Table 1 is the analysis of each function in terms of parameters, returns, values of input, exceptions, identifying their characteristics and determining if they are covered by other characteristics

Method	**Params**	**returns**	**Values**	**Exception**	**Ch id**	**characteristics**	**Covered by**
adjacent Hash	String Hash **Direction direction**	String	Null,string bottom,top,right ,left		C1 C2 C3	whether hash is null Length of hash Direction	
				IllegalArgumentException			C1 , c2
				nullPointerexception			c3
right	String hash	string	null,string				c1,c2
				IllegalArgumentException			c1,c2
left	String hash	string	null,string				c1,c2
				IllegalArgumentException			c1,c2
top	String hash	string	null,string				c1,c2
				IllegalArgumentException			c1,c2
bottom	String hash	string	null,string				c1,c2

				IllegalArgumentException			c1,c2
adjacent Hash	String Hash, **Direction direction, int steps**	string	Null,string bottom,top,right,left 0<n or n<=0		C4	Value of steps	c1,c2,c3
				IllegalArgumentException			C1 c2
				nullPointerexception			c3
neighbours	String hash	list<String>	**Null,string**				C1 c2
				IllegalArgumentException			c1,c2
encodeHash	Double Latitude Double longitude	string	**-90 to 90 for latitude** **-180 to 180 for longitude**.		C5 **C6**	**Latitude between -90 or 90** **Longitude Between -180 or 180**	
				Illegalargumentexception			C5
encodeHash	LatLongp, int length	string	**-90 to 90 for latitude**				C5

			-180 to 180 for longitude. **length>0**		**C7**	**Length between 1 and 12**	C6
				Illegalargumentexception			C7 c5
encodeHash	LatLong p	string	**-90 to 90 for latitude of the point** **-180 to 180 for longitude of the point**				C5 C6
				Illegalargumentexception			C7,c5
encodeHash	double latitude, double longitude, int length	string	**-90 to 90 for latitude** **-180 to 180 for longitude**. **length>=0**				C5 C6 C7
				IllegalArgumentException.			C5 C7
decodeHash	String geohash	latlong	**String , null** **-90 to 90 for latitude for point** **-180 to 180 for**				C1,c5,c6,c2

			longitude.**for point**				
hashLengthToCoverBoundingBox	double topLeftLat, double topLeftLon, double bottomRightLat, double bottomRightLon	int	**-90 to 90 for latitude** **-180 to 180 for longitude**. Length is >=0				C5 C6 C7
hashContains	String hash, double lat, double lon	boolean	**String , null** **-90 to 90 for latitude** **-180 to 180 for longitude**. true,false		C8	**Whether the hash contains the given lat and long**	C1 C5 C6 c2
				NullpointerException			c1
coverBoundingBox	double topLeftLat, double topLeftLon, double bottomRightLat, double bottomRightLon	coverage	**-90 to 90 for latitude** **-180 to 180 for longitude**.		C9	Returns more than max	C5 C6

coverBoundingBoxMaxHashes	double topLeftLat, double topLeftLon, double bottomRightLat, double bottomRightLon, int maxHashes	coverage	**-90 to 90 for latitude** **-180 to 180 for longitude**. **maxhashes>0**		**C10**	**Value of max hashes**	C5 C6 C9
coverBoundingBox	double topLeftLat, double topLeftLon, double bottomRightLat, double bottomRightLon, int length	coverage	-90 to 90 for latitude -180 to 180 for longitude. Length >=0				C5 C6 C7 C9
heightDegrees	Int n	double	n>=0 Geohash height >=0				C7
widthDegrees	int n	double	n>=0 Geohash width>=0				C7
gridAsString	String hash, int size, Set<Strin	string	**String , null** Size >=1		**C11**	**size of square grid in**	C1

	g> highlight These		Set of elements can be nulls			**hashes**	C2
gridAsString	String hash, int fromRight, int fromBottom, int toRight, int toBottom	string	**String , null** **0<fromRight<=0** **0<fromBottom<=0** **0<toRight<=0** **0<toBottom<=0**		**C12** **C13** **C14** **C15**	top left of the grid in hashes to the right top left of the grid in hashes to the bottom bottom right of the grid in hashes to the bottom bottom right of the grid in hashes to the bottom	C1 C2
gridAsString	String hash, int fromRight, int fromBottom, int toRight, int toBottom, Set<String> highlight These	string	**String , null** **0<fromRight<=0** **0<fromBottom<=0** **0<toRight<=0** **0<toBottom<=0**				C1 c12 C13 C14 C15 C2

Table 2 shows how will the different characteristics be partitioned based on their input and behaviour

id	**characteristic**		**partition**
C1	Whether string is null	a1 a2	True false **base**
C2	Length of hash	b1 b2	>0 **base** = 0
C3	Direction	c1 C2	Valid direction base null
C4	Value of steps	D1 D2	n<=0 "negative or positive" n>0 base
C5	Latitude between -90 or 90 **base**	F1 F2	true **base** **False**
C6	Longitude between -180 or 180 **base**	G1 g2	true **base** **False**
C7	Length between 1 and 12 **base**	H1 H2	true **base** **False**
C8	**Whether the hash contains the given lat and long**	i1 i2	True **base** false
C9	Returns more than max	J1 J2	True **base** false
C10	**Value of max hashes**	K1 K2	max<=1 max>1 **base**

C11	**size of square grid in hashes**	N1
		N2
		N3
		N4
			Size =0
			Size =1
			size>1 base
			Size =-1

C12	top left of the grid in hashes to the right	p1 p2	**fromRight**<=0 "negative or positive" **fromRight**>0 base
C13	top left of the grid in hashes to the bottom	q1 q2	**fromBottom**<=0 "negative or positive" **fromBottom**>0 base
C14	bottom right of the grid in hashes to the bottom	r1 r2	**toRight**<=0 "negative or positive" **toRight**>0 base
C15	bottom right of the	s1	**toBottom**<=0 "negative or positive"

	grid in hashes to the bottom	s2	**toBottom**>0 base

Coverage criterion is Base choice to highlight the important values that should be tested and determining a happy path
One test case that includes all the best choices combined
(a2,b1,c1,d2,f1,g1,h1,i1,j1,k2,l1,m1,n3,p2,q2,r2,s2)

Method	**characteristics**	**Test Requirements**	**Infeasible, RevisedTRs**
adjacentHash	C1 c2 c3	{**a2b1c1,**a1b1c1,a2b2c1,a2b1c2,}	a1b1c1->a1b2c1 A2B2C1->A2B1C1
right	C1,c2	{**a2b1,**a1b1,a2b2}	a1b1->a1b2 a2b2-> a2b1
left	C1,c2	**{a2b1,**a1b1,a2b2}	a1b1->a1b2 A2B2->A2B1
top	C1,c2	**{a2b1,**a1b1,a2b2}	a1b1->a1b2 A2b2 -> **a2b1**
bottom	C1,c2	**{a2b1,**a1b1,a2b2}	a1b1->a1b2, A2b2 -> **a2b1**
adjacentHash	C1,c2,C3, c4	{**a2b1c1d2,**a1b1c1d2,a2b2c1d2,a2b1c2d2,,a2b1c1d1}	a1b1c1d2->a1b2c1d2 2b2c1d2->a2b1c1d2
Neighbours	C1 c2	{**a2b1,**a1b1,a2b2}	a1b1-> a1b2
encodeHash	c5 c6	**f1g1**,f2g1,f1g2	
encodeHash	c5 c6 c7	**{f1g1h1**,f2g1h1,f1g2h1,f1g1h2**}**	
encodeHash	c5 c6	**f1g1**,f2g1,f1g2	
encodeHash	c5 c6 c7	**{f1g1h1**,f2g1h1,f1g2h1,f1g1h2**}**	
decodeHash	C1 c5 c6 C2	**a2b1f1g1,**a1b1f1g1,a2b1f2g1,a2b1f1g2,a2b2f1g1	a1b1f1g1 -> a1b2f1g1

hashLengthToCoverBoundingBox	c5 c6 c7	**f1g1h1,**f2g1h1,f1g2h1,f1g1h2	
hashContains	C1 c5 c6 c2 c8	**A2b1f1g1i1**,A1b1f1g1i1,A2b2f1g1i1,A2b1f2g1i1,A2b1f1g2i1,A2b1f1g1i2	a1b1f1g1i1->a1b2f1g1i1,A2b1f2g1i1->,A2b1f2g1i2
coverBoundingBox	C5 c6 c9	**F1g1j1**,f1g1j2,f2g1j1,f1g2j1	f1g1j2->f1g1j1
coverBoundingBoxMaxHashes	C5 c6 c9 c10	**f1g1j1k2,**f1g1j2k2,f1g1j1k1,**f**2g1j1k2,f1g2j1k2	f1g1j2k2->f1g1j1k2
coverBoundingBox	C5 c6 c7 c9	**f1g1h1j1**, f1g1h1j2,**f**2g1h1j1,f1g2h1j1,f1g1h2j1	f1g1h1j2->f1g1h1j1
heightDegrees	C7	**h1,**h2	
widthDegrees	C7	**H1,**h2	
gridAsString	C1 c2 c11	**a2b1n3,**a1b1n3,a2b2n3,a2b1n2,a2b1n1,a2b1n4	a1b1n3-> A2b1n3
gridAsString	C1 c2 c12 c13 c14 c15	**a2b1p2q2r2s2**,a1b1p2q2r2s2,a2b2p2q2r2s2,a2b1p1q2r2s2,a2b1p2q1r2s2,a2b1p2q2r1s2,a2b1p2q2r2s1	a1b1p2q2r2s2->a2b1p2q2r2s2
gridAsString	C1 c2 c12 c13 c14 c15	**a2b1p2q2r2s2**,a1b1p2q2r2s2,a2b2p2q2r2s2,a2b1p1q2r2s2,a2b1p2q1r2s2,a2b1p2q2r1s2,a2b1p2q2r2s1	a1b1p2q2r2s2->a2b1p2q2r2s2

Boundary values

1. Equivalence partitions
    - D: value of steps < 0, value of steps =0, value of steps>0
    - F: latitude<-90, latitude between -90 and 90, latitude>90
    - G: latitude<-180, latitude between -180 and 180, latitude>180
    - H: length <0, length between 0 and 12, length >12
    - K: max<1, max=1, max>1
    - N: size<0, size=0,size=1, size>0
    - P: fromRight<=0, fromRight>0
    - Q: fromBottom<=0, fromBottom>0
    - R: toRight<=0, toRight>0
    - S: toBottom<=0, toBottom>0

2. Boundary values
    - D: -1, 0, 1 boundary is 0

- F: -91, -90, -89, 89, 90, 91   boundary is 90 and -90
- G: -181, -180, -179, 179, 180, 181 boundary is 180 and -180
- H: -1, 1, 12,13   boundary is 0 and 12
- K   0, 1, 2   boundary is 1
- N: -1, 0, 1 boundary is 0
- P: -1, 0, 1 boundary is 0
- Q: -1, 0, 1 boundary is 0
- R -1, 0, 1   boundary is 0
- S: -1, 0, 1   boundary is 0