Informe 3 Algoritmos y Estructuras de Datos III

Jazmín Alvazer Vico75/15jazminalvarezvico@gmail.com Marcelo Pedraza393/14marcelopedraza314@gmail.co
Uriel Jonathan Rozenberg838/12rozenberguriel@gmail.com Javier María Cortés Conde Tití252/15javiercortescondetito@gmail.com

Grupo
Alvarez Vico Jazmín
Cortés Conde Titó Javier María
Pedraza Marcelo
Rozenberg Uriel Jonathan

6 de noviembre de 2016

# Índice

# 1. Introducción

En nuestro problema Brian quiere convertirse en ”maestro pokemon.en el menor tiempo posible. Para lograr este objetivo debe ir a todos los ”gimnasiosz conquistarlos. Para poder hacerlo, cada gimnasio requiere una cantidad determinada de pociones. Estas pociones pueden obtenerse en las ”pokeparadas”. Las pokeparadas solo pueden visitarse una vez y de cada visita obtenemos tres pociones.

Formalmente, podemos caracterizar nuestras pokeparadas y gimnasios como nodos formando un grafo completo, es decir que existen aristas para unir cualquier par de nodo. nuestras aristas deben tener peso, que equivalga a la distancia entre dos nodos. entonces queremos encontrar el camino mínimo que une todo los nodos gimnasios y los nodos pokeparada que hagan falta para poder conquistar todos los gimnasios.

# 2. Algoritmo exacto

## 2.1. Exlicación de la solución

Para modelar este problema, creamos dos clases: Nodo y Mochila. La clase Mochila contiene dos enteros “*capacidad*” el cual corresponde con la capacidad de la entrada (k) y “*peso*” en el cual se guarda la cantidad de pociones que se tiene en cada momento. en el Pseudocogigo se llaman a las funciones DameCapacidad y DamePeso, las mismas devuelven estos valores.

La clase Nodo tiene un entero “*indice*” como identificador de cada nodoy un entero “*pociones*” en el cual se guardan las pociones que dan las pokeparadas o las que requieren los gimnasios(en este caso el valor es negativo). Ademas cuenta con un booleano para marcar si esta o no recorrido y dos enteros “$x$” e “$y$” para identificar las coordenadas del nodo.

Nuestra solución es utiliza la técnica de Backtracking para resolver el problema. Utilizamos variables globales (las cuales estan mencionadas en el pseudocodigo) ya que las utilizamos todo el tiempo en las funciones de nuestra resolución. De esta forma evitamos pasarlas cada vez como parametros de entrada.

Nuestro algoritmo de BT va formando recursivamente todas las soluciones posibles eligiendo gimnasios y pokeparas siempre que esto sea posible. El ciclo principal de la función va desde 0 hasta el Maximo entre el largo de los vectores de Nodos y pokeparadas. De esta forma vamos probando empezar con todos los nodos posibles. Cada vez que se elige un nodo para continuar se vuelve a llamar a la función BT para seguir construyendo la solución, luego se revierten los cambios para probar las demás soluciones.

En nuestro algoritmo, cortamos una posible solución en el momento que supera la solución minima encontrada hasta el momento. De esta forma evitamos seguir ejecutando el algoritmo para una solución que claramente no va a ser la mejor, y así reducimos el tiempo de cómputo.

Mientras hacemos la lectura de datos, guardamos la cantidad máxima de pociones que requieren los gimnasios y el total de pociones que se requieren para ganar a todos los gimnasios. De esta forma si la capacidad de la mochila es menor al máximo de las pociones que piden los gimnasios o la cantidad de pociones que se pueden obtener (cantidad de pokeparadas multiplicado por tres) es menor a lo que se necesita para ganar a todos los gimnasios, entonces devolvemos -1 (evitando entrar en el backtracking) ya que el algoritmo no tiene solución.

## 2.2. Pseudocodigo

**variables Globales**

```
1: real MinActual
2: real MinGlobal
3: vint RecorrifoGlobal
4: vint RecorridoActual
5: vnod PokeParadas
6: vnod Gimnasios
7: int PokeParadasRecorridas
8: int PocionesNecesarias
9: Mochila Moch
10: entero GimRecorridos
```

**BT**()

```
1: MinActual ← ∞
2: MinGlobal ← 0
3: Si MinActual>MinGlobal
4:    cortar
5: finSi
6: Si GimRecorridos = |Gimnasios|
7:    Si MinActual < Min Gobal
8:       MinGlobal ← MinActual
9:       RecorridoGlobal ← RecorridoAtual
10:   finSi
11: finSi
12: Desde i=0 hasta i<Max(|Gimnasios|,|PokeParadas|)
13:      Si i< |Gimnasios|
14:         gim ← Gimnasios[i]
15:         Si puedoIr(gim)
16:            Voy(gim)
17:      finSi
18:   finSi
19:   Si i < |PokeParadas|
20:   pp ← Pokeparadas[i]
21:   PokeParadasRecorridas ← PokeParadasRecorridas + 1
22:      Si puedoIr(pp)
23:      Voy(pp)
24:      finSi
25:   PokeParadasRecorridas ← PokeParadasRecorridas - 1
26:   finSi
27: finDesde
28: finSi
```

**Voy**(Nodo n)

1: Marco n
2: Si RecorridoActual $\neq \emptyset$
3: origen $\leftarrow$ ultimo de RecorridoActual
4: minActual $\leftarrow$ minActual+ Distancia(origen, p)
5: finSi
6: Agrego n a RecorridoActual
7: Modifico el peso de moch segun las pociones n
8: BT()
9: revierto todas las modificaciones

**puedoIrPP**(Nodo n)bool

1: NoConsumoDeMas $\leftarrow$ Si al entrar a esta pokeparada se tiene que descartar pokebolas que no nos dejan ganar a todos los gimnasios asigno True, si no False.
2: res $\leftarrow$ moch no esta llena y n no esta recorrido y NoConsumoDeMas

**PuedoIrGim**(Nodo n)

res $\leftarrow$ n no esta recorrido y DamePeso(moch) $\geq$ -(DamePosciones(n))

## 2.3. Complejidad

## 2.4. Experimentación y análisis de resultados

# 3. Heurística constructiva golosa

El metodo de heurística goloza consiste en elegir siempre la mejor opción posible para continuar y formas la solución total. En este caso el criterio para elegir una mejor opción seria la cercanía.

## 3.1. Exlicación de la solución

## 3.2. Pseudocodigo

**goloso**()

1: Mientras Gimnasios $\neq \emptyset$
2: proxLugar=GimMasCercano
3: Si no puedo ir al gimnasio
4: proxLugar=PokeParadaMasCercana
5: finSi
6: Si No puedo ir a ningun nodo
7: distanciaRecorrida=-1
8: Recorrido = $\emptyset$
9: finSi
10: moverse(proxLugar)
11: finMientras

**moverse**(Nodo n)

```
1: Modifico las mochilas según las pociones de n
2: Agregar atras de Recorrido a n
3: distanciaRecorrida + dist(n, posiciónActual)
4: Actualizar posición actual
5: Si n es gimnasio
6:    sacar a n de Gimnasios
7: Si No
8:    sacar a n de pokeParadas
9: finSi
```

### 3.3. Complejidad

### 3.4. Experimentación y análisis de resultados

## 4. Heurística de búsqueda local

### 4.1. Exlicación de la solución

### 4.2. Pseudocodigo

### 4.3. Complejidad

### 4.4. Experimentación y análisis de resultados

## 5. Metaheurística

Como metaheurística decidimos implementar un algoritmo "GRASP". La técnica de este tipo de algoritmos consiste en combinar sucecivamente un algoritmo golozo randomizado para explorar nuevos espacios de soluciones y luego aplicar una busqueda local para mejorar cada solución. Al mismo tiempo se van comparando las soluciones y siempre se va eligiendo la mejor posible hasta que el algoritmo finaliza bajo algún criterio.

### 5.1. Exlicación de la solución

Utilizamos como criterio de terminación el tamaño del conjunto de "entradas validas". Este conjunto esta formado por Todos los gimnasios que pidan cero pociones para ser vencidos y todas las pokeparadas. Aplicamos el algoritmo golozo comenzando de cada uno de los nodos de este conjunto.

El algoritmo golozo randomizado es una variación de nuestra heuristica goloza presentada en la sección (). Para explorar el nuevo espacio de soluciones seleccionamos como candidatos al veinte porciento de los nodos por recorrer más cercanos a la posición actual. Luego le asignamos una probabilidad a cada uno de ellos. La misma utiliza el siguiente criterio:

- Si el nodo es gimnasio y tengo suficientes pociones en mi mochila para ir, +30
- Si el nodo es pokeparada y entran las 3 pociones en la mochila, +25
- Si el nodo es pokeparada y entran 2 pociones en la mochila, +15

- Si el nodo es pokeparada y entra solo 1 pocion en la mochila, +10
- Se le suma a cada nodo bajo criterio de cercania 10*#MasCercanos - 10* i donde i representa la posición en ese orden.

Guardamos el valor de la sumatoria de los valores asignados a cada nodo y creamos un número aleatorio entre 1 y este valor. El nodo elegido será el nodo con valor mayor más proximo al número aleatorio. Estas acciones se realizan en la función que llamamos en el pseudocodigo "ElegirProximo".

Finalmente aplicamos la función de busqueda local de la sección ().

## 5.2. Pseudocodigo

**GRASP**(vnod entradasValidas)

```
1: solu = ∞
2: Desde i=0 hasta |entradasValidas|
3:    GolozoRand(entradasValidas[i])
4:    Blocas(RecorridoGlobal, MinGlobal)
5:    Si MinGlobal<solu
6:       solu=MinGlobal
7:       RecorridoSolu=RecorridoGlobal
8:    finSi
9: fin Desde
```

## 5.3. Complejidad

## 5.4. Experimentación y análisis de resultados