Primer parcial – Gobstones Positioning System

Introducción a la Programación – Algoritmos y Programación
UNQ – 1er semestre de 2013

22 de abril de 2013

El presente documento contiene las directivas y el enunciado del primer parcial. El incumplimiento de las directivas puede significar la reprobación del examen. Por ello, **debe leer este documento con atención**. Cualquier duda respecto de las directivas o el enunciado, puede consultarla a la lista de docentes.

1. Directivas para la resolución del parcial

La presentes directivas deben ser tenidas en cuenta para la resolución del parcial.

- El parcial está disponible en la página de la materia desde el **20 de abril**.
- El enunciado se presenta en la clase práctica del **20 de abril** (AyP) o **22 de abril** (InPr).
- La resolución debe realizarse en grupos de dos personas. En caso de imposibilidad, hágale saber sus motivos a los docentes **antes del 24 de abril**, enviando un mail con subject **alumno individual para parcial**. Los docentes determinarán quiénes están habilitados a resolver el parcial individualmente.
- Los grupos deben informarse antes del **24 de abril**, enviando un mail con subject **grupo parcial** a la lista de docentes.
- Aquellos alumnos que no logren formar un grupo, **DEBEN** avisar que resolverán el parcial, enviando un mail con subject **alumno sin grupo para parcial** a la lista de docentes. Estos alumnos serán asignados a un compañero por los docentes.
- Aquellos alumnos que no informen su grupo o su voluntad de resolver el parcial, quedarán excluidos de la materia, con nota **ausente**.
- La resolución debe entregarse en la clase práctica del **4 de mayo** (AyP) o **2 de mayo** (InPr).
- Debe entregarse un folio que contenga la resolución en papel y el código fuente en soporte digital (cd-rom, dvd-rom, etc).
- Los docentes aceptarán **una única** copia del parcial. En caso de tener inconvenientes grupales, ambos integrantes **deben** comunicarse con los docentes lo antes posible.
- Las hojas deben:
  - estar numeradas,
  - tener escrito el nombre de los autores del trabajo,
  - estar correctamente abrochadas y sujetadas al folio.
- El soporte digital debe:
  - contener todo el código fuente en un archivo `parcial.gbs`, salvo por el código fuente que se incluya en `Biblioteca.gbs`.
  - tener una única copia de los archivos entregados,

    - tener escrito el nombre de los autores del trabajo,
    - estar correctamente sujetado al folio.
- El código fuente entregado en el medio digital debe:
    - tener vacío el procedimiento `Main`,
    - funcionar correctamente en PyGobstones versión 0.97b,
    - corresponder con la resolución en papel,
    - estar correctamente tabulado y comentado.

  Cada procedimiento y función del código fuente debe:
    - contener su propósito, precondición y el tipo de cada parámetro,
    - explicitar el tipo y el propósito de cada variable.
- Los docentes podrán requerir que el código fuente sea entregado por e-mail.
- Las consultas por e-mail acerca del parcial deben dirigirse **exclusivamente** a la lista de docentes. La lista de alumnos puede ser usada **únicamente** para la formación de grupos.

Se reitera que el incumplimiento de las presentes directivas puede acarrear la reprobación del parcial.

2. Evaluación del parcial

A la hora de evaluar los parciales domiciliarios, los docentes revisarán no solo la corrección del código, sino también:

- la correcta división en subtareas,
- la correcta reutilización de los distintos procedimientos y de los ejercicios prácticos,
- la elección de los nombres de variables, procedimientos y funciones,
- el buen uso de los esquemas de recorridos,
- la inclusión de propósitos y precondiciones en cada ejercicio,
- la inclusión de comentarios donde sea pertinente,
- el correcto uso de los distintos conceptos explicados en la materia, y
- todos los temas que hayan sido explicados en la materia.

Se aclara que la corrección del código es condición necesaria pero **no suficiente** para la aprobación del examen.

3. Consultas y clases

Durante las semanas que dura la resolución del parcial, las clases teóricas y prácticas continuarán normalmente. Sin embargo, se espera que durante las clases prácticas los alumnos muestren sus avances del parcial y consulten sus dudas a los docentes. Durante las clases prácticas, los docentes responderán consultas de **todos** los temas, incluyendo y no limitado a:

- enunciado,
- resolución de un ejercicio,
- tabulación y formas de comentar,
- forma de dividir en subtareas,

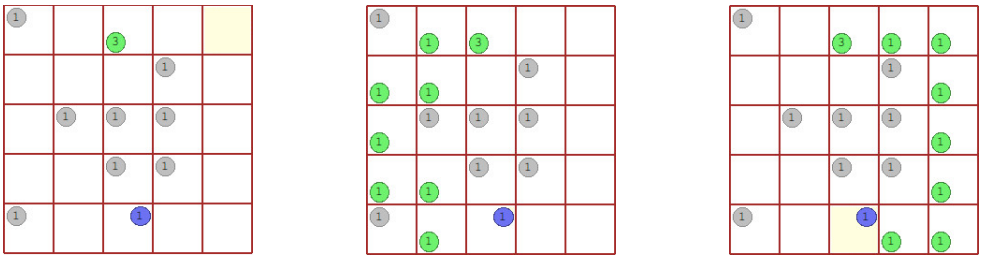
- fallos en la ejecución del código,

Se avisa, sin embargo, que la respuesta del docente dependerá de los aportes que hagan los alumnos y **en ningún caso** se proporcionará la solución a un ejercicio. También puede enviar sus consultas a la lista de docentes, pero **no hay garantía** de que sean respondidas.

4. Enunciado

En este parcial vamos a modelar un sistema de navegación llamado Gobstones Positioning System (GPS). Dado un mapa, el objetivo de GPS es determinar el camino más corto que tiene que hacer un móvil para llegar a algún destino, informando los posibles cambios de dirección que tiene que hacer el móvil.

En este parcial modelamos el mapa usando el tablero de Gobstones. Las celdas del tablero se dividen entre aquellas intransitables, llamadas *obstáculos*, y aquellas transitables, llamadas *libres*. En el tablero las celdas obstáculo son aquellas que contienen bolitas negras, mientras que las celdas libres son aquellas que no contienen bolitas negras (ver Figura 1 (a)). Un *camino* es una secuencia de celdas libres que están conectadas a través de movimientos hacia el Norte, Este, Sur u Oeste, donde ninguna celda es transitada dos veces (ver Figura 1 (b)). Cada camino tiene un *origen* y un *destino*, que son justamente las celdas inicial y final de la secuencia. La *longitud* del camino es la cantidad de celdas que contiene, y un camino entre un origen y un destino es *mínimo* cuando tiene la mínima cantidad de celdas. Para representar la celda de destino usamos un única bolita azul, mientras que para representar al móvil usamos tres bolitas verdes (ver Figura 1 (a)). Asimismo, vamos a usar bolitas rojas para representar ciertas marcas temporales. El objetivo de GPS es, entonces, encontrar el camino mínimo (y las indicaciones necesarias) para ir desde la celda del móvil hasta la celda del destino (ver Figura 1 (b) y (c)).



(a) (b) (c)

Figura 1: (a) ejemplos de mapa representado en Gobstones: las celdas con bolitas negras son obstáculos, la celda con una bolita azul es el destino y la celda con tres bolitas verdes es el móvil. (b) y (c) ejemplos de caminos desde el móvil (origen) al destino formado por bolitas verdes; ambos tienen longitud 8 y son mínimos.

El problema de encontrar el camino mínimo podría no tener solución. Por ejemplo, si las celdas lindantes al móvil son todas obstáculo, entonces el móvil nunca podrá llegar a destino. Como precondición para todos los ejercicios, vamos a suponer, sin explicitarlo, que el tablero de Gobstones representa un mapa *válido*, en el cual:

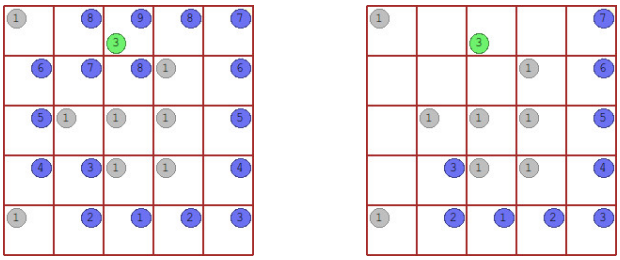
- hay una única celda de destino, i.e., hay una única celda que contiene exactamente una bolita azul,
- hay una única celda de móvil, i.e., hay una única celda que contiene exactamente tres bolitas verdes,
- las celdas obstáculo sólo tienen bolitas negras; notar que, consecuentemente, no es posible que la celdas del móvil y destino sean obstáculo,
- hay al menos un camino de cualquier celda libre a la celda de destino.

Los tres tableros de la Figura 1 son válidos; notar que pueden haber más bolitas además de las del móvil, el destino y los obstáculos. El tablero puede ser modificado durante la ejecución de los distintos procedimientos. Sin embargo, las condiciones anteriores **deben** ser satisfechas por los tableros que generan

los distintos procedimientos. Por ejemplo, el procedimiento `DibujarCamino` (Ejercicio 8) exige dibujar el camino desde el móvil al destino agregando bolitas verdes. En este caso, las celdas del recorrido deben modificarse sin alterar la existencia de la única celda de móvil.

4.1. El mapa de distancias

Como parte de este parcial, necesitamos calcular la distancia de cada celda libre hacia el destino. La *distancia* desde una celda libre al destino es la longitud del camino mínimo que va desde dicha celda al destino. Por ejemplo, en los mapas de la Figura 1, la distancia de la celda del móvil al destino es 8. Para poder calcular estas distancias, es necesario primero generar el mapa de distancias. En cada celda libre del tablero, el *mapa de distancias* tiene $d+1$ bolitas azules cuando la distancia a la celda de destino es d (ver Figura 2 (a)). El mapa de distancias se va calculando de a poco en los Ejercicios 2–6. En el proceso de construcción del mapa, algunas celdas libres contendrán bolitas azules y otras no (ver Figura 2 (b)). Lo que diferencia a las celdas con bolitas azules del resto es que de las primeras conocemos un camino al destino de distancia `nroBolitas(Azul) - 1` (que puede no ser el más corto), mientras que de las celdas restantes no sabemos cómo llegar al destino.



(a) (b)

Figura 2: (a) Mapa de distancias del tablero mostrado en la Figura 1 (a): la celda de destino tiene 1 bolita azul porque esta a distancia 0 del destino, mientras que la celda del móvil tiene 9 bolitas azules porque está a distancia 8 del destino. (b) Construcción intermedia del mapa de distancias; se conocen algunos caminos, pero no todos.

Ejercicio 1

Escribir las siguientes funciones:

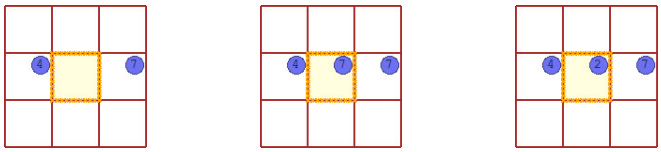
- `colorMapa()` que denota el color `Azul`.
- `colorObstaculo()` que denota el color `Negro`.
- `colorMarca()` que denota el color `Rojo`.
- `colorNavegacion()` que denota el color `Verde`.
- `esObstaculo()` que denota `True` si la celda actual es obstáculo. En otras palabras, denota `True` si la celda actual contiene bolitas de color `colorObstaculo()`.
- `esDestino()` que denota `True` si la celda actual es la celda destino. En otras palabras, denota `True` cuando la celda actual contiene exactamente una bolita de color `colorMapa()`.
- `alcanzaDestinoEnMapa()` que denota `True` si a partir del mapa de distancias conocemos un camino de la celda actual a la celda destino. En otras palabras, denota `True` si la celda actual contiene bolitas de color `colorMapa()`.
- `distanciaEnMapa()` que denota la distancia de la celda actual a la celda destino de acuerdo al mapa de distancias. En otras palabras, denota el número de bolitas de color `colorMapa()` que hay en la celda actual menos 1. Notar que si `alcanzaDestinoEnMapa()` denota `False`, entonces la distancia no está bien definida más alla de que quizá `distanciaEnMapa()` no haga `BOOM`. En consecuencia, la precondición de esta función es que `alcanzaDestinoEnMapa()` denote `True`.

Fin del Ejercicio 1

Como dijimos, el mapa de distancias se calcula con un proceso repetitivo que consiste en *mejorar* cada celda mientras sea posible. Para poder definir este proceso repetitivo, primero definimos qué significa que una celda sea *mejorable*. Decimos que una celda libre c es *mejorable* por la celda libre x lindante en dirección `d` cuando conocemos un camino desde x a la celda destino en el mapa de distancias y

- no se conoce ningún camino desde c a la celda destino en el mapa, o
- la distancia de c al destino en el mapa es mayor que la de x por al menos dos unidades.

La Figura 3 muestra ejemplos de celdas mejorables y no mejorables.



(a) (b) (c)

Figura 3: (a) La celda actual es mejorable por la celdas al este y al oeste, ya que para ambas se conocen caminos al destino y no para la celda actual. En cambio, no es mejorable por las celdas al norte y al sur porque no se conocen caminos de dichas celdas al destino. (b) La celda actual es mejorable por la celda al oeste ya que su distancia al destino es 4 que es menor a la distancia de la celda actual - 1 (i.e., 6). En cambio, no es mejorable por la celda al este. (c) La celda actual no es mejorable por ninguna de sus lindantes.

Ejercicio 2

Escribir la función `esMejorablePor(d)` que denote `True` si la celda actual es mejorable por la celda lindante en dirección `d`. Debe suponer que la celda actual es libre, pero no puede suponer que la celda lindante en dirección `d` exista o sea libre.

Fin del Ejercicio 2

Supongamos que una celda libre c es mejorable por alguna de sus lindantes. Más aún, supongamos que x es la celda lindante a c que está a menor distancia del destino, de acuerdo al mapa de distancias (en la Figura 3 (a), x sería la celda al oeste). *Mejorar* la celda c significa cambiar su distancia en el mapa (poniendo o sacando bolitas de mapa) de forma tal que la nueva distancia sea 1 más que la distancia de x al objetivo (si se mejora la celda actual en la Figura 3 (a), la cantidad de bolitas de color `colorMapa()` pasarían a ser 5). Observar que después de mejorar, c ya no puede mejorarse, a no ser que cambie la distancia de alguna celda lindante.

Ejercicio 3

Escribir el procedimiento `Mejorar()` que mejore la celda actual. Si la celda actual no es mejorable por ninguna celda lindante, entonces el procedimiento no hace nada. Debe suponer que la celda actual es libre.

Fin del Ejercicio 3

Ejercicio 4

Escribir el procedimiento `MejorarTodas()` que mejore todas las celdas libres del tablero una vez utilizando algún esquema de recorrido.

Fin del Ejercicio 4

Ejercicio 5

Escribir la función `hayMejorable()` que denote `True` si alguna celda del tablero es mejorable por alguna de sus celdas lindantes.

Fin del Ejercicio 5

Como mencionamos anteriormente, el mapa de distancias se calcula mejorando cada celda hasta que no queden más celdas mejorables. La única salvedad es que para mejorar necesitamos que el tablero

esté *limpio*, en el sentido que la única celda con bolitas de color `colorMapa()` sea el destino (por ejemplo, los tableros de la Figura 1 están limpios, mientras que los de la Figura 2 no). Caso contrario, las bolitas de color `colorMapa()` podrían confundir el proceso.

Ejercicio 6

Escribir el procedimiento `CalcularMapaDistancias()` que calcule el mapa de distancias. Para ello, deben mejorarse todas las celdas del tablero hasta que no queden celdas mejorables. Notar que el tablero inicial debe estar limpio.

Fin del Ejercicio 6

4.2. Distancias y recorridos

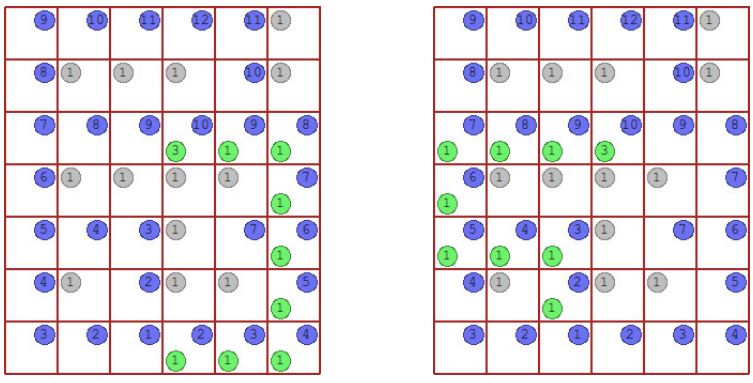
Ahora que ya sabemos calcular el mapa de distancias, podemos determinar la distancia de cualquier celda hacia el destino. En efecto, la distancia de una celda al destino es la distancia de dicha celda en el mapa de distancias. Como para poder calcular el mapa de distancias necesitamos que el tablero esté limpio (i.e., contenga una única celda con bolitas de color azul), para los ejercicios que siguen vamos a suponer que el tablero está limpio sin aclararlo explícitamente. Más aún, para calcular la distancia va a ser necesario marcar algunas celdas del tablero con bolitas de `colorMarca()`. Por ello, para los ejercicios siguientes vamos a suponer también que ninguna celda del tablero está marcada (aunque las celdas pueden marcarse antes de invocar las funciones de los Ejercicios 1–6).

Ejercicio 7

Escribir la función `distancia()` que denote la distancia de la celda actual al destino. Sugerencia: marcar la celda actual con bolitas de color `colorMarca()` antes de calcular el mapa de distancias, de forma tal de no perder la posición de la celda actual. Debe suponer que la celda actual es libre.

Fin del Ejercicio 7

La función distancia nos permite calcular el camino más corto desde la celda actual al destino. Supongamos que la celda actual está a distancia d del destino. Entonces, uno de los caminos mínimos puede obtenerse moviéndose primero a una celda libre a distancia $d-1$ del destino, luego yendo a una celda libre a distancia $d-2$ del destino, y así siguiendo hasta llegar al destino, que es la celda a distancia 0 del destino (ver Figura 4).



(a) (b)

Figura 4: Ejemplos de mapas de distancias superpuestos con caminos mínimos de la celda del móvil al destino. Notar que cada celda del camino tiene una distancia menor al destino que la celda anterior.

Ejercicio 8

Escribir el procedimiento `DibujarCamino()` que deje **exactamente** una bolita de color `colorNavegacion()` en cada celda de algún camino mínimo que vaya de la celda actual al destino, incluyendo la celda actual pero no la celda de destino. Debe suponer que la celda actual no es obstáculo. **Nota**: podrían haber celdas con bolitas de color `colorNavegacion()` en el tablero inicial, incluyendo aquellas del camino.

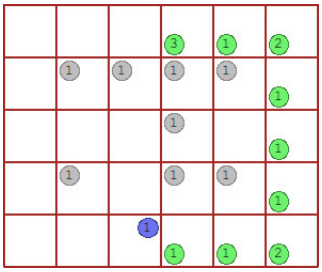


Figura 5: Tablero con las indicaciones para ir desde el móvil al destino. Las curvas tienen 2 bolitas verdes, y las no curvas tienen 1 sóla.

Fin del Ejercicio 8

Una curva en un camino es una celda en la que el camino cambia de sentido (ver Figura 5). Para que el sistema de navegación pueda dar las indicaciones, es importante marcar las curvas del camino.

Ejercicio 9

Escribir el procedimiento `DibujarCurvas()` que deje **exactamente** dos bolitas de color `colorNavegacion()` en cada curva del camino mínimo que se calcula en `DibujarCamino()`. Debe suponer que la celda actual no es obstáculo. **Nota**: podrían haber celdas con bolitas de color `colorNavegacion()` en el tablero inicial, incluyendo las curvas del camino.

Fin del Ejercicio 9

4.3. Indicaciones

El último procedimiento es el que dibuja el camino mínimo desde la celda del móvil hasta el destino, indicando también las curvas del camino.

Ejercicio 10

Escribir el procedimiento `Indicaciones()` que deje exactamente una bolita de color `colorNavegacion()` en cada celda no curva del camino mínimo, y dos bolitas de color `colorNavegacion()` en cada curva del camino mínimo. **Nota:** no debe modificarse la cantidad de bolitas de la celda del móvil.

Fin del Ejercicio 10