SP Assignment 2

USN: 303039534

1 Skyscraper Problem

We start by writing a function skyplot() that plots a skyscraper problem given a list of edges and a solution or partial solution to plot.

```r
skyplot = function(edges, soln = '', ...){
  #edges is a vector of visible skyscrapers going clockwise from the top left
  #soln is a vector of values to put in the grid starting at the top left corner
  edges[edges == 0] = ''; soln[soln==0]='' #zero values indicate blanks
  N = length(edges) / 4; s = 0.8/N; cex = 7/N #step length and character size
  plot.new(); title(...) #initiate plot and add title
  segments( c(0.1,0.1,0.1,0.9), c(0.1,0.9,0.1,0.1),
            c(0.9,0.9,0.1,0.9), c(0.1,0.9,0.9,0.9) ) #outline
  segments(0.1, 0.1+(1:N)*s, 0.9, 0.1+(1:N)*s) #horizontal lines
  segments(0.1+(1:N)*s, 0.1, 0.1+(1:N)*s, 0.9) #vertical lines
  #add top + bottom edge values
  text( s*(0:(N-1)+0.5)+0.1, y=rep(c(.91, 0.09),each=N),
        edges[c(1:N,(3*N):(2*N+1))], pos = rep(c(3,1),each=N), cex = cex )
  #add left+right edge values
  text( rep(c(0.91,0.09),each=N), y=s*(0:(N-1)+0.5)+0.1,
        edges[c((2*N):(N+1),(3*N+1):(4*N))], pos = rep(c(4,2),each=N), cex = cex )
  #add solution
  text( rep(s*(0:(N-1)+0.5)+0.1, N), y=rep(s*((N-1):0+0.5)+0.1, each=N),
        soln, cex = cex)
}
```

We plot the skyscraper problems given in figure 1 using skyplot (figure 1).

```r
par(mfrow=c(1,2),mar=c(1,1,1,1))
skyplot( 1:20, 1:25, main = 'A')
skyplot( c(2,0,0,0,2,  0,0,0,2,0,  0,0,3,4,0,  0,1,0,2,0),
         c(0,0,0,0,0,  0,0,0,0,0,  0,0,0,0,5,  5,0,0,0,4, 0,0,0,0,0), main='B')
```

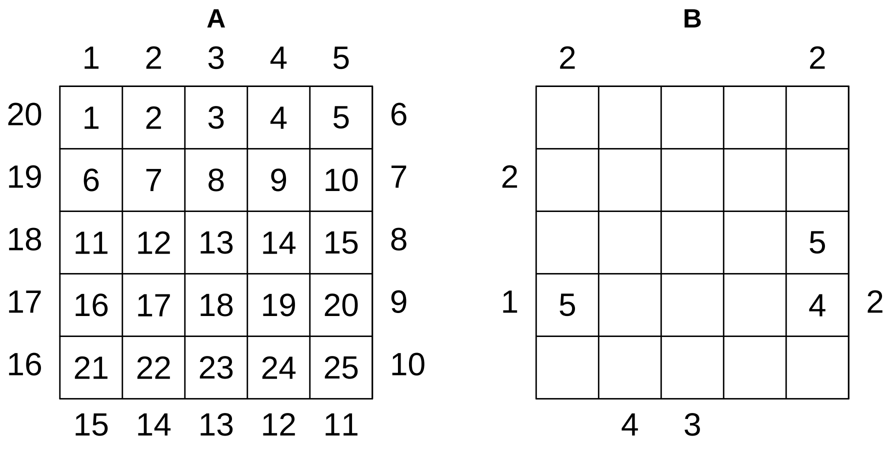

Figure 1: Layout of the skyscraper problem (left) and a partially solved 5x5 problem (right).

We proceed to write a function solve_sky() to solve any 4x4 skyscraper problem. This function takes as input a set of edges specifying the number of visible skyscrapers at each point. The algorithmic structure involves iterating through all possible allowed solutions (i.e. with each row and column containing the numbers 1:4) and checking it against the specified edges using the function all_visible(). all_visible() returns a list of the number of visible skyscrapers at each edge point of a given solution.

```
visible = function(vec, reverse = 0){
  #given a vector of skyscraper heights  (solution), returns the number visible ones
  #if reverse, looks from finish to start rather thans start to finish
  if (reverse) vec = rev(vec) #look from other direction
  n = 0; max = 0
  for (sky in vec){ if (sky > max){ n = n+1; max = sky } }
  return(n)
}

all_visible = function(mat){
  #given a matrix, generates a vector of the number of visible skyscrapers
  vis =
    c(apply(mat, 2, visible), #consider top edges
    apply(mat, 1, visible, reverse = 1), #right edges
    rev(apply(mat, 2, visible, reverse = 1)), #bottom edges, reverse output
    rev(apply(mat, 1, visible))) #left edges, reversing output order
  return(vis)
}

solve_sky = function(edges,main = 'puzzle', print=0){
  #given a list of edge values of visible skyscrapers for a 4x4 problem,
  #finds, prints and plots a solution to the problem
  sol = matrix(,4,4)
  sols = list() #store all solutions
  perms1 = permutations(4)
  new_main = main

  for (i in 1:factorial(4)){ #consider all permutations of 1:4
    sol[1,] = perms1[i,]
    #we can now remove the permutations that are not allowed together with
    #the first permutation (gives Nsol = 24*9*4*1 = 864
      #rather than 24^4 = 331776)
    perms2 = perms1[apply(perms1, 1, function(x){ !any(x == sol[1,]) }), ]

    for (j in 1:dim(perms2)[1]){
      sol[2,] = perms2[j,] #add next row
      perms3 = perms2[apply(perms2, 1, function(x){ !any(x == sol[2,]) }), ]

      for (k in 1:dim(perms3)[1]){
        sol[3,] = perms3[k,] #add next row
        #now we can infer final row
        sol[4,] = perms3[apply(perms3, 1, function(x){ !any(x == sol[3,]) }), ]
        vis = all_visible(sol) #get list of visible skyscrapers for sol
        vis[ edges == 0 ]=0 #don't look at edges with no information

        if ( all(edges == vis) ){ #check if solution consistent with problem
          if(print){cat('\nfound solution\n'); print(sol)}
          L = length(sols)
          if (L>0) new_main = paste0(main,as.character(L+1))
          skyplot(edges,c(t(sol)),main=new_main)
          sols[[L+1]] = c(t(sol)) #store solution
        }
      }
    }
  }
  N = length(sols)
  if (N == 0){cat('puzzle not solvable, sorry\n')} #if unsolvable
  #}else{cat('found', N, 'solutions\n\n')}
  return(sols)
}
```

Using these, we can solve the problems given to us in the assignment. Note that problem A only has a single solution as specified but may be modified to have multiple solutions. Problem B has four solutions and we plot all of them.

```
par(mfrow=c(1,2),mar=c(2,2,1.5,1))
e1 = scan(comment.char="#", quiet=T,
        'https://raw.githubusercontent.com/sje30/rpc2018/master/a2/e1.dat')
a = solve_sky(e1, main = 'A')
#testm = c(0,2,1,0, 0,0,1,3, 0,2,0,0, 1,0,2,2)
#c = solve_sky(testm, main = 'test')
```

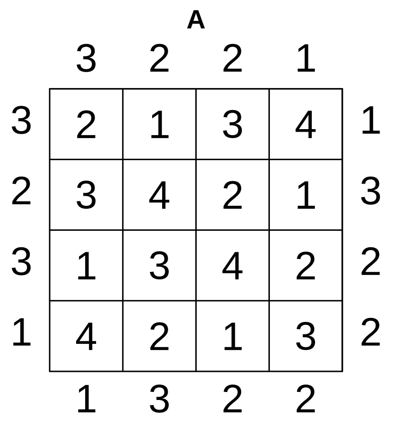

Figure 2: Solution(s) to the problem e1.

```r
par(mfrow=c(1,2),mar=c(2,2,1.5,1))
e2 = c(3,0,1,0, 0,0,1,0, 0,0,0,0, 0,0,0,2)
b = solve_sky(e2, main = 'B')
```

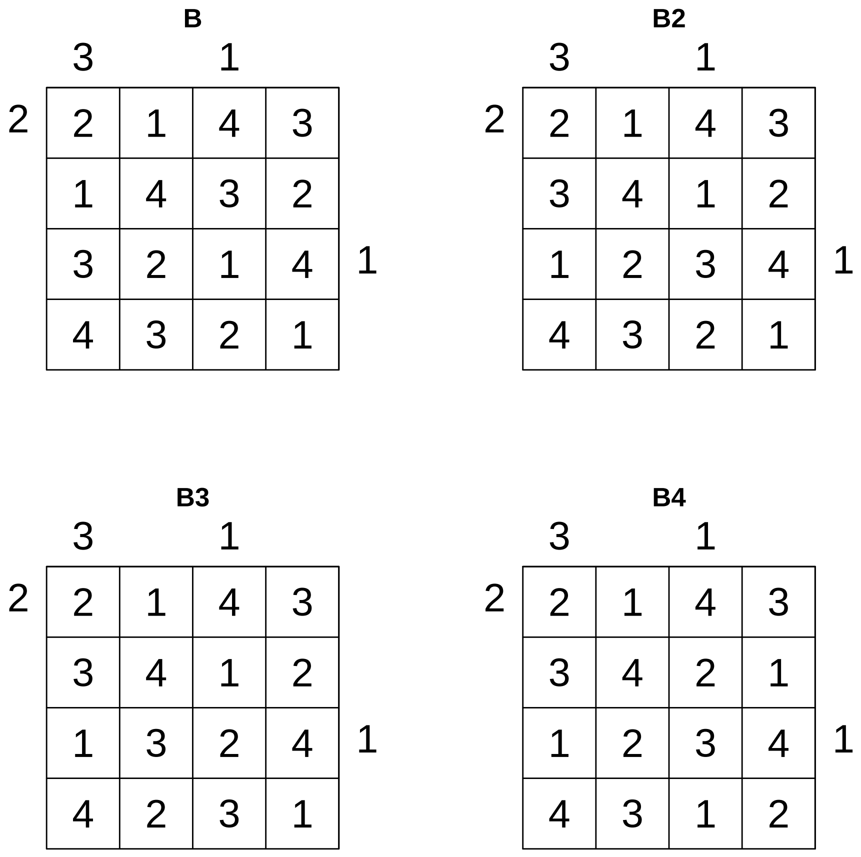

Figure 3: Solutions to the problem e2.

2 Casino Problems

We intially define functions to play the four games A, B, C and D. The functions take no arguments and play the specified game a single time.

```r
set.seed(09111023) #set seed for repeatable simulation

play_A = function(){
  if( runif(1) < (18/37) ){return( c(1, 1) ) #if red, win £1. Always place 1 bet
  }else{return( c(-1,1) )} #lose £1
}

play_B = function(){
  if( runif(1) < (1/37) ){return( c(35, 1) ) #win £35. Always place 1 bet
  }else{return(c(-1,1))} #lose £1
}

play_C = function(show = 0){
  wins = 0; bet = 1; n = 0

  while ( wins < 10 & bet <= 100 ){ #continue playing until won 10 or bet > 100
```

```
    if(show) cat('\nnet bet', bet, '\n')
    n = n+1 #count number of bets
    if( runif(1) < (18/37) ){ wins = wins+bet; bet = 1 #win amount bet, reset bet
    }else{ wins = wins-bet; bet = 2*bet } #lose amount bet, double bet
  }
  return( c(wins, n) )
}

play_D = function(show = 0){
  nums  = 1:4; wins = 0; n = 0; bet = 5

  while (length(nums) > 0 & bet <= 100){ #stop when empty list or bet > 100
    if(show) cat('\nnew nums', nums, 'bet', bet, '\n')
    n = n+1 #count number of bets

    if( runif(1) < (18/37) ){ wins = wins+bet; nums = nums[-c(1,length(nums))]
    #win amount bet, remove first, last num

    }else{ wins = wins-bet; nums = c(nums, bet) }
    #lose amount bet, add sum of first, last num to list

    if (length(nums)==0){bet = 0 #we're done playing
    }else if (length(nums) == 1){bet = nums #if only one num, bet that next
    }else{bet = nums[1]+nums[length(nums)]} #bet sum of first and last number}

  }

  return( c(wins, n) )
}
```

We also define a function play_many() to play a given type of game 100,000 times and return means and standard deviations of the amount won, games won and number of bets placed. Additionally, we write a function sim_games() to play each game 100,000 times and return the results as a dataframe and list for displaying and further analysis.

```
play_many = function(type, N=100000, print=1){
  #Takes as input a type of game (A,B,C,D) and plays it N times
  #Returns a list of mean and sd of amount won/lost, games won and bets placed
  f = switch(type, #pick correct game
        'A' = play_A,
        'B' = play_B,
        'C' = play_C,
        'D' = play_D,
        )

  amounts = bets = wins = numeric(N)
  for (i in 1:N){
    out = f() #play game and add results to lists
    amounts[i]=out[1]
    bets[i]=out[2]
    wins[i] = as.numeric( out[1] > 0 )
  }
  if (print){text = paste0('\nPlayed game ', type, ' won ', round(100*mean(wins),1),
                        ' percent of games. Max amount won was ', max(amounts),
                        ', max amount lost was ', abs(min(amounts)), '\n')
            cat(text)}
  return(c(mean(amounts), mean(wins), mean(bets), sd(amounts), sd(wins), sd(bets)))
}

sim_games = function(print = 1){
  #for each of (A,B,C,D), play 100000 times and return a df and list with results
  awin = pwin = nbets = numeric(4)
  types = c('A', 'B', 'C', 'D')
  outs = matrix(,4,6)
  for (type in 1:4){
    out = play_many(types[type], print=print) #play 100000 games and store results
    awin[type] = paste0(as.character(round(out[1],3)),'  (',
                        as.character(round(out[4],2)),')')
    pwin[type] = paste0(as.character(round(100*out[2],1)),'%  (',
                        as.character(round(100*out[5],1)), '%)')
    nbets[type] = paste0(as.character(round(out[3],1)),'  (',
                         as.character(round(out[6],1)),')')
    outs[type,] = out
  }
  df = data.frame('Winnings_mean_sd' = awin,
                  'Prop.wins_mean_sd' = pwin,
                  'Play.time_mean_sd' = nbets)
  row.names(df) = types
  return(list(df, outs))
}
```

Using these functions, we play each game 100,000 times and give the means and standard deviations of the previously mentioned quantities in table 1.

```
res = sim_games()
```

```
Played game A won 48.9 percent of games. Max amount won was 1, max amount lost was 1
Played game B won 2.7 percent of games. Max amount won was 35, max amount lost was 1
Played game C won 91 percent of games. Max amount won was 10, max amount lost was 127
Played game D won 95.7 percent of games. Max amount won was 10, max amount lost was 1051
```

```
xres = xtable(res[[1]], caption = 'Summary of game results',
              digits=c(0,3,3,1), label='tab:sim')
addtorow = list()
addtorow$pos = list(0)
addtorow$command <- c(' & Winnings (sd) & Prop wins (sd) & Play time (sd) \\\\\n')
align(xres) <- rep("r", 4)
print(xres, include.colnames = FALSE, add.to.row = addtorow)
```

We see that the best games to play in terms of losing the least money are games A and B. These are also the shortest games as they always involve a single bet, but they still provide the smallest loss per bet. They also have the least variation in the amount won whereas game D is most variable in the amount won or lost. While the amount won in D is always the same, the amount lost is highly variable and can be very large. Games A and B have fixed playing times of 1 bet whereas C and D

	Winnings (sd)	Prop wins (sd)	Play time (sd)
A	-0.022 (1)	48.9% (50%)	1 (0)
B	-0.023 (5.85)	2.7% (16.3%)	1 (0)
C	-1.889 (37.89)	91% (28.6%)	19.5 (4.5)
D	-3.899 (70.9)	95.7% (20.3%)	8.8 (7.6)

Table 1: Summary of game results

have more variable playing times. Out of these two, game C has the highest average playing time, but game d is more variable in its playing time and can lead to some very long games. For games A and B, we can calcuate the exact result for the expected winnings and expected proportion of games won. These are given in table 2.

```
outs = res[[2]]
wins = c(18/37*1-19/37*1, 1/37*35-36/37*1)
props = c(18/37, 1/37);
df_ex = data.frame('Winnings' = wins,
                   'Error' = paste0(round((c(outs[1,1], outs[2,1])-wins)/
                                              wins*100,1),'%'),
                    'Proportion' = paste0(round(100*props,1),'%'),
                   'Error' = paste0(round((c(outs[1,2], outs[2,2])-props)/
                                              props*100,1),'%'),
                    row.names = c('A', 'B') )
xres = xtable(df_ex, caption = 'Exact expected winnings',
              digits = c(0,3,1,3,1), label='tab:ex')
addtorow = list()
addtorow$pos = list(0)
addtorow$command <- c(' & Expected Winnings & Error & Prop wins & Error \\\\\n')
align(xres) <- rep("r", 5)
print(xres, include.colnames = FALSE, add.to.row = addtorow)
```

	Expected Winnings	Error	Prop wins	Error
A	-0.027	-16.8%	48.6%	0.5%
B	-0.027	-16.4%	2.7%	0.5%

Table 2: Exact expected winnings

We see that the proportion of games won is very similar to the expected proportion of games won from above. The absolute discrepancy in amount won is also very small as it is completely determined by the proportion of games won, but the relative error is quite large as the amount lost per game is a rather small number.
We can also work out exactly the maximum amounts won and lost in a given type of game. This is given in table 3.

```
df_wl = data.frame('A' = c(1,1),
                   'B' = c(35,1),
                    'C' = c(10, 1+2+4+8+16+32+64),
                   'D' = c(10, sum(5:100)),
                   row.names = c('Max Win', 'Max Loss'))
xres = xtable(df_wl, caption = 'Exact maximum amount won and lost',
              digits = c(0,0,0,0,0), label='tab:max')
align(xres) <- rep("r", 5)
```

```
print(xres)
```

	A	B	C	D
Max Win	1	35	10	10
Max Loss	1	1	127	5040

Table 3: Exact maximum amount won and lost

We trivially observe that the maximum amounts won and lost are \$1 and \$1 respectively in game A and \$35 and \$1 in game B. These results are observed many time. In games C and D, the only amount that can be won is \$10, and this is thus commonly observed. In game C, the maximum loss is \$127 (requiring 7 losses in a row) and this is seen on average 1 in 106 times. The maximum loss of 5040 in game D requires 96 losses in a row, which only occurs 1 in 10^{28} times and is thus not observed in our simulation.
Finally, we run the simulation of 100,000 games 5 times and report the results in table 4 to see how variable the results of our previous simulation are.

```
repeat_sim = function(rownames=c('A', 'B', 'C', 'D')){
  #run 100000 game simulations five times and return data frame
  #with min and max of amounts won, proportion won and bets placed.
  wins = props = bets = matrix(,4,5)
  for (i in 1:5){
    outs = sim_games(print=0)[[2]]
    wins[,i] = outs[,1]; props[,i] = outs[,2]; bets[,i] = outs[,3]
  }
  df_summ = data.frame('Winnings_min' = apply(wins, 1, min),
                'Winnings_max' = apply(wins, 1, max),
                'Prop.wins_min' = paste0(round(apply(props, 1, min)*100,1),'%'),
                'Prop.wins_max' = paste0(round(apply(props, 1, max)*100,1),'%'),
                'Play.time_min' = apply(bets, 1, min),
                'Play.time_max' = apply(bets, 1, max))
  row.names(df_summ) = rownames
  return(df_summ)
}
df_summ = repeat_sim()
xres = xtable(df_summ, caption = 'Variability of simulation',
       digits=c(0,3,3,3,3,3,3), label='tab:var')
addtorow = list()
addtorow$pos = list(0,0)
addtorow$command = c(' & Winnings & Winnings & Prop wins &
                     Prop wins & Play time & Play time \\\\n',
                     ' & min & max & min & max & min & max \\\\n')
align(xres) <- rep("r", 7)
print(xres, include.colnames = FALSE, add.to.row = addtorow)
```

We observe that the amount lost has the highest relative variability for game B as it has the lowest and thus most variable frequency of winning. In fact, for the best round of 100,000 games we almost win money with strategy B. The proportion of games won has very little absolute variation, but again the largest relative variation for game B leading to the aforementioned variation in winnings. The playing time is of course still fixed for games A and B, but even for games C and D we see that the expected playing time per game over 100,000 games is actually remarkably constant.

	Winnings	Winnings	Prop wins	Prop wins	Play time	Play time
	min	max	min	max	min	max
A	-0.031	-0.023	48.4%	48.8%	1.000	1.000
B	-0.056	-0.002	2.6%	2.8%	1.000	1.000
C	-2.259	-1.756	90.8%	91.1%	19.518	19.537
D	-3.872	-3.568	95.7%	95.8%	8.740	8.783

Table 4: Variability of simulation

In summary, we should thus play game A or B if we want to minimize losses per bet or maximum losses, game A if we want low variability, game C if we want the longest playing time, and game D if we won the highest proportion of games won.

3 Appendix

```r
## Taken from: http://stackoverflow.com/questions/11095992
permutations <- function(n){
  #returns a matrix or all possible permutations of the numbers 1:n
  if(n==1){
  return(matrix(1))
    } else {
    sp <- permutations(n-1)
    p <- nrow(sp)
    A <- matrix(nrow=n*p,ncol=n)
    for(i in 1:n){
      A[(i-1)*p+1:p,] <- cbind(i,sp+(sp>=i))
    }
    return(A)
  }
}
```