Simulación Iterativa de Supervivencia mediante Canje de Dulces

Universidad Nacional del Altiplano - Puno

Facultad de Ingeniería Estadística e Informática

Escuela Profesional: Ingeniería Estadística e Informática

Estudiante: Wily Calib Caira Huancollo
Docente: Fred Torres Cruz

Curso: Programación Numérica

30 de octubre de 2025

Índice

1. Introducción

Este documento presenta la implementación iterativa en R de un *ejercicio de supervivencia* mediante canje y cooperación: los participantes (integrantes de equipos) obtienen chupetines a partir de canjes de dulces, usan devolución colaborativa y bonos para ayudar a que todos sus compañeros sobrevivan. El trabajo cumple el enunciado del curso: *Realiza la implementación de una solución iterativa que pueda abstraer un ejercicio de supervivencia.*

2. Objetivo

Modelar de forma iterativa y programática un juego cooperativo de supervivencia donde el objetivo operativo es formar grupos de dulces para obtener chupetines y —por cooperación mediante un fondo común de dulces y devolución de chupetines— lograr que todos los integrantes de un equipo consigan al menos un chupetín.

3. Reglas del juego (resumidas y claras)

1. **Equipos:** Dos equipos (Equipo A y Equipo B). El número de integrantes es variable y se ingresa por consola al ejecutar el script.
2. **Dulces:** Hay 3 tipos distintos de dulce (Dulce1, Dulce2, Dulce3).
3. **Inicio:** Cada integrante inicia con una cantidad aleatoria de dulces entre 1 y 2 (tipos asignados al azar).
4. **Reparto por ronda:** En cada iteración (ronda), los integrantes que todavía estén activos (no "salvados") reciben 1 o 2 dulces aleatorios.
5. **Canje:**
   - Un **grupo** de 3 *dulces distintos* se canjea por **1 chupetín**.
   - Si un integrante entrega **2 grupos de 3 dulces distintos** (6 dulces totales) en una misma oportunidad, recibe **2 chupetines** y además el equipo recibe **1 dulce extra** que va al *fondo común* (bonificación para seguir formando grupos).
6. **Fondo común (modelo A):** Los dulces extra de bonificación se depositan en un fondo del equipo para que los miembros que aún no tienen chupetín puedan usar esos dulces para formar grupos.
7. **Devolución:** Un integrante que ya posee un chupetín puede **devolverlo voluntariamente** para recuperar 3 dulces que pasan al fondo del equipo (permitiendo ayudar a otros). Esta devolución se procesa en la simulación como una acción que el algoritmo puede solicitar cuando el fondo está vacío y hay integrantes por salvar.
8. **Retiro tras salvarse:** Por diseño, cuando un integrante consigue al menos 1 chupetín, queda marcado como *salvado* (retira de recibir nuevos dulces). Sin embargo, puede decidir devolver su chupetín —en cuyo caso deja de estar marcado como salvado y vuelve a participar— según la estrategia establecida por la simulación.

9. **Condición de terminación:** La simulación es iterativa y finaliza cuando **un equipo** logra que **todos sus integrantes** hayan conseguido al menos 1 chupetín (es decir, el equipo completo está salvado). El primer equipo que alcanza esa condición se declara *ganador*.

4. Diseño iterativo y algoritmo (pseudocódigo)

A continuación se presenta la idea iterativa que implementa el algoritmo:

```
Pseudocódigo (resumen):
1. Leer n = número de integrantes por equipo (usuario)
2. Inicializar estado: para cada integrante -> dulces (1-2 aleatorios), chupetines=0,
3. Inicializar fondoA = 0, fondoB = 0, ronda = 0
4. Mientras (ningún equipo esté completamente salvado) y (ronda < tope):
   a) ronda += 1
   b) Para cada integrante activo (no salvado):
        - Recibe 1 o 2 dulces aleatorios (se añaden a su inventario)
   c) Para cada integrante activo:
        - Intentar formar grupos de 3 dulces distintos tantas veces como sea posible
        - Por cada grupo -> +1 chupetín (remover 3 dulces distintos)
        - Si forma 2 grupos en una misma vez -> además fund += 1 dulce extra
        - Si consigue al menos 1 chupetín -> marcar salvado = TRUE
   d) Si hay integrantes sin chupetín y el fondo es insuficiente:
        - Solicitar devolución de chupetines a salvados que aún no hayan devuelto
        - Cada devolución -> resta 1 chupetín al salvado y añade 3 dulces al fondo
   e) Usar el fondo para completar canjes de integrantes que no pueden formar grupo po
   f) Mostrar resumen de la ronda (número de salvados, fondo, canjes, devoluciones)
5. Declarar ganador: el equipo que completa todos sus salvados primero
```

5. Implementación iterativa en R

El siguiente script en R implementa la simulación completa. Copia y pega en un archivo `simulacion_supervivencia.R` y ejecútalo en R o RStudio. El script solicita interactivamente el número de integrantes por equipo.

```r
# simulacion_supervivencia.R
# Simulaci n iterativa de supervivencia mediante canje de dulces (
    fondo com n)
# Reglas implementadas seg n el enunciado del estudiante.

set.seed(123)  # reproducibilidad (opcional)

# --- Par metros y entrada ---
n <- as.integer(readline(prompt = "Ingrese numero de integrantes
    por equipo (entero): "))
if (is.na(n) || n <= 0) stop("Debe ingresar un entero mayor que 0."
    )

```

```r
tipos_dulces <- c("Dulce1", "Dulce2", "Dulce3")
num_estudiantes <- 2 * n
equipoA_idx <- 1:n
equipoB_idx <- (n+1):(2*n)

# Estado por estudiante
dulces_por_estudiante <- vector("list", num_estudiantes)
chupetines_por_estudiante <- integer(num_estudiantes)
salvado <- rep(FALSE, num_estudiantes)     # si tiene al menos 1 chupet n y est  "salvado"
devolvio <- rep(FALSE, num_estudiantes)   # si ya devolvi  alguna vez
canjes_por_estudiante <- integer(num_estudiantes)

# Fondo por equipo
fondoA <- 0
fondoB <- 0

# Inicializar dulces (entre 1 y 2 aleatorios por estudiante)
for (i in 1:num_estudiantes) {
  k <- sample(1:2, 1)
  dulces_por_estudiante[[i]] <- sample(tipos_dulces, k, replace = TRUE)
}

# Funciones auxiliares
dar_dulces <- function() {
  k <- sample(1:2, 1)
  sample(tipos_dulces, k, replace = TRUE)
}

# Intenta formar grupos de 3 dulces distintos para el estudiante i
# Devuelve lista: updated dulces vector, grupos_formados (integer)
formar_grupos <- function(dulces_vec) {
  grupos <- 0
  # while hay al menos 1 de cada tipo
  repeat {
    tipos_unicos <- unique(dulces_vec)
    if (length(tipos_unicos) < 3 || length(dulces_vec) < 3) break
    # removemos una unidad de cada tipo (si hay varias repeticiones, quitar cualquiera)
    # asegurarse de tener al menos una de cada tipo
    # seleccionar los 3 tipos distintos
    grupos <- grupos + 1
    # eliminar una de cada tipo: remover la primera ocurrencia de cada tipo
    for (t in c("Dulce1","Dulce2","Dulce3")) {
      pos <- match(t, dulces_vec)
      if (!is.na(pos)) dulces_vec <- dulces_vec[-pos]
    }
  }
```

```
  return(list(dulces = dulces_vec, grupos = grupos))
}

# Usar fondo para completar canjes: intenta usar 'fondo' para dar dulces a estudiantes
usar_fondo_para_estudiante <- function(i, fondo) {
  # si estudiante tiene 2 tipos y necesita 1 dulce para completar grupo, podemos dar uno
  # Estrategia simple: mientras hay fondo y el estudiante puede formar grupo increment ndolo, darle dulces
  cambios <- 0
  while (fondo > 0) {
    res <- formar_grupos(dulces_por_estudiante[[i]])
    if (res$grupos > 0) break  # ya puede formar
    # si no puede formar, intentar a adir 1 dulce del fondo
    # dar un dulce aleatorio del tipo que m s falta
    # simplificamos: dar un dulce aleatorio
    nueva <- sample(tipos_dulces, 1, replace = TRUE)
    dulces_por_estudiante[[i]] <<- c(dulces_por_estudiante[[i]], nueva)
    fondo <- fondo - 1
    cambios <- cambios + 1
    # comprobar si ahora puede formar grupo
    res2 <- formar_grupos(dulces_por_estudiante[[i]])
    if (res2$grupos > 0) {
      # aplicamos las eliminaciones correspondientes
      dulces_por_estudiante[[i]] <<- res2$dulces
      return(list(fondo = fondo, formar = TRUE))
    }
  }
  return(list(fondo = fondo, formar = FALSE))
}

# Simulaci n principal
ronda <- 0
tope_rondas <- 1000
victoria <- NA
ronda_victoria <- NA

cat("\nINICIO DE LA SIMULACI N\n")
cat(sprintf("Integrantes por equipo: %d | Total: %d\n\n", n, num_estudiantes))

while (ronda < tope_rondas) {
  ronda <- ronda + 1
  cat("========== Ronda", ronda, "==========\n")

  # 1) Reparto de dulces a estudiantes no salvados
  for (i in 1:num_estudiantes) {
    if (!salvado[i]) {
      nuevos <- dar_dulces()
```

```
      dulces_por_estudiante[[i]] <- c(dulces_por_estudiante[[i]],
          nuevos)
    }
  }

  # 2) Intentar canjes por estudiante (cada estudiante por su lado)
  total_canjes_ronda <- 0
  total_grupos_ronda <- 0
  for (i in 1:num_estudiantes) {
    if (salvado[i]) next  # si ya salvado, por ahora no canjea
    res <- formar_grupos(dulces_por_estudiante[[i]])
    grupos <- res$grupos
    if (grupos > 0) {
      # otorgar chupetines: uno por grupo
      chupetines_por_estudiante[i] <- chupetines_por_estudiante[i]
          + grupos
      canjes_por_estudiante[i] <- canjes_por_estudiante[i] + grupos
      total_canjes_ronda <- total_canjes_ronda + grupos
      total_grupos_ronda <- total_grupos_ronda + grupos
      # si hizo pares de grupos (cada 2 grupos) -> por cada par,
          agregar 1 dulce al fondo del equipo
      pares <- floor(grupos / 2)
      if (i %in% equipoA_idx) {
        fondoA <- fondoA + pares
      } else {
        fondoB <- fondoB + pares
      }
      # actualizar dulces restantes del estudiante
      dulces_por_estudiante[[i]] <- res$dulces
      # si obtuvo al menos 1 chupet n , marcar salvado (retira)
      if (chupetines_por_estudiante[i] >= 1) {
        salvado[i] <- TRUE
      }
    }
  }

  # 3) Usar fondos para ayudar a estudiantes que no pueden formar
      grupo pero necesitan apoyo
  # Primero intentar con fondoA para equipo A
  for (i in equipoA_idx) {
    if (salvado[i]) next
    if (length(dulces_por_estudiante[[i]]) < 3) {
      resF <- usar_fondo_para_estudiante(i, fondoA)
      fondoA <- resF$fondo
      if (resF$formar) {
        # si pudo formar por uso de fondo, procesar de nuevo para
            contabilizar grupos
        res2 <- formar_grupos(dulces_por_estudiante[[i]])
        grupos2 <- res2$grupos
        if (grupos2 > 0) {
          chupetines_por_estudiante[i] <- chupetines_por_estudiante
```

```
            [i] + grupos2
149         canjes_por_estudiante[i] <- canjes_por_estudiante[i] +
               grupos2
150         pares2 <- floor(grupos2 / 2)
151         fondoA <- fondoA + pares2
152         dulces_por_estudiante[[i]] <- res2$dulces
153         if (chupetines_por_estudiante[i] >= 1) salvado[i] <- TRUE
154       }
155     }
156   }
157 }
158 # Luego fondoB similar
159 for (i in equipoB_idx) {
160   if (salvado[i]) next
161   if (length(dulces_por_estudiante[[i]]) < 3) {
162     resF <- usar_fondo_para_estudiante(i, fondoB)
163     fondoB <- resF$fondo
164     if (resF$formar) {
165       res2 <- formar_grupos(dulces_por_estudiante[[i]])
166       grupos2 <- res2$grupos
167       if (grupos2 > 0) {
168         chupetines_por_estudiante[i] <- chupetines_por_estudiante
             [i] + grupos2
169         canjes_por_estudiante[i] <- canjes_por_estudiante[i] +
             grupos2
170         pares2 <- floor(grupos2 / 2)
171         fondoB <- fondoB + pares2
172         dulces_por_estudiante[[i]] <- res2$dulces
173         if (chupetines_por_estudiante[i] >= 1) salvado[i] <- TRUE
174       }
175     }
176   }
177 }
178
179 # 4) Si a n hay integrantes sin chupet n y fondo es 0,
     solicitar devoluciones autom ticas si hay salvados dispuestos
180 # estrategia: devolver de salvados que no hayan devuelto aun, en
     orden, hasta que fondo >=1 o no queden salvados
181 # devolver implica: -1 chupet n (al salvado), devolvio=TRUE,
     a adir 3 dulces al fondo correspondiente, y marcar salvado
     FALSE
182 solicitar_devoluciones <- function(equipo_idx, fondo_var) {
183   for (i in equipo_idx) {
184     if (fondo_var >= 1) break
185     if (salvado[i] && !devolvio[i] && chupetines_por_estudiante[i
         ] >= 1) {
186       # devolver
187       chupetines_por_estudiante[i] <- chupetines_por_estudiante[i
           ] - 1
188       devolvio[i] <<- TRUE
189       # el salvado que devuelve deja de estar marcado salvado (
```

```
            vuelve a participar)
        if (chupetines_por_estudiante[i] == 0) salvado[i] <<- FALSE
        # se agregan 3 dulces al fondo
        fondo_var <- fondo_var + 3
        cat(sprintf("Integrante %d devolvio 1 chupet n y aport  3
            dulces al fondo.\n", i))
      }
    }
    return(fondo_var)
  }

  # Solo solicitar si hay integrantes sin chupet n en el equipo
  if (any(!salvado[equipoA_idx]) && fondoA == 0) {
    fondoA <- solicitar_devoluciones(equipoA_idx, fondoA)
  }
  if (any(!salvado[equipoB_idx]) && fondoB == 0) {
    fondoB <- solicitar_devoluciones(equipoB_idx, fondoB)
  }

  # 5) Mostrar resumen de la ronda
  salvadosA <- sum(salvado[equipoA_idx])
  salvadosB <- sum(salvado[equipoB_idx])
  cat(sprintf("Ronda %d - Salvados: Equipo A %d/%d | Equipo B %d/%d
     \n",
              ronda, salvadosA, n, salvadosB, n))
  cat(sprintf("Fondo - Equipo A: %d dulces | Equipo B: %d dulces\n"
     , fondoA, fondoB))
  cat(sprintf("Canjes esta ronda: %d (grupos formados en total: %d)
     \n", total_canjes_ronda, total_grupos_ronda))
  cat(sprintf("Total chupetines (A): %d | (B): %d\n", sum(
     chupetines_por_estudiante[equipoA_idx]), sum(chupetines_por_
     estudiante[equipoB_idx])))
  cat("\n")

  # 6) Condici n de victoria: primer equipo en completar todos sus
      salvados
  if (salvadosA == n) {
    victoria <- "Equipo A"
    ronda_victoria <- ronda
    cat(">>> Victoria: Todos los integrantes del Equipo A tienen al
        menos 1 chupet n.\n\n")
    break
  }
  if (salvadosB == n) {
    victoria <- "Equipo B"
    ronda_victoria <- ronda
    cat(">>> Victoria: Todos los integrantes del Equipo B tienen al
        menos 1 chupet n.\n\n")
    break
  }

```

```
231   # 7) Continuar hasta tope de rondas
232   if (ronda >= tope_rondas) {
233     cat("Se alcanz  el tope de rondas. Terminando simulaci n por
          seguridad.\n")
234     break
235   }
236 }
237
238 # RESULTADOS FINALES - modo texto explicativo
239 cat("======== RESULTADOS FINALES ========\n")
240 cat(sprintf("Rondas ejecutadas: %d\n", ronda))
241 if (!is.na(victoria)) {
242   cat(sprintf("Equipo ganador: %s (en ronda %d)\n", victoria, ronda
        _victoria))
243 } else {
244   cat("No se alcanz  un ganador por tope de rondas.\n")
245 }
246 cat("\nDetalle por integrante (formato: Integrante | Equipo |
      Chupetines | Canjes | Dulces restantes):\n")
247 for (i in 1:num_estudiantes) {
248   equipo <- ifelse(i %in% equipoA_idx, "A", "B")
249   dulces_str <- if (length(dulces_por_estudiante[[i]])==0) "-" else
        paste(dulces_por_estudiante[[i]], collapse=", ")
250   cat(sprintf("Integrante %02d | Equipo %s | Chupetines: %d |
        Canjes: %d | Dulces: %s\n",
251               i, equipo, chupetines_por_estudiante[i], canjes_por_
                    estudiante[i], dulces_str))
252 }
253 cat("\nResumen final por equipo:\n")
254 cat(sprintf("Equipo A - Chupetines totales: %d | Salvados: %d/%d |
      Fondo: %d dulces\n",
255             sum(chupetines_por_estudiante[equipoA_idx]), sum(
                  salvado[equipoA_idx]), n, fondoA))
256 cat(sprintf("Equipo B - Chupetines totales: %d | Salvados: %d/%d |
      Fondo: %d dulces\n",
257             sum(chupetines_por_estudiante[equipoB_idx]), sum(
                  salvado[equipoB_idx]), n, fondoB))
```

6. Ejemplo de uso

1. Guardar el código en `simulacion_supervivencia.R`.
2. Ejecutar en RStudio o R: `source("simulacion_supervivencia.R")` y seguir la solicitud para ingresar el número de integrantes por equipo.
3. Observar la salida paso a paso en la consola. El programa imprimirá cada ronda y, al final, un resumen en modo texto (como solicitaste).

7. Conclusiones breves

La implementación presentada es un modelo iterativo y acumulativo de supervivencia cooperativa con recursos discretos (dulces). El uso de un fondo común y la posibilidad de devolución de chupetines introduce una dinámica estratégica interesante: sacrificar un salvado temporalmente para permitir que otros formen grupos y, por ende, aumentar la probabilidad de que el equipo complete la condición colectiva de supervivencia.

Documento preparado por: Wily Calib Caira Huancollo
Docente: Fred Torres Cruz – Curso: Programación Numérica
Universidad Nacional del Altiplano - Puno