PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

ALGORITMOS AVANZADOS

Segundo Examen
(Primer Semestre 2025)

Duración: 2h 50 min.

- **No puede utilizar apuntes, solo hojas sueltas en blanco.**
- En cada función el alumno deberá incluir, a modo de comentario, la forma de solución que utiliza para resolver el problema. De no incluirse dicho comentario, el alumno perderá el derecho a reclamo en esa pregunta.
- No puede emplear plantillas o funciones no vistas en los cursos de programación de la especialidad.
- Los programas deben ser desarrollados en el lenguaje C++. Si la implementación es diferente a la estrategia indicada o no la incluye, la pregunta no será corregida.
- Un programa que no muestre resultados coherentes y/o útiles será corregido sobre el 50% del puntaje asignado a dicha pregunta.
- Debe utilizar comentarios para explicar la lógica seguida en el programa elaborado. El orden será parte de la evaluación.
- Se utilizarán herramientas para la detección de plagios, por tal motivo si se encuentran soluciones similares, se anulará la evaluación a todos los implicados y se procederá con las medidas disciplinarias dispuestas por la FCI.
- **Solo está permitido acceder a la plataforma de PAIDEIA, cualquier tipo de navegación, búsqueda o uso de herramientas de comunicación se considera plagio por tal motivo se anulará la evaluación y se procederá con las medidas disciplinarias dispuestas por la FCI.**
- Para esta evaluación solo se permite el uso de las librerías **iostream, iomanip, climits cmath, fstream, vector, algorithm, string o cstring**
- Su trabajo deberá ser subido a PAIDEIA.
- **Es obligatorio usar como compilador NetBeans.**
- Los archivos deben llevar como nombre su código de la siguiente forma codigo_EX2_P# (donde # representa el número de la pregunta a resolver)

Pregunta 1 (10 puntos)

El **Exam Time Tabling Problem (ETT)** consiste en asignar un conjunto de **exámenes** a **intervalos de tiempo específicos** (slots) y, en algunas variantes, a **ubicaciones físicas (aulas)**, **recursos (supervisores)**, etc., **de modo que se respeten todas las restricciones duras** (conflictos, capacidad, disponibilidad) y **se optimice algún criterio relacionado con la calidad del calendario**, típicamente expresado como la minimización de incomodidades o solapamientos para los estudiantes.

Datos de entrada:

1. **Conjunto de exámenes**
   - Lista de exámenes:

     `E = {e1, e2, e3, ..., en}`
2. **Conjunto de intervalos de tiempo disponibles**
   - Por ejemplo, bloques horarios en los que se pueden programar exámenes:

     `T = {t1, t2, t3, ..., tk}`

3. **Conjunto de estudiantes**
    - Lista de estudiantes registrados en el sistema:
      `S = {s1, s2, s3, ..., sm}`
4. **Matriz de inscripción (relación estudiante–examen)**
    - Es una matriz binaria `R` de tamaño m x n, donde:
        - `R[i][j] = 1` si el estudiante `si` está inscrito en el examen `ej`
        - `R[i][j] = 0` en caso contrario

Use los siguientes datos de entrada:

- **Número de exámenes**: 5
- **Número de estudiantes**: 4
- **Número de slots disponibles**: 4
- Por lo tanto, **se requieren 2 bits por examen** para la codificación binaria.

Estudiante	e0	e1	e2	e3	e4
s0	1	1	0	0	0
s1	0	1	1	0	0
s2	0	0	1	1	0
s3	1	0	0	1	1

A partir de esta matriz se infiere la **matriz de conflictos** conflicto[i][j], donde conflicto[i][j] = 1 si al menos un estudiante está inscrito en ambos exámenes i y j:

	e0	e1	e2	e3	e4
e0	0	1	0	1	1
e1	1	0	1	0	0
e2	0	1	0	1	0
e3	1	0	1	0	1
e4	1	0	0	1	0

Dentro de las restricciones tenemos:

Asignación única: Cada examen debe ser asignado a exactamente un intervalo de tiempo.

No solapamiento de exámenes conflictivos: Si dos exámenes comparten al menos un estudiante, no pueden ser programados al mismo tiempo.

Se pide:

a) (2 puntos) Desarrollar un algoritmo GRASP construcción que reciba los datos de entrada mencionados y arroje la mejor planificación de horarios a franjas horarias, considerando cada franja horaria de 3 horas. Use un valor de $\alpha = 0.25$
b) (8 puntos) Desarrollar un algoritmo genético basado en cromosomas de 0-1 (**obligatoriamente**) que tengan dentro de su primera población diferentes soluciones GRASP construcción basadas en el algoritmo desarrollado en la parte a.

Nota 1: usar las siguientes variables para el algoritmo GRASP Construcción

Nombre	Tipo	Descripción
Estudiantes	Entero	Número de estudiantes
examenes	Lista	Lista de todos los exámenes. Ejemplo: ["e1", "e2", ..., "en"]
slots_disponibles	Lista de enteros	Lista de los slots disponibles para asignar. Ejemplo: [0, 1, 2, ..., k-1]
R	Matriz m x n de enteros	Matriz de inscripción. R[i][j] = 1 si el estudiante i rinde el examen j, 0 en caso contrario
conflictos	Lista	Para cada examen ej, lista de otros exámenes con los que tiene conflicto (comparten al menos un estudiante)
asignacion	Lista	Asignación parcial/progresiva de exámenes a slots durante la construcción
examenes_pendientes	Lista	Exámenes que aún no han sido asignados
RCL (Restricted Candidate List)	Lista	Lista de exámenes candidatos a asignar en cada iteración, seleccionados de forma greedy-randomizada

Nota 2: para el caso del algoritmo genético, Cada individuo de la población representa una solución candidata al problema: es decir, una asignación de slots a exámenes. Como estamos usando codificación binaria (0–1), el cromosoma de cada individuo será una cadena binaria que codifica para cada examen el slot al que fue asignado, en forma binaria.

Considerar:

- Hay E exámenes.
- Hay S slots disponibles
- Cada examen debe ser asignado a un **único slot** en el rango [0, S-1].
- Para representar cada slot, se necesitan $\lceil \log_2(S) \rceil$ bits por examen.

Ejemplo**:**

- Si hay S = 8 slots disponibles $\Rightarrow \log_2(8) = 3$ bits por examen.

- Si hay E = 10 exámenes ⇒ el cromosoma tendrá 3 × 10 = 30 bits.

Supongamos:

- 5 exámenes: e1, e2, e3, e4, e5
- 4 slots disponibles: S = {0, 1, 2, 3}
- Por tanto, se necesitan **2 bits por examen**

e1 e2 e3 e4 e5

01 | 10 | 00 | 11 | 01

Luego, si el objetivo es minimizar la cantidad de **conflictos entre exámenes**. Un conflicto ocurre cuando **dos exámenes que comparten al menos un estudiante** están asignados al **mismo slot**. La función de fitness debe ser f(I)=número total de conflictos.

Implemente solo los operadores de casamiento y mutación:

- **Casamiento con tasa del 50% (la mitad de los bloques de cada padre)**

  Padre 1: 00 | 11 | 10 | 01

  Padre 2: 10 | 10 | 00 | 11

  Hijo : 00 | 10 | 00 | 11 ← combinación válida por bloques

- **Mutación, cambiando un solo bit de un solo bloque.**

  Original: 00 | 11 | 10 | 01

Mutado: 00 | 10 | 10 | 01 ← solo

Pregunta 2 (10 puntos)

Supongamos que tenemos 9 tareas y 3 agentes, y necesitamos asignar cada tarea a un agente de manera que se maximice la ganancia total, pero respetando un presupuesto máximo por agente. El costo y la ganancia de asignar la i-ésima tarea al j-ésimo agente están dados por las siguientes matrices:

Costos

	1	2	3	4	5	6	7	8	9
1	43	50	38	33	39	39	59	54	45
2	21	42	23	51	28	16	52	19	36
3	25	54	17	32	20	58	40	28	48

Ganancias

	1	2	3	4	5	6	7	8	9
1	30	23	59	9	38	30	51	16	38
2	16	12	43	47	37	5	30	19	49
3	16	29	19	41	25	48	40	29	48

Presupuesto por agente = 90. NTAREAS=9. NAGENTES=3.

Así, por ejemplo, el costo y la ganancia de asignar la tarea 1 al agente 4 son 33 y 9, respectivamente. Cade tarea deberá asignarse a un solo agente, pero está permitido asignar varias tareas al mismo agente. También es posible que queden tareas sin asignar por cuestiones de presupuesto – en este caso, 90 por agente.

Se pide implementar un algoritmo genético que maximice las ganancias posibles respetando las restricciones de presupuesto. Tendrán la elección de usar cromosomas binarios o enteros, los detalles de representación y operadores genéticos se dan continuación:

Cromosomas Enteros (LCROM=9): Cada cromosoma es una secuencia de 9 números enteros, donde el valor del gen *i* indica el agente asignado a la i-ésima tarea. Valores de 0 corresponden a tareas que no pudieron ser asignadas. Por ejemplo, el cromosoma [3 0 1 2 2 1 3 0 2] indica que la tarea 1 y 3 fueron asignadas a los agentes 3 y 1, respectivamente, y la tarea 1 no pudo ser asignada.

Cromosomas Binarios (LCROM=18): Los cromosomas binarios usan las representaciones binarias de lo números de agentes asignados a cada tarea - 2 bits por cada asignación. Por ejemplo, [3 0 1 2 2 1 3 0 2] es representado como [11 00 01 10 10 01 11 00 10] o [110001101001110010].

Aptitud/Fitness: La función de aptitud está definida como la ganancia total, calculada como la suma de las ganancias de cada asignación representada por el cromosoma según la matriz de ganancias. Si el cromosoma no cumple con las condiciones de presupuesto, se asignará un valor de aptitud igual a 0.

Inicialización (NPOP=16, ALPHA=0.3): Generar la población inicial, de tamaño NPOB, usando un algoritmo GRASP (fase Construcción) diseñado para maximizar la ganancia total mediante asignaciones sucesivas que usan la heurística de escoger la asignación disponible de tenga mejor (mayor) ratio valor/costo. Se deberá verificar siempre que cada agente cumpla las restricciones de presupuesto. Generar los RCL usando el parámetro de relajación ALPHA=3.

En caso no se tenga disponible la implementación de GRASP, usar poblaciones iniciales completamente aleatorias.

Selección (K=2, NPADRES=16): Seleccionar NPADRES usando el método de selección por torneo. La selección por torneo escoge aleatoriamente K individuos de la población y selecciona al más apto.

Cruce: Generar el mismo número de hijos que padres, emparejando sucesivamente todos los padres seleccionados en la etapa anterior de la siguiente manera: 1-2, 3-4, 5-6, etc. Realizar NPADRES/2 cruces en total. Usar los siguientes operadores:

- **Cruce Enteros**: Usar el operador de cruce *uniform crossover,* en el cual cada gen de los cromosomas hijos tiene la misma probabilidad (PCROSS=0.5) de haber sido copiado de cualquiera de los padres.

- **Cruce Binarios (CRATIO=0.5):** Usar el operador clásico *onepoint crossover*, el cual parte e intercambia los contenidos de los padres a partir de una posición generalmente seleccionada de manera aleatoria. En nuestro caso, la posición es igual a LCROM*CRATIO.

Mutación (PMUT=0.3) Mutar, con una probabilidad de mutación PMUT, los hijos generados por la etapa de cruce. Usar los siguientes métodos:

- Mutación Enteros: Escoger aleatoriamente una posición del cromosoma, y reemplazar el valor con un nuevo valor (distinto) aleatorio entre 0 y NAGENTES.

- Mutación Binarios (PMUT=0.3): Elegir aleatoriamente y mutar (flip) PMUT*LCROM bits del cromosoma.

Reemplazo: Seleccionar los NPOP mejores cromosomas del conjunto compuesto por los cromosomas de la generación actual y los hijos – mutados y no mutados - generados por la etapa de cruce. Eliminar repetidos antes de realizar el corte. Para esto último, se recomienda compactar los cromosomas a strings (por ejemplo, [3 0 1 2 2 1 3 0 2] se convierte en "301221302") y usar el container map<string,Tl>.

Terminar: Realizar **NGEN**=80 generaciones. Imprimir la mejor ganancia de cada nueva generación par, y el cromosoma más apto luego de la última generación.

Pregunta 3 (10 puntos)

En una empresa dedicada al corte de barras de metal desea optimizar su trabajo minimizando el desperdicio al cortar las barras completas. Se sabe que recibe un pedido de **N** barras que debe cortar, cada una con un tamaño que es inferior a la longitud de una barra completa. También se conoce que la empresa tiene **NBARRAS** completas en su almacén. A continuación, un ejemplo:

Si la empresa tiene **NBARRAS**=9 cada una con una longitud de 10 metros y recibe un pedido de **N**=8 barras con las siguientes dimensiones **{5, 2, 3, 10, 3, 7, 2, 10}**

Debe minimizar el desperdicio tratando de atender el pedido empleando las barras ya cortadas y evitando emplear barras completas nuevas ya que generarían muchas barras incompletas que no se pueden vender.

Una asignación óptima sería la siguiente: {5, 5, 5, 4, 1, 1, 3, 8} con un desperdicio de 8 metros y solo dejo una barra incompleta.

Otra asignación sería la siguiente: {3, 1, 3, 5, 6, 1, 3, 4} aunque tiene un desperdicio total de 8 metros deja 2 barras incompletas lo que perjudica a la empresa.

Desarrolle un algoritmo genético basado en números enteros (no binario) que resuelva el problema propuesto. A continuación, algunas consideraciones:

- Su población inicial debe ser un algoritmo GRASP de no realizarse así se le consideran 2.0 puntos menos.
- El GRASP debe tener un alfa = 0.3 con 10,000 iteraciones
- La población inicial debe tener 20 individuos
- La función objetivo debe ser definida de forma clara considerando como prioridad minimizar la cantidad de barras incompletas y en segundo lugar la suma del desperdició de los cortes.
- Considere 1,000 iteraciones para el algoritmo genético
- Puede realizar inversión y/o mutación

- La forma de diseñar los operadores genéticos y las tasas a emplearse quedan a libertad del alumno

Al finalizar el examen, comprima la carpeta de su proyecto empleando el programa Zip que viene por defecto en el Windows, **no se aceptarán los trabajos compactados con otros programas como RAR, WinRAR, 7zip o similares**. Luego súbalo a la tarea programa en Paideia para este examen.

Profesores del curso:

Manuel Tupia
Rony Cueva
Igor Siveroni

San Miguel, 15 de julio del 2025

30 de noviembre, madrugada

Fin, 1:22 am

Fin, 1:23 am