16

Aim:- To find a hashed file organization strategy that can be effectively used to store and manage the movie database record.

:- To find a proper hash function

Given:- Movie database record

Imdb ID	Title	Year Released	gross(M)	rating
100	Practical Magic	1998	46	7
101	Dark Knight	2008	532	9
102	Beetle juice	1998	73	7
103	Heart breakers	2001	40	6
104	Shrek	2001	267	8
105	The Simpson movie	2007	183	8
106	The holiday	2006	63	7
107	No time to die	2021	520	8
108	Mr. Queen	2021	340	8
109	Captain Fantastic	2016	280	8
110	Gifted	2008	102	8
111	The family	2013	6	7
112	Aladdin	2019	143	7

Hashed file organization

Hash file organization uses the computation of hash function on some fields of the records. The has function's output determines the location of block where the records are kept.

Extendible hashing method can be used to store and manage the database.

Static hashing is not used as database size in static hashing is fixed and this can cause certain issues. If initial no. of buckets is too small, the performance will degrade due to too much overflow. While on the other hand if very large space is alloted, there will a wastage of memory resources. Thus a dynamic hashing is preferred over it.

Linear hashing, being a dynamic hashing method could have been a probable choice. However compared with Extendible hashing, Linear hashing does not use a bucket directory and when an overflow occurs its not always the overflown bucket that splits. In case, if no. of movies in our DB increases, there are high chances that a lot of overflow buckets get created. Although linear hashing has some advantage in certain areas like queries involving exact match but for our case extendible hashing strategy is also a good choice

Thus, we can effectively manage and store our DB for movie records with extendible hashing strategy.

Extendible hashing uses hash function, directories and buckets to hash data and store the records in a random yet uniform way (provided a good hash function is chosen). For this case, I would be hashing movie title to allot random keys to records.

Hash function

For this DB, polynomial rolling hash function is used
It is defined as :- For a given movie title "s" ⇒

$$hash(s) = (s(0) + s(1) \cdot p + s(2) \cdot p^2 \dots s(n-1) \cdot p^{n-1}) \bmod m$$

where p and m are some chosen positive number.
It is reasonable to make p a prime number roughly equal to no. of input characters (Usually all movie names are smaller than 31 characters (lets assume)

∴ $p = 31$

m should be a large number since probability of two random strings colliding is $\approx \frac{1}{m}$. Therefore,

let $m = 10^9 + 9$

Generating hash values

The hashing could be done manually but since the computation become quite comple, one could write a simple c++ code to generate the required hash values.
The code is as given below :-

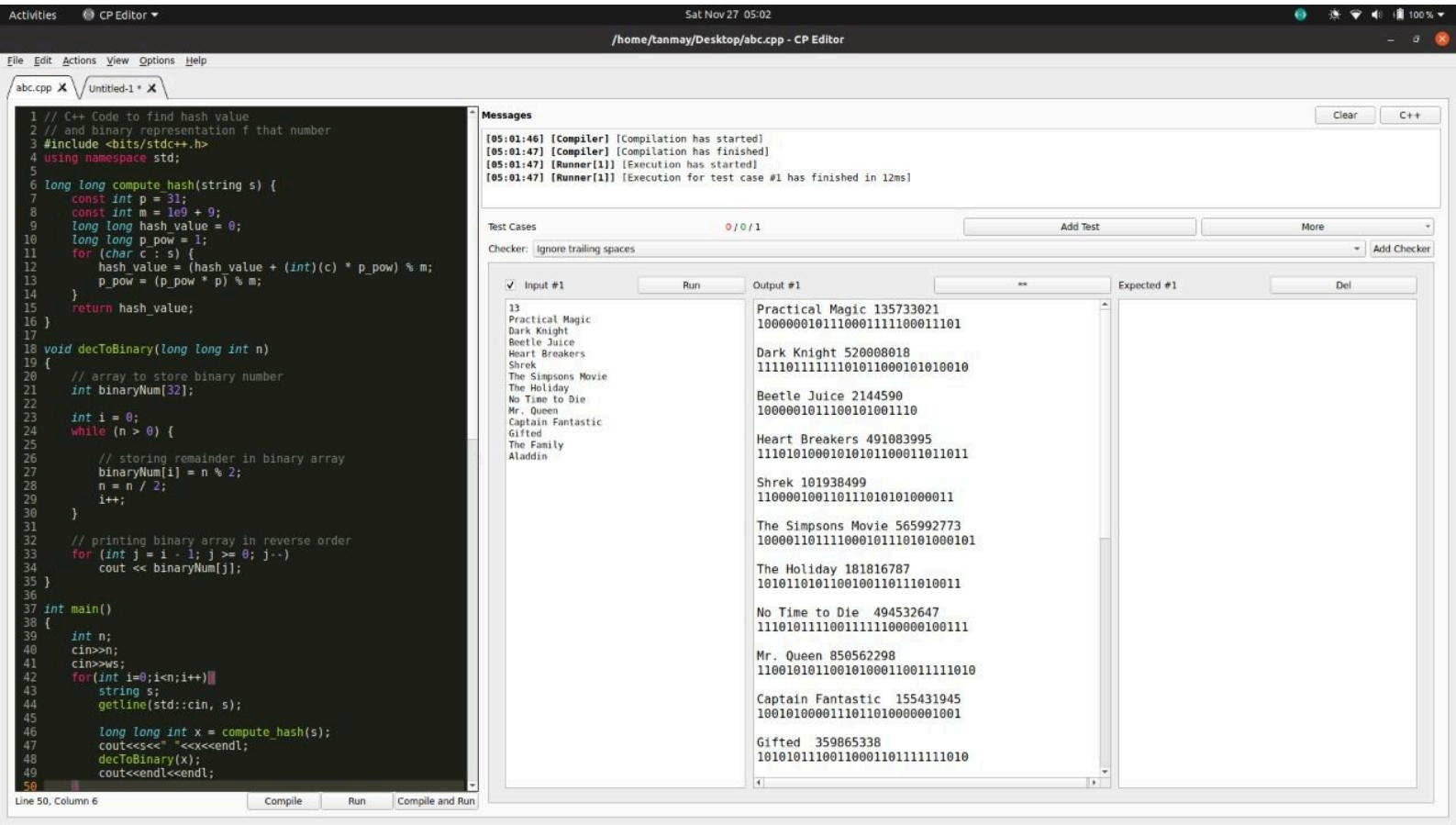
Activities CP Editor

File Edit Actions View Options Help

abc.cpp * Untitled-1 *

```cpp
// C++ Code to find hash value
// and binary representation f that number
#include <bits/stdc++.h>
using namespace std;

long long compute_hash(string s) {
    const int p = 31;
    const int m = 1e9 + 9;
    long long hash_value = 0;
    long long p_pow = 1;
    for (char c : s) {
        hash_value = (hash_value + (int)(c) * p_pow) % m;
        p_pow = (p_pow * p) % m;
    }
    return hash_value;
}

void decToBinary(long long int n)
{
    // array to store binary number
    int binaryNum[32];

    int i = 0;
    while (n > 0) {

        // storing remainder in binary array
        binaryNum[i] = n % 2;
        n = n / 2;
        i++;
    }

    // printing binary array in reverse order
    for (int j = i - 1; j >= 0; j--)
        cout << binaryNum[j];
}

int main()
{
    int n;
    cin>>n;
    cin>>ws;
    for(int i=0;i<n;i++)
        string s;
        getline(std::cin, s);

        long long int x = compute_hash(s);
        cout<<s<<" "<<x<<endl;
        decToBinary(x);
        cout<<endl<<endl;
```

Line 50, Column 6 Compile Run Compile and Run


Activities
CP Editor
Sat Nov 27 05:02
/home/tanmay/Desktop/abc.cpp - CP Editor
File Edit Actions View Options Help
abc.cpp
Untitled-1 *
// C++ Code to find hash value
// and binary representation f that number
#include <bits/stdc++.h>
using namespace std;
long long compute_hash(string s) {
const int p = 31;
const int m = 1e9 + 9;
long long hash_value = 0;
long long p_pow = 1;
for (char c : s) {
hash_value = (hash_value + (int)(c) * p_pow) % m;
p_pow = (p_pow * p) % m;
}
return hash_value;
}
void decToBinary(long long int n)
{
// array to store binary number
int binaryNum[32];
int i = 0;
while (n > 0) {
// storing remainder in binary array
binaryNum[i] = n % 2;
n = n / 2;
i++;
}
// printing binary array in reverse order
for (int j = i - 1; j >= 0; j--)
cout << binaryNum[j];
}
int main()
{
int n;
cin>>n;
cin>>ws;
for(int i=0;i<n;i++)
string s;
getline(std::cin, s);
long long int x = compute_hash(s);
cout<<s<<" "<<x<<endl;
decToBinary(x);
cout<<endl<<endl;
}
Line 50, Column 6
Compile
Run
Compile and Run
Messages
Clear
C++
[05:01:46] [Compiler] [Compilation has started]
[05:01:47] [Compiler] [Compilation has finished]
[05:01:47] [Runner[1]] [Execution has started]
[05:01:47] [Runner[1]] [Execution for test case #1 has finished in 12ms]
Test Cases
0 / 0 / 1
Add Test
More
Checker:
Ignore trailing spaces
Add Checker
Input #1
Run
13
Practical Magic
Dark Knight
Beetle Juice
Heart Breakers
Shrek
The Simpsons Movie
The Holiday
No Time to Die
Mr. Queen
Captain Fantastic
Gifted
The Family
Aladdin
Output #1
**
Practical Magic 135733021
10000001011100011111100011101
Dark Knight 520008018
11110111111110101100010101010010
Beetle Juice 2144590
10000010111001010011110
Heart Breakers 491083995
11101010001010101100011011011
Shrek 101938499
11000010011011101010100011
The Simpsons Movie 565992773
10000110111100010111010100010l
The Holiday 181816787
10101101011001001101110100l1
No Time to Die 494532647
11101011110011111100000100111
Mr. Queen 850562298
11001010110010100011001111l010
Captain Fantastic 155431945
10010100001110110100000010011
Gifted 359865338
10101011100110001101111111010
Expected #1
Del

Note:- $hash(s) = (s(0) + s(1)\cdot p + \ldots + s(n-1)\cdot p^{n-1}) \bmod m$,
here $s(i)$ denotes ascii value of i^{th} character of string
For ex ⇒ ascii value of "1" is 49 and of "a" is 97 etc.

Hash values

Title	hashed no.	*(only last 6 digits)(lsb) Binary
Practical magic	135733021	011101
Dark knight	52000801 8	010010
Beetle juice	2144590	001110
Heart breakers	491083995	011011
Shrek	101938499	000011
The simpson movies	565992773	000101
The holiday	181816787	010011
No time to die	494532647	100111
Mr. Queen	850562298	111010
Captain fantastic	155431945	001001
Gifted	359865338	111010
The family	836279542	110110
Aladdin	726892695	010111

Schematic of file organization

Note:- using $(ID_i)^*$ to represent a full record i.e $\Rightarrow$

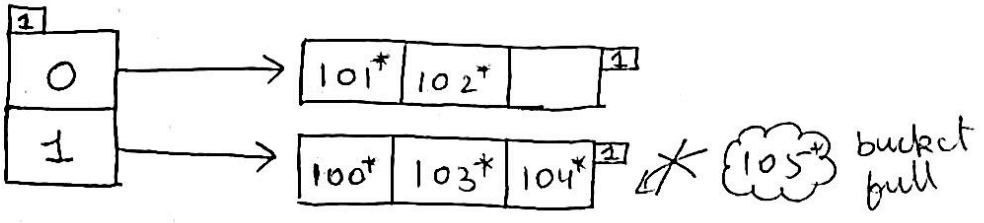
$(ID_i)^* \Rightarrow (ImdbID_i), (Title_i), (year\ released)_i, (gross(M))_i, (rating)_i$

Terms used :-

1. Directories $\Rightarrow$ These containers store pointers to bucket. No. of directories $= 2^{\wedge}$ global depth.
2. Buckets $\Rightarrow$ They store the hashed keys
3. global depth $\Rightarrow$ Number of bits used to categorize the key. global depth = no. of bits in directory.
4. local depth $\Rightarrow$ Same as global depth but it is used for buckets.
5. overflow $\Rightarrow$ It is the situation when the bucket can no longer hold extra records and a new record is pushed in that bucket.

*(Inserting data in the order of ImdbID)

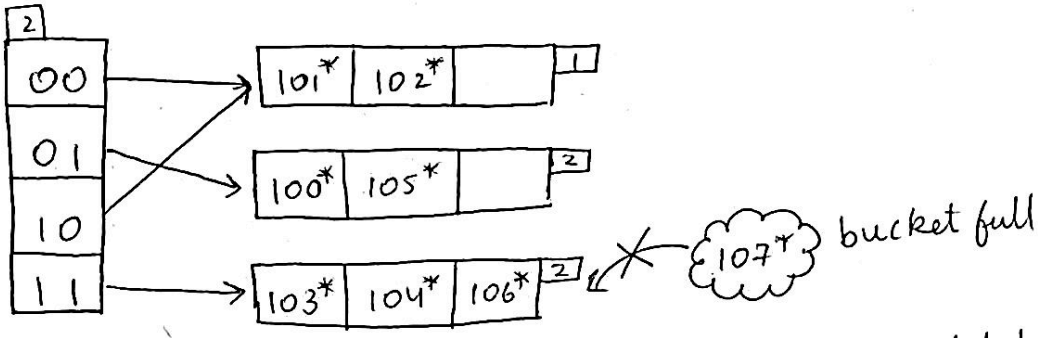
1.) Global depth = 1 *(Note: A bucket can hold upto 3 values)



overflow on inserting 105^*, since bucket is already full.

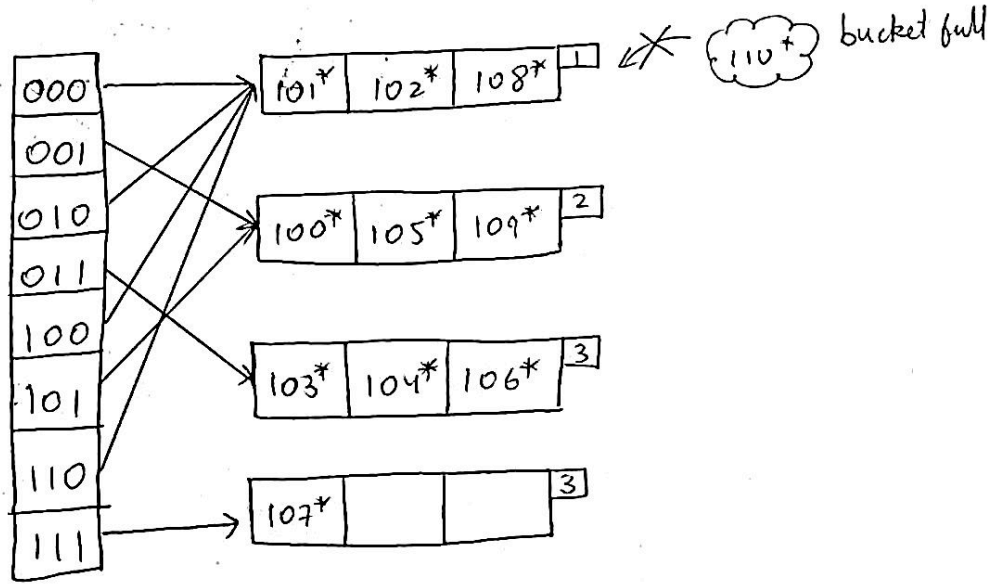
local depth = global depth splitting occurs + Directory expansion.

global depth = 2

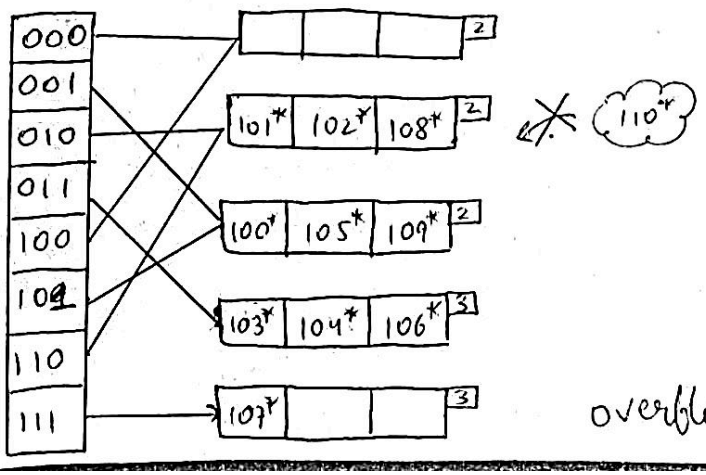


overflow on inserting 107^* => local depth = global depth
directory expansion plus bucket splitting

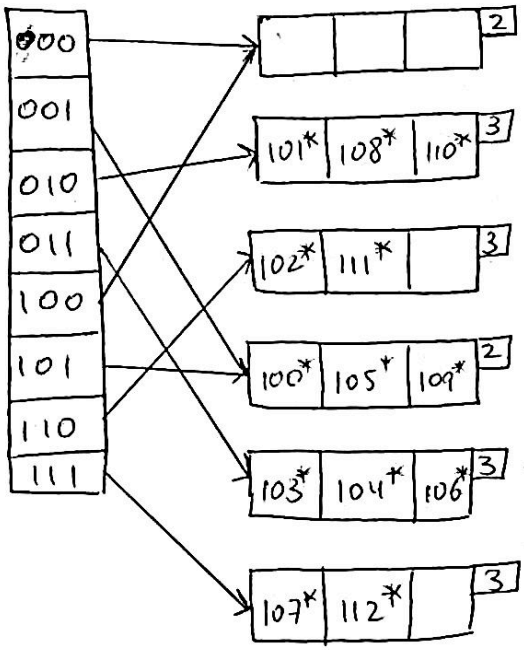
global depth = 3



Overflow on inserting 110^* => local depth < global depth
only bucket splitting



local depth < global depth
overflow again -> bucket split

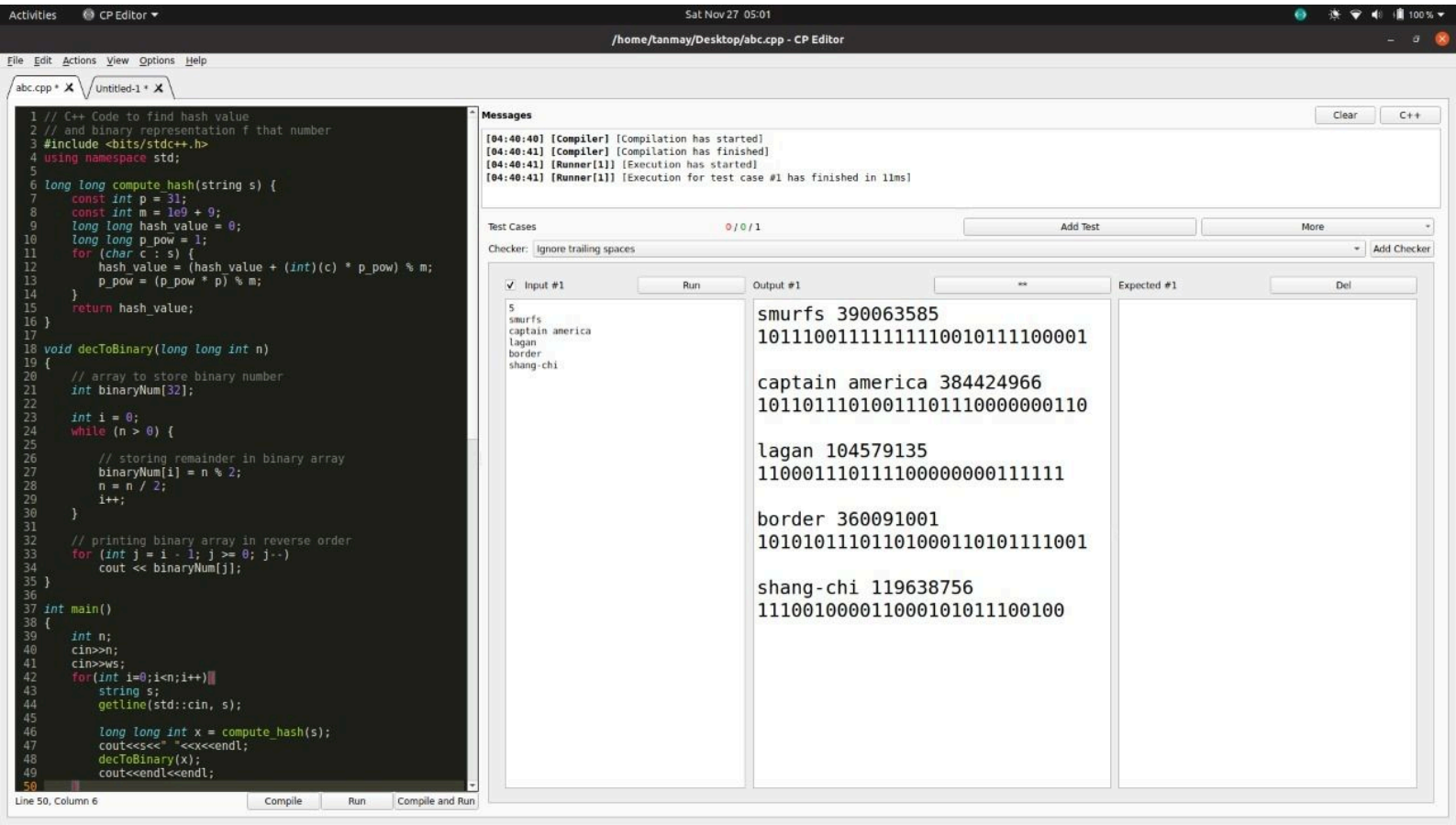


This is the final schematic diagram of file organization for given movie database record.

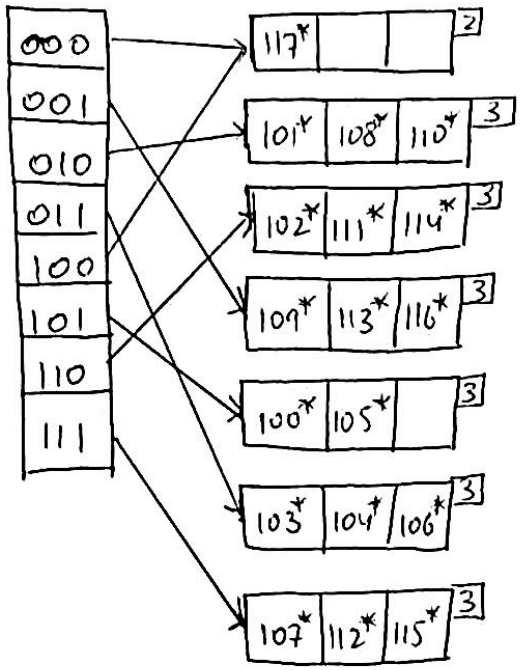
Adding 5 new records

Imdb ID	Titlee	Year Released	gross(M)	rating	hash value	Binary (6 lsb)
113	smurfs	2005	180	7	390063585	100001
114	captain america	2004	450	8	384424966	000110
115	laigan	2001	300	9	104579135	111111
116	border	2002	150	9	360091001	111 001
117	shang-chi	2021	500	9	119638756	100 100

(P.T.O)


Activities
CP Editor
Sat Nov 27 05:01
/home/tanmay/Desktop/abc.cpp - CP Editor
File Edit Actions View Options Help
abc.cpp *
Untitled-1 *
// C++ Code to find hash value
// and binary representation f that number
#include <bits/stdc++.h>
using namespace std;
long long compute_hash(string s) {
const int p = 31;
const int m = 1e9 + 9;
long long hash_value = 0;
long long p_pow = 1;
for (char c : s) {
hash_value = (hash_value + (int)(c) * p_pow) % m;
p_pow = (p_pow * p) % m;
}
return hash_value;
}
void decToBinary(long long int n)
{
// array to store binary number
int binaryNum[32];
int i = 0;
while (n > 0) {
// storing remainder in binary array
binaryNum[i] = n % 2;
n = n / 2;
i++;
}
// printing binary array in reverse order
for (int j = i - 1; j >= 0; j--)
cout << binaryNum[j];
}
int main()
{
int n;
cin>>n;
cin>>ws;
for(int i=0;i<n;i++){
string s;
getline(std::cin, s);
long long int x = compute_hash(s);
cout<<s<<" "<<x<<endl;
decToBinary(x);
cout<<endl<<endl;
}
Line 50, Column 6
Compile
Run
Compile and Run
Messages
Clear
C++
[04:40:40] [Compiler] [Compilation has started]
[04:40:41] [Compiler] [Compilation has finished]
[04:40:41] [Runner[1]] [Execution has started]
[04:40:41] [Runner[1]] [Execution for test case #1 has finished in 11ms]
Test Cases
0 / 0 / 1
Add Test
More
Checker:
Ignore trailing spaces
Add Checker
Input #1
Run
Output #1
Expected #1
Del
5
smurfs
captain america
lagan
border
shang-chi
smurfs 390063585
10111001111111110010111100001
captain america 384424966
10110111010011101110000000110
lagan 104579135
11000111011110000000111111
border 360091001
10101011101101000110101111001
shang-chi 119638756
11100100001100010101110010

global depth = 3



This is the updated schematic diagram after 5 new records are inserted

(Note ⇒ $(\text{Imdb ID})_i^*$ denotes full record containing $(\text{Imdb ID})_i$, $(\text{Title})_i$, $(\text{year released})_i$, $(\text{gross(M)})_i$, $(\text{rating})_i$)

After adding 5 new records the hash function has generated quite random and uniform hash_values

Thus extendible hashing along with our polynomial rolling hash function can be efficiently use to store and manage this movie Database.

— The End —