**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](#_Toc161696775)

[**A.** **Algoritma Brute Force** 3](#_Toc161696776)

[**B.** **Algoritma Divide and Conquer** 3](#_Toc161696777)

[**C.** **Bezier Curve** 3](#_Toc161696778)

[BAB II ANALISIS DAN IMPLEMENTASI ALGORITMA 5](#_Toc161696779)

[**A.** **Algoritma *Brute Force*** 5](#_Toc161696780)

[**B.** **Algoritma *Divide and Conquer*** 6](#_Toc161696781)

[BAB III SOURCE CODE 9](#_Toc161696782)

[**A.** **Struktur Data dan Metode** 9](#_Toc161696783)

[**B.** **Algoritma Brute Force** 10](#_Toc161696784)

[***C.*** **Algoritma *Divide and Conquer*** 11](#_Toc161696785)

[BAB IV EKSPERIMEN 13](#_Toc161696786)

[**A.** **Uji kasus 1** 13](#_Toc161696787)

[**B.** **Uji Kasus 2** 14](#_Toc161696788)

# 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. Agar proses kalkulasi lebih optimal, proses untuk fungsi rekursi diatas dilakukan secara iteratif (*bottom-up*).

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 . 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 . Jadi, parameter akan memiliki bentuk dengan dan bilangan bulat. Misalkan banyak iterasi adalah 2, maka banyak titik (tanpa titik kontrol) adalah 3 sehingga .
3. Untuk setiap , akan dilakukan kalkulasi menggunakan fungsi rekursi diatas secara iteratif. Lakukan kalkulasi dimulai dari *bezier curve* orde 1 hingga orde ke-n. Sebagai ilustrasi, berikut adalah contoh urutan perhitungan untuk kasus .

A diagram of a number of letters and numbers

Description automatically generated

1. Masukkan setiap titik hasil kalkulasi *t* ke dalam *list of points* hasil.

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 .

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

Dengan menyederhanakan ekspresi tersebut, diperoleh

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

## **Algoritma *Divide and Conquer***

Algoritma *divide* *and conquer* dapat digunakan untuk menyelesaikan persoalan *bezier* *curve* dengan *n* buah titik kontrol dan *k* buah iterasi. Algoritma ini akan menghasilkan sebanyak titik yang akan digunakan untuk mengkontruksi *bezier curve*.

Berdasarkan permasalahan yang diberikan, penulis sudah melakukan generalisasi algoritma *middle point* dengan *divide and conquer* untuk sebarang orde *n*. Secara umum, untuk buah iterasi dan buah titik, langkah-langkah dari algoritma ini adalah sebagai berikut.

1. Inisialisasi *kedalaman* saat ini sebagai 0.
2. Lakukan operasi terhadap titik-titik kontrol saat ini sehingga dihasilkan *list of point* *left*, *mid*, dan *right*. Operasi tersebut dilakukan dengan cara mencari titik tengah dari segmen garis yang dibentuk oleh titik kontrol yang berurutan sehingga dihasilkan titik tengah. Ulangi operasi tersebut dengan titik tengah yang didapatkan sebelumnya sebagai titik kontrol baru hingga ditemukan satu titik tengah. Berikut adalah ilustrasi operasi ini untuk banyak titik kontrol awal sebanyak .

A diagram of a line with letters and numbers

Description automatically generated

Pada awalnya, titik-titik kontrol pada operasi tersebut adalah A, B, C, dan D. Selanjutnya, diperoleh titik-titik tengah dari titik kontrol tersebut sebagai E, F, G. Lakukan operasi tersebut hingga ditemukan 1 titik tengah (*mid*), yang dalam hal ini adalah J. Selain *mid*, operasi ini juga akan menghasilkan *list* *left* dan *right* yang berisi titik kontrol baru untuk *kedalaman* selanjutnya. *List left* berisi titik kontrol awal pertama dan kumpulan titik tengah paling awal dari setiap iterasi pada operasi pencarian titik tengah, sedangkan *right* berisi titik kontrol terakhir dan kumpulan titik tengah paling akhir dari setiap iterasi pada operasi pencarian titik tengah. Misalkan, pada contoh diatas, *left* akan berisi elemen {C, E, H, J}, sedangkan *right* akan berisi elemen {J, I, G, D}.

1. Ulangi Langkah (2) dengan titik kontrol *left* dan *right* hingga *kedalaman* sama dengan banyak iterasi yang diinginkan.
2. Gabungkan hasil titik tengah pada titik kontrol *left* dan *right* dengan mid pada langkah (2).

Kita dapat melakukan analisis kompleksitas waktu terhadap algoritma tersebut. Untuk buah titik dan buah iterasi diperoleh kompleksitas waktu sebagai berikut

Dengan melakukan penyederhanaan terhadap persamaan diatas, diperoleh kompleksitas waktu sebagai berikut.

Jadi,

Namun, kompleksitas waktu tersebut belum mempertimbangkan proses penggabungan hasil proses *left*, *mid*, dan *right* yang pada kode di atas menggunakan *InsertAfter*. Untuk *k* buah iterasi, dilakukan *insertion* titik pada *list* sebanyak kali sehingga kompleksitas waktu total menjadi.

Dengan menyederhanakan ekspresi tersebut diperoleh hasil sebagai berikut.

Jadi kompleksitas waktu total dalam notasi big-O adalah

# BAB III SOURCE CODE

Algoritma pada program ini menggunakan bahasa Go, sedangkan bagian *frontend* menggunakan bahasa *typescript* menggunakan *React*. Karena *source code* untuk program ini cukup panjang, penulis hanya akan menampilkan *source code* yang berkaitan dengan algoritma pada laporan ini.

## **Struktur Data dan Metode**

|  |
| --- |
| bezier\_points.go |
| package models  type BezierPoints struct {    Points    []Point `json:"points"`    Neff      int     `json:"neff"`    Iteration int     `json:"iteration"`  }  func (bp \*BezierPoints) InsertBefore(newBp BezierPoints) {    bp.Points = append(newBp.Points, bp.Points...)    bp.Iteration = newBp.Iteration    bp.Neff += newBp.Neff  }  func (bp \*BezierPoints) InsertAfter(newBp BezierPoints) {    bp.Points = append(bp.Points, newBp.Points...)    bp.Iteration = newBp.Iteration    bp.Neff += newBp.Neff  } |

|  |
| --- |
| point.go |
| package models  type Point struct {    X float64 `json:"x"`    Y float64 `json:"y"`  } |

|  |
| --- |
| response.go |
| package models  type Response struct {    Result BezierPoints `json:"result"`    Time   float64      `json:"time"`  } |

## **Algoritma Brute Force**

|  |
| --- |
| bruteforce.go |
| package algorithms  import (    "bezier/backend/models"    "math"  )  func calculateBezier(t float64, bp models.BezierPoints) models.Point {    res := models.BezierPoints{Iteration: bp.Iteration, Neff: 0, Points: []models.Point{}}    res.InsertAfter(bp)    for i := 0; i < bp.Neff-1; i++ {      for j := 0; j < res.Neff-1; j++ {        res.Points[j] = models.Point{X: (1-t)\*res.Points[j].X + t\*res.Points[j+1].X, Y: (1-t)\*res.Points[j].Y + t\*res.Points[j+1].Y}      }      res.Neff--      res.Points = res.Points[:res.Neff]    }    return res.Points[0]  }  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))    }    return result  } |

## **Algoritma *Divide and Conquer***

|  |
| --- |
| dnc.go |
| package algorithms  import (    "bezier/backend/models"  )  func getMidPoints(bp models.BezierPoints) (models.BezierPoints, models.BezierPoints, models.BezierPoints) {    mid := models.BezierPoints{Iteration: bp.Iteration, Neff: 0, Points: []models.Point{}}    left := models.BezierPoints{Iteration: bp.Iteration, Neff: 0, Points: []models.Point{}}    right := models.BezierPoints{Iteration: bp.Iteration, Neff: 0, Points: []models.Point{}}    left.InsertAfter(models.BezierPoints{Neff: 1, Points: []models.Point{bp.Points[0]}, Iteration: bp.Iteration})    right.InsertAfter(models.BezierPoints{Neff: 1, Points: []models.Point{bp.Points[bp.Neff-1]}, Iteration: bp.Iteration})    mid.InsertAfter(bp)    for i := 0; i < bp.Neff-1; i++ {      for j := 0; j < mid.Neff-1; j++ {        mid.Points[j] = models.Point{X: (mid.Points[j].X + mid.Points[j+1].X) / 2, Y: (mid.Points[j].Y + mid.Points[j+1].Y) / 2}      }      mid.Neff--      mid.Points = mid.Points[:mid.Neff]      left.InsertAfter(models.BezierPoints{Neff: 1, Points: []models.Point{mid.Points[0]}, Iteration: bp.Iteration})      right.InsertBefore(models.BezierPoints{Neff: 1, Points: []models.Point{mid.Points[mid.Neff-1]}, Iteration: bp.Iteration})    }    return left, mid, right  }  func findPoints(bp models.BezierPoints, dep int) models.BezierPoints {    if dep >= bp.Iteration {      return models.BezierPoints{Iteration: bp.Iteration}    }    left, mid, right := getMidPoints(bp)    leftMidPoints := findPoints(left, dep+1)    rightMidPoints := findPoints(right, dep+1)    leftMidPoints.InsertAfter(mid)    leftMidPoints.InsertAfter(rightMidPoints)    return leftMidPoints  }  func GetPointsDnc(bp models.BezierPoints) models.BezierPoints {    if bp.Neff < 2 {      return bp    }    result := models.BezierPoints{Iteration: bp.Iteration, Neff: 2}    result.Points = append(result.Points, bp.Points[0])    result.InsertAfter(findPoints(bp, 0))    result.Points = append(result.Points, bp.Points[bp.Neff-1])    return result  } |

# BAB IV EKSPERIMEN

## **Uji kasus 1**

|  |
| --- |
| Algoritma *Brute Force* |
|  |

|  |
| --- |
| Algoritma *divide and conquer* |
|  |

## **Uji Kasus 2**

|  |
| --- |
| Algoritma *brute force* |
|  |

|  |
| --- |
| Algoritma *divide and conquer* |
|  |

## **Kasus Uji 3**

|  |
| --- |
| Algoritma *brute force* |
|  |

|  |
| --- |
| Algoritma *divide and conquer* |
|  |

## **Kasus Uji 4**

|  |
| --- |
| Algoritma *brute force* |
|  |

|  |
| --- |
| Algoritma *divide and conquer* |
|  |

## **Kasus Uji 5**

|  |
| --- |
| Algoritma *brute force* |
|  |

|  |
| --- |
| Algoritma *divide and conquer* |
|  |

## **Kasus Uji 6**

|  |
| --- |
| Algoritma *brute force* |
|  |

|  |
| --- |
| Algoritma *divide and conquer* |
|  |

# BAB V ANALISIS HASIL EKSPERIMEN

## **Perbandingan Hasil Uji Kasus**

|  |  |  |
| --- | --- | --- |
| Uji Kasus | *Brute Force* (ms) | *Divide and Conquer* (ms) |
| 1 | 0.984 | 2.67 |
| 2 | 0.239 | 0.992 |
| 3 | 0 | 0.071 |
| 4 | 0.096 | 0.349 |
| 5 | 0.304 | 0.439 |
| 6 | 0 | 0.398 |

## **Analisis Hasil Uji Kasus**

Berdasarkan table,