PERANCANGAN & ANALISIS ALGORITMA

TUGAS AKHIR

Disusun Oleh :

Nama	: Dzunnurain Arif Malika
NIM	: 21346005
Prodi	: Informatika (NK)
Dosen Pengampu	: Widya Darwin, S.Pd., M.Pd.T

PROGRAM STUDI INFORMATIKA

JURUSAN TEKNIK ELEKTRONIKA

FAKUTAS TEKNIK

UNIVERSITAS NEGERI PADANG

2023

KATA PENGANTAR

Rasa syukur kita hanya milik Allah SWT atas segala semua rahmatnya, sehingga saya dapat menyelesaikan tugas akhir yang saya susun ini. Meskipun banyak rintangan dan hambatan yang saya alami dalam proses pekerjaan tetapi saya berhasil mengerjakan dengan baik dan tetap pada waktunya.

Dan harapan saya di sini semoga atas tugas akhir yang saya buat ini bisa menambah pengetahuan dan pengalaman bagi para pembaca, dan untuk kedepan nya kiat juga sama-sama memperbaiki bentuk atau menambah isi dari makalah agar semua akan lebih baik dengan sebelumnya.

Saya mengucapkan terima kasih kepada Ibu Widya Darwin, S.Pd., M.Pd.T sebagai Dosen Pengampu pada mata kuliah Perancangan & Analisis Algoritma yang telah membimbing saya dalam menyelesaikan tugas akhir yang saya buat ini.

Karena dari semua keterbatasan dari pengetahuan atau pun pengalaman saya, saya yakin masih banyak sekali dari kekurangan yang terdapat pada makalah ini. Oleh karena itu saya sangat berharap untuk saran dan kritik yang bisa membangun dari pembaca demi semua tugas akhir ini akan terselesaikan dengan benar.

Sago, Juni 2023

Dzunnuain Arif Malika

DAFTAR ISI

KATA PENGANTAR i
DAFTAR ISI ii
BAB I 1
ANALISIS ALGORITMA 1
A. Pengertian 1
B. Langkah – Langkah dalam Melakukan Analisis Algoritma 1
C. Dasar – dasar Algortima 2
D. Problem Solving 4
BAB II 5
ANALISIS EFISIENSI ALGORITMA 5
A. Measuring an Input's Size 5
B. Unit for Measuring Running Time 5
C. Order of Growth 6
D. Worst-Case, Best-Case, and Average-Case Efficiency 7
BAB III 8
BRUTE-FORCE EXHAUSTIVE SEARCH 8
A. Selection Sort and Bubble Sort 8
B. Sequential Search and Brute –Force String Maching 8
C. Closest -Pair and Convex -Hull Problems 9
D. Exhaustive Search 9
E. Depth-First Search and Breath -First Search 9
BAB IV 11
DECREASE & CONQUER 11
A. Three Major Varian of Decrease-andConquer 11
B. Sort 11
C. Topological Sorting 12
BAB V 14
DEVIED & CONQUER 14
A. Mergesort 14
B. Quicksort 14
C. Binary Tree Traversals and Related Properties 15
BAB VI 17
TRANSFORM & CONQUER 17

A. Instance Simplification 17
B. Representation Change 17
C. Problem Reduction 18
BAB VII 19
SPACE & TIME TRADE -OFFS 19
A. Sorting by Counting 19
B. Input Enhancement in String Matching 19
C. Hasing 20
BAB VIII 21
DYNAMIC PROGRAMMING 21
A. Three Basic 21
B. The Knapsack Problem and Memory Fungctions 22
C. Warshall's and Floyd's Algorithms 22
BAB IX 23
GREEDY TECHNIQUE 23
A. Prims's Algorithm 23
B. Kruskal's Algorithm 23
C. Djikstra's Algorithm 24
D. Huffman Tress and Codes 24
BAB X 26
ITERATIVE IMPROVEMENT 26
A. The Simplex Method 26
B. The maximum-Flow Problem 26
C. Maximum Matching in Bipartite Graphs 27
D. The Stable Marriage Problem 28
BAB XI 29
LIMITATIONS OF ALGORITHM POWER 29
A. Lower-Bounf Arguments 29
B. Decision Tree 29
C. P, NP and-Complete Problems 30
D. Challenge of Numerical Algorithms 31
BAB XII 32
COPING WITH THE LIMITATIONS OF ALGORITHM POWER 32
A. Backtracking 32
B. Branch and Bound 32

C. Algorithms for Solving Nonlinear Problems 33
DAFTAR PUSTAKA 34

BAB I
ANALISIS ALGORITMA

A. Pengertian

Analisis algoritma adalah proses mempelajari kinerja suatu algoritma dengan memperhatikan faktor-faktor seperti kecepatan (waktu eksekusi), penggunaan memori, dan keakuratan (ketepatan) dalam menghasilkan output yang diinginkan.

B. Langkah – Langkah dalam Melakukan Analisis Algoritma

1. Definisikan masalah: Langkah pertama dalam melakukan analisis algoritma adalah memahami masalah yang ingin dipecahkan. Definisikan dengan jelas input yang diberikan dan output yang diharapkan.

2. Pilih algoritma yang sesuai: Setelah memahami masalah, pilih algoritma yang paling sesuai untuk menyelesaikan masalah tersebut. Berbagai algoritma memiliki kelebihan dan kelemahan dalam hal kecepatan dan ketepatan, jadi pilihlah algoritma yang paling cocok untuk tujuan Anda.

3. Identifikasi notasi waktu (time complexity): Notasi waktu menggambarkan berapa banyak waktu yang dibutuhkan oleh algoritma untuk menyelesaikan tugasnya sehubungan dengan ukuran input yang diberikan. Identifikasi tingkat pertumbuhan waktu algoritma Anda dengan menggunakan notasi waktu, seperti O(n), O(n^2), O(log n), dan sebagainya.

4. Identifikasi notasi ruang (space complexity): Notasi ruang menggambarkan berapa banyak ruang memori yang dibutuhkan oleh algoritma dalam menjalankan tugasnya sehubungan dengan ukuran input yang diberikan. Identifikasi tingkat pertumbuhan ruang algoritma Anda dengan menggunakan notasi ruang, seperti O(1), O(n), O(n^2), dan sebagainya.

5. Analisis langkah-langkah algoritma: Setelah mengidentifikasi notasi waktu dan ruang algoritma, analisis langkah-langkah algoritma secara rinci. Identifikasi loop (perulangan), kondisional (percabangan), dan operasi-operasi utama lainnya dalam algoritma. Tentukan berapa banyak operasi

yang dilakukan pada setiap langkah dan seberapa sering langkah-langkah tersebut diulang dalam kaitannya dengan ukuran input.

6. Hitung kompleksitas: Gunakan informasi yang diperoleh dari analisis langkah-langkah algoritma untuk menghitung kompleksitas algoritma secara keseluruhan. Misalnya, jika algoritma memiliki loop yang diulang sebanyak n kali dan dalam setiap perulangan melakukan m operasi, kompleksitas waktu algoritma bisa diwakili oleh O(n * m).

7. Uji dengan contoh input: Untuk memeriksa keakuratan (ketepatan) algoritma, uji algoritma dengan berbagai contoh input. Pastikan algoritma menghasilkan output yang benar dan sesuai dengan masalah yang ingin diselesaikan.

8. Bandingkan dengan algoritma lain: Terakhir, bandingkan algoritma yang Anda gunakan dengan algoritma-algoritma lain yang tersedia untuk menyelesaikan masalah yang sama. Evaluasi kecepatan dan ketepatan (keakuratan) algoritma Anda dalam perbandingan dengan algoritma-algoritma lain tersebut.

C. Dasar – dasar Algortima

Dasar-dasar algoritma adalah konsep dan prinsip-prinsip dasar yang membentuk landasan dalam merancang dan menganalisis algoritma. Berikut adalah beberapa dasar-dasar algoritma yang penting untuk dipahami:

1. Input dan Output: Algoritma menerima input tertentu, yang dapat berupa data, informasi, atau masalah yang ingin diselesaikan. Algoritma kemudian memproses input tersebut dan menghasilkan output yang diinginkan, yang bisa berupa solusi, hasil komputasi, atau informasi yang berguna.

2. Langkah-langkah: Algoritma terdiri dari serangkaian langkah-langkah yang ditentukan secara terperinci. Setiap langkah dalam algoritma harus jelas, terdefinisi dengan baik, dan dapat dilaksanakan secara berurutan.

3. Kontrol aliran: Algoritma mengatur aliran eksekusi dengan menggunakan struktur kontrol seperti percabangan (if-else, switch-case) dan perulangan

(loop). Ini memungkinkan algoritma untuk mengambil keputusan berdasarkan kondisi tertentu atau mengulang langkah-langkah tertentu sesuai dengan kebutuhan.

4. Variabel dan Operasi: Algoritma menggunakan variabel untuk menyimpan data atau informasi yang sedang diproses. Variabel dapat diubah nilainya selama eksekusi algoritma. Selain itu, algoritma melibatkan operasi matematika dan operasi lainnya, seperti pengulangan, pembandingan, dan manipulasi data.

5. Pengulangan: Pengulangan memungkinkan algoritma untuk melakukan serangkaian langkah secara berulang sesuai dengan kondisi atau jumlah tertentu. Ini memungkinkan algoritma untuk menangani kasus-kasus yang berulang atau mengolah sejumlah data dalam bentuk yang serupa.

6. Penggunaan Struktur Data: Algoritma seringkali menggunakan struktur data seperti array, daftar, tumpukan, antrian, dan pohon untuk menyimpan dan mengelola data dengan efisien. Pemilihan struktur data yang tepat dapat mempengaruhi kinerja dan efisiensi algoritma.

7. Analisis Kinerja: Algoritma harus dianalisis untuk memahami kinerjanya, seperti waktu eksekusi, penggunaan memori, dan efisiensi. Analisis kinerja membantu dalam memilih algoritma yang paling efisien dan memperbaiki algoritma yang ada untuk meningkatkan kinerjanya.

8. Rekursi: Rekursi adalah teknik di mana algoritma memanggil dirinya sendiri untuk menyelesaikan masalah yang lebih kecil atau serupa. Rekursi sangat berguna dalam pemecahan masalah yang membutuhkan pendekatan berulang atau pemecahan secara terbagi-bagi.

9. Abstraksi dan Modularitas: Algoritma dapat diorganisasi menjadi bagian-bagian yang lebih kecil dan terpisah, yang disebut modul. Modul-modul ini dapat mengabstraksi operasi atau tugas tertentu dalam algoritma, membuat algoritma lebih mudah dipahami, diimplementasikan, dan dikelola.

10. Kepastian dan Keakuratan: Algoritma harus menghasilkan hasil yang benar dan konsisten sesuai dengan tujuan yang ditetapkan. Keakuratan adalah faktor penting dalam merancang algoritma yang dapat diandalkan dan diuji secara matematis.

11. Memahami dasar-dasar ini akan membantu dalam merancang, menganalisis, dan mengimplementasikan algoritma dengan lebih baik serta memahami prinsip-prinsip yang melandasi komputasi dan pemrograman.

D. Problem Solving

Problem solving adalah proses mengidentifikasi, menganalisis, dan menyelesaikan masalah dengan menggunakan pendekatan logis dan sistematis. Dalam konteks komputasi dan pemrograman, problem solving melibatkan merancang algoritma yang efektif dan efisien untuk menyelesaikan masalah yang diberikan.

BAB II
ANALISIS EFISIENSI ALGORITMA

A. Measuring an Input's Size

Untuk menganalisis dan membandingkan kinerja algoritma, penting untuk mengukur ukuran input yang diberikan kepada algoritma. Ukuran input dapat diukur dalam berbagai cara, tergantung pada jenis masalah yang sedang diselesaikan. Beberapa metode umum untuk mengukur ukuran input antara lain:

1. Jumlah elemen: Jumlah elemen dalam struktur data seperti array, daftar, atau himpunan.
2. Jumlah bit atau byte: Jumlah bit atau byte yang dibutuhkan untuk menyimpan data input.
3. Panjang string: Panjang karakter dalam string yang diberikan.
4. Tingkat kedalaman: Tingkat kedalaman pohon atau struktur data terkait.

Dalam analisis algoritma, ukuran input sering digunakan untuk memperkirakan kompleksitas waktu atau ruang algoritma. Dengan mengetahui ukuran input, kita dapat mengidentifikasi bagaimana algoritma tumbuh sehubungan dengan ukuran input tersebut.

B. Unit for Measuring Running Time

Dalam mengukur kinerja algoritma, waktu eksekusi sering digunakan sebagai metrik untuk mengukur efisiensi algoritma. Ada beberapa satuan umum yang digunakan untuk mengukur waktu eksekusi algoritma:

1. Detik (s): Satuan waktu paling umum untuk mengukur waktu eksekusi algoritma. Misalnya, jika algoritma membutuhkan 5 detik untuk menyelesaikan tugas, itu berarti algoritma memakan waktu 5 detik untuk mengeksekusi.

2. Milidetik (ms): Satuan waktu yang lebih kecil dari detik. Satu milidetik sama dengan 0,001 detik. Digunakan untuk mengukur waktu eksekusi algoritma yang lebih cepat.

3. Mikrodetik (μs): Satuan waktu yang lebih kecil dari milidetik. Satu mikrodetik sama dengan 0,000001 detik. Digunakan untuk mengukur waktu eksekusi algoritma yang sangat cepat atau operasi mikro.

4. Nanodetik (ns): Satuan waktu yang lebih kecil dari mikrodetik. Satu nanodetik sama dengan 0,000000001 detik. Digunakan untuk mengukur waktu eksekusi algoritma yang sangat cepat atau operasi nanoskala.

Pilihan satuan pengukuran waktu tergantung pada tingkat kehalusan yang diperlukan dan tingkat kecepatan eksekusi algoritma yang sedang dianalisis.

C. Order of Growth

Order of growth (orde pertumbuhan) adalah konsep yang digunakan dalam analisis algoritma untuk menggambarkan bagaimana waktu eksekusi atau penggunaan memori sebuah algoritma tumbuh seiring dengan ukuran input. Dalam analisis ini, orde pertumbuhan diwakili oleh notasi Big O (O).

Misalnya, jika suatu algoritma memiliki orde pertumbuhan O(n), itu berarti waktu eksekusinya tumbuh linier dengan ukuran input (n). Jika orde pertumbuhan adalah O(n^2), maka waktu eksekusi tumbuh kuadratik terhadap ukuran input.

Konsep orde pertumbuhan memungkinkan kita untuk memperkirakan seberapa efisien algoritma dalam menangani input yang semakin besar. Algoritma dengan orde pertumbuhan yang lebih rendah cenderung lebih efisien daripada algoritma dengan orde pertumbuhan yang lebih tinggi.

D. Worst-Case, Best-Case, and Average-Case Efficiency

Ketika menganalisis algoritma, penting untuk mempertimbangkan efisiensi dalam tiga skenario berbeda: kasus terburuk, kasus terbaik, dan kasus rata-rata.

1. Kasus terburuk (worst-case): Ini adalah skenario di mana algoritma membutuhkan waktu atau sumber daya maksimum untuk menyelesaikan tugas. Analisis kasus terburuk memberikan batasan atas untuk waktu eksekusi atau penggunaan memori algoritma dalam kondisi terburuk.

2. Kasus terbaik (best-case): Ini adalah skenario di mana algoritma membutuhkan waktu atau sumber daya minimum untuk menyelesaikan tugas. Analisis kasus terbaik memberikan batasan bawah untuk waktu eksekusi atau penggunaan memori algoritma dalam kondisi terbaik.

3. Kasus rata-rata (average-case): Ini adalah skenario yang mencerminkan kinerja algoritma secara umum dengan mempertimbangkan semua kemungkinan input dengan probabilitas tertentu. Analisis kasus rata-rata memberikan perkiraan rata-rata untuk waktu eksekusi atau penggunaan memori algoritma dalam kondisi umum.

Dengan mempertimbangkan efisiensi dalam ketiga skenario ini, kita dapat memperoleh pemahaman yang lebih komprehensif tentang kinerja algoritma dalam berbagai situasi.

BAB III
BRUTE-FORCE EXHAUSTIVE SEARCH

A. Selection Sort and Bubble Sort

Selection Sort: Selection Sort adalah algoritma sorting sederhana yang bekerja dengan memilih elemen terkecil dari daftar dan menukar posisinya dengan elemen pertama. Kemudian, elemen terkecil kedua dipilih dari sisa daftar dan ditukar dengan elemen kedua, dan seterusnya. Proses ini diulang hingga seluruh daftar terurut. Meskipun sederhana, Selection Sort memiliki kompleksitas waktu O(n^2).

Bubble Sort: Bubble Sort juga merupakan algoritma sorting sederhana yang berulang kali membandingkan pasangan elemen bersebelahan dan menukar posisi jika diperlukan. Selama setiap iterasi, elemen terbesar akan "mengapung" ke posisi akhir seperti gelembung. Proses ini diulang hingga seluruh daftar terurut. Bubble Sort juga memiliki kompleksitas waktu O(n^2) dan sering digunakan untuk mengilustrasikan konsep dasar dalam algoritma sorting.

B. Sequential Search and Brute –Force String Maching

Sequential Search: Sequential Search, juga dikenal sebagai linear search, adalah metode pencarian sederhana yang memeriksa setiap elemen dalam urutan tertentu untuk menemukan elemen yang dicari. Dimulai dari elemen pertama, pencarian berlanjut hingga elemen ditemukan atau mencapai akhir daftar. Jika elemen ditemukan, pencarian berhenti. Sequential Search memiliki kompleksitas waktu O(n), di mana n adalah jumlah elemen dalam daftar.

Brute-Force String Matching: Brute-Force String Matching adalah algoritma pencocokan string sederhana yang mencari pola dalam teks dengan membandingkan karakter per karakter. Algoritma ini mencoba memadankan pola dengan teks, dan jika ada ketidakcocokan pada suatu posisi, algoritma maju ke posisi berikutnya dan mencoba lagi. Proses ini diulang hingga pola ditemukan atau mencapai akhir teks. Brute-Force String Matching memiliki

kompleksitas waktu O(mn), di mana m adalah panjang pola dan n adalah panjang teks.

C. Closest -Pair and Convex -Hull Problems

Closest-Pair Problem: Closest-Pair Problem melibatkan mencari pasangan titik terdekat dalam himpunan titik. Tujuannya adalah menemukan dua titik dalam himpunan yang memiliki jarak terkecil di antara mereka. Untuk menyelesaikan masalah ini, diperlukan algoritma yang efisien dalam mencari jarak antara setiap pasangan titik dalam himpunan. Salah satu algoritma yang digunakan adalah Divide and Conquer dengan kompleksitas waktu O(n log n).

Convex-Hull Problem: Convex-Hull Problem melibatkan pencarian cangkang cembung terkecil yang meliputi seluruh himpunan titik. Cangkang cembung adalah poligon tertutup yang terbentuk oleh himpunan titik, di mana semua titik dalam himpunan berada di dalam atau pada batas poligon. Algoritma yang umum digunakan untuk menyelesaikan masalah ini adalah Graham's Scan atau Jarvis's March dengan kompleksitas waktu O(n log n).

D. Exhaustive Search

Exhaustive Search, juga dikenal sebagai Brute-Force Search, adalah metode pencarian yang mencoba semua kemungkinan solusi secara sistematis untuk menemukan solusi optimal. Algoritma ini mencoba semua kombinasi atau permutasi yang mungkin dari elemen-elemen masalah untuk menemukan solusi yang memenuhi kriteria yang ditentukan. Exhaustive Search memeriksa setiap kemungkinan secara eksplisit dan biasanya digunakan ketika ukuran masalah relatif kecil.

E. Depth-First Search and Breath -First Search

Depth-First Search (DFS): DFS adalah algoritma pencarian graf yang menggunakan pendekatan berbasis tumpukan (stack-based). Pencarian dimulai dari simpul awal, kemudian berlanjut ke simpul tetangga yang belum dikunjungi secara rekursif. Jika tidak ada simpul tetangga yang tersisa, algoritma mundur ke simpul sebelumnya dan melanjutkan pencarian dari simpul tetangga berikutnya yang belum dikunjungi. DFS digunakan untuk

menjelajahi atau mencari jalur melalui struktur data seperti graf, pohon, atau grafik terarah.

Breadth-First Search (BFS): BFS adalah algoritma pencarian graf yang menggunakan pendekatan berbasis antrian (queue-based). Pencarian dimulai dari simpul awal, kemudian berlanjut ke semua simpul tetangga yang belum dikunjungi secara sejajar. Setelah semua simpul tetangga telah dikunjungi, algoritma melanjutkan ke simpul tetangga dari simpul tetangga sebelumnya. BFS digunakan untuk menjelajahi atau mencari jalur melalui struktur data seperti graf, pohon, atau grafik terarah, dan sering digunakan untuk mencari jalur terpendek dalam graf yang tidak memiliki bobot**.**

BAB IV
DECREASE & CONQUER

A. Three Major Varian of Decrease-andConquer

Decrease-and-Conquer adalah paradigma desain algoritma yang melibatkan pemecahan masalah dengan mengurangi ukuran masalah pada setiap langkah iterasi. Ada tiga varian utama dari Decrease-and-Conquer:

1. Decrease by a Constant Amount: Pada varian ini, ukuran masalah dikurangi dengan jumlah konstan pada setiap langkah iterasi. Misalnya, dalam pencarian linier, ukuran masalah dikurangi satu dengan memeriksa satu elemen pada setiap langkah.

2. Decrease by a Fraction: Pada varian ini, ukuran masalah dikurangi dengan fraksi tetap pada setiap langkah iterasi. Misalnya, dalam pencarian biner, ukuran masalah dikurangi menjadi setengahnya pada setiap langkah dengan membagi array menjadi dua bagian dan mencari di bagian yang relevan.

3. Decrease by Variable Amount: Pada varian ini, ukuran masalah dikurangi dengan jumlah variabel pada setiap langkah iterasi. Misalnya, dalam algoritma Quicksort, ukuran masalah dikurangi berdasarkan partisi elemen-elemen yang membagi array menjadi subset yang lebih kecil untuk diproses lebih lanjut.

B. Sort

Sort (pengurutan) adalah proses mengatur elemen-elemen dalam suatu urutan tertentu, umumnya dari urutan yang tidak teratur atau acak menjadi urutan yang teratur atau berurutan. Pengurutan digunakan dalam berbagai aplikasi, seperti pengolahan data, pemrosesan daftar, database, dan banyak lagi. Ada berbagai algoritma pengurutan yang berbeda, termasuk:

1. Bubble Sort: Algoritma Bubble Sort membandingkan pasangan elemen bersebelahan dan menukar posisi jika diperlukan. Elemen-elemen yang lebih besar "mengapung" ke posisi yang tepat selama iterasi.

2. Selection Sort: Algoritma Selection Sort memilih elemen terkecil dari daftar dan menukar posisinya dengan elemen pertama. Proses ini diulang dengan memilih elemen terkecil berikutnya dari sisa daftar dan menukar posisinya.

3. Insertion Sort: Algoritma Insertion Sort membangun daftar yang terurut satu elemen pada satu waktu dengan membandingkan setiap elemen dengan elemen-elemen sebelumnya dan memasukkannya ke posisi yang tepat.

4. Merge Sort: Algoritma Merge Sort menggunakan pendekatan Divide and Conquer dengan membagi daftar menjadi bagian-bagian yang lebih kecil, mengurutkan setiap bagian secara terpisah, dan menggabungkannya kembali menjadi daftar yang terurut.

5. Quick Sort: Algoritma Quick Sort juga menggunakan pendekatan Divide and Conquer dengan memilih elemen pivot, mempartisi daftar berdasarkan elemen pivot, dan mengurutkan setiap bagian secara terpisah.

Ada banyak lagi algoritma pengurutan yang tersedia, dan pilihan algoritma yang tepat tergantung pada kebutuhan spesifik dan sifat data yang diurutkan.

C. Topological Sorting

Topological Sorting adalah proses pengurutan simpul-simpul dalam graf berarah acyclic (DAG) sedemikian rupa sehingga setiap simpul terurut sebelum simpul yang dituju oleh tepat satu busur. Topological Sorting banyak digunakan dalam analisis jaringan, penjadwalan tugas, kompilasi, dan masalah lain yang melibatkan ketergantungan antara elemen-elemen yang terkait.

Topological Sorting dapat dilakukan menggunakan algoritma seperti:

1. Depth-First Search (DFS): DFS dapat digunakan untuk melakukan Topological Sorting dengan memanfaatkan urutan finish time pada penjelajahan graf menggunakan DFS.

2. Kahn's Algorithm: Kahn's Algorithm menggunakan pendekatan berbasis antrian (queue-based) untuk mencari simpul-simpul dengan derajat masuk (in-degree) nol, menghapus simpul tersebut, dan mengulangi proses hingga semua simpul terurut.

Topological Sorting adalah alat penting dalam analisis struktur data berarah dan membantu dalam menyelesaikan masalah yang melibatkan ketergantungan antara elemen-elemen yang terkait.

BAB V
DEVIED & CONQUER

A. Mergesort

Mergesort adalah algoritma pengurutan yang menggunakan pendekatan Divide and Conquer. Algoritma ini membagi daftar yang akan diurutkan menjadi dua bagian yang lebih kecil secara rekursif, mengurutkan masing-masing bagian secara terpisah, dan menggabungkan kembali bagian-bagian yang terurut menjadi daftar yang utuh.

Langkah-langkah dalam Mergesort adalah sebagai berikut:

1. Bagi daftar menjadi dua bagian sama besar.
2. Lakukan Mergesort rekursif pada kedua bagian tersebut.
3. Gabungkan kembali dua bagian yang terurut menjadi satu daftar yang terurut.

Keuntungan dari Mergesort adalah memiliki kompleksitas waktu terbaik dan terburuk yang stabil, yaitu O(n log n), di mana n adalah jumlah elemen dalam daftar. Mergesort juga efisien dalam mengurutkan daftar dengan ukuran besar dan cocok untuk data yang terdistribusi secara acak atau hampir terurut.

B. Quicksort

Quicksort adalah algoritma pengurutan yang juga menggunakan pendekatan Divide and Conquer. Algoritma ini memilih sebuah elemen pivot dari daftar, mempartisi daftar menjadi dua subdaftar berdasarkan elemen pivot, dan mengurutkan masing-masing subdaftar secara terpisah.

Langkah-langkah dalam Quicksort adalah sebagai berikut:

1. Pilih sebuah elemen pivot dari daftar.
2. Partisi daftar menjadi dua subdaftar, di mana semua elemen yang lebih kecil dari pivot berada di satu subdaftar, dan semua elemen yang lebih besar dari pivot berada di subdaftar lainnya.
3. Lakukan Quicksort rekursif pada kedua subdaftar tersebut.
4. Gabungkan kembali subdaftar yang terurut menjadi satu daftar yang utuh.

Keuntungan dari Quicksort adalah memiliki kompleksitas waktu yang rata-rata sangat baik, yaitu O(n log n). Namun, dalam kasus terburuk, yaitu ketika pivot dipilih secara suboptimal, Quicksort dapat memiliki kompleksitas waktu O(n^2). Namun, dengan pemilihan pivot yang cerdas, seperti pivot acak atau pivot yang berada di tengah daftar, Quicksort dapat memberikan kinerja yang baik pada sebagian besar kasus.

C. Binary Tree Traversals and Related Properties

Binary Tree Traversals merujuk pada proses mengunjungi atau memeriksa semua simpul dalam pohon biner. Ada tiga metode utama untuk melakukan traversal pada pohon biner:

1. Inorder Traversal: Inorder Traversal mengunjungi simpul kiri, kemudian simpul itu sendiri, dan akhirnya simpul kanan. Secara rekursif, langkah-langkahnya adalah: (1) Traversing simpul kiri secara rekursif, (2) Mengunjungi simpul saat ini, (3) Traversing simpul kanan secara rekursif.

2. Preorder Traversal: Preorder Traversal mengunjungi simpul saat ini terlebih dahulu, kemudian simpul kiri, dan akhirnya simpul kanan. Secara rekursif, langkah-langkahnya adalah: (1) Mengunjungi simpul saat ini, (2) Traversing simpul kiri secara rekursif, (3) Traversing simpul kanan secara rekursif.

3. Postorder Traversal: Postorder Traversal mengunjungi simpul kiri, kemudian simpul kanan, dan akhirnya simpul saat ini. Secara rekursif, langkah-langkahnya adalah: (1) Traversing simpul kiri secara rekursif, (2) Traversing simpul kanan secara rekursif, (3) Mengunjungi simpul saat ini.

Sifat-sifat terkait yang sering dikaitkan dengan traversal pohon biner adalah tinggi (height) dan kedalaman (depth). Tinggi pohon biner adalah jumlah maksimum tingkat simpul dalam pohon, sedangkan kedalaman simpul adalah jumlah tingkat simpul dari akar ke simpul tertentu. Traversal pohon biner berguna dalam pemrosesan dan analisis struktur data pohon biner, serta dalam implementasi algoritma dan manipulasi data yang berhubungan dengan pohon biner.

BAB VI
TRANSFORM & CONQUER

A. Instance Simplification

Instance Simplification (vereinfachung) adalah metode dalam analisis algoritma yang melibatkan penyederhanaan atau transformasi masalah yang kompleks menjadi masalah yang lebih sederhana atau lebih terkelola. Tujuan dari Instance Simplification adalah mengurangi kompleksitas masalah atau mengubah masalah menjadi bentuk yang lebih mudah dipahami atau diatasi.

Dalam Instance Simplification, langkah-langkah umum yang dapat diambil termasuk:

1. Menghilangkan detail yang tidak relevan atau kompleks dari masalah.
2. Menggantikan masalah dengan kasus khusus yang lebih sederhana.
3. Memperkecil ukuran masalah dengan mengurangi jumlah variabel atau parameter yang terlibat.
4. Mengubah representasi masalah menjadi bentuk yang lebih mudah dipahami atau diproses.

Instance Simplification dapat membantu dalam menganalisis dan memecahkan masalah yang sulit atau kompleks dengan mengurangi kompleksitasnya menjadi masalah yang lebih terkelola dan dapat ditangani dengan algoritma yang lebih sederhana atau efisien.

B. Representation Change

Representation Change (perubahan representasi) adalah proses mengubah cara representasi data atau informasi yang terkait dengan masalah tertentu. Representasi data yang digunakan dapat mempengaruhi kompleksitas algoritma dan efisiensi pemrosesan masalah.

Dalam Representation Change, langkah-langkah umum yang dapat diambil termasuk:

1. Mengubah struktur data yang digunakan untuk menyimpan dan mengelola informasi.
2. Mengubah cara data diorganisir atau diurutkan.
3. Mengubah format data atau representasi numerik.
4. Mengubah representasi grafis atau visual dari informasi.

Perubahan representasi dapat membantu meningkatkan efisiensi algoritma, mengurangi kompleksitas, atau mempermudah pemahaman dan manipulasi data dalam konteks masalah yang diberikan.

C. Problem Reduction

Problem Reduction (reduksi masalah) adalah teknik dalam analisis algoritma yang melibatkan pengurangan suatu masalah tertentu menjadi bentuk yang lebih mudah atau lebih terkelola dengan mengaitkannya dengan masalah lain yang sudah dikenal atau lebih sederhana. Dalam Problem Reduction, masalah asli dikurangi menjadi masalah yang lebih sederhana yang dapat dipecahkan menggunakan algoritma atau metode yang sudah ada.

Langkah-langkah umum dalam Problem Reduction meliputi:

1. Mengidentifikasi masalah yang sulit atau kompleks.
2. Mengaitkan masalah tersebut dengan masalah lain yang sudah diketahui solusinya.
3. Mengurangi masalah asli menjadi bentuk yang setara dengan masalah yang sudah dikenal.
4. Menggunakan algoritma atau metode yang ada untuk memecahkan masalah yang lebih sederhana.

Problem Reduction membantu dalam menganalisis masalah kompleks dengan membaginya menjadi submasalah yang lebih sederhana atau memanfaatkan pengetahuan tentang masalah yang sudah ada untuk menyelesaikan masalah yang lebih sulit. Teknik ini dapat menghemat waktu dan upaya dalam merancang algoritma yang efisien atau menemukan solusi yang optimal.

BAB VII
SPACE & TIME TRADE -OFFS

A. Sorting by Counting

Sorting by Counting (pengurutan dengan menghitung) adalah algoritma pengurutan yang efisien untuk mengurutkan elemen-elemen dengan rentang nilai terbatas. Algoritma ini bekerja dengan menghitung jumlah kemunculan setiap elemen dalam daftar dan membangun daftar terurut berdasarkan perhitungan tersebut.

Langkah-langkah dalam Sorting by Counting adalah sebagai berikut:

1. Menghitung jumlah kemunculan setiap elemen dalam daftar.
2. Membangun daftar terurut berdasarkan jumlah kemunculan dengan mempertimbangkan urutan nilai elemen.

Keuntungan dari Sorting by Counting adalah memiliki kompleksitas waktu O(n + k), di mana n adalah jumlah elemen dalam daftar dan k adalah rentang nilai elemen. Algoritma ini efisien saat rentang nilai terbatas, tetapi tidak cocok untuk data yang memiliki rentang nilai yang sangat besar.

B. Input Enhancement in String Matching

Input Enhancement (perbaikan masukan) dalam String Matching adalah metode untuk meningkatkan kinerja algoritma pencocokan string dengan memanfaatkan sifat-sifat atau informasi tambahan tentang pola dan teks yang sedang dicocokkan.

Beberapa teknik input enhancement yang umum digunakan dalam String Matching adalah:

1. Preprocessing: Melakukan persiapan atau pemrosesan awal pada pola atau teks sebelum dilakukan pencocokan string. Contohnya, membangun struktur data yang mempercepat pencarian atau mengidentifikasi pola khusus dalam teks.

2. Indexing: Membuat indeks atau struktur data tambahan yang menyimpan informasi yang dapat mempercepat pencarian atau memperkecil ruang pencarian.
3. Compression: Mengompresi pola atau teks menjadi bentuk yang lebih efisien, sehingga mempercepat proses pencocokan.

Input Enhancement membantu dalam meningkatkan efisiensi algoritma pencocokan string dengan memanfaatkan pengetahuan khusus tentang pola dan teks yang sedang dicocokkan.

C. Hasing

Hashing (penghakisan) adalah teknik dalam pemrosesan data yang digunakan untuk mengonversi data input menjadi indeks dalam struktur data yang disebut tabel hash. Tabel hash adalah struktur data yang digunakan untuk menyimpan dan mengambil data dengan waktu konstan (O(1)).

Proses hashing melibatkan penggunaan fungsi hash, yang mengubah data input menjadi nilai hash yang unik atau hampir unik. Nilai hash ini kemudian digunakan sebagai indeks untuk mengakses dan menyimpan data dalam tabel hash. Jika fungsi hash yang baik digunakan, pencarian dan penyisipan data dalam tabel hash dapat dilakukan dengan kompleksitas waktu konstan.

Beberapa aplikasi hashing termasuk:

1. Pencarian dan pencocokan data: Hashing digunakan untuk mempercepat pencarian dan pencocokan data dalam struktur data seperti tabel hash, hashmap, atau kamus.
2. Pemindaian dan deduplikasi data: Hashing dapat digunakan untuk memindai dan menghapus duplikat data dengan cepat.
3. Keamanan: Hashing digunakan dalam algoritma kriptografi untuk mengamankan data dan verifikasi integritas data.

Hashing adalah teknik yang penting dalam pemrosesan data yang cepat dan efisien. Fungsi hash yang baik dan pemilihan struktur data yang tepat sangat penting dalam implementasi hashing yang efektif.

BAB VIII
DYNAMIC PROGRAMMING

A. Three Basic

Three Basic (tiga dasar) merujuk pada tiga algoritma dasar yang sering digunakan dalam analisis algoritma dan pemrograman:

1. Brute Force: Brute Force adalah pendekatan yang sederhana tetapi naif dalam mencari solusi untuk masalah. Pendekatan ini melibatkan mencoba semua kemungkinan solusi secara sistematis dan memilih solusi yang benar. Meskipun dapat memecahkan masalah, Brute Force sering kali tidak efisien dan tidak praktis untuk masalah dengan ukuran yang besar.

2. Greedy Algorithm: Greedy Algorithm adalah algoritma yang memilih langkah terbaik secara lokal pada setiap langkah untuk mencapai solusi yang optimal secara keseluruhan. Pendekatan ini berfokus pada memaksimalkan atau meminimalkan nilai tertentu pada setiap langkah tanpa mempertimbangkan konsekuensi jangka panjang. Meskipun Greedy Algorithm dapat memberikan solusi yang cepat, namun tidak selalu menghasilkan solusi optimal.

3. Divide and Conquer: Divide and Conquer (bagi dan taklukkan) adalah pendekatan yang melibatkan memecah masalah besar menjadi submasalah yang lebih kecil, memecahkan submasalah tersebut secara rekursif, dan menggabungkan solusi submasalah untuk memperoleh solusi akhir. Pendekatan ini sangat berguna untuk memecahkan masalah yang kompleks dengan membaginya menjadi masalah yang lebih sederhana dan terkelola.

B. The Knapsack Problem and Memory Fungctions

The Knapsack Problem (masalah ransel) adalah masalah optimasi yang melibatkan pemilihan item dari himpunan item yang tersedia dengan mempertimbangkan bobot dan nilai setiap item, dengan tujuan memaksimalkan nilai total item yang dapat dimasukkan ke dalam ransel dengan kapasitas tertentu. Terdapat dua jenis Knapsack Problem, yaitu Knapsack Problem 0/1 (item hanya dapat diambil atau tidak diambil) dan Knapsack Problem Kontinu (item dapat diambil sebagian).

Memory Functions (fungsi memori) dalam konteks ini merujuk pada teknik yang digunakan dalam pemecahan masalah Knapsack untuk menghindari duplikasi dan mempercepat pencarian solusi. Dalam masalah Knapsack, algoritma dapat menggunakan struktur data seperti tabel hash atau tabel memoization untuk menyimpan hasil yang sudah dikomputasi sebelumnya dan menghindari perhitungan ulang yang tidak perlu.

C. Warshall’s and Floyd’s Algorithms

Warshall's Algorithm (Algoritma Warshall) dan Floyd's Algorithm (Algoritma Floyd) adalah dua algoritma yang terkait dengan analisis dan manipulasi graf.

1. Warshall's Algorithm: Algoritma Warshall digunakan untuk mencari jalur terpendek antara setiap pasang simpul dalam sebuah graf berbobot atau berarah. Algoritma ini menggunakan pendekatan matriks untuk mencari jalur terpendek dengan memanipulasi matriks jarak antara simpul-simpul.
2. Floyd's Algorithm: Algoritma Floyd, juga dikenal sebagai Algoritma Floyd-Warshall, digunakan untuk mencari jalur terpendek antara setiap pasang simpul dalam sebuah graf berbobot atau berarah. Algoritma ini juga menggunakan pendekatan matriks, tetapi menggunakan pendekatan perulangan untuk memperbarui matriks jarak antara simpul-simpul.

Kedua algoritma ini berguna dalam analisis jaringan, optimisasi rute, dan permasalahan terkait graf yang melibatkan pencarian jalur terpendek.

BAB IX
GREEDY TECHNIQUE

A. Prims's Algorithm

Prim's Algorithm (Algoritma Prim) adalah algoritma yang digunakan untuk mencari Minimum Spanning Tree (MST) dari sebuah graf berbobot terhubung. MST adalah himpunan subgraf berbobot terhubung dengan semua simpul dalam graf asli dan memiliki total bobot terkecil.

Langkah-langkah dalam Prim's Algorithm adalah sebagai berikut:

1. Pilih simpul awal secara acak dan masukkan ke dalam MST.
2. Cari tepi terkecil yang terhubung dengan MST saat ini dan tambahkan ke MST.
3. Ulangi langkah 2 hingga semua simpul termasuk dalam MST.

Prim's Algorithm menggunakan pendekatan greedy, dimana pada setiap langkah memilih tepi terkecil yang akan ditambahkan ke MST saat ini. Algoritma ini terus memperluas MST hingga mencakup semua simpul dalam graf.

B. Kruskal's Algorithm

Kruskal's Algorithm (Algoritma Kruskal) juga digunakan untuk mencari Minimum Spanning Tree (MST) dari sebuah graf berbobot terhubung. Algoritma ini menggunakan pendekatan greedy dengan memilih tepi terkecil secara berurutan dan menambahkannya ke MST jika tidak membentuk siklus.

Langkah-langkah dalam Kruskal's Algorithm adalah sebagai berikut:

1. Urutkan semua tepi berdasarkan bobotnya.
2. Mulai dengan himpunan kosong untuk MST.
3. Ambil tepi terkecil dan tambahkan ke MST jika tidak membentuk siklus.
4. Ulangi langkah 3 hingga MST mencakup semua simpul.

Kruskal's Algorithm membangun MST dengan menggabungkan himpunan tepi yang tidak membentuk siklus. Algoritma ini terus memilih tepi terkecil yang aman untuk ditambahkan ke MST hingga semua simpul termasuk dalam MST.

C. Djikstra’s Algorithm

Dijkstra's Algorithm (Algoritma Dijkstra) adalah algoritma yang digunakan untuk mencari jalur terpendek antara simpul awal dan semua simpul lainnya dalam sebuah graf berbobot. Algoritma ini bekerja dengan asumsi bahwa semua bobot tepi adalah non-negatif.

Langkah-langkah dalam Dijkstra's Algorithm adalah sebagai berikut:

1. Inisialisasi jarak awal ke simpul awal dengan 0 dan jarak ke semua simpul lainnya dengan tak hingga.
2. Ambil simpul dengan jarak terkecil yang belum dikunjungi.
3. Perbarui jarak ke simpul-simpul terhubung dengan simpul saat ini, jika ditemukan jalur yang lebih pendek.
4. Ulangi langkah 2 dan 3 hingga semua simpul dikunjungi.

Dijkstra's Algorithm menggunakan pendekatan greedy, dimana pada setiap langkah memilih simpul dengan jarak terkecil yang belum dikunjungi. Algoritma ini terus memperbarui jarak ke simpul-simpul terhubung dengan simpul saat ini hingga menemukan jalur terpendek ke semua simpul.

D. Huffman Tress and Codes

Huffman Trees (Pohon Huffman) dan Huffman Codes (Kode Huffman) digunakan dalam kompresi data, terutama dalam kompresi teks. Algoritma Huffman digunakan untuk menghasilkan kode biner yang efisien berdasarkan frekuensi kemunculan karakter dalam teks.

Langkah-langkah dalam pembentukan Huffman Trees dan Huffman Codes adalah sebagai berikut:

1. Hitung frekuensi kemunculan karakter dalam teks.
2. Buat pohon Huffman dengan membangun pohon biner yang memiliki karakter sebagai daun dan bobot frekuensi sebagai bobot tepi.
3. Assign kode biner unik ke setiap karakter berdasarkan jalur dari akar ke daun dalam pohon Huffman.
4. Gantikan karakter dalam teks dengan kode Huffman yang sesuai.

Huffman Trees dan Huffman Codes memanfaatkan sifat kemunculan karakter untuk memberikan kode biner yang lebih pendek untuk karakter yang lebih sering muncul. Ini menghasilkan kompresi data dengan menggantikan karakter dengan kode Huffman yang lebih pendek dan lebih efisien secara ruang.

BAB X
ITERATIVE IMPROVEMENT

A. The Simplex Method

Metode Simpleks adalah algoritma yang digunakan untuk menyelesaikan masalah pemrograman linear. Metode ini banyak digunakan untuk mengoptimalkan fungsi tujuan yang tunduk pada kendala kesetaraan dan ketidaksamaan linear.

Metode Simpleks dimulai dengan solusi awal yang memenuhi syarat dan secara iteratif ditingkatkan dengan berpindah dari satu titik sudut dari wilayah layak ke titik sudut lainnya. Pada setiap iterasi, metode ini memilih elemen pivot dan melakukan operasi baris untuk memutar tabel dan menuju solusi optimal. Proses ini berlanjut hingga solusi optimal tercapai.

Metode Simpleks efisien dalam menyelesaikan masalah pemrograman linear dengan jumlah variabel dan kendala yang besar. Metode ini memiliki berbagai aplikasi dalam bidang-bidang seperti riset operasi, ekonomi, rekayasa, dan manajemen.

B. The maximum-Flow Problem

Masalah Aliran Maksimum adalah masalah klasik dalam teori graf dan jaringan. Masalah ini melibatkan mencari aliran maksimum yang dapat dikirim melalui suatu jaringan dari suatu simpul sumber ke simpul tujuan, dengan mematuhi batasan kapasitas pada setiap sisi.

Masalah ini dapat direpresentasikan sebagai graf berarah, di mana simpul-simpul mewakili entitas (seperti simpul, kota, atau komputer) dan sisi-sisi mewakili aliran antara entitas tersebut. Setiap sisi memiliki kapasitas yang menentukan jumlah aliran maksimum yang dapat dilalui.

Tujuan dari Masalah Aliran Maksimum adalah mencari aliran yang memaksimalkan total aliran dari sumber ke tujuan, sambil memastikan bahwa

aliran memenuhi batasan kapasitas dan memenuhi kriteria aliran di setiap simpul.

Berbagai algoritma dapat digunakan untuk memecahkan masalah Aliran Maksimum, seperti Algoritma Edmonds-Karp dan Algoritma Ford-Fulkerson.

C. Maximum Matching in Bipartite Graphs

Pencocokan Maksimum dalam Graf Bipartit adalah masalah yang melibatkan mencari himpunan pasangan yang cocok (pencocokan) yang memenuhi sebanyak mungkin simpul di kedua himpunan simpul yang saling berhubungan dalam graf bipartit.

Graf bipartit adalah graf yang terdiri dari dua himpunan simpul, di mana setiap sisi hanya menghubungkan simpul dari himpunan yang berbeda. Misalnya, graf bipartit dapat mewakili hubungan antara pria dan wanita, di mana simpul dari himpunan pertama mewakili pria dan simpul dari himpunan kedua mewakili wanita.

Pencocokan Maksimum dalam Graf Bipartit berusaha untuk menemukan pencocokan yang memadankan sebanyak mungkin simpul di kedua himpunan simpul, tanpa adanya dua sisi yang terhubung dengan simpul yang sama.

Algoritma yang umum digunakan untuk menyelesaikan masalah ini adalah Algoritma Hopcroft-Karp dan Algoritma Berge-Tutte.

D. The Stable Marriage Problem

Masalah Pernikahan yang Stabil adalah masalah yang melibatkan mencari pencocokan stabil dalam situasi pernikahan di mana setiap pria dan wanita memberikan preferensi mereka terhadap satu sama lain.

Dalam masalah ini, terdapat sejumlah pria dan wanita yang harus dipasangkan menjadi pasangan yang stabil. Sebuah pencocokan stabil adalah pencocokan di mana tidak ada pasangan yang saling lebih memilih satu sama lain daripada pasangan mereka yang saat ini.

Masalah Pernikahan yang Stabil dapat dipecahkan menggunakan Algoritma Gale-Shapley, yang melibatkan proses proposisi dan penerimaan di antara pria dan wanita. Algoritma ini menghasilkan pencocokan stabil yang memenuhi preferensi semua pria dan wanita.

BAB XI
LIMITATIONS OF ALGORITHM POWER

A. Lower-Bounf Arguments

Lower-Bound Arguments (Argumen Batas Bawah) adalah pendekatan dalam analisis algoritma yang digunakan untuk membuktikan bahwa tidak ada algoritma yang lebih efisien atau lebih cepat dari suatu batas tertentu dalam menyelesaikan masalah tertentu.

Dalam analisis algoritma, seringkali kita tertarik untuk mengetahui batas bawah (lower-bound) dari kompleksitas waktu atau ruang yang dibutuhkan untuk menyelesaikan suatu masalah. Lower-Bound Arguments digunakan untuk membuktikan bahwa tidak ada algoritma yang dapat menyelesaikan masalah dengan kompleksitas di bawah batas tertentu.

Contohnya, dalam masalah pengurutan (sorting), lower-bound argument dapat digunakan untuk membuktikan bahwa tidak ada algoritma pengurutan yang dapat mengurutkan array dengan kompleksitas waktu di bawah O(n log n), di mana n adalah ukuran array.

B. Decision Tree

Decision Tree (Pohon Keputusan) adalah struktur data berbentuk pohon yang digunakan dalam analisis algoritma dan pemrosesan data. Pohon keputusan menyajikan serangkaian keputusan atau pilihan yang harus diambil berdasarkan kondisi atau kriteria yang diberikan.

Dalam konteks analisis algoritma, Decision Tree sering digunakan untuk memodelkan algoritma pemilihan atau pengambilan keputusan. Setiap simpul dalam pohon keputusan mewakili keputusan atau pemilihan tertentu, sedangkan cabang-cabang dari simpul tersebut mewakili kemungkinan hasil dari keputusan tersebut.

Pohon keputusan dapat digunakan untuk memodelkan algoritma kompleks dengan serangkaian keputusan yang harus diambil. Mereka juga digunakan dalam pemrosesan data, machine learning, dan pemecahan masalah lainnya.

C. P, NP and-Complete Problems

Dalam teori kompleksitas komputasional, P, NP, dan masalah NP-sulit adalah konsep penting. P adalah kelas masalah yang dapat diselesaikan dalam waktu yang efisien menggunakan algoritma deterministik. NP adalah kelas masalah yang dapat diverifikasi dalam waktu yang efisien menggunakan algoritma non-deterministik.

Masalah NP-sulit adalah kelas masalah yang paling sulit dalam kelas NP. Mereka adalah masalah yang setidaknya seberat masalah-masalah dalam kelas NP, yang berarti bahwa jika ada algoritma efisien untuk menyelesaikan satu masalah NP-sulit, maka ada algoritma efisien untuk menyelesaikan semua masalah dalam kelas NP.

Masalah NP-sulit tidak perlu berada dalam kelas NP. Namun, jika ada algoritma efisien yang dapat menyelesaikan salah satu masalah NP-sulit, maka semua masalah NP-sulit dapat diselesaikan secara efisien.

P, NP, dan masalah NP-sulit memiliki implikasi yang signifikan dalam teori komputasi dan pemecahan masalah. Membuktikan bahwa suatu masalah adalah NP-sulit atau menemukan algoritma efisien untuk suatu masalah NP-sulit adalah tantangan yang kompleks.

D. Challenge of Numerical Algorithms

Algoritma Numerik adalah algoritma yang digunakan untuk memecahkan masalah matematika yang melibatkan perhitungan numerik atau pendekatan numerik. Mereka digunakan untuk menemukan solusi numerik atau perkiraan solusi untuk masalah matematika yang tidak dapat diselesaikan secara analitik.

Tantangan Algoritma Numerik meliputi berbagai masalah seperti:

1. Stabilitas numerik: Memastikan bahwa algoritma memberikan hasil yang akurat dan stabil dalam menghadapi kesalahan pembulatan atau ketidakstabilan numerik.
2. Efisiensi: Mencari algoritma yang memberikan solusi dengan waktu komputasi yang efisien.
3. Ketepatan: Memastikan bahwa solusi numerik mendekati solusi yang sebenarnya dengan tingkat kesalahan yang dapat diterima.
4. Skalabilitas: Memastikan bahwa algoritma dapat digunakan untuk menyelesaikan masalah numerik dengan ukuran yang besar atau kompleks.

Tantangan dalam Algoritma Numerik melibatkan pengembangan algoritma yang mengatasi masalah ini dan memberikan solusi numerik yang akurat dan efisien untuk berbagai masalah matematika.

BAB XII
COPING WITH THE LIMITATIONS OF ALGORITHM POWER

A. Backtracking

Backtracking adalah teknik pemrograman yang digunakan untuk mencari solusi melalui pencarian berbagai kemungkinan secara sistematis. Teknik ini cocok digunakan saat mencari solusi untuk masalah yang memiliki struktur pohon atau graf, di mana kita harus menguji beberapa kemungkinan sebelum menemukan solusi yang tepat.

Backtracking bekerja dengan menjelajahi semua kemungkinan langkah-langkah yang mungkin dalam pencarian solusi. Jika pada suatu titik langkah yang diuji tidak mengarah ke solusi yang valid, maka langkah tersebut akan dibatalkan (backtrack) dan mencoba langkah alternatif lainnya.

Contoh penggunaan backtracking adalah dalam pemecahan masalah Sudoku atau TSP (Traveling Salesman Problem). Dalam kedua masalah tersebut, backtracking digunakan untuk mencoba semua kemungkinan langkah dan membatalkan langkah jika mengarah ke solusi yang tidak valid.

B. Branch and Bound

Branch and Bound adalah teknik algoritma yang digunakan untuk mencari solusi optimal dalam masalah optimisasi atau pencarian. Teknik ini membagi masalah menjadi submasalah yang lebih kecil (branching) dan menggunakan batasan (bound) untuk menghindari pencarian pada submasalah yang tidak mengarah ke solusi yang optimal.

Algoritma Branch and Bound bekerja dengan cara membagi masalah menjadi submasalah yang lebih kecil. Pada setiap langkah, algoritma mengevaluasi batasan (bound) untuk setiap submasalah dan memilih submasalah dengan batasan terbaik untuk dieksplorasi lebih lanjut. Hal ini dilakukan dengan tujuan untuk menghindari pencarian pada submasalah yang tidak mungkin

menghasilkan solusi yang lebih baik dari solusi yang telah ditemukan sebelumnya.

Contoh penggunaan Branch and Bound adalah dalam masalah Penugasan (Assignment Problem) atau Ransum Pengiriman (Knapsack Problem). Dalam kedua masalah tersebut, Branch and Bound digunakan untuk mencari solusi optimal dengan mengurangi ruang pencarian melalui penggunaan batasan.

C. Algorithms for Solving Nonlinear Problems

Algoritma untuk memecahkan masalah nonlinear adalah algoritma yang digunakan untuk menemukan solusi numerik dari masalah yang melibatkan persamaan atau fungsi nonlinear. Masalah nonlinear sering kali tidak dapat diselesaikan secara analitik, sehingga algoritma numerik digunakan untuk mendekati solusinya.

Beberapa algoritma yang umum digunakan untuk memecahkan masalah nonlinear adalah Metode Newton-Raphson, Metode Gradient, Metode Levenberg-Marquardt, dan Metode Quasi-Newton. Algoritma-algoritma ini berfokus pada mencari titik-titik di mana turunan fungsi mendekati nol, yang menandakan kemungkinan solusi dari persamaan nonlinear.

Algoritma untuk memecahkan masalah nonlinear memiliki tantangan tersendiri karena kompleksitas fungsi nonlinear dan ketergantungan pada kondisi awal yang dianggap. Oleh karena itu, pemilihan algoritma yang tepat dan pengaturan kondisi awal yang baik sangat penting dalam memperoleh solusi yang akurat dan konvergen.

DAFTAR PUSTAKA

- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms. MIT Press.
- Skiena, S. S. (2008). The Algorithm Design Manual. Springer.
- Dasgupta, S., Papadimitriou, C. H., & Vazirani, U. V. (2008). Algorithms. McGraw-Hill.
- Kleinberg, J., & Tardos, É. (2005). Algorithm Design. Addison-Wesley.
- Goodrich, M. T., & Tamassia, R. (2014). Algorithm Design and Applications. John Wiley & Sons.
- Levitin, A. (2011). Introduction to the Design and Analysis of Algorithms. Pearson.
- Sedgewick, R., & Wayne, K. (2011). Algorithms, Part I (Online Course). Princeton University (available on Coursera).
- Sedgewick, R. (2001). Algorithms in C++. Addison-Wesley.
- Goodrich, M. T., & Tamassia, R. (2002). Algorithm Design: Foundations, Analysis, and Internet Examples. John Wiley & Sons.
- Brassard, G., & Bratley, P. (1996). Fundamentals of Algorithmics. Prentice Hall.