PERTEMUAN 7

TEKNIK SEARCH

A. TUJUAN PEMBELAJARAN

Setelah mempelajari Materi ini diharapkan mahasiswa mengerti dan paham teknik search, teknik binary, pencarian binery dan mampu membuat pencarian binary dengan pemrograman python.

B. URAIAN MATERI

1. Konsep Dasar Search

Pencarian atau search, yang telah dibahas di seluruh teks, adalah pilihan yang sangat umum. operasi dan telah dipelajari secara ekstensif. Pencarian linier dari sebuah array atau Python list sangat lambat, tetapi itu dapat ditingkatkan dengan pencarian biner. Bahkan dengan waktu pencarian yang lebih baik, array dan list Python memiliki kelemahan untuk penyisipan dan penghapusan tombol pencarian. Ingat, pencarian biner hanya bisa dilakukan pada urutan yang diurutkan. Saat kunci ditambahkan atau dihapus dari file array atau list Python, urutannya harus dipertahankan. Ini bisa memakan waktu karena kunci harus digeser untuk memberi ruang saat menambahkan kunci baru atau untuk menutup celah saat menghapus kunci yang ada. Penggunaan list tertaut memberikan penyisipan yang lebih cepat dan penghapusan tanpa harus menggeser tombol yang ada. Sayangnya, satu-satunya jenis pencarian yang dapat dilakukan pada list tertaut adalah pencarian linier, meskipun file list diurutkan. Dalam bab ini, kami mengeksplorasi beberapa dari banyak cara dalam struktur pohon dapat digunakan dalam melakukan pencarian yang efisien.

Struktur pohon dapat digunakan mengatur data dinamis secara hierarkis. Pohon memiliki berbagai bentuk dan ukuran tergantung pada aplikasinya dan hubungan antar node. Saat digunakan untuk mencari, setiap node berisi kunci pencarian sebagai bagian dari datanya entri (terkadang disebut payload) dan node diatur berdasarkan hubungan antar kunci. Ada banyak jenis pohon pencarian, beberapa di antaranya hanya variasi yang lain, dan beberapa di

antaranya dapat digunakan untuk mencari data disimpan secara eksternal. Tetapi tujuan utama dari semua pohon pencarian adalah untuk menyediakan efisiensi operasi pencarian untuk dengan cepat menemukan item spesifik yang terdapat di pohon.

Pohon pencarian dapat digunakan untuk mengimplementasikan berbagai jenis wadah, beberapa di antaranya mungkin hanya perlu menyimpan kunci pencarian dalam setiap node pohon. Lebih umum, bagaimanapun, aplikasi mengasosiasikan data atau muatan dengan masing-masing cari dan gunakan struktur dengan cara yang sama seperti ADT peta bekas. Peta ADT pada saat itu kami menerapkannya itu menggunakan struktur list. Latihan dalam beberapa bab menawarkan kesempatan untuk menyediakan implementasi baru menggunakan berbagai struktur data.

Konsep teknik pencarian dikembangkan karena keterbatasan yang diberlakukan oleh teknologi pencarian kata kunci Boolean klasik saat menangani kumpulan teks digital yang besar dan tidak terstruktur. Pencarian kata kunci sering kali memberikan hasil yang menyertakan banyak item yang tidak relevan (positif palsu) atau yang mengeluarkan terlalu banyak item relevan (negatif palsu) karena efek sinonim dan polisemi. Sinonimi berarti bahwa satu dari dua atau lebih kata dalam bahasa yang sama memiliki arti yang sama, dan polisemi berarti bahwa banyak kata individual memiliki lebih dari satu arti. Polisemi merupakan kendala utama bagi semua sistem komputer yang berusaha menangani bahasa manusia. Dalam bahasa Inggris, istilah yang paling sering digunakan memiliki beberapa arti yang sama. Misalnya, kata api dapat berarti: aktivitas pembakaran; untuk menghentikan pekerjaan; untuk meluncurkan, atau untuk menggairahkan (seperti dalam api). Untuk 200 istilah paling polysemous dalam bahasa Inggris, kata kerja tipikal memiliki lebih dari dua belas arti atau pengertian umum. Kata benda khas dari himpunan ini memiliki lebih dari delapan pengertian umum. Untuk 2000 istilah paling polysemous dalam bahasa Inggris, kata kerja tipikal memiliki lebih dari delapan pengertian umum dan kata benda tipikal memiliki lebih dari lima.

Selain masalah polisema dan sinonim, penelusuran kata kunci dapat mengecualikan kata yang salah eja secara tidak sengaja serta variasi pada batang (atau akar) kata (misalnya, serang vs. mencolok). Pencarian kata kunci juga rentan terhadap kesalahan yang disebabkan oleh proses pemindaian

pengenalan karakter optik (OCR), yang dapat menyebabkan kesalahan acak ke dalam teks dokumen (sering disebut sebagai teks berisik) selama proses pemindaian.

Pencarian konsep dapat mengatasi tantangan ini dengan menggunakan disambiguasi arti kata (WSD), dan teknik lainnya, untuk membantunya mendapatkan arti sebenarnya dari kata-kata tersebut, dan konsep dasarnya, bukan hanya dengan mencocokkan string karakter seperti teknologi pencarian kata kunci .

2. Teknik Pencarian

Proses pencarian suatu data tertentu pada sekumpulan data bertipe sama.Dalam proses pemrograman serching/pencarian biasanya digunakan untuk pemprosesan dalam update maupun hapus data sebelum melakukan pemprosesan pencarian data. Kedua yatu cara menyiyisipkan data dalam kumpulan data, apabila data berada maka proses penyisipan tidak diperkenankan. Jika data tidak ada maka proses penyisipan dilakukan untuk tujuan menghindari duplikat data. Tehnik Pencarian Tunggal meliput tehnik Sequential Search / Linier Search dan tehnik Binary Search

a. Pencarian Sekuensial

Teknik dengan menngunakan algoritma searching yang sederhana dimana pencarian secara runtun dengan melakukan pembanding antar elemen satu satu secara urut diawali element pertama sampai ditemukannya data yang diinginkan. Pencarian adalah kebutuhan yang sangat mendasar ketika Anda menyimpan data dalam struktur data yang berbeda. Pendekatan yang paling sederhana adalah untuk pergi di setiap elemen dalam struktur data dan mencocokkannya dengan nilai yang Anda cari. Ini dikenal sebagai pencarian linear. Ini tidak efisien dan jarang digunakan, tetapi membuat program untuk itu memberikan gambaran tentang bagaimana kita dapat menerapkan beberapa algoritma pencarian canggih.

b. Pencarian Linear

Dalam jenis pencarian ini, pencarian berurutan dilakukan di semua item satu per satu. Setiap item diperiksa dan jika kecocokan ditemukan maka item

tertentu dikembalikan, jika tidak, pencarian berlanjut sampai akhir struktur data. Contoh pencarian linier dalam python seperti dibawah ini.

```
def linear_search(values, search_for):
   search_at = 0
   search_res = False
# Match the value with each data element
   while search_at < len(values) and search_res is False:
      if values[search_at] == search_for:
         search_res = True
      else:
         search_at = search_at + 1
   return search_res
l = [64, 34, 25, 12, 22, 11, 90]
print(linear_search(l, 12))
print(linear_search(l, 91))

```

Hasil

Ketika kode di atas dieksekusi, ia menghasilkan hasil berikut :

True

False

c. Pencarian Interpolasi

Algoritma pencarian ini bekerja pada posisi menyelis nilai yang diperlukan. Agar algoritma ini berfungsi dengan baik, pengumpulan data harus dalam bentuk yang diurutkan dan didistribusikan secara merata. Awalnya, posisi probe adalah posisi item paling tengah dari koleksi. Jika kecocokan terjadi, maka indeks item dikembalikan. Jika item tengah lebih besar dari item, maka posisi probe kembali dihitung dalam sub-array di sebelah kanan item tengah. Jika tidak, item dicari di subarray di sebelah kiri item tengah. Proses ini berlanjut pada sub-array juga sampai ukuran subarray berkurang menjadi nol.Pencarian Biner menggunakan Rekursi Menerapkan algoritma pencarian biner menggunakan python seperti yang ditunjukkan di bawah ini. Kami menggunakan daftar item yang dipesan dan merancang fungsi rekursif untuk mengambil dalam daftar bersama dengan indeks awal dan akhir sebagai input. Kemudian, fungsi pencarian biner memanggil dirinya sendiri sampai

menemukan item yang dicari atau menyimpulkan tentang ketidakhadirannya dalam daftar. Contoh sebagai berikut.

```
def bsearch(list, idx0, idxn, val):
   if (idxn < idx0):
      return None
   else:
      midval = idx0 + ((idxn - idx0) // 2)
# Compare the search item with middle most value
   if list[midval] > val:
      return bsearch(list, idx0, midval-1,val)
   elif list[midval] < val:
      return bsearch(list, midval+1, idxn, val)
   else:
      return midval
list = [8,11,24,56,88,131]
print(bsearch(list, 0, 5, 24))
print(bsearch(list, 0, 5, 51))

```

Hasil

Ketika kode di atas dieksekusi, ia menghasilkan hasil berikut :

2

None

d. Pencarian biner

Pencarian Biner adalah algoritma pencarian yang digunakan untuk mencari elemen dari array yang diurutkan. Ini tidak dapat digunakan untuk mencari dari array yang tidakort. Pencarian biner adalah algoritma yang efisien dan lebih baik daripada pencarian linier dalam hal kompleksitas waktu.

Kompleksitas waktu pencarian linier adalah O (n). Sedangkan kompleksitas waktu pencarian biner adalah O (log n). Oleh karena itu, pencarian biner adalah algoritma yang efisien dan lebih cepat mencari tetapi hanya dapat digunakan untuk mencari dari array yang diurutkan.

3. Cara Kerja Binary Search

Ide dasar di balik pencarian biner adalah bahwa alih-alih membandingkan elemen yang diperlukan dengan semua elemen array, kita akan membandingkan elemen yang diperlukan dengan elemen tengah array. Jika ini

ternyata menjadi elemen yang kita cari, kita selesai dengan pencarian dengan sukses. Lain, jika elemen yang kita cari kurang dari elemen tengah, dapat dipastikan bahwa elemen tersebut terletak pada bagian pertama atau kiri dari array, karena array diurutkan. Demikian pula, jika elemen yang kita cari lebih besar dari elemen tengah, dapat dipastikan bahwa elemen tersebut terletak di paruh kedua array.

Dengan demikian, pencarian biner terus mengurangi array menjadi dua. Proses di atas diterapkan secara rekursif pada bagian yang dipilih dari array sampai kita menemukan elemen yang kita cari. Akan mulai mencari dengan indeks kiri 0 dan indeks kanan sama dengan indeks terakhir dari array. Indeks elemen tengah (tengah) dihitung yang merupakan jumlah indeks kiri dan kanan dibagi dengan 2. Jika elemen yang diperlukan kurang dari elemen tengah, maka indeks yang tepat diubah menjadi pertengahan 1, yang berarti kita sekarang akan melihat paruh pertama array saja. Demikian juga, jika elemen yang diperlukan lebih besar dari elemen tengah, maka indeks kiri diubah menjadi pertengahan + 1, yang berarti kita sekarang akan melihat paruh kedua array saja. Kami akan mengulangi proses di atas untuk setengah array yang dipilih.

Bagaimana tahu jika elemen tidak ada dalam array. Kita perlu memiliki beberapa kondisi untuk berhenti mencari lebih lanjut yang akan menunjukkan bahwa elemen tidak ada dalam array. Kami akan secara berulang mencari elemen dalam array selama indeks kiri kurang dari atau sama dengan indeks yang tepat. Setelah kondisi ini berubah menjadi salah dan kami belum menemukan elemennya, ini berarti bahwa elemen tersebut tidak ada dalam array. Contoh nya dibawah ini mari ambil array yang diurutkan berikut dan kita perlu mencari elemen 6.

2	5	6	8	10	11	13	15	16

L=0 H=8 Mid=4

2	5	6	8	10	11	13	15	16

6<10, oleh karena itu mengambil babak pertama.

H=Pertengahan-1

L=0 H=3 Mid=1

2	5	6	8	10	11	13	15	16

6>5, jadi pilih babak kedua.

L=Tengah+1

L=2 H=3 Mid=2

2	5	6	8	10	11	13	15	16

6==6, elemen yang ditemukan

Oleh karena itu elemen 6 ditemukan pada indeks 2.

Pelaksanaan

Dari array yang diurutkan tertentu, cari elemen yang diperlukan dan cetak indeksnya jika elemen hadir dalam array. Jika elemen tidak ada, cetak -1.

Kode untuk implementasi pencarian biner diberikan di bawah ini.

Contoh

```
def binary_search(arr,x):
    l=0
    r=len(arr)-1
    while(l<=r):
        mid=(l+r)//2
        if(arr[mid]==x):
            return mid
        elif(x<arr[mid]):
            r=mid-1
        elif(x>arr[mid]):
            l=mid+1
    return -1
array=[1,2,3,4,5,6,7,8,9,10]
a=7
print(binary_search(array,a))
b=15
print(binary_search(array,b))

```

Hasilnya jika di run.

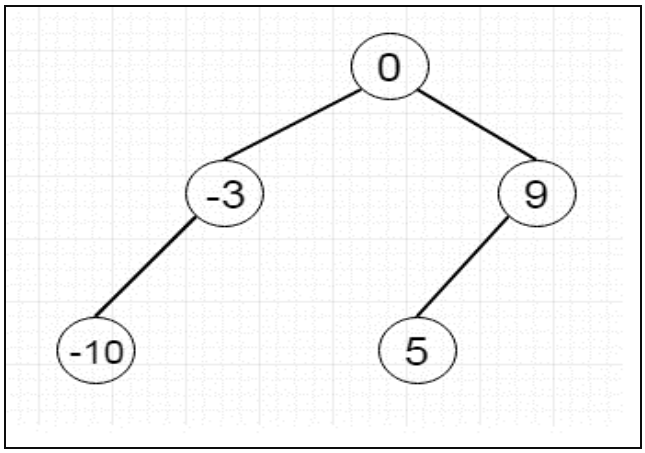
6

-1

Elemen 7 hadir di indeks 6.

Elemen 15 tidak ada dalam array, maka -1 dicetak.

Misalkan kita memiliki satu array A yang diurutkan. Kita harus menghasilkan satu pencarian biner yang seimbang tinggi. Dalam masalah ini, pohon biner yang seimbang tinggi sebenarnya adalah pohon biner di mana kedalaman dua subtrees dari setiap node tidak pernah berbeda lebih dari 1. Misalkan array seperti [-10, -3, 0, 5, 9]. Jadi satu output yang mungkin akan seperti: [0, -3, 9, -10, null, 5]



Untuk mengatasi hal ini, kami akan mengikuti langkah-langkah ini.

Jika A kosong, maka kembalikan Null

menemukan elemen tengah, dan menjadikannya root

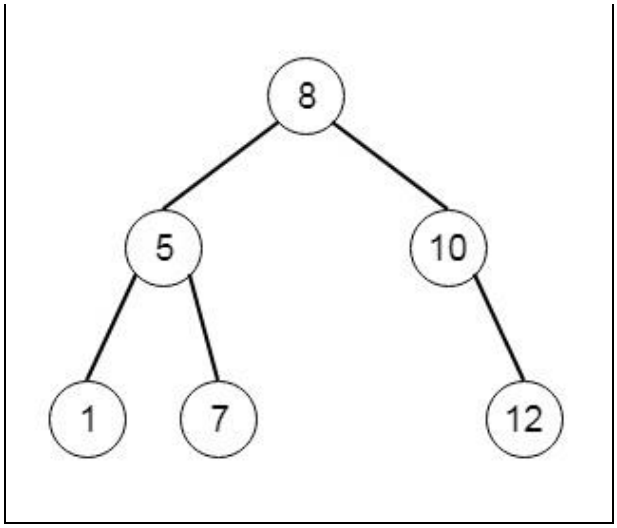
Bagi array menjadi dua sub-array, bagian kiri elemen tengah, dan bagian kanan elemen tengah

rekursif melakukan tugas yang sama untuk subarray kiri dan subarray kanan.

Mari kita lihat implementasi berikut untuk mendapatkan pemahaman yang lebih baik.

Misalkan kita harus membuat pohon pencarian biner yang cocok dengan traversal preorder yang diberikan. Jadi jika traversal pre-order seperti

[8,5,1,7,10,12], maka outputnya akan [8,5,10,1,7,7,12], jadi pohon akan menjadi -



Untuk mengatasi hal ini, ikuti langkah-langkah ini :

akar := 0th node dari daftar traversal preorder

tumpukan := tumpukan, dan dorong akar ke dalam tumpukan

untuk setiap elemen i dari elemen kedua dari daftar preorder

i := node dengan nilai i

jika nilai i < bagian atas elemen atas tumpukan, maka

kiri dari node atas tumpukan := i

masukkan i ke dalam tumpukan

sebaliknya

sementara tumpukan tidak kosong, dan menumpuk nilai elemen atas < nilai i

terakhir := bagian atas tumpukan

elemen pop dari stack

kanan node terakhir := i

masukkan i ke dalam tumpukan

akar pengembalian

4. Pencarian Binery di Python

Penerapkan algoritma pencarian binery menggunakan Python untuk menemukan posisi indeks elemen dalam daftar yang diberikan. Pencarian biner untuk menemukan elemen tertentu dalam daftar. Misalkan kita memiliki daftar seribu elemen, dan kita perlu mendapatkan posisi indeks dari elemen tertentu. Kita dapat menemukan posisi indeks elemen dengan sangat cepat menggunakan algoritma pencarian biner. Ada banyak algoritma pencarian tetapi pencarian biner paling populer di antara mereka. Elemen dalam daftar harus diurutkan untuk menerapkan algoritma pencarian biner. Jika elemen tidak diurutkan maka urutkan terlebih dahulu.

Konsep Pencarian Biner Dalam algoritma pencarian biner, kita dapat menemukan posisi elemen menggunakan metode berikut.

a. Metode Rekursif

b. Metode Iteratif

Teknik pendekatan divide and conquer diikuti dengan metode rekursif. Dalam metode ini, fungsi disebut sendiri lagi dan lagi sampai menemukan elemen dalam daftar. Satu set pernyataan diulang beberapa kali untuk menemukan posisi indeks elemen dalam metode berulang. Sementara loop digunakan untuk menyelesaikan tugas ini. Pencarian biner lebih efektif daripada pencarian linier karena kita tidak perlu mencari setiap indeks daftar. Daftar harus diurutkan untuk mencapai algoritma pencarian biner. Langkah demi langkah implementasi pencarian biner, memiliki daftar elemen yang diurutkan, dan kami mencari posisi indeks 45.

[12, 24, 32, 39, 45, 50, 54]

Jadi penetapan dua petunjuk dalam daftar kami. Satu pointer digunakan untuk menunjukkan nilai yang lebih kecil yang disebut rendah dan pointer kedua digunakan untuk menunjukkan nilai tertinggi yang disebut tinggi. Selanjutnya, menghitung nilai elemen tengah dalam array.

pertengahan = (rendah+tinggi)/2

Di sini, rendah adalah 0 dan tinggi adalah 7.

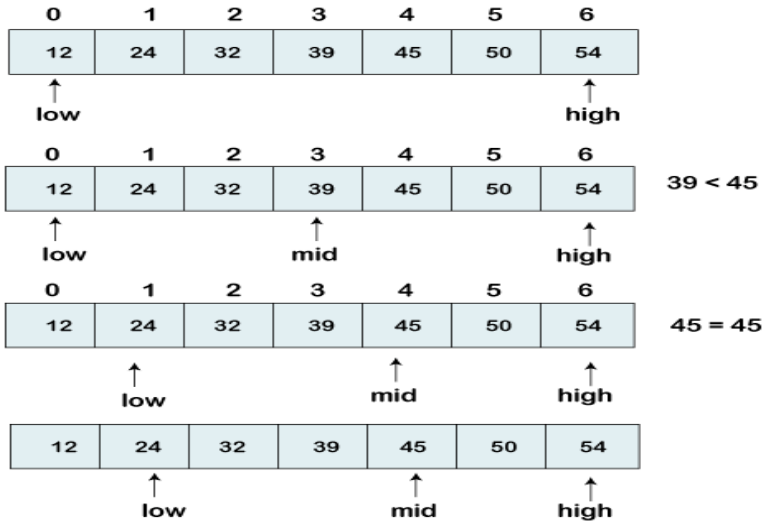
pertengahan = (0+7)/2

pertengahan = 3 (Integer)

Sekarang akan membandingkan elemen yang dicari dengan nilai indeks menengah. Dalam hal ini, 32 tidak sama dengan 45. Jadi perlu melakukan perbandingan lebih lanjut untuk menemukan elemen. Jika angka yang dicari sama dengan pertengahan. Kemudian kembali pertengahan sebaliknya pindah ke perbandingan lebih lanjut. Angka yang akan dicari lebih besar dari angka tengah, membandingkan n dengan elemen tengah elemen di sisi kanan pertengahan dan diatur rendah ke rendah

= pertengahan + 1.

Jika tidak, bandingkan n dengan elemen tengah elemen di sisi kiri tengah dan atur tinggi ke tinggi = pertengahan - 1.



Ulangi sampai nomor yang di cari ditemukan.

Penerapan pencarian biner dengan metode berulang kemudian mengulangi serangkaian pernyataan dan mengulangi setiap item dari daftar, menemukan nilai tengah sampai pencarian selesai. Menciptakan fungsi yang disebut fungsi binary_search() yang membutuhkan dua argumen - daftar untuk diurutkan dan nomor yang akan dicari. Dua variabel untuk menyimpan nilai terendah dan tertinggi dalam daftar. Yang rendah diberi nilai awal ke 0, tinggi ke len (list1) - 1 dan pertengahan sebagai 0.

Selanjutnya, pernyatakan loop sementara dengan kondisi bahwa yang terendah sama dan lebih kecil dari yang tertinggi Sementara loop akan berulang jika jumlahnya belum ditemukan. Sementara loop, menemukan nilai tengah dan membandingkan nilai indeks dengan jumlah yang cari. Jika nilai

indeks menengah lebih kecil dari n,kita meningkatkan nilai tengah sebesar 1 dan menetapkannya ke Pencarian bergerak ke sisi kiri. Jika tidak, kurangi nilai tengah dan tetapkan ke yang tinggi. Pencarian bergerak ke sisi kanan. Jika n sama dengan nilai tengah maka kembali pertengahan. Ini akan terjadi sampai rendah sama dan lebih kecil dari tinggi.

Jika mencapai di akhir fungsi, maka elemen tidak hadir dalam daftar. Kembali 1 ke fungsi panggilan.

Kompleksitas algoritma pencarian biner adalah O (1) untuk kasus terbaik. Ini terjadi jika elemen elemen yang dicari ditemukan dalam perbandingan pertama. O (logn) adalah yang terburuk dan kompleksitas kasus rata-rata pencarian biner. Hal ini tergantung pada jumlah pencarian yang dilakukan untuk menemukan elemen yang cari. Algoritma pencarian biner adalah cara yang paling efisien dan cepat untuk mencari elemen dalam daftar. Ini melewatkan perbandingan yang tidak perlu. Seperti namanya, pencarian dibagi menjadi dua bagian. Ini berfokus pada sisi daftar, yang dekat dengan jumlah yang kami cari.

C. LATIHAN SOAL

1. Jelaskan yang anda ketahui teknik search
2. sebutkan 2 dan jelaskan metode dalam teknik search!
3. Jelaskan kelebihan dan kekurangan linier search!
4. Jelaskan bagaimana cara kerja dalam pencarian binary!
5. Buatlah contoh implementasi teknik search dengan bahasa python!

D. REFERENSI

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

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.

Pradana Setialana, T. B. (2017). Pencarian Hubungan Kekerabatan Pada Struktur Data Genealogy Dalam Graph Databas.

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

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.