**LAPORAN TUGAS KECIL II**

**IF2211 STRATEGI ALGORITMA**

Membangun Kurva Bézier dengan Algoritma Titik Tengah berbasis *Divide and Conquer*

A black background with white text

Description automatically generated

Disusun oleh:

Adril Putra Merin 13522068

**Program Studi Teknik Informatika**

**Sekolah Teknik Elektro dan Informatika**

**Institut Teknologi Bandung**

**2024**

**Daftar Isi**

[BAB I DASAR TEORI 3](#_Toc161676534)

[**A.** **Algoritma Brute Force** 3](#_Toc161676535)

[**B.** **Algoritma Divide and Conquer** 3](#_Toc161676536)

[**C.** **Bezier Curve** 3](#_Toc161676537)

[BAB II ANALISIS DAN IMPLEMENTASI ALGORITMA 5](#_Toc161676538)

[**A.** **Algoritma *Brute Force*** 5](#_Toc161676539)

# BAB I DASAR TEORI

## **Algoritma Brute Force**

Algoritma *brute force* adalah algoritma yang memiliki pendekatan sederhana, tetapi cukup mampu untuk menyelesaikan berbagai persoalan. Sesuai dengan namanya, algoritma *brute force* atau, diterjemahkan secara bebas, *tenaga kasar*, menyelesaikan persoalan dengan mencoba semua kemungkinan solusi hingga ditemukan solusi yang benar atau optimal. Secara umum, algoritma *brute force* tidak selalu menjadi pilihan yang efisien karena cenderung membutuhkan waktu yang sangat besar seiring meningkatnya ukuran ruang solusi. Karena itu, algoritma yang lebih cerdas dan efisien seringkali lebih dipilih untuk menyelesaikan persoalan tertentu.

## **Algoritma Divide and Conquer**

Algoritma *divide and conquer* adalah sebuah metode pemecahan masalah yang terdiri dari tiga langkah utama: *divide*, *conquer*, dan *combine*. Metode ini sering digunakan untuk memecahkan persoalan kompleks dengan membagi persoalan tersebut menjadi subpersoalan yang lebih kecil, menyelesaikan subpersoalan tersebut secara rekursif, dan kemudian menggabungkan solusi dari subpersoalan tersebut menjadi solusi untuk masalah asli. Berikut adalah langkah-langkah umum dari algoritma *divide* *and conquer*.

1. *Divide*: Persoalan utama dibagi menjadi dua atau lebih subpersoalan yang lebih kecil dan serupa dengan persoalan asli. Pembagian dilakukan hingga submasalah memiliki ukuran yang cukup kecil untuk diselesaikan (*base case*) atau hingga iterasi tertentu.
2. *Conquer*: Setiap subpersoalan diselesaikan secara rekursif. Hal ini dapat dilakukan dengan menerapkan algoritma *divide and conquer* pada setiap subpersoalan, atau dengan menggunakan algoritma lain bergantung pada jenis persoalan dan kompleksitas submasalah.
3. *Combine*: Solusi dari berbagai subpersoalan digabungkan hingga memberikan solusi untuk masalah utama atau asli.

## **Bezier Curve**

*Bezier curve* adalah kurva parametrik yang banyak digunakan dalam grafika computer dan banyak bidang lainnya. *Bezier curve* didefinisikan oleh sekumpulan titik-titik kontrol yang melalui  hingga dengan adalah orde dari kurva tersebut ( linear, kuadratik, kubik, dan seterusnya). Titik kontrol dan berturut-turut adalah titik awal dan titik akhir dari kurva yang dihasilkan, sedangkan titik kontrol diantara kedua titik tersebut biasanya tidak terletak pada kurva yang dihasilkan. Misalkan dan adalah dua titik yang berbeda, *linear bezier curve* dalam kasus ini adalah garis yang menghubungkan kedua titik tersebut sehingga fungsi parameter dari kurva ini adalah

*Bezier curve* dapat didefinisikan secara rekursif untuk sebarang derajat dengan mengekspresikan fungsi parameter sebagai interpolasi linear antara pasangan titik terkait pada dua *bezier curve* dengan derajat . Misalkan adalah fungsi parameter *bezier curve* dengan titik-titik kontrol , maka definisi rekursif dari *bezier curve* adalah

# BAB II ANALISIS DAN IMPLEMENTASI ALGORITMA

## **Algoritma *Brute Force***

Salah satu pendekatan yang digunakan untuk memecahkan masalah ini adalah algoritma *brute force*. Pada program ini, *brute force* dilakukan dengan rekursif sesuai dengan definisi general *bezier curve* secara rekursif, yaitu:

dengan adalah titik-titik kontrol dari sebuah kurva bezier dengan derajat . Jadi derajat yang digunakan pada algoritma ini sudah digeneralisasi untuk sebarang orde n dengan n bilangan bulat positif. Adapun langkah-langkah algoritma *brute force* pada program ini adalah sebagai berikut.

1. Jika banyak titik kontrol kurang dari satu, maka kembalikan titik kontrol tersebut sebagai titik hasil. Jika tidak, lanjutkan pada langkah ke-2.
2. Hitung berapa banyak titik yang dihasilkan dimana dengan adalah banyak iterasi. Banyak titik ini akan menentukan banyak yang digunakan pada fungsi parameter dengan . Misalkan banyak iterasi adalah 2, maka banyak titik adalah 3 sehingga . Perhatikan bahwa banyak titik yang dihitung di awal adalah jumlah titik hasil kurva yang bukan merupakan titik kontrol awal dan akhir sehingga pada akhirnya, banyak titik yang dihasilkan adalah .
3. Lakukan kalkulasi untuk setiap berdasarkan fungsi parameter diatas sehingga tiap kalkulasi akan menghasilkan tepat satu titik sehingga jumlah semua titik adalah .

Alasan mengapa banyak titik yang dihasilkan adalah perpangkatan dua dari banyak iterasi adalah untuk mempermudah perbandingan algoritma *brute force* dengan *divide and conquer*. Hal ini karena banyak titik yang dihasilkan dengan algoritma *divide and conquer* untuk *k* iterasi adalah . Adapun implementasi algoritma ini dalam bahasa Go adalah sebagai berikut.

|  |
| --- |
| package algorithms  import (    "bezier/backend/models"    "math"  )  func calculateBezier(t float64, bp models.BezierPoints, low int, hi int) models.Point {    if low == hi {      return bp.Points[low]    }    point1 := calculateBezier(t, bp, low, hi-1)    point2 := calculateBezier(t, bp, low+1, hi)    point1.X \*= (1 - t)    point1.Y \*= (1 - t)    point2.X \*= t    point2.Y \*= t    result := models.Point{X: point1.X + point2.X, Y: point1.Y + point2.Y}    return result  }  func GetPointsBruteforce(bp models.BezierPoints) models.BezierPoints {    if bp.Neff < 2 {      return bp    }    iter := int(math.Pow(2, float64(bp.Iteration))) - 1    result := models.BezierPoints{Iteration: bp.Iteration, Neff: 2 + iter}    for i := 0; i < iter+2; i++ {      result.Points = append(result.Points, calculateBezier(float64(i)/float64(1+iter), bp, 0, bp.Neff-1))    }    return result  } |

Berdasarkan algoritma tersebut kita dapat melakukan analisis terhadap kompleksitas waktunya. Untuk proses *calculateBezier* dengan banyak titik *n*, banyak iterasi *k*, dan konstanta kompleksitas waktu yang diperoleh adalah

Dengan menyederhanakan ekspresi tersebut, diperoleh

Selanjutnya, proses *calculateBezier* dilakukan sebanyak kali sehingga kompleksitas waktu total adalah .