Algoritma Branch and Bound untuk Memecahkan 15-Puzzle dengan Parallel Computing

Penggunaan Algoritma Branch and Bound dengan Parallel Computing untuk Menghitung Fungsi Ongkos

Muhammad Daffa Dinaya (*13518141*)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
muhammaddaffaddinaya@gmail.com 13518141@std.stei.itb.ac.id

***Abstrak*— Algoritma Branch and Bound adalah algoritma yang umum dipelajari dalam dunia *computer science* dimana implementasinya banyak sekali diterapkan dalam berbagai hal. Beberaoa dari contoh implementasinya banyak ditemui dalam persoalan klasik seperti Travelling Salesman Person, Knapsack 0/1 Problem dan 15-Puzzle Problem. 15-Puzzle merupakan permainan klasik dimana permainan tersebut bertujuan menyusun petak ubin dengan urutan yang sesuai. Implementasi algoritma pemecahan masalah tersebut dalam Branch and Bound memilki bagian yang dapat bekerja secara terpisah sehingga memungkinkan untuk dijalankan dengan Parallel Computing.**

Kata Kunci—15-puzzle, Parallel Computing, branch and bound, synchronization, locking

I. PENDAHULUAN

15-Puzzle merupakan permainan dimana terdapat persegi berukuran 4x4 satuan ubin.. Didalam persegi tersebut akan terdapat 15 petak yang terisi ubin yang tersusun secara acak. Pemain harus menyelesaikan puzzle tersebut sehingga seluruh ubin pada puzzle terurut menaik dari sebelah kiri-atas ke kanan bawah. Pemain menata pola tersebut dengan memanfaatkan petak kosong yang digunakan untuk mengubah posisi ubin lainnya.

Pada permainan 15-Puzzle banyak pendekatan yang dilakukan seperti algoritma brute force dimana algoritma tersebut akan mencoba seluruh kemungkinan pada perpindahan yang dilakukan menggunakan petak kosong. Hal ini bukannlah merupakan pendekatan yang baik karena akan melakukan banyak sekali komputasi dan terlalu banyak menggunakan resource.

Pada pendekatan branch and bound akan digunakan pendekatan yang lebih mangkus, yaitu dengan mempertimbangkan ongkos yang telah digunakan untuk mencapai suatu kondisi dan ongkos perkiraan untuk mencapai kondisi tujuan. Branch and bound memiliki pendekatan yang mirip dengan breadth first search dalam ekspansi simpul, tetapi breadth first search harus mengevaluasi simpul yang bahkan tidak mungkin lagi mencapai tujuan. Sedangkan branch and bound dapat melakukan perkiraan melalui ongkos yang dihitung pada simpul

Pada makalah ini sebenarnya merupakan pengembangan dari algoritma branch and bound. Dimana bagian yang dikembangkan adalah bagian yang memiliki pemrosesan secara terpisah menggunakan thread tersendiri. Makalah ini juga dapat sebagai wawasan dalam penggunaan thread dalam algoritma graf lainnya seperti A*, Uniform Cost Search, Deep First Search, Greedy Best First Search, dll. Sebab di dalam algoritma graf sering muncul pola evaluasi ekspan simpul yang dapat diproses secara terpisah.

Makalah ini juga akan membahas performa yang dilakukan ketika melakukan Parallel Computing. Apakah performa akan meningkat karena penggunaan Parallel Computing atau terdapat kasus lain yang menyebabkan pemrosesan menggunakan Parallel Computing merugikan karena perlu menggunakan resource tambahan.

II. LANDASAN TEORI

A. *Branch and Bound*

Branch and Bound adalah algoritma yang sering digunakan untuk melakukan optimasi yang digunakan untuk menyelesaikan permasalahan optimasi pada persoalan kombinatorial, yaitu persoalan mencari objek yang memenuhi persyaratan tertentu. Misalnya pada 15-Puzzle, kita perlu mencari urutan pengerjaan puzzle (objek) yang memenuhi tujuan akhir dari puzzle tersebut, yaitu mengurutkan puzzle tersebut (persyaratan) dengan kondisi ekspansi simpul se-optimal mungkin. Karakteristik permasalahan yang diselesaikan menggunakan Branch and Bound biasanya berupa persoalan yang memiliki kompleksitas eksponensial dan diperlukan eksplorasi secara menyeluruh terhadap kemungkinan permutasi yang ada jika mengalami kasus paling buruk. Namun biasanya Branch and Bound dapat menyelesaikan permasalahan tersebut relatif cepat karena terdapat pemilihan ekspansi simpul melalui nilai ongkos.

Branch and Bound merupakan algoritma yang berdasarkan pada teori algoritma Breadth First Search. Namun seperti yang sudah dijelaskan sebelumnya, algortima ini memiliki evaluasi ongkos yang digunakan agar mempercepat pencarian ke simpul tujuan dengan meminimalkan ekspansi simpul yang dilakukan. Simpul yang diekspansi bukanlah seperti pada algoritma BFS yang akan melakukan ekspansi simpul melalui antrian FIFO (First In First Out), tetapi antrian tersebut sudah diurutkan dengan berdasarkan nilai ongkos simpul yang paling kecil.

Cara kerja dari algoritma branch and bound secara umum adalah sebagai berikut, misal terdapat sebuah antrian dan sebuah simpul awal yang akan dilakukan pencarian ke sebuah simpul solusi maka

1. Masukkan simpul akar ke dalam antrian Q. Jika simpul akar adalah simpul solusi (goal simpul), maka solusi telah ditemukan. Stop.
2. Jika Q kosong, tidak ada solusi. Stop.
3. Jika Q tidak kosong, pilih dari antrian Q simpul i yang mempunyai nilai 'cost' ĉ(i) paling kecil. Jika terdapat beberapa simpul i yang memenuhi, pilih satu secara sembarang.
4. Jika simpul i adalah simpul solusi, berarti solusi sudah ditemukan, stop. Jika simpul i bukan simpul solusi, maka bangkitkan semua anak-anaknya. Jika i tidak mempunyai anak, kembali ke langkah 2.
5. Untuk setiap anak j dari simpul i, hitung ĉ(j), dan masukkan semua anak-anak tersebut ke dalam Q.
6. Kembali ke langkah 2.

B. *Implementasi Fungsi Ongkos*

Pada pembahasan sebelumnya sudah disebutkan mengenai algoritma dari Branch and Bound. Di dalam algoritma tersebut sudah ditunjukan bahwa algoritma tersebut memerlukan perhitungan nilai ongkos yang akan digunakan untuk mengatur urutan dalam antrian. Kemudian bagaimana perkiraan ongkos yang akan digunakan dalam pengukuran nilai ongkos tersebut? Berikut adalah implementasinya.

Untuk implementasi dari nilai cost adalah sebagai berikut

$$\hat{c}(i) = \hat{f}(i) + \hat{g}(i)$$

$\hat{c}(i)$ = ongkos untuk simpul i

$\hat{f}(i)$ = ongkos mencapai simpul i dari akar

$\hat{g}(i)$ = ongkos mencapai simpul tujuan dari simpul i.

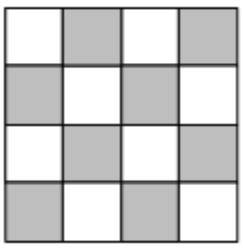
Pada implementasi algoritma pencarian solusi persoalan 15-puzzle maka f(i) merupakan kedalaman simpul i dari simpul akar dan g(i) adalah nilai taksiran fungsi Kurang(i).

Fungsi Kurang(i) merupakan fungsi yang bertujuan menaksir jarak simpul i ke simpul tujuan. Definisi Fungsi Kurang adalah sebagai berikut Kurang(i) = banyaknya ubin bernomorj sedemikian sehingga j < i dan Posisi(j) > Posisi(i). Posisi(i) = posisi ubin bernomor i pada susunan yang diperiksa.

Fungsi Kurang(i) juga akan menunjukan apakah suatu persoalan dari 15-puzzle akan memiliki sebuah solusi. Solusi tersebut dapat dicari dengan cara sebagai berikut

$$\sum_{i=1}^{16} KURANG(i) + X$$

Dengan nilai X adalah 1 apabila posisi ubin kosong menempati daerah arsir di bawah dan 0 apabila sebaliknya



Gambar II.1 : Posisi ubin kosong

C. *Parallel Computing*

Sistem secara parallel sering dilakukan apabila sistem digunakan untuk menyelesaikan permasalahan besar dan kompleks. Misal dalam penghitungan banyak bilangan dari suatu himpunan bilangan, jika melakukan penghitungan dengan cara menghitung satu-persatu bilangan secara menyeluruh akan menyebabkan sebuah bagian sistem bekerja terlalu keras dan akan memakan waktu yang cukup lama. Namun dengan penghitungan dengan sistem parallel sistem dapat membagi tugas untuk diproses secara independen.

Dengan pembagian tugas tersebut, sistem akan dapat melakukan penghitungan secara sekaligus dan sistem tidak akan terlalu terbebani di satu bagian saja. Dari beberapa keuntungan sistem parallel inilah yang akan dijadikan sebagai dasar pembuatan program implementasi algoritma Branch and Bound secara parallel computing. Program ini akan menjalankan beberapa proses dengan masing-masing dari proses tersebut melakukan penghitungan nilai ongkos dari kemungkinan pergerakan ubin kosong. Proses yang berjalan akan bekerja sama menghasilkan nilai ongkos secara keseluruhan dari kemungkinan pergerakan dan akan diperoleh solusi final dari nilai ongkos paling minimum.

Parallel Computing adalah pemrosesan dengan program dimana di dalam program terdapat berbagai instruksi yang saling bekerja sama untuk menyelesaikan permasalahan (Peter, 2011). Dalam parallel computing, instruksi akan dijalankan secara bersamaan dan akan menjalankan instruksi yang dibagikan namun tetap adanya kooperasi dari program utama. Jadi progress dari masing-masing instruksi tetap tidak bisa berjalan secara sendiri-sendiri dan masih bergantung dengan instruksi lainnya

Dalam parallel computing terdapat istilah yang dikenal dengan threading dimana program dapat menjalankan suatu perintah dengan tugas tertentu secara independen. Hal ini dapat berguna dalam proses yang memiliki permasalahan yang mirip sehingga dapat diselesaikan secara sekaligus menggunakan threading.

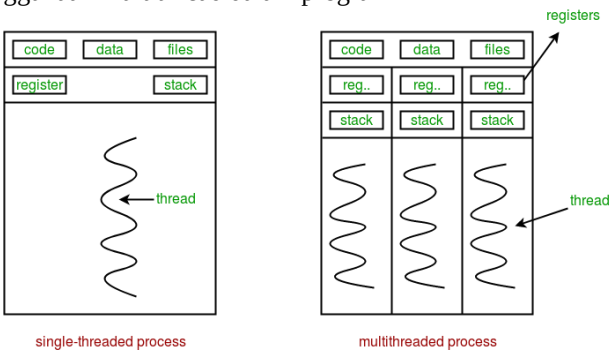
D. Multithreading and Sinkronisasi

Pada proses parallel computing terdapat bagian proses yang sangat penting yaitu pada bagian melakukan multithreading dan sinkronisasi.

Penggunaan Multithreading pada dasarnya memungkinkan menjalankan beberapa instruksi secara langsung dan bersamaan. Hal ini dilakukan dengan membuat thread yang akan berjalan secara independen. Ketika sebuah thread dijalankan maka secara otomatis program akan menjalan intruksi dari thread dan sekaligus melanjutkan instruksi dari program utama.

Dalam pembuatan thread, antara thread dengan program utama tetap akan hubungan. Hubungan tersebut terdapat pada pembagian data global yang sama. Namun selama ada data yang diambil untuk diproses di dalam area local oleh thread, tidak dapat diakses oleh pogram utama ataupun thread lain.

Untuk mempermudah pemahaman, berikut ilustrasi dari penggunaan multithread dalam program



Dari ilustrasi di atas dapat dilihat bahwa masing-masing thread perlu untuk mengakses data global yang sama untuk memulai program di masing-masing thread. Di dalam pengaksesan data yang sama, dapat terjadi perilaku program yang tidak wajar apabila data tersebut diakses dengan waktu yang bersamaan. Hal ini disebabkan karena akses yang bersamaan akan menimbulkan resource terkunci oleh akses luar atau apabila ada perubahan dan sudah tertulis ulang di resource tersebut, kondisi awal dari data sudah diperbarui dan akan menimbulkan hasil yang berbeda.

Sinkronisasi adalah bagian di saat menunggu semua thread selesai melaksanakan tugas dan menyatukan solusinya. Hal ini juga perlu diperhatikan apabila thread harus mengakses data yang sama di akhir pemrosesan karena dapat terjadi kasus yang sama seperti di atas.

E. Deadlock

Dalam proses threading terdapat sebuah constraint utama yang perlu diperhatikan, yaitu deadlock. Deadlock adalah sekumpulan proses yang di-blok, dimana masing-masing proses membawa resource dan menunggu mendapatkan resource yang dibawa proses lain dalam kumpulan resource yang dibawa proses lain dalam kumpulan tersebut

Deadlock dapat terjadi jika terdapat 4 kondisi yang terjadi secara simultan

1. Mutual exclusion: hanya satu proses pada satu waktu yang dapat menggunakan satu resource.
2. Hold and wait: sebuah proses membawa sedikitnya satu resource sedang menunggu mendapatkan resource tambahan yang dibawa oleh proses-proses lain
3. No preemption: sebuah resource dapat dibebaskan hanya oleh proses yang membawanya, setelah proses menyelesaikan task/pekerjaan
4. Circular wait: terdapat sekumpulan {P0, P1, …, P0} dari proses yang menunggu dimana P0 menunggu resource yang dibawa oleh P1, P1 menunggu resource yang dibawa oleh P2, …, Pn–1 menunggu resource yang dibawa oleh Pn, dan P0 menunggu resource yang dibawa oleh P0.

Untuk menangani kasus Deadlock dapat digunakan dua buah cara yaitu Deadlock Prevention dan Deadlock Avoidance. Deadlock Prevention adalah langkah yang digunakan untuk memastikan tidak pernah terjadi syarat yang menjadi kondisi deadlock di bagian yang rawan terjadi deadlock, seperti pada akses data yang sama. Deadlock Avoidance adalah langkah yang diambil untuk memastikan tidak ada proses yang melewati daerah lawan, semisal menyalin data terlebih dahulu ke beberapa variabel berbeda dan masing-masing variabel tersebut digunakan oleh thread yang terpisah.

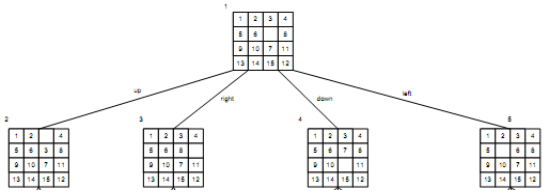
III. LANGKAH PENYELESAIAN BNB PARALLEL COMPUTING

Alasan mengapa threading dapat digunakan dalam persoalan 15-Puzzle Branch and Bound sebab terdapat karakteristik yang sama pada proses setiap perubahan gerakan ubin kosong. Karakteristik tersebut dapat dilihat dari simpul yang dihasilkan ketika ubin kosong bergerak yaitu

1. Simpul parent akan menurunkan matriks yang mewakili posisi dari setiap ubin dalam puzzle dengan kondisi yang sama dengan simpul parent sebanyak kemungkinan pergerakan ubin kosong pada puzzle, yaitu sejumlah 4 buah (ke atas, ke bawah, ke kanan, dan ke kiri). Namun perlu diingat bahwa apabila sudah melakukan pergerakan dari asal yang sama, maka kemungkinan pergerakan tersebut tidak boleh menghasilkan kondisi yang sama seperti sebelumnya. Contoh ketika simpul parent saat ini sebelumnya melakukan pergerakan ubin kosong ke bawah, maka pergerakan ubin kosong yang kembali membawa ke kondisi awal, yaitu pergerakan ke atas tidak boleh diturunkan ke matriks.

2. Dari masing-masing kondisi simpul yang sudah diturunkan akan menjadi kondisi awal dari simpul child yang kemudian akan digerakan berdasarkan kemungkinan pergerakan ubin kosong yang diperbolehkan.

3. Ketika ubin kosong telah digerakan. Masing-masing dari matriks tersebut akan melakukan komputasi tersendiri untuk menghitung nilai ongkos menggunakan fungsi ongkos yang sudah ditentukan. Di dalam perhitungan ini sebuah matriks **tidak bergantung** pada kondisi maupun perhitungan dari matriks lainnya. Ilustrasinya sebagai berikut



4. Ketika perpindahan ubin kosong sudah selesai dilakukan maka memasukan semua matriks ke dalam antrian dan diurutkan berdasarkan nilai ongkosnya

Bisa diperhatikan bahwa pada langkah 2 dan 3 memiliki karakteristik yang sama untuk setiap matriks dan bersifat independen terhadap matriks lainnya sehingga dapat dilakukan multithreading pada langkah tersebut. Kemudian di langkah 4 secara pola mirip dengan proses sinkronisasi karena menyatukan solusi ke dalam antrian

A. Pseudocode Main Program

Berikut adalah pseudo-code dari main program. Dimana main program merupakan kontrol dari algortima dasar branch and bound, sehingga pseudo code-nya akan mirip dengan algoritma Branch and Bound yang sudah ditunjukan sebelumnya

```
begin
        activeset :={∅}
        while activeset is not empty do
                choose a branching node, node k ∈
        activeset;
                remove node k from activeset;
                generate the children of node k, child
        i, i=1,. . . ,nk, and corresponding optimistic
        bounds obi ;
                for i=1 to nk do
                        if child is a complete solution
                then break
                        else add child i to activeset
end
```

B. Pseudocode Fungsi Kurang(X)

Berikut adalah adalah pseudo-code dari fungsi yang akan mengevaluasi nilai kurang dari suatu matriks

```
less(matrix):
begin
result := 0
      for i := 0...3 do
         for j := 0...3 do
            result_item := 0
            if matrix[i][j] is not empty tile then
// empty tile is marked as 16 (for ease of
implementation)
               for k := 0...3 do
                  if k == i then
                     for l := 0...3 do
                        if matrix [i][j] > matrix [k][l] then
                           result := result + 1
                           result_item := result_item + 1
                  else then
                     for l := 0...3 do
                        if matrix [i][j] > matrix [k][l] then
                           result := result + 1
                           result_item := result_item + 1
```

C. Pseudocode Inherit Child

Pada bagian ini adalah perbedaan mendasar yang ada di dalam penyelesaian Branch and Bound teknik biasa dengan menggunakan multithread. Sebab pada bagian ini adalah bagian implementasi dari pola yang sudah dibahas dalam langkah penyelesaian Branch and Bound secara Parallel Computing.

```
def inherit(matrix):
begin
      resultNodeQueue :={∅}
      movementList := {UP, LEFT, DOWN, RIGHT}
      t :={∅}
      for i := 0...3 do
         if matrix can move to direction movementList[i]
then
            insert clone matrix (child) to resultNodeQueue
            insert thread to move clone matrix to t
      for element in t do
         start()
      for element in t do
         synchronize()
      ➔ resultNodeQueue
end
```

Untuk pseudo-code tersebut perlu diberi sedikit tambahan, implementasi dari multithreading yang digunakan adalah menggunakan Deadlock Avoidance, sebab sejak awal sudah memisahkan matriks yang akan digunakan ke dalam sebuah list yang berisi beberapa variabel yang masing-masing akan digunakan oleh thread yang terpisah. Jadi tidak perlu menggunakan penguncian akses secara manual pada resource yang ada sebab sejak awal resource sudah dimiliki masing-masing oleh thread. Untuk bagian synchronize merupakan bagian yang menunggu hingga semua thread sudah memenuhi

tugas yang diberikan. Penggabungan solusi sebenarnya sudah terjadi di awal yaitu pada peletakan clone child matriks di resultNodeQueue. Jadi pada tahap synchronize tidak dilakukan akses pada resource yang sama.

IV. HIPOTESIS

Untuk bahan eksperimen, akan dibuat hipotesis pada pernyataan diawal yaitu

"Penggunaan Parallel Computing akan menaikan performa dari pemecahan permainan 15-Puzzle menggunakan algoritma Branch and Bound menurunkan performa"

"Penggunaan Parallel Computing akan menurunkan performa dari pemecahan permainan 15-Puzzle menggunakan algoritma Branch and Bound"

Performa akan diukur berdasarkan waktu yang diperlukan untuk memecahkan permainan 15-Puzzle.

V. PEMBUKTIAN TEORI

A. Kompleksitas

Pada algoritma Branch and Bound, kompleksitas waktu yang digunakan pada dasarnya adalah

$$O(b^m)$$

b : branching factor, merupakan kemungkinan banyaknya percabangan yang terjadi untuk simpul child. Karena mayoritas pergerakan adalah 3 kemungkinan b = 3

m : maksimum kedalaman status ruang, kasus terburuk adalah ∞

Dalam Parallel Computation perlu diingat bahwa thread perlu dibangkitkan sebanyak branch factor atau simpul child.

VI. PENGUJIAN

Berikut adalah tiga buah puzzle yang akan diujikan.

1	2	3	4
5	6	X	8
9	10	7	11
13	14	15	12

Gambar VI.1 : Puzzle 1

1	2	3	4
5	6	7	8
9	11	X	12
13	10	14	15

Gambar VI.2 : Puzzle 2

1	2	3	4
5	6	7	8
9	11	14	12
13	10	15	X

Gambar VI.3 : Puzzle 3

Pengujian akan dilakukan dengan dua buah cara, cara pertama yaitu setiap ada matriks yang melakukan penghitungan terhadap matriks child akan dijeda selama 0.5 detik dan cara kedua adalah tanpa penggunaan jeda. Masing-masing pengujian akan dilakukan sebanyak tiga kali dan diambil rata-rata waktunya. Penambahan waktu jeda digunakan untuk melihat secara lebih jelas perbedaan alur pada waktu eksekusi multithreading.

Berikut adalah tabel hasil uji dari percobaan yang sudah dijalankan

Tabel VI.1 : Pengujian tanpa jeda

No	Singlethread	Multithread
1	0,0167044	0,020311197
2	0,026224534	0,0358905
3	0,032614946	0,041107734

Tabel VI.2 : Pengujian menggunakan jeda

No	Singlethread	Multithread
1	5,556244612	2,074436982
2	8,630413612	4,118474325
3	10,63711365	5,346064727

VII. KESIMPULAN

Pada hasil uji dapat dilihat bahwa Singlethread mampu menjalankan tugas lebih cepat ketika pengujian tanpa jeda. Sedangkan Multithread mampu menjalankan lebih baik ketika terdapat jeda. Dengan demikian kedua hipotesis masih belum bisa ditolak karena terdapat faktor pembangkitan thread pada setiap simpul yang dihasilkan. Faktor tersebut dapat dilihat ketika ongkos yang dibayar untuk membangkitkan thread memperlambat ketika pengujian tanpa jeda, sedangkan saat menggunakan jeda Multithread akan lebih menguntungkan.

Dengan demikian apabila pembangkitan sebuah thread dinyatakan dalam t sekon, maka perlu diperhatikan ketika akan mendesain program sehingga dalam potongan proses yang akan dijadikan Multithread (Tm) yang dibandingkan dengan waktu pemrosesan Singlethread (Ts) menjadi

$$Ts - (n * t + Tm) > 0,$$

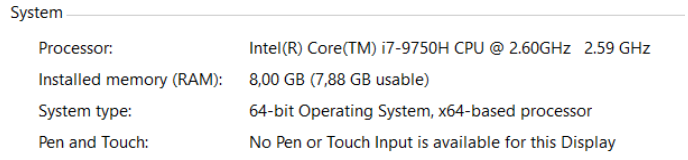
Sehingga pendesain program akan mendapatkan keuntungan apabila melakukan implementasi dengan Multithreading

VIII. SARAN

Sarannya adalah semoga ke depannya penerapan teknik Parallel Computing dapat diterapkan dengan cara yang lebih efektif dan mangkus sehingga tidak membayar ongkos pembangkitan thread yang terlalu besar. Sebab dengan mengurangi ongkos yang diperlukan dari hal tersebut dapat membantu pada pemrosesan yang memiliki waktu eksekusi yang lama. Meskipun perbandingan tingkat penurunan waktu eksekusi keci namun apabila diterapkan ke dalam proses yang memiliki waktu eksekusi yang lama maka akan sangat membantu dalam perkembangan komputasi tingkat lanjut.

IX. LAMPIRAN

Spesifikasi alat uji yang digunakan



Gambar IX.1 : Spesifikasi alat uji

VIDEO LINK AT YOUTUBE

https://youtu.be/MHPq8NfFDSM

GITHUB REPOSITORY LINK

https://github.com/mdaffad/bnbmultithread

UCAPAN TERIMA KASIH DAN PENUTUP

Puji syukur kepada Allah SWT karena atas berkat dan karunia-Nya, makalah berjudul “Algoritma Branch and Bound untuk Memecahkan 15-Puzzle dengan Parallel Computing” dapat selesai tepat waktu. Saya mengucapkan terima kasih kepada Yth. Bapak Rinaldi Munir, Ibu Nur Ulfa Maulidevi, dan Ibu Masayu Leylia Khodra atas segala bantuan dan bimbingannya selama satu semester sehingga saya mampu menyelesaikan makalah ini.

REFERENSI

[1] Munir, Rinaldi. 2018. *Slide Kuliah Dynamic Programming IF2211 Strategi Algoritma*. Bandung: Institut Teknologi Bandung. Diakses pada 1 Mei 2020, 20.00 WIB.

[2] Silberschatz, Abraham. 2012. *Operating System Concept* (Edisi 9). New York: Wiley

[3] https://www.geeksforgeeks.org/8-puzzle-problem-using-branch-and-bound/. Diakses pada 30 April 2020, 21.00 WIB

[4] https://www.geeksforgeeks.org/multithreading-python-set-1/. Diakses pada 30 April 2020, 21.00 WIB

[5] Enric Rodr´ıguez-Carbonell. 2020. *Introduction: Combinatorial Problems*. Diakses pada pada 2 Mei 2020, 11.00 WIB

[6] Mackworth, Alan. 2013. *Branch & Bound (B&B) and Constraint Satisfaction Problems*. Diakses pada 2 Mei 2020, 11.00 WIB

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 3 Mei 2020

Muhammad Daffa Dinaya, 13518141