中山大学计算机学院本科生实验报告

（2021 学年第 1 学期）

课程名称：Data structures and algorithms　　任课教师：张子臻

年级	20 级	专业（方向）	软件工程
学号	20337270	姓名	钟海财
电话	13397996670	Email	2940599563@qq.com
开始日期	2021/10/28	完成日期	2021/11/3

1. 实验题目

1）Query

2）爸爸去哪儿之一

3）爸爸去哪儿之二

2.实验目的

1）Query： 通过实现字符串的哈希函数（使散列值尽可能均匀地分布在存储空间内）并使用恰当的解决冲突的方法，判断 B[]中元素是否存在 A[],若存在，则输出。

2）爸爸去哪儿之一： 按要求（将字母转化成数字（如 a=1，b=2）并求和）实现字符串的哈希函数并使用“线性探测法”解决冲突，将输入的字符串（name）存到哈希表中并输出最终的哈希表。并输出平均查找长度。

3）爸爸去哪儿之二： 按要求（将字母转化成数字（如 a=1，b=2）并求和）实现字符串的哈希函数并使用“平方探测法”解决冲突，将输入的字符串（name）存到哈希表中并输出最终的哈希表。并输出平均查找长度。

3.程序设计

1）Query

设计思路：

设计的字符串的哈希函数要使散列值尽可能均匀地分布在存储空间内，目的是为了减少冲突，加快从哈希表中存数据和查询数据的速度（尤其是查询数据）。所以哈希函数得到散列值的方式不能用字母转化成数字再求和这种方式，当数据量大时，这种方式会使哈希表的前部分几乎充满而后部分得不到有效利用。所以我们可以用类似进制转换的方法，选取一个数（最好为质数）为“进制”，每次将对应的字母转化为数字加上 hash 之后，再将 hash 乘上“进制”，再取模：

```
int myhash(const string s,int n){
    int hash = 0 ;
    int i = 0 ;
    while(s[i]){
        hash += (int)(s[i]);
        hash =(hash*17)%n;
        ++i;
    }
    return hash%n;
}
```

同时采用“平方探测法”来解决冲突。查询时也使用“平方探测法”查询。

代码：

```
1. #include"query.h"
2. using namespace std;
3. #include<iostream>
4.
5. int myhash(const string s,int n){
6.     int hash = 0 ;
7.     int i = 0 ;
8.     while(s[i]){
```

```
        hash += (int)(s[i]);
        hash =(hash*17)%n;
        ++i;
    }
    return hash%n;
}

void query(string A[], int n, string B[], int m){
    int N=n*2;//1.2*n 时基本能过
    string hash_A[N];
    for(int i = 0 ; i < n ; ++i){  //将 A[]存进入 hash_A[N]
        int key = myhash(A[i],N);
        int k = 0;
        while(hash_A[(key+k*k)%N]!=""){//平方探测法
            ++k;
        }
        hash_A[(key+k*k)%N]=A[i] ;
    }
    for(int i = 0 ; i < m ; ++i){ //查询 B[]中元素是否在 hash_A[N]中
        int key = myhash(B[i],N);
        int k = 0;
        while(hash_A[(key+k*k)%N]!= ""){
            if(B[i]==hash_A[(key+k*k)%N]){
                cout<<B[i]<<endl;
                break;
            }
            ++k;
        }
    }
}
```

2）爸爸去哪儿之一

设计思路：

由于题目要求：1）将字母转化成数字（如 a=1，b=2）并求和来实现字符串的哈希函数；2）使用“线性探测法”解决冲突；3）并输出平均查找长度。

1）2）直接就按题目要求实现了，3）我们通过逆向求解的思路，想要求平均查找长度，先求出总的查找长度，再求平均值。要求总的查找长度，即可以求每次在存数据时的查找长度之和。初始时定义总查找长度 double length = n，

每次存数据时如果发现当前位置非空就++length。最后平均查找长度为 length/n.

由于平均查找长度的输出要保留 3 位小数，故使用

#include<iomanip>

cout<<fixed<<setprecision(3)<<aver_length<<endl;

输出平均查找长度。

代码：

```
1.  #include<iostream>
2.using namespace std;
3.#include<iomanip>
4.#include<string>
5.
6.int otoi(const char c){
7.    return (int)c-96 ;
8.}
9.int numName_mod_m(const string name,int m){
10.    int sum = 0 ;
11.    for(int i = 0 ; i < name.size();++i)
12.        sum+=otoi(name[i]);
13.    return sum%m;
14.}
15.
16.int main(){
17.    int n,m;
18.    cin>>n>>m;
19.    string room[m];
20.    for(int i = 0 ; i < m ; ++i){
21.        room[i]="NULL";
22.    }
23.    double length = n ;
24.    for(int i = 0 ; i < n ; ++i){
25.        string name;
26.        cin>>name;
27.        int key = numName_mod_m(name,m);
28.        int k = 0;
29.        while(room[(key+k)%m]!="NULL"){
30.            ++k ;
31.            ++length;
32.        }
```

```
        room[(key+k)%m]= name;
    }
    for(int i = 0 ; i < m ; ++i)
        cout<<i<<":"<<room[i]<<endl;
    double aver_length = length/n;
    cout<<fixed<<setprecision(3)<<aver_length<<endl;
}
```

3）爸爸去哪儿之二

设计思路：

同“2）爸爸去哪儿之一”，将冲突解决方法改成“平方探测法”。

代码：

```
#include<iostream>
using namespace std;
#include<iomanip>
#include<string>

int otoi(const char c){
    return (int)c-96 ;
}
int numName_mod_m(const string name,int m){
    int sum = 0 ;
    for(int i = 0 ; i < name.size();++i)
        sum+=otoi(name[i]);
    return sum%m;
}

int main(){
    int n,m;
    cin>>n>>m;
    string room[m];
    for(int i = 0 ; i < m ; ++i){
        room[i]="NULL";
    }
    double length = n ;
    for(int i = 0 ; i < n ; ++i){
        string name;
```

```
        cin>>name;
        int key = numName_mod_m(name,m);
        int k = 0;
        while(room[(key+k*k)%m]!="NULL"){
            ++k;
            ++length;
        }
        room[(key+k*k)%m]= name;
    }
    for(int i = 0 ; i < m ; ++i)
        cout<<i<<":"<<room[i]<<endl;
    double aver_length = length/n;
    cout<<fixed<<setprecision(3)<<aver_length<<endl;
}
```

4.程序运行与测试

1）Query

序号	测试输入 (n,m,A[n],B[m])	输出	是否通过
1	3 4 ab ac ad bc ab as ad ds	ab ad	是
2	5 6 abc acd bsc aab ass awd das bscc bsc abc asd	bsc abc	是
3	4 3 ABC ADS asd Abc ads abc ABC	ABC	是

2）爸爸去哪儿之一

序号	输入	输出	是否通过
1	1.100 150 2.aaron 3.abbott 4.abel 5.abner 6.abraham 7.adair 8.adam 9.addison 10.adolph 11.adonis 12.adrian 13.ahern 14.alan 15.albert 16.aldrich 17.alexander 18.alfred 19.alger 20.algernon 21.allen 22.alston 23.alva 24.alvin 25.alvis 26.amos 27.andre 28.andrew 29.andy 30.angelo 31.augus 32.ansel 33.antony 34.antoine 35.antonio 36.archer 37.archibald 38.aries 39.arlen 40.armand 41.armstrong 42.arno 43.arnold 44.arthur	1.0:NULL 2.1:NULL 3.2:NULL 4.3:NULL 5.4:NULL 6.5:NULL 7.6:NULL 8.7:NULL 9.8:NULL 10.9:ed 11.10:NULL 12.11:NULL 13.12:NULL 14.13:NULL 15.14:NULL 16.15:NULL 17.16:chad 18.17:NULL 19.18:NULL 20.19:adam 21.20:abel 22.21:asa 23.22:beck 24.23:ben 25.24:dana 26.25:bard 27.26:dean 28.27:NULL 29.28:alan 30.29:beau 31.30:eden 32.31:cash 33.32:cecil 34.33:adair 35.34:baird 36.35:carl 37.36:alva 38.37:dave 39.38:earl 40.39:edgar 41.40:abner 42.41:bart 43.42:andre 44.43:alger	是

		45. arvin		45. 44:abraham	
		46. asa		46. 45:allen	
		47. ashbur		47. 46:ahern	
		48. atwood		48. 47:adrian	
		49. aubrey		49. 48:alfred	
		50. august		50. 49:aaron	
		51. augustine		51. 50:amos	
		52. avery		52. 51:andy	
		53. baird		53. 52:ansel	
		54. baldwin		54. 53:archer	
		55. bancroft		55. 54:angelo	
		56. bard		56. 55:aldrich	
		57. barlow		57. 56:adolph	
		58. barnett		58. 57:aries	
		59. baron		59. 58:albert	
		60. barret		60. 59:alvin	
		61. barry		61. 60:abbott	
		62. bartholome		62. 61:archibald	
		63. bart		63. 62:adonis	
		64. barton		64. 63:alvis	
		65. bartley		65. 64:arlen	
		66. basil		66. 65:andrew	
		67. beacher		67. 66:addison	
		68. beau		68. 67:armand	
		69. beck		69. 68:arno	
		70. ben		70. 69:augus	
		71. caesar		71. 70:arnold	
		72. calvin		72. 71:arvin	
		73. carey		73. 72:ashbur	
		74. carl		74. 73:aubrey	
		75. carr		75. 74:avery	
		76. carter		76. 75:baldwin	
		77. cash		77. 76:barlow	
		78. cecil		78. 77:baron	
		79. cedric		79. 78:antoine	
		80. chad		80. 79:atwood	
		81. channing		81. 80:bancroft	
		82. chapman		82. 81:alston	
		83. dana		83. 82:barnett	
		84. daniel		84. 83:barret	
		85. darcy		85. 84:alexander	
		86. darnell		86. 85:barry	
		87. darren		87. 86:algernon	
		88. dave		88. 87:arthur	
		89. david		89. 88:antonio	
		90. dean		90. 89:antony	

		91. dempsey 92. dennis 93. earl 94. ed 95. eden 96. edgar 97. edmund 98. edison 99. edward 100. edwiin 101. egbert		91. 90:august 92. 91:barton 93. 92:bartley 94. 93:basil 95. 94:beacher 96. 95:caesar 97. 96:calvin 98. 97:carey 99. 98:carr 100. 99:carter 101. 100:cedric 102. 101:channing 103. 102:chapman 104. 103:daniel 105. 104:darcy 106. 105:darnell 107. 106:darren 108. 107:david 109. 108:dempsey 110. 109:bartholome 111. 110:dennis 112. 111:edmund 113. 112:edison 114. 113:edward 115. 114:edwiin 116. 115:egbert 117. 116:NULL 118. 117:augustine 119. 118:NULL 120. 119:NULL 121. 120:NULL 122. 121:NULL 123. 122:NULL 124. 123:NULL 125. 124:NULL 126. 125:armstrong 127. 126:NULL 128. 127:NULL 129. 128:NULL 130. 129:NULL 131. 130:NULL 132. 131:NULL 133. 132:NULL 134. 133:NULL 135. 134:NULL 136. 135:NULL	

		136:NULL 137:NULL 138:NULL 139:NULL 140:NULL 141:NULL 142:NULL 143:NULL 144:NULL 145:NULL 146:NULL 147:NULL 148:NULL 149:NULL 13.810	
2	5 11 ee df fd gc cg	0:df 1:fd 2:gc 3:cg 4:NULL 5:NULL 6:NULL 7:NULL 8:NULL 9:NULL 10:ee 3.000	是

3）爸爸去哪儿之二

序号	输入	输出	是否通过
1	100 150 aaron abbott abel abner abraham	0:NULL 1:NULL 2:NULL 3:NULL 4:NULL 5:NULL	是

		7. adair		7. 6:NULL	
		8. adam		8. 7:NULL	
		9. addison		9. 8:NULL	
		10. adolph		10. 9:ed	
		11. adonis		11. 10:NULL	
		12. adrian		12. 11:NULL	
		13. ahern		13. 12:NULL	
		14. alan		14. 13:NULL	
		15. albert		15. 14:NULL	
		16. aldrich		16. 15:NULL	
		17. alexander		17. 16:chad	
		18. alfred		18. 17:NULL	
		19. alger		19. 18:NULL	
		20. algernon		20. 19:adam	
		21. allen		21. 20:abel	
		22. alston		22. 21:asa	
		23. alva		23. 22:beck	
		24. alvin		24. 23:NULL	
		25. alvis		25. 24:dana	
		26. amos		26. 25:bard	
		27. andre		27. 26:NULL	
		28. andrew		28. 27:NULL	
		29. andy		29. 28:alan	
		30. angelo		30. 29:beau	
		31. augus		31. 30:ben	
		32. ansel		32. 31:cash	
		33. antony		33. 32:cecil	
		34. antoine		34. 33:adair	
		35. antonio		35. 34:baird	
		36. archer		36. 35:carl	
		37. archibald		37. 36:alva	
		38. aries		38. 37:earl	
		39. arlen		39. 38:NULL	
		40. armand		40. 39:edgar	
		41. armstrong		41. 40:abner	
		42. arno		42. 41:bart	
		43. arnold		43. 42:andre	
		44. arthur		44. 43:alger	
		45. arvin		45. 44:abraham	
		46. asa		46. 45:allen	
		47. ashbur		47. 46:ahern	
		48. atwood		48. 47:adrian	
		49. aubrey		49. 48:amos	
		50. august		50. 49:aaron	
		51. augustine		51. 50:alfred	
		52. avery		52. 51:ansel	

		53. baird		53. 52:aries	
		54. baldwin		54. 53:andy	
		55. bancroft		55. 54:angelo	
		56. bard		56. 55:aldrich	
		57. barlow		57. 56:adolph	
		58. barnett		58. 57:archer	
		59. baron		59. 58:albert	
		60. barret		60. 59:alvin	
		61. barry		61. 60:abbott	
		62. bartholome		62. 61:calvin	
		63. bart		63. 62:adonis	
		64. barton		64. 63:alvis	
		65. bartley		65. 64:arno	
		66. basil		66. 65:andrew	
		67. beacher		67. 66:addison	
		68. beau		68. 67:archibald	
		69. beck		69. 68:arnold	
		70. ben		70. 69:augus	
		71. caesar		71. 70:ashbur	
		72. calvin		72. 71:avery	
		73. carey		73. 72:aubrey	
		74. carl		74. 73:arvin	
		75. carr		75. 74:baldwin	
		76. carter		76. 75:arlen	
		77. cash		77. 76:armand	
		78. cecil		78. 77:carey	
		79. cedric		79. 78:antoine	
		80. chad		80. 79:atwood	
		81. channing		81. 80:bancroft	
		82. chapman		82. 81:alston	
		83. dana		83. 82:darnell	
		84. daniel		84. 83:bartley	
		85. darcy		85. 84:alexander	
		86. darnell		86. 85:darren	
		87. darren		87. 86:algernon	
		88. dave		88. 87:arthur	
		89. david		89. 88:antonio	
		90. dean		90. 89:antony	
		91. dempsey		91. 90:august	
		92. dennis		92. 91:beacher	
		93. earl		93. 92:basil	
		94. ed		94. 93:egbert	
		95. eden		95. 94:daniel	
		96. edgar		96. 95:barton	
		97. edmund		97. 96:barlow	
		98. edison		98. 97:edmund	

```
99. edward
100.   edwiin
101.   egbert
```

```
99. 98:NULL
100.   99:baron
101.   100:barret
102.   101:carter
103.   102:edison
104.   103:dempsey
105.   104:carr
106.   105:barnett
107.   106:cedric
108.   107:NULL
109.   108:NULL
110.   109:bartholome
111.   110:NULL
112.   111:caesar
113.   112:NULL
114.   113:barry
115.   114:dennis
116.   115:darcy
117.   116:NULL
118.   117:augustine
119.   118:NULL
120.   119:channing
121.   120:chapman
122.   121:david
123.   122:NULL
124.   123:NULL
125.   124:dean
126.   125:armstrong
127.   126:NULL
128.   127:NULL
129.   128:eden
130.   129:NULL
131.   130:NULL
132.   131:NULL
133.   132:dave
134.   133:NULL
135.   134:NULL
136.   135:NULL
137.   136:edward
138.   137:NULL
139.   138:NULL
140.   139:NULL
141.   140:NULL
142.   141:NULL
143.   142:NULL
144.   143:NULL
```

		145. 144:NULL 146. 145:edwiin 147. 146:NULL 148. 147:NULL 149. 148:NULL 150. 149:NULL 151. 3.580	
2	1. 5 11 2. ee 3. df 4. fd 5. gc 6. cg	1. 0:df 2. 1:NULL 3. 2:NULL 4. 3:fd 5. 4:cg 6. 5:NULL 7. 6:NULL 8. 7:NULL 9. 8:gc 10. 9:NULL 11. 10:ee 12. 3.000	是

5.实验总结与心得

哈希表中查询元素的速度取决于两个方面：1）哈希函数生成的散列值分布的均匀程度 2）哈希表空间的大小。实质上都是减少冲突。

刚开始我做 1）Query 的时候，总是运行时间过长，开始我以为是我设计的哈希函数生成的散列值不够均匀，于是我设计两个不同的哈希函数取返回值之和来得到对应的散列值，但是这样也有时能过，有时不能过。

后来我再仔细看我的代码，发现第二个哈希函数我根本没有到，写错了函数名，用的仍然是第一个哈希函数，这时我发现我能过的原因是给哈希表分配的空间是 2*n，所以问题出在了这！当空间太小时无论多么完美的哈希函数总

是不可避免地产生冲突。所以我删去了第二个哈希函数，只用第一个哈希函数生成散列值，发现完全能过，但是有点慢，我再看下我的哈希函数，发现“进制 101”给的太大了，导致生成散列值的计算时间过长，于是改成更小的质数如 7，13，17 都明显快了许多。再在后来的测试中，发现哈希表分配 1.2*n 的空间时就基本能过，推测是测试样例给的有点少。

所以当我们好像解决某个问题的时候，事实上可能并没有通过我们认为的解决方式解决该问题。

附录、提交文件清单