Perancangan Analisis Algoritma
Rangkuman Pembelajaran

Disusun Oleh :

21346016 Nafisa Aura Dwiyanti

Dosen Pengampu :

Widya Darwin S.kom,. M.Pd.T

PROGRAM STUDI INFORMATIKA

JURUSAN TEKNIK ELEKTRONIKA

FAKUTAS TEKNIK

UNIVERSITAS NEGERI PADANG

2023

RINGKASAN MATERI

A. PENGERTIAN ANALISIS ALGORITMA

Algoritma adalah urutan langkah-langkah logis penyelesaian masalah yang disusun secara sistematis dan logis. Kata Logis merupakan kata kunci dalam Algoritma. Langkah-langkah dalam Algoritma harus logis dan harus dapat ditentukan bernilai salah atau benar.

Algoritma adalah sekelompok aturan atau langkah-langkah untuk menyelesaikan perhitungan yang dilakukan oleh tangan atau mesin (komputer0 hampir semua bidang pekerjaan memerlukan komputer untuk membantu perhitungan atau pekerjaan yang membutuhkan ketepatan waktu dan kepresisian dimensi, komputrer akan bisa mengerjakan pekerjaan tersebut jika diberikan program yang sesuai Untuk membuat priogram yang baik diperlukan perancangan program serta analisis algoritma yang memadai.

1. Algoritma Brute Force

   Brute force adalah sebuah pendekatan yang lempang (straightforward) untukmemecahkan suatu masalah, biasanya didasarkan pada pernyataan masalah(problem statement) dan definisi konsep yang dilibatkan.dilibatkan. Algoritma brute force memecahkan masalah dengan sangat sederhana, langsungdan dengan cara yang jelas (obvious way).

   Karakteristik Algoritma Brute Force :

   - Algoritma brute force umumnya tidak “cerdas” dan tidak mangkus, karena ia membutuhkan jumlah langkah yang besar dalam penyelesaiannya. Kadang-kadang algoritma brute force disebut juga algoritma naif (naïve algorithm).
   - Algoritma brute force seringkali merupakan pilihan yang kurang disukai karena ketidakmangkusannya itu, tetapi dengan mencari pola-pola yang mendasar, keteraturan, atau trik-trik khusus, biasanya akan membantu kita menemukan algoritma yang lebih cerdas dan lebih mangkus.
   - Untuk masalah yang ukurannya kecil, kesederhanaan brute force biasanya lebihdiperhitungkan daripada ketidakmangkusannya.

   Kelebihan Algoritma Brute Force :

   - Sederhana dan lebih mudah dimengerti.
   - Cocok untuk memberikan solusi seperti pengurutan, pencarian, pencocokan.
   - Proses pemecahan masalah menjadi lebih baik dan optimal.

   Kekurangan Algoritma Brute Force :

   - Tidak cocok untuk kebutuhan pemecahan masalah yang cepat
   - Proses pemecahan menggunakan Algoritma Brute Force agak lama.
   - Terlalu banyak mempertimbangkan opsi sebelum eksekusi.

Algoritma adalah deretan langkah-langkah komputasi yang mentransformasikan data masukan menjadi keluaran. Algoritma adalah deretan instruksi yang jelas untuk memecahkan masalah, yaitu untuk memperoleh keluaran yang diinginkan dari suatu masukan dalam jumlah waktu yang terbatas.

Notasi apapun dapat digunakan untuk menuliskan algoritma asalkan mudah dibaca dan dipahami. Algoritma dapat ditulis dengan notasi:

- Bagan alir (flow chart)
- Kalimat-kalimat deskriptif
- Pseudo-code (gabungan antara bahasa alami dengan bahasa pemrograman)

Kriteria Algoritma yang Baik :

- Mempunyai logika yang tepat untuk memecahkan masalah
- Menghasilkan output yang benar dalam waktu yang singkat
- Ditulis dalam bahasa baku terstruktur sehingga tidak menimbulkan arti ganda
- Ditulis dengan format baku sehingga mudah diimplementasikan ke dalam bahasa pemrograman
- Semua operasi didefinisikan dengan jelas dan berakhir sesudah sejumlah Langkah

Pemrograman Terstruktur

Pemrograman terstruktur adalah pola penyusunan program komputer hanya dengan menggunakan tiga struktur kontrol, yaitu :

- Squence : merupakan urutan pengerjaan dari perintah/statement pertama sampai dengan perintah/statement terakhir. Pada umumnya bahasa pemrograman memiliki *squence* mulai dari atas ke bawah dan dari kiri ke kanan (*top-down*)
- Selection : struktur kontrol *selection* adalah penggambaran sebuah kondisi dan pilihan diantara dua aksi. Statement pertama akan dikerjakan, jika kondisi benar, jika tidak maka akan mengerjakan perintah setelah keyword (else).
- Repetition : merupakan perintah untuk melakukan pengulangan dengan kondisi tertentu.

Dasar Pemrograman

Pemrograman memiliki delapan operasi dasar, diantaranya adalah :

- Membaca data (input)
- Menampilkan data (output)
- Melakukan perhitungan aritmatika (compute)
- Memberikan nilai ke suatu identifier (store)
- Membandingkan dan memilih (compare)
- Melakukan pengulangan (loop)
- Array
- Function

Prinsip cara kerja algoritma

Pada dasarnya, algoritma merupakan deskripsi proses untuk mengerjakan sesuatu yang disusun dalam sederet aksi. Secara sederhana, prinsip kerja algoritma terbagi menjadi, masukan (input), proses, dan keluaran (output).

Dalam kehidupan sehari-hari, prinsip kerja algoritma dapat dipahami ketika kita ingin membuat telur dadar. Sebelum membuat algoritma, hal yang perlu kita lakukan adalah mendefinisikan masukan (input) dan keluaran (output).

Berdasarkan contoh di atas, maka yang menjadi masukan adalah telur mentah dan yang menjadi keluaran adalah telur dadar matang. Dengan demikian, susunan algoritmanya menjadi sebagai berikut:

1. Nyalakan api kompor,
2. Tuangkan minyak ke dalam wajan,
3. Pecahkan telur ayam ke dalam mangkuk,
4. Tuangkan garam secukupnya,
5. Kocok campuran telur dan garam,
6. Tuang adonan telur ke dalam wajan,
7. Masak telur hingga matang.

Struktur Dasar Algoritma
Secara umum, struktur dasar algoritma terdiri dari sekuensial (sequential), test kondisi atau percabangan (branching), dan perulangan (looping).

1. Algoritma Sekuensial
Algoritma sekuensial adalah langkah-langkah yang dilakukan secara berurutan sesuai dengan urutan penulisannya. Struktur ini merupakan struktur yang paling sering dilakukan.

Contoh:
Algoritma memiliki empat baris aksi, yaitu t1, t2, t3, dan t4, maka semua aksi akan dilakukan secara berurutan mulai dari aksi t1 sampai t4.

2. Algoritma Percabangan (Branching)
Dalam kehidupan sehari-hari ada kalanya suatu kegiatan akan dilakukan dan tidak dilakukan tergantung situasi tertentu. Begitu pun dengan algoritma, ada kalanya satu atau beberapa aksi akan dikerjakan dan tidak dikerjakan tergantung situasi tertentu.

Nah, struktur algoritma percabangan ini digunakan untuk mengerjakan satu aksi dari beberapa pilihan yang diberikan.

3. Algoritma Perulangan (Looping)
Sama halnya dengan manusia, algoritma juga mengenal kegiatan pengulangan, yakni melakukan satu atau beberapa kegiatan secara berulang-ulang. Namun, berbeda dengan manusia, komputer tidak

mengenal istilah lelah atau bosan dalam melakukan kegiatan yang sama secara berulang.

Dengan demikian, struktur perulangan atau looping digunakan untuk menjalankan kegiatan yang dilakukan berulang-ulang.

Cara Penyajian Algoritma
Penyajian algoritma akan lebih baik jika ditulis secara sistematis. Ada tiga cara yang bisa kamu gunakan untuk menyajikan algoritma, yakni secara naratif, flowchart atau diagram/bagan alir, dan pseudocode.

1. Naratif
Penyajian algoritma secara naratif dituliskan dengan menggunakan cerita seperti dalam bahasa sehari-hari.

Contoh: Menghitung luas segitiga menggunakan naratif
Langkah-1 : Mulai
Langkah-2 : Baca nilai Alas
Langkah-3 : Baca nilai Tinggi
Langkah-4 : Hitung Luas = (Alas x Tinggi) / 2
Langkah-5 : Cetak Hasil Luas
Langkah-6 : Selesai

2. Flowchart
Dengan flowchart, cara penyajian algoritma dibuat dalam urutan simbol-simbol khusus. Urutan simbol digambarkan sesuai dengan arah tanda panah.
Contoh:

Contoh flowchart Foto: Ist
Sumber gambar: Algoritma dan Pemrograman (Sitorus, 2015)

3. Pseudocode
Langkah-langkah penyelesaian masalah ini ditulis dengan cara yang mirip atau menyerupai program. Pseudocode tidak spesifik terhadap salah satu bahasa pemrograman sehingga algoritma ini dapat diterjemahkan menyesuaikan bahasa pemrograman yang ada dalam suatu program.

Contoh: Menghitung luas segitiga menggunakan pseudocode

Input (Alas)
Input (Tinggi)
Luas ← (Alas x Tinggi) / 2
Struktur Dasar Algoritma
Secara umum, struktur dasar algoritma terdiri dari sekuensial (sequential), test kondisi atau percabangan (branching), dan perulangan (looping).

1. Algoritma Sekuensial
Algoritma sekuensial adalah langkah-langkah yang dilakukan secara

berurutan sesuai dengan urutan penulisannya. Struktur ini merupakan struktur yang paling sering dilakukan.

Contoh:
Algoritma memiliki empat baris aksi, yaitu t1, t2, t3, dan t4, maka semua aksi akan dilakukan secara berurutan mulai dari aksi t1 sampai t4.

2. Algoritma Percabangan (Branching)
Dalam kehidupan sehari-hari ada kalanya suatu kegiatan akan dilakukan dan tidak dilakukan tergantung situasi tertentu. Begitu pun dengan algoritma, ada kalanya satu atau beberapa aksi akan dikerjakan dan tidak dikerjakan tergantung situasi tertentu.

Nah, struktur algoritma percabangan ini digunakan untuk mengerjakan satu aksi dari beberapa pilihan yang diberikan.

3. Algoritma Perulangan (Looping)
Sama halnya dengan manusia, algoritma juga mengenal kegiatan pengulangan, yakni melakukan satu atau beberapa kegiatan secara berulang-ulang. Namun, berbeda dengan manusia, komputer tidak mengenal istilah lelah atau bosan dalam melakukan kegiatan yang sama secara berulang.

Dengan demikian, struktur perulangan atau looping digunakan untuk menjalankan kegiatan yang dilakukan berulang-ulang.

Cara Penyajian Algoritma
Penyajian algoritma akan lebih baik jika ditulis secara sistematis. Ada tiga cara yang bisa kamu gunakan untuk menyajikan algoritma, yakni secara naratif, flowchart atau diagram/bagan alir, dan pseudocode.

1. Naratif
Penyajian algoritma secara naratif dituliskan dengan menggunakan cerita seperti dalam bahasa sehari-hari.

Contoh: Menghitung luas segitiga menggunakan naratif
Langkah-1 : Mulai
Langkah-2 : Baca nilai Alas
Langkah-3 : Baca nilai Tinggi
Langkah-4 : Hitung Luas = (Alas x Tinggi) / 2
Langkah-5 : Cetak Hasil Luas
Langkah-6 : Selesai

2. Flowchart
Dengan flowchart, cara penyajian algoritma dibuat dalam urutan simbol-simbol khusus. Urutan simbol digambarkan sesuai dengan arah tanda panah.
Contoh:

Contoh flowchart Foto: Ist
Sumber gambar: Algoritma dan Pemrograman (Sitorus, 2015)

3. Pseudocode

Langkah-langkah penyelesaian masalah ini ditulis dengan cara yang mirip atau menyerupai program. Pseudocode tidak spesifik terhadap salah satu bahasa pemrograman sehingga algoritma ini dapat diterjemahkan menyesuaikan bahasa pemrograman yang ada dalam suatu program.

Contoh: Menghitung luas segitiga menggunakan pseudocode

Input (Alas)
Input (Tinggi)
Luas ← (Alas x Tinggi) / 2
Struktur Dasar Algoritma
Secara umum, struktur dasar algoritma terdiri dari sekuensial (sequential), test kondisi atau percabangan (branching), dan perulangan (looping).

1. Algoritma Sekuensial
Algoritma sekuensial adalah langkah-langkah yang dilakukan secara berurutan sesuai dengan urutan penulisannya. Struktur ini merupakan struktur yang paling sering dilakukan.

Contoh:
Algoritma memiliki empat baris aksi, yaitu t1, t2, t3, dan t4, maka semua aksi akan dilakukan secara berurutan mulai dari aksi t1 sampai t4.

2. Algoritma Percabangan (Branching)
Dalam kehidupan sehari-hari ada kalanya suatu kegiatan akan dilakukan dan tidak dilakukan tergantung situasi tertentu. Begitu pun dengan algoritma, ada kalanya satu atau beberapa aksi akan dikerjakan dan tidak dikerjakan tergantung situasi tertentu.

Nah, struktur algoritma percabangan ini digunakan untuk mengerjakan satu aksi dari beberapa pilihan yang diberikan.

3. Algoritma Perulangan (Looping)
Sama halnya dengan manusia, algoritma juga mengenal kegiatan pengulangan, yakni melakukan satu atau beberapa kegiatan secara berulang-ulang. Namun, berbeda dengan manusia, komputer tidak mengenal istilah lelah atau bosan dalam melakukan kegiatan yang sama secara berulang.

Dengan demikian, struktur perulangan atau looping digunakan untuk menjalankan kegiatan yang dilakukan berulang-ulang.

Cara Penyajian Algoritma
Penyajian algoritma akan lebih baik jika ditulis secara sistematis. Ada tiga cara yang bisa kamu gunakan untuk menyajikan algoritma, yakni secara naratif, flowchart atau diagram/bagan alir, dan pseudocode.

1. Naratif
Penyajian algoritma secara naratif dituliskan dengan menggunakan cerita seperti dalam bahasa sehari-hari.

Contoh: Menghitung luas segitiga menggunakan naratif
Langkah-1 : Mulai

Langkah-2 : Baca nilai Alas
Langkah-3 : Baca nilai Tinggi
Langkah-4 : Hitung Luas = (Alas x Tinggi) / 2
Langkah-5 : Cetak Hasil Luas
Langkah-6 : Selesai

2. Flowchart
Dengan flowchart, cara penyajian algoritma dibuat dalam urutan simbol-simbol khusus. Urutan simbol digambarkan sesuai dengan arah tanda panah.
Contoh:

Contoh flowchart Foto: Ist
Sumber gambar: Algoritma dan Pemrograman (Sitorus, 2015)

3. Pseudocode
Langkah-langkah penyelesaian masalah ini ditulis dengan cara yang mirip atau menyerupai program. Pseudocode tidak spesifik terhadap salah satu bahasa pemrograman sehingga algoritma ini dapat diterjemahkan menyesuaikan bahasa pemrograman yang ada dalam suatu program.

Contoh: Menghitung luas segitiga menggunakan pseudocode

Input (Alas)
Input (Tinggi)
Luas ← (Alas x Tinggi) / 2
Output.

Jenis-jenis Algoritma

Terdapat beberapa klasifikasi algoritma yang dibagi berdasarkan karakter tertentu. Salah satu cara dalam melakukan pembagian jenis tersebut adalah berdasarkan paradigma dan metode yang digunakan dalam perancangan algoritma tersebut. Beberapa paradigma yang digunakan untuk menyusun suatu algoritma antara lain adalah sebagai berikut.

- **Divide and Conquer**,
  merupakan paradigma untuk membagi suatu permasalahan yang besar menjadi permasalahan-permasalahan yang kecil. Pembagian masalah ini dilakukan secara terus-menerus sampai ditemukan bagian masalah yang kecil dan mudah untuk dipecahkan.

- **Dynamic programming**,
  paradigma pemrograman dinamik akan sesuai jika digunakan pada suatu masalah yang mengandung sub-struktur yang optimal dan mengandung beberapa bagian permasalahan yang tumpang tindih. Paradigma ini sekilas terlihat mirip dengan paradigma divide and conquer, sama-sama mencoba untuk membagi permasalahan menjadi subpermasalahan yang lebih kecil, tapi secara intrinsic ada perbedaan dari karakter permasalahan yang dihadapi.

- **Metode serakah**,
  merupakan paradigma yang mirip dengan pemrograman dinamik, namun jawaban dari setiap submasalah tidak perlu diketahui dari setiap tahap, dan menggunakan pilihan apa yang terbaik pada saat itu.

- **Search and enumeration**,
  merupakan paradigma pemodelan yang memberikan aturan tertentu dalam pemecahan masalah dan optimalisasi.

Analisis Algoritma

Tujuan dari analisis algoritma adalah untuk membandingkan algoritma (atau solusi) terutama dalam hal waktu berjalan tetapi juga dalam hal faktor lain. Misalnya berapa banyak penggunaan sumber daya yang dibutuhkan? Bagaimana kontinuitasnya untuk dikembangkan? Berapa banyak waktu yang dibutuhkan untuk menyelesaikan suatu permasalahan? Apakah algoritma dapat menyelesaikan beberapa persoalan sekaligus? Dan sebagainya.
Berdasarkan uraian di atas, analisis algoritma harus memperhatikan beberapa hal di bawah ini.

1. Kebenaran (Correctness)
   Dalam pembuktian kebenaran suatu algoritma, hasil akhir dari algoritma tersebut haruslah diperiksa apakah sudah sesuai dengan kondisi-kondisi yang telah diberikan pada awal masukan. Untuk melakukan pemeriksaan suatu algoritma yang kompleks, kita dapat membagi algoritma tersebut menjadi beberapa modul kecil, sehingga jika modul kecil tersebut benar maka seluruh program akan benar.

2. Jumlah Operasi yang Dilakukan (Amount of Work Done)
   Penghitungan jumlah operasi yang dilakukan ini digunakan untuk membandingkan tingkat efisiensi suatu algoritma dengan algoritma lain dalam memecahkan suatu masalah yang sama. Hal ini dilakukan untuk mendapatkan algoritma yang dapat menghasilkan waktu eksekusi yang lebih cepat. Cara paling mudah dalam membandingkan dua buah algoritma adalah dengan menghitung jumlah operasi dasar yang dilakukan oleh algoritma tersebut, karena apabila dilakukan pembandingan langsung pada komputer, sering kali kondisi setiap komputer dan cara pembacaan setiap bahasa pemrograman mempengaruhi waktu pemecahan masalah.

3. Analisis Kemungkinan Terburuk (Worst Case)
   Analisis worst case merupakan analisis yang digunakan untuk melihat tingkat efektifitas suatu algoritma dalam menyelesaikan masalah-masalah yang masukannya merupakan masukan yang terkadang tidak perlu dihitung atau cara mengatasi pada saat kemungkinan masukan salah.

4. Optimal (Optimality)
   Untuk menganalisis suatu algoritma, biasanya selalu menggunakan kelas algoritma dan ukuran kompleksitas, misalnya, jumlah operasi dasar yang dilakukan. Sebuah algoritma disebut optimal (untuk worst case) jika tidak ada algoritma yang dapat melakukan operasi dasar yang lebih sedikit (untuk worst case).

5. Ikatan Terendah (Lower Bound)
   Untuk membuktikan bahwa suatu algoritma adalah optimal, tidak diperlukan menganalisis setiap algoritma. Dengan membuktikan teorema teorema yang menentukan lower bound pada jumlah operasi yang diperlukan untuk menyelesaikan masalah, maka algoritma yang dapat melakukan jumlah operasi tersebut disebut optimal.

B. DECREASE & CONQUER

Terdapat tiga varian utama dari paradigma desain algoritma decrease-and-conquer dalam bahasa Indonesia. Varian-varian ini melibatkan pengurangan ukuran masalah dengan cara yang berbeda untuk akhirnya menyelesaikan masalah tersebut. Yang pertama yaitu :

- Pengurangan dengan Faktor Konstan: Pada varian ini, masalah dikurangi dengan faktor konstan pada setiap langkah rekursif. Algoritma menyelesaikan submasalah yang lebih kecil dengan ukuran n/c, di mana c adalah konstanta yang lebih besar dari
- Pengurangan ini dilakukan hingga mencapai kasus dasar, di mana ukuran masalah menjadi cukup kecil untuk langsung diselesaikan. Contoh algoritma yang mengikuti varian ini termasuk pencarian biner dan pengurutan gabung (merge sort).
- Pengurangan dengan Faktor Variabel: Pada varian ini, ukuran masalah dikurangi dengan faktor variabel pada setiap langkah rekursif. Algoritma menyelesaikan submasalah dengan ukuran n/k, di mana k adalah variabel yang bergantung pada instansi masalah atau input. Nilai k dapat bervariasi berdasarkan karakteristik dari masalah yang sedang diselesaikan. Varian ini sering digunakan dalam algoritma seperti quicksort, di mana pemilihan pivot menentukan ukuran submasalah.
- Pengurangan dengan Jumlah Konstan: Pada varian ini, ukuran masalah dikurangi dengan jumlah konstan pada setiap langkah rekursif. Algoritma menyelesaikan submasalah yang lebih kecil dengan ukuran n-d, di mana d adalah konstanta. Pengurangan ini terus dilakukan hingga mencapai kasus dasar, di mana ukuran masalah menjadi cukup kecil untuk langsung diselesaikan.
- Contoh algoritma yang mengikuti varian ini termasuk selection sort dan insertion sort. Varian-varian algoritma decrease-and-conquer ini ditandai dengan cara spesifik mereka dalam mengurangi ukuran masalah dan menyelesaikan submasalah. Dengan secara iteratif menyelesaikan instansi masalah yang lebih kecil, algoritma tersebut akhirnya mencapai solusi dari masalah asli.

1. Short
   Pengurutan (sort) adalah proses mengatur elemen-elemen data

dalam urutan tertentu, seperti secara menaik (ascending) atau menurun (descending), berdasarkan suatu kunci atau kriteria tertentu. Pengurutan merupakan salah satu operasi dasar dalam pemrograman dan sering digunakan dalam berbagai aplikasi. Terdapat berbagai algoritma pengurutan yang dapat digunakan, masing-masing memiliki kelebihan dan kelemahan dalam hal waktu eksekusi, kebutuhan memori, dan kompleksitasnya. Berikut ini beberapa algoritma pengurutan yang umum digunakan:

- Selection Sort (Pengurutan Pilihan):Algoritma ini secara berulang mencari elemen terkecil dari sisa array dan menukar posisinya dengan elemen pertama yang belum terurut. Proses ini terus berlanjut hingga seluruh array terurut. Selection sort

- Insertion Sort (Pengurutan Sisip):Algoritma ini secara berulang membandingkan setiap elemen dengan elemen-elemen sebelumnya dalam array terurut dan menyisipkannya ke posisi yang tepat. Insertion sort juga memiliki kompleksitas waktu O(n^2), tetapi lebih efisien daripada selection sort dalam kasus terbaik ketika array sudah hampir terurut.

- Bubble Sort (Pengurutan Gelembung):Algoritma ini berulang kali membandingkan pasangan elemen berturut-turut dan menukar posisi mereka jika urutannya salah. Proses ini terus berlanjut hingga seluruh array terurut. Bubble sort juga memiliki kompleksitas waktu O(n^2) dan umumnya digunakan pada array dengan ukuran kecil atau pada kasus yang hampir terurut.

- Merge Sort (Pengurutan Gabung):Algoritma divide-and-conquer yang membagi array menjadi subarray yang lebih kecil, mengurutkan masing-masing subarray secara rekursif, dan menggabungkannya kembali untuk mendapatkan array terurut. Merge sort memiliki kompleksitas waktu O(n log n), yang membuatnya efisien untuk array dengan ukuran besar.

- Quick Sort (Pengurutan Cepat):Algoritma divide-and-conquer yang menggunakan pendekatan pemilihan elemen pivot dan membagi array menjadi dua subarray berdasarkan pivot tersebut. Subarray kemudian diurutkan secara rekursif.
Quick sort memiliki kompleksitas waktu rata-rata O(n log n), tetapi dapat mencapai O(n^2) pada kasus terburuk. Namun, quick sort sering kali lebih cepat dalam praktiknya dibandingkan dengan algoritma pengurutan lainnya. Selain algoritma-algoritma di atas, ada juga algoritma pengurutan lainnya seperti heap sort, radix sort, dan tim sort. Pemilihan algoritma pengurutan yang tepat tergantung pada sifat data, ukuran masalah, dan kebutuhan performa yang diinginkan.

C. TRANSFORM & CONQUER

a. Instance Simplification

Instance Simplification (Pemudahan Instansi) adalah teknik yang digunakan dalam analisis algoritma untuk mengurangi kompleksitas masalah dengan mengubah atau menyederhanakan instansi masalah yang diberikan menjadi instansi yang lebih mudah dipecahkan. Tujuan dari instance simplification adalah memperoleh pemahaman yang lebih baik tentang sifat masalah dan mencari solusi yang lebih efisien. Dengan menyederhanakan instansi masalah, kita dapat menghilangkan beberapa aspek yang tidak relevan atau memperkecil ukuran masalah sehingga memungkinkan penggunaan algoritma yang lebih efisien.

Proses instance simplification dapat melibatkan beberapa strategi, antara lain:

- Menghilangkan Informasi yang Tidak Relevan:
Kadang-kadang, instansi masalah mengandung informasi yang tidak diperlukan dalam pencarian solusi. Dalam hal ini, langkah pertama adalah mengidentifikasi informasi yang tidak relevan dan menghapusnya untuk mengurangi kompleksitas masalah.

- Menyederhanakan Kasus Khusus:
Beberapa masalah memiliki kasus khusus yang dapat disederhanakan secara terpisah. Dengan mengidentifikasi kasus khusus dan menyederhanakannya, kita dapat mengurangi kompleksitas keseluruhan masalah.

- Mengganti Representasi Masalah:
Dalam beberapa kasus, mengubah representasi masalah menjadi bentuk yang lebih sederhana atau lebih efisien dapat membantu dalam analisis dan pemecahan masalah. Representasi masalah yang baru mungkin lebih mudah dipahami atau lebih mudah dipecahkan menggunakan algoritma tertentu.

- Memperkecil Ukuran Instansi Masalah:
Jika masalah terdiri dari banyak entitas atau objek, mungkin memungkinkan untuk memperkecil ukuran masalah dengan mempertimbangkan hanya sebagian dari entitas tersebut atau dengan menggabungkan beberapa entitas yang setara menjadi satu entitas.

Penerapan instance simplification dapat membantu meningkatkan efisiensi dalam analisis algoritma dan pemecahan masalah. Dengan menyederhanakan instansi masalah, kita dapat mengurangi kompleksitas waktu atau ruang yang diperlukan untuk mencari solusi yang optimal atau memahami karakteristik masalah dengan lebih baik.

b. Representation Change

Representation Change (Perubahan Representasi) adalah teknik yang digunakan dalam analisis algoritma untuk mengubah cara data atau masalah direpresentasikan. Tujuan dari perubahan representasi adalah untuk memperoleh pemahaman yang lebih baik tentang masalah dan mencari solusi yang lebih efisien. Dalam beberapa kasus, representasi asli dari data atau masalah mungkin tidak cocok untuk penggunaan algoritma tertentu atau mungkin menghasilkan kompleksitas yang tinggi. Dalam hal ini, perubahan representasi dapat membantu memperkecil kompleksitas masalah atau membuatnya lebih mudah untuk dipecahkan.

Beberapa strategi yang dapat digunakan dalam perubahan representasi meliputi:

- Struktur Data yang Berbeda:

Mengubah struktur data yang digunakan untuk merepresentasikan masalah dapat memiliki dampak signifikan pada efisiensi algoritma. Misalnya, mengubah array menjadi struktur data seperti linked list, stack, atau queue dapat memungkinkan operasi yang lebih efisien.

- Encoding atau Komprimasi Data:

Jika masalah melibatkan representasi data yang berulang atau memiliki pola tertentu, encoding atau kompresi data dapat digunakan untuk mengurangi ukuran data yang diperlukan. Ini dapat mengurangi kompleksitas ruang yang diperlukan dan mempercepat operasi pada data yang dikodekan atau dikompresi.

- Representasi Matematis:

Dalam beberapa kasus, masalah dapat diubah menjadi bentuk matematis yang lebih sederhana atau lebih mudah untuk dianalisis. Misalnya, memodelkan masalah sebagai graf atau matriks dapat membantu menerapkan algoritma graf atau operasi matriks yang efisien.

- Representasi Berbasis Aturan:

Beberapa masalah kompleks dapat dipecahkan dengan merumuskan aturan atau konstrain yang lebih sederhana. Dengan memperumum masalah dan mengidentifikasi aturan yang membatasi solusi yang mungkin, kompleksitas masalah dapat berkurang.

Perubahan representasi dapat membantu dalam analisis algoritma, pemecahan masalah, dan optimisasi performa. Dengan memilih representasi yang lebih sesuai dan efisien, kita dapat mengurangi kompleksitas waktu dan ruang yang diperlukan untuk mencari solusi dan meningkatkan efisiensi algoritma yang digunakan.

c. Problem Reduction

Problem Reduction (Reduksi Masalah) adalah teknik yang digunakan dalam pemecahan masalah untuk mengurangi kompleksitas atau kesulitan suatu masalah dengan mengubahnya menjadi masalah yang lebih sederhana atau sudah diketahui solusinya.

Tujuan utama dari reduksi masalah adalah untuk mengidentifikasi hubungan antara masalah yang sulit dan masalah yang lebih mudah, sehingga solusi yang ada untuk masalah yang lebih mudah dapat diterapkan pada masalah yang sulit. Proses reduksi masalah melibatkan dua langkah utama:

1. Reduksi dari Masalah Target ke Masalah Referensi: Pertama, masalah yang sulit atau kompleks (masalah target) diubah menjadi masalah yang lebih sederhana atau sudah diketahui solusinya (masalah referensi). Ini dilakukan dengan mengidentifikasi kemiripan atau kesamaan antara masalah target dan masalah referensi. Dalam beberapa kasus, masalah referensi dapat menjadi varian khusus dari masalah target.

2. Penerapan Solusi Masalah Referensi ke Masalah Target: Setelah masalah target direduksi menjadi masalah referensi, solusi yang ada atau algoritma yang diketahui untuk masalah referensi dapat diterapkan pada masalah target. Ini memanfaatkan kesamaan atau korelasi antara dua masalah untuk menghasilkan solusi untuk masalah yang sulit.

Reduksi masalah adalah teknik yang penting dalam pemecahan masalah kompleks, dan digunakan dalam berbagai bidang seperti teori kompleksitas, kecerdasan buatan, dan optimisasi kombinatorial. Ini memungkinkan pemecahan masalah yang lebih efisien dan memperluas ruang solusi dengan memanfaatkan pengetahuan dan solusi yang sudah ada. Dalam beberapa kasus, reduksi masalah juga dapat digunakan untuk membuktikan sifat-sifat tertentu tentang kekerasan atau kesulitan suatu masalah.

D. DYNAMIC PROGRAMMING

a. Three Basic

Tiga teknik dasar dalam algoritma komputer adalah:

1.Sequential Search (Pencarian Berurutan):

Teknik ini melibatkan pencarian secara berurutan satu per satu dari awal hingga akhir elemen dalam rangkaian data. Pada setiap langkah, elemen yang sedang dicari dibandingkan dengan elemen saat ini dalam urutan. Jika ada kesamaan, elemen ditemukan. Jika tidak, pencarian berlanjut hingga seluruh rangkaian data diperiksa. Sequential search sederhana namun memiliki kompleksitas waktu linear O(n), di mana n adalah jumlah elemen dalam rangkaian data.

2.Binary Search (Pencarian Biner):

Teknik ini digunakan pada rangkaian data yang telah diurutkan. Pencarian dimulai dengan membandingkan elemen tengah dengan elemen yang sedang dicari. Jika elemen tengah adalah elemen yang dicari, pencarian selesai. Jika elemen tengah lebih besar dari elemen yang dicari, pencarian dilanjutkan pada setengah kiri dari rangkaian data. Jika elemen tengah lebih kecil, pencarian dilanjutkan pada setengah kanan. Binary search memiliki kompleksitas waktu logaritmik O(log n), di mana n adalah jumlah elemen dalam rangkaian data.

3.Hashing:

Teknik ini melibatkan penggunaan fungsi hash untuk mengonversi data menjadi nilai hash, yang kemudian digunakan sebagai kunci untuk penyimpanan, pencocokan, atau identifikasi data. Hashing memungkinkan akses langsung ke data dengan kompleksitas waktu konstan O(1) dalam kasus terbaiknya. Namun, ada kemungkinan tabrakan hash jika dua data menghasilkan nilai hash yang sama. Untuk mengatasi tabrakan hash, teknik seperti chaining atau open addressing dapat digunakan. Ketiga teknik dasar ini memiliki kelebihan dan kekurangan masing-masing. Pemilihan teknik yang tepat tergantung pada karakteristik data dan kebutuhan spesifik dalam pemrosesan data.

b. The Knapsack Problem and Memory Fungctions

Masalah Knapsack (Knapsack Problem) adalah sebuah masalah optimasi yang sering dihadapi dalam ilmu komputer dan matematika. Dalam masalah ini, terdapat sebuah knapsack (tas) dengan kapasitas terbatas dan sejumlah objek yang memiliki nilai dan bobot yang berbeda.

Tujuan adalah memilih kombinasi objek yang akan dimasukkan ke dalam knapsack sedemikian rupa sehingga nilai total objek yang dipilih maksimum, tetapi total bobotnya tidak melebihi kapasitas knapsack. Dalam konteks masalah knapsack, memory functions (fungsi memori) merujuk pada penggunaan penyimpanan sementara untuk menyimpan dan mengingat solusi-solusi submasalah yang telah dihitung.

Dalam pendekatan pemrograman dinamis untuk memecahkan masalah knapsack, memory functions digunakan untuk menghindari perhitungan berulang pada submasalah yang sama.

Memory functions dapat berupa struktur data seperti tabel atau matriks yang digunakan untuk menyimpan hasil perhitungan submasalah. Setiap entri dalam struktur data tersebut mewakili solusi terbaik yang ditemukan untuk submasalah tertentu. Dengan menggunakan memory functions, kita dapat mengakses solusi

submasalah yang telah dihitung sebelumnya secara efisien, tanpa perlu menghitung ulang.

c. Warshall’s and Floyd’s AlgorithmsWarshall's Algorithm (Algoritma Warshall)
dan Floyd's Algorithm (Algoritma Floyd) adalah dua algoritma yang digunakan dalam teori graf untuk menyelesaikan masalah jalur terpendek antara semua pasangan simpul (all-pairs shortest path).

1.Warshall's Algorithm:
Algoritma Warshall digunakan untuk menemukan jalur terpendek antara semua pasangan simpul dalam graf berbobot positif atau negatif (tetapi tanpa siklus negatif). Algoritma ini mengoperasikan matriks kedekatan (adjacency matrix) yang merepresentasikan graf.

Langkah-langkah utama dalam algoritma Warshall adalah sebagai berikut: - Inisialisasi matriks kedekatan dengan bobot langsung antara simpul-simpul yang terhubung secara langsung, dan mengisi nilai tak hingga (Infinity) untuk pasangan simpul yang tidak terhubung secara langsung. - Lakukan iterasi untuk semua simpul sebagai simpul tengah dan perbarui nilai bobot antara simpul-simpul dengan mempertimbangkan simpul tengah.

Jika bobot jalur baru lebih kecil dari bobot jalur sebelumnya, maka nilai bobot diperbarui. - Setelah semua iterasi selesai, matriks kedekatan akan berisi bobot jalur terpendek antara semua pasangan simpul.

2.Floyd's Algorithm:
Algoritma Floyd, juga dikenal sebagai Algoritma Floyd-Warshall, digunakan untuk menemukan jalur terpendek antara semua pasangan simpul dalam graf berbobot positif atau negatif (tetapi tanpa siklus negatif). Algoritma ini juga mengoperasikan matriks kedekatan yang merepresentasikan graf.

Langkah-langkah utama dalam algoritma Floyd adalah sebagai berikut:
- Inisialisasi matriks kedekatan dengan bobot langsung antara simpul-simpul yang terhubung secara langsung, dan mengisi nilai tak hingga (Infinity) untuk pasangan simpul yang tidak terhubung secara langsung.
- Lakukan iterasi untuk semua simpul sebagai simpul tengah dan perbarui nilai bobot antara simpul-simpul dengan mempertimbangkan simpul tengah. Jika bobot jalur baru lebih

kecil dari bobot jalur sebelumnya, maka nilai bobot diperbarui.
- Setelah semua iterasi selesai, matriks kedekatan akan berisi bobot jalur terpendek antara semua pasangan simpul.

Perbedaan utama antara Algoritma Warshall dan Algoritma Floyd terletak pada urutan iterasinya. Pada Algoritma Warshall, iterasi dilakukan pada semua pasangan simpul sebagai simpul tengah secara berurutan, sedangkan pada Algoritma Floyd, iterasi dilakukan pada semua simpul sebagai simpul tengah secara berurutan.
Algoritma Warshall memiliki kompleksitas waktu O(n^3) dan Algoritma Floyd juga memiliki kompleksitas waktu O(n^3), di mana n adalah jumlah simpul dalam graf. Kedua algoritma ini sangat berguna dalam menemukan jalur terpendek antara semua pasangan simpul dalam graf berbobot.