BAB XI
HASHING

A. TUJUAN

1) Mahasiswa mampu membuat dan memdeklarasikan struktur algoritma tabel hash
2) Mahasiswa mampu menerapkan dan mengimplementasikan struktur algoritma tabel hash

B. DASAR TEORI

- **Pengertian Hash Tabel**

  Hash Table adalah sebuah struktur data yang terdiri atas sebuah tabel dan fungsi yang bertujuan untuk memetakan nilai kunci yang unik untuk setiap record (baris) menjadi angka (hash) lokasi record tersebut dalam sebuah tabel. Keunggulan dari struktur hash

  table ini adalah waktu aksesnya yang cukup cepat, jika record yang dicari langsung berada pada angka hash lokasi penyimpanannya. Akan tetapi pada kenyataannya sering sekali ditemukan hash table yang record-recordnya mempunyai angka hash yang sama (bertabrakan). Karena pemetaan hash function yang digunakan bukanlah pemetaan satusatu, (antara dua record yang tidak sama dapat dibangkitkan angka hash yang sama) maka dapat terjadi bentrokan (collision) dalam penempatan suatu data record. Untuk mengatasi hal ini, maka perlu diterapkan kebijakan resolusi bentrokan (collision resolution policy) untuk menentukan lokasi record dalam tabel. Umumnya kebijakan resolusi bentrokan adalah dengan mencari lokasi tabel yang masih kosong pada lokasi setelah lokasi yang berbentrokan.

- **Operasi Pada Hash Tabel**
  - ✓ insert: diberikan sebuah *key* dan nilai, insert nilai dalam tabel
  - ✓ find: diberikan sebuah *key*, temukan nilai yang berhubungan dengan *key*
  - ✓ remove: diberikan sebuah *key*,temukan nilai yang berhubungan dengan *key* kemudian hapus nilai tersebut
  - ✓ getIterator: mengambalikan iterator,yang memeriksa nilai satu demi satu
- **Struktur dan Fungsi pada Hash Tabel**

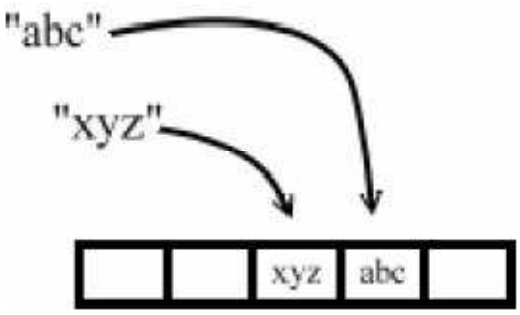
  Hash table menggunakan struktur data array asosiatif yang mengasosiasikan record dengan sebuah field kunci unik berupa bilangan (hash) yang merupakan representasi dari record tersebut. Misalnya, terdapat data berupa string yang hendak disimpan dalam sebuah hash table. String tersebut direpresentasikan dalam sebuah field kunci k. Cara untuk mendapatkan field kunci ini sangatlah beragam, namun hasil akhirnya adalah sebuah bilangan hash yang digunakan untuk menentukan lokasi record.

  Bilangan hash ini dimasukan ke dalam hash function dan menghasilkan indeks lokasi record dalam tabel.

  k(x) = fungsi pembangkit field kunci (1)

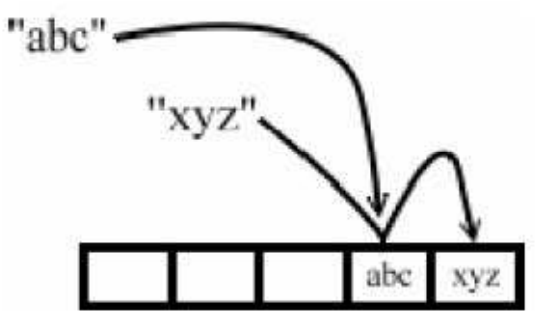
  h(x) = hash function (2)

Contohnya, terdapat data berupa string “abc” dan “xyz” yang hendak disimpan dalam struktur hash table. Lokasi dari record pada tabel dapat dihitung dengan menggunakan h(k(“abc”)) dan h(k(“xyz”)).



Gambar 1. Penempatan record pada hash table

Jika hasil dari hash function menunjuk ke lokasi memori yang sudah terisi oleh sebua record maka dibutuhkan kebijakan resolusi bentrokan. Biasanya masalah ini diselesaikan dengan mencari lokasi record kosong berikutnya secara incremental



Gambar 2. Resolusi bentrokan pada hash table

Deklarasi utama hash tabel adalah dengan struct, yaitu:

```
typedef struct hashtbl {
      hash_size size;
      struct hashnode_s **nodes;
      hash_size (*hashfunc)(const char *);
} HASHTBL;
```

Anggota simpul pada hashtbl mempersiapkan penunjuk kepada unsur pertama yang terhubung dengan daftar. unsur ini direpresentasikan oleh struct hashnode_:

```
struct hashnode_s {
      char *key;
      void *data;
      struct hashnode_s *next;
};
```

- **Inisialisasi**

  Deklarasi untuk inisialisasi hash tabel seperti berikut:

```
HASHTBL *hashtbl_create(hash_size size, hash_size
(*hashfunc)(const char *))
{
  HASHTBL *hashtbl;

  if(!(hashtbl=malloc(sizeof(HASHTBL)))) return NULL;
```

```
  if(!(hashtbl->nodes=calloc(size, sizeof(struct hashnode_s*))))
{
        free(hashtbl);
        return NULL;
  }

  hashtbl->size=size;

  if(hashfunc) hashtbl->hashfunc=hashfunc;
  else hashtbl->hashfunc=def_hashfunc;

  return hashtbl;
}
```

- **Cleanup**

  Hash Tabel dapat menggunakan fungsi linked lists untuk menghapus element atau daftar anggota dari hash tabel .

  Deklarasinya:

```
void hashtbl_destroy(HASHTBL *hashtbl)
{
        hash_size n;
        struct hashnode_s *node, *oldnode;

        for(n=0; n<hashtbl->size; ++n) {
                node=hashtbl->nodes[n];
                while(node) {
                        free(node->key);
                        oldnode=node;
                        node=node->next;
                        free(oldnode);
                }
        }
        free(hashtbl->nodes);
        free(hashtbl);
}
```

- **Menambah Elemen Baru**

  Untuk menambah elemen baru maka harus menentukan ukuran pada hash tabel. Dengan deklarasi sebagai berikut:

```
int hashtbl_insert(HASHTBL *hashtbl, const char *key, void
*data)
{
        struct hashnode_s *node;
        hash_size hash=hashtbl->hashfunc(key)%hashtbl->size;
```

  Penambahan elemen baru dengan teknik pencarian menggunakan linked lists untuk mengetahui ada tidaknya data dengan key yang sama yang sebelumnya sudah dimasukkan, menggunakan deklarasi berikut:

```
node=hashtbl->nodes[hash];
        while(node) {
                if(!strcmp(node->key, key)) {
                        node->data=data;
                        return 0;
```

```
                }
                node=node->next;
        }
```

Jika tidak menemukan key yang sama, maka pemasukan elemen baru pada linked lists dengan deklarasi berikut:

```
if(!(node=malloc(sizeof(struct hashnode_s)))) return -1;
        if(!(node->key=mystrdup(key))) {
                free(node);
                return -1;
        }
        node->data=data;
        node->next=hashtbl->nodes[hash];
        hashtbl->nodes[hash]=node;

        return 0;
}
```

- **Menghapus sebuah elemen**

Untuk menghapus sebuah elemen dari hash tabel yaitu dengan mencari nilai hash menggunakan deklarasi linked list dan menghapusnya jika nilai tersebut ditemukan. Deklarasinya sebagai berikut:

```
int hashtbl_remove(HASHTBL *hashtbl, const char *key)
{
        struct hashnode_s *node, *prevnode=NULL;
        hash_size hash=hashtbl->hashfunc(key)%hashtbl->size;

        node=hashtbl->nodes[hash];
        while(node) {
                if(!strcmp(node->key, key)) {
                        free(node->key);
                        if(prevnode) prevnode->next=node->next;
                        else hashtbl->nodes[hash]=node->next;
                        free(node);
                        return 0;
                }
                prevnode=node;
                node=node->next;
        }

        return -1;
}
```

- **Searching**

Teknik pencarian pada hash tabel yaitu dengan mencari nilai hash yang sesuai menggunakan deklarasi sama seperti pada linked list. Jika data tidak ditemukan maka menggunakan nilai balik NULL. Deklarasinya sebagai berikut:

```
void *hashtbl_get(HASHTBL *hashtbl, const char *key)
{
        struct hashnode_s *node;
        hash_size hash=hashtbl->hashfunc(key)%hashtbl->size;
        node=hashtbl->nodes[hash];
        while(node) {
                if(!strcmp(node->key, key)) return node->data;
                node=node->next;
```

```
        }

        return NULL;
}
```

- **Resizing**

  Jumlah elemen pada hash tabel tidak selalu diketahui ketika terjadi penambahan data. Jika jumlah elemen bertambah begitu besar maka itu akan mengurangi operasi pada hash tabel yang dapat menyebabkan terjadinya kegagalan memory. Fungsi Resizing hash tabel digunakan untuk mencegah terjadinya hal itu.Dekalarsinya sebagai berikut:

```
int hashtbl_resize(HASHTBL *hashtbl, hash_size size)
{
        HASHTBL newtbl;
        hash_size n;
        struct hashnode_s *node,*next;

        newtbl.size=size;
        newtbl.hashfunc=hashtbl->hashfunc;

        if(!(newtbl.nodes=calloc(size, sizeof(struct
hashnode_s*)))) return -1;

        for(n=0; n<hashtbl->size; ++n) {
                for(node=hashtbl->nodes[n]; node; node=next) {
                        next = node->next;
                        hashtbl_insert(&newtbl, node->key,
node->data);
                        hashtbl_remove(hashtbl, node->key);

                }
        }

        free(hashtbl->nodes);
        hashtbl->size=newtbl.size;
        hashtbl->nodes=newtbl.nodes;

        return 0;
}
```

- **Lookup pada Hash table**

  Salah satu keunggulan struktur hash table dibandingkan dengan struktur tabel biasa adalah kecepatannya dalam mencari data. Terminologi lookup mengacu pada proses yang bertujuan untuk mencari sebuah record pada sebuah tabel, dalam hal ini adalah hash table.

  Dengan menggunakan hash function, sebuah lokasi dari record yang dicari bisa diperkirakan. Jika lokasi yang tersebut berisi record yang dicari, maka pencarian berhasil. Inilah kasus terbaik dari pencarian pada hash table. Namun, jika record yang hendak dicari tidak ditemukan di lokasi yang diperkirakan, maka akan dicari lokasi berikutnya sesuai dengan kebijakan resolusi bentrokan. Pencarian akan berhenti jika record ditemukan, pencarian bertemu dengan tabel kosong, atau pencarian telah kembali ke lokasi semula.

- **Collision (Tabrakan)**

  Keterbatasan tabel hash menyebabkan ada dua angka yang jika dimasukkan ke dalam fungsi hash maka menghasilkan nilai yang sama. Hal ini disebut dengan collision.

  contoh :

  Kita ingin memasukkan angka 6 dan 29.

  Hash(6) = 6 % 23 = 6

  Hash(29)= 29 % 23 = 6

  Pertama-tama anggap tabel masih kosong. Pada saat angka 6 masuk akan ditempatkan pada posisi indeks 6, angka kedua 29 seharusnya ditempatkan di indeks 6 juga, namun karena indeks ke-6 sudah ditempati maka 29 tidak bisa ditempatkan di situ, di sinilah terjadi collision. Cara penanganannya bermacam-macam :

  a.Collision Resolution Open Addressing

  1. Linear Probing

  Pada saat terjadi collision, maka akan mencari posisi yang kosong di bawah tempat terjadinya collision, jika masih penuh terus ke bawah, hingga ketemu tempat yang kosong. Jika tidak ada tempat yang kosong berarti HashTable sudah penuh.

  Contoh deklarasi program:

```
struct { ... } node;
node Table[M]; int Free;
/* insert K */
addr = Hash(K);
if (IsEmpty(addr)) Insert(K,addr);
else {
/* see if already stored */
test:
if (Table[addr].key == K) return;
else {
addr = Table[addr].link; goto test;}
/* find free cell */
Free = addr;
do { Free--; if (Free<0) Free=M-1; }
while (!IsEmpty(Free) && Free!=addr)
if (!IsEmpty(Free)) abort;
else {
Insert(K,Free); Table[addr].link = Free;}
}
```

  2. Quadratic Probing

     Penanganannya hampir sama dengan metode linear, hanya lompatannya tidak satu-satu, tetapi quadratic (12, 22, 32, 42, …)

  3. Double Hashing

     Pada saat terjadi collision, terdapat fungsi hash yang kedua untuk menentukan posisinya kembali.

a. Collision Resolution Chaining

- Tambahkan *key* and *entry* di manapun dalam list (lebih mudah dari depan)
- Keunggulan dibandingkan *open addressing*:
  - Proses *insert* dan *remove* lebih sederhana
  - Ukuran Array bukan batasan (tetapi harus tetap meminimalisir *collision*: buat ukuran tabel sesuai dengan jumlah *key* dan *entry* yang diharapkan)
- Kerugian:
  - Overhead pada memory tinggi jika jumlah entry sedikit

Contoh Program:

```
#include <string.h>
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>

/* membuat deklarsi struktur hash tabel dengan linkedlist */
typedef struct _node {
     char *name;
     char *desc;
     struct _node*next;
}node;

#define HASHSIZE 5
static node*hashtab[HASHSIZE];

/* keadaan awal */
void inithashtab() {
     int i;
     for(i=0;i<HASHSIZE;i++)
     hashtab[i];
}

unsigned int hash(char *s){
  unsigned int h=0;
  for(;*s;s++)
    h=*s+h*31;
  return h%HASHSIZE;
```

```
}

node* lookup(char *n){
  unsigned int hi=hash(n);
  node* np=hashtab[hi];
  for(;np!=NULL;np=np->next){
    if(!strcmp(np->name,n))
      return np;
  }

  return NULL;
}

char* m_strdup(char *o){
  int l=strlen(o)+1;
  char *ns=(char*)malloc(l*sizeof(char));
  strcpy(ns,o);
  if(ns==NULL)
    return NULL;
  else
    return ns;
}

char* get(char* name){
  node* n=lookup(name);
  if(n==NULL)
    return NULL;
  else
    return n->desc;
```

```
}

int install(char* name,char* desc){/*menyisipkan data*/
  unsigned int hi;
  node* np;
  if((np=lookup(name))==NULL){
     hi=hash(name);
     np=(node*)malloc(sizeof(node));
     if(np==NULL)
        return 0;
     np->name=m_strdup(name);
     if(np->name==NULL) return 0;
     np->next=hashtab[hi];
     hashtab[hi]=np;
  }
  else
     free(np->desc);
  np->desc=m_strdup(desc);
  if(np->desc==NULL) return 0;

  return 1;
}

void displaytable() {/*menampilkan data*/
 int i;
 node*t;
for(i=0;i<HASHSIZE;i++){
     if(hashtab[i]==NULL)
```

```
        printf("");
    else{
        t=hashtab[i];

        printf("");
        for(;t!=NULL;t=t->next)
      printf("(%s:%s) ",t->name,t->desc);
        printf(" ");
        printf(" ");
    }
  }
}
void cleanup() {/*inisialisasi perintah cleanup*/
 node *np, *t;
 int i;

  for(i=0;i<HASHSIZE;i++){
    if(hashtab[i]!=NULL){
        np=hashtab[i];
        while(np!=NULL){
      t=np->next;
      free(np->name);
      free(np->desc);
      free(np);
      np=t;
        }
    }
  }
}
void data(){
  int i;

  char* names[]={"name  ","address","Phone   ","Cita-cita  ","Sekolah  "};

  char* descs[]={"Nina  ","Lamongan","26300788","Guru","SMA"};


  inithashtab();
  for(i=0;i<5;i++)
    install(names[i],descs[i]);

  printf("\nDone\n");
  printf("\nKerjakan tugas dengan jujur");

  install("phone","9433120451");/*data yang disisipkan*/

  printf("\nhasilnya adalah",get("name"),get("address"),get("phone"),get("Cita-cita"),get("Sekolah"));
  }
  main(){
 int pilihan;
do {
    printf("+_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ +\n");
    printf("|name  ||address ||phone   ||Cita-cita  ||Sekolah  |\n");
    printf("+_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ +\n");
    printf("|Nina  ||Lamongan||26300788||Guru      ||SMA     |\n");
    printf("+_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ +\n");
 cout << "+--------------------+\n";
 cout << "| MENU PILIHAN       |\n";
```

```
cout << "+--------------------+\n";
cout << "| 1. Tampilkan data  |\n";
cout << "| 2. Cleanup         |\n";
cout << "| 3. Keluar          |\n";
cout << "+--------------------+\n";
cout << "| PILIHAN ANDA ? [ ] |\n";
cout << "+--------------------+\n";
cin >> pilihan;
switch (pilihan) {
case 1 :
data();
case 2 :
displaytable();
break;
case 3 :
cleanup();
break;
case 6:
cout << "Trims!!!";
break;
}
getch();
} while (pilihan != 4);
return 0;
}
```

Latihan:

Kembangkan program diatas dengan menginputkan data manual

TUGAS RUMAH

1) Buatlah program Hashing tabel yang mempunyai menu minimal sebagai berikut:
   a) Inputan data
   b) Delete data
   c) Searching data
2) Buatlah sebuah program yang mengalami collision beserta penanganannya