Christopher Aboshear

Data structures 3319

4/20/19

Successfully completed “C” option

**PART A.)**

Average Number of Probes for first 30: 8.83333E+00

Min Number of Probes for First 30: 1

Max Number of Probes for First 30: 27

Average Number of Probes for Last 30: 3.86667E+01

Min Number of Probes for Last 30: 17

Max Number of Probes for Last 30: 59

Expected Result: 1.50000E+00

INDEX HASH KEY HASH ADDRESS NUMBER OF PROBES

0 -1 0

1 -1 0

2 -1 0

3 -1 0

4 -1 0

5 -1 0

6 -1 0

7 -1 0

8 -1 0

9 -1 0

10 -1 0

11 -1 0

12 -1 0

13 -1 0

14 -1 0

15 -1 0

16 -1 0

17 -1 0

18 -1 0

19 -1 0

20 -1 0

21 -1 0

22 -1 0

23 -1 0

24 -1 0

25 -1 0

26 -1 0

27 -1 0

28 -1 0

29 -1 0

30 -1 0

31 -1 0

32 -1 0

33 -1 0

34 -1 0

35 -1 0

36 -1 0

37 -1 0

38 -1 0

39 -1 0

40 -1 0

41 -1 0

42 -1 0

43 -1 0

44 -1 0

45 -1 0

46 -1 0

47 -1 0

48 -1 0

49 1234567890123456 49 1

50 -1 0

51 -1 0

52 -1 0

53 -1 0

54 -1 0

55 -1 0

56 -1 0

57 -1 0

58 -1 0

59 -1 0

60 -1 0

61 -1 0

62 -1 0

63 -1 0

64 -1 0

65 Aguirrie 65 1

66 Alcantara 65 2

67 Bhandari 66 2

68 Carmona 67 2

69 Casper 67 3

70 Cook 67 4

71 Daniels 68 4

72 Acevedo 65 8

73 Ajose 65 9

74 Arauza 65 10

75 Buck 66 10

76 Clark 67 10

77 Crouch 67 11

78 Nienberg 78 1

79 Davies 68 12

80 Paschal 80 1

81 Qamruddin 81 1

82 Red 82 1

83 Salkowski 83 1

84 Dugger 68 17

85 Egbe 69 17

86 Ellington 69 18

87 Farral 70 18

88 Garza 71 18

89 Gurung 71 19

90 Zulfiqar 90 1

91 Joseph 74 18

92 Kelly 75 18

93 Corey 67 27

94 Adam 65 30

95 Clayton 67 29

96 Dustin 68 29

97 Robert 82 16

98 Kyle 75 24

99 Scott 83 17

100 Octavio 79 22

101 Judy 74 28

102 Derek 68 35

103 Jeffrey 74 30

104 Jordon 74 31

105 Vinnela 86 20

106 Lisa 76 31

107 Todd 84 24

108 Veronica 86 23

109 Matthew 77 33

110 Michael 77 34

111 Akhila 65 47

112 John 74 39

113 Charles 67 47

114 James 74 41

115 Chris 67 49

116 Wade 87 30

117 Christopher 67 51

118 Fernando 70 49

119 Batbold 66 54

120 Joel 74 47

121 Fabulous 70 52

122 Misogamist 77 46

123 Maiden 77 47

124 Eye 69 56

125 Constriction 67 59

126 Necromancer 78 49

127 Syncopate 83 45

**Theoretical vs Empirical results: Looking at this table the expected number of probes is at 1.5. The first 30 keys yielded about 8.83 probes and the last 30 keys entries had an average of 38.6. To help explain the jump in probes from the first 30 keys to the last 30 keys we can reference the Saint Petersburg Paradox. The theory is if 23 people are in a room then there is around a 49% chance no two people will have the same birthday. Hence going past 23, in our case first 30 and last 30 keys, will produce more and more collisions. Although there is a jump from 1.5 to 8.83 likely due to some slight clustering for the first 30 keys, we can observe the big jump from 1.5 to 38.6 due to the primary clustering that is occurring in the last 30 entries.**

**Part B.)**

Average Number of Probes for first 30: 8.83333E+00

Min Number of Probes for First 30: 1

Max Number of Probes for First 30: 27

Average Number of Probes for Last 30: 5.44333E+01

Min Number of Probes for Last 30: 34

Max Number of Probes for Last 30: 75

Expected Result: 5.50000E+00

INDEX HASH KEY HASH ADDRESS NUMBER OF PROBES

0 Yolk 89 40

1 Afterwards 65 65

2 Person 80 51

3 Northwest 78 54

4 Irreversible 73 60

5 Fabricate 70 64

6 Honor 72 63

7 Staple 83 53

8 Under 85 52

9 Jutty 74 64

10 Finagle 70 69

11 Cook 67 73

12 Rush 82 59

13 Wine 87 55

14 Screen 83 60

15 Perfect 80 64

16 mole 109 36

17 parasympathetic 112 34

18 poison 112 35

19 brutalize 98 50

20 cap 99 50

21 ratiocination 114 36

22 cauldron 99 52

23 prepossess 112 40

24 wince 119 34

25 orthodontist 111 43

26 live 108 47

27 magnetic 109 47

28 inlet 105 52

29 constrain 99 59

30 marsupial 109 50

31 rationalize 114 46

32 scat 115 46

33 toluene 116 46

34 wet 119 44

35 sparse 115 49

36 quandary 113 52

37 dactyl 100 66

38 nosegay 110 57

39 option 111 57

40 forgetful 102 67

41 privilege 112 58

42 sponsor 115 56

43 exhilarate 101 71

44 guard 103 70

45 noggin 110 64

46 prologue 112 63

47 seal 115 61

48 seat 115 62

49 1234567890123456 49 1

50 tiller 116 63

51 ichthyosaur 105 75

52 -1 0

53 -1 0

54 -1 0

55 -1 0

56 -1 0

57 -1 0

58 -1 0

59 -1 0

60 -1 0

61 -1 0

62 -1 0

63 -1 0

64 -1 0

65 Aguirrie 65 1

66 Alcantara 65 2

67 Bhandari 66 2

68 Carmona 67 2

69 Casper 67 3

70 Cook 67 4

71 Daniels 68 4

72 Acevedo 65 8

73 Ajose 65 9

74 Arauza 65 10

75 Buck 66 10

76 Clark 67 10

77 Crouch 67 11

78 Nienberg 78 1

79 Davies 68 12

80 Paschal 80 1

81 Qamruddin 81 1

82 Red 82 1

83 Salkowski 83 1

84 Dugger 68 17

85 Egbe 69 17

86 Ellington 69 18

87 Farral 70 18

88 Garza 71 18

89 Gurung 71 19

90 Zulfiqar 90 1

91 Joseph 74 18

92 Kelly 75 18

93 Corey 67 27

94 Adam 65 30

95 Clayton 67 29

96 Dustin 68 29

97 Robert 82 16

98 Kyle 75 24

99 Scott 83 17

100 Octavio 79 22

101 Judy 74 28

102 Derek 68 35

103 Jeffrey 74 30

104 Jordon 74 31

105 Vinnela 86 20

106 Lisa 76 31

107 Todd 84 24

108 Veronica 86 23

109 Matthew 77 33

110 Michael 77 34

111 Akhila 65 47

112 John 74 39

113 Charles 67 47

114 James 74 41

115 Chris 67 49

116 Wade 87 30

117 Christopher 67 51

118 Fernando 70 49

119 Batbold 66 54

120 Joel 74 47

121 Fabulous 70 52

122 Misogamist 77 46

123 Maiden 77 47

124 Eye 69 56

125 Constriction 67 59

126 Necromancer 78 49

127 Syncopate 83 45

**Theoretical vs Empirical results: In this table we can see expected number of probes is at 5.5. Similar to the 50% full table the 90% full table has 8.83 probs for the first 30 keys. Unlike the 50% table the 90% full table has 54.4 expected probes for the last 30 keys. Keeping the Saint Petersburg Paradox in mind from the first table explanation we can shed some light as to why the last 30 keys have such a high discrepancy between expected probes and actual probes. Having the table 90% full leads to having such high collision count in the last 30 keys due to the nature of linear probes. Linear probes sequentially check if spots are used and traverse downward until an open spot is available, hence explaining clustering in the last 30 keys. So, having the table 90% full is what leads to the huge primary cluster in the last 30 keys in the above table. Thus, the linear probe technique is the reason the first 30 keys are the same number of probes at 50% & 90%, but the last 30 keys in the 90% are higher than the 50%.**

**Part C.)**

Average Number of Probes for first 30: 2.93333E+00

Min Number of Probes for First 30: 1

Max Number of Probes for First 30: 8

Average Number of Probes for Last 30: 6.70000E+00

Min Number of Probes for Last 30: 1

Max Number of Probes for Last 30: 14

Expected Result: 1.38629E+00

INDEX HASH KEY HASH ADDRESS NUMBER OF PROBES

0 -1 0

1 Chris 67 11

2 -1 0

3 Adam 65 7

4 -1 0

5 Crouch 67 7

6 Dustin 68 7

7 -1 0

8 James 74 11

9 -1 0

10 -1 0

11 Maiden 77 11

12 Jeffrey 74 7

13 -1 0

14 Corey 67 8

15 Derek 68 8

16 -1 0

17 Fabulous 70 8

18 -1 0

19 -1 0

20 -1 0

21 Jordon 74 8

22 -1 0

23 -1 0

24 Michael 77 8

25 -1 0

26 -1 0

27 -1 0

28 Charles 67 10

29 -1 0

30 -1 0

31 -1 0

32 -1 0

33 -1 0

34 -1 0

35 John 74 10

36 -1 0

37 -1 0

38 Misogamist 77 10

39 -1 0

40 Constriction 67 14

41 -1 0

42 -1 0

43 -1 0

44 -1 0

45 -1 0

46 -1 0

47 -1 0

48 -1 0

49 1234567890123456 49 1

50 -1 0

51 -1 0

52 -1 0

53 -1 0

54 -1 0

55 -1 0

56 -1 0

57 Akhila 65 9

58 -1 0

59 Clayton 67 9

60 -1 0

61 -1 0

62 -1 0

63 -1 0

64 -1 0

65 Aguirrie 65 1

66 Alcantara 65 2

67 Bhandari 66 2

68 Carmona 67 2

69 Daniels 68 2

70 Egbe 69 2

71 Acevedo 65 3

72 Buck 66 3

73 Casper 67 3

74 Davies 68 3

75 Ellington 69 3

76 Farral 70 3

77 Garza 71 3

78 Nienberg 78 1

79 Octavio 79 1

80 Paschal 80 1

81 Qamruddin 81 1

82 Red 82 1

83 Salkowski 83 1

84 Scott 83 2

85 Todd 84 2

86 Vinnela 86 1

87 Judy 74 6

88 Robert 82 3

89 Syncopate 83 3

90 Zulfiqar 90 1

91 -1 0

92 Veronica 86 3

93 Arauza 65 5

94 Joel 74 13

95 Clark 67 5

96 Ajose 65 4

97 Batbold 66 4

98 Cook 67 4

99 Dugger 68 4

100 Eye 69 4

101 Fernando 70 4

102 Gurung 71 4

103 Kyle 75 5

104 -1 0

105 Joseph 74 4

106 Kelly 75 4

107 Lisa 76 4

108 Matthew 77 4

109 Necromancer 78 4

110 -1 0

111 -1 0

112 -1 0

113 -1 0

114 -1 0

115 -1 0

116 -1 0

117 -1 0

118 Wade 87 4

119 -1 0

120 -1 0

121 -1 0

122 Christopher 67 12

123 -1 0

124 -1 0

125 -1 0

126 -1 0

127 -1 0

**Theoretical vs Empirical results: In this table we can see expected number of probes is at 1.38. The random probe technique gives us 2.93 for the first 30 keys and 6.7 for the last 30 keys. Thinking about our results from the linear probe technique and the results it yielded, we can compare its efficiency and collision handling to the random probe. Random probe shows lower expected and actual number of probes compared to linear. If we consider the nature of random probes, we can explain the difference in the empirical and theoretical results. Random probes use a random number generator as a tool to “jump around” the table when inserting. Unlike, the linear which sequentially traverses. What does this mean? Basically, the random number generator is what helps prevent primary clustering. This helps minimize the number of collisions that occur, but when collisions do occur, we are more likely to have some secondary clustering instead of primary. Thus, explaining the lower number of probes expected and used in the random technique compared to linear.**

**Random at 90%**

Average Number of Probes for first 30: 2.93333E+00

Min Number of Probes for First 30: 1

Max Number of Probes for First 30: 8

Average Number of Probes for Last 30: 7.66667E+00

Min Number of Probes for Last 30: 1

Max Number of Probes for Last 30: 18

Expected Result: 2.55843E+00

INDEX HASH KEY HASH ADDRESS NUMBER OF PROBES

0 scat 115 6

1 Chris 67 11

2 cap 99 4

3 Adam 65 7

4 exhilarate 101 4

5 Crouch 67 7

6 Dustin 68 7

7 seal 115 13

8 James 74 11

9 magnetic 109 5

10 Honor 72 7

11 Maiden 77 11

12 Jeffrey 74 7

13 nosegay 110 4

14 Corey 67 8

15 Derek 68 8

16 quandary 113 4

17 Fabulous 70 8

18 Perfect 80 7

19 toluene 116 4

20 -1 0

21 Jordon 74 8

22 wet 119 4

23 -1 0

24 Michael 77 8

25 -1 0

26 Afterwards 65 10

27 -1 0

28 Charles 67 10

29 -1 0

30 Screen 83 8

31 Finagle 70 10

32 ichthyosaur 105 12

33 prologue 112 18

34 dactyl 100 11

35 John 74 10

36 seat 115 18

37 cauldron 99 7

38 Misogamist 77 10

39 privilege 112 12

40 Constriction 67 14

41 guard 103 7

42 sponsor 115 12

43 inlet 105 7

44 -1 0

45 -1 0

46 constrain 99 8

47 Jutty 74 14

48 -1 0

49 1234567890123456 49 1

50 prepossess 112 7

51 -1 0

52 rationalize 114 7

53 sparse 115 7

54 tiller 116 7

55 -1 0

56 marsupial 109 8

57 Akhila 65 9

58 -1 0

59 Clayton 67 9

60 -1 0

61 Cook 67 15

62 Fabricate 70 9

63 forgetful 102 10

64 -1 0

65 Aguirrie 65 1

66 Alcantara 65 2

67 Bhandari 66 2

68 Carmona 67 2

69 Daniels 68 2

70 Egbe 69 2

71 Acevedo 65 3

72 Buck 66 3

73 Casper 67 3

74 Davies 68 3

75 Ellington 69 3

76 Farral 70 3

77 Garza 71 3

78 Nienberg 78 1

79 Octavio 79 1

80 Paschal 80 1

81 Qamruddin 81 1

82 Red 82 1

83 Salkowski 83 1

84 Scott 83 2

85 Todd 84 2

86 Vinnela 86 1

87 Judy 74 6

88 Robert 82 3

89 Syncopate 83 3

90 Zulfiqar 90 1

91 Northwest 78 6

92 Veronica 86 3

93 Arauza 65 5

94 Joel 74 13

95 Clark 67 5

96 Ajose 65 4

97 Batbold 66 4

98 Cook 67 4

99 Dugger 68 4

100 Eye 69 4

101 Fernando 70 4

102 Gurung 71 4

103 Kyle 75 5

104 Irreversible 73 4

105 Joseph 74 4

106 Kelly 75 4

107 Lisa 76 4

108 Matthew 77 4

109 Necromancer 78 4

110 mole 109 2

111 Person 80 4

112 parasympathetic 112 1

113 Rush 82 4

114 Staple 83 4

115 Wine 87 5

116 Under 85 4

117 orthodontist 111 3

118 Wade 87 4

119 wince 119 1

120 Yolk 89 4

121 live 108 6

122 Christopher 67 12

123 noggin 110 6

124 option 111 6

125 poison 112 6

126 brutalize 98 5

127 ratiocination 114 6

**Theoretical vs Empirical results: In this table we can see expected number of probes is at 2.55. The number of probes for the first 30 keys are 2.93, as in the 50% full table, and 7.66 probes for the last 30 keys. Unlike the linear probe technique, we can see, from 50% to 90% the random probe did not have a huge increase in the number of probes for the last 30 keys. It only went up by about 1. This is due to the beauty of the random probe! As stated previously, the random number generator helps spread out the entries to prevent primary clusters and have secondary clusters spread throughout the table. This is why on a larger scale, regardless of how full a table is, the random probe technique will almost always outperform the linear technique in avoiding collisions.**

**Part D.)**

**Let us some up all of the data we collected. For linear technique the number of probes for the first 30 entries does not change when you increase how full the table is from 50% to 90%. We attributed this to the nature of linear probe technique, which is to sequentially traverse, and slightly to the Saint Petersburg Paradox. We then observe the huge discrepancy in the last 30 keys for linear when we increase from 50% to 90% table filled. This is where the nature of the linear probe technique is its greatest weakness. When a spot is filled it must sequentially traverse straight “downward” until and open spot is available. This is why there is such a large amount of primary clustering in the last 30 keys compared to the first 30 when it comes to the linear probe. Now we will talk about random probe technique. First, lets observe a key characteristic of the random number generator. This feature helps the random probe technique avoid the issue produced by the linear probe. Which was primary clustering in the last 30 keys. It used the generator to “jump” to an open spot on the table instead of sequentially traversing down until finding a spot. This key characteristic attributed to secondary clusters throughout the table. Hence, why the random probe had a very small increase in number of probes from 50% to 90% in the last 30 keys. In conclusion, generally the random probe technique will yield a lot less collisions than linear because it tries to have secondary clusters spread throughout the table instead of large primary clusters produced by linear probes.**

**Part E.)**

**Linear at 50%**

Average Number of Probes for first 30: 1.13333E+00

Min Number of Probes for First 30: 1

Max Number of Probes for First 30: 2

Average Number of Probes for Last 30: 2.00000E+00

Min Number of Probes for Last 30: 1

Max Number of Probes for Last 30: 6

Expected Result: 1.50000E+00

INDEX HASH KEY HASH ADDRESS NUMBER OF PROBES

0 -1 0

1 -1 0

2 -1 0

3 -1 0

4 -1 0

5 -1 0

6 Crouch 6 1

7 -1 0

8 -1 0

9 -1 0

10 Aguirrie 10 1

11 -1 0

12 Red 12 1

13 John 12 2

14 -1 0

15 -1 0

16 Lisa 16 1

17 -1 0

18 Alcantara 18 1

19 Michael 18 2

20 Misogamist 18 3

21 -1 0

22 -1 0

23 -1 0

24 -1 0

25 -1 0

26 -1 0

27 -1 0

28 Zulfiqar 28 1

29 Jeffrey 28 2

30 Corey 30 1

31 Kyle 30 2

32 Dugger 32 1

33 -1 0

34 1234567890123456 34 1

35 Arauza 34 2

36 Eye 34 3

37 -1 0

38 -1 0

39 -1 0

40 -1 0

41 -1 0

42 Ellington 42 1

43 -1 0

44 Jordon 44 1

45 -1 0

46 Carmona 46 1

47 Scott 46 2

48 -1 0

49 -1 0

50 Adam 50 1

51 Maiden 50 2

52 Nienberg 52 1

53 Farral 52 2

54 Garza 54 1

55 Fabulous 52 4

56 -1 0

57 -1 0

58 -1 0

59 -1 0

60 Joseph 60 1

61 Robert 60 2

62 Casper 62 1

63 -1 0

64 -1 0

65 -1 0

66 Ajose 66 1

67 -1 0

68 -1 0

69 -1 0

70 -1 0

71 -1 0

72 -1 0

73 -1 0

74 -1 0

75 -1 0

76 Batbold 76 1

77 -1 0

78 Syncopate 78 1

79 -1 0

80 Derek 80 1

81 -1 0

82 -1 0

83 -1 0

84 Bhandari 84 1

85 Vinnela 84 2

86 Gurung 86 1

87 Charles 86 2

88 Paschal 88 1

89 Fernando 84 6

90 Akhila 90 1

91 -1 0

92 Buck 92 1

93 Joel 92 2

94 Constriction 94 1

95 -1 0

96 Dustin 96 1

97 Todd 96 2

98 Egbe 98 1

99 Matthew 98 2

100 -1 0

101 -1 0

102 Chris 102 1

103 Christopher 102 2

104 -1 0

105 -1 0

106 Acevedo 106 1

107 -1 0

108 James 108 1

109 -1 0

110 Cook 110 1

111 Salkowski 110 2

112 Daniels 112 1

113 Davies 112 2

114 -1 0

115 -1 0

116 Veronica 116 1

117 Necromancer 116 2

118 Clark 118 1

119 Clayton 118 2

120 Octavio 118 3

121 Wade 118 4

122 Qamruddin 122 1

123 -1 0

124 Judy 124 1

125 -1 0

126 Kelly 126 1

127 -1 0

**Theoretical vs Empirical results: In this table we can see expected number of probes is at 1.5. The number of probes for the first 30 keys are 1.13 and 2.0 probes for the last 30 keys. The first 30 exceeds expectations and the last 30 is only slightly worse.**

**Linear at 90%**

Average Number of Probes for first 30: 1.13333E+00

Min Number of Probes for First 30: 1

Max Number of Probes for First 30: 2

Average Number of Probes for Last 30: 1.43667E+01

Min Number of Probes for Last 30: 1

Max Number of Probes for Last 30: 58

Expected Result: 5.50000E+00

INDEX HASH KEY HASH ADDRESS NUMBER OF PROBES

0 live 0 1

1 wet 86 44

2 inlet 2 1

3 marsupial 2 2

4 sparse 118 15

5 dactyl 0 6

6 Crouch 6 1

7 nosegay 4 4

8 parasympathetic 8 1

9 forgetful 100 38

10 Aguirrie 10 1

11 privilege 0 12

12 Red 12 1

13 John 12 2

14 scat 14 1

15 sponsor 86 58

16 Lisa 16 1

17 exhilarate 10 8

18 Alcantara 18 1

19 Michael 18 2

20 Misogamist 18 3

21 mole 18 4

22 wince 22 1

23 guard 6 18

24 prologue 96 57

25 tiller 112 42

26 quandary 26 1

27 ichthyosaur 106 50

28 Zulfiqar 28 1

29 Jeffrey 28 2

30 Corey 30 1

31 Kyle 30 2

32 Dugger 32 1

33 cauldron 30 4

34 1234567890123456 34 1

35 Arauza 34 2

36 Eye 34 3

37 brutalize 36 2

38 constrain 30 9

39 toluene 32 8

40 option 30 11

41 seal 30 12

42 Ellington 42 1

43 seat 30 14

44 Jordon 44 1

45 ratiocination 44 2

46 Carmona 46 1

47 Scott 46 2

48 orthodontist 46 3

49 rationalize 44 6

50 Adam 50 1

51 Maiden 50 2

52 Nienberg 52 1

53 Farral 52 2

54 Garza 54 1

55 Fabulous 52 4

56 Northwest 52 5

57 Fabricate 52 6

58 Finagle 52 7

59 -1 0

60 Joseph 60 1

61 Robert 60 2

62 Casper 62 1

63 -1 0

64 prepossess 64 1

65 -1 0

66 Ajose 66 1

67 Irreversible 66 2

68 noggin 68 1

69 -1 0

70 -1 0

71 -1 0

72 -1 0

73 -1 0

74 -1 0

75 -1 0

76 Batbold 76 1

77 -1 0

78 Syncopate 78 1

79 cap 78 2

80 Derek 80 1

81 -1 0

82 magnetic 82 1

83 -1 0

84 Bhandari 84 1

85 Vinnela 84 2

86 Gurung 86 1

87 Charles 86 2

88 Paschal 88 1

89 Fernando 84 6

90 Akhila 90 1

91 Staple 86 6

92 Buck 92 1

93 Joel 92 2

94 Constriction 94 1

95 Under 90 6

96 Dustin 96 1

97 Todd 96 2

98 Egbe 98 1

99 Matthew 98 2

100 Wine 86 15

101 Screen 94 8

102 Chris 102 1

103 Christopher 102 2

104 Person 104 1

105 Honor 104 2

106 Acevedo 106 1

107 Yolk 106 2

108 James 108 1

109 Perfect 104 6

110 Cook 110 1

111 Salkowski 110 2

112 Daniels 112 1

113 Davies 112 2

114 Afterwards 114 1

115 Cook 110 6

116 Veronica 116 1

117 Necromancer 116 2

118 Clark 118 1

119 Clayton 118 2

120 Octavio 118 3

121 Wade 118 4

122 Qamruddin 122 1

123 poison 104 20

124 Judy 124 1

125 Jutty 124 2

126 Kelly 126 1

127 Rush 124 4

**Theoretical vs Empirical results: In this table we can see expected number of probes is at 5.55. The number of probes for the first 30 keys are 1.13, as in the 50% full table, and 14.3 probes for the last 30 keys.**

**Random at 50%**

Average Number of Probes for first 30: 1.13333E+00

Min Number of Probes for First 30: 1

Max Number of Probes for First 30: 2

Average Number of Probes for Last 30: 1.93333E+00

Min Number of Probes for Last 30: 1

Max Number of Probes for Last 30: 4

Expected Result: 1.38629E+00

INDEX HASH KEY HASH ADDRESS NUMBER OF PROBES

0 -1 0

1 -1 0

2 -1 0

3 -1 0

4 -1 0

5 -1 0

6 Crouch 6 1

7 -1 0

8 -1 0

9 -1 0

10 Aguirrie 10 1

11 -1 0

12 Red 12 1

13 John 12 2

14 -1 0

15 -1 0

16 Lisa 16 1

17 -1 0

18 Alcantara 18 1

19 Michael 18 2

20 -1 0

21 Wade 118 4

22 -1 0

23 -1 0

24 Misogamist 18 3

25 -1 0

26 -1 0

27 -1 0

28 Zulfiqar 28 1

29 Jeffrey 28 2

30 Corey 30 1

31 Kyle 30 2

32 Dugger 32 1

33 -1 0

34 1234567890123456 34 1

35 Arauza 34 2

36 -1 0

37 -1 0

38 -1 0

39 -1 0

40 Eye 34 3

41 -1 0

42 Ellington 42 1

43 -1 0

44 Jordon 44 1

45 -1 0

46 Carmona 46 1

47 Scott 46 2

48 -1 0

49 -1 0

50 Adam 50 1

51 Maiden 50 2

52 Nienberg 52 1

53 Farral 52 2

54 Garza 54 1

55 -1 0

56 -1 0

57 -1 0

58 Fabulous 52 3

59 -1 0

60 Joseph 60 1

61 Robert 60 2

62 Casper 62 1

63 -1 0

64 -1 0

65 -1 0

66 Ajose 66 1

67 -1 0

68 -1 0

69 -1 0

70 -1 0

71 -1 0

72 -1 0

73 -1 0

74 -1 0

75 -1 0

76 Batbold 76 1

77 -1 0

78 Syncopate 78 1

79 -1 0

80 Derek 80 1

81 -1 0

82 -1 0

83 -1 0

84 Bhandari 84 1

85 Vinnela 84 2

86 Gurung 86 1

87 Charles 86 2

88 Paschal 88 1

89 -1 0

90 Akhila 90 1

91 -1 0

92 Buck 92 1

93 Joel 92 2

94 Constriction 94 1

95 -1 0

96 Dustin 96 1

97 Todd 96 2

98 Egbe 98 1

99 Matthew 98 2

100 -1 0

101 -1 0

102 Chris 102 1

103 Christopher 102 2

104 -1 0

105 -1 0

106 Acevedo 106 1

107 -1 0

108 James 108 1

109 -1 0

110 Cook 110 1

111 Salkowski 110 2

112 Daniels 112 1

113 Davies 112 2

114 -1 0

115 Fernando 84 4

116 Veronica 116 1

117 Necromancer 116 2

118 Clark 118 1

119 Clayton 118 2

120 -1 0

121 -1 0

122 Qamruddin 122 1

123 -1 0

124 Octavio 118 3

125 Judy 124 2

126 Kelly 126 1

127 -1 0

**Theoretical vs Empirical results: In this table we can see expected number of probes is at 1.38 The number of probes for the first 30 keys are 1.13 and 1.93 probes for the last 30 keys.**

**Random at 90%**

Average Number of Probes for first 30: 1.13333E+00

Min Number of Probes for First 30: 1

Max Number of Probes for First 30: 2

Average Number of Probes for Last 30: 7.03333E+00

Min Number of Probes for Last 30: 1

Max Number of Probes for Last 30: 26

Expected Result: 2.55843E+00

INDEX HASH KEY HASH ADDRESS NUMBER OF PROBES

0 live 0 1

1 constrain 30 16

2 Jutty 124 3

3 inlet 2 2

4 poison 104 5

5 nosegay 4 2

6 Crouch 6 1

7 Perfect 104 4

8 parasympathetic 8 1

9 ichthyosaur 106 4

10 Aguirrie 10 1

11 exhilarate 10 2

12 Red 12 1

13 John 12 2

14 scat 14 1

15 marsupial 2 6

16 Lisa 16 1

17 -1 0

18 Alcantara 18 1

19 Michael 18 2

20 Wine 86 11

21 Wade 118 4

22 wince 22 1

23 prologue 96 12

24 Misogamist 18 3

25 seal 30 24

26 quandary 26 1

27 Rush 124 4

28 Zulfiqar 28 1

29 Jeffrey 28 2

30 Corey 30 1

31 Kyle 30 2

32 Dugger 32 1

33 Staple 86 8

34 1234567890123456 34 1

35 Arauza 34 2

36 brutalize 36 1

37 guard 6 4

38 toluene 32 3

39 seat 30 26

40 Eye 34 3

41 -1 0

42 Ellington 42 1

43 cauldron 30 6

44 Jordon 44 1

45 ratiocination 44 2

46 Carmona 46 1

47 Scott 46 2

48 -1 0

49 mole 18 4

50 Adam 50 1

51 Maiden 50 2

52 Nienberg 52 1

53 Farral 52 2

54 Garza 54 1

55 -1 0

56 sparse 118 7

57 sponsor 86 16

58 Fabulous 52 3

59 wet 86 14

60 Joseph 60 1

61 Robert 60 2

62 Casper 62 1

63 -1 0

64 prepossess 64 1

65 Fabricate 52 6

66 Ajose 66 1

67 Irreversible 66 2

68 noggin 68 1

69 -1 0

70 -1 0

71 -1 0

72 -1 0

73 tiller 112 10

74 -1 0

75 rationalize 44 4

76 Batbold 76 1

77 orthodontist 46 4

78 Syncopate 78 1

79 cap 78 2

80 Derek 80 1

81 -1 0

82 magnetic 82 1

83 Northwest 52 4

84 Bhandari 84 1

85 Vinnela 84 2

86 Gurung 86 1

87 Charles 86 2

88 Paschal 88 1

89 privilege 0 10

90 Akhila 90 1

91 Under 90 2

92 Buck 92 1

93 Joel 92 2

94 Constriction 94 1

95 Screen 94 2

96 Dustin 96 1

97 Todd 96 2

98 Egbe 98 1

99 Matthew 98 2

100 forgetful 100 1

101 -1 0

102 Chris 102 1

103 Christopher 102 2

104 Person 104 1

105 Honor 104 2

106 Acevedo 106 1

107 Yolk 106 2

108 James 108 1

109 option 30 20

110 Cook 110 1

111 Salkowski 110 2

112 Daniels 112 1

113 Davies 112 2

114 Afterwards 114 1

115 Fernando 84 4

116 Veronica 116 1

117 Necromancer 116 2

118 Clark 118 1

119 Clayton 118 2

120 dactyl 0 9

121 -1 0

122 Qamruddin 122 1

123 Cook 110 6

124 Octavio 118 3

125 Judy 124 2

126 Kelly 126 1

127 Finagle 52 8

**Theoretical vs Empirical results: In this table we can see expected number of probes is at 2.55. The number of probes for the first 30 keys are 1.13, as in the 50% full table, and 7.03 probes for the last 30 keys.**

**Criticism: If we look at the hash function HA = abs( ( ( Key[4..5] - Key[1..2] ) / 256 + Key[3] ) / 256 + Key[1] ) modulo 128 we can observe some weaknesses of the algorithm that lead to the big differences in linear and random probes used. After choosing any random number numbers for the key, anything <= 127, and working out the arithmetic you start to notice a pattern. No matter the key chosen, whether it be 1 or worst case 127, the result would always be whatever Key [1] is chosen to be. Reason being, the dividing by 256 virtually zeroed out all arithmetic prior to it. So after the last “/256” you add Key [1], which adds Key[1] to a very small number(<1), and then modulo 128. Basically after the modulo 128 all you get back is Key[1]. After noticing this flaw, we see why this hash function will fail and lead to many collisions. It very much limits the possible outcomes of the functions, which intern, leads to primary clustering and high volume of collisions.**

**My Hash Funciton: lets take a look at the hash function that I was able to produce. HA = ((Key[1]\*2) + (Key[2]\*8) + (Key[3]\*16) modulo 128. The method that I decided to use to avoid collisions in my hash table was using weighted code. This technique is useful for dealing with simple code collisions. As seen above I multiply each key with a different “weight” so that even if the same keys are re-arranged the differing “weights” should help deal with collisions much more effectively than that lousy original hash function. Also, I use the modulo 128 to make sure to keep all results in range(0-127). Prime examples are, when you compare the tables of the old and new hash functions for linear, at either 50% or 90%, you can see the difference in efficiency, especially at the last 30 keys. The hash function I created efficiently deals with multiple keys making mapping to the same location than the original hash whether its linear or random probes.**

**Hash.ads**

**with** **Ada.Numerics.Elementary\_Functions**;

**with** **Ada.Text\_IO**;

**package** **Hash** **is**

**type** **HashRecord** **is** **record**

key : **String**(**1.**.**16**);

initialHashAddress : **Integer**;

numProbes : **Integer**;

**end record**;

**type** **RecordArray** **is** **array** (**Integer** **range** <>) **of** HashRecord;

**type** **OffsetArray** **is** **array** (**Integer** **range** <>) **of** **Integer**;

**function** **calculateHash**(hashKey : in String) **return** **Integer**;

**function** **chrisHash**(hashKey : in String) **return** **Integer**;

**function** **linearProbe**(hashTable : in RecordArray;hashAddress : in Integer;toInsert : in out HashRecord) **return** **Integer**;

**procedure** **generateRandomInteger**(offsetTable : in out OffsetArray);

**function** **randomProbe**(hashTable : in RecordArray;offsetTable : in OffsetArray;hashAddress : in Integer;toInsert : in out HashRecord) **return** **Integer**;

**procedure** **initializeRecordArray**(hashTable : in out RecordArray);

**function** **findKeyLinear**(hashTable : in RecordArray;hashAddress : in Integer ;key : in String) **return** **Integer**;

**function** **findKeyRandom**(hashTable : in RecordArray; offsetTable : in OffsetArray; hashAddress : in INteger; key : in String) **return** **Integer**;

**procedure** **printRecord**(re : in HashRecord);

**end** **Hash**;

**Hash.adb**

**package** **body** **Hash** **is**

-- HA = abs( ( ( Key[4..5] - Key[1..2] ) / 256 + Key[3] ) / 256 + Key[1] ) modulo 128.

**function** **calculateHash**(hashKey : in String) **return** **Integer** **is**

key4 : **Integer** := **Character**'Pos(hashKey(hashKey'First+**3**));

key5 : **Integer** := **Character**'Pos(hashKey(hashKey'First+**4**));

key1 : **Integer** := **Character**'Pos(hashKey(hashKey'First));

key2 : **Integer** := **Character**'Pos(hashKey(hashKey'First+**1**));

key3 : **Integer** := **Character**'Pos(hashKey(hashKey'First+**2**));

key45 : **Integer** := key4 + key5;

key12 : **Integer** := key1 + key2;

diff : **Integer** := key45 - key12;

div : **Long\_Float** := **Long\_Float**(diff) / **Long\_Float**(**256**);

sum3 : **Long\_Float** := div + **Long\_Float**(key3);

div2 : **Long\_Float** := sum3 / **Long\_Float**(**256**);

sum1 : **Long\_Float** := div2 + **Long\_Float**(key1);

modR : **Integer** := **Integer**(sum1) **mod** **128**;

**begin**

**return** modR;

**end** **calculateHash**;

--HA = ((Key[1]\*2) + (Key[2]\*8) + (Key[3]\*16) modulo 128

**function** **chrisHash**(hashKey : in String) **return** **Integer** **is**

key1 : **Integer** := **Character**'Pos(hashKey(hashKey'First));

key2 : **Integer** := **Character**'Pos(hashKey(hashKey'First+**1**));

key3 : **Integer** := **Character**'Pos(hashKey(hashKey'First+**2**));

calcKey1: **Integer** := key1 \* **2**;

calcKey2: **Integer** := key2 \* **8**;

calcKey3: **Integer** := key3 \* **16**;

totelKey: **Integer** := calcKey1 + calcKey2 + calcKey3;

modK : **Integer** := totelKey **mod** **128**;

**begin**

**return** modK;

**end** **chrisHash**;

**function** **linearProbe**(hashTable : in RecordArray;hashAddress : in Integer;toInsert : in out HashRecord) **return** **Integer** **is**

tempIndex : **Integer** := (hashAddress + **1**) **mod** **128**;

**begin**

**if** hashTable(hashAddress).numProbes = **0** **then**

**return** hashAddress;

**else**

toInsert.numProbes := toInsert.numProbes + **1**;

**while** tempIndex /= hashAddress **loop**

**if** hashTable(tempIndex).numProbes = **0** **then**

**return** tempIndex;

**else**

tempIndex := (tempIndex + **1**) **mod** **128**;

toInsert.numProbes := toInsert.numProbes + **1**;

**end** **if**;

**end** **loop**;

**return** -**1**;

**end** **if**;

**end** **linearProbe**;

**procedure** **generateRandomInteger**(offsetTable : in out OffsetArray) **is**

r : **Integer** := **1**;

n : **Integer** := **Integer**(Ada.Numerics.Elementary\_Functions.Log(X => **Float**(offsetTable'Length),

Base => **2.0**));

**begin**

**for** i **in** offsetTable'Range **loop**

r := r \* **5**;

r := r **mod** **2**\*\*(n+**2**);

offsetTable(i) := **Integer**(r / **4**);

**end** **loop**;

**end** **generateRandomInteger**;

**function** **randomProbe**(hashTable : in RecordArray;offsetTable : in OffsetArray;hashAddress : in Integer;toInsert : in out HashRecord) **return** **Integer** **is**

tempIndex : **Integer**;

**begin**

**if** hashTable(hashAddress).numProbes = **0** **then**

**return** hashAddress;

**else**

toInsert.numProbes := toInsert.numProbes + **1**;

**for** i **in** offsetTable'Range **loop**

tempIndex := (hashAddress +offsetTable(i)) **mod** **128**;

**if** hashTable(tempIndex).numProbes = **0** **then**

**return** tempIndex;

**else**

toInsert.numProbes := toInsert.numProbes + **1**;

**end** **if**;

**end** **loop**;

**return** -**1**;

**end** **if**;

**end** **randomProbe**;

**procedure** **initializeRecordArray**(hashTable : in out RecordArray) **is**

tempRecord : HashRecord;

**begin**

**for** i **in** hashTable'Range **loop**

tempRecord.key := " ";

tempRecord.initialHashAddress := -**1**;

tempRecord.numProbes := **0**;

hashTable(i) := tempRecord;

**end** **loop**;

**end** **initializeRecordArray**;

**function** **findKeyLinear**(hashTable : in RecordArray; hashAddress : in Integer;key : in String) **return** **Integer** **is**

tempIndex : **Integer** := (hashAddress + **1**) **mod** **128**;

**begin**

**if** hashTable(hashAddress).key = key **then**

**return** hashAddress;

**else**

**while** tempIndex /= hashAddress **loop**

**if** hashTable(tempIndex).key = key **then**

**return** tempIndex;

**else**

tempIndex := (tempIndex + **1**) **mod** **128**;

**end** **if**;

**end** **loop**;

**return** -**1**;

**end** **if**;

**end** **findKeyLinear**;

**function** **findKeyRandom**(hashTable : in RecordArray;offsetTable : in OffsetArray;hashAddress : in Integer;key : in String) **return** **Integer** **is**

tempIndex : **Integer**;

**begin**

**if** hashTable(hashAddress).key = key **then**

**return** hashAddress;

**else**

**for** i **in** offsetTable'Range **loop**

tempIndex := (hashAddress +offsetTable(i)) **mod** **128**;

**if** hashTable(tempIndex).key = key **then**

**return** tempIndex;

**end** **if**;

**end** **loop**;

**return** -**1**;

**end** **if**;

**end** **findKeyRandom**;

**procedure** **printRecord**(re : in HashRecord) **is**

**begin**

Ada.Text\_IO.Put\_Line(" " & re.key & " " & **Integer**'Image(re.initialHashAddress) & " " & **Integer**'Image(re.numProbes));

**end** **printRecord**;

**end** **Hash**;

**main.adb**

**with** **Ada.Text\_IO**;

**use** **Ada.Text\_IO**;

**with** **Ada.Exceptions**;

**use** **Ada.Exceptions**;

**with** **Hash**;

**use** **Hash**;

**with** **Ada.Numerics.Elementary\_Functions**;

**procedure** **Main** **is**

**type** **StringArray** **is** **array** (**Integer** **range** <>) **of** **String**(**1.**.**16**);

tableSize : **Integer** := **128**;

hashTable : Hash.RecordArray(**0.**.tableSize-**1**);

invalidArgumentException : **Exception**;

outOfMemoryException : **Exception**;

offsetTable : Hash.OffsetArray(**1.**.tableSize);

first30Keys : StringArray(**1.**.**30**);

last30Keys : StringArray(**1.**.**30**);

**begin**

initializeRecordArray(hashTable => hashTable);

Ada.Text\_IO.Put\_Line("Press 0 for Linear or 1 for Random: ");

**declare**

option : **Integer** := **Integer**'Value(Get\_Line);

**begin**

**if** option < **0** **or** option > **1** **then**

**raise** invalidArgumentException **with** "Input should be 0 for Linear or 1 for Random";

**end** **if**;

**if** option = **1** **then**

generateRandomInteger(offsetTable => offsetTable);

**end** **if**;

Ada.Text\_IO.Put\_Line("Press 0 for Default Hash or 1 for New Hash: ");

**declare**

hashSelection : **Integer** := **Integer**'Value(Get\_Line);

**begin**

**if** hashSelection < **0** **or** hashSelection > **1** **then**

**raise** invalidArgumentException **with** "Input should be 0 for Default Hash or 1 for New Hash";

**end** **if**;

Ada.Text\_IO.Put\_Line("Please enter what percent of the table should be full (decimal): ");

**declare**

percentFull : **Float** := **Float**'Value(Get\_Line);

totalFull : **Integer**;

**begin**

**if** percentFull < **Float**(**0**) **or** percentFull > **Float**(**1**) **then**

**raise** invalidArgumentException **with** "Input should be between 0 and 1";

**end** **if**;

totalFull := **Integer**(percentFull \* **Float**(tableSize));

Ada.Text\_IO.Put\_Line("Please enter input file: ");

**declare**

inputFileName : **String** := Get\_Line;

inputFile : **File\_Type**;

counter : **Integer** := **0**;

extraCounter : **Integer** :=**0**;

**begin**

Open(File => inputFile,

Mode => Ada.Text\_IO.In\_File,

Name => inputFileName);

**while** **not** End\_Of\_File(inputFile) **loop**

**declare**

tempString : **String** := Get\_Line(inputFile);

hashAddress : **Integer**;

hashKey : **String**(**1.**.**16**);

offset : **Integer**;

insertionIndex : **Integer**;

toInsert : HashRecord;

**begin**

**if** tempString'Length /= **16** **then**

offset := tempString'Length - **16**;

**for** i **in** tempString'Range **loop**

hashKey(i) := tempString(i);

**end** **loop**;

**for** i **in** **Integer** **range** tempString'Length+**1.**.**16** **loop**

hashKey(i) := ' ';

**end** **loop**;

**else**

hashKey := tempString;

**end** **if**;

**if** counter < **30** **then**

first30Keys(counter+**1**) := hashKey;

**end** **if**;

**if** counter > (totalFull - **31**) **then**

last30Keys(extraCounter+**1**) := hashKey;

extraCounter := extraCounter + **1**;

**end** **if**;

**if** hashSelection = **0** **then**

hashAddress := calculateHash(hashKey => hashKey);

**else**

hashAddress := chrisHash(hashKey => hashKey);

**end** **if**;

toInsert.key := hashKey;

toInsert.initialHashAddress := hashAddress;

toInsert.numProbes := **1**;

**if** option = **0** **then**

insertionIndex := linearProbe(hashTable => hashTable,

hashAddress => hashAddress,

toInsert => toInsert);

**else**

insertionIndex := randomProbe(hashTable => hashTable,

offsetTable => offsetTable,

hashAddress => hashAddress,

toInsert => toInsert);

**end** **if**;

**if** insertionIndex < **0** **then**

**raise** outOfMemoryException **with** "Out Of Memory While Inserting in Hash Table";

**else**

hashTable(insertionIndex) := toInsert;

**end** **if**;

**if** counter + **1** = totalFull **then**

**exit**;

**end** **if**;

counter := counter + **1**;

**end**;

**end** **loop**;

Close(inputFile);

**declare**

avg : **Float** := **0.0**;

min : **Integer** := **1000**;

max : **Integer** := -**1**;

hashAddress : **Integer**;

index : **Integer**;

loadFactor : **Float**;

expected : **Float**;

**begin**

**for** i **in** first30Keys'Range **loop**

**if** hashSelection = **0** **then**

hashAddress := calculateHash(hashKey => first30Keys(i));

**else**

hashAddress := chrisHash(hashKey => first30Keys(i));

**end** **if**;

**if** option = **0** **then**

index := findKeyLinear(hashTable => hashTable,

hashAddress => hashAddress,

key => first30Keys(i));

**else**

index := findKeyRandom(hashTable => hashTable,

offsetTable => offsetTable,

hashAddress => hashAddress,

key => first30Keys(i));

**end** **if**;

**if** hashTable(index).numProbes > max **then**

max := hashTable(index).numProbes;

**end** **if**;

**if** hashTable(index).numProbes < min **then**

min := hashTable(index).numProbes;

**end** **if**;

avg := avg + **Float**(hashTable(index).numProbes);

**end** **loop**;

avg := avg / **Float**(**30**);

Ada.Text\_IO.Put\_Line("Average Number of Probes for first 30: " & **Float**'Image(avg));

Ada.Text\_IO.Put\_Line("Min Number of Probes for First 30: " & **Integer**'Image(min));

Ada.Text\_IO.Put\_Line("Max Number of Probes for First 30: " & **Integer**'Image(max));

avg := **0.0**;

max := -**1**;

min := **1000**;

**for** i **in** last30Keys'Range **loop**

**if** hashSelection = **0** **then**

hashAddress := calculateHash(hashKey => last30Keys(i));

**else**

hashAddress := chrisHash(hashKey => last30Keys(i));

**end** **if**;

**if** option = **0** **then**

index := findKeyLinear(hashTable => hashTable,

hashAddress => hashAddress,

key => last30Keys(i));

**else**

index := findKeyRandom(hashTable => hashTable,

offsetTable => offsetTable,

hashAddress => hashAddress,

key => last30Keys(i));

**end** **if**;

**if** hashTable(index).numProbes > max **then**

max := hashTable(index).numProbes;

**end** **if**;

**if** hashTable(index).numProbes < min **then**

min := hashTable(index).numProbes;

**end** **if**;

avg := avg + **Float**(hashTable(index).numProbes);

**end** **loop**;

avg := avg / **Float**(**30**);

Ada.Text\_IO.Put\_Line("Average Number of Probes for Last 30: " & **Float**'Image(avg));

Ada.Text\_IO.Put\_Line("Min Number of Probes for Last 30: " & **Integer**'Image(min));

Ada.Text\_IO.Put\_Line("Max Number of Probes for Last 30: " & **Integer**'Image(max));

loadFactor := (**Float**(tableSize) \* percentFull)/**Float**(tableSize);

**if** option = **0** **then**

expected := (**Float**(**1**) - loadFactor/**Float**(**2**))/(**Float**(**1**)-loadFactor);

**else**

expected := -(**Float**(**1**)/loadFactor)\***Float**(Ada.Numerics.Elementary\_Functions.Log(**Float**(**1**)-loadFactor));

**end** **if**;

Ada.Text\_IO.Put\_Line("Exptected Result: " & **Float**'Image(Expected));

Ada.Text\_IO.Put\_Line("INDEX HASH KEY HASH ADDRESS NUMBER OF PROBES");

**for** i **in** hashTable'Range **loop**

Ada.Text\_IO.Put(**Integer**'Image(i));

printRecord(hashTable(i));

**end** **loop**;

**end**;

**end**;

**end**;

**end**;

**end**;

**end** **Main**;