Penerapan Algoritma IDS dan Branch and Bound dalam Penyelesaian Permainan 8-Puzzle

Suthasoma Mahardhika Munthe - 13522098
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13522098@std.stei.itb.ac.id

***Abstract*—Permainan 8-Puzzle adalah salah satu teka-teki yang terkenal dalam bidang kecerdasan buatan dan teori permainan. Penelitian ini membahas kinerja penerapan dua algoritma pencarian, yaitu Iterative Deepening Search (IDS) dan algoritma Branch and Bound dalam penyelesaian permainan 8-Puzzle. Algoritma IDS menggabungkan kelebihan dari algoritma Breadth-First Search (BFS) dan Depth-First Search (DFS) yang memungkinkan menghasilkan solusi optimal dengan kebutuhan ruang memori yang rendah, tetapi memerlukan waktu eksekusi yang lebih lama. Algoritma Branch and Bound menggunakan teknik pendekatan pemangkasan simpul yang tidak menjanjikan sehingga mempercepat proses pencarian dengan mengurangi jumlah simpul yang perlu dievaluasi. Penelitian ini melakukan evaluasi terhadap kinerja kedua algoritma ini dengan menguji beberapa uji kasus 8-Puzzle. Hasil penelitian menunjukkan bahwa algoritma Branch and Bound memiliki keunggulan dalam kasus permainan 8-Puzzle.**

Keywords—8-Puzzle; Iterative Deepening Search; Branch and Bound; Algoritma Pencarian; Optimasi.

I. PENDAHULUAN

Permainan 8- Puzzle adalah salah satu teka-teki yang terkenal dalam bidang kecerdasan buatan dan teori permainan. Permainan ini terdiri dari sebuah papan 3x3 yang berisi ubin-ubin dengan angka bernomor 1 hingga 8 serta satu ubin kosong yang dapat digeser dengan ubin berangka lainnya. Tujuan dari permainan ini adalah menyusun ubin-ubin sesuai dengan urutan numerik dengan ubin kosong berada pada posisi sudut kanan bawah. Meskipun aturan permainan ini cukup sederhana, kompleksitas yang dihadapi dalam mencari solusi optimal permainan 8-Puzzle menjadi tantangan yang menarik dalam penelitian algoritma pencarian[1].

Penyelesaian permainan ini menghadapkan kita pada masalah optimasi yang memerlukan pendekatan algoritmik yang efisien. Beberapa algoritma telah dikembangkan untuk menangani permasalahan ini, di antaranya adalah algoritma Iterative Deepening Search (IDS) dan algoritma Branch and Bound. Algoritma IDS merupakan algoritma yang dikembangkan dengan mengambil keuntungan yang dimiliki dengan pendekatan pencarian dalam (Depth-First Search) dan pencarian melebar (Breadth-First Search). Hal ini memungkinkan IDS untuk menemukan solusi optimal tanpa memerlukan memori yang besar[2]. Namun, proses pencarian dapat memakan waktu lebih lama karena waktu adanya proses berulang yang dilakukan setiap kali terjadi perubahan kedalaman pencarian.

Sementara itu, algoritma Branch and Bound menggunakan pendekatan yang berbeda dengan melakukan pemangkasan pada cabang-cabang yang tidak menjanjikan[2]. Algoritma ini menggunakan sebuah fungsi *bound* yang berguna untuk memperkirakan biaya minimum (kasus minimasi) yang kemudian dipakai sebagai nilai untuk memilih cabang tertentu untuk diperiksa. Jika biaya yang didapat lebih besar dari suatu solusi terbaik saat ini, maka cabang-cabang itu akan dipangkas sehingga jumlah cabang yang harus diperiksa akan berkurang. Pemangkasan ini menjadi sebuah teknik dalam mempercepat proses pencarian dan mempersempit ruang pencarian[3].

Penelitian ini bertujuan untuk melakukan eksplorasi terhadap penerapan dan kinerja kedua algoritma ini dalam penyelesaian permainan 8-Puzzle. Penelitian ini akan dimulai dengan pembahasan prinsip kerja algoritma IDS dan Branch and Bound. Selanjutnya akan dilakukan evaluasi kinerja kedua algoritma dengan melakukan pengujian untuk beberapa kasus tertentu pada permainan 8-Puzzle. Aspek-aspek yang akan dievaluasi meliputi waktu eksekusi, jumlah simpul yang dieksplorasi, dan kualitas solusi yang dihasilkan.

Analisis secara komparatif akan dilakukan terhadap dua algoritma ini. Analisis ini diharapkan dapat menemukan keunggulan dan kelemahan masing-masing metode dalam penyelesaian permainan 8-Puzzle. Selain itu, penelitian ini bertujuan untuk menemukan rekomendasi kondisi di mana salah satu algoritma lebih baik dari algoritma yang lain. Hasil dari penelitian ini dapat pula diaplikasikan masalah yang lain dengan karakteristik serupa.

Penelitian ini diharapkan dapat memberikan kontribusi yang signifikan dalam bidang kecerdasan buatan, khususnya dalam pengembangan dan pemilihan algoritma yang lebih baik dalam masalah pencarian yang efektif dan efisien. Selain itu, pemahaman yang lebih terhadap bagaimana algoritma IDS dan Branch and Bound berinteraksi terhadap masalah seperti 8-Puzzle, pembaca dapat mengembangkan solusi yang lebih baik untuk berbagai tantangan optimasi yang lebih kompleks.

II. LANDASAN TEORI

A. *Teori Permainan*

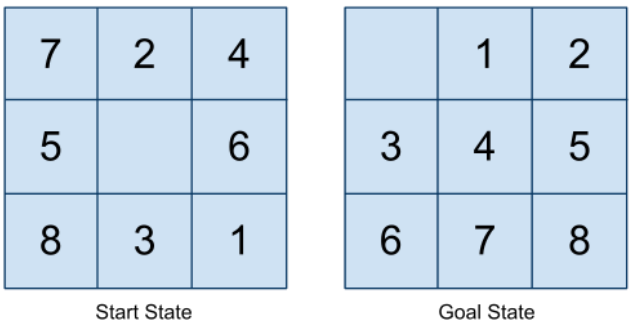
Teori permainan adalah perpaduan dari ilmu matematika dan ekonomi yang mempelajari model matematika dan interaksi antara beberapa agen yang mana setiap keputusan yang dibuat agen tertentu mempengaruhi hasil yang diperoleh agen lainnya. Teori bertujuan untuk memodelkan situasi kerjasama atau konflik dan memberikan solusi optimal atas tindakan yang rasional.

Teori permainan mencakup beberapa jenis permainan, yang dapat diklasifikasikan menjadi beberapa kriteria:

1) *Jumlah pemain: permainan dua atau lebih dari dua pemain[4].*
2) *Jumlah informasi:* permainan informasi sempurna yang menyediakan semua informasi untuk semua pemain dan permainan informasi tidak sempurna di mana beberapa informasi tersembunyi[4].
3) *Jumlah strategi:* permainan strategi murni tanpa probabilitas dan permainan strategi campuran yang disertai elemen probabilitas[4].
4) *Interaksi pemain:* permainan *zero-sum* di mana keuntungan salah satu pemain merupakan kerugian bagi pemain lain dan permainan *non-zero-sum* yang mana keuntungan dan kerugian tidak saling meniadakan[4].

Permainan dapat dimodelkan menggunakan matriks *payoff* untuk permainan dua pemain atau pohon keputusan untuk permainan berurutan[5]. Misalnya, permainan catur dapat dimodelkan sebagai permainan informasi sempurna dengan setiap pemain mengetahui gerakan yang dilakukan sebelumnya untuk membuat suatu gerakan berdasarkan strategi yang optimal.

Permainan 8-Puzzle adalah salah satu dari permainan deterministik dengan tujuan menyusun ubin dengan urutan bernomor satu sampai delapan pada papan berukuran 3x3 dengan urutan yang benar. Permainan ini dapat dimodelkan dengan model graf dengan simpul menunjukkan konfigurasi papan dan setiap tepi atau sisi menunjukkan transisi perubahan (pergeseran ubin kosong) yang valid pada papan[2].



Gambar 1. Permainan 8-Puzzle
Sumber: https://en.wikipedia.org/wiki/File:8puzzle_example.svg

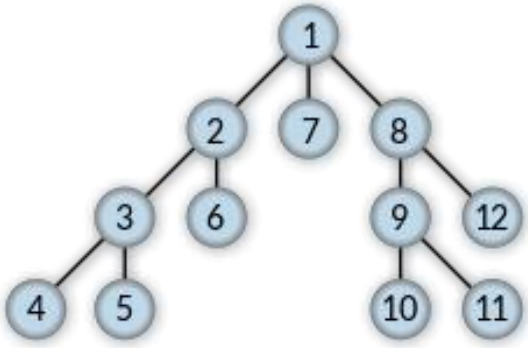
Solusi optimal pada permainan ini adalah solusi yang dihasilkan dari strategi optimal untuk menentukan langkah atau tahap pergeseran ubin kosong yang akan menuntun permainan ke solusi dengan jumlah tahap minimal[6]. Algoritma pencarian seperti algoritma Iterative Deepening Search (IDS) dan Branch and Bound digunakan untuk mencari solusi optimal dari teka-teki seperti 8-Puzzle[2].

B. *Algoritma Pencarian*

Dalam konteks teori permainan, algoritma memainkan peran penting dalam menemukan solusi yang optimal untuk berbagai permainan, termasuk permainan 8-Puzzle. Dua algoritma pencarian yang mendasar adalah algoritma Depth-First Search (DFS) dan algoritma Breadth-First Search (BFS). Kedua algoritma ini memiliki pendekatan yang berbeda dalam menjelajahi pohon pencarian yang dibentuk selama proses pencarian, yang mempengaruhi efektivitas dan efisiensinya dalam menemukan solusi.

DFS adalah algoritma pencarian yang menjelajahi pohon status atau graf dengan memulai pencarian dan akar dan menjelajahi sejauh mungkin sebelum kembali pada simpul sebelumnya[1]. Eksplorasi yang dilakukan mirip dengan struktur data *stack* (tumpukan). Berikut ini adalah langkah-langkah umum dari DFS:

- Memulai pencarian dari simpul awal.
- Tandai simpul yang sudah dikunjungi sebagai simpul yang telah dikunjungi.
- Untuk setiap simpul tetangga yang terhubung langsung dengan simpul saat ini lakukan proses DFS secara rekursif.
- Jika solusi yang diinginkan tercapai, proses pencarian selesai.
- Jika tidak ada lagi simpul yang dapat dikunjungi dari simpul saat ini kembali ke simpul sebelumnya dan coba simpul lain yang belum dikunjungi.

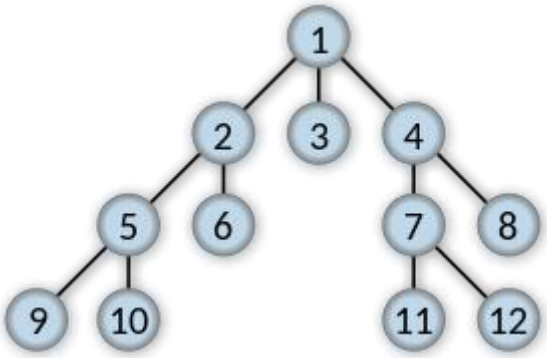


Gambar 2. Proses Pencarian dengan Algoritma DFS
Sumber: https://en.wikipedia.org/wiki/Depth-first_search

DFS memiliki kelebihan dalam kebutuhan ruang memori yang rendah karena sifatnya yang hanya perlu menyimpan data untuk simpul yang ada dalam jalur pencarian saat ini hingga mencapai simpul daun. Namun, proses DFS dapat terjebak pada proses yang sangat lama jika ukuran graf atau pohon pencarian sangat besar[1].

Breadth-First Search (BFS) adalah algoritma pencarian yang menjelajahi pohon atau graf dengan memulai proses dari simpul akar. Kemudian, pencarian dilakukan dengan pencarian pada simpul tetangga dari level proses saat ini sebelum melanjutkan proses pencarian pada level berikutnya[2]. BFS menggunakan struktur data *queue* (antrian) untuk melacak simpul-simpul yang perlu dieksplorasi. Berikut ini adalah langkah-langkah umum dari BFS:

- Mulai dari simpul awal yang dimasukkan ke dalam antrian.
- Tandai simpul tersebut sebagai simpul yang telah dikunjungi
- Selama antrian tidak kosong, lakukan proses pencarian dengan mengambil satu simpul untuk diperiksa dari antrian. Tambahkan setiap simpul tetangga dari simpul tersebut dan tandai sebagai simpul yang dikunjungi.
- Jika mencapai solusi atau antrian kosong, proses selesai.



Gambar 3. Proses Pencarian dengan Algoritma BFS
Sumber: https://en.wikipedia.org/wiki/Breadth-first_search

Keuntungan utama dari BFS adalah jaminan ditemukannya solusi terpendek yang paling dekat dengan akar. Namun, proses BFS memerlukan ruang memori yang besar karena proses BFS perlu melacak semua simpul yang akan dikunjungi sebelum pindah ke level berikutnya[2].

Kedua algoritma ini memiliki kelebihan dan kekurangan yang bergantung pada karakteristik masalah yang dihadapi. DFS memiliki keuntungan pada kebutuhan ruang memori yang kecil. DFS umumnya digunakan untuk mencari semua solusi. Sementara BFS umumnya digunakan dalam mencari solusi terpendek[3].

C. Algoritma Depth-Limited Search

Depth-Limited Search (DLS) adalah varian dari algoritma DFS yang membatasi kedalaman pencarian. Algoritma ini digunakan untuk menghindari kedalaman tak berujung pada graf yang sangat dalam dan mengontrol penggunaan memori serta waktu eksekusi[1].

Prinsip kerja DLS mirip dengan DFS yang umumnya dilakukan secara rekursi. Namun, pada DLS ada tambahan parameter yang akan membatasi kedalaman pencarian sehingga proses dapat dihentikan meskipun solusi belum ditemukan. Berikut ini adalah langkah-langkah umum proses DLS:

- Mulai dari simpul awal.
- Tandai simpu tersebut sebagai simpul yang telah dikunjungi.
- Jika kedalaman saat ini sama dengan batas kedalaman maka hentikan proses pencarian.
- Untuk setiap simpul yang terhubung lakukan proses DLS dengan menambah tingkat kedalaman sebesar satu.
- Jika mencapai simpul tujuan, proses berhenti.

DLS memiliki kelebihan yang mana dapat menghindari proses pencarian mendalam yang tidak diperlukan. DLS juga memiliki kelebihan seperti yang DFS miliki, yaitu penggunaan ruang memori yang rendah[2].

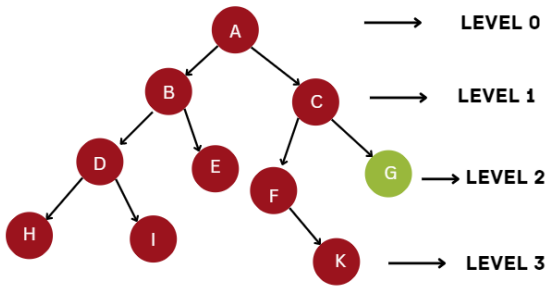
Namun, DLS tidak dapat menjamin solusi ditemukan karena solusi dapat berada pada kedalaman tertentu yang tidak terjangkau oleh batas kedalaman saat ini. Selain itu, DLS juga tidak menjamin ditemukannya solusi yang optimal[1].

D. Algoritma Iterative Deepening Search

Iterative Deepening Search (IDS) adalah algoritma yang menggabungkan kekuatan dari BFS dan DFS untuk menemukan solusi optimal dengan penggunaan memori yang efisien. IDS beroperasi dengan melakukan iterasi terdapat proses DLS yang akan melakukan penambahan kedalaman untuk setiap iterasi jika solusi belum ditemukan[1].

Prinsip kerja IDS adalah penambahan kedalaman untuk setiap iterasi, yang dimulai dari nol dan bertambah satu pada setiap tahap. Berikut ini adalah langkah-langkah umum dari IDS:

- Tetapkan batas kedalaman awal sebagai nol.
- Lakukan proses DLS dengan kedalaman saat ini.
- Jika ditemukan solusi maka hentikan proses pencarian.
- Jika tidak, lakukan proses 2 dan 3 dengan menambah kedalaman sebesar satu.



Gambar 4. Proses Pencarian dengan Algoritma IDS
Sumber: https://www.naukri.com/

IDS menjamin solusi optimal karena setiap iterasi akan melakukan pencarian menyeluruh sebelum melakukan pencarian dengan kedalaman yang lebih. Seperti halnya DLS dan DFS, IDS memiliki kelebihan dalam efisiensi penggunaan

ruang memori. Meskipun IDS melakukan pencarian berulang, algoritma ini memiliki kompleksitas waktu keseluruhan yang masih dapat diterima.

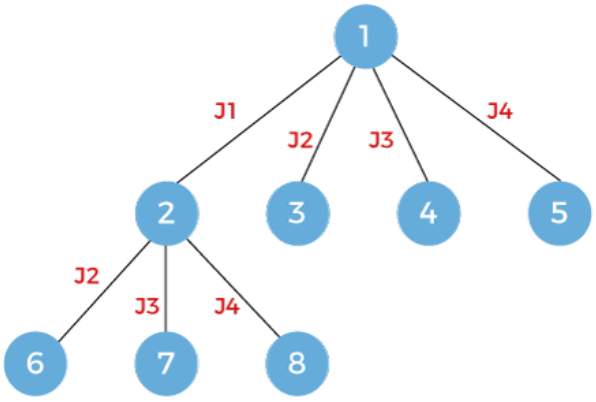
IDS memiliki kelemahan karena prosesnya yang dilakukan secara berulang sehingga terjadi pemborosan sumber daya. Namun, dalam praktiknya *overhead* ini relatif kecil dibandingkan keunggulan dalam optimalisasi penggunaan memori[3]. Selain itu, Performa IDS dapat menurun karena dengan kedalaman pencarian dari graf yang sangat dalam dapat meningkatkan waktu pencarian secara eksponensial.

E. Algoritma Branch and Bound

Branch and Bound adalah salah satu algoritma pencarian yang digunakan untuk menyelesaikan berbagai masalah optimasi, termasuk pencarian solusi optimal dari permainan 8-Puzzle. Algoritma ini melakukan pencarian dengan cara membagi ruang pencarian ke dalam sub masalah yang dipecah (*branching)* dan menggunakan batasan yang berguna dalam memangkas cabang atau simpul yang tidak menjanjikan solusi *(bounding)*[7].

Algoritma ini menggabungkan konsep pencarian dengan metode eksplorasi yang diikuti dengan pemangkasan simpul hasil ekspansi yang tidak menjanjikan menuju solusi yang dicari. Berikut ini adalah langkah-langkah umum dari algoritma Branch and Bound:

- *Branching*: membagi masalah ke dalam sub masalah yang dihasilkan dengan melakukan ekspansi dari masalah yang dievaluasi saat ini. Setiap sub masalah ini diwakilkan oleh simpul pada pohon atau graf pencarian.
- *Bounding*: menghitung batas dari solusi terbaik yang dimiliki saat ini. Jika ada simpul yang melanggar batasan dari solusi saat ini, maka simpul tersebut dibuang (dipangkas).
- *Selection*: Memilih simpul yang memiliki nilai terkecil (kasus minimasi) atau terbesar (kasus maksimasi) untuk dievaluasi.
- *Termination*: proses berhenti jika semua simpul telah selesai dievaluasi atau dipangkas.



Gambar 5. Proses Pencarian dengan Algoritma Branch and Bound
Sumber: https://www.javatpoint.com/branch-and-bound

Secara umum, algoritma ini menjamin tercapainya solusi optimum dengan melakukan pemangkasan terhadap simpul yang tidak menjanjikan (melanggar batasan). Pemangkasan terhadap simpul yang tidak menjanjikan dengan signifikan meningkatkan efisiensi karena jumlah simpul yang perlu diperiksa berkurang sehingga proses pencarian menjadi lebih cepat[8].

Kelemahan algoritma ini adalah kebergantungannya terhadap fungsi heuristik (*bounding*) yang digunakan. Fungsi heuristik yang tidak sesuai dapat menyebabkan simpul yang seharusnya dipangkas, terevaluasi sehingga mengurangi efisiensi pencarian[7].

III. Penerapan Algoritma IDS dan Branch and Bound dalam Penyelesaian Permainan 8-Puzzle

A. Pemetaan Permainan 8-Puzzle pada Domain Algoritma IDS dan Branch and Bound

Penyelesaian permainan 8-Puzzle dapat dimodelkan sebagai pohon pencarian, di mana setiap simpul mewakili konfigurasi papan permainan, dan setiap sisi mewakili gerakan pergeseran ubin bernomor ke ubin kosong.

Setiap elemen dari permainan ini akan dipetakan pada domain di bawah ini:

- Simpul: konfigurasi papan permainan.
- Sisi: pergeseran ubin yang mungkin dilakukan seperti kiri, kanan, atas, atau bawah.
- Simpul ekspan: semua simpul hasil pergeseran ubin kosong yang mungkin dari simpul yang sedang dievaluasi.
- Simpul awal: *state* awal yang akan dijadikan simpul akar.
- Simpul akhir: *state* tujuan.
- Setiap ubin akan direpresentasikan sebagai elemen di dalam sebuah matriks yang dinomori mulai dari 1-8. Khusus ubin kosong akan diberikan nomor 9 untuk memudahkan perbandingan.

Pemetaan di atas akan diterapkan pada kedua jenis algoritma. Tetapi, pada implementasi algoritma Branch and Bound akan ditambahkan fungsi heuristik yang akan digunakan sebagai *bounding function*.

Nilai heuristik (*f(n)*) yang digunakan pada eksperimen ini adalah nilai dari total langkah dari simpul akar ke simpul ke-*n* atau jarak dari simpul akar ke simpul ke-*n* (*f(n)*) dan jumlah ubin yang tidak berada pada posisinya (*g(n)*).

$$f(n) = g(n) + h(n)$$

Nilai heuristik ini kemudian akan digunakan sebagai batas atas (*upper bound*) karena tujuan penelitian ini adalah menemukan solusi dengan langkah terpendek. Nilai ini digunakan pada *bounding function* untuk memangkas simpul yang tidak optimal atau tidak menjanjikan serta digunakan untuk menentukan simpul mana yang akan dievaluasi terlebih dahulu.

B. Implementasi

Berikut ini adalah langkah-langkah yang digunakan pada proses pencarian solusi 8-Puzzle menggunakan algoritma IDS:

- Tetapkan batas kedalaman pencarian dengan nilai awal nol.
- Lakukan proses DLS dengan kedalaman yang ditetapkan.
- Jika solusi ditemukan, proses pencarian dihentikan.
- Jika solusi tidak ditemukan, tambah kedalaman sebesar satu dan lakukan kembali proses 2 dan 3.

Berikut ini adalah *pseudocode* untuk implementasi program pencarian dengan algoritma IDS.

```
procedure searchIDS(Node puzzle):
{ melakukan proses pencarian dengan algoritma
IDS, menyimpan nilai solusi pada variable global
}
{ indikator bahwa solusi telah ditemukan atau
belum diasumsikan sudah dideklarasikan }
DECLARATION
   currentDepth : integer
ALGORITHM
   currentDepth <- 0
   while not found do
      DLS(puzzle, currentDepth, 0)
      currentDepth <- currentDepth + 1
   endwhile

procedure DLS(Node puzzle, limitDepth, depth):
{ melakukan proses DLS secara rekursif dengan
batas kedalaman limitDepth }
DECLARATION
ALGORITHM
   if found or depth ≥ limitDepth then
      return
   endif
   counter <- counter + 1 { menghitung jumlah}
                          { node yang dikunjungi}
   if puzzle.isSolusi() then
      found <- true
      solusi <- puzzle
      return
   endif
   for child in simpul anak yang mungkin diekspan
do
      DLS(child, limitDepth, depth+1)
   endfor
```

Berikut ini adalah langkah-langkah yang digunakan pada proses pencarian solusi 8-Puzzle menggunakan algoritma Branch and Bound:

- Tambahkan simpul awal ke dalam antrian.
- Jika antrian sudah kosong proses berhenti.
- Jika antrian belum kosong, keluarkan satu simpul yang disimpan di dalam antrian untuk dilakukan evaluasi.
- Jika simpul saat ini merupakan solusi dengan nilai heuristik yang lebih kecil (optimal) ganti dengan nilai yang baru.
- Untuk semua simpul di dalam antrian yang memiliki nilai *cost* yang lebih besar dari solusi yang ditemukan hingga saat ini dipangkas.
- Jika simpul yang saat ini dievaluasi bukan solusi, untuk setiap simpul yang dapat diekspan, ditambahkan ke dalam antrian.
- Kembali pada proses 2.

Berikut ini adalah *pseudocode* untuk implementasi program pencarian dengan algoritma Branch and Bound.

```
procedure searchIDS(Node puzzle):
{ melakukan proses pencarian dengan algoritma
BnB, menyimpan nilai solusi pada variable global
}
{ indikator bahwa solusi telah ditemukan atau
belum diasumsikan sudah dideklarasikan }
{ variable global visited dan queue diasumsikan
sudah dideklarasikan }
DECLARATION
   currentPuzzle : Node
ALGORITHM
   queue.add(puzzle)
   while not found and not queue.isEmpty() do
      currentPuzzle <- queue.poll()
      visited.add(currentPuzzle)
                { ditambahkan sebagai node yang }
                { telah dikunjungi }
      if currentPuzzle.isSolusi() then
          if solution = null then
              if solution.length >
currentPuzzle.length then
                 solution <- currentPuzzle
             endif
          endif
          else
              solution <- currentPuzzle
          endif
          queue.pangkas(solution.length)
               { melakukan pemangkasan untuk }
               { membuang simpul yang tidak }
               { menjanjikan: }
               { f(n) > solution.length }
      endif
      for child in simpul anak yang mungkin
diekspan do
          if not visited.contains(child) then
              queue.add(child)
              { menambah child yang belum pernah }
              { dikunjungi untuk menghindari }
              { pemeriksaan berulang }
          endif
      endfor
   endwhile
```

C. Uji dan Analisis

Pengujian dilakukan dengan menggunakan algoritma di atas terhadap beberapa uji coba yang didapat dari permainan 8-Puzzle yang dapat diakses pada pranala ini.

Pada percobaan ini setiap matriks papan permainan akan ditransformasikan menjadi *array* satu dimensi.

TABLE I. UJI COBA DENGAN ALGORITMA IDS

Papan Permainan	Panjang Lintasan Solusi	Waktu Eksekusi (s)	Banyak Simpul Dikunjungi
[1,7,5,4,2,6,8,3,9]	18	15.24	138937849
[3,6,2,1,4,7,5,8,9]	18	19.20	180294657
[6,2,8,3,4,5,1,7,9]	20	81.93	789324272
[3,5,6,4,1,2,7,8,9]	16	3.21	28528730

Sumber: dokumen penulis

TABLE II. UJI COBA DENGAN ALGORITMA BRANCH AND BOUND

Papan Permainan	Panjang Lintasan Solusi	Waktu Eksekusi (ms)	Banyak Simpul Dikunjungi
[1,7,5,4,2,6,8,3,9]	18	4	1176
[3,6,2,1,4,7,5,8,9]	18	4	967
[6,2,8,3,4,5,1,7,9]	20	7	2532
[3,5,6,4,1,2,7,8,9]	16	3	537

Sumber: dokumen penulis

```
Memory used before: 506.3671875 KB
Algoritma IDS
Waktu pencarian: 15246ms
Banyak node dikunjungi: 138937849
Solusi - dengan panjang lintasan 18:
LEFT UP LEFT UP RIGHT DOWN RIGHT UP LEFT LEFT DOWN
DOWN RIGHT UP LEFT DOWN RIGHT RIGHT
Memory used after: 2682.265625 KB
Memory used by task: 2175.8984375 KB
```

Gambar 6. Hasil Percobaan Konfigurasi Papan Pertama dengan Algoritma IDS
Sumber: dokumen penulis

```
Memory used before: 506.3671875 KB
Algoritma Branch and Bound
Waktu pencarian: 4ms
Banyak node dikunjungi: 1176
Solusi - dengan panjang lintasan 18:
LEFT UP LEFT UP RIGHT DOWN RIGHT UP LEFT LEFT DOWN
DOWN RIGHT UP LEFT DOWN RIGHT RIGHT
Memory used after: 3007.9375 KB
Memory used by task: 2501.5703125 KB
```

Gambar 7. Hasil Percobaan Konfigurasi Papan Pertama dengan Algoritma Branch and Bound
Sumber: dokumen penulis

Berdasarkan empat uji coba di atas, waktu eksekusi yang diperlukan algoritma Branch and Bound jauh lebih cepat daripada algoritma IDS. Hal ini berbanding lurus dengan jumlah simpul yang dikunjungi dari proses di atas.

Jumlah simpul yang dikunjungi algoritma IDS sangat banyak, karena terjadi pengulangan evaluasi simpul. Hal ini terjadi karena setiap kali kedalaman ditambah, proses diulangi dari awal. Proses pengulangan ini yang kemudian menyebabkan algoritma IDS kurang efisien dalam aspek waktu eksekusi.

Solusi yang diberikan kedua algoritma adalah solusi optimal karena hasil kedua proses sama. Solusi yang dihasilkan algoritma IDS menjadi pembanding keoptimalan solusi yang dihasilkan algoritma Branch and Bound. Dalam percobaan ini, seluruh solusi yang dihasilkan algoritma Branch and Bound adalah solusi optimal.

IV. KESIMPULAN

Penelitian ini mengeksplorasi dan mengevaluasi penerapan dua algoritma pencarian, yaitu Iterative Deepening Search (IDS) dan Branch and Bound dalam penyelesaian permainan 8-Puzzle. Hasil evaluasi menunjukkan bahwa IDS memastikan solusi optimal dengan pencarian iterative dengan kedalaman yang bertambah secara bertahap. Namun, waktu eksekusi bertumbuh dengan pesat karena adanya proses pencarian berulang untuk menambah kedalaman.

Algoritma Branch and Bound menggunakan teknik pemangkasan simpul yang tidak menjanjikan berdasarkan fungsi heuristik yang digunakan. Hal ini menyebabkan proses pencarian menjadi jauh lebih cepat daripada algoritma IDS karena mengurangi jumlah simpul yang perlu dievaluasi. Namun, fungsi heuristik yang tidak sesuai dapat mengurangi efektivitas dan efisiensi dari proses pencarian. Pencarian dengan Algoritma Branch and Bound lebih cepat dibandingkan dengan IDS pada kasus permainan 8-Puzzle.

V. UCAPAN TERIMA KASIH

Puji dan syukur penulis panjatkan kepada Tuhan Yang Maha Esa karena atas rahmat dan berkat-Nya, penulis dapat menyelesaikan makalah ini dengan baik. Penulis juga mengucapkan terima kasih kepada kedua orang tua penulis karena sudah mendukung penulis selama masa penyusunan makalah ini. Penulis mengucapkan terima kasih kepada Bapak Dr. Ir. Rinaldi Munir, M.T., Bapak Ir. Rila Mandala, M.Eng., Ph.D., dan Ibu Dr. Nur Ulfa Maulidevi S.T., M.Sc selaku dosen mata kuliah IF2120 Matematika Diskrit yang telah membimbing penulis dan memberikan banyak ilmu yang bermanfaat selama perkuliahan.

REFERENSI

[1] K. R. E., "Depth-first iterative-deepening: An optimal admissible tree search," dalam *Artificial Intelligence*, vol. 27, 1985, hlm. 97–109.
[2] R. S. dan N. P., *Artificial Intelligence: A Modern Approach*, 3 ed. Upper Saddle River, NJ, USA: Prentice Hall, 2010.
[3] S. R. dan W. K., *Algorithms*, 4 ed. Addison-Wesley, 2011.
[4] O. M. J. dan R. A., *A Course in Game Theory*. MIT Press, 1994.
[5] J. von Neumann dan M. O., *Theory of Games and Economic Behavior*. Princeton University Press, 1944.
[6] F. D. dan T. J., *Game Theory*. MIT Press, 1991.
[7] L. E. L. dan W. D. E., "Branch-and-bound methods: A survey," dalam *Operations Research*, vol. 14, 1966, hlm. 699–719.
[8] W. H. P., *Model Building in Mathematical Programming*, 5 ed. John Wiley & Sons, 2013.

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

Suthasoma Mahardhika Munthe 135220908