**poker.js**

//poker.js 11/27/18

const RANKS\_INDEX = '--23456789TJQKA'

// return 0:equal; 1: a> b; -1: a < b

// assume a and b have same length

// this compArray() uses to compare ranks when the two hands have the same score.

const **compArray =** (a, b) => {

for (let i=0; i< a.length; i++) {

let flag = a[i] - b[i]

if (flag > 0)

return 1

else if (flag < 0) {

return -1

}

}

return 0

}

// I need a **compKind(**a, b) return a === b => true; otherwise => false

**const compKind = (a, b) =>** {

if (a.length !== b.length) {

return false

}

for (let i=0; i < a.length; i++) {

if (a[i] !== b[i]) {

return false

}

}

return true

}

// return 0:equal; 1: a> b; -1: a < b

// compare Map objects, I only need to compare the keys of the Map objects not the values

const **compMap** = (map1, map2) => {

// get keys of two maps, then compare arrays is easier

const keys1\_iterator = map1.keys()

const keys2\_iterator = map2.keys()

const keys1 = Array.from(keys1\_iterator)

const keys2 = Array.from(keys2\_iterator)

if ( compArray(keys1, keys2) > 0) {

return 1

} else if (compArray(keys1, keys2) < 0) {

return -1

} else {

return 0

}

}

const **bestHands** = hands => {

let best\_hands = hands[0]

const pk1 = new Poker(hands[0])

let best\_score\_map = pk1.score\_hand() // => [score, Map object]

let best\_score = best\_score\_map[0]

let best\_map = best\_score\_map[1]

// parse each hand

for(let i=1; i<hands.length; i++) {

const hand = hands[i]

let pk = new Poker(hand)

// get score and map object from poker.score\_hand()

const score\_map = pk.score\_hand()

const score = score\_map[0]

const map = score\_map[1]

// compare score

if (best\_score < score) {

best\_hands = hand

best\_score = score

} else if (best\_score === score) {

// need to compare ranks map

const flag = compMap(best\_map, map) // only place use compMap()

if (flag < 0) {

best\_hands = hand

best\_score = score

} else if (flag === 0) {

// it is tie, add hand to best\_hands

// change best\_hands from String to String[]

best\_hands = [best\_hands].concat(hands[i])

}

}

}

console.log(best\_hands)

console.log(best\_score)

return best\_hands

}

class Poker {

**constructor**(hand) {

this.hand = hand

this.suits = this.parseHand(true) // get suits

this.ranks = this.parseHand(false) // get ranks

// ranks\_map: keys= ranks, values = ranks counts

// ranks\_map is reverse sorted by ranks(key), ex: [ {11:1}, {8:2}, {2:2}]

this.ranks\_map = this.create\_ranks\_map()

}

// => ['2', 'S', '4', 'H', '6', 'S', '4', 'D', 'J', 'H']

get\_hand\_ary() {

const hand = this.hand

const hand1 = hand.replace(/\s/g,'')

const hand2 = hand1.replace(/10/g,'T') // '10' to 'T'

const hand\_ary = hand2.split('')

return hand\_ary

}

//const isOdd = x => x % 2 === 0

**parseHand(flag**) {

const hand\_ary = this.get\_hand\_ary()

let suits = [] // => ["S", "H", "S", "D", "H"]

let ranks = [] // => ["5", "2", "4", "A", "3"]

hand\_ary.map( (elem, idx) => {

if ( idx%2 === 0) { // works too

//if (isOdd(idx)) {

ranks.push(hand\_ary[idx])

} else {

suits.push(hand\_ary[idx])

}

})

if (flag) {

return suits

} else {

return ranks

}

}

// ranks = ["2", "8", "2", "8", "2"]

// => ranks\_map = [ {11:1}, {8:2}, {2:2} ] a map with keys reverse sorted

**create\_ranks\_map**() {

let ranks\_ary = []

// convert ranks chars to its index values

this.ranks.map( e => {

ranks\_ary.push(RANKS\_INDEX.indexOf(e))

})

**// reverse sorted ranks**

// sort() is char sorted => 11 is small than 2, bc 1 < 2

// for numeric sort I need sort( (a,b) => a -b))

// for reverse sort sort( (a,b) => b -a))

// => [11,8,8,2,2] is reverse sort of ranks array

ranks\_ary.sort( (a,b) => b - a )

// **create ranks\_map with reverse sorted ranks key**, value = rank count

let ranks\_map = new Map()

ranks\_ary.forEach( key => {

// if key exists

if (**ranks\_map.has(key**)) {

ranks\_map.set(key, ranks\_map.get(key) + 1)

} else {

ranks\_map.set(key, 1)

}

})

// key in ranks\_map is already key reverse sorted, ex: [{11:1}, {8:2}, {2:2}]

// when calculate the ranks score, I will need to reverse sort by volue

return ranks\_map

}

**score\_hand()** {

// check for flush

const flush = this.suits.every( e => this.suits[0] === e )

// create sorted\_ranks in value, sorted by ASC

// get keys from ranks\_map, then reverse sort keys from DESC to ASC

const sortedIterator = this.ranks\_map.keys() // => iterator

// convert iterator to Array

**let sorted\_ranks** = Array.from(sortedIterator).reverse() // change from DESC to ASC for straight comparison

// check for **special straight [**2,3,4,5,14] => modify this.rank\_map

if (compKind(sorted\_ranks, [2,3,4,5,14]) ) {

// console.log("special straight")

this.ranks\_map = [ [1,1], [2,1], [3,3], [4,4], [5,5] ]

sorted\_ranks = [1,2,3,4,5]

}

// **check straight**

let straight = true

for ( let i=0; i<sorted\_ranks.length; i++) {

if ( sorted\_ranks[i] !== **sorted\_ranks[0] + i)** {

straight = false

break

}

}

// ranks\_map is already key reverse sorted, ex:[ {11,1}, {8:2}, {2:2} ]

// create **ranks\_values Map = reverse sort values** on ranks\_map

// ranks\_values Map will be included in return statement

//const ranks\_values = Array.from(this.ranks\_map.entries()).sort((a,b) => b[1] - a[1]) // works

const ranks\_values = **new Map([...this.ranks\_map.entries()].sort((a,b) => b[1] - a[1]) )**

// create **kinds array**

const values\_map = ranks\_values.values() // => Iterator of values

const kinds = Array.from(values\_map) // convert Iterator to array of values, [4,1], [2,1,1,1]

**// calculate score: kinds, flush, straight**

// the reason why I need to check flush and straight before compare to any kinds of array,

// because the compArray will gives wrong result in comparing [1,2,3,4,5] and [4,1]

let score = 0

if (flush && straight) score = 8

else if ( compKind(kinds, [4,1]) ) score = 7

else if ( compKind(kinds, [3,2]) ) score = 6

else if ( flush ) score = 5

else if ( straight ) score = 4

else if ( compKind(kinds, [3,1,1]) ) score = 3

else if ( compKind(kinds, [2,2,1]) ) score = 2

else if ( compKind(kinds, [2,1,1,1]) ) score = 1

else score = 0

return **[score].concat(ranks\_values)** // ranks\_values is a Map object

}

}

export { bestHands }