S07 – DS – HashTables

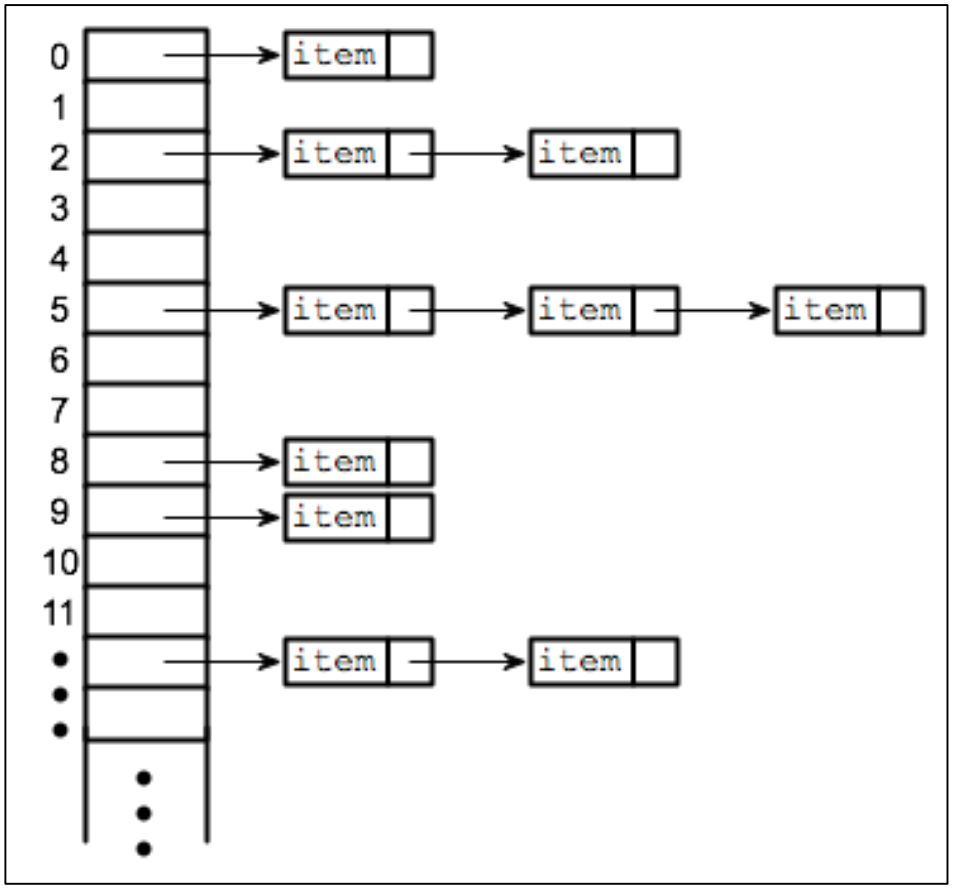
Alin Zamfiroiu
alin.zamfiroiu@csie.ase.ro

Used structures

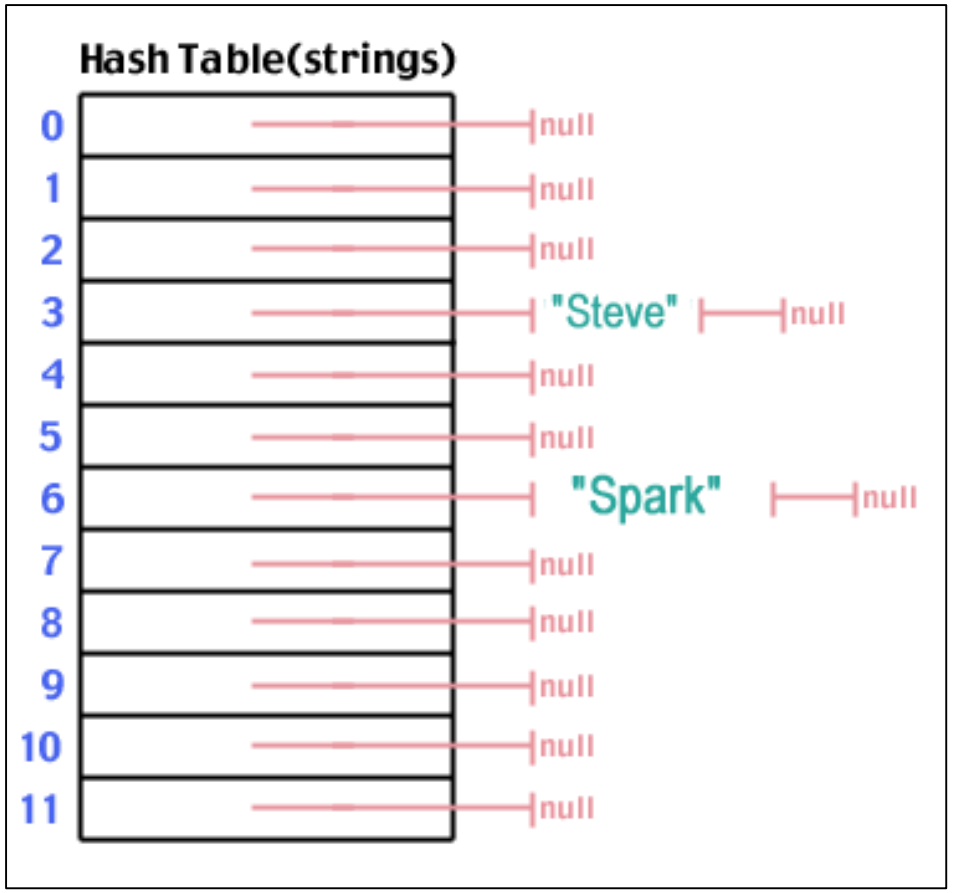
- Vagon will be the useful information.
- A node for a list.
- HashTables are represented by an array of lists.

```
struct vagon{
    int cod;
    char* tip;
};

struct nod{
    vagon info;
    nod* next;
};

struct HashStruct{
    int dimensiune;
    nod** elemente;
};
```

HashTables



http://4spills.blogspot.ro/



http://www.sparknotes.com/

HashTables

- Implement the function that create and return a HashStruct:
- Set its dimension;
- Allocate the space for the array of lists;
- All lists are initialized with NULL;
- Return the new initialized structure.

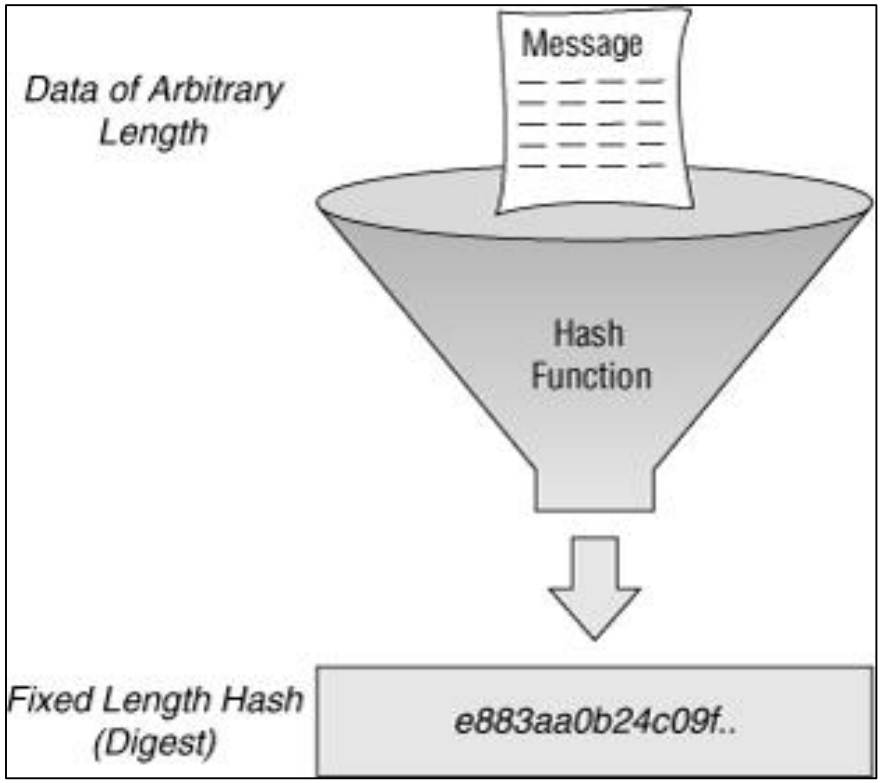
Initialize a HashStruct

```
HashStruct AlocaMemorie()
{
    HashStruct hs;
    hs.dimensiune=50;
    hs.elemente=(nod**)malloc(sizeof(nod*)*hs.dimensiune);
    //hs.elemente=new nod*[hs.dimensiune];
    for(int i=0;i<hs.dimensiune;i++)
    {
        hs.elemente[i]=NULL;
    }
    return hs;
}
```

- Implement the Hash function – that return a hash code for a code and the HashTable dimension.

Hash function

```
int functieHash(int cod, HashStruct hs)
{
    return cod%hs.dimensiune;
}
```



http://ciscodocuments.blogspot.ro/

Insertion

- We use hash function to determine the position in our array.
- If we have a collision, we resolve it by chaining: that means to insert at that position in a list.

Insertion

```
int insereazaVagon(HashStruct hs, vagon v)
{
    int pozitie=-1;

    if(v.cod<0)
    {
        return pozitie;
    }
    if(hs.elemente!=NULL)
    if(hs.elemente!=NULL)
    {
        pozitie=functieHash(v.cod, hs);
        nod* nod_nou=(nod*)malloc(sizeof(nod));
        nod_nou->next=NULL;
        nod_nou->info=v;
        if(hs.elemente[pozitie]==NULL)
            hs.elemente[pozitie]=nod_nou;
        else
        {
            nod* temp=hs.elemente[pozitie];
            while(temp->next)
                temp=temp->next;
            temp->next=nod_nou;
        }
    }
    return pozitie;
}
```

What problem presets this code?

Read from file

- To read from file we use fscanf() function.

- This function get by parameters: the file, and the format of reading.

Read from file

```
vagon citesteVagonDinFisier(FILE*f)
{
    char aux[20];
    vagon v;
    fscanf(f, "%d", &v.cod);
    fscanf(f, "%s", &aux);
    v.tip = (char*)malloc(sizeof(char)*(strlen(aux)+1));
    strcpy(v.tip, aux);
    return v;
}
```

```
FILE*f;
f = fopen("vagoane.txt", "r+");

insereazaVagon(hs, citesteVagonDinFisier(f));
```

Cross the HashTable

- To cross the entire HashTable, we have to cross the array and then the lists of this array.
- The display can be made inline or by calling a function that will display information to the console.
- We can also write in a file by using the fprintf() method.

Cross the HashTable

```
void afiseazaVagon(vagon v)
{
    printf("%d. %s\n",v.cod,v.tip);
}
```

```
void parcurgere(HashStruct hs)
{
    if(hs.elemente)
    {
        for(int i=0;i<hs.dimensiune;i++)
        {
            nod* temp=hs.elemente[i];
            while(temp)
            {
                afiseazaVagon(temp->info);
                temp=temp->next;
            }
        }
    }
}
```

Writing the file

```
void AfisareInFisier(HashStruct hs)
{
    FILE *f=fopen("test.txt","w");

    if(hs.elemente)
    {
        for(int i=0;i<hs.dimensiune;i++)
        {
            nod* temp=hs.elemente[i];
            while(temp)
            {
                //afiseazaVagon(temp->info);
                fprintf(f,"%d. %s.\n",temp->info.cod,temp->info.tip);
                temp=temp->next;
            }
        }
    }
    fclose(f);
}
```

Search by id

- The search function will get the code of the searched object and HashTable, where we have to search.
- We calculate the hash code, and then we will search in the list from that position.

Search by id

```
vagon CautaDupaCod(int cod, HashStruct hs)
{
    vagon v;
    v.cod=-1;
    if(cod<0)
    {
        return v;
    }
    if(hs.elemente!=NULL)
    {
        int pozitie=functieHash(cod, hs);
        if(hs.elemente[pozitie]==NULL)
        {
            return v;
        }
        if(hs.elemente[pozitie]->info.cod==cod)
        {
            return hs.elemente[pozitie]->info;
        }
        else
        {
            nod*temp=hs.elemente[pozitie];
            while(temp&& temp->info.cod!=cod)
            {
                temp=temp->next;
            }
            if(temp!=NULL)
                return temp->info;
            else
            {
                return v;
            }
        }
    }
    return v;
}
```

Delete a HashTable

- To delete the entire structure, we cross the array and delete the lists of this array.
- After we delete all lists, we will delete the array.
- To delete the lists, we implement a recursive function.

Delete a HashTable

```
void stergereHash(HashStruct hs)
{
    if(hs.elemente!=NULL)
    {
        for(int i=0;i<hs.dimensiune;i++)
            hs.elemente[i]=stergeLista(hs.elemente[i]);
        free(hs.elemente);
    }
}
```

```
nod* stergeLista(nod* cap)
{
    if(cap)
    {
        cap->next=stergeLista(cap->next);
        free(cap);
        return NULL;
    }
    else
        return NULL;
}
```

Homework

- Use linear probing to avoid collisions.