Algoritma Penyelesaian Travelling Salesman Problem Menggunakan Dynamic Programming

Disusun oleh:
Benedict Presley
13523067
13523067@std.stei.itb.ac.id

Pendahuluan

Travelling Salesman Problem (TSP) merupakan salah satu masalah klasik dalam bidang optimasi. Diberikan sejumlah kota dan jarak antar kota, tujuan dari TSP adalah menentukan urutan kunjungan ke seluruh kota tepat satu kali, kembali ke kota asal, dan meminimalkan total jarak perjalanan. Meskipun tampak sederhana, TSP termasuk dalam kategori masalah NP-Hard, artinya tidak diketahui algoritma yang dapat menyelesaikannya dalam waktu polinomial untuk semua kasus.

Salah satu pendekatan yang dapat digunakan untuk menyelesaikan TSP dengan ukuran kecil hingga menengah adalah Dynamic Programming (DP). Pendekatan ini memanfaatkan sifat *overlapping subproblems* dan *optimal substructure* untuk menghindari perhitungan yang tidak perlu sehingga mempercepat perhitungan secara signifikan. Dalam paper ini, akan dijelaskan bagaimana TSP dapat diselesaikan menggunakan pendekatan DP dalam bahasa pemrograman Scala.

Strategi Penyelesaian

Dynamic Programming adalah teknik pemrograman yang digunakan untuk menyelesaikan masalah kompleks dengan memecahnya menjadi submasalah yang lebih sederhana. DP menyimpan hasil perhitungan submasalah dalam memori (biasanya menggunakan tabel atau cache) sehingga tidak perlu menghitung ulang submasalah yang sama berkali-kali. Teknik ini sangat efektif untuk masalah yang memiliki dua sifat utama:

1. *Overlapping Subproblems*: Submasalah yang sama muncul berulang kali selama perhitungan.
2. *Optimal Substructure*: Solusi dari masalah utama dapat dibentuk dari solusi optimal submasalah.

TSP cocok diselesaikan dengan DP karena kedua sifat tersebut terpenuhi.

Untuk memformulasikan solusi TSP dengan DP, kita mendefinisikan state sebagai pasangan dari:

- mask, *bitmask* yang merepresentasikan kota mana saja yang telah dikunjungi.
- u, kota terakhir yang dikunjungi.

State dan definisi DP:
$dp(mask,\ u)$ = jarak minimum untuk mengunjungi semua kota yang dinyalakan dalam mask, berakhir di kota u

Base Case:
Jika hanya kota awal 0 yang dikunjungi:
$dp(1, 0)\ =\ 0$
Karena kita mulai dari kota 0 dan belum menempuh jarak apa pun. Bitmask yang hanya mengunjungi kota 0 adalah 1.

Transition Function:
Untuk menghitung $dp(mask,\ u)$, kita mencoba semua kota v yang sudah ada di mask dan bukan u, dan menghitung:
$dp(mask,\ u)\ =\ min(dp(mask - 2^{u}, v)\ +\ dist(v, u)$
Artinya, jika kita sekarang berada di kota u dan telah mengunjungi semua kota di mask, maka kita datang dari salah satu kota v di mask, lalu menambahkan jarak dari v ke u.

Implementasi

Main.scala

```
object Main {
    val INF: Int = Int.MaxValue / 2

    def solve(M: Array[Array[Int]]): (Int, List[Int], Double) = {
        val startTime = System.nanoTime()

        val N = M.length
        val dp = Array.fill(1 << N, N)(INF)
        val pre = Array.fill(1 << N, N)(-1)

        dp(1)(0) = 0

        for (mask ← 0 until (1 << N)) {
            for (i ← 0 until N if (mask & (1 << i)) ≠ 0) {
                val oldMask = mask ^ (1 << i)
                for (j ← 0 until N if (mask & (1 << j)) ≠ 0) {
```

```
                    val cost = dp(oldMask)(j) + M(j)(i)
                    if (cost < dp(mask)(i)) {
                        dp(mask)(i) = cost
                        pre(mask)(i) = j
                    }
                }
            }
        }

        var minCost = INF
        var last = -1
        for (i <- 0 until N) {
            val cost = dp((1 << N) - 1)(i) + M(i)(0)
            if (cost < minCost) {
                minCost = cost
                last = i
            }
        }

        var mask = (1 << N) - 1
        val path = scala.collection.mutable.ArrayBuffer[Int]()
        while (mask != 1) {
            path += last
            val to = pre(mask)(last)
            mask ^= (1 << last)
            last = to
        }
        path += 0

        val finalPath = path.reverse.toList :+ 0

        val endTime = System.nanoTime()
        val elapsedTime = (endTime - startTime) / 1e6

        (minCost, finalPath, elapsedTime)
    }
}
```

Uji Coba

<table>
<tr><th>No</th><th>Kasus Uji</th></tr>
<tr><td>1</td><td><pre>Test 1
0, 10, 15, 20
10, 0, 35, 25
15, 35, 0, 30
20, 25, 30, 0
Minimum Cost: 80
Path: 0 -> 2 -> 3 -> 1 -> 0
Time taken: 7.6578 ms</pre></td></tr>
<tr><td>2</td><td><pre>Test 2
0, 1, 1, 1, 1
1, 0, 1, 1, 1
1, 1, 0, 1, 1
1, 1, 1, 0, 1
1, 1, 1, 1, 0
Minimum Cost: 5
Path: 0 -> 4 -> 3 -> 2 -> 1 -> 0
Time taken: 1.3904 ms</pre></td></tr>
<tr><td>3</td><td><pre>Test 3
0, 345, 872, 123, 567, 789, 234, 901
345, 0, 456, 678, 234, 890, 123, 567
872, 456, 0, 345, 678, 234, 890, 123
123, 678, 345, 0, 456, 789, 234, 901
567, 234, 678, 456, 0, 345, 678, 234
789, 890, 234, 789, 345, 0, 456, 678
234, 123, 890, 234, 678, 456, 0, 345
901, 567, 123, 901, 234, 678, 345, 0
Minimum Cost: 1983
Path: 0 -> 3 -> 6 -> 7 -> 2 -> 5 -> 4 -> 1 -> 0
Time taken: 2.9413 ms</pre></td></tr>
</table>

4	Test 4 0, 234, 567, 890, 123, 456, 789, 321, 654, 987 234, 0, 345, 678, 901, 234, 567, 890, 123, 456 567, 345, 0, 456, 789, 123, 890, 234, 567, 901 890, 678, 456, 0, 345, 678, 123, 456, 789, 234 123, 901, 789, 345, 0, 567, 890, 123, 456, 678 456, 234, 123, 678, 567, 0, 345, 789, 234, 890 789, 567, 890, 123, 890, 345, 0, 567, 901, 345 321, 890, 234, 456, 123, 789, 567, 0, 678, 123 654, 123, 567, 789, 456, 234, 901, 678, 0, 345 987, 456, 901, 234, 678, 890, 345, 123, 345, 0 Minimum Cost: 2007 Path: 0 -> 4 -> 7 -> 9 -> 6 -> 3 -> 2 -> 5 -> 8 -> 1 -> 0 Time taken: 8.9984 ms
5	Test 5 0, 789, 234, 567, 890, 123, 456, 789, 321, 654, 987, 123 789, 0, 345, 678, 234, 901, 567, 890, 123, 456, 789, 234 234, 345, 0, 456, 789, 123, 890, 234, 567, 901, 345, 678 567, 678, 456, 0, 345, 678, 123, 456, 789, 234, 890, 567 890, 234, 789, 345, 0, 567, 890, 123, 456, 678, 234, 901 123, 901, 123, 678, 567, 0, 345, 789, 234, 890, 567, 345 456, 567, 890, 123, 890, 345, 0, 567, 901, 345, 678, 890 789, 890, 234, 456, 123, 789, 567, 0, 678, 123, 456, 789 321, 123, 567, 789, 456, 234, 901, 678, 0, 345, 789, 234 654, 456, 901, 234, 678, 890, 345, 123, 345, 0, 567, 890 987, 789, 345, 890, 234, 567, 678, 456, 789, 567, 0, 123 123, 234, 678, 567, 901, 345, 890, 789, 234, 890, 123, 0 Minimum Cost: 2340 Path: 0 -> 8 -> 1 -> 11 -> 10 -> 4 -> 7 -> 9 -> 3 -> 6 -> 5 -> 2 -> 0 Time taken: 43.4667 ms

6	Test 6 0, 921, 372, 874, 628, 196, 538, 884, 347, 779, 190, 935, 421, 578, 436, 233, 337, 601, 498, 255 921, 0, 188, 337, 804, 972, 421, 628, 812, 477, 590, 786, 608, 483, 667, 915, 753, 192, 696, 800 372, 188, 0, 608, 599, 723, 207, 914, 681, 898, 418, 834, 912, 700, 742, 385, 471, 486, 584, 355 874, 337, 608, 0, 939, 667, 544, 386, 597, 932, 499, 448, 286, 642, 617, 783, 494, 987, 706, 314 628, 804, 599, 939, 0, 211, 476, 554, 258, 812, 891, 746, 596, 857, 177, 339, 292, 753, 483, 918 196, 972, 723, 667, 211, 0, 488, 798, 152, 726, 873, 621, 330, 879, 117, 285, 153, 600, 431, 703 538, 421, 207, 544, 476, 488, 0, 760, 709, 833, 495, 771, 888, 697, 592, 300, 453, 619, 672, 622 884, 628, 914, 386, 554, 798, 760, 0, 968, 487, 727, 357, 329, 750, 806, 964, 765, 967, 313, 401 347, 812, 681, 597, 258, 152, 709, 968, 0, 901, 594, 862, 515, 952, 174, 262, 181, 498, 278, 674 779, 477, 898, 932, 812, 726, 833, 487, 901, 0, 910, 255, 368, 543, 837, 947, 841, 693, 339, 316 190, 590, 418, 499, 891, 873, 495, 727, 594, 910, 0, 943, 460, 781, 710, 389, 581, 597, 702, 237 935, 786, 834, 448, 746, 621, 771, 357, 862, 255, 943, 0, 336, 513, 776, 842, 775, 926, 202, 339 421, 608, 912, 286, 596, 330, 888, 329, 515, 368, 460, 336, 0, 742, 547, 431, 190, 790, 487, 305 578, 483, 700, 642, 857, 879, 697, 750, 952, 543, 781, 513, 742, 0, 664, 746, 802, 832, 362, 392 436, 667, 742, 617, 177, 117, 592, 806, 174, 837, 710, 776, 547, 664, 0, 401, 170, 553, 473, 679 233, 915, 385, 783, 339, 285, 300, 964, 262, 947, 389, 842, 431, 746, 401, 0, 348, 671, 510, 715 337, 753, 471, 494, 292, 153, 453, 765, 181, 841, 581, 775, 190, 802, 170, 348, 0, 528, 455, 631 601, 192, 486, 987, 753, 600, 619, 967, 498, 693, 597, 926, 790, 832, 553, 671, 528, 0, 707, 811 498, 696, 584, 706, 483, 431, 672, 313, 278, 339, 702, 202, 487, 362, 473, 510, 455, 707, 0, 241 255, 800, 355, 314, 918, 703, 622, 401, 674, 316, 237, 339, 305, 392, 679, 715, 631, 811, 241, 0 Minimum Cost: 5310 Path: 0 -> 15 -> 6 -> 2 -> 1 -> 17 -> 8 -> 4 -> 14 -> 5 -> 16 -> 12 -> 3 -> 7 -> 9 -> 11 -> 18 -> 13 -> 19 -> 10 -> 0 Time taken: 8158.4995 ms

Kesimpulan

Dalam *paper* ini, telah dibahas solusi untuk Travelling Salesman Problem (TSP) menggunakan pendekatan Dynamic Programming *bottom-up* berbasis *bitmasking* dalam bahasa Scala. Pendekatan ini menghindari perhitungan ulang submasalah dengan membangun solusi dari ukuran terkecil ke terbesar secara eksplisit, yang merupakan ciri khas dari *bottom-up* dynamic programming.

Kompleksitas waktu dari solusi ini adalah $O(2^n n^2)$, di mana n adalah jumlah kota. Ini berasal dari 2^n kemungkinan subset kota, dan untuk setiap subset kita mengevaluasi hingga n pilihan kota awal dan n pilihan kota tujuan.

Kompleksitas ruang juga adalah $O(2^n n)$ karena kita menyimpan tabel DP dan tabel pendahulu (pre) untuk semua kombinasi subset dan posisi kota.