Penyelesaian Traveling Salesman Problem Dengan Dynamic Programming

Maulana Akmal, 1356084
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
maulana2904@gmail.com

***Abstrak*— Traveling Salesman Problem atau yang biasa disingkat dengan TSP adalah salah satu permasalahan di dalam dunia *Computer Science* dimana diberikan sebuah graf tak berarah berbobot kemudian kemudian menentukan lintasan terpendek yang dapat dilalui dengan syarat memulai di sebuah simpul kemudian kembali lagi ke simpul tersebut setelah melalui seluruh node. Salah satu metode penyelesaian masalah Traveling Salesman ini adalah dengan menggunakan Paradigma Dynamic Programming atau Program Dinamis.**

***Keywords*— Dynamic Programming, TSP, bitmask.**

I. PENDAHULUAN

Bayangkan pada suatu hari terdapat seorang salesman yang ingin menjual barangnya di sebuah negara. Sang salesman ingin agar barangnya dapat tersebar ke seluruh kota. Tentunya dia harus mengunjungi setiap kota agar dapat mencapai tujuannya. Namun tentunya sang salesman ingin menjual barang nya dengan waktu secepat-cepat nya. Waktu yang dihabiskan sang salesman untuk menjual seluruh barangnya dipengaruhi oleh jarak perjalanan yang dia tempuh. Oleh karena itu kita harus menentukan lintasan yang harus ditempuh oleh sang salesman agar jarak yang ditempuh seminimal mungkin.

Permasalahan diatas adalah sebuah permasalahan dalam dunia *Computer Science* yang disebut dengan traveling salesman. Permasalahan ini dapat dimodel dengan pendekatan graf sehingga memiliki struktur yang optimal. Permasalahan traveling salesman ini tentu saja dapat diselesaikan dengan metode bruteforce dengan cara mencari seluruh kemungkinan lintasan yang akan dilalui oleh sang salesman. Dengan metode bruteforce kompleksitas yang dihasilkan sangat buruk sehingga kita harus mencari metoda yang lebih optimal sehingga kompleksitasnay membaik. Salah satu metoda yang menghasilkan kompleksitas yang bagus pada permasalahan ini adalah Dynamic Programming.

Permasalahan Traveling Salesman ini memiliki aplikasi yang sangat luas di dunia nyata. Model permasalahan yang mulanya dimodelkan pada seorang salesman dapat diubah kepada entity lain yang membutuhkan solusi yang sama. Salah satu contohnya adalah permasalahan ketika sebuah perusahaan yang akan mengirimkan barang ke seluruh kota dengan menggunakan trek. Perusahaan tersebut tentunya ingin menempuh jarak dan waktu yang seminimal mungkin sehingga tidak terjadi pemborosan pada bensin maupun energi supir trek tersebut. Solusi dari permasalahan tentunya dengan menemukan jarak atau rute terpendek yang akan ditempuh oleh trek sama seperti permasalahan sang salesman sebelumnya.

II. DYNAMIC PROGRAMMING

Dalam dunia *Computer Science*, Dynamic Programming atau program dinamis adalah salah satu paradigma penyelesaian masalah dengan cara terlebih dulu membagi masalah tersebut menjadi masalah yang lebih kecil kemudian mencatat solusi yang diperoleh dari masalah yang lebih kecil. Pada masalah yang lebih lainnya muncul lagi maka algoritma tidak perlu menghitungnya lagi sehingga menghemat wasktu pada saat melakukan perhitungan. Proses penyimpanan hasil dari masalah yang lebih kecil disebut *memoization* atau memoisasi.

Dynamic Programming Seringkali digunakan pada permasalahan optimisasi. Algoritma akan menguji solusi dari *masalah* yang lebih kecil sebelumnya dan akan menggabungkan solusi sehingga dihasilkan solusi yang optimal. Sebagai perbandingan algoritma greedy mengambil solusi optimal pada setiap langkah yang dilaluinya akan tetapi pada saat solusi tersebut digabungkan belum tentu menghasilkan solusi yang optimal.

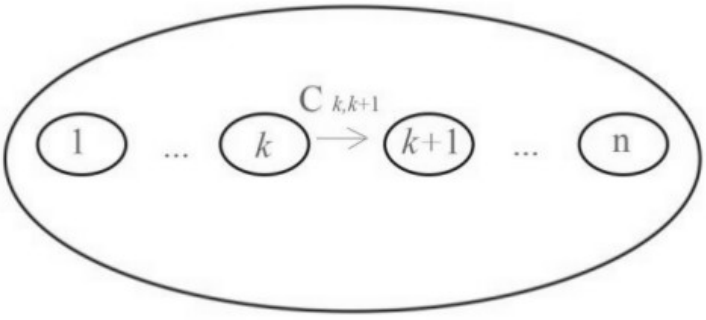
Paradigma *Dynamic Programming* sekilas tampak seperti paradigma divide and conquer karena sama-sama membagi membagi masalah menjadi masalah yang lebih kecil. Hal tersebut tidak benar, Sangat banyak perbedaan antara dynamic programming dan algoritma divide and conquer. Pada algorithma dynamic programming persoalan dibagi menjadi sub-persoalan yang overlaping. Pada dynamic programming, solusi dari setiap sub-persoalan akan dicatat dan digunakan untuk memecahkan subproblem yang sama jika muncul kembai.

Divide & Conquer	Dynamic Programming
1. Partitions a problem into independent smaller sub-problems	1. Partitions a problem into overlapping sub-problems
2. Doesn't store solutions of sub-problems. (Identical sub-problems may arise - results in the same computations are performed repeatedly.)	2. Stores solutions of sub-problems: thus avoids calculations of same quantity twice
3. Top down algorithms: which logically progresses from the initial instance down to the smallest sub-instances via intermediate sub-instances.	3. Bottom up algorithms: in which the smallest sub-problems are explicitly solved first and the results of these used to construct solutions to progressively larger sub-instances

Terdapat dua kunci utama agar algoritma dynamic programming dapat di digunakan dalam suatu persoalan. Yaitu :

- *Optimal substructur* yaitu dimana solusi yang akan diberikan kepada masalah yang akan diselesaikan dapat diperoleh dari kombinasi solusi optimal dari subproblem.
- *Overlapping subroblem* yaitu dimana solusi dari sebuah subproblem dapat digunakan untuk menyelesaikan subproblem yang sama berulang-ulang.

Pada dynamic programming, rangkaian keputusan yang optimal dibuat dengan menggunakan prinsip optimalitas yang berbunyi jika solusi total optimal maka bagian solusi sampai tahap ke-k juga optimal. Prinsip optimalitas berarti bahwa jika kita bekerja dari tahap k ke tahap k+1, kita dapat menggunakan hasil optimal dari tahap ke k tanpa harus kembali ke tahap awal. Dengan begitu ongkos pada tahap k+1 = (ongkos yang dihasilkan pada tahap k) + (ongkos dari tahap k ke tahap k+1).



Dynamic programming merupakan strategi yang sangat ampuh untuk menyelesaikan masalah-masalah yang berhubungan dengan optimalisasi, akan tetapi tidak semua permasalahan dapat diselesaikan dengan dynamic programming, kriteria permasalahan yang diselesaikan dengan dynamic programming adalaha

1. Persoalan dapat dibagi menjadi beberapa tahap atau stage, yang pada setiap tahap hanya di ambil satu keputusan.
2. Masing-masing tahap terdiri sejumlah status atau state yang berhubungan dengan tahap tersebut. Secara umum, status merupakan bermacam kemungkinan masukanyang ada pada tahap tersebut
3. Hasil dari keputusan yang diambil pada setiap tahap ditransformasikan dari status yang bersangkutan ke status selanjutnya pada tahap selanjutnya.
4. Ongkos atau cost pada suatu tahap meningkat secara teratur dengan bertambahya jumlah tahapan.
5. Ongkos pada suatu tahap bergantung pada tahap-tahap yang sudah berjalan dan ongkos pada tahap tersebut.
6. Keputusan terbaik pada suatu tahap bersifat independen terhadap keputusan yang dilakukan pada tahap sebelumnya.
7. Adanya hubungan rekursif yang mengidentifikasi keputusan terbaik untuk setiap status pada tahap k memberikan keputusan terbaik untuk status pada tahap k+1
8. Prinsip optimalitas berlaku pada persoalan tersebut.

Dynamic programming dapat di dekati dengan dua macam pendekatan yaitu.

- Dynamic programming maju (forward atau up-down)
- Dynamic programming mundur (backward atau up-down)

Misalkan x1, x2, …, xn menyatakan sebuah variabel keputusan yang harus dibuat masing-masing untuk tahap 1, 2, …, n. maka

1. Dynamic programming maju. Dynamic programming bergerak dari tahap 1, terus maju ke tahap 2, 3 dan seterusnya sampai tahap n. runtunan variabel adalah x1, x2, …, x.
2. Dynamic programming mundur. Dynamic programming bergerak dari mulai tahap n, terus mundur ke tahap n-1, n-2 dan seterusnya sampai tahap 1. runtunan variabel keputusannya adalah xn, xn-1, …, x1.

Prinsip optimalistas pada dynamic programming maju adalah

ongkos pada tahap k+1 = (ongkos yang dihasilkan pada tahap k) + (ongkos dari tahap k ke tahap k+1), dengan k=1,2, …, n-1

Prinsip optimalistas pada dynamic programming maju adalah

ongkos pada tahap k = (ongkos yang dihasilkan pada tahap k+1) + (ongkos dari tahap k ke tahap k), dengan k=1,2, …, n-1

Langkah-langkah pengembangan algoritma dynamic programming

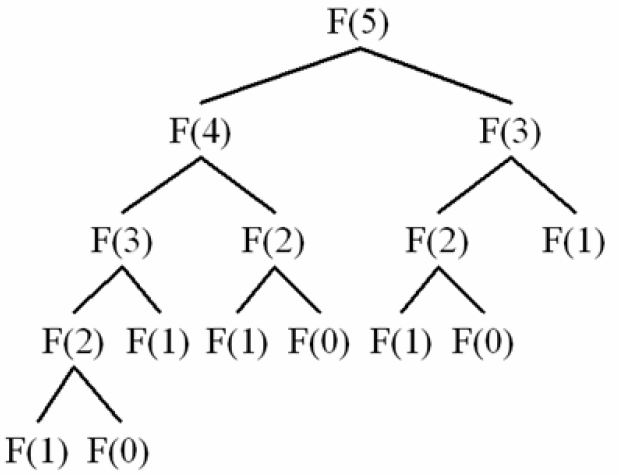
1. karakteristik struktur optimal
2. definisikan secara rekursif nilai solusi optimal
3. hitung nilai solusi optimal secara maju atau mundur
4. konstruksi solusi optimal

Untuk memperjelas pemahaman mengenai dynamic programming mari kita meninjau sebuah aplikasi dalam permasalahan deret fibonacci. Deret fibonacci adalah deret dimana suku k nya diperoleh dengan penjumlaha 2 suku sebelumnya yaitu suku k-1 dan k-2.

$$f(n) = \begin{cases} 0 & if \quad n = 0 \\ 1 & if \quad n = 1 \\ F(n-1) + F(n-2) & if \quad n > 1 \end{cases}$$

Dengan meninjau rumus yang diberikan maka permasalhan deret fibonacci ini dapat dipecahkan dengan menggunakan dynamic programming. Hal ini karena permasalahan deret fibonacci ini telah memenuhi dua kunci utama dalam dynamic programming yaitu overlapping subproblem dan optimal substructur.

Optimal substructur dalam deret fibonacci dapat dilihat melalui representasi pohon dibawah ini. Misalkan kita akan menghitung deret fibonacci ke 5 maka kita membagi permasalahan ini menjadi permasalahan yang lebih kecil yaitu denga mengghitung deret fibonacci ke 4 dan ke 3 maka dengan begitu permasalahan ini memiliki kriteria optimal substructur.



Overlapping subproblem dalam permasalahan deret fibonacci juga dapat dilihat pada representasi pohon diatas yaitu kita pada kita akan menghitung suku ke 5 maka kita juga ajan menghitung suku ke 4 dan 3. kemudian pada saat kita akan melanjutkan perhitungan suku ke empat kita juga harus menghitung suku 3 dan kedua. Pada ilustrasi diatas kita mendapatkan bahwa untuk menghitung suku ke 5 dan ke 4 kita sama-sama harus menghitung suku ketiga. Maka dengan begitu dapat dikatakan permasalahan deret fibonaaci memiliki kriteria overlapping subproblem.

Selanjutnya kita tinggal menambahkan satu aspek yang paling penting dalam algoritma dynamic programming yaitu memoisasi. Memoiasi digunakan untuk mencatat hasil dari subproblem sebelumnya agar dapat digunakan jika ditemukan subprobelem yang sama. Memoisasi biasanya dilakukan dengan menggunakan tabel. Sebagai contoh, pada ilustrasi pada permasalahan sebelumnya kita mendapati bahwa suku ketika perlu dihitung 2 kali, maka dengan menggunakan teknik memoisasi ini kita hanya perlu menghitungnya sekali dan mencatatnya dalam tabel. Selanjutnya ketika diperlukan nilai tersebut tinggal diambil dari tabel tanpa harus menghitung dari awal permasalahan yang sama. Pada permasalahan fibonacci tabel yang digunakan cukup hanya memiliki dimensi 1. akan tetapi untuk menyelesaikan pemasalahan yang lebih kompleks sering kali dibutuhkan tabel yang lebih dari 1 dimensi.

1	1	2	3	5	7

Tabel memoisasi untuk pemaaslahan fibonacci

Dengan begitu kita telah memiki semua aspek dalam dynamic programming. Kita tinggal mengaplikasikan hal tersebut pada algoritma fibonacci kita.

```
Program fibonacci;

KAMUS GLOBAL
memo : array

function F(k: integer) → integer
{ menghitung suku fibonacci ke k}

KAMUS LOKAL

ALGORITMA
      if k=0 or k=1 then
            → 1
      else if  memo[k] != -1 then
            → memo[k]
      else
            memo[k] = F(k-1)+F(k-2)
            → memo[k]
```

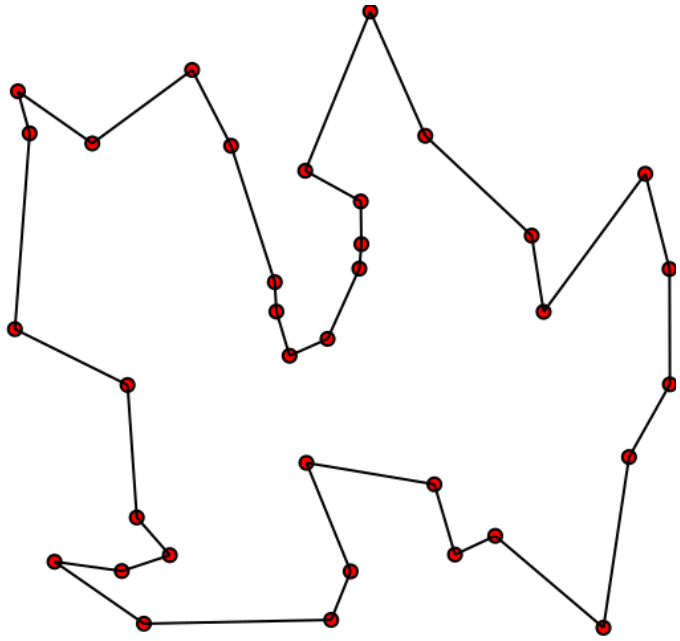
III. Traveling Salesman Problem

Traveling salesman problem atau yang sering disingkat dengan TSP adalah sebuah permasalahan dalam dunia *computer science* yang biasanya diibaratkan dengan seorang salesman yang ingin menjual barangnya ke seluruh kota yang akan dikunjunginya. Secara formal permasalahan ini dapat dinyataan dengan “Diberikan daftar kota dan jarak antar kota, temukanlah rute terpendek yang paling memungkinkan untuk dilewati agar semua kota dapat dikunjungi dan kembali lagi ke kota asal”.

Banyak permasalahan yang ada di dunia nyata ini yang merupakan generalisasi dari permasalahan ini sebagai contoh adalah permasalahan menentukan rute terbaik sebuah truk mengantarkan barang.

Permasalahan traveling salesman ini pertama kali diformulasikan pada tahun 1930 dan merupakan permasalahan yang di sering dipelajari dalam optimisasi. Permasalahan ini digunakan sebagai patokan atau benchmark dari permasalahan-

permasalahan optimisasi lainnya. Walaupun permasalahan ini termasuk permasalahan NP-Hard, banyak sekali algoritma yang sudah ditemukan untuk menyelesaikan permasalahan ini.



Ilustrasi permasalahan TSP.

TSP dalam formulasi paling dasarnya memiliki banyak aplikasi seperti pada dunia perencanaan, logistik, serta juga dalam proses pembuatan microchip. Dengan memodifikasinya sedikit permasalahan ini juga dapat diaplikasikan dalam bidang yang luas lagi seperti sekuensing DNA. TSP juga dapat diaplikasikan dalam bidang astronomi. Misalnya seorang astronom yang sedang mengami banyak benda langit. Tentunya dia harus banyak memindah-mindahkan teleskop sehingga banyak waktu yang terbuang. Maka pada permasalahan tersebut kita juga bisa mengguanakan permasalahan ini untuk menentukan rute pemindahan teleskop terpendek.

TSP dapat di modelkan dengan bentuk graf tak bearah berbobot dengan setiap simpul menyatakan sebuah kota dan setiap sisi menyatakan sebagai jalan dan bobot sisi menyatakan jarak. Dengan demikian kita dapat menyelesaikan permasalahan ini sebagai persoalan graf.

Salah satu cara untuk menyelesaikan masalah ini adalah dengan menggunakan metode dynamic programming. Penyelesaian dengan metode dynamic programming dilakukan dengan pertama-tama kita tentukan dulu apa yang akan menjadi state atau tahapannya dan bagaimana transisi dari suatu tahapan ke tahapan lainnya.

Pada permasalahan ini kita akan mengambil state berupa sebuah set graf yang telah dikunjungi. State ini akan direpresentasikan menggunakan teknik bitmasking, hal ini akan mempercepat proses dan akan memperkecil kompleksitas ruang dari sisi pemograman.

Bitmask adalah sebuah teknik manipulasi bit yang akan digunakan untuk merepresentasikan suatu data. Pada permalasan ini kita akan menggunakannya untuk merepresentasikan state dari sebuah simpul apakah simpul tersebut sudah dikunjungi atau belum sebagai contoh kita akan mengambil sebuah bilangan misal 11. bilangan 11 ini bisa direpresentasikan dalam bentuk binernya yaitu

0	1	0	1	1

Pada representasi di atas kita bisa menggap bahwa state dari simpul yang memiliki nomor 0 akan disimpan oleh bit paling kiri dari representasi biner angka 11. dalam hal ini bit paling kiri bernilai 1 dan dengan demikian kita bisa menganggap bahwa simpul nomor 0 telah dikunjungi sebelumnya. Lantas bagaimana kita sapat mengubah state tersebut jika misalkan ada simpul baru yang telah dikunjungi?. Hal ini dapat dimungkinkan dengan adanya operator shift left (<<) dan shift right (>>). misalkan kita memiliki state sekarang yang akan kita sebut cur. Kemudian kita mendapati bahwa simpul nomor n telah di kunjungi maka kita hendak mengubah state cur ke state yang baru. Kita tinggal menggati state cur dengan operasi cur = cur & (1<<n). setelah itu kita akan mendapatkan state baru dimana bit ke n akan bernilai 1.

setelah kita bisa merepresentasikan state dengan sebuah bitmask maka kita sudah bisa menentukan formula dynamic programming untuk permasalahan ini yaitu

```
tsp(pos, 2^n−1)=dist[pos][0]
tsp(pos, mask)=min(dist[pos][nxt] +
tsp(nxt, mask | (1 << nxt))), ∀ nxt
∈ [0..n-1], nxt != pos
```

Dengan formulasi ini kita sudah bisa mendapatkan dua kunci utama dalam dynamic programming yaitu optimal substructur dan overlaping subproblem.

Optimal substructur dalam permasalahan ini dapat dilihat bahwa suatu problem bisa menghasilkan problem yang lebih kecil misalkan tsp(1,1) akan memunculkan subproblem-subproblem baru yang seperti tsp(2,3), tsp(3,9).

Overlaping subproblem dalam permasalahan ini didalam subproblem yang telah terbentuk tadi masih ada banyak kemungkinan akan terbentuk subproblem-subproblem baru yang sebelumnya telah muncul pada tahap yang sama dengan demikian formulasi ini dapat dikatakan memiliki overlaping subproblem.

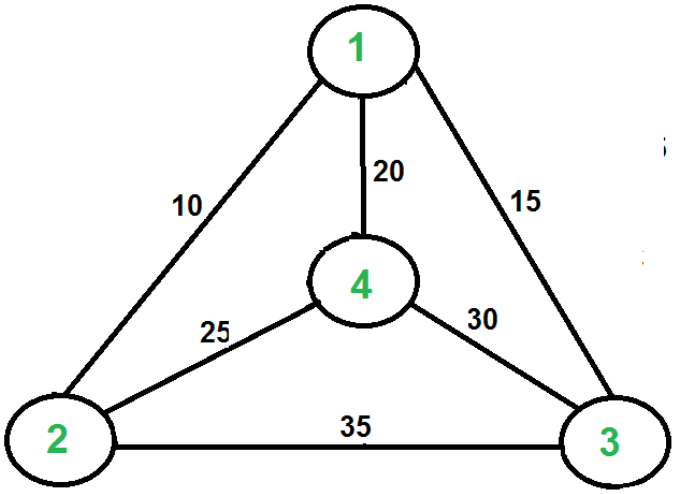
Setelah menentukan statenya maka kita juga harus menentukan bagaiman bentuk tabel memoisasinya. Biasanya tabel memoisasi yang digunakan memiliki jumlah dimensi yang sama dengan jumlah state yang dimiliki sebuah subproblem itu. Dengan demikian kita membutuhkan tabel dua dua dimensi yang akan menyimpan nilai dari setiap pasang state.

Dengan demikian kita sudah memnpunyai semua komponen untuk membuat algoritma dynamic programming untuk permasalahan ini. Berikut ini adalah implementasi dalam bahasa C.

```
Int n; //jumlah simpul

int memo[n][1<<(n+1)-1]; //tabel memo

int map[n][n]; // jarak antar kota

Int tsp(int I, int mask) {
      if ( mask == 1<<(n+1)-1 ) {
            return map[i][0];
      } else
      if ( memo[i][mask] != -1 ) {
            return memo[i][mask];
      } else {
            for (int j=0; j<n;j++) {
                  if (I!=j &&
(mask&1<<j)) {
                        memo[i][mask]  =
min(memo[i][mask], map[i][j] + tsp(j,
mask | 1<<j));
                  }
            }
            return memo[i][mask];
      }
}
```

untuk menjalankan algoritma tersebut kita tinggal memanggil tsp(0,0) dimana kita akan mulai mengunjungi setiap simpul mulai dari simpul bernomor 0

Untuk menguji kebenaran algoritma kita maka kita menggunakan sebuah testcase yang memiliki representasi bentuk graf seperti dibawah ini.



Pada sebuah testcase diatas terdapat 4 simpul dan 6 sisi. Graf diatas dapat direpresesntasikan dengan matrix adjacency berikut

U / V	1	2	3	4
1	0	10	15	20
2	10	0	35	25
3	15	35	0	30
4	20	25	30	0

Hal yan pertama yang harus dilakukan sebelum jalanan algoritma ini dadalah kita harus menyimpan tabel memo terlebih dahulu tabel akan menampung solusi dari subproblem yang akan dipecahkan. Pada keaadaan awal seluruh isi tabel diinisiasi dengan nilai -1 yang menandakan bahwa solusi untuk subproblem yang diwakilkan oleh pasangan baris dan kolom tersebut belum dipecahkan. Ukuran dari tabel tersebut juga harus diperhatikan agar tidak terjadi index out of bound. Kita akan mengalokasikan memori dengan jumlah baris = jumlah simpul dan jumlah kolom = jumlah semua kemungkinan state oleh variabel mask. Dalam hal ini nilai maksimal varibel mask akan tergantung dari variabel n yaitu 1<<(n+1)-1.

Setelah tsp(0,0) dipanggil maka dia akan mencoba seluruh kemungkinan simpul yang dapat dikunjungi untuk satu tahapan kemudian algoritma akan memilih state dengan jarak sementara yang terkecil kemudian akan mencatatnya pada tabel memo. Hal ini akan dilakukan secara berulang sehingga semua seluruh simpul terkunjungi. Setelah menjalankan algoritma ini maka akan diperoleh hasil 80 yang merupakan jarak terpendek dari satu lintasan yang melalui seluruh simpul. Dengan demikian kita sudah berhasil membangun sebuah algoritma dynamic programming untuk menyelesaikan masalah traveling salesman yang merupakan permasalahan yang dikategorikan sebagai NP-Hard.

Algoritma diatas akan berjalan dengan kompleksitas waktu O (2^n * n^2) dengan n sebagai jumlah simpul atau kota pada permasalahan tersebut. Hal ini termasuk solusi yang bagus berhubung permasalahan ini dikategorikan sebagai NP-Hard.

IV. KESIMPULAN DAN SARAN

Dynamic programming adalah sebuah paradigma pemecahan masalah dengan cara membagi masalah menjadi masalah yang lebih kecil kemudian mencari hasilnya dan mencatat hasil yang ditemukan dalam sebuah tabel memo dengan harapan solusi tersebut dapat digunakan kembali ketika permasalahan yang sama muncul kembali.

Traveling salesman problem atau yang biasa disebut dengan TSP merupakan sebuah permasalahan dalam dunia *computer science* yang dikategorikan sebagai permasalahan NP-Hard. Salah satu cara yang ampuh untuk menyelesaikan permasalahan tsp ini adalah dengan menggunakan metode dynamic programming. Solusi yang dihasilkan dengan metode

ini memiliki kompleksitas waktu O(2^n * n^2) yang merupakan sebuah solusi yang cukup bagus untuk permasalahan yang dikategorikan sebagai NP-Hard.

UCAPAN TERIMA KASIH

Puji syukur kepada Allah Yang Maha Esa karena atas berkat-Nya telah memampukan penulis untuk menyelesaikan makalah ini dengan baik dan telap waktu. Penulis juga menyampaikan terima kasih kepada Bapak Rinaldi Munir yang telah mengajarkan dasar-dasar teori yang diperlukan untuk Penulis dapat menyelesaikan makalah ini.

REFERENSI

[1] https://www.geeksforgeeks.org/travelling-salesman-problem-set-1/
[2] Steven, Felix, Halim, Competitive programming 3rd edition, p110-111
[3] https://en.wikipedia.org/wiki/Travelling_salesman_problem
[4] https://en.wikipedia.org/wiki/Dynamic_programming

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, 13 Mei 2018

Maulana Akmal
13516084