 3.12.0 untuk projek ini. Kode lengkapnya dapat dilihat di https://github.com/frdmmm/Water_sort.

A. Fungsi is_goal_state

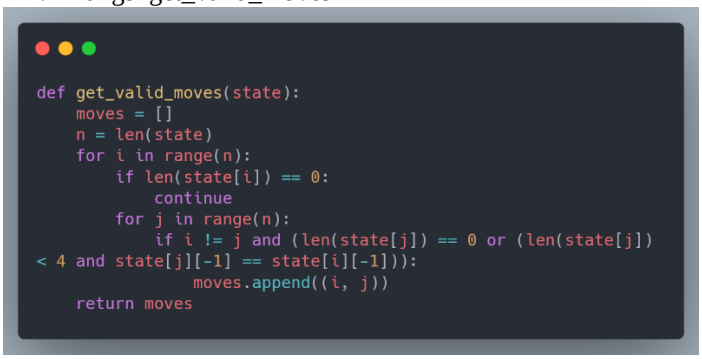


```python
def is_goal_state(state):
    a = []
    for bottle in state:
        if len(bottle) > 0:
            a.append(bottle[0])
            if len(set(bottle)) > 1:
                return False
    if len(set(a))!=len(a):
        return False
    return True
```

Gambar 3.1 Fungsi is_goal_state
(Sumber : Dokumentasi Pribadi)

Kode ini digunakan untuk memeriksa apakah suatu keadaan (state) dalam permainan Water Sort memenuhi syarat sebagai keadaan tujuan. Ini memeriksa setiap botol dalam keadaan tersebut, dan jika ada botol yang tidak kosong dan memiliki lebih dari satu jenis cairan di dalamnya, maka keadaan tersebut bukan merupakan keadaan tujuan. Jika dalam dua atau lebih botol berbeda terdapat cairan yang sama, keadaan tersebut juga bukan merupakan keadaan tujuan.

B. Fungsi get_valid_moves

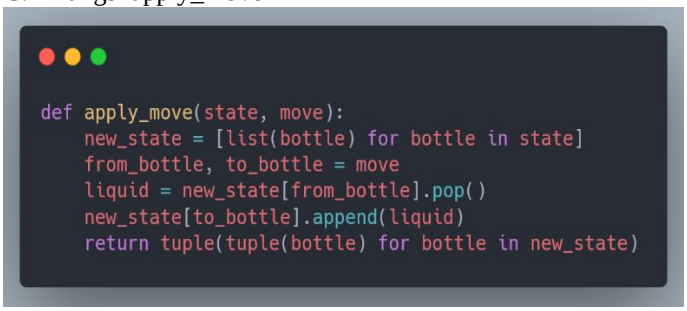


```python
def get_valid_moves(state):
    moves = []
    n = len(state)
    for i in range(n):
        if len(state[i]) == 0:
            continue
        for j in range(n):
            if i != j and (len(state[j]) == 0 or (len(state[j])
< 4 and state[j][-1] == state[i][-1])):
                moves.append((i, j))
    return moves
```

Gambar 3.2 Fungsi get_valid_moves
(Sumber : Dokumentasi Pribadi)

Fungsi get_valid_moves ini digunakan dalam solver water sort untuk menghasilkan semua gerakan valid yang dapat dilakukan dari suatu keadaan. Ini memeriksa kemungkinan menuangkan cairan dari satu tabung ke tabung lainnya dengan memperhatikan kapasitas tabung dan jenis cairan yang ada di ujung tabung.

C. Fungsi apply_move

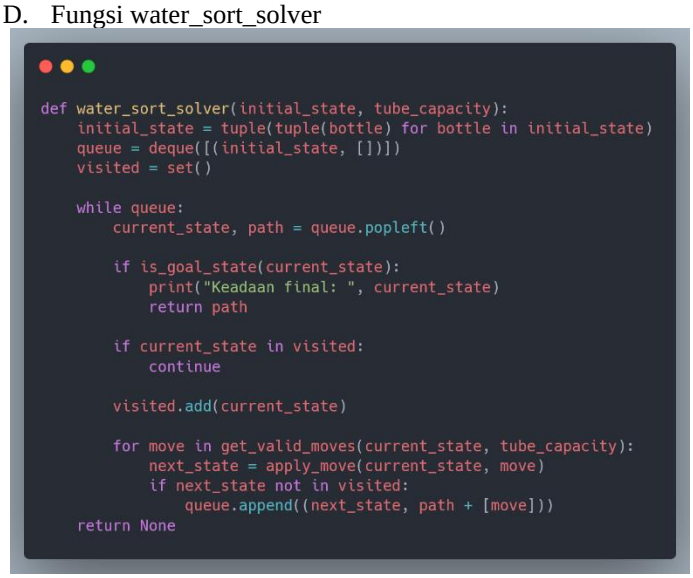


```python
def apply_move(state, move):
    new_state = [list(bottle) for bottle in state]
    from_bottle, to_bottle = move
    liquid = new_state[from_bottle].pop()
    new_state[to_bottle].append(liquid)
    return tuple(tuple(bottle) for bottle in new_state)
```

Gambar 3.3 Fungsi apply_move
(Sumber : Dokumentasi Pribadi)

Fungsi apply_move digunakan dalam solver water sort untuk menerapkan gerakan yang dipilih ke dalam keadaan saat ini. Ini menciptakan keadaan baru dengan menerapkan gerakan menuangkan cairan dari satu botol ke botol lainnya.

D. Fungsi water_sort_solver



```python
def water_sort_solver(initial_state, tube_capacity):
    initial_state = tuple(tuple(bottle) for bottle in initial_state)
    queue = deque([(initial_state, [])])
    visited = set()

    while queue:
        current_state, path = queue.popleft()

        if is_goal_state(current_state):
            print("Keadaan final: ", current_state)
            return path

        if current_state in visited:
            continue

        visited.add(current_state)

        for move in get_valid_moves(current_state, tube_capacity):
            next_state = apply_move(current_state, move)
            if next_state not in visited:
                queue.append((next_state, path + [move]))
    return None
```

Gambar 3.4 Fungsi water_sort_solver
(Sumber : Private Documentation)

Fungsi water_sort_solver adalah sebuah algoritma pencarian solusi yang bertujuan untuk menyelesaikan permainan water sort. Pertama, keadaan awal dari permainan, yang direpresentasikan dalam bentuk tupel botol, disiapkan. Kemudian, sebuah antrian (queue) diinisialisasi dengan keadaan awal dan jalur yang kosong. Sebuah set visited juga dibuat untuk melacak keadaan yang sudah dikunjungi sebelumnya.

Selama antrian tidak kosong, keadaan saat ini dan jalur yang terkait diambil dari depan antrian. Jika keadaan saat ini sudah merupakan keadaan tujuan, maka jalur yang ditemukan langsung dikembalikan sebagai solusi.

Jika keadaan saat ini belum pernah dikunjungi sebelumnya, maka langkah-langkah berikut dilakukan:

1. Keadaan saat ini ditandai sebagai telah dikunjungi dengan menambahkannya ke dalam set visited.
2. Selanjutnya, semua gerakan valid dari keadaan saat ini diperoleh menggunakan fungsi get_valid_moves.
3. Setiap gerakan valid diaplikasikan ke keadaan saat ini menggunakan fungsi apply_move untuk menghasilkan keadaan baru.

4. Keadaan baru tersebut bersama dengan jalur yang diperoleh dengan menambahkan gerakan saat ini ke jalur yang sudah ada ditambahkan ke dalam antrian.

Jika seluruh keadaan yang mungkin telah dieksplorasi tanpa menemukan solusi, maka fungsi mengembalikan None untuk menandakan bahwa tidak ada solusi yang ditemukan.

E. Cara Penggunaan Program

Program dapat digunakan dengan menjalankan ‘python main.py’ di terminal. Untuk mengubah parameter, dapat dilakukan dengan mengubah variable initial_state dan tube_capacity di dalam main.py.

```
initial_state = [
    [1, 2, 2, 1],
    [3, 1, 3, 2],
    [3, 1, 3, 2],
    [],
    []
]

tube_capacity = 4
```

Gambar 3.5 Variabel persoalan
(Sumber : Dokumentasi Pribadi)

Setiap list mewakili sebuah tube, dan nilai unik apapun digunakan untuk mewakili warna, dengan warna teratas di tube berada di akhir list. Pada contoh ini, saya menggunakan integer sebagai pengganti warna.

IV. ANALISIS

Berikut beberapa test case yang dilakukan

```
initial_state = [
    [1, 2, 2, 1],
    [3, 1, 3, 2],
    [3, 1, 3, 2],
    [],
    []
]

tube_capacity = 4
```

Gambar 4.1 Parameter Test Case 1
(Sumber : Dokumentasi Pribadi)

```
Keadaan final: ((), (3, 3, 3, 3), (), (1, 1, 1, 1), (2, 2, 2, 2))
Solution found:
Move liquid from bottle 0 to bottle 3
Move liquid from bottle 0 to bottle 4
Move liquid from bottle 0 to bottle 4
Move liquid from bottle 0 to bottle 3
Move liquid from bottle 1 to bottle 4
Move liquid from bottle 1 to bottle 0
Move liquid from bottle 1 to bottle 3
Move liquid from bottle 0 to bottle 1
Move liquid from bottle 2 to bottle 4
Move liquid from bottle 2 to bottle 1
Move liquid from bottle 2 to bottle 3
Move liquid from bottle 2 to bottle 1
```

Gambar 4.2 Keluaran Test Case 1
(Sumber : Dokumentasi Pribadi)

```
initial_state = [
    [1, 2, 2, 1],
    [3, 1, 4, 2],
    [4, 5, 3, 2],
    [5, 1, 3, 2],
    [4, 5, 3, 1],
    [],
    []
]

tube_capacity = 4
```

Gambar 4.3 Parameter Test Case 2
(Sumber : Dokumentasi Pribadi)

```
No solution found
```

Gambar 4.4 Keluaran Test Case 2
(Sumber : Dokumentasi Pribadi)

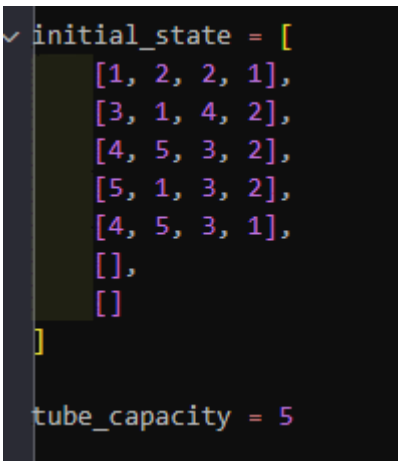
```
initial_state = [
    [1, 2, 2, 1],
    [3, 1, 4, 2],
    [4, 5, 3, 5],
    [5, 4, 3, 2],
    [4, 5, 3, 1],
    [],
    []
]

tube_capacity = 4
```

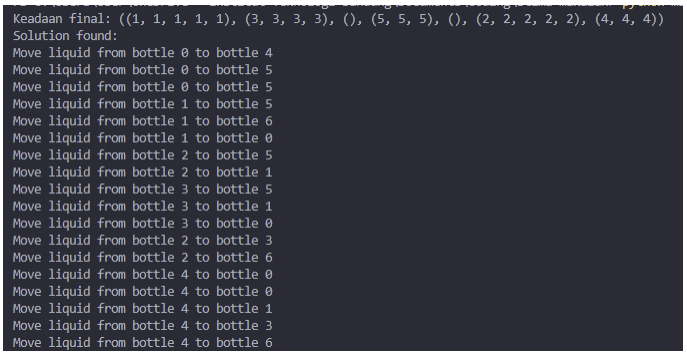
Gambar 4.5 Parameter Test Case 3
(Sumber : Dokumentasi Pribadi)

```
Keadaan final: ((4, 4, 4, 4), (3, 3, 3, 3), (), (5, 5, 5, 5), (), (1, 1, 1, 1), (2, 2, 2, 2))
Solution found:
Move liquid from bottle 0 to bottle 5
Move liquid from bottle 0 to bottle 6
Move liquid from bottle 0 to bottle 6
Move liquid from bottle 0 to bottle 5
Move liquid from bottle 1 to bottle 6
Move liquid from bottle 1 to bottle 0
Move liquid from bottle 1 to bottle 5
Move liquid from bottle 3 to bottle 6
Move liquid from bottle 3 to bottle 1
Move liquid from bottle 3 to bottle 0
Move liquid from bottle 2 to bottle 3
Move liquid from bottle 2 to bottle 1
Move liquid from bottle 2 to bottle 3
Move liquid from bottle 2 to bottle 0
Move liquid from bottle 4 to bottle 5
Move liquid from bottle 4 to bottle 1
Move liquid from bottle 4 to bottle 3
Move liquid from bottle 4 to bottle 0
```

Gambar 4.6 Keluaran Test Case 3
(Sumber : Dokumentasi Pribadi)



Gambar 4.7 Parameter Test Case 4
(Sumber : Dokumentasi Pribadi)



Gambar 4.8 Keluaran Test Case 4
(Sumber : Dokumentasi Pribadi)

A. Kompleksitas Ruang

Kompleksitas ruang dari algoritma BFS dipengaruhi oleh jumlah node yang disimpan dalam antrian (queue) dan himpunan (set) untuk melacak node yang sudah dikunjungi.

1. Antrian (Queue):

Pada kasus terburuk, antrian BFS dapat menyimpan semua node pada level terdalam dari pencarian. Jumlah node pada level terdalam bisa mencapai jumlah node maksimal dalam graf, yang kira-kira sebesar jumlah total state yang mungkin.

2. Himpunan (Set):

Himpunan digunakan untuk melacak semua state yang sudah dikunjungi untuk menghindari pengecekan berulang. Ukuran himpunan juga kira-kira sebesar jumlah total state yang mungkin.

Jumlah total state yang mungkin bergantung pada jumlah botol n dan kapasitas tiap botol k. Untuk n botol yang masing-masing dapat menampung k cairan, jumlah kombinasi state dapat diestimasi sebagai $O(k+1)^{nk}$.

Sehingga, kompleksitas ruang adalah:

$$O((k+1)^{nk})$$

B. Kompleksitas Waktu

Kompleksitas waktu dari algoritma BFS ditentukan oleh jumlah state yang dieksplorasi dan jumlah operasi yang dilakukan untuk setiap state.

1. Jumlah State yang Dieksplorasi:

Pada kasus terburuk, algoritma BFS akan mengeksplorasi semua state yang mungkin. Jumlah state ini kira-kira $(k+1)^{nk}$, seperti pada kompleksitas ruang.

2. Operasi per State:

Untuk setiap state, algoritma akan menghasilkan semua move yang valid. Pada setiap langkah, sebuah botol dapat memindahkan cairan ke botol lainnya, menghasilkan $O(n^2)$ kemungkinan move (karena setiap botol dapat mencoba menuang ke botol lainnya). Setiap move membutuhkan waktu konstan $O(1)$ untuk mengubah state dan menambahkan state baru ke antrian dan himpunan.

Jadi, kompleksitas waktu untuk BFS dapat diestimasi sebagai:

$$O((k+1)^{nk} \cdot n^2).$$

V. PEMBAHASAN

Berikut adalah beberapa kelebihan dan kekurangan penggunaan algoritma Breadth-First Search dalam penyelesaian permainan Water Sort.

Kelebihan:

1. Solusi Optimal
   BFS secara alami menjamin solusi optimal dalam hal jumlah langkah, karena menjelajahi semua node pada kedalaman(langkah) saat ini sebelum berpindah ke kedalaman selanjutnya.
2. Kesederhanaan
   Algoritma BFS sederhana untuk diimplementasikan dan mudah dipahami.
3. Kelengkapan
   Jika solusi ada, BFS pasti akan menemukannya. BFS tidak akan berhenti sampai solusi ditemukan atau semua kemungkinan sudah dieksplorasi.

Kekurangan:

1. Pengguanaan Memori Sangat Besar
   BFS membutuhkan banyak memori karena harus menyimpan semua state yang pernah di eksplorasi dalam antrian dan himpunan *visited*.
2. Kompleksitas Waktu
   Kompleksitas waktu BFS dalam permasalahan ini adalah eksponensial, $O((k+1)^{nk} \cdot n^2)$, yang tidak praktis untuk permainan dengan jumlah atau kapasitas botol yang besar.
3. Skalabilitas
   BFS tidak skalabel untuk persoalan dengan kemungkinan keadaan sangat besar, karena waktu dan ruang yang dibutuhkan akan meningkat secara eksponensial sesuai dengan jumlah dan kapasitas botol.

VI. KESIMPULAN

Algoritma BFS adalah pilihan yang baik ketika Anda memerlukan solusi optimal dalam hal jumlah langkah dan ketika ukuran masalahnya cukup kecil sehingga memori dan waktu komputasi yang diperlukan masih dalam batas yang wajar. Namun, untuk permainan Water Sort yang lebih kompleks dengan banyak botol atau kapasitas botol yang besar, algoritma BFS bisa menjadi sangat tidak efisien karena konsumsi memori dan waktu yang tinggi.

Untuk masalah yang lebih kompleks, alternatif seperti algoritma A* (A-star) yang menggunakan heuristik untuk membimbing pencarian menuju solusi, atau pendekatan berbasis Deep Learning, mungkin lebih efisien.

VII. UCAPAN TERIMA KASIH

Penulis ingin menyampaikan rasa terima kasih yang tulus kepada beberapa pihak atas kontribusi mereka terhadap makalah ini. Pertama-tama, penulis menyampaikan rasa syukur yang mendalam kepada Allah yang telah memberikan petunjuk selama proses belajar dan penulisan. Selain itu, penulis juga mengakui dukungan dan pengajaran yang sangat berharga yang diterima dari dosen Strategi Algoritma IF2210 ITB, Ibu Nur Ulfa Maulidevi. Pengetahuan dan bimbingannya telah memperkaya pengalaman belajar di kelas secara signifikan. Ucapan terima kasih yang istimewa juga ditujukan kepada keluarga dan teman-teman penulis atas dukungan mereka yang tak tergoyahkan sepanjang semester ini.

PRANALA GITHUB

https://github.com/frdmmm/Water_Sort

REFERENCES

[1] https://www.geeksforgeeks.org/level-order-tree-traversal/ diakses pada 10 Juni 2024.
[2] https://jerryan.medium.com/mastering-color-water-sort-puzzle-games-e7dcafec9a41 diakses pada 9 Juni 2024.
[3] https://www.hackerearth.com/practice/algorithms/graphs/breadth-first-search/tutorial/ diakses pada 11 Juni 2024.
[4] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/BFS-DFS-2021-Bag1-2024.pdf diakses pada 8 Juni 2024.
[5] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf diakses pada 8 Juni 2024.

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

Ahmad Farid Mudrika 13522008