Aplikasi Pohon dalam Binary Space Partitioning dalam Penentuan Lokasi di ITB Ganesha

Fajar Kurniawan - 13523027[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]*13523027@mahasiswa.itb.ac.id, 13523027@std.stei.itb.ac.id*

***Abstract*—BSP merupakan salah satu algoritma yang penting dalam komputasi dan representasi geometri. BSP dapat digunakan sebagai ekstensi dari pencarian dengan *binary search tree* untuk mentelesaikan makalah berkaitan dengan pembagian ruang geometri. Makalah ini membahas mengenai percobaan untuk membuat program penentuan lokasi di dalam peta Kampus ITB Ganesha. Akan ditentukan apakah titik yang diuji terletak di dalam gedung tertentu atau di luar gedung.**

***Kata Kunci*—BSP, Pohon, ITB, Peta, Pencarian.**

I. PENDAHULUAN

Kampus ITB terletak di lebih dari satu daerah sehingga mendapatkan julukan multicampus. Salah satunya adalah ITB Kampus Ganesha di Kota Bandung. Di ITB Ganesha terdapat banyak bangunan yang digunakan untuk berbagai keperluan termasuk melaksanakan kuliah.

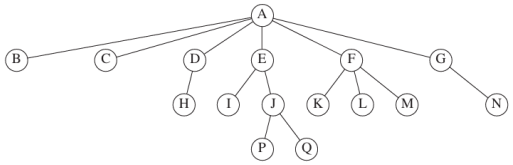
BSP (*Binary Space Partition*) adalah algoritma yang membagi ruang dengan partisi-partisi tertentu sehingga dan Menyusun partisi-partisi tersebut sedemikian rupa sehingga bisa menjadi pohon biner. Pada daun-daun dari pohon biner yang dihasilkan BSP terdapat subruang yang merupakan hasil partisi. Salah satu penerapan BSP yang membuatnya menjadi sangat popular pada awal pengembangan gim video 3D adalah penggunaannya pada DOOM untuk mengoptimasi algoritma *rendering*-nya sehingga jauh lebih efisien dibandingkan *ray casting* karena komputer pada masa itu memiliki keterbatasan kapasitas dan performa komputasi.

Gambar 1. Binary Space Partition DOOM, diambil dari https://doomwiki.org/wiki/File:All_SS_E1M1.png

II. LANDASAN TEORI

1. *Pohon*

Sebuah pohon atau *tree* bisa didefinisikan dalam beberapa cara, salah satu caranya adalah dengan rekursi. Suatu pohon merupakan Kumpulan dari *nodes* atau simpul. Sebuah pohon bisa kosong. Namun jika tidak kosong, sebuah pohon terdiri dari sebuah akar dan nol atau lebih *subtree* atau upapohon yang masing-masing terhubung dengan akar melalui sebuah sisi berarah. Akar dari setiap upapohon bisa disebut sebagai orang tua (selanjutnya akan kita sebut ayah) dari upapohon tersebut dan anak dari akar [1].



Gambar 2. Pohon, diambil dari [1]

Sebuah pohon adalah Kumpulan dari N buah simpul dengan N-1 buah sisi karena setiap sisi menghubungkan sebuah simpul dengan orang tuanya dan setiap simpul memiliki ayah kecuali simpul akar.

Pada gambar 2, akar dari pohon adalah A. Simpul F mempunyai ayah A dan anak K, L, M. Setiap simpul memiliki jumlah anak tertentu (bisa nol). Simpul dengan anak berjumlah nol disebut daun (*leaf).* Pada gambar 2, daunnya adalah simpul B, C, H, I, P, Q, K, L, M, N. Simpul dengan ayah yang sama disebut sebagai saudara, maka dari itu, K, L, dan M adalah saudara.

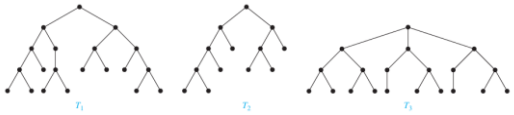
Lintasan dari n1 ke nk dapat didefinisikan sebagai barisan simpul n1, n2, …, nk sedemikian rupa sehingga ni adalah ayah dari ni+1 untuk $1 \leq i < k$. Panjnag lintasan ini adalah banyaknya simpul pada lintasan tersebut yaitu k-1. Pada sebuah pohon, terdapat hanya tepat satu lintasan dari akarnya ke setiap simpul. Kedalaman simpul ni adalah panjang lintasan dari akar ke ni. Dengan demikian, kedalaman akar adalah nol. Pada gambar 2, E ada pada kedalaman 1. Kedalaman sebuah pohon adalah kedalaman dari daun yang paling dalam.

1.1. Pohon Biner

Pohon biner adalah pohon yang jumlah anak dari setiap simpul tidak lebih dari dua. Gambar 2 bukanlah

sebuah pohon biner karena ada simpul yang memiliki lebih dari dua anak.

Sebuah pohon N-er berarti setiap simpul pohon tersebut memiliki maksimal N buah anak. Pohon N-er yang seimbang adalah pohon yang setiap setiap memiliki kedalaman yang relative sama. Definisi ini kurang jelas sehingga dapat ditingkatkan lagi yaitu bahwa sebuah pohon dengan kedalaman H dikatakan seimbang jika pohon ini semua daunnya berada pada kedalaman H atau H-1. Sebagai contoh pada gambar 3, pohon di sebelah kiri adalah pohon seimbang karena setiap daunnya ada pada kedalaman 3 atau 4 sedangkan pohon yang di tengah tidak seimbang karena daunnya terletak pada kedalaman 2, 3, dan 4. Pohon di sebelah kanan adalah pohon seimbang yang dapat dikatakana ideal karena semua daunnya ada pada kedalaman yang tepat sama, yaitu contohnya pada kedalaman 3 [2].



Gambar 3. Pohon Seimbang dan Tidak Seimbang, diambil dari [2]

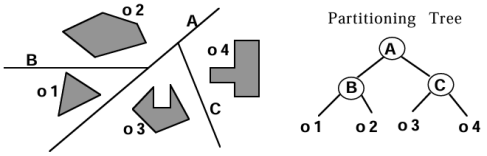
1.2 Binary Search Tree

Operasi pencarian (searching) adalah salah satu tugas paling penting dalam komputasi. Binary *search tree* adalah salah satu metode untuk mencari suatu objek dengan cukup efisien. *Binary search tree* adalah pohon biner yang setiap anak dari setiap simpulnya dibagi menjadi anak kiri atau anak kanan, tidak ada simpul yang memiliki lebih dari satu anak kanan atau anak kiri dan setiap simpul diberi label dengan sebuah *key* atau penanda yang merupakan salah satu dari objek atau *item* yang ingin dicari. Setiap simpul diberi *key* sedemikian rupa sehingga semua simpul pada upapohon anak kanan dan semua keturunanannya dari sebuah simpul memiliki *key* yang lebih besar simpul tersebut sedangkan upapohon anak kirinya akan memiliki *key* yang lebih kecil [2].

Prosedur pembuatan *binary search tree* ini dilakukan secara rekursif dengan pertama-tama membuat sebuah simpul sebagai akar dari pohon. Kemudian untuk menambahkan simpul baru, bandingkan terlebih dahulu *key* dari simpul tersebut dengan simpul yang sudah ada. Jika *key* nya lebih kecil makan ditambahkan ke pohon sebagai anak kiri dan jika lebih besar sebagai anak kanan.

2. *Binary Space Partition (BSP)*

Pada sebagian besar komputasi dengan model geometri 3D, manipulasi objek dan menghasilkan gambarnya adalah operasi yang sangat penting. Untuk melakukan operasi ini, perlu ditentukan hubungan spasial antar objek, bagaimana objek-objek tersebut bersinggungan dan saling menutupi. Namun, objek-objek tersebut sering kali terdiri dari banyak bagian, seperti sejumlah poligon yang membentuk permukaan kompleks. Untuk menghitung hubungan spasial antara N poligon dengan cara brute force memerlukan perbandingan setiap pasangan poligon, sehingga membutuhkan waktu $O(n^2)$. Jumlah operasi dapat dikurangi secara signifikan menjadi $O(n\times \log^2 n)$ ketika objek-objek saling berpotongan ketika mereka agak terpisah satu sama lain. Hal ini dapat ini dapat dicapai dengan menggunakan *Binary Space Partition Tree* atau Pohon BSP. Pohon BSP menyediakan representasi komputasi dari ruang sekaligus sebuah struktur pencarian dan representasi geometri [3].

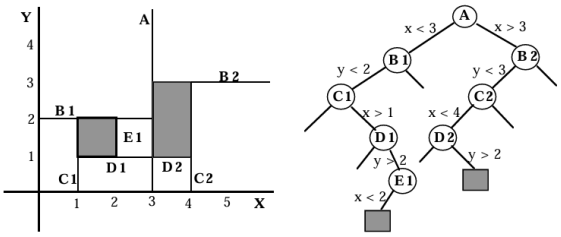


Gambar 4. Pohon BSP Membagi Wilayah Geometri, diambil dari [3]

Pohon BSP memberikan Solusi komputasi geometri dengan memanfaatkan dua sifat yang sangat sederhana yang terjadi setiap kali suatu bidang memisahkan (atau terletak di antara) dua objek atau lebih:

1) Semua objek di satu sisi bidang tidak memotong objek apa pun di sisi lain,
2) Dengan adanya suatu sudut pandang (penonton), objek di sisi yang sama dengan penonton akan digambar di atas gambar objek di sisi yang berlawanan (Painter's Algorithm).

Sifat kedua adalah salah satu penyebab penggunaan BSP sempat sangat popular karena pembuatan *video game* DOOM. Melakukan *rendering* dengan Painter's Algorithm sangat lambat dan akan menggambar banyak gambar yang tertutup oleh gambar lain yang lebih dekat dengan penonton. Penggunaan BSP di gim DOOM memastikan hanya objek (dinding) yang paling dekat yang akan di-*render* sehingga tidak membuang-buang komputasi dan mengurangi beban pada komputer sehingga performa gim menjadi jauh lebih lancar.



Gambar 5. Pohon BSP Sebagai Pengembangan Binary Search Tree, diambil dari [3]

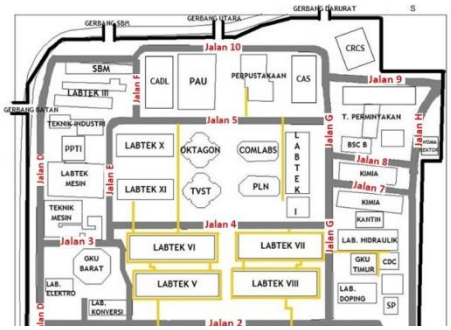
Pohon BST dapat berperan sebagai pengembangan dari pohon *binary search*. Pada gambar 5 dapat dilihat bahwa kitab isa menemukan bidang di sebelah kiri dengan melakukan pencarian secara traversal di pohon BSP dengan membandingkan koordinat dari posisi bidang dengan partisi

yang ada (A, B1, C1, dst.).

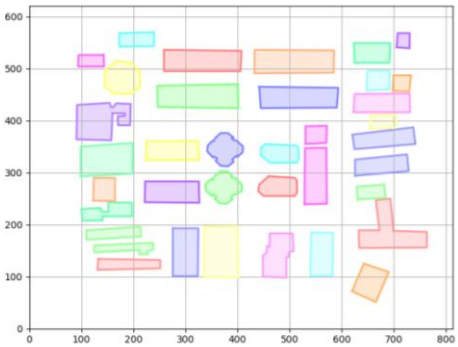
III. PEMBAHASAN

A. Perancangan

Dalam makalah ini, dilakukan eksperimen untuk membuat sebuah program berdasarkan pohon BSP. Penulis ingin mencoba membuat sebuah program yang bisa menentukan lokasi dari sebuah titik pada peta kampus ITB Ganesha. Misalkan sebuah titik (x,y) akan ditentukan apakah ia berada di luar atau di dalam sebuah gedung di ITB Ganesha, jika di dalam gedung maka akan disebutkan nama gedungnya. Gedung-gedung di ITB Ganesha akan dibuat menjadi referensi untuk garis-garis partisi pada BSP. Peta yang digunakan hanyalah potongan dari kira-kira setengah bagian ITB Ganesha karena keterbatasan waktu pada pembuatan makalan ini sehingga desain peta yang terlalu besar akan memakan waktu dalam pembuatannya. Berikut merupakan peta yang digunakan sebagai referensi dan hasil objek-objek geometeri yang dibentuk.



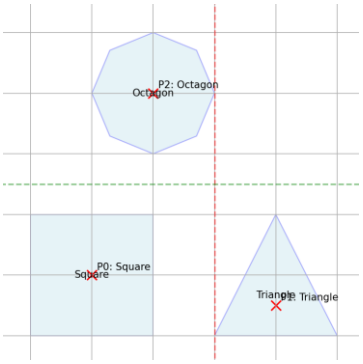
Gambar 6. Potongan Peta ITB Ganesha, diambil dari [4]



Gambar 7. Peta Geometri Gedung-Gedung di ITB Ganesha

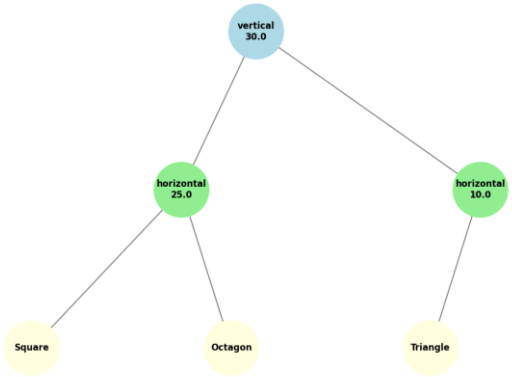
B. Pembuatan Program

Pada awalnya dirancang sebuah program BSP sederhana dengan OOP. Dibuat kelas-kelas yang merepresentasikan garis, polygon, dan pohon BSP. Kemudian digunakan bentuk-bentuk sederhana seperti segitiga dan persegi untuk prototipe. Partisi yang dilakukan juga hanya horizontal dan vertikal tanpa memedulikan sisi-sisi poligon.



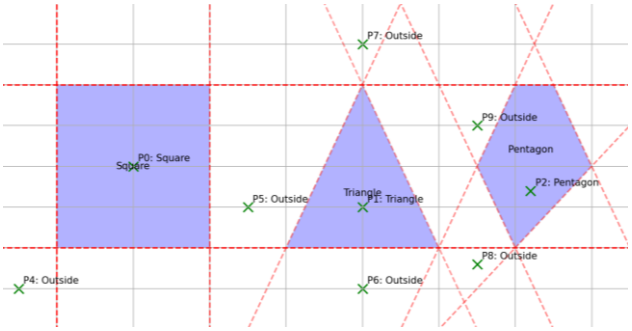
Gambar 8. Prototipe 1

Digunakan tiga titik uji yang terletak di dalam masing-masing poligon 2 dimensi dan didapatkan bahwa ketiganya mengembalikan nama yang sesuai. Pohon BSP yang dihasilkan adalah sebagai berikut.



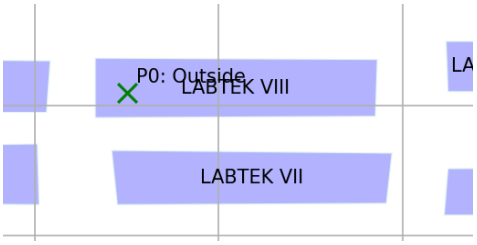
Gambar 9. Pohon BSP Prototipe 1

Untuk prototipe selanjutnya, garis partisi dibuat dari sisi-sisi poligon yang merepresentasikan gedung-gedung di ITB Ganesha. Hasilnya masih akurat dan titik-titik uji mengembalikan nama gedung yang sesuai.

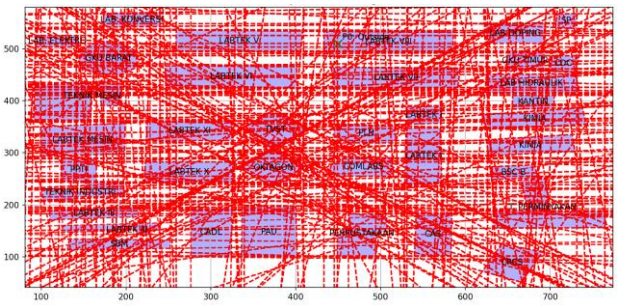


Gambar 10. Prototipe 2

Namun ketika menggunakan bentuk-bentuk yang lebih kompleks dan poligon (gedung) berjumlah banyak yaitu pada gambar 7, diperoleh hasil yang meleset. Titik-titik uji tidak mengembalikan nama gedung yang benar.



Gambar 11. Titik Uji Salah



Gambar 12. Garis-Garis Partisi Peta ITB Ganesha

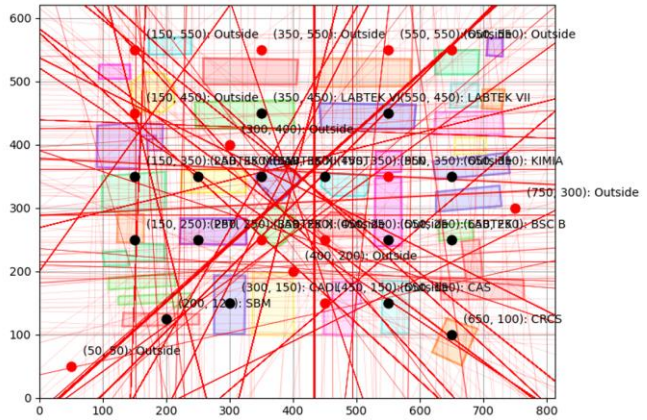
Penulis mencoba berbagai cara untuk mencoba men-*debugging* masalah yang terjadi. Namun apapun yang dilakukan hasilnya tetap tidak akurat. Seharusnya titik P0 menampilkan teks "LABTEK VIII" tetapi malah menampilkan "outside" yang berarti di luar gedung.

Cara kerja dari algoritma ini adalah dengan mengambil semua sisi dari seluruh gedung yang ada dan memperpanjang segmen garis tersebut menjadi garis dengan panjang tak hingga, dengan demikian bisa ditentukan dengan logika pencarian seperti gambar 5 dengan membandingkan nilai absis dan ordinat titik dnegan berbagai segmen hingga dicapai daun dari pohon BSP. Dapat dilihat bahwa algoritma ini bekerja untuk bentuk-bentuk yang sederhana seperti gambar 10 tetapi tidak untuk gambar yang rumit dengan banyak sekali garis partisi saling memotong satu sama lain dan garis partisi memotong bagian gedung-gedung lain.

Pohon BSP dari percobaan dengan peta ITB Ganesha tidak dapat ditampilkan karena ukurannya terlalu besar dan jumlah simpulnya terlalu banyak. Terdapat kemungkinan bahwa hal ini terjadi karena titik uji berada di sisi yang salah dari garis partisi akhir sehingga meskipun terletak di dalam gedung tetapi tidak terdeteksi oleh program.

Penulis kemudian mencoba untuk mengubah cara kerja dari algoritma pencarian pada pohon BSP yang terbentuk. Awalnya dilakukan pencarian dengan membandingkan lokasi titik uji. Pada perbandingan pertama dengan garis partisi pertama, jika titik berada di sebelah kiri garis (relatif, dihitung dengan *cross product* dari vektor) maka akan traversal ke upagraf sebelah kiri melalui anak kiri alih-alhir ke upagraf kanan. Traversal ini dilakukan hingga tercapai daun yang berisikan gedung yang dituju. Algoritma ini kemudian diubah, kita menggunakan algoritma tambahan untuk mengecek apakah sebuah titik berada di dalam sebuah gedung. Operasi pengecekan ini dilakukan dengan menghitung jumlah perpotongan antara garis yang ditarik dari titik dengan sisi poligon gedung, jika jumlahnya genap berarti titik berada di dalam gedung, sedangkan jika ganjil berarti titik tidak berada di dalam gedung. Pengecekan ini dilakukan dengan semua sisi pada bangunan yang didapatkan dari pencarian traversal pohon BSP hingga sampai di daun. Setelah implementasi algoritma ini, mungkin sekilas terlintas pemikiran bahwa BSP yang diimplementasikan menjadi tidak berguna, hal ini tentu tidak benar karena algoritma BSP tetap berguna untk menimalkan jumlah gedung yang perlu dicek dengan algoritma tersebut. Tanpa BSP, algoritma ini harus dilakukan untuk semua gedung untuk menemukan apakah titik berada di dalam sebuah gedung. Namun, dengan BSP, kita bisa mengetahui gedung-gedung mana saja yang perlu dicek dengan algoritma tersebut sehingga akan mengurangi waktu pencarian secara signifikan. Algoritma untuk membangun BSP juga tidak perlu dilakukan secara terus menerus setiap pencarian karena pohon BSP bisa disimpan di database sehingga kita hanya perlu melakukan pencarian selama tidak ada gedung baru yang ditambahkan ke dalam peta.

Berikut merupakan hasil dari pengujian dengan algoritma terbaru. Dapat dilihat bahwa semua titik uji sesuai.



Gambar 13. Program Final

Dapat dilihat pada gambar 13 bahwa ada banyak sekali garis-garis partisi yang digunakan. Hasil titik uji yang didapatkan akurat karena menggunakan algoritma pengecekan dengan menghitung banyak perpotongan dari garis yang ditarik dari titik uji ke sisi gedung. Maka jika kita ingin menggunakan BSP pada situasi seperti ini di mana kita memiliki peta dan banyak jumlah gedung, maka perlu digunakan algoritma tambahan selain BSP. Inilah yang menjadi alasan mengapa BSP jarang diimplementasikan pada bidang-bidang sejenis.

C. Kode Program

Berikut merupakan kode untuk algoritma yang menghitung banyak perpotongan antara garis yang ditarik dari titik menuju sisi gedung.

```
    def is_point_inside_building(self, point, building):
        cross = 0
        n = len(building.vertices)

        for i in range(n):
            j = (i + 1) % n
            vi = building.vertices[i]
            vj = building.vertices[j]
```

```
            if ((vi.y > point.y) != (vj.y > point.y)):
                if vi.y == vj.y:
                    if point.x < max(vi.x, vj.x):
                        cross += 1
                else:
                    x_intersect = (vj.x - vi.x) * (point.y - vi.y) / (vj.y - vi.y) + vi.x
                    if point.x < x_intersect:
                        cross += 1

        return cross % 2 == 1
```

Berikut merupakan algoritma pencarian dari pohon BSP hingga ditemukan daun yang mengandung gedung.

```
def find_building_containing_point(self, point):
        def traverse(node, point):
            if not node:
                return None

            if not node.partition or node.buildings:
                for building in node.buildings:
                    if self._is_point_inside_building(point, building):
                        return building.name

            if not node.partition:
                return None

            side = node.partition.get_point_side(point)

            if side >= -self.epsilon:
                result = traverse(node.front, point)
                if result:
                    return result

            if side <= self.epsilon:
                result = traverse(node.back, point)
                if result:
                    return result

            return None

        result = _traverse(self.root, point)
        return result if result else "Outside"
```

Kemudian berikut adalah program untuk membangung pohon BSP dengan rekursif.

```
def build_node(self, edges, buildings):
        if not edges:
            return BSPNode(buildings=buildings)

        partition, partition_building = self.select_best_partition(edges, buildings)
        node = BSPNode(partition=partition, buildings=buildings)

        front_buildings = []
        back_buildings = []
        front_edges = []
        back_edges = []

        for edge, building in edges:
            if edge == partition:
                continue

            start_side = partition.get_point_side(edge.start)
            end_side = partition.get_point_side(edge.end)

            center = Vector2(
                sum(v.x for v in building.vertices) / len(building.vertices),
                sum(v.y for v in building.vertices) / len(building.vertices)
            )
            center_side = partition.get_point_side(center)

            if center_side >= 0:
                if building not in front_buildings:
                    front_buildings.append(building)
                front_edges.append((edge, building))
                if start_side * end_side < 0:
                    back_edges.append((edge, building))
            else:
                if building not in back_buildings:
```

```
                    back_buildings.append(build
ing)
                back_edges.append((edge,
building))
                if start_side * end_side < 0:
                    front_edges.append((edge,
building))

        if front_edges:
            node.front =
self._build_node(front_edges, front_buildings)
        if back_edges:
            node.back =
self._build_node(back_edges, back_buildings)

        return node
```

IV. Kesimpulan

Pohon BSP dapat digunakan untuk membuat program penentuan lokasi di Kampus ITB Ganesha dengan tambahan algoritma untuk pengecekan jumlah titik potong suatu garis yang ditarik dari titik uji/lokasi ke sisi gedung yang didapatkan dari pencarian traversal pohon BSP hingga ditemukan daun yang berisi gedung.

VII. Ucapan Terima Kasih

Penulis mengucapkan puji syukur kepada Tuhan Yang Maha Esa atas rahmat dan karunia-Nya sehingga makalah berjudul “Aplikasi Pohon dalam Binary Space Partitioning dalam Penentuan Lokasi di ITB Ganesha ini dapat selesai tepat waktu. Penulis juga ingin mengucapkan terima kasih kepada orang tua dan teman-teman yang selalu mendukung dan memberikan semangat terutama secara mental sehingga penulis mampu terus menyusun makalah ini. Penulis juga ingin berterima kasih kepada Dr. Rinaldi Munir selaku dosen pengajar Matematika Diskrit K1 2024/2025 yang telah membagikan ilmunya dan membimbing penulis dalam KBM. Penulis juga mengucapkan terima kasih kepada Dr. Rila Mandala sebagai salah satu dosen pengampu mata kuliah Matematika Diskrit. Terakhir, penulis ingin mengucapkan terima kasih kepada pihak-pihak lain yang telah membantu dalam penyusunan makalah ini.

Referensi

[1] Mark Allen Weiss. Data Structures and Algorithm Analysis in Java 3rd Edition. Florida International University.

[2] Kenneth H. Rosen, Discrete Mathematics and Application to Computer Science 7th Edition, Mc Graw-Hill.

[3] A Tutorial on Binary Space Partitioning Trees Bruce F. Naylor Spatial Labs Inc. [Daring], diakses dari https://cgvr.cs.uni-bremen.de/teaching/geom_literatur/A%20Tutorial%20on%20Binary%20Space%20Partitioning%20Trees%20-%20Bruce%20F.%20Naylor.pdf pada 6 Januari 2025.

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, 8 Januari 2025

Fajar Kurniawan - 13523027