i

Pontificia Universidad Católica de Chile
Departamento de Ciencia de la Computación
IIC1001 - Algoritmos y Sistemas Computacionales

Pauta Examen

1º semestre 2024 - Profesor Cristian Ruz

Pregunta 1

Archivo en bytes (notación hexadecimal) que cumple las características pedidas:

```
61 62 41 42 0a 4f 30 39 50 fd
50 3c a1 39 ff 3f ff ff ff ff
```

Explicación:

- `61 62`: Primeras dos letras del nombre en minúscula (ej. 'ab') (varían según el nombre)
- `41 42`: Primera letra de cada apellido en mayúscula (ej. 'AB') (varían según los apellidos)
- `0a`: Carácter de cambio de línea (LF)
- Sala puede ser:
  - `4f 30 39`: Caracteres de la sala O09.
  - `42 31 33`: Caracteres de la sala B13,
  - `42 32 33`: Caracteres de la sala B23,
- `50 fd`: Valor `0x50FD`
- `50 3c`: Valor `0x503C`
- `a1 39`: Suma `0x50FD` + `0x503C` = `0xA139`
- `ff 3f`: Diferencia `0x503C` − `0x50FD` = $(-193)_{10}$ = −`0xC1` = `0xFF3F` en complemento a 2 de 16 bits
- Se rellena con `0xFF` hasta 10 bytes por línea

Criterios. Para cada requisito de bytes:

- Todo el puntaje si está correcto
- Medio puntaje si hay un byte con error
- Sin puntaje si hay dos o más byte con error

Para el último ítem, 1 punto por cumplir cada uno de lo siguiente (0 si no cumple):

- Todo escrito en bytes hexadecimales. Se acepta si usan `0x` delante de cada byte
- Máximo 10 byte por línea. El byte `0a` **NO** provoca un cambio de línea.
- Rellenar con los `ff` al final.

Pregunta 2

2.1) [**4p**] Tabla de verdad del circuito:

S1	S0	Salida
0	0	I0
0	1	I1
1	0	I2
1	1	I3

Criterios: Un punto por cada línea correcta.

2.2) [**3p**] El circuito es un multiplexor o selector de 4 a 1. S1 y S0 seleccionan cuál entrada I0, I1, I2 ó I3, se propaga a la salida.

Criterios: No es necesario usar la palabra multiplexor, pero sí debe indicar que S1 y S0 permiten seleccionar o escoger entre una de las entradas. No se trata de explicar la tabla de verdad con palabras.

- 3p. Explicación clara y precisa. Indica que se selecciona una entrada.
- 2p. Error leve.
- 1p. Explica tabla de verdad con palabras (ej: si entra 0 y 1, sale I1, ... etc)

2.3) [**4p**] El Program Counter (PC) indica la dirección de la próxima instrucción a ejecutar. Se actualiza al final de cada ciclo de instrucción o por instrucciones de control de flujo.

Criterios: No es correcto decir que PC se actualiza en cada tick, si bien está conectado al *clock*.

- 2p por decir que indica la próxima instrucción (pueden decir "la instrucción actual").
- 2p por indicar cuándo se actualiza. Al menos decir que después de cada instrucción.

2.4) [**4p**] La memoria de instrucciones almacena el programa. La unidad de control interpreta las instrucciones y genera señales para coordinar la ejecución.

- 2p por indicar correctamente lo que hace la memoria de instrucciones. 1p error leve.
- 2p por indicar que la unidad de control interpretar instrucciones, o que contiene una traducción de instrucción a señales, o que genera señales para los otros componentes.

Pregunta 3

3.1) [3p] Operaciones Stack:

- Inicio: (vacío)
- push(A[2]): Pangui
- push(A[4]): Ale / Pangui
- push(A[6]): Cabra / Ale / Pangui
- pop(): Ale / Pangui
- push(A[9]): Nacho / Ale / Pangui
- pop(): Ale / Pangui
- pop(): Pangui
- push(A[0]): Gabi / Pangui
- pop(): Pangui
- Final: Pangui

Criterios: Basta con indicar el estado final.

- 3p. Estado final correcto.
- 2p. Estado final incorrecto. 1 error en el desarrollo.
- 1p. Estado final incorrecto. 2 errores en el desarrollo.

3.2) [3p] Operaciones Queue:

- Inicio: (vacío)
- enqueue(A[2]): | Pangui ← **Cabeza** |
- enqueue(A[4]): | Pangui ← **Cabeza** | Ale |
- enqueue(A[6]): | Pangui ← **Cabeza** | Ale | Cabra |
- dequeue(): | Ale ← **Cabeza** | Cabra |
- enqueue(A[9]): | Ale ← **Cabeza** | Cabra | Nacho |
- dequeue(): | Cabra ← **Cabeza** | Nacho |
- dequeue(): | Nacho ← **Cabeza** |
- enqueue(A[0]): | Nacho ← **Cabeza** | Gabi |
- dequeue(): | Gabi ← **Cabeza** |
- Final: | Gabi ← **Cabeza** |

Criterios: Basta con indicar el estado final.

- 3p. Estado final correcto.
- 2p. Estado final incorrecto. 1 error en el desarrollo.
- 1p. Estado final incorrecto. 2 errores en el desarrollo.

3.3) **[4p]** Tabla de hash resultante: Los *hash* correspondientes a cada elemento son:

String	String[0]	ASCII Decimal	Hash
`Gabi`	`G`	71	7
`Su`	`S`	83	3
`Pangui`	`P`	80	0
`Feña`	`F`	70	0
`Ale`	`A`	65	5
`Jorge Pérez`	`J`	74	4
`Cabra`	`C`	67	7
`Bugedo`	`B`	66	6
`Don Yadran`	`D`	68	8
`Nacho`	`N`	78	8
`CRuz`	`C`	67	7

El *hash* es un arreglo de 10 posiciones, con índices desde 0 hasta 9.

Índice	Elementos
0	→ `Pangui` → `Feña`
1	→ `Gabi`
2	→
3	→ `Su`
4	→ `Jorge Pérez`
5	→ `Ale`
6	→ `Bugedo`
7	→ `Cabra` → `CRuz`
8	→ `Don Yadran` → `Nacho`
9	→

Criterios: Basta con indicar el estado final.

- 2p por asignar correctamente las posiciones de hash, sin considerar colisiones.
- 2p por manejar correctamente las colisiones. Deben estar en el orden de ingreso.

3.4) [**4p**] Para contar eficientemente las repeticiones de palabras, usar una Tabla de Hash:

- Clave: palabra única
- Valor: contador de ocurrencias

Complejidad de búsqueda y actualización: O(1) en promedio.

Criterios:

- 1p por escoger correctamente la estructura tabla de hash.
- 1p si la descripción permite que la estructura mantenga la información
- 1p si la estructura permite mantener la información en $O(1)$
- 1p si calcula correctamente la complejidad

Pregunta 4

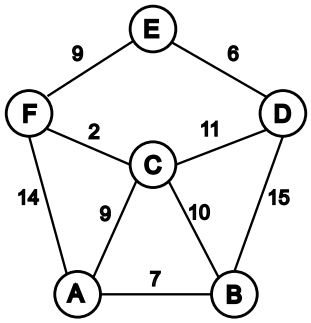
4.1) [**4p**] Recorrido DFS desde nodo E. Hay 20 recorridos válidos.

- E, D, B, A, C, F
- E, D, B, A, F, C
- E, D, B, C, A, F
- E, D, B, C, F, A
- E, D, C, A, B, F
- E, D, C, A, F, B
- E, D, C, B, A, F
- E, D, C, B, F, A
- E, D, C, F, A, B
- E, D, C, F, B, A
- E, F, A, B, C, D
- E, F, A, B, D, C
- E, F, A, C, B, D
- E, F, A, C, D, B
- E, F, C, A, B, D
- E, F, C, A, D, B
- E, F, C, B, A, D
- E, F, C, B, D, A
- E, F, C, D, A, B
- E, F, C, D, B, A

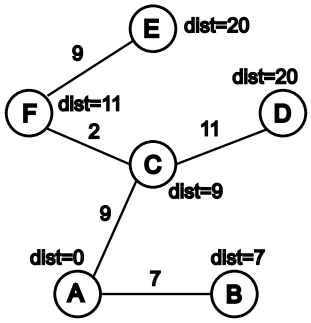
Criterios: Basta uno de los mencionados.

- 4p si es uno de los de la lista
- 2p si tiene diferencia de 1 con alguno de la lista
- 0p cualquier otro caso

4.2) [6p] Rutas más cortas desde A usando Dijkstra. Grafo original:



Pueden presentar el grafo:



o mostrar las tablas:

Nodo	dist
A	0
B	7
C	9
D	~~22~~ 20
E	20
F	~~14~~ 11

y

Nodo	dist
A	-
B	A
C	A
D	~~B~~ C
E	F
F	~~A~~ C

Las rutas y costos obtenidos:

- A a B: A $\rightarrow$ B, costo 7
- A a C: A $\rightarrow$ C, costo 9
- A a D: A $\rightarrow$ C $\rightarrow$ D, costo 20
- A a E: A $\rightarrow$ C $\rightarrow$ F $\rightarrow$ E, costo 20
- A a F: A $\rightarrow$ C $\rightarrow$ F, costo 11

Criterios:

Rutas (2p)

- 2p si todas las rutas más cortas están correctas
- 1p si todas las rutas más cortas están correctas, menos una
- 0p si hay más de una incorrecta

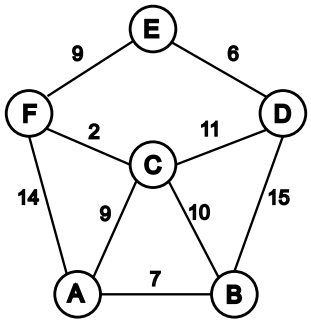
Costos (2p)

- 2p si todos los costos de las rutás más cortas están correctos
- 1p si todos los costos de las rutas más cortas están correctos, menos uno
- 0p si hay más de uno incorrecta

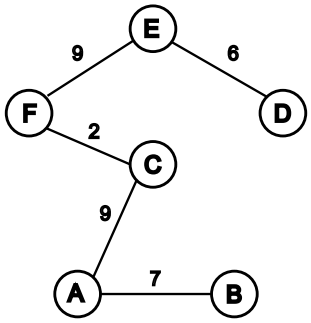
Adicionalmente (2p)

- 2p si muestra el proceso correctamente
- 1p si hay un error en el proceso
- 0p si hay más de un error en el proceso

4.3) [6p] Grafo original:



Grafo G' sin ciclos, conectado y sacando las aristas más grandes:



G' no preserva todas las rutas más cortas del grafo original. Por ejemplo, en G' la ruta más corta de A a D es A $\rightarrow$ C $\rightarrow$ F $\rightarrow$ E $\rightarrow$ D, con costo 26. En cambio, en el grafo G, la ruta más corta obtenido a partir de Dijkstra es A $\rightarrow$ C $\rightarrow$ D, con costo 20.

Este enfoque es una estrategia codiciosa, la cual no siempre garantiza obtener el resultado óptimo.

Criterios:

Grafo (3p):

- 3p por obtener el grafo correctamente
- 1p por una arista equivocada
- 0p por dos o más aristas equivocadas

Rutas (3p):

- 3p por justificar que no se obtienen las rutas más cortas. Puede ejemplificarlo con alguna de las obtenidas en la pregunta anterior.
- 1p por justificación con un error leve
- 0p por justificación con un error importante, o más de un error.