RESUME

“PERANCANGAN ANALISIS DAN ALGORITMA”

“Disusun sebagai tugas akhir pada mata kuliah Perancangan dan Analisis Algoritma , dengan Dosen Randi Proska Sandra, M.Sc dan Widya Darwin S.Pd., M.Pd.T”

Disusun Oleh :

Putri Maharani || 21346018

PROGRAM STUDI S1 TEKNIK INFORMATIKA

JURUSAN TEKNIK ELEKTRONIKA

FAKULTAS TEKNIK

UNIVERSITAS NEGERI PADANG

TAHUN 2023

KATA PENGANTAR

Puji syukur kita hanturkan kehadiran Tuhan Yang Maha Esa, yang telah melimpahkan rahmat dan karunianya kepada kita, sehingga kita dapat menyusun dan menyajikan makalah Perancangan dan Analisis Algoritma ini dengan tepat waktu. Tak lupa pula kita mengucapkan terima kasih kepada berbagai pihak yangtelah memberikan dorongan dan motivasi. Sehingga makalah ini dapat tersusun dengan baik.

Perancangan algoritma adalah proses mengembangkan rencana atau langkah-langkah yang sistematis untuk menyelesaikan suatu masalah. Hal ini melibatkan pemikiran kritis, kreativitas, dan pemahaman mendalam tentang struktur data serta teknik pemrograman. Dengan menggunakan algoritma yang efisien dan optimal, kita dapat meningkatkan kinerja sistem komputer, mengoptimalkan penggunaan sumber daya, dan mencapai solusi yang lebih baik.

Namun, hanya merancang algoritma saja tidaklah cukup. Analisis algoritma juga menjadi bagian penting dalam proses ini. Dalam analisis algoritma, kita mengevaluasi kinerja algoritma secara matematis, mengukur efisiensi waktu dan ruang yang dibutuhkan, serta memprediksi bagaimana algoritma tersebut akan berperilaku dalam skenario terburuk atau terbaik. Analisis algoritma memungkinkan kita untuk membandingkan algoritma yang berbeda dan memilih yang paling sesuai dengan kebutuhan kita.

Dengan mempelajari perancangan dan analisis algoritma, Anda akan mendapatkan keterampilan yang tak ternilai dalam pemrograman dan pemecahan masalah. Anda akan mampu mengoptimalkan algoritma yang ada, mengembangkan solusi yang efisien, dan meningkatkan performa sistem secara keseluruhan.

Saya berharap kata pengantar ini dapat memberikan gambaran yang jelas tentang pentingnya perancangan dan analisis algoritma dalam dunia komputer. Semoga dengan pemahaman yang mendalam tentang topik ini, Anda akan siap menghadapi tantangan pemrograman yang kompleks dan menjadi seorang yang ahli dalam merancang algoritma yang efisien.

Sago, 28 Mei 2023

Putri Maharani

DAFTAR ISI

Perancangan & Analisis Algoritma

BAB I
PENGANTAR ALGORITMA

A. Pengertian Algoritma

Algoritma adalah sekumpulan instruksi atau langkah sistematis yang digunakan untuk memecahkan masalah atau tugas dalam kalkulus atau matematika. Algoritma biasanya terdiri dari serangkaian langkah yang disusun secara logis dan dieksekusi dalam urutan tertentu untuk mencapai tujuan tertentu.

Secara umum, algoritma adalah panduan atau resep untuk memecahkan masalah secara efisien dan efektif. Algoritma dapat diterapkan di berbagai bidang, termasuk pemrograman komputer, ilmu data, matematika, dan sains.

Biasanya, agar dianggap baik, suatu algoritma harus memenuhi beberapa kriteria, seperti kejelasan, efisiensi, efektivitas, keunikan solusi, dan kemampuan mengoreksi. Dalam dunia pemrograman, algoritma merupakan dasar untuk membuat program dan aplikasi. Seorang programmer harus memiliki pemahaman yang baik tentang konsep algoritma untuk membuat program yang baik dan efisien.

B. Dasar Algoritma

Beberapa metode untuk merancang algoritma dalam program komputer :

1) Diagram Alir (Flow Chart)
2) Kode Semu (Pseudo Code)
3) Algoritma Fundamental

Knuth (1973) menyatakan 5 komponen utama dalam algoritma yaitu finiteness, definiteness, input, output dan effectiveness

Komponen yang harus ada dalam merancang algoritma:

1) Komponen masukan : terdiri dari pemilihan variable, jenis variable, tipe variable, konstanta dan parameter (dalam fungsi).
2) Komponen keluaran: merupakan tujuan dari perancangan algoritma dan program. Permasalahan yang diselesaikan dalam algoritma dan program harus ditampilkan dalam komponen keluaran. Karakteristik keluaran yang baik adalah menjawab permasalahan dan tampilan yang ramah
3) Komponen proses : merupakan bagian utama dan terpenting dalam merancang sebuah algoritma. Dalam bagian ini terdapat logika masalah, logika algoritma (sintaksis dan semantik), rumusan, metode (rekursi, perbandingan, penggabungan, pengurangan dll).

C. Problem Solving

Yaitu mengidentifikasi dan menemukan solusi yang efektif untuk memecahkan masalah. Pada dasarnya, keterampilan ini terkait dengan keterampilan lain seperti mendengarkan, analisis, penelitian, kreativitas, komunikasi, kerja tim, dan pengambilan keputusan.

Menurut Oemar Hamalik, pengertian pemecahan masalah adalah proses mental dan intelektual dalam menemukan masalah. Kemudian menyelesaikan masalah berdasarkan data dan informasi yang akurat untuk menarik kesimpulan secara cermat dan cepat. Ini sebenarnya sudah menjadi tujuan utama untuk menyelesaikan masalah ini.

Proses problem solving adalah langkah-langkah yang diambil untuk mengidentifikasi, menganalisis, dan menyelesaikan suatu masalah. Berikut adalah langkah-langkah umum dalam proses problem solving:

1) Identifikasi masalah: Langkah pertama adalah mengidentifikasi masalah dengan jelas. Pahami masalah yang perlu diselesaikan, pastikan pemahaman yang tepat tentang sifat masalah, dan tetapkan tujuan yang ingin dicapai.
2) Analisis masalah: Setelah masalah diidentifikasi, lakukan analisis lebih lanjut untuk memahami penyebab dan dampaknya. Identifikasi faktor-faktor yang berkontribusi terhadap masalah dan cari tahu informasi yang relevan untuk memperoleh pemahaman yang lebih baik tentang masalah tersebut.
3) Generasi solusi: Di tahap ini, hasilkan sebanyak mungkin solusi yang potensial untuk mengatasi masalah. Berpikirlah secara kreatif dan luangkan waktu untuk mempertimbangkan berbagai pendekatan yang berbeda. Jangan khawatir tentang kepraktisan atau kelayakan solusi pada tahap ini; tujuan utamanya adalah membangun keragaman ide.
4) Evaluasi dan pemilihan solusi: Setelah menghasilkan berbagai solusi, lakukan evaluasi kritis terhadap setiap solusi. Pertimbangkan kelebihan dan kekurangan masing-masing solusi, pertimbangkan keterkaitan dengan tujuan yang telah ditetapkan, dan tinjau kembali ketersediaan sumber daya yang diperlukan. Pilih solusi yang paling memadai dan cocok dengan masalah yang dihadapi.
5) Implementasi solusi: Setelah memilih solusi, langkah selanjutnya adalah mengimplementasikan solusi tersebut. Buat rencana tindakan yang jelas, tentukan langkah-langkah yang harus diambil, dan alokasikan sumber daya yang diperlukan. Pastikan komunikasi yang baik dan kolaborasi yang efektif dengan orang-orang yang terlibat dalam implementasi.
6) Evaluasi dan pengawasan: Setelah solusi diimplementasikan, lakukan evaluasi terhadap hasilnya. Tinjau apakah solusi berhasil menyelesaikan masalah atau mencapai tujuan yang telah ditetapkan. Jika perlu, lakukan penyesuaian atau perbaikan tambahan untuk memastikan efektivitas solusi.

Penting untuk diingat bahwa proses problem solving bersifat iteratif. Anda mungkin perlu melalui langkah-langkah ini beberapa kali untuk menemukan solusi yang optimal atau untuk menghadapi masalah baru yang muncul selama proses. Fleksibilitas, kreativitas, dan kemampuan untuk berpikir analitis sangat penting dalam proses ini.

D. Tujuan Metode Pembelajaran *Problem Solving*

Metode pembelajaran *problem solving* mengembangkan kemampuan berfikir yang dipupuk dengan adanya kesempatan untuk mengobservasi problema, mengumpulkan data, menganalisa data, menyusun suatu hipotesa, mencari hubungan (data) yang hilang dari data yang telah terkumpul untuk kemudian menarik kesimpulan yang merupakan hasil pemecahan masalah tersebut. Cara berfikir semacam itu lazim disebut cara berfikir ilmiah. Cara berfikir yang menghasilkan suatu kesimpulan atau keputusan yang diyakini kebenarannya karena seluruh proses pemecahan masalah itu telah diikuti dan dikontrol dari data yang pertama yang ¥berhasil dikumpulkan dan dianalisa sampai kepada kesimpulan yang ditarik atau ditetapkan.

Tujuan utama dari penggunaan metode pemecahan masalah adalah:

1) Mengembangkan kemampuan berfikir, terutama didalam mencari sebab-akibat dan tujuan suatu masalah. Metode ini melatih murid dalam cara-cara mendekati dan cara-cara mengambil langkah-langkah apabila akan memecahkan suatu masalah.
2) Memberikan kepada murid pengetahuan dan kecakapan praktis yang bernilai atau bermanfaat bagi keperluan hidup sehari-hari. Metode ini memberikan dasar-dasar pengalaman yang praktis mengenai bagaimana cara-cara memecahkan masalah dan kecakapan ini dapat diterapkan bagi keperluan menghadapi masalah-masalah lainnya didalam masyarakat.

Problem solving melatih siswa terlatih mencari informasi dan mengecek silang validitas informasi itu dengan sumber lainnya, juga *problem solving* melatih siswa berfikir kritis dan metode ini melatih siswa memecahkan dilema. Sehingga dengan menerapkan metode *problem solving* ini siswa menjadi lebih dapat mengerti bagaimana cara memecahkan masalah yang akan dihadapi pada kehidupan nyata atau di luar lingkungan sekolah.

Untuk mendukung strategi belajar mengajar dengan menggunakan metode *problem solving* ini, guru perlu memilih bahan pelajaran yang memiliki permasalahan. Materi pelajaran tidak terbatas hanya pada buku teks di sekolah, tetapi juga di ambil dari sumbersumber lingkungan seperti peristiwa-peristiwa kemasyarakatan atau peristiwa dalam lingkungan sekolah.[1] Tujuannya agar memudahkan siswa dalam menghadapi dan memecahkan masalah yang terjadi di lingkungan sebenarnya dan siswa memperoleh pengalaman tentang penyelesaian masalah sehingga dapat diterapkan di kehidupan nyata.

E. Langkah-langkah Metode Pembelajaran *Problem Solving*

Metode *problem solving* (metode pemecahan masalah) bukan hanya sekedar metode mengajar tetapi juga merupakan suatu metode berpikir, sebab dalam *problem solving* dapat menggunakan metode- metode lainnya dimulai dengan mencari data sampai kepada menarik kesimpulan. Langkah- langkah metode ini antara lain:

1) Adanya masalah yang jelas untuk dipecahkan. Masalah ini harus tumbuh dari siswa sesuai dengan taraf kemampuannya.
2) Mencari data atau keterangan yang dapat digunakan untuk memecahkan masalah tersebut. Misalnya, dengan jalan membaca buku- buku, meneliti, bertanya, berdiskusi, dan lain-lain.
3) Menetapkan jawaban sementara dari masalah tersebut. Dugaan jawaban ini tentu saja didasarkan kepada data yang telah diperoleh, pada langkah kedua diatas.
4) Menguji kebenaran jawaban sementara tersebut. Dalam langkah ini siswa harus berusaha memecahkan masalah sehingga betul-betul yakin bahwa jawaban tersebut betul-betul cocok. Apakah sesuai dengan jawaban sementara atau sama sekali tidak sesuai. Untuk menguji kebenaran jawaban ini tentu saja diperlukan metode-metode lainnya seperti, demonstrasi, tugas diskusi, dan lain-lain.

Menarik kesimpulan. Artinya siswa harus sampai kepada kesimpulan terakhir tentang jawaban dari masalah yang ada

BAB II
ANALISIS FRAMEWORK

A. Measuring an Input Size

Dalam analisis algoritma, penting untuk memahami bagaimana mengukur ukuran input algoritma. Ukuran input dapat bervariasi tergantung pada jenis masalah yang sedang diselesaikan, misalnya jumlah elemen dalam sebuah array atau jumlah simpul dalam sebuah graf. Memahami cara mengukur ukuran input membantu dalam menganalisis kompleksitas algoritma.

- Tujuan analisa : mengukur efesiensi algoritma
- Efisiensi diukur dari diukur dari: waktu (time) dan memori (space).
- Dua besaran yang digunakan: kompleksitas algoritma
  1. Kompleksitas waktu – T(n)
  2. Kompleksitas ruang – S(n)

Analisis Framework

- Mengukur ukuran atau jumlah input
- Mengukur waktu eksekusi
- Tingkat pertumbuhan
- Efiesiensi worst-case, best-case dan average-case

Measuring Input Space (n)
(Mengukur jumlah input)

- Ukuran masukan (n): jumlah data yang diproses oleh sebuah algoritma.
- Hampir semua algoritma membutuhkan waktu yang lebih lama untuk menyelesaikan jika inputannya lebih banyak
- Contoh
  - pengurutan 1000 elemen larik, maka n = 1000
  - pencarian elemen pada matriks 3 x 4, maka n=12
  - algoritma TSP pada sebuah graf lengkap dengan 100 simpul, maka n = 100.

- Dalam praktek perhitungan kompleksitas, ukuran masukan dinyatakan sebagai variabel n.

Measuring Running Time (Pengukuran Running Time)

- Dapat menggunakan satuan waktu standar (s/ms), tapi tergantung pada kecepatan komputer, kualitas program, compiler.

- Running time ditentukan oleh operasi dasar, yaitu operasi yang paling mendominasi sebagian besar running time algoritma. Operasi tersebut memakai waktu yang paling banyak dari keseluruhan running time.
- Operasi dasar dapat berupa:
  - penjumlahan/pengurangan
  - pengurangan
  - perkalian
  - pengaksesan memori
  - pembacaan
  - penulisan
- Menghitung jumlah waktu yang diperlukan untuk mengeksekusi operasi dasar.
- Framework yang telah ada untuk analisis efisiensi waktu suatu algoritma : menghitung jumlah waktu eksekusi operasi dasar dalam algoritma dengan inputan ukuran n.

Contoh operasi khas di dalam algoritma •

Algoritma pencarian di dalam larik

Operasi khas: perbandingan elemen larik

- Algoritma pengurutan

  Operasi khas: perbandingan elemen, pertukaran elemen

- Algoritma penjumlahan 2 buah matriks Operasi khas: penjumlahan

- Algoritma perkalian 2 buah matriks

  Operasi khas: perkalian dan penjumlahan

Contoh : Algoritma untuk mencari elemen terbesar di dalam sebuah larik (*array*) yang berukuran *n* elemen.

```
procedure CariElemenTerbesar(input a1, a2, ..., an : integer, output
maks : integer)
{ Mencari elemen terbesar dari sekumpulan elemen larik integer a1, a2,
..., an.
  Elemen terbesar akan disimpan di dalam maks.
  Masukan: a1, a2, ..., an
  Keluaran: maks (nilai terbesar)
}
Deklarasi
   k : integer

Algoritma
   maks ← a1
   k ← 2
   while k < n do
     if ak > maks then
        maks ← ak
     endif
     i ← i+1
   endwhile
   { k > n }
```

Kompleksitas waktu algoritma dihitung berdasarkan j perbandingan elemen larik (A[i] > maks).

Kompleksitas waktu CariElemenTerbesar: $T(n) = n$

Tingkat Pertumbuhan

- Untuk n dengan jumlah besar, yang diperhitungkan adalah fungsi tingkat pertumbuhan
- Abaikan pengali yang konstan • Exp : ½ n n, 50 n log n n log n
- Fungsi penting :
- $\log_2 n$, n, $n \log_2 n$, n^2 n^3 2^n n!

Order of Growth

n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	n!
10	3.3	10	3.3 x 10	10^2	10^3	10^3	3.6×10^6
10^2	6.6	10^2	6.6×10^2	10^4	10^6	1.3×10^{30}	9.3×10^{157}
10^3	10	10^3	1.0×10^4	10^6	10^9		
4		4	5	8	12		
10^5	17	10^5	1.7×10^6	10^{10}	10^{15}		
10^6	20	10^6	2.0×10^7	10^{12}	10^{18}		

Worst case, best case, average case

Kompleksitas waktu dibedakan atas tiga macam :

1. $T_{max}(n)$: kompleksitas waktu untuk kasus terburuk *(worst case),* kebutuhan waktu maksimum.
2. $T_{min}(n)$: kompleksitas waktu untuk kasus terbaik *(best case),* kebutuhan waktu minimum.
3. $T_{avg}(n)$: kompleksitas waktu untuk kasus rata-rata *(average case)* kebutuhan waktu secara rata-rata

Contoh 2. Algoritma *sequential search*.

procedure PencarianBeruntun(input $a_1, a_2, ..., a_n$: integer, x : integer, output idx integer
ketemu : boolean *{ bernilai true jika x ditemukan atau false jika x tidak ditemukan }*

Algoritma:
k←1 ketemu ← false while (k ≤ n) and (not ketemu) do

```
  if  ak  =  x  then
ketemu←true  else  k
← k + 1  endif  endwhile
 { k > n or ketemu }
  if ketemu then  { x ditemukan }  idx←k  else  idx← 0  { x
tidak ditemukan }  endif
```

Jumlah operasi perbandingan elemen tabel:

1. *Kasus terbaik*: ini terjadi bila $a_1 = x$.

   $$T_{\min}(n) = 1$$

2. *Kasus terburuk*: bila $a_n = x$ atau x tidak ditemukan.

   $$T_{\max}(n) = n$$

3. *Kasus rata-rata*: Jika x ditemukan pada posisi ke-j, maka operasi perbandingan (a_k = x)akan dieksekusi sebanyak j kali.

   $$T_{\text{avg}}(n) = \frac{(1+2+3+\ldots+n)}{n} = \frac{n(1+n)}{2n} = \frac{1}{2}(n+1)$$

Contoh 3. Algoritma pengurutan seleksi (*selection sort*).

procedure Urut(input/output $a_1, a_2, ..., a_n$: integer) **Deklarasi** i, j, imaks, temp : integer

Algoritma
for i←n downto 2 do *{ pass sebanyak n – 1 kali }* imaks←1 for j←2 to i do if $a_j > a_{imaks}$ then

imaks←j endif
endfor
{ pertukarkan a_{imaks} dengan a_i } temp←a_i
ai←aimaks a_{imaks}←temp endfor

(i) Jumlah operasi perbandingan elemen Untuk setiap *pass* ke-*i*,

$i = n \rightarrow$ jumlah perbandingan $= n - 1$

$i = n - 1 \rightarrow$ jumlah perbandingan $= n - 2$

$i = n - 2 \rightarrow$ jumlah perbandingan $= n - 3$

M

$i = 2 \rightarrow$ jumlah perbandingan = 1

Jumlah seluruh operasi perbandingan elemen-elemen larik adalah

$$T(n) = (n-1) + (n-2) + \ldots + 1 = \sum_{i=1}^{n-1} n-k = \frac{n(n-1)}{2}$$

Ini adalah kompleksitas waktu untuk kasus terbaik dan terburuk, karena algoritma Urut tidak bergantung pada batasan apakah data masukannya sudah terurut atau acak.

Latihan

• Contoh 4. Hitung kompleksitas waktu algoritma berikut berdasarkan jumlah operasi kali.

```
procedure Kali(input x:integer, n:integer, output jumlah : integer)
{Mengalikan x dengan i = 1, 2, …, j, yang dalam hal ini j = n, n/2, n/4, …,1
  Masukan: x dan n (n adalah perpangakatan dua).
  Keluaran: hasil perkalian (disimpan di dalam peubah jumlah).
}
Deklarasi
   i, j, k : integer

Algoritma
   j   n
   while  j  1 do
     for i   1 to j do
       x   x * i
     endfor
     j   j div 2
   endwhile
   { j > 1 }
   jumlah x
```


Jawaban

• Untuk

j = n, jumlah operasi perkalian = n j = n/2, jumlah operasi perkalian = n/2 j = n/4, jumlah operasi perkalian = n/4

…

j = 1, jumlah operasi perkalian = 1 Jumlah operasi perkalian seluruhnya adalah

$$= n + n/2 + n/4 + \ldots + 2 + 1 \qquad \text{deret geometri}$$

$$= \frac{n(1-2^{1+\log_2 n-1})}{1-\frac{1}{2}} = 2(n-1)$$

B. Unit of Measuring Running Time

Unit pengukuran waktu yang umum digunakan dalam analisis algoritma adalah langkah operasi dasar. Langkah operasi dasar adalah operasi paling dasar yang dilakukan dalam algoritma, seperti penjumlahan, perkalian, perbandingan, atau akses ke elemen dalam struktur data. Dengan menggunakan langkah operasi dasar sebagai unit pengukuran, kita dapat menganalisis berapa banyak langkah operasi dasar yang diperlukan oleh algoritma seiring pertumbuhan ukuran input.

C. Order of Growth

Order of Growth menggambarkan bagaimana waktu eksekusi algoritma meningkat seiring pertumbuhan ukuran input. Dalam analisis algoritma, kita tertarik untuk mengetahui bagaimana waktu eksekusi algoritma berubah saat ukuran inputnya membesar. Order of Growth biasanya dinyatakan menggunakan notasi Big O, yang memberikan batasan atas tentang kompleksitas waktu algoritma.

D. Worst-Case and Average-Case Efficiency

Ketika menganalisis algoritma, kita sering kali mempertimbangkan efisiensi terburuk dan rata-rata. Efisiensi terburuk adalah waktu eksekusi maksimum yang dapat terjadi pada algoritma untuk setiap ukuran input. Sedangkan efisiensi rata-rata adalah perkiraan waktu eksekusi yang diharapkan berdasarkan distribusi kemungkinan input yang mungkin terjadi. Dengan menganalisis efisiensi terburuk dan rata-rata, kita dapat memahami bagaimana algoritma berperilaku dalam berbagai situasi dan membuat keputusan yang lebih baik dalam pemilihan algoritma yang tepat.

Materi ini adalah beberapa aspek penting dalam analisis algoritma yang membantu dalam memahami kinerja dan kompleksitas algoritma saat menghadapi ukuran input yang berbeda.

BAB III
BRUTE FORCE DAN EXHAUSTIVE SEARCH

A. Selection Sort and Bubble Sort

Selection Sort dan Bubble Sort adalah contoh dari algoritma pengurutan yang sederhana. Selection Sort secara berulang memilih elemen terkecil dari sisa array yang belum diurutkan dan menukar posisinya dengan elemen pertama yang belum diurutkan. Bubble Sort secara berulang membandingkan pasangan elemen berturut-turut dalam array dan menukar posisinya jika diperlukan, sehingga elemen terbesar naik ke posisi yang benar.

B. Sequential Search and Brute-Force String Matching

Sequential Search adalah algoritma pencarian sederhana yang secara berurutan memeriksa setiap elemen dalam array atau daftar hingga menemukan elemen yang dicari. Brute-Force String Matching adalah algoritma pencarian pola dalam teks dengan memeriksa setiap posisi dalam teks secara berurutan untuk mencocokkan pola yang dicari.

C. Closest-Pair and Convex-Hull Problems

Closest-Pair Problem adalah masalah menemukan sepasang titik terdekat di dalam himpunan titik yang diberikan. Convex-Hull Problem adalah masalah menemukan himpunan titik terluar yang membentuk poligon konveks yang melingkupi semua titik dalam himpunan tersebut.

D. Exhaustive Search

Exhaustive Search adalah pendekatan algoritma yang secara sistematis menguji semua kemungkinan solusi yang mungkin untuk menemukan solusi optimal. Meskipun metode ini bisa efektif untuk masalah dengan ukuran input kecil, kompleksitasnya meningkat secara eksponensial dengan ukuran input, sehingga tidak efisien untuk masalah yang lebih besar.

E. Depth-First Search and Breadth-First Search

Depth-First Search (DFS) dan Breadth-First Search (BFS) adalah algoritma pencarian graf. DFS melakukan eksplorasi sejauh mungkin ke dalam graf sebelum mundur untuk melanjutkan ke cabang lain. BFS melakukan eksplorasi secara bertahap, mengunjungi semua tetangga dari simpul saat ini sebelum melanjutkan ke tetangga-tetangga berikutnya.

Materi ini mencakup berbagai algoritma brute force dan exhaustive search, serta beberapa contoh pencarian dan pengurutan sederhana. Anda dapat menggunakan materi ini untuk memahami konsep dasar dan implementasi dari masing-masing algoritma tersebut.

BAB IV
DECREASE AND CONQUER

A. Three Major Variants of Decrease-and-Conquer

Decrease-and-Conquer adalah strategi umum dalam perancangan algoritma di mana masalah awal dikurangi menjadi masalah yang lebih kecil, yang kemudian diselesaikan secara rekursif. Ada tiga variasi utama dari pendekatan Decrease-and-Conquer:

1) Decrease by a Constant Factor: Masalah awal dikurangi menjadi masalah yang berukuran konstan kali lebih kecil. Contohnya adalah algoritma binary search.
2) Decrease by a Variable Factor: Masalah awal dikurangi menjadi masalah yang berukuran variabel kali lebih kecil. Contohnya adalah algoritma merge sort.
3) Decrease by a Constant Amount: Masalah awal dikurangi dengan jumlah tetap pada setiap langkah. Contohnya adalah algoritma insertion sort.

B. Sort

Pengurutan (sorting) adalah proses mengatur elemen-elemen dalam suatu urutan tertentu. Algoritma pengurutan sangat penting dalam perancangan dan analisis algoritma. Beberapa algoritma pengurutan yang umum digunakan termasuk:

- Insertion Sort: Mengurutkan elemen satu per satu dengan membandingkan dan memasukkan elemen ke posisi yang benar.
- Selection Sort: Memilih elemen terkecil dari sisa array dan menukarnya dengan elemen pertama yang belum diurutkan.
- Merge Sort: Membagi array menjadi dua bagian, mengurutkan masing-masing bagian secara rekursif, dan kemudian menggabungkan kembali bagian-bagian yang terurut.
- Quick Sort: Memilih elemen pivot, mempartisi array menjadi dua bagian berdasarkan pivot, dan kemudian mengurutkan bagian-bagian tersebut secara rekursif.
- Heap Sort: Membangun heap dari array dan secara berulang menghapus elemen teratas heap untuk menghasilkan urutan yang diinginkan.

C. Topological Sorting

Topological Sorting adalah proses mengurutkan simpul-simpul dalam graf berarah sedemikian rupa sehingga tidak ada sisi yang mengarah ke simpul sebelumnya. Topological Sorting sering digunakan dalam pemodelan masalah yang melibatkan ketergantungan antara tugas-tugas yang harus diselesaikan. Algoritma umum untuk Topological Sorting adalah algoritma DFS (Depth-First Search) dan algoritma BFS (Breadth-First Search).

Materi ini mencakup konsep dasar Decrease-and-Conquer, algoritma pengurutan, dan Topological Sorting. Anda dapat menggunakan materi ini untuk memahami cara kerja dan implementasi algoritma-algoritma tersebut serta penerapannya dalam menyelesaikan berbagai masalah.

BAB V
DIVIDE AND CONQUER

A. Mergesort

Mergesort adalah algoritma pengurutan yang mengikuti pendekatan Divide and Conquer. Algoritma ini membagi array yang akan diurutkan menjadi dua bagian secara rekursif, kemudian mengurutkan masing-masing bagian tersebut, dan akhirnya menggabungkan bagian-bagian yang terurut untuk menghasilkan array yang diurutkan. Mergesort memiliki kompleksitas waktu rata-rata O(n log n) dan merupakan algoritma stabil.

B. Quicksort

Quicksort juga merupakan algoritma pengurutan yang menggunakan pendekatan Divide and Conquer. Algoritma ini memilih elemen pivot dari array dan membagi array menjadi dua bagian berdasarkan pivot. Bagian yang lebih kecil dari pivot ditempatkan sebelum pivot, sedangkan bagian yang lebih besar ditempatkan setelah pivot. Langkah ini diulang secara rekursif pada setiap bagian hingga seluruh array terurut. Quicksort memiliki kompleksitas waktu rata-rata O(n log n), tetapi dapat mencapai O(n^2) dalam kasus terburuk.

C. Binary Tree Traversals and Related Properties

Binary Tree Traversals adalah proses mengunjungi setiap simpul dalam pohon biner. Ada tiga jenis utama dari Binary Tree Traversals:

- Inorder Traversal: Mengunjungi simpul kiri, kemudian simpul itu sendiri, dan akhirnya simpul kanan.
- Preorder Traversal: Mengunjungi simpul itu sendiri, kemudian simpul kiri, dan akhirnya simpul kanan.
- Postorder Traversal: Mengunjungi simpul kiri, kemudian simpul kanan, dan akhirnya simpul itu sendiri.
- Selain itu, terdapat juga beberapa properti yang terkait dengan Binary Tree, seperti:
- Height: Ketinggian (jumlah tingkatan) dari pohon biner.
- Depth: Jarak dari simpul tertentu ke akar pohon.
- Balance: Sejauh mana pohon biner seimbang dalam hal jumlah simpul di setiap anak cabang.
- Complete Binary Tree: Pohon biner di mana semua tingkatan kecuali mungkin terakhir, terisi penuh.

Materi ini mencakup konsep dasar dan implementasi dari algoritma Divide and Conquer seperti Mergesort dan Quicksort, serta Binary Tree Traversals dan properti terkait. Anda dapat menggunakan materi ini untuk memahami cara kerja dan implementasi algoritma-algoritma tersebut serta konsep terkait pohon biner.

BAB VI
TRANSFORM AND CONQUER

A. Instance Simplification

Instance Simplification adalah salah satu pendekatan dalam Transform and Conquer yang melibatkan penyederhanaan atau pengurangan kasus-kasus khusus dari suatu masalah yang lebih kompleks. Dalam pendekatan ini, masalah awal yang sulit dipecahkan diubah menjadi kasus-kasus yang lebih sederhana yang dapat dipecahkan dengan mudah. Misalnya, dalam masalah Traveling Salesman, jika semua jarak antar kota adalah sama, maka masalahnya menjadi lebih sederhana.

B. Representation Change

Representation Change adalah pendekatan dalam Transform and Conquer yang melibatkan perubahan representasi data atau masalah ke bentuk yang lebih mudah untuk dikerjakan atau dipecahkan. Dengan mengubah representasi, masalah yang kompleks dapat diubah menjadi masalah yang lebih mudah dikelola atau dipecahkan. Misalnya, dalam masalah penjadwalan, representasi waktu dapat diubah menjadi graf yang memudahkan analisis dan pemecahan masalah.

C. Problem Reduction

Problem Reduction adalah pendekatan dalam Transform and Conquer yang melibatkan mengurangi atau mentransformasikan suatu masalah yang sulit menjadi beberapa masalah yang lebih sederhana yang telah diketahui cara menyelesaikannya. Dalam pendekatan ini, masalah sulit dipecahkan dengan cara memecahkan masalah-masalah yang lebih kecil atau menggunakan algoritma yang telah dikembangkan sebelumnya untuk masalah yang serupa. Contohnya adalah menggunakan algoritma sorting untuk memecahkan masalah pencarian atau penggabungan.

Materi ini mencakup konsep Transform and Conquer serta tiga pendekatan utamanya: Instance Simplification, Representation Change, dan Problem Reduction. Anda dapat menggunakan materi ini untuk memahami cara mengubah masalah yang kompleks menjadi masalah yang lebih sederhana dan lebih mudah dipecahkan atau dikerjakan.

BAB VII
SPACE AND TIME TRADE-OFFS

A. Sorting by Counting

Sorting by Counting adalah sebuah teknik dalam Trade-Off antara ruang dan waktu yang digunakan dalam pengurutan elemen-elemen. Teknik ini efektif ketika rentang nilai elemen yang diurutkan terbatas. Algoritma ini bekerja dengan menghitung jumlah kemunculan setiap elemen dalam rentang nilai yang diberikan, kemudian menggunakan informasi tersebut untuk membangun kembali array yang terurut. Kelebihan dari Sorting by Counting adalah kompleksitas waktu yang linier, tetapi kelemahannya adalah penggunaan memori yang bergantung pada rentang nilai.

B. Input Enhancement in String Matching

Input Enhancement adalah sebuah teknik dalam Trade-Off yang bertujuan untuk meningkatkan efisiensi algoritma dengan memodifikasi inputnya. Dalam konteks String Matching, teknik ini dapat digunakan untuk meningkatkan kinerja algoritma pencocokan string dengan melakukan pra-pemrosesan pada pola dan teks yang akan dicocokkan. Misalnya, menggunakan struktur data seperti tabel hash atau tabel indeks untuk mempercepat pencocokan pola dengan teks.

C. Hashing

Hashing adalah teknik dalam Trade-Off yang digunakan untuk mengubah atau mengkonversi data menjadi angka (hash) yang unik dan representatif. Teknik ini digunakan dalam banyak aplikasi seperti penyimpanan dan pencarian data, pengamanan, dan pengindeksan. Dalam konteks Trade-Off, penggunaan hashing memungkinkan akses data yang cepat dengan memperkenalkan ruang tambahan untuk menyimpan hash dan memperkecil waktu pencarian dengan mengurangi jumlah data yang harus dibandingkan secara langsung.

Materi ini mencakup konsep dasar tentang Trade-Off antara ruang dan waktu dalam algoritma dan tiga teknik yang terkait: Sorting by Counting, Input Enhancement dalam String Matching, dan Hashing. Anda dapat menggunakan materi ini untuk memahami prinsip-prinsip dasar, penerapan, dan manfaat dari Trade-Off dalam konteks pengurutan, pencocokan string, dan pengindeksan data.

BAB VIII
DYNAMIC PROGRAMMING

A. Three Basic

Dynamic Programming (Pemrograman Dinamis) adalah teknik algoritma yang digunakan untuk memecahkan masalah dengan membaginya menjadi submasalah yang lebih kecil, menyelesaikan submasalah tersebut secara rekursif, dan menyimpan solusinya untuk digunakan kembali dalam mengatasi masalah yang lebih besar. Ada tiga elemen dasar dalam dynamic programming: 1) Pemecahan masalah menjadi submasalah yang lebih kecil, 2) Solusi optimal untuk setiap submasalah, 3) Penggabungan solusi submasalah untuk mendapatkan solusi optimal dari masalah asli.

B. The Knapsack Problem and Memory Functions

Masalah Knapsack adalah salah satu contoh yang umum digunakan dalam dynamic programming. Dalam masalah ini, terdapat sejumlah item dengan bobot dan nilai tertentu, serta batasan kapasitas pada sebuah knapsack (tas). Tujuan adalah memilih kombinasi item yang memiliki nilai maksimum tanpa melampaui kapasitas knapsack. Dynamic programming dapat digunakan untuk menyelesaikan masalah Knapsack dengan menggunakan fungsi memoisasi (memory functions) untuk menyimpan hasil komputasi submasalah agar dapat digunakan kembali.

C. Warshall's and Floyd's Algorithms

Algoritma Warshall dan Floyd adalah dua algoritma terkenal yang menggunakan teknik dynamic programming untuk memecahkan masalah tertentu.

Algoritma Warshall digunakan untuk mencari jalur terpendek antara semua pasangan simpul dalam suatu graf berbobot. Dalam konteks ini, dynamic programming digunakan untuk memperbarui matriks jarak antara simpul-simpul secara iteratif.

Algoritma Floyd, juga dikenal sebagai algoritma Floyd-Warshall, juga digunakan untuk mencari jalur terpendek antara semua pasangan simpul dalam suatu graf berbobot. Algoritma ini berbeda dengan Warshall dalam pendekatan penggunaan tabel jarak.

Materi ini mencakup konsep dasar dynamic programming dan tiga topik yang terkait: Three Basic, The Knapsack Problem and Memory Functions, serta Warshall's dan Floyd's Algorithms. Anda dapat menggunakan materi ini untuk memahami prinsip-prinsip dynamic programming, penerapan dalam masalah Knapsack, dan algoritma Warshall dan Floyd dalam konteks mencari jalur terpendek dalam graf berbobot.

BAB IX
GREEDY TECHNIQUE

A. Prim's Algorithm

Prim's Algorithm adalah algoritma Greedy yang digunakan untuk mencari Minimum Spanning Tree (MST) dalam sebuah graf berbobot. Algoritma ini dimulai dengan memilih satu simpul acak sebagai simpul awal, kemudian secara berulang memilih sisi dengan bobot terendah yang terhubung dengan simpul yang sudah ada dalam MST yang sedang dibangun. Langkah ini diulang hingga semua simpul terhubung dalam MST.

B. Kruskal's Algorithm

Kruskal's Algorithm juga merupakan algoritma Greedy untuk mencari Minimum Spanning Tree (MST) dalam sebuah graf berbobot. Algoritma ini memulai dengan mengurutkan semua sisinya berdasarkan bobotnya secara menaik. Kemudian, algoritma ini memilih sisi dengan bobot terkecil dan memasukkannya ke dalam MST jika tidak menyebabkan siklus. Langkah ini diulang hingga MST terbentuk dengan semua simpul terhubung.

C. Dijkstra's Algorithm

Dijkstra's Algorithm adalah algoritma Greedy yang digunakan untuk mencari jalur terpendek antara dua simpul dalam sebuah graf berbobot, asalkan tidak ada bobot negatif dalam graf tersebut. Algoritma ini dimulai dengan menginisialisasi jarak awal dari simpul awal ke semua simpul lainnya dengan jarak tak terhingga. Selanjutnya, algoritma ini secara berulang memilih simpul dengan jarak terpendek yang belum dikunjungi dan memperbarui jarak dari simpul-simpul terhubung. Langkah ini diulang hingga jarak terpendek ke semua simpul tercapai.

D. Huffman Trees and Codes

Huffman Trees dan Codes adalah metode kompresi data yang menggunakan algoritma Greedy untuk membangun sebuah pohon biner khusus yang disebut Huffman Tree. Pohon Huffman ini digunakan untuk menghasilkan kode biner variabel untuk setiap karakter dalam suatu teks berdasarkan frekuensinya. Karakter yang muncul lebih sering akan memiliki kode biner yang lebih pendek. Dengan demikian, metode Huffman dapat menghasilkan kompresi data dengan memanfaatkan kemunculan karakter dalam teks.

Materi ini mencakup konsep dasar Greedy Technique dan empat topik yang terkait: Prim's Algorithm, Kruskal's Algorithm, Dijkstra's Algorithm, dan Huffman Trees and Codes. Anda dapat menggunakan materi ini untuk memahami prinsip-prinsip Greedy Technique, penerapan dalam mencari Minimum Spanning Tree, mencari jalur terpendek, serta kompresi data menggunakan Huffman Trees dan Codes.

BAB X
ITERATIVE IMPROVEMENT

A. The Simplex Method

The Simplex Method adalah sebuah algoritma iteratif yang digunakan untuk memecahkan masalah optimasi linier. Algoritma ini mencari solusi optimal dari sebuah fungsi objektif linier dengan mengiterasi melalui sekumpulan solusi yang memenuhi batasan-batasan linier yang diberikan. Pada setiap iterasi, algoritma mencari arah perbaikan yang mengarah ke solusi yang lebih baik, dan secara bertahap mendekati solusi optimal.

B. The Maximum-Flow Problem

The Maximum-Flow Problem adalah masalah yang melibatkan mencari aliran maksimum dalam sebuah jaringan terarah dengan kapasitas pada setiap jalur. Tujuan dari masalah ini adalah untuk menentukan jumlah maksimum aliran yang dapat mengalir dari sumber ke tujuan dalam jaringan, dengan mematuhi batasan kapasitas pada setiap jalur. Algoritma iteratif digunakan untuk menemukan aliran maksimum dengan mencari jalan yang memungkinkan aliran lebih besar pada setiap iterasi.

C. Maximum Matching in Bipartite Graphs

Maximum Matching in Bipartite Graphs adalah masalah mencari himpunan tepi terbesar (matching) dalam sebuah graf bipartit, di mana simpul-simpulnya dapat dibagi menjadi dua himpunan terpisah. Tujuan dari masalah ini adalah untuk menemukan jumlah maksimum pasangan simpul yang saling terhubung di antara kedua himpunan simpul. Algoritma iteratif digunakan untuk menemukan matching maksimum dengan memperluas matching pada setiap iterasi.

D. The Stable Marriage Problem

The Stable Marriage Problem adalah masalah pencocokan pasangan yang stabil di antara dua himpunan orang dengan preferensi masing-masing. Tujuan dari masalah ini adalah untuk menemukan pencocokan pasangan yang tidak ada pasangan yang saling lebih memilih satu sama lain daripada pasangan mereka. Algoritma iteratif digunakan untuk mencari pencocokan stabil dengan mengajukan proposal dan melakukan perbaikan dalam setiap iterasi hingga pencocokan yang stabil ditemukan.

Materi ini mencakup konsep dasar tentang Iterative Improvement dan empat topik yang terkait: The Simplex Method, The Maximum-Flow Problem, Maximum Matching in Bipartite Graphs, dan The Stable Marriage Problem. Anda dapat menggunakan materi ini untuk memahami prinsip-prinsip Iterative Improvement, penerapan dalam pemecahan masalah optimasi linier, jaringan aliran maksimum, pencocokan maksimum dalam graf bipartit, dan pencocokan stabil dalam masalah pencocokan pasangan.

BAB XI
LIMITATIONS OF ALGORITHM POWER

A. Lower-Bound Arguments

Lower-Bound Arguments (Argumen Batas Bawah) adalah pendekatan dalam analisis algoritma yang bertujuan untuk menentukan batas bawah dari kompleksitas waktu atau ruang yang diperlukan oleh suatu masalah. Dengan menggunakan teknik matematis dan argumen logika, argumen batas bawah digunakan untuk menunjukkan bahwa tidak ada algoritma yang dapat memecahkan masalah tersebut lebih efisien dari batas yang ditentukan.

B. Decision Tree

Decision Tree (Pohon Keputusan) adalah struktur data yang digunakan dalam analisis algoritma untuk memodelkan keputusan yang diambil dalam algoritma. Pohon keputusan digunakan untuk menggambarkan serangkaian keputusan yang diambil berdasarkan kondisi-kondisi tertentu, dan hasil akhir yang dicapai. Analisis decision tree dapat memberikan wawasan tentang kompleksitas waktu atau ruang algoritma.

C. P, NP, and NP-Complete Problems

P, NP, dan NP-Complete adalah kelas-kelas masalah dalam teori kompleksitas algoritma. P adalah kelas masalah yang dapat diselesaikan dalam waktu polinomial, sedangkan NP adalah kelas masalah yang dapat diverifikasi dalam waktu polinomial. NP-Complete adalah kelas masalah yang merupakan yang paling sulit di dalam kelas NP, di mana jika ada algoritma efisien untuk memecahkan satu masalah NP-Complete, maka akan ada algoritma efisien untuk memecahkan semua masalah NP-Complete.

D. Challenge of Numerical Algorithms

Tantangan dalam algoritma numerik melibatkan pemecahan masalah yang melibatkan operasi matematika pada angka-angka real atau kompleks. Algoritma numerik seringkali berhadapan dengan kesalahan pembulatan, kestabilan numerik, dan kompleksitas komputasional yang tinggi. Tantangan ini memerlukan pendekatan khusus dalam perancangan dan analisis algoritma numerik.

Materi ini mencakup batasan-batasan kekuatan algoritma dan empat topik yang terkait: Lower-Bound Arguments, Decision Tree, P, NP, dan NP-Complete Problems, serta tantangan dalam algoritma numerik. Anda dapat menggunakan materi ini untuk memahami batasan dalam perancangan algoritma, kompleksitas masalah, kelas masalah yang terkait dengan NP-Complete, serta tantangan khusus dalam algoritma numerik.

BAB XII
COPING WITH THE LIMITATION OF ALGORITHM POWER

A. Backtracking

Backtracking adalah sebuah teknik dalam algoritma yang digunakan untuk mencari solusi melalui eksplorasi secara sistematis pada ruang pencarian solusi. Algoritma backtracking mencoba solusi secara bertahap, dan jika pada suatu titik tidak memungkinkan untuk mencapai solusi yang benar, maka algoritma akan mundur (backtrack) ke langkah sebelumnya untuk mencoba alternatif lain. Backtracking sering digunakan untuk memecahkan masalah kombinatorial seperti penyelesaian papan catur atau pencarian jalur dalam graf.

B. Branch and Bound

Branch and Bound adalah sebuah teknik dalam algoritma yang digunakan untuk memecahkan masalah optimasi dengan menggabungkan eksplorasi ruang pencarian dengan batasan-batasan (bounds) pada solusi yang ada. Algoritma branch and bound menciptakan sebuah pohon pencarian di mana setiap simpul mewakili suatu titik dalam ruang pencarian. Algoritma tersebut membagi ruang pencarian menjadi bagian-bagian yang lebih kecil (branching) dan menghitung batasan-batasan solusi untuk setiap bagian. Dengan menggunakan batasan-batasan tersebut, algoritma dapat memotong (pruning) bagian-bagian ruang pencarian yang tidak perlu lagi dieksplorasi.

C. Algorithms for Solving Nonlinear Problems

Algoritma untuk memecahkan masalah nonlinear digunakan untuk menemukan solusi numerik dari persamaan atau sistem persamaan nonlinear. Persamaan nonlinear tidak memiliki solusi yang dapat ditemukan dengan metode aljabar, sehingga diperlukan algoritma numerik yang menggunakan pendekatan iteratif. Algoritma-algoritma seperti metode Newton-Raphson, metode iterasi sederhana, atau metode interpolasi digunakan untuk mencari solusi pendekatan yang mendekati solusi yang sebenarnya.

Materi ini mencakup cara mengatasi batasan kekuatan algoritma dengan tiga topik yang terkait: Backtracking, Branch and Bound, serta algoritma-algoritma untuk memecahkan masalah nonlinear. Anda dapat menggunakan materi ini untuk memahami prinsip-prinsip dalam mengatasi batasan algoritma, teknik backtracking dan branch and bound dalam mencari solusi, serta algoritma-algoritma numerik untuk masalah nonlinear.

KESIMPULAN

Berdasarkan materi yang telah disebutkan, berikut adalah beberapa kesimpulan dari bab-bab perancangan dan analisis algoritma yang telah dipelajari:

Pengantar Analisis Algoritma:

- Algoritma merupakan langkah-langkah atau instruksi-instruksi yang digunakan untuk memecahkan suatu masalah.
- Analisis algoritma penting untuk memahami efisiensi dan kinerja algoritma.
- Analisis Framework:
- Measuring an Input Size membahas tentang cara mengukur ukuran input dalam algoritma.
- Unit of Measuring Running Time digunakan untuk mengukur waktu yang dibutuhkan oleh algoritma.
- Order of Growth digunakan untuk membandingkan kinerja algoritma ketika ukuran inputnya meningkat secara eksponensial.
- Worst-Case, dan Average-Case Efficiency membahas tentang kasus terburuk dan kasus rata-rata dalam analisis algoritma.
- Brute Force dan Exhaustive Search:
- Metode brute force dan exhaustive search digunakan untuk mencari solusi secara sistematis dengan mencoba semua kemungkinan.
- Selection Sort, Bubble Sort, Sequential Search, dan Brute-Force String Matching adalah contoh-contoh algoritma brute force.
- Decrease and Conquer:
- Decrease and Conquer adalah pendekatan yang memecahkan masalah dengan mereduksi ukuran masalah menjadi lebih kecil dan menyelesaikan masalah tersebut.
- Sort dan Topological Sorting adalah contoh-contoh algoritma yang menggunakan pendekatan decrease and conquer.
- Divide and Conquer:
- Divide and Conquer adalah pendekatan yang memecahkan masalah dengan membagi masalah menjadi beberapa submasalah yang lebih kecil, menyelesaikan masing-masing submasalah secara rekursif, dan menggabungkan solusi-solusi submasalah untuk mendapatkan solusi akhir.
- Mergesort, Quicksort, dan Binary Tree Traversals adalah contoh-contoh algoritma divide and conquer.
- Transform and Conquer:
- Transform and Conquer adalah pendekatan yang memecahkan masalah dengan mentransformasi masalah menjadi bentuk yang lebih mudah dan kemudian menyelesaikannya.
- Instance Simplification, Representation Change, dan Problem Reduction adalah contoh-contoh pendekatan transform and conquer.
- Space and Time Trade-Offs:

- Space and Time Trade-Offs membahas tentang kompromi antara penggunaan ruang (memori) dan waktu dalam algoritma.
- Sorting by Counting, Input Enchantment in String Matching, dan Hashing adalah contoh-contoh trade-off antara ruang dan waktu dalam algoritma.
- Dynamic Programming:
- Dynamic Programming adalah teknik perancangan algoritma yang menggunakan pendekatan pemecahan masalah yang lebih besar dengan memecahnya menjadi submasalah yang lebih kecil.
- Three Basic, The Knapsack Problem and Memory Functions, Warshall's dan Floyd's Algorithms adalah contoh-contoh algoritma yang menggunakan teknik dynamic programming.
- Greedy Technique: Greedy Technique adalah pendekatan perancangan algoritma yang memilih langkah terbaik secara lokal pada setiap langkah untuk mencapai solusi yang optimal secara keseluruhan.
- Prim's Algorithm, Kruskal's Algorithm, Dijkstra's Algorithm, dan Huffman Trees