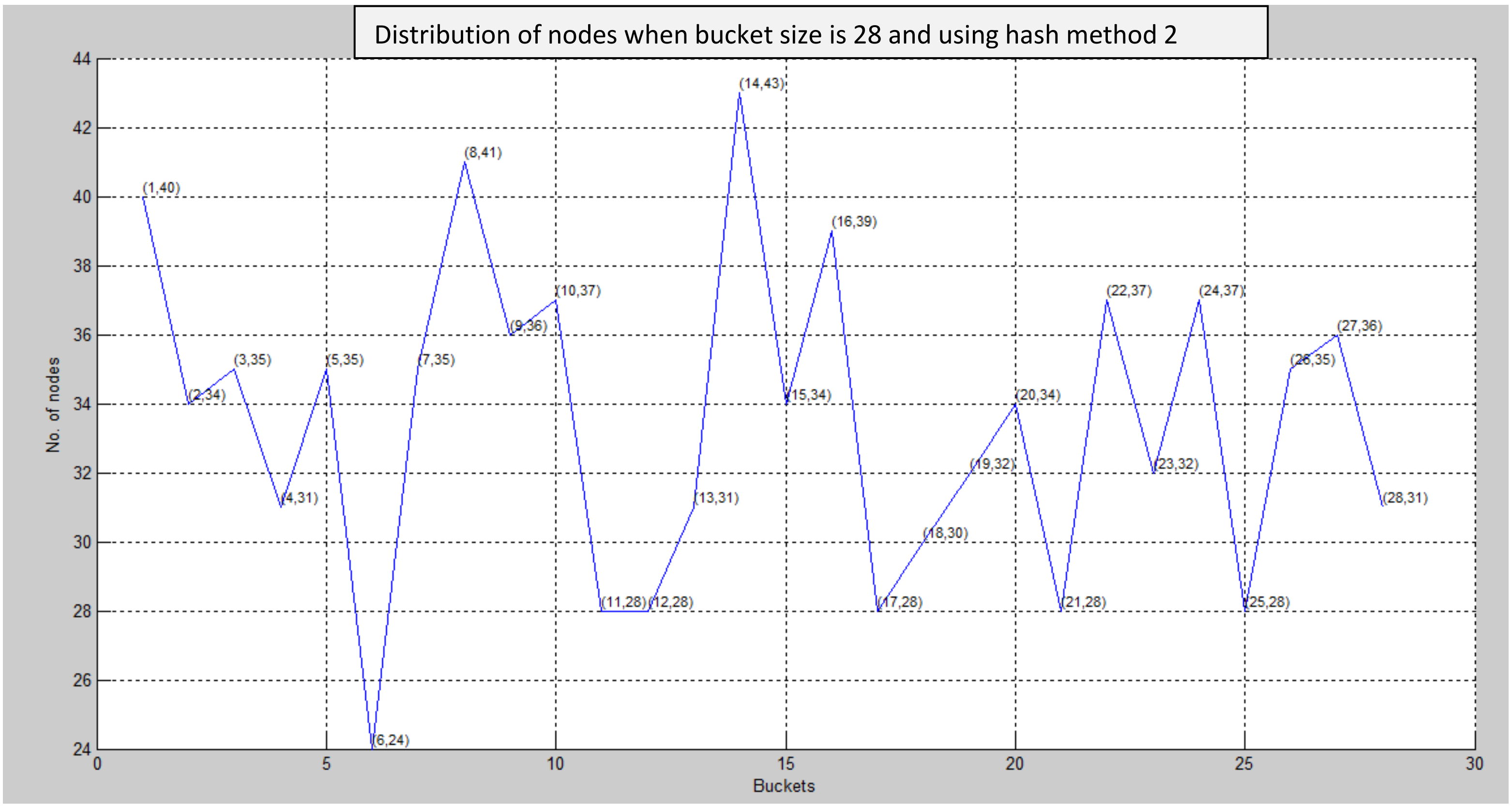
Report on Hash Table

First consider sample-text1.txt

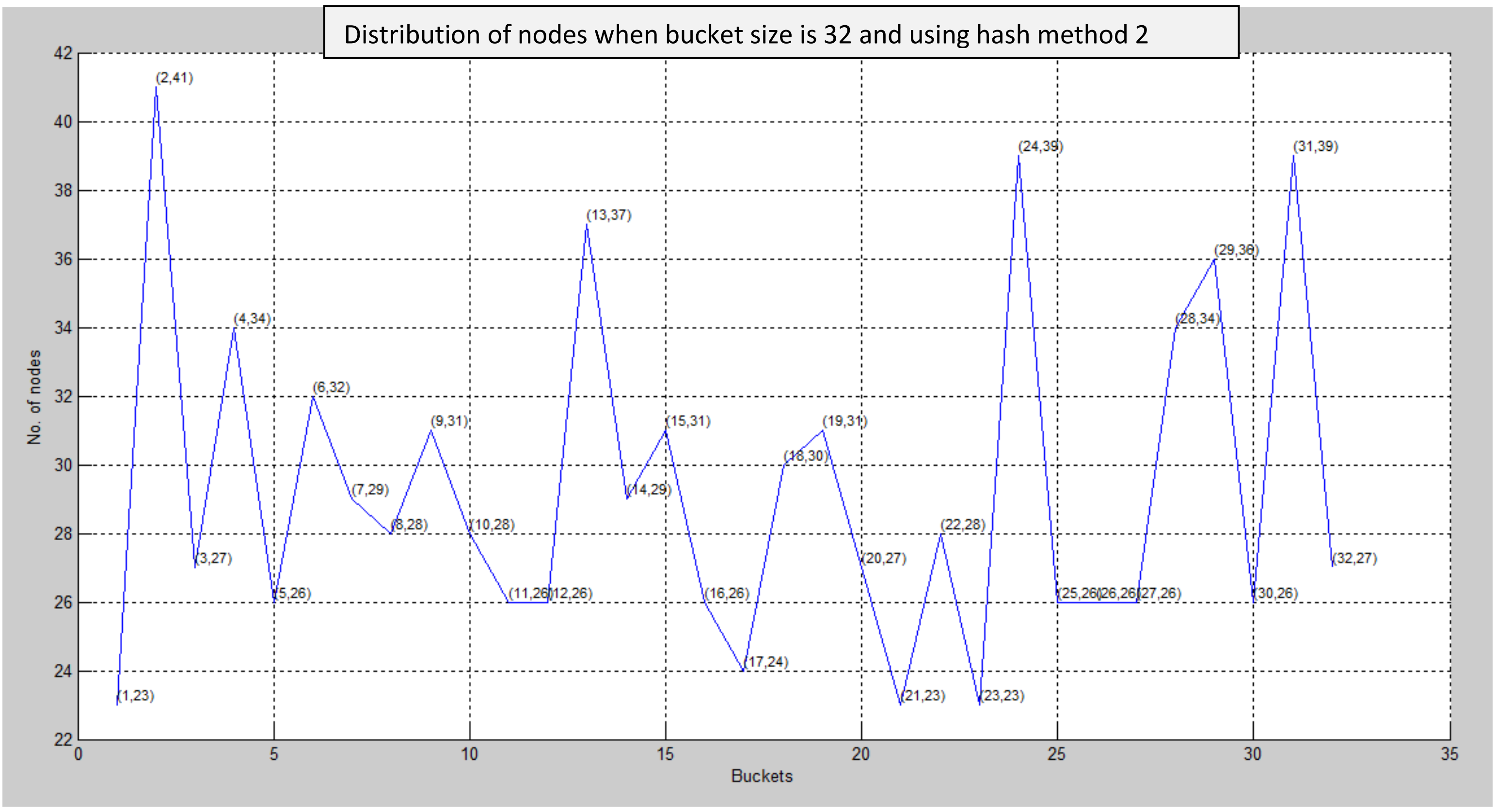
1) **Hash method 2**

```
public int hashing(String key){
        int hash = 0;
        for (int i = 0; i < key.length(); i++){
                hash = (31* hash + key.charAt(i))%table.length;
        }
        return (hash%table.length);
}
```

Normally java hash function for String combines successive characters by multiplying the current hash by 31 and then adding the new character.



Maximum: 43
Minimum: 24
Avg: 33.535713
Deviation: 4.3995405477447305

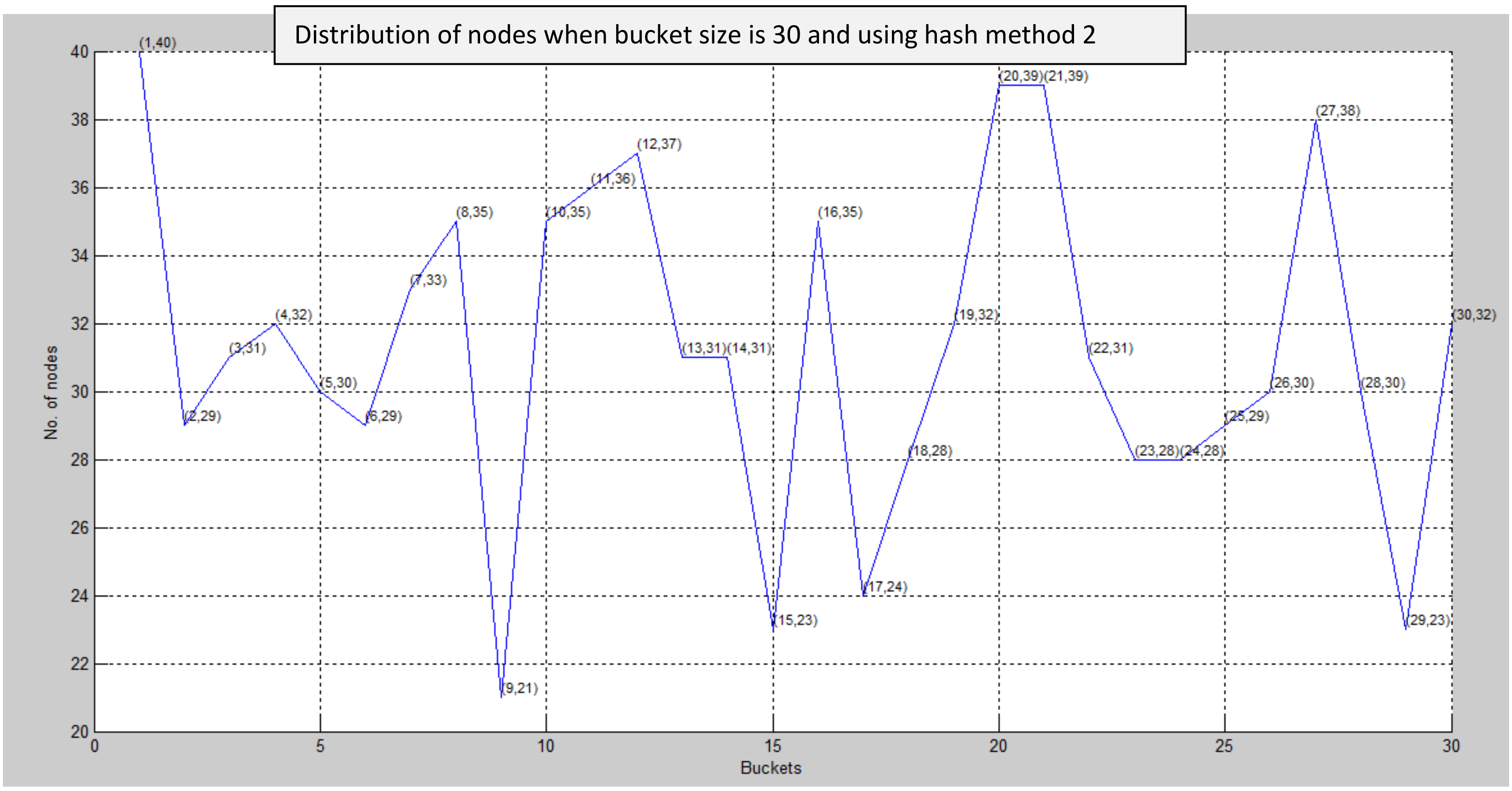


Maximum: 41

Minimum: 23

Avg: 29.34375

Deviation: 4.790024628068211



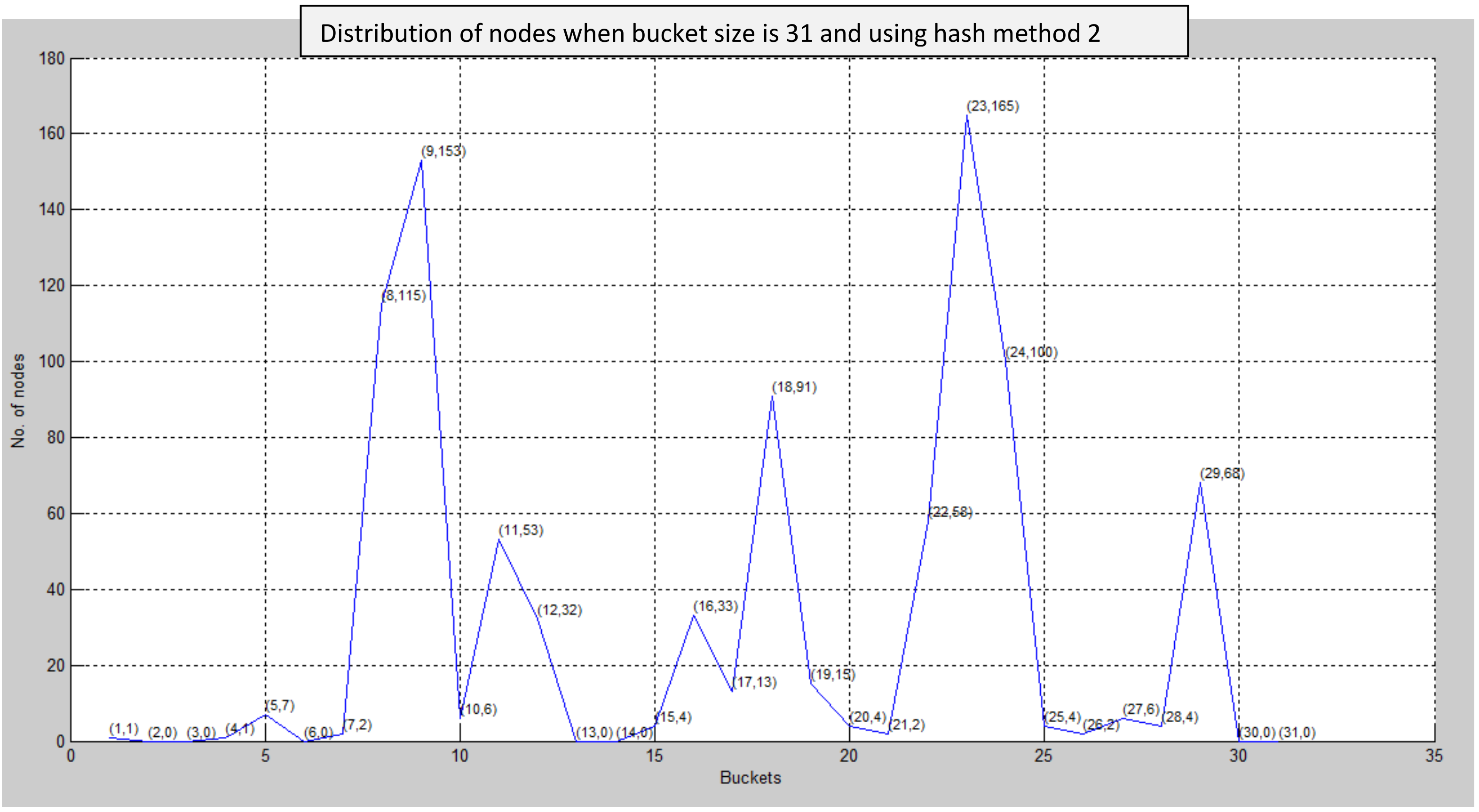
Maximum: 40

Minimum: 21

Avg: 31.3

Deviation: 4.723352474011441

For different buckets it shows different distributions. They all have a distribution which is very much close to uniform distribution. But we should not use the bucket size as 31. It will be very much far from being a uniform distribution.



Maximum: 165

Minimum: 0

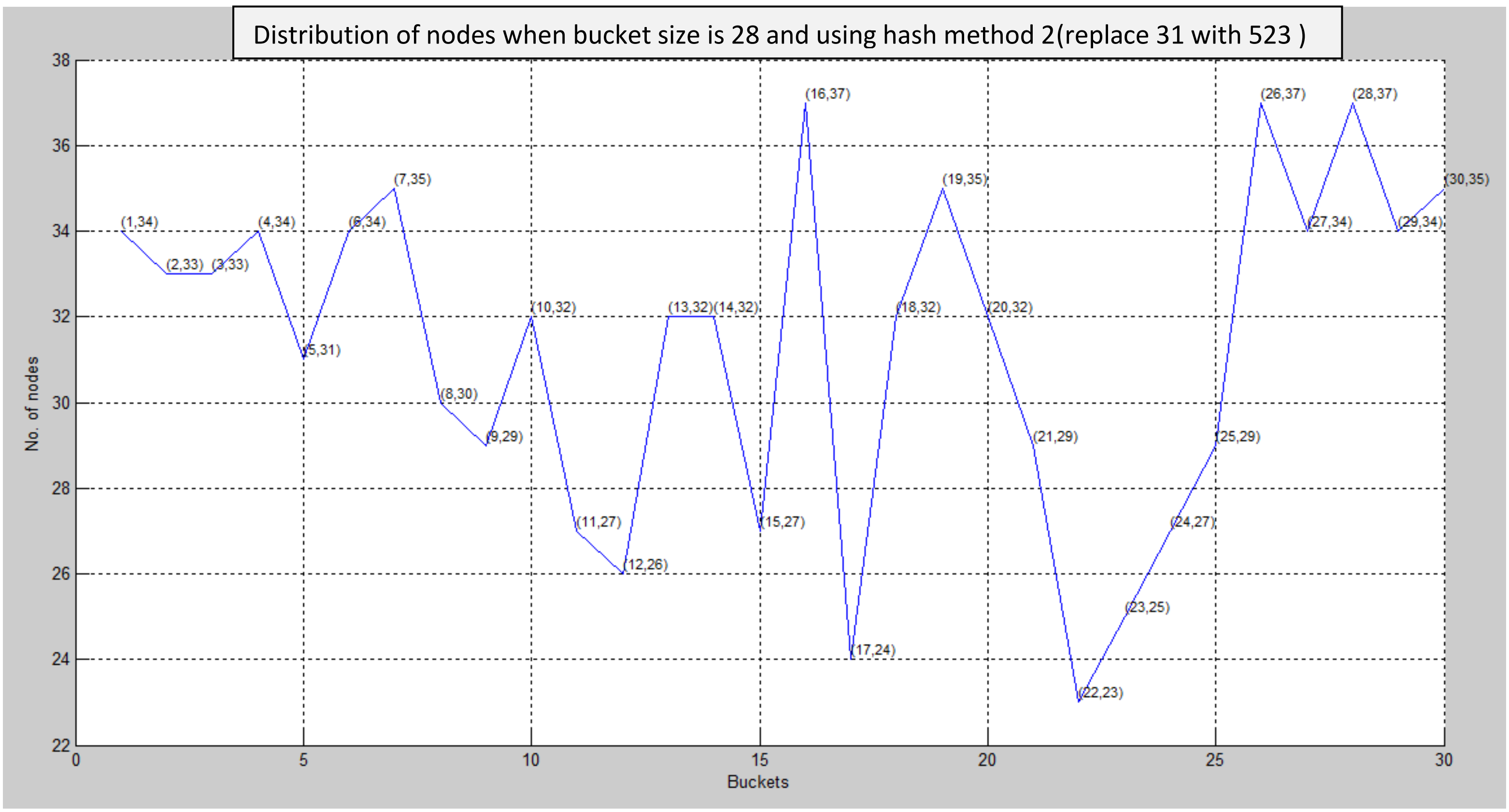
Avg:30.290323

Deviation:46.556378058120295

So using 31 buckets for this hash method is not suitable.

If we use 523 instead of 31 we can make the buckets distributed even more uniformly.

```
public int hashing(String key){
        int hash = 0;
        for (int i = 0; i < key.length(); i++){
                hash = (523*hash + key.charAt(i))%table.length;
        }
        return (hash%table.length);
}
```



Maximum: 37

Minimum: 23

Avg: 31.3

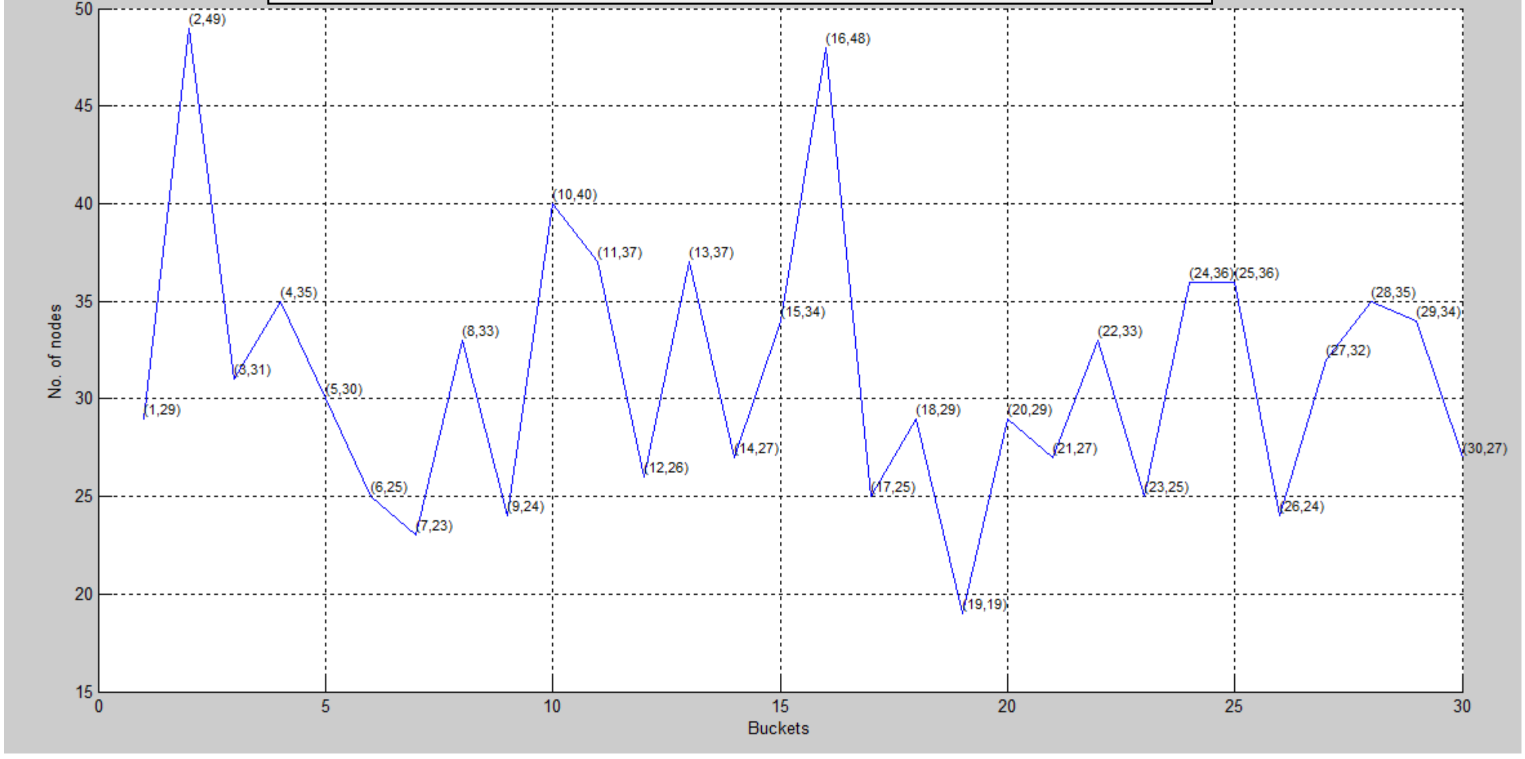
Deviation: 3.7828638085120114

2) Hash method 1

Then let's consider what will happen if we use 32 (even number) for the hashing

```
public int hashing(String key){
        int hash = 0;
        for (int i = 0; i < key.length(); i++){
                hash = (32* hash + key.charAt(i))%table.length;
        }
        return (hash%table.length);
}
```

Distribution of nodes when bucket size is 30 and using hash method 1



Maximum: 49

Minimum: 19

Avg: 31.3

Deviation: 6.731274663371716

This graph shows pretty much high fluctuation when compared with graph of hashing method 2 with 30 buckets. Deviation is also high. This graph is very much far from being an uniform distribution.

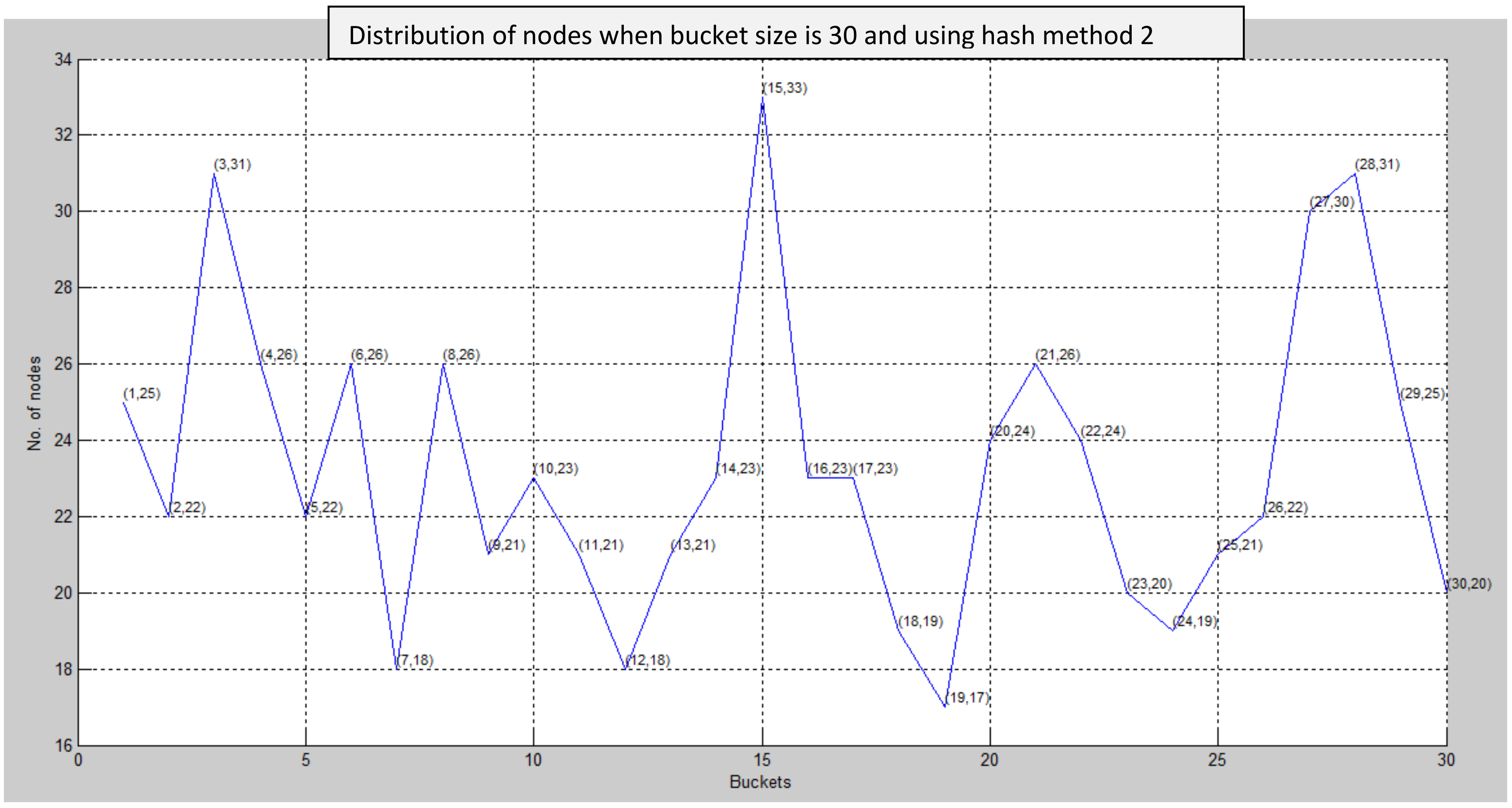
So in this situation using hash method 2 is the best when compared to hash method 1.

Then consider sample-text2.txt

1) **Hash method 2**

   Let's compare using the results obtained from above.

```
public int hashing(String key){
        int hash = 0;
        for (int i = 0; i < key.length(); i++){
                hash = (31* hash + key.charAt(i))%table.length;
        }
        return (hash%table.length);
}
```



Maximum: 33

Minimum: 17

Avg: 23.333334

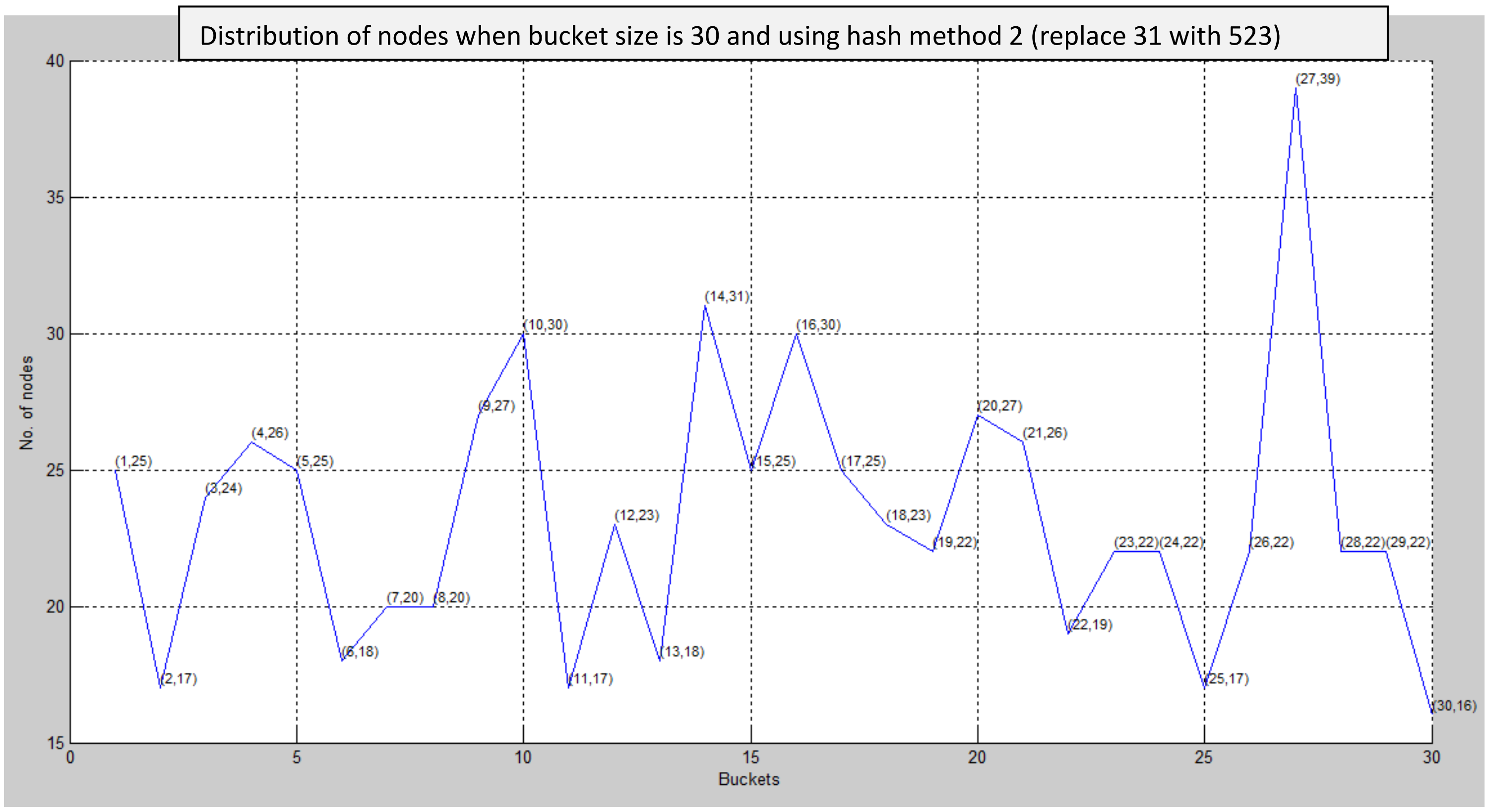
Deviation: 3.9440514692619693

This graph is also pretty much close to a uniform distribution. It has a low deviation also.

If we change 31 to 523

```
public int hashing(String key){
        int hash = 0;
        for (int i = 0; i < key.length(); i++){
                hash = (523* hash + key.charAt(i))%table.length;
        }
        return (hash%table.length);
}
```

We can make it more uniform distribution

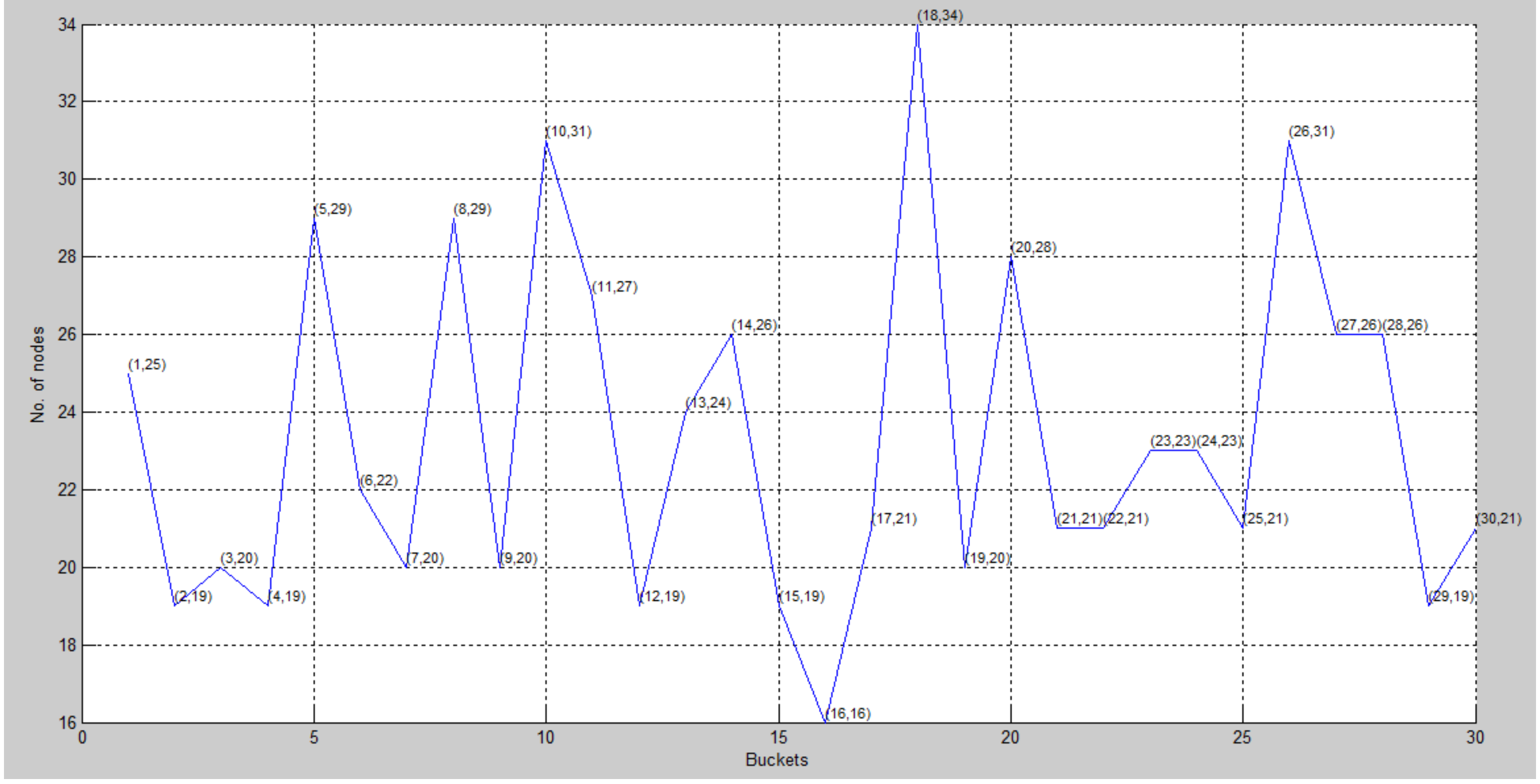


Maximum: 39
Minimum: 16
Avg: 23.333334
Deviation: 4.853405195549564

Here the average is same but the deviation has slightly increased. But since it is close to uniform distribution than the above distribution this will be better.

2) Hash method 1

Distribution of nodes when bucket size is 30 and using hash method 1



Maximum: 34

Minimum: 16

Avg: 23.333334

Deviation: 4.307614420092344

It will give different results for different text files even when we are using the same hash function. It is because different text files different distribution of words. So for the above 2 sample text files they show different distribution words in buckets.

And also distribution is different for different buckets.

Furthermore if we use an odd multiplier (31) it will give a better distribution rather than using an even multiplier (32).

And from the odd multipliers, we can increase the uniform distribution by increasing the odd value (523).