**Sistemas Adaptativos**
**Tarea 2**
**Heurísticas Greedy para el 2-Optimality Consensus Problem**

Alvaro Matamala
Miguel Villa
Diego Maldonado

## Pseudocódigo

GRASP

```
// Función que implementa el algoritmo de búsqueda local con Grasp
// Recibe un vector de strings con las secuencias de ADN
// Retorna un par con el valor objetivo y el tiempo de ejecución
FUNCION grasp(s: vector de caracteres,tam_string: entero, t_limite: entero) → (entero, largo largo) {
    num_init_sol: entero = 30;
    m: entero= tam_string;
    best_dist = Valor_maximo_de_entero
    best_sol: caracteres;

    start_time: tiempo de inicio; // Marcar el tiempo de inicio
    duration: Duración en segundos

    WHILE (Obtener_tiempo_actual() - start_time) en segundos < t_limite
        solucion_inicial: Vector de caracteres
        dist_solucion_inicial: entero;
        sol_actual: caracteres;
        dist_actual: entero;
        // Inicia el algoritmo de búsqueda local
        // Map de mejores soluciones con la solución y su distancia
        mejores_soluciones: mapa de caracteres y enteros;

        FOR i DESDE 0 HASTA num_init_sol CON PASO 1 { // Recorremos soluciones iniciales
            // Genera una solución inicial aleatoria usando el algoritmo Greedy
            (dist_solucion_inicial, solucion_inicial) = greedy(s, 1, m)
            sol_actual = Crear_cadena(solucion_inicial)
            dist_actual = dist_solucion_inicial;

            FOR j DESDE 0 HSATA m - 1 CON PASO 1  // Recorremos cada caracter de la solución
                PARA CADA c en {'A', 'T', 'C', 'G'} HACER
                    SI sol_actual[j] ES IGUAL A c ENTONCES
                        CONTINUAR
                    FIN SI
                    nueva_solucion = Copiar(sol_actual)
                    nueva_solucion[j] = c
                    nueva_dist = Calcular_distancia(nueva_solucion, s)
                    SI nueva_dist < dist_actual Entonces
                        dist_actual = nueva_dist
                        sol_actual = Copiar(nueva_solucion)
                    FIN SI
                FIN PARA
            FIN PARA

            Insertar_en_mapa(sol_actual, dist_actual)
        }

        PARA CADA x en mejores_soluciones Hacer // Elige la mejor solución entre mejores_soluciones
            Si x.distancia < dist_actual Entonces
            dist_actual = x.distancia
            sol_actual = x.solucion
            Fin Si
        FIN PARA

        SI best_sol está vacío O dist_actual < best_dist Entonces // Termina el algoritmo de búsqueda local
            best_dist = dist_actual;
            best_sol = sol_actual;
            end_time = Obtener_tiempo_actual(); // Marcar el tiempo de finalización
            duration = Duración_en_segundos(end_time - start_time);
        FIN SI

    RETORNA la mejor distancia y el tiempo de ejecución;
FIN FUNCION
```

## Explicación código

Se inicia el proceso con la definición de un número de soluciones iniciales y la inicialización de un contador de tiempo de ejecución. Luego, se entra en un bucle que se ejecuta hasta que se alcance el límite de tiempo. En cada iteración de este bucle, se generan soluciones iniciales aleatorias utilizando el algoritmo greedy aleatorio, las cuales se almacenan en "sol_actual," junto con su distancia en "dist_actual."

A continuación, se lleva a cabo una búsqueda local en la solución inicial, con el objetivo de mejorarla mediante modificaciones en la secuencia y el cálculo de la distancia resultante. Las mejores soluciones locales se registran en un mapa llamado "mejores_soluciones." La mejor solución local se selecciona en función de su distancia.

Si la solución local actual es superior a la mejor solución global, se actualizan "best_sol" y "best_dist." Luego, se registra el tiempo de finalización de la iteración actual y se calcula la duración de la ejecución.

El bucle continúa su ejecución hasta que se alcance el límite de tiempo establecido. Finalmente, la función devuelve la mejor distancia encontrada “best_dist” y el tiempo total de ejecución “duration.count()”.