**Sorting dan Searching**

Dalam kehidupan kita, secara tidak sadar maupun saat kita sadar, kita menggunakan searching algorithm. Misalnya, setiap pencarian yang kita masukkan ke dalam mesin pencari diperlakukan sebagai substring dan mesin pencari menggunakan searching algorithm untuk memberi Anda hasil. Saat Anda memasukkan string, mesin pencari mencari sejumlah besar data di internet dan memberi Anda hasil pencarian.

Dalam ilmu komputer, algoritma pencarian string, kadang-kadang disebut algoritma pencocokan string, adalah kelas penting dari algoritma string yang mencoba menemukan tempat di mana satu atau beberapa string (juga disebut pola) ditemukan dalam string atau teks yang lebih besar.

Contoh dasar pencarian string adalah ketika pola dan teks yang dicari adalah array dari elemen alfabet (kumpulan hingga) . mungkin alfabet bahasa manusia, misalnya, huruf A sampai Z dan aplikasi lain dapat menggunakan alfabet biner (Σ = {0,1}) atau alfabet DNA (Σ = {A,C,G,T}) dalam bioinformatika.

Dalam prakteknya, metode algoritma pencarian string yang layak dapat dipengaruhi oleh pengkodean string. Secara khusus, jika pengkodean lebar variabel sedang digunakan, maka mungkin lebih lambat untuk menemukan karakter ke-N, mungkin memerlukan waktu yang sebanding dengan N. Hal ini dapat secara signifikan memperlambat beberapa algoritme pencarian. Salah satu dari banyak solusi yang mungkin adalah mencari urutan unit kode sebagai gantinya, tetapi melakukan hal itu dapat menghasilkan kecocokan yang salah kecuali pengkodean dirancang khusus untuk menghindarinya. Algoritma pencarian lebih cepat didasarkan pada pra-pemrosesan teks. Setelah membangun indeks substring, misalnya pohon sufiks atau larik sufiks, kemunculan suatu pola dapat ditemukan dengan cepat. Sebagai contoh, pohon sufiks dapat dibangun dalam waktu \Theta(n), dan semua z kemunculan suatu pola dapat ditemukan dalam waktu O(m+z) (jika ukuran alfabet dipandang sebagai konstanta). Algoritma Pencarian dirancang untuk memeriksa elemen atau mengambil elemen dari struktur data mana pun yang menyimpannya.

Berdasarkan jenis operasi pencarian, algoritma ini umumnya diklasifikasikan menjadi dua kategori: Pencarian Berurutan: Dalam hal ini, daftar atau larik dilalui secara berurutan dan setiap elemen diperiksa. Misalnya: Pencarian Linier. Pencarian Interval: Algoritma ini dirancang khusus untuk pencarian dalam struktur data yang diurutkan. Jenis algoritma pencarian ini jauh lebih efisien daripada Pencarian Linear karena mereka berulang kali menargetkan pusat struktur pencarian dan membagi ruang pencarian menjadi dua. Misalnya: Pencarian Biner. Algoritma pencarian dapat diklasifikasikan berdasarkan mekanisme pencariannya menjadi 3 jenis algoritma: linier, biner, dan hashing. Algoritme pencarian linier memeriksa setiap catatan untuk yang terkait dengan kunci target secara linier. Pencarian biner, atau setengah interval, berulang kali menargetkan pusat struktur pencarian dan membagi ruang pencarian menjadi dua. Algoritme pencarian perbandingan meningkatkan pencarian linier dengan menghilangkan catatan secara berurutan berdasarkan perbandingan kunci sampai catatan target ditemukan, dan dapat diterapkan pada struktur data dengan urutan yang ditentukan.

Algoritma pencarian digital bekerja berdasarkan sifat-sifat digit dalam struktur data yang menggunakan kunci numerik. Terakhir, hashing langsung memetakan kunci ke record berdasarkan fungsi hash. Algoritma sering dievaluasi berdasarkan kompleksitas komputasinya, atau waktu berjalan teoritis maksimum. Fungsi pencarian biner, misalnya, memiliki kompleksitas maksimum O(log n), atau waktu logaritmik. Ini berarti bahwa jumlah operasi maksimum yang diperlukan untuk menemukan target pencarian adalah fungsi logaritmik dari ukuran ruang pencarian.

Quick sort adalah algoritma internal yang didasarkan pada divide dan conquer Array elemen dibagi menjadi beberapa bagian berulang kali sampai tidak mungkin untuk membaginya lebih lanjut. Ini juga dikenal sebagai &quot;jenis pertukaran partisi&quot;. Ini menggunakan elemen kunci (pivot) untuk mempartisi elemen. Satu partisi kiri berisi semua elemen yang lebih kecil dari pivot dan satu partisi kanan berisi semua elemen yang lebih besar dari elemen kunci.

Merge sort adalah algoritma eksternal dan berdasarkan strategi bagi dan taklukkan. Di dalam: Elemen-elemen dipecah menjadi dua sub-array (n/2) berulang-ulang hingga hanya tersisa satu elemen. Merge sort menggunakan penyimpanan tambahan untuk menyortir array tambahan. Merge sort menggunakan tiga array di mana dua digunakan untuk menyimpan setiap setengahnya, dan yang ketiga eksternal digunakan untuk menyimpan daftar terurut terakhir dengan menggabungkan dua lainnya dan setiap array kemudian diurutkan secara rekursif. Akhirnya, semua sub array digabungkan untuk menjadikannya ukuran elemen &#39;n&#39; dari array.

Perbandingan Quick sort &amp; Merge sort Partisi elemen dalam array: Dalam pengurutan gabungan, array dibagi menjadi hanya 2 bagian (yaitu n/2). Sedangkan Dalam hal pengurutan cepat, array dibagi menjadi rasio apa pun. Tidak ada paksaan untuk membagi array elemen menjadi bagian yang sama dalam pengurutan cepat.

Kompleksitas kasus terburuk: Kompleksitas kasus terburuk dari quick sort adalah O(n2) karena ada kebutuhan banyak perbandingan dalam kondisi terburuk. Sedangkan Dalam pengurutan gabungan, kasus terburuk dan kasus rata-rata memiliki kompleksitas yang sama O(n log n).

Penggunaan dengan kumpulan data: Pengurutan gabungan dapat bekerja dengan baik pada semua jenis kumpulan data terlepas dari ukurannya (baik besar atau kecil). Sedangkan Pengurutan cepat tidak dapat bekerja dengan baik dengan kumpulan data besar. Persyaratan ruang penyimpanan tambahan: Merge sort tidak pada tempatnya karena membutuhkan ruang memori tambahan untuk menyimpan array tambahan. Sedangkan Penyortiran cepat tersedia karena tidak memerlukan penyimpanan tambahan.

Efisiensi: Pengurutan gabungan lebih efisien dan bekerja lebih cepat daripada pengurutan cepat jika ukuran array atau kumpulan data lebih besar. Sedangkan Pengurutan cepat lebih efisien dan bekerja lebih cepat daripada pengurutan gabungan jika ukuran larik atau kumpulan data lebih kecil.

Metode penyortiran: Quick sort adalah metode pengurutan internal dimana data diurutkan dalam memori utama. Sedangkan Merge sort adalah metode pengurutan eksternal dimana data yang akan disortir tidak dapat ditampung dalam memori dan membutuhkan memori tambahan untuk pengurutan.

Stabilitas: Merge sort stabil karena dua elemen dengan nilai yang sama muncul dalam urutan yang sama dalam output yang diurutkan seperti halnya dalam array input yang tidak disortir. Sedangkan Penyortiran cepat tidak stabil dalam skenario ini. Tapi itu bisa dibuat stabil menggunakan beberapa perubahan kode.

Lebih disukai untuk: Sortir cepat lebih disukai untuk array. Sedangkan Pengurutan gabungan lebih disukai untuk daftar tertaut. Lokalitas referensi: Quicksort menunjukkan lokalitas cache yang baik dan ini membuat quicksort lebih cepat daripada merge sort (dalam banyak kasus seperti di lingkungan memori virtual).

Pivot merupakan elemen terbesar atau terkecil pada tiap pemanggilan rekursif, sehingga menjadi suatu bagian yang lebih kecil pivot, pivot dan bagian yang kosong

Running Time untuk Best case : O(nlog2n)

Running Time untuk Worst case : O(n2)

Running Time untuk Average case : O(n log2n)

Kompleksitas waktu dari proses Rekursif.

T(n) adalah running time untuk worst-case untuk mengurutkan n data/bilangan.

Diasumsikan n=2k, untuk integer k.

Bubble Sort

Metode sorting dengan bubble sort adalah membandingkan dua buah nilai yang bersebelahan

contoh:

void Bubble(int \*DataArr, int n)

{

int i, j;

for(i=1; i<n; i++)

for(j=n-1; j>=i; j--)

if(DataArr[j-1] > DataArr[j])

Swap(&DataArr[j-1],&DataArr[j]);

}

Selection Sort

Mengambil Bilangan dan membandingkan nilainya dengan angka yang lain

contoh:

for(i=0; i<N-1; i++){ /\* N=number of data \*/

Set idx\_smallest equal to i

for(j=i+1; j<N; j++){

If array[ j ] < array [ idx\_smallest ] then idx\_smallest = j

}

Swap array[ i ] with array[ idx\_smallest ]

}

Quick Sort

Membagi array menjadi 2 dan kemudian menetukan pivot, biasa nya pivot yang diambil adalah array terakhir dan array pertama. jika diambil array terakhir dan pertama maka pivot akan bergerak ke kiri , dan array pertama akan bergerak ke kanan, kemudian dilakukan pengecekan apakah bilangan yang bergerak kanan lebih besar daripada pivot pertama, dan apakah bilangan ke kiri lebih kecil dibanding pivot terakhir jika kondisi terpenuhi maka akan dilakukan pengecekan berdasarkan kondisi misal lebih keci maka akan di cek apakah bilangan di sebelah kiri tadi lebih kecil dari bilangan di sebelah kanan , jika kondisi terpenuhi maka akan dilakukan swap antar bilangan tersebut.

contoh:

void QuickSort(int left, int right)

{

if(left < right){

//arrange elements R[left],...,R[right] that

//producing new sequence:

R[left],...,R[J-1] < R[J] and R[J+1],...,R[right] > R[J].

QuickSort(left, J-1);

QuickSort(J+1, right);

}

}