**LAPORAN DESAIN ANALISIS ALGORITMA:   
HASH TABLE DAN RED-BLACK TREE**

Dosen Pengampu : Imam Marzuki Shofi



Disusun Oleh:   
Kelompok D

|  |  |  |
| --- | --- | --- |
| Lintas Atlantik Mannawang | : | NIM 11230910000088 |
| M. Daffa Aryanto Gumita | : | NIM 11230910000098 |
| Muhammad Aufar Ar Rosyid | : | NIM 11230910000033 |
| Muhammad Jidan Setyobimo | : | NIM 11230910000085 |
| Pramuditya Zindu Pratama | : | NIM 11230910000121 |

**PRODI TEKNIK INFORMATIKA**

**FAKULTAS SAINS DAN TEKNOLOGI**

**UNIVERSITAS ISLAM NEGERI SYARIF HIDAYATULLAH JAKARTA**

**2025 M / 1446 H**

# DAFTAR ISI

[DAFTAR ISI 2](#_Toc201216270)

[HASH TABLE 3](#_Toc201216271)

[Pendahuluan 3](#_Toc201216272)

[Konsep Dasar: Direct Addressing 3](#_Toc201216273)

[Hash Table dan Keunggulannya 3](#_Toc201216274)

[Penanganan Collision 3](#_Toc201216275)

[Fungsi Hash 4](#_Toc201216276)

[Hashing untuk Input Panjang 4](#_Toc201216277)

[Analisis Kompleksitas 5](#_Toc201216278)

[Penutup 5](#_Toc201216279)

[RED-BLACK TREE 6](#_Toc201216280)

[Pendahuluan 6](#_Toc201216281)

[Properti Red-Black Tree 6](#_Toc201216282)

[Rotasi 6](#_Toc201216283)

[Proses Penyisipan 6](#_Toc201216284)

[Proses Penghapusan 6](#_Toc201216285)

[Efisiensi dan Analisis 7](#_Toc201216286)

[Kesimpulan 7](#_Toc201216287)

# HASH TABLE

## Pendahuluan

Struktur data hash table merupakan solusi efisien untuk menyimpan dan mengakses data secara cepat, khususnya dalam konteks operasi kamus seperti INSERT, SEARCH, dan DELETE. Salah satu contoh penggunaan nyata adalah pada symbol table dalam compiler atau implementasi tipe data dict di Python. Hash table menawarkan kecepatan rata-rata akses yang sangat baik, bahkan mencapai waktu konstan O(1), meskipun dalam kasus terburuk bisa menjadi O(n) apabila terjadi collision yang parah.

## Konsep Dasar: Direct Addressing

Sebelum masuk ke hash table, konsep yang paling sederhana adalah direct-address table, yaitu array yang digunakan untuk menyimpan data berdasarkan nilai kunci secara langsung. Setiap elemen dalam array merepresentasikan satu kemungkinan kunci dari semesta kunci U. Teknik ini sangat cepat dan efisien apabila U relatif kecil dan padat, karena setiap operasi dapat dilakukan dalam O(1). Namun, ketika semesta kunci U sangat besar atau jarang digunakan (sparse), pendekatan ini menjadi tidak efisien dari sisi memori karena akan banyak slot yang tidak terpakai.

## Hash Table dan Keunggulannya

Untuk mengatasi keterbatasan direct addressing, digunakan pendekatan hashing. Alih-alih menggunakan kunci sebagai indeks langsung, hashing menggunakan fungsi hash h(k) untuk memetakan kunci ke indeks array. Ini memungkinkan ukuran array yang digunakan jauh lebih kecil dibanding semesta kunci sebenarnya, sehingga penggunaan memori menjadi efisien.

Namun, karena h(k) mungkin menghasilkan nilai yang sama untuk dua kunci berbeda (collision), maka diperlukan strategi untuk mengatasi masalah tersebut. Penanganan collision menjadi bagian paling penting dari desain hash table.

## Penanganan Collision

Dua pendekatan utama digunakan untuk mengatasi collision: chaining dan open addressing.  
  
Pada metode chaining, setiap slot dalam hash table berisi sebuah linked list yang menampung semua elemen yang memiliki nilai hash sama. Ini memungkinkan beberapa elemen untuk disimpan pada slot yang sama, dan operasi pencarian atau penghapusan dilakukan dengan menyusuri list tersebut. Analisis menunjukkan bahwa dengan asumsi uniform hashing, pencarian baik yang berhasil maupun tidak, memiliki waktu rata-rata O(1 + α), dengan α adalah load factor (rasio antara jumlah elemen dan ukuran tabel).  
Sementara itu, open addressing menyimpan semua elemen langsung di dalam array hash table tanpa struktur data eksternal. Ketika terjadi collision, algoritma akan mencari slot kosong lain menggunakan skema probing tertentu seperti linear probing atau double hashing. Salah satu tantangan dalam open addressing adalah bagaimana menangani penghapusan data tanpa merusak urutan probing.

## Fungsi Hash

Efektivitas hash table sangat ditentukan oleh kualitas fungsi hash yang digunakan. Fungsi hash yang baik harus menyebarkan kunci secara merata ke seluruh slot yang tersedia dan mudah dihitung. Dua pendekatan statis umum yang dibahas adalah division method (menggunakan h(k) = k mod m) dan multiplication method, yang lebih fleksibel terhadap ukuran tabel. Namun, pendekatan ini bisa rawan terhadap distribusi kunci yang buruk jika tidak dipilih dengan hati-hati.

Sebagai solusi yang lebih kuat, digunakan pendekatan random hashing, khususnya universal hashing. Dalam universal hashing, fungsi hash dipilih secara acak dari sebuah keluarga fungsi hash pada saat program dijalankan. Dengan cara ini, probabilitas collision antara dua kunci dapat dijaga tidak melebihi 1/m. Hal ini membuat hash table lebih tahan terhadap input yang dirancang untuk memicu kasus terburuk.

## Hashing untuk Input Panjang

Untuk kunci yang panjang seperti string atau vektor, digunakan pendekatan khusus. Salah satu metode adalah mengubah input menjadi representasi numerik yang dapat diproses menggunakan fungsi hash standar. Alternatif lainnya adalah menggunakan hash function kriptografis seperti SHA-256. Fungsi-fungsi ini dirancang untuk menghasilkan distribusi hash yang sangat baik meskipun pada input yang kompleks dan panjang. Beberapa arsitektur komputer modern bahkan menyediakan instruksi khusus (seperti AES-NI) untuk mempercepat eksekusi fungsi-fungsi tersebut.

## Analisis Kompleksitas

Secara umum, waktu rata-rata pencarian, penyisipan, dan penghapusan dalam hash table adalah O(1) jika distribusi kunci merata dan load factor tidak terlalu tinggi (biasanya disarankan α < 0.7). Pada chaining, performa sangat tergantung pada panjang rata-rata linked list pada setiap slot. Pada open addressing, jika load factor mendekati 1, maka pencarian bisa menjadi lambat karena banyak slot sudah terisi.

Universal hashing membuktikan bahwa jika fungsi hash dipilih secara acak dari keluarga yang universal, maka bahkan dalam skenario adversarial (kunci dipilih secara sengaja untuk memicu collision), waktu rata-rata operasi tetap O(1) selama load factor dikontrol.

## Penutup

Hash table merupakan salah satu struktur data paling efisien dan banyak digunakan dalam ilmu komputer dan pemrograman. Dengan pemilihan fungsi hash yang tepat dan metode penanganan collision yang sesuai, hash table mampu menjalankan berbagai operasi dasar dengan sangat cepat. Konsep seperti chaining, open addressing, dan universal hashing memberikan fleksibilitas dan efisiensi tinggi dalam implementasinya di dunia nyata.

# RED-BLACK TREE

## Pendahuluan

Red-black tree adalah struktur data pohon pencarian biner yang memastikan pohon tetap seimbang secara logaritmik. Pohon ini memungkinkan operasi seperti pencarian, penyisipan, dan penghapusan dilakukan dalam waktu O(log n). Dengan menambahkan satu bit warna (merah atau hitam) pada tiap node dan menerapkan aturan warna tertentu, red-black tree menjaga agar tinggi pohon tetap rendah meskipun banyak operasi dinamis dilakukan.

## Properti Red-Black Tree

Red-black tree adalah pohon pencarian biner yang memenuhi lima aturan utama: (1) setiap node berwarna merah atau hitam, (2) root selalu hitam, (3) semua daun (NIL) adalah hitam, (4) anak dari node merah harus hitam, dan (5) setiap jalur dari node ke daun memiliki jumlah node hitam yang sama. Aturan-aturan ini menjamin keseimbangan pohon secara relatif.

## Rotasi

Untuk menjaga keseimbangan dan memperbaiki pelanggaran aturan setelah penyisipan atau penghapusan, dilakukan rotasi pohon. Ada dua jenis rotasi: rotasi kiri dan rotasi kanan. Keduanya hanya memodifikasi sejumlah kecil pointer dan menjaga sifat-sifat dasar pohon pencarian biner.

## Proses Penyisipan

Node baru selalu disisipkan sebagai node merah. Jika hal ini menyebabkan dua node merah berurutan, maka dilakukan perbaikan melalui prosedur RB-INSERT-FIXUP. Terdapat tiga kasus yang menangani berbagai kombinasi warna antara node, orang tua, dan pamannya, serta memperbaiki pohon melalui rotasi dan recoloring. Operasi ini berjalan dalam O(log n) dan menjamin struktur tetap valid.

## Proses Penghapusan

Penghapusan node dalam red-black tree lebih kompleks karena bisa menyebabkan pelanggaran beberapa properti secara bersamaan. Jika node yang dihapus adalah hitam, maka bisa terjadi ketidakseimbangan black-height. Prosedur RB-DELETE-FIXUP digunakan untuk memperbaiki ini melalui empat kasus rotasi dan recoloring, tergantung pada warna sibling dan anak-anaknya. Operasi ini juga dijamin berjalan dalam O(log n).

## Efisiensi dan Analisis

Karena tinggi red-black tree dibatasi maksimal 2 log(n + 1), maka semua operasi dasar berlangsung dalam O(log n). RB-INSERT dan RB-DELETE membutuhkan paling banyak dua hingga tiga rotasi saja. Red-black tree sangat efisien untuk sistem yang membutuhkan performa stabil dan update data yang sering, seperti database, memory manager, dan lainnya.

## Kesimpulan

Red-black tree dan hash table adalah dua struktur data yang kuat untuk efisiensi pencarian dan pemeliharaan data dinamis. Hash table menawarkan kecepatan rata-rata terbaik untuk lookup, sementara red-black tree lebih konsisten dalam performa dan menjaga data dalam urutan yang bisa dinavigasi. Pemilihan antara keduanya tergantung pada kebutuhan spesifik aplikasi.