PERTEMUAN 11

TEKNIK QUICK SORT

A. TUJUAN PEMBELAJARAN

Setelah mempelajari materi pertemuan ini diharapkan mahasiswa mengerti dan paham teknik merge sort, logika dan teknik divide dan conquer pada quick sort dan mempu membuat program quick sort dengan python

B. URAIAN MATERI

1. Quick Sort

Dalam arti tertentu, algoritme quicksort adalah kebalikan dari algoritme pengurutan gabungan. Algoritma ini juga yang paling banyak digunakan dan salah satu algoritma penyortiran paling efisien yang dikenal. Quicksort sekali lagi merupakan algoritma divide and conquer dan biasanya ditulis secara rekursif. Tapi, di mana merge sort memisahkan list sampai kita mencapai ukuran 1 dan kemudian merge sort list, algoritma quicksort melakukan penggabungan terlebih dahulu, lalu memisahkan list. Kami tidak bisa menggabungkan list yang tidak disortir. Sebagai gantinya, kami mempartisi menjadi dua list. Apa yang kita inginkan adalah mempersiapkan listnya agar quicksort dapat dipanggil secara rekursif. Tapi, jika kita ingin sukses, kedua sublist harus lebih mudah diurutkan daripada yang asli. Persiapan ini untuk pemisahan disebut pemartisian. Untuk mempartisi list kami memilih elemen pivot. Pikirkan quicksort mempartisi list menjadi semua item yang lebih besar dari pivot dan semua elemen lebih kecil dari poros. Kami meletakkan semua barang yang lebih besar di sebelah kanan poros dan semua item yang lebih kecil di sebelah kiri poros. Setelah kita melakukan ini, ada dua hal yang benar:

a. Poros berada di lokasi akhirnya dalam list.

b. Kedua sublist sekarang lebih kecil dan oleh karena itu dapat diurutkan dengan cepat.

Begitu keduanya sublist diurutkan, ini akan menyebabkan seluruh list menjadi dalam urutan terurut, karena kiri akan menjadi nilai yang naik ke poros, poros

berada di tempat yang benar, dan nilai yang lebih besar dari pivot semuanya akan berada di lokasi yang benar juga.

Quicksort adalah algoritma bagi dan taklukkan. Untuk mendapatkan performa terbaik, kami ingin membagi urutan tepat di tengah menjadi dua berukuran sama urutan. Ini berarti bahwa kita harus memilih nilai persis ditengah menjadi poros karena poros digunakan untuk membagi dua list. Sayangnya ini tidak mungkin jika kita melakukannya dengan efisien. Untuk quicksort memiliki kompleksitas O (n log n), likemerge sort, itu harus mempartisi list dalam waktu O (n). Kita harus memilih pivot cepat dan kita harus memilihnya dengan baik. Jika kita tidak memilih poros yang dekat dengan tengah, kita tidak akan mendapatkan kompleksitas O (n log n) yang kita harapkan. Ternyata memilih sebuah poros acak dari list sudah cukup baik. Salah satu cara untuk menjamin pilihan acak pivotnya adalah membuat algoritme quicksort dimulai dengan mengacak urutannya.

Gambar 11. 1 Ilustrasi Quick Sort

Quick Sort dimana algoritma pengurutan datanya menggunakan teknik pemecahan data menjadi partisi-partisi, sehingga metode ini disebut juga dengan nama partition exchange sort. Untuk memulai irterasi pengurutan, pertama-tama sebuah elemen dipilih dari data, kemudian elemen-elemen data

akan diurutkan diatur sedemikian rupa, sehingga nilai variabel Sementara berada di suatu posisi ke I yang memenuhi kondisi sebagai berikut :

a. Semua elemen di posisi ke 1 sampai dengan ke I-1 adalah lebih kecil atau sama dengan Sementara.

b. Semua elemen di posisi ke I+1 sampai dengan ke N adalah lebih besar atau sama dengan Sementara.

2. Logika Quick Sort

Algoritma tipe cepat adalah algoritma penyortiran di tempat tanpa perlu ruang ekstra atau array tambahan karena semua operasi akan dilakukan dalam daftar yang sama, karena kami membagi daftar yang diberikan menjadi tiga bagian sebagai elemen pivot, elemen kurang dari pivot sebagai satu sub-daftar dan elemen yang lebih besar dari pivot sebagai sub-daftar lain. Ini juga disebut sebagai jenis pertukaran partisi. Quicksort adalah algoritma penyortiran yang lebih baik dan, di sebagian besar bahasa pemrograman, tersedia sebagai algoritma penyortiran bawaan.

Diberikan di bawah ini adalah langkah-langkah utama untuk logika implementasi tipe cepat:

a. Pertama, kita akan memilih elemen pivot dari daftar yang diberikan.

   1) Kita dapat memilih elemen pivot dengan cara yang berbeda seperti di bawah ini:

   2) Kita dapat memilih elemen pertama dari daftar sebagai pivot.

   3) Kita dapat memilih elemen terakhir dari daftar sebagai pivot.

   4) Kita dapat memilih beberapa elemen acak dari daftar sebagai pivot.

   5) Akhirnya, kita dapat memilih median elemen daftar sebagai pivot.

b. Dalam partisi, kami akan mengatur ulang daftar sedemikian rupa sehingga semua elemen daftar yang kurang dari pivot akan berada di sisi kiri, dan semua elemen daftar yang lebih besar dari pivot akan berada di sisi kanan, dan elemen yang sama akan berada di salah satu sisi pivot. Proses ini disebut partisi. Setelah akhir proses partisi, elemen pivot akan berada di posisi akhir pada daftar.

c. Kami akan menerapkan langkah-langkah di atas secara rekursif pada kedua sub-array sampai semua sub-array diurutkan.

Contoh 1 penggambaran logika sebagai berikut:

Ambil daftar dengan 6 elemen sebagai 9 7 15 10 2 5. Di sini akan mengambil elemen pertama dari daftar sebagai elemen pivot dan memulai daftar dan mengakhiri daftar.

Langkah 1: Pivot = 9 mulai = 9 ujung = 5

9 7 15 10 2 5

Sekarang kita akan memanggil proses partisi pada daftar yang diberikan, dan kita akan mendapatkan daftar yang diatur ulang dengan elemen pivot berada di posisi akhir seperti di bawah ini:

5 7 2 9 15 10

Seperti yang kita lihat elemen pivot berada dalam posisi terurutkan terakhirnya. Sekarang kita akan menerapkan langkah-langkah yang sama ke sub-daftar kiri dan kanan elemen pivot.

Langkah 2: pivot = 5 start = 5 end =2 (sub-list kiri)

2 5 7

pivot = 15 start = 15 end = 10 (sub-list kanan)

10 15

Pada langkah di atas, kami telah memanggil metode partisi pada sub-daftar kiri dan kanan, dan kami mengatur ulang seperti di bawah ini:

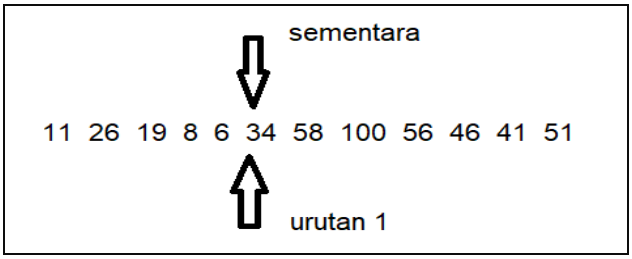
2 5 7 9 10 15

Jika mengamati daftar yang kita dapatkan di langkah di atas, semua elemen berada dalam posisi yang diurutkan. Dengan mengikuti logika di atas, kita dapat menerapkan jenis cepat, dan ini adalah salah satu cara untuk menerapkan jenis cepat dengan kompleksitas waktu kasus rata-rata O (NlogN) dan kompleksitas waktu terburuk menjadi O (n2). Jika kita mengamati hal di atas, penyortiran terjadi di tempat tanpa menggunakan ruang ekstra.

Sebagai contoh 2, data yang akan diurutkan sejumlah 12 elemen sebagai berikut.

34 46 19 8 6 100 58 26 56 11 41 51

Misalnya element yang dipilih adalah element yang pertama, maka variabel Sementara bernilai 34. Setelah diatur, maka nilai 34 akan menempati posisi ke I, yaitu posisi urutan ke 6 sebagai berikut :



Tampak bahwa kondisi berikut terpenuhi, yaitu :

a. Semua elemen di posisi ke 1 sampai dengan posisi ke 5 (11, 26, 19, 8,dan 6) akan lebih kecil atau sama dengan nilai 34 yang dipilih.
b. Semua elemen di posisi ke 7 sampai dengan ke 12 (58,100,56,46,41 dan 51) aka lebih besar atau sama dengan nilai 34 yang dipilih.

Dengan demikian, data tersebut akan terpecah menjadi 2 partisi, sebagai berikut :

(11 26 19 8 6) 34 (58,100,56,46,41 51)

Proses ini diulangi kembali untuk masing-masing partisi data, yaitu untuk data (11 26 19 8 6) dan data (58,100,56,46,41 51). Untuk partisi yang pertama, bila nilai Sementara yang diambil adalah data pertama kembali dalam partisi bersangkutan, yaitu 10 dan diatur kembali sedemikian rupa, maka nilai data yang dipilih akan terletak di posisi sebagai berikut:

(6 8) 11 (19 26) 34 (58 100 56 46 41 51)

Untuk mengurutkan masing-masing partisi, maka proses tersebut diulangi kembali dan tiap-tiap partisi dipecah-pecah kembali lebih lanjut. Kurung yang menutupi partisi menunjukkan data yang belum urut dan perlu diurutkan kembali. Sedang data yang tidak berada diantara tanda kurung merupakan data

yang sudah diurut. Iterasi selanjutya sampai didapatkan data yang telah urut semuanya adalah sebagai berikut ini.

6 (8) 11 (19 26) 34 (58 100 56 46 41 51)

6 8 11 19 (26) 34 (58 100 56 46 41 51)

6 8 11 19 26 34 (51 41 56 46) 58 (100)

6 8 11 19 26 34 (51 41 56 46) 58 100

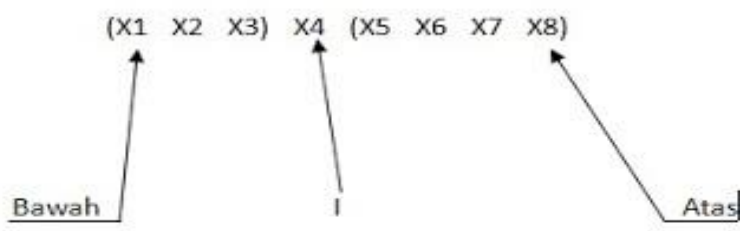
6 8 11 19 26 34 (46 41) 51 (56) 58 100

6 8 11 19 26 34 (46 41) 51 56 58 100

6 8 11 19 26 34 41 (46) 51 56 58 100

6 8 11 19 26 34 41 46 51 56 58 100

Bila diamati lebih lanjut, maka quick sort dapat didefinisikan dengan lebih mudah menggunakan prosedur rekursi. Misalnya untuk partisi sebagai berikut



Maka Proses dari rekursi terlihat dalam pengurutan data dari bawah sampai I-1 dan pengurutan I+1 sampai atas.

Setelah list diacak, memilih poros acak menjadi lebih mudah. Partisi fungsi mengambil item pertama dalam urutan sebagai poros. Partisi dimulai dari kedua ujungnya dan bekerja sampai ke tengah. Intinya setiap kali nilainya semakin besar daripada pivot ditemukan di sisi kiri dan nilai yang lebih kecil dari pivot ditemukan di sisi kanan, kedua nilai tersebut ditukar. Begitu kita mencapai tengah dari keduanya sisi, poros ditukar ke tempatnya. Setelah urutan dipartisi, quicksort

algoritma dipanggil secara rekursif pada dua bagian. Variabel i dan j adalah indeks dari nilai kiri dan kanan, masing-masing, selama proses partisi.

Jika Anda melihat kode partisi, keduanya berkomentar sementara kondisi loop sedang mungkin lebih mudah dipahami daripada kode yang tidak diberi komentar. Namun, yang tidak diberi komentar kode hanya menggunakan operator less than. Quicksort adalah algoritma pengurutan yang digunakan dengan metode sortir pada list. Ini hanya membutuhkan operator kurang dari yang didefinisikan antar item dalam urutan. Dengan menulis dua while loop seperti yang kita miliki, satu-satunya pemesanan yang diperlukan didefinisikan oleh operator less than seperti yang disyaratkan Python. Potret pada Gambar 4.7 menunjukkan efek partisi pada suatu urutan. Di dalam gambar, urutan telah dipartisi dua kali. Partisi pertama dipilih poros yang hampir mati. Namun, partisi kedua memilih poros itu tidak terlalu bagus. Garis merah menunjukkan bagian dari urutan yang sekarang sedang dipartisi. Lihat bagaimana nilai paling kiri di sub-urutan itu menjadi nilai pivot. Dua titik hijau adalah nilai pivot yang sudah berada di lokasi yang benar. Semua nilai di atas poros akan berakhir di partisi di sebelah kanan poros dan semuanya nilai di sebelah kiri pivot lebih kecil dari pivot. Inilah sifat quicksort. Sekali lagi, dengan kompleksitas yang diamortisasi kita dapat menemukan bahwa algoritma quicksort berjalan O (n log n) waktu. Pertimbangkan untuk menyortir list [5 8 2 6 9 1 0 7] menggunakan quicksort. Gambar 4.8 menggambarkan list setelah setiap panggilan ke fungsi partisi. Pivot di setiap panggilan diidentifikasi oleh item berwarna oranye. Fungsi partisi mempartisi list yang meluas ke kanan dari porosnya. Setelah dipartisi, pivot dipindahkan ke lokasi akhirnya dengan menukar itu dengan item terakhir yang kurang dari poros. Kemudian partisi dilakukan pada file menghasilkan dua sublist.

Pengacakan yang dilakukan pada langkah pertama quicksort membantu memilih yang lebih acak nilai poros. Ini memiliki konsekuensi nyata dalam algoritme quicksort, terutama saat urutan yang diteruskan ke quicksort memiliki peluang untuk diurutkan. Jika berurutan diberikan ke quicksort diurutkan, atau hampir diurutkan, baik dalam ascending atau descending order, maka quicksort tidak akan mencapai kompleksitas O (n log n). Faktanya, kasus terburuk kompleksitas algoritma adalah O (n2). Jika poros yang dipilih adalah yang terkecil atau terbesar berikutnya nilai, maka partisi tidak akan membagi masalah menjadi sublist yang lebih kecil sebagai terjadi ketika 9 dipilih sebagai poros untuk sublist pada Gambar 4.8. Algoritme akan cukup letakkan satu nilai pada tempatnya dan akhiri dengan satu partisi besar dari semua yang lainnya

nilai. Jika ini terjadi setiap kali pivot dipilih, itu akan mengarah ke O (n2) kompleksitas. Mengacak daftar sebelum quicksorting akan membantu untuk memastikan bahwa ini tidak terjadi.

Urutan penggabungan tidak dipengaruhi oleh pilihan pivot, karena tidak ada pilihan yang diperlukan.Oleh karena itu, jenis gabungan tidak memiliki kasus pertama atau kasus terbaik untuk dipertimbangkan. Itu akan selalu mencapai kompleksitas O (n log n). Meski begitu, quicksort berkinerja lebih baik dalam praktiknya daripada merge sort karena algoritme quicksort tidak perlu menyalin ke daftar baru, lalu kembali lagi. Algoritme quicksort adalah standar de facto untuk algoritme pengurutan.

Algoritma penyortiran quicksort yang mengikuti algoritma divide and conquer. Pertama, kita akan belajar apa itu membagi dan menaklukkan algoritma.

3. Teknik Divide and Conquer

Divide and Conquer adalah paradigma algoritmik yang sama dengan Greedy dan Dynamic Programming. Algoritma membagi dan menaklukkan standar mengikuti tiga langkah untuk memecahkan masalah.

a. Bagi: Pecahkan masalah yang diberikan menjadi subproblems yang termasuk dalam jenis yang sama.
b. Conquer: Selesaikan subproblems secara rekursif.
c. Gabungkan: Gabungkan semua subproblems di akhir untuk mendapatkan jawabannya.

Algoritma Quicksort memilih elemen sebagai pivot dan partisi array yang diberikan di sekitar elemen pivot yang dipilih. Ada banyak cara elemen pivot dapat dipilih. Selalu pilih elemen pertama sebagai pivot.

a. Selalu pilih elemen terakhir sebagai pivot.
b. Pilih elemen acak sebagai pivot.
c. Pilih elemen tengah atau median sebagai pivot.

Setelah memilih elemen pivot, algoritma mengatur ulang sedemikian rupa sehingga semua elemen yang lebih kecil dari elemen pivot yang dipilih bergeser

ke sisi kiri elemen pivot dan semua elemen yang lebih besar dari elemen pivot yang dipilih bergeser ke sisi kanan elemen pivot. Akhirnya, algoritma secara rekursif mengurutkan subarrays di sisi kiri dan kanan elemen pivot.

4. Quicksort dengan Python

Dalam kehidupan nyata, kita harus selalu menggunakan jenis builtin yang disediakan oleh Python. Namun, memahami algoritma quicksort adalah instruktif.

Tujuan di sini adalah untuk memecah subjek sehingga mudah dipahami dan ditiru oleh pembaca tanpa harus kembali ke bahan referensi. Algoritma quicksort pada dasarnya adalah sebagai berikut:

a. Pilih titik data pivot.

b. Pindahkan semua titik data kurang dari (di bawah) pivot ke posisi di bawah pivot - pindahkan yang lebih besar dari atau sama dengan (di atas) pivot ke posisi di atasnya.

c. Terapkan algoritma ke area di atas dan di bawah pivot

d. Jika data didistribusikan secara acak, pilih titik data pertama sebagai pivot setara dengan pilihan acak.

Implementasi sebelumnya menciptakan banyak daftar tambahan yang tidak perlu. Jika kita bisa melakukan ini di tempat, kita akan menghindari membuang-buang ruang.

Algoritma ini sering diajarkan dalam kursus ilmu komputer dan diminta untuk wawancara kerja. Ini membantu kita berpikir tentang rekursi dan membagi-dan-menaklukkan. Quicksort tidak terlalu praktis dalam Python karena algoritma timsort builtin cukup efisien, dan kami memiliki batas rekursi. Kami berharap untuk mengurutkan daftar di tempat dengan list.sort atau membuat daftar yang diurutkan baru dengan diurutkan yang keduanya mengambil dan argumen

Dalam python, quick sort didefinisikan sebagai algoritma pemilahan yang mengikuti metode divide dan conquers untuk memilih elemen pivot berdasarkan mana array yang tersisa akan dibagi menjadi dua elemen sub-array yang kurang dari pivot akan berada di sub-array kiri dan elemen yang lebih besar dari

pivot akan berada di sub-array kanan dan proses akan berulang secara rekursif sampai semua sub-array diurutkan tanpa menggunakan array tambahan atau ekstra. Ruang disebut quick sort. Quick sort adalah algoritma penyortiran yang efisien dan paling banyak digunakan yang lebih baik daripada algoritma serupa jika diimplementasikan dengan baik. Ini memiliki kompleksitas waktu kasus rata-rata O (NlogN), dan kompleksitas waktu terburuk adalah O (n ^ 2).

```
4 def quickSort(alist):
5    quickSortHelper(alist,0,len(alist)-1)
6
7 def quickSortHelper(alist,first,last):
8    if first<last:
9
10       splitpoint = partition(alist,first,last)
11
12       quickSortHelper(alist,first,splitpoint-1)
13       quickSortHelper(alist,splitpoint+1,last)
```

```
16 def partition(alist,first,last):
17     pivotvalue = alist[first]
18 
19     leftmark = first+1
20     rightmark = last
21 
22     done = False
23     while not done:
24 
25         while leftmark <= rightmark and \
26                 alist[leftmark] <= pivotvalue:
27             leftmark = leftmark + 1
28 
29         while alist[rightmark] >= pivotvalue and \
30                 rightmark >= leftmark:
31             rightmark = rightmark -1
32 
33         if rightmark < leftmark:
34             done = True
35         else:
36             temp = alist[leftmark]
37             alist[leftmark] = alist[rightmark]
38             alist[rightmark] = temp
39 
40     temp = alist[first]
41     alist[first] = alist[rightmark]
42     alist[rightmark] = temp
43 
44 
45     return rightmark
46 
47 alist = [54,26,93,17,77,31,44,55,20]
48 quickSort(alist)
49 print(alist)
```

Hasilnya jika di run,

[17, 20, 26, 31, 44, 54, 55, 77, 93]

Contoh 2 implementasi Python dengan teknik quick sort

```
def quickSort(alist):
  quickSortHelper(alist,0,len(alist)-1)

def quickSortHelper(alist,first,last):
  if first<last:
    splitpoint = partition(alist,first,last)
    quickSortHelper(alist,first,splitpoint-1)
    quickSortHelper(alist,splitpoint+1,last)
def partition(alist,first,last):
  pivotvalue = alist[first]
  leftmark = first+1
  rightmark = last
  done = False
  while not done:
    while leftmark <= rightmark and alist[leftmark] <= pivotvalue:
      leftmark = leftmark + 1
    while alist[rightmark] >= pivotvalue and rightmark >= leftmark:
      rightmark = rightmark -1
    if rightmark < leftmark:
      done = True
    else:
      temp = alist[leftmark]
      alist[leftmark] = alist[rightmark]
      alist[rightmark] = temp
    temp = alist[first]
    alist[first] = alist[rightmark]
    alist[rightmark] = temp
    return rightmark

alist = [55,20,9,27,37,31,64,15,50]
quickSort(alist)
print(alist)
```

Hasilnya

[9, 20, 15, 27, 31, 37, 50, 64, 55]

Pada Quick Sort di atas adalah pivot yang diambil adalah data paling kiri. Berikut Quick Sort dengan pivot yang digunakan adalah data yang terakhir.contoh 3 implementasi quick sort pada python sebagai berikut.

```
1 def quickSort(alist):
2   quickSortHelper(alist,0,len(alist)-1)
3
4 def quickSortHelper(alist,first,last):
5   if first<last:
6     splitpoint = partition(alist,first,last)
7     quickSortHelper(alist,first,splitpoint-1)
8     quickSortHelper(alist,splitpoint+1,last)
9 def partition(alist,first,last):
10  pivotvalue = alist[last]
11  leftmark = first
12  rightmark = last-1
13  done = False
14  while not done:
15    while leftmark <= rightmark and alist[leftmark] <= pivotvalue:
16      leftmark = leftmark + 1
17    while alist[rightmark] >= pivotvalue and rightmark >= leftmark:
18      rightmark = rightmark -1
19    if rightmark < leftmark:
20      done = True
21    else:
22      temp = alist[leftmark]
23      alist[leftmark] = alist[rightmark]
24      alist[rightmark] = temp
25    temp = alist[last]
26    alist[last] = alist[leftmark]
27    alist[leftmark] = temp
28    return leftmark
29 alist = [55,20,9,27,37,31,64,15,50]
30 quickSort(alist)
31 print(alist)
32
33
```

Hasilnya

[50, 15, 9, 20, 27, 31, 37, 55, 64]

Contoh 4 quick Sort dengan pivot ditengah

```
def quickSort(alist):
  print('Dengan Quick Sort')
  quickSortHelper(alist,0,len(alist)-1)
def quickSortHelper(alist,start,end):
  if start >= end:
    return
  i,j = start, end
  pivot = (start + (end - start)//2)
  while i<=j:
    while(alist[i] < alist[pivot]):
      i+=1
    while(alist[j] > alist[pivot]):
      j-=1
    if(i<=j):
      alist[i],alist[j] = alist[j],alist[i]
      i+=1
      j-=1
  quickSortHelper(alist,start,j)
  quickSortHelper(alist,i,end)
  return alist

alist = [55,20,9,27,37,31,64,15,50]

quickSort(alist)

print(alist)

```

Hasilnya

Dengan Quick Sort

[9, 15, 20, 27, 31, 37, 50, 55, 64]

Akhirnya, ini semua tentang algoritma tipe cepat dalam python. Sejauh ini, kita telah melihat definisi jenis cepat, logika di balik implementasi tipe cepat dengan penjelasan langkah demi langkah, dan seberapa cepat penyortiran dapat diimplementasikan menggunakan berbagai metode python dengan contoh dan output yang sesuai.

C. LATIHAN SOAL

1. Jelaskan yang anda ketahui teknik quick Sort!
2. Buatlah analogi deret bilangan dengan quick sort dalam 6 angka yang akan dilakukan pengurutan!
3. Jelaskan teknik Teknik Divide and Conquer pada quick sort!
4. Jelaskan Bagaimana pengecekan kondisi diawal antrian
5. Buatlah contoh implementasi teknik quick sort dengan bahasa python!

D. REFERENSI

Distance Untuk Deteksi Kemiripan Gambar. *Repository Its*.

Basant Agarwal, B. B. (2018). *Hand-On Data Structures And Algorithms With Python.* London: Packt Publishing.

Emi Sita Eriana, A. Z. (2021). Praktikum Algoritma Dan Pemrograman. Tangerang Selatan: Unpam Press.

Jodi, U. R. (2020). Algoritma Dan Struktur Data.

Mohamad Aslam Katahman, M. F. (2021). Pembangunan Aplikasi Realiti Terimbuh Untuk Pengenalan Struktur Data. Information Technology And Computer Science.

Nasrullah, A. H. (2021). Implementasi Algoritma Decision Tree Untuk Klasifikasi Produk Laris. Jurnal Ilmiah Ilmu Komputer I.

Peng Qi, Y. Z. (2020). Stanza: A Python Natural Language Processing Toolkit For Many Human Languages.

Ranny Meilisa, D. P. (2020). Model Pembelajaran Flipped Classroom Pada Mata Kuliah Algoritma Dan Struktur Data. Jurnal Ilmiah Pendidikan Dan Pembelajaran (Jipp).

Revanza, M. G. (2020). Struktur Data Dan Bahasa Pemrograman.

Risah Subariah, E. S. (T.Thn.). Praktikum Analisis & Perancangan Sistem (Uml).

Sianipar, R. H. (2013). Pemrograman & Struktur Data C: Belajar Dari Contoh Untuk Programmer Pemula Maupun Berpengalaman. Penerbit Informatika, 2013.

Thanaki, J. (2017). Python Natural Language Processing. Mambai.

Zein, A. (2018). Pendeteksian Kantuk Secara Real Time Menggunakan Pustaka Opencv Dan Dlib Python. Sainstech : Jurnal Penelitian Dan Pengkajian Sains Dan Teknologi.

Zein, A. (2019). Pendeteksian Penyakit Malaria Menggunakan Medical Images Analisis Dengan Deep Learning Python. Jurnal Penelitian Dan Pengkajian Sains Dan Teknologi.