Práctica Sistema de Búsqueda

SMART PAPAYA WAREHOUSE 4.1

Autores:
Ángel Martín Ramos
Javier Matilla Martín

1. Prolog

Prolog es un lenguaje declarativo en el cual somos nosotros los que le decimos "cómo ha de resolver este nuestro problema", básicamente le indicamos una serie de reglas que el programa debe seguir para conseguir el objetivo, que en nuestro caso es el reparto equitativo de frutas en bandejas con la condición de que estas tengan un peso mínimo de 2 kg.

Inicialmente hemos declarado estos dominios:

```
domains

        /*Componentes de una pieza*/
        peso=real
        id=integer
        coste=real
        dias=real

        fruta=p(peso,id,dias)
        bandeja=fruta*

        bandejaFull=b(bandeja,peso,coste)
```

Donde peso, id, coste y días son los atributos de cada fruta, también declarada, los usaremos para decidir si una fruta va o no va en la bandeja.

Por otro lado, hemos declarado la bandeja, la cual es una lista de frutas donde iremos añadiendo frutas hasta conseguir los 2kg de peso mínimo.

Para finalizar hemos declarado la variable *"bandejaFull"*, donde tendremos la bandeja anteriormente rellenada, el peso total de la bandeja y el coste total de la misma.

En cuanto a los predicados:

```
predicates

        estaLlena(bandejaFull,bandeja)
        noLlena(peso)
        annadir(fruta, bandeja, bandejaFull) /*b([A|B],P,C)*/

        /*Escribe el contenido de las bandejas*/
        escribe(bandejaFull)
        escribeId(bandeja)
        backtrack(bandeja, bandeja, peso, coste)
  papayasfinal(bandeja)
```

En la parte de predicados tendremos las relaciones entre nuestros argumentos, entre ellas tendremos declaradas:

estaLlena(bandejaFull,bandeja)
noLlena(peso)
annadir(fruta, bandeja, bandejaFull)
escribe(bandejaFull)
escribeId(bandeja)
backtrack(bandeja, bandeja, peso, coste)
papayasfinal(bandeja)

La parte de clauses es la base de conocimientos donde se expresa el conocimiento en forma de reglas y aserciones. En nuestro caso tenemos:

escribeId : escribe por pantalla el id de forma recursiva para una lista de frutas dada

escribe : escribe por pantalla todos los parametros de una “bandejaFull”.

noLlena : comprueba si el peso es menor o igual a dos

estaLlena : comprueba si el peso es mayor que dos ,en caso afirmativo escribe la bandeja y llama a backtrack con una bandeja vacía

annadir : comprueba si no esta llena, en caso afirmativo calcula el peso nuevo y el coste nuevo, añadimos a nuestra bandeja la fruta y se llama a backtrack

backtrack : realiza una llamada a annadir pasandole la pieza de fruta ,la bandeja con la lista de piezas que lleva y la bandejaFull la cual contiene el peso y coste de la bandeja

papayasfinal : función a la que le pasamos el almacén y llama a backtrack con una bandeja nueva para iniciar la ejecución.

Imagen de las cláusulas:

```
clauses

        escribeId([]).

        escribeId([p(_,Id,_)|T]):-
                write('\t'),
                write(Id,'\n'),
                escribeId(T),
                write('\n').

        escribe(b(Bandeja,Pt,Ct)):-
                write('\t'),
                write(Pt,'\t'),
                write(Ct,'\t'),
                write('\n'),
                escribeId(Bandeja).

        noLlena(Pt):-
                Pt<=2.0.

        estaLlena(b([H|T],Pt,Ct),B):-
                Pt>2.0,
                escribe(b([H|T],Pt,Ct)),
                backtrack(B,[],0.0,0.3).

        annadir(p(PeF,Id,D), Band, b(Bandeja,Pt,Ct)):-
                /*No llena*/
                noLlena(Pt),

                /*Calculamos el peso nuevo*/
                Pnew=PeF+Pt,
                Ctnew=Ct+0.1+2.0*PeF+0.05*D,
                NuevaBandeja=[p(PeF,Id,D)|Bandeja],
                not(estaLlena(b(NuevaBandeja,Pnew,Ctnew),Band)),
                backtrack(Band,NuevaBandeja,Pnew,Ctnew).

        backtrack([H|T],B,Pt,Ct):-

                annadir(H,T,b(B,Pt,Ct)).

        papayasfinal(Almacen):-
                backtrack(Almacen,[],0.0,0.3).
```

En la parte de goal la fórmula que se pretende que sea cierta en nuestro caso es papayasfinal al cual le pasamos el almacén con todas las piezas que tenemos

```
goal

  papayasfinal([
p(0.273, 1400001, 1.1),
p(0.405, 1400002, 1.0),
p(0.517, 1400003, 1.1),
p(0.533, 1400004, 1.7),
p(0.358, 1400005, 1.5),
p(0.562, 1400006, 1.9),
p(0.322, 1400007, 2.4),
p(0.494, 1400008, 1.8),
p(0.39, 1400009, 1.6),
p(0.281, 1400010, 2.2),
p(0.395, 1400011, 2.0),
p(0.407, 1400012, 2.0),
p(0.329, 1400013, 3.0),
p(0.629, 1400014, 2.7),
p(0.417, 1400015, 1.2),
p(0.278, 1400016, 1.4),
p(0.583, 1400017, 2.2),
p(0.598, 1400018, 1.9),
p(0.271, 1400019, 1.6),
p(0.265, 1400020, 2.1)
]).
```

Output

```
        2.086     5.292
        1400005
        1400004
        1400003
        1400002
        1400001




        2.049     5.393
        1400010
        1400009
        1400008
        1400007
        1400006




        2.177     5.699
        1400015
        1400014
        1400013
        1400012
        1400011



no
```

En cuanto al output obtenemos 3 bandejas que cumplen con el requisito del peso mínimo pero observamos un detalle, es que el proceso de “llenado” de la bandeja se hace secuencialmente

2. Algoritmo A*

Para desarrollar el algoritmo A*, hemos considerado la siguiente funcion de evaluacion: **f(n)=g(n)/h(n) siendo g(n) el peso y h(n) el coste.**

```
"""coeficiente tiene que ser cuanto mayor sea el coste y menor el
precio
    si disminuye el coste, aumenta la función de ev
    si aumenta el peso, disminuye el coeficiente
    Peso/Coste              1 / 3 = 0.33 --> si aumento coste 1 / 4 =
0.25
                                         --> si aumento el peso 2 / 3 =
0.66"""
```

Básicamente llegamos a una conclusión, se nos pide en el enunciado encontrar bandejas de coste mínimo, con lo cual lo que nosotros queremos es aumentar al máximo nuestra función de evaluación por lo que cogeremos la fruta en base a este razonamiento de arriba

Este algoritmo de A* ha sido desarrollado en el lenguaje de programación python, para su correcto funcionamiento hemos precisado de crear dos clases, una para las bandejas y otra para las frutas, con sus métodos internos de funcionamiento, a continuación mostramos las clases:

En cuanto a la clase fruta, tenemos los métodos:

- **__init__** se ejecutará cuando se cree un objeto de la clase
- **muestraFruta()** nos muestra la fruta con un formato amigable
- **getters** nos proporcionarán el valor del atributo
- **setters** podremos indicarle el valor que queramos asignar al atributo

En cuanto a la clase bandeja, tenemos los métodos:

- **__init__** se ejecutará cuando se cree un objeto de la clase
- **insertaFruta()** añade la fruta pasada por parametro al parámetro lista, que será una lista de frutas (análogamente como hicimos en prolog)
- **getters** nos proporcionarán el valor del atributo
- **setters** podremos indicarle el valor que queramos asignar al atributo

Imagen con las clases fruta y bandeja

```
class fruta:

    def __init__(self,peso,id,dias):
        self.peso=peso
        self.id=id
        self.dias=dias

    def muestraFruta(self):
        return 'Peso -->'+str(self.peso)+'\tId -->'+str(self.id)+'\tDias -->'+str(self.dias)

    def getPeso(self): return self.peso
    def getId(self): return self.id
    def getDias(self): return self.dias
    def getCoste(self): return 0.1+2.0*self.getPeso()+0.05*self.getDias()

    def setPeso(self,peso): self.peso=peso
    def setId(self,id): self.id=id
    def setDias(self,dias): self.dias=dias

class bandeja:

    def __init__(self,pesotot,costetot):
        self.lista=[]
        self.pesotot=pesotot
        self.costetot=costetot

    def getLista(self): return self.lista
    def getPesotot(self): return self.pesotot
    def getCostetot(self): return self.costetot

    def insertaFruta(self,fruta):
        self.lista.append(fruta)

    def setLista(self,lista):
        for element in lista:
            self.lista.append(element)
    def setPesotot(self,pesotot): self.pesotot=pesotot
    def setCostetot(self,costetot): self.costetot=costetot
```

Imagen función calculaFn y obtieneFruta()

```
def calculaFn(lista):
    tmp=0.0
    valor=0.0
    for i in lista:
        tmp=i.getPeso()/i.getCoste()
        if(valor<tmp): valor=tmp
    return valor

def obtieneFruta(lista,valor):
    tmp=0.0
    for el in lista:
        tmp=el.getPeso()/el.getCoste()
        if(tmp==valor): return el
    return fruta(0.0,0.0,0.0)
```

En cuanto a estas dos funciones, la primera calcula el mejor valor para la f(n) y lo devuelve, mientras que la segunda, lo que hacemos es en base al valor de f(n) obtenido, localizamos la fruta poseedora de estas características y la devolvemos.

Lógica del programa

```
while(longitudAbiertos != longitudCerrados) :
                valor=calculaFn(abiertos)
                if(mi_bandeja.pesotot<=pesoMax):
                    elemento=obtieneFruta(abiertos,valor)
                    if(not(elemento in cerrados) & (elemento in abiertos)):
                        abiertos.remove(elemento)
                        cerrados.append(elemento)
                        longitudCerrados = len(cerrados)
                        pesoTot=pesoTot+elemento.getPeso()
                        costeTot=costeTot+elemento.getCoste()

                        mi_bandeja.setPesotot(pesoTot)
                        mi_bandeja.setCostetot(costeTot)
                        mi_bandeja.insertaFruta(elemento)
                if(mi_bandeja.getPesotot()>pesoMax):
                        listaBandejas.append(mi_bandeja)
                        n+=1
                        mi_bandeja = bandeja(0.0,0.0)
                        pesoTot=0.0
                        costeTot=0.3
                longitudCerrados = len(cerrados)
```

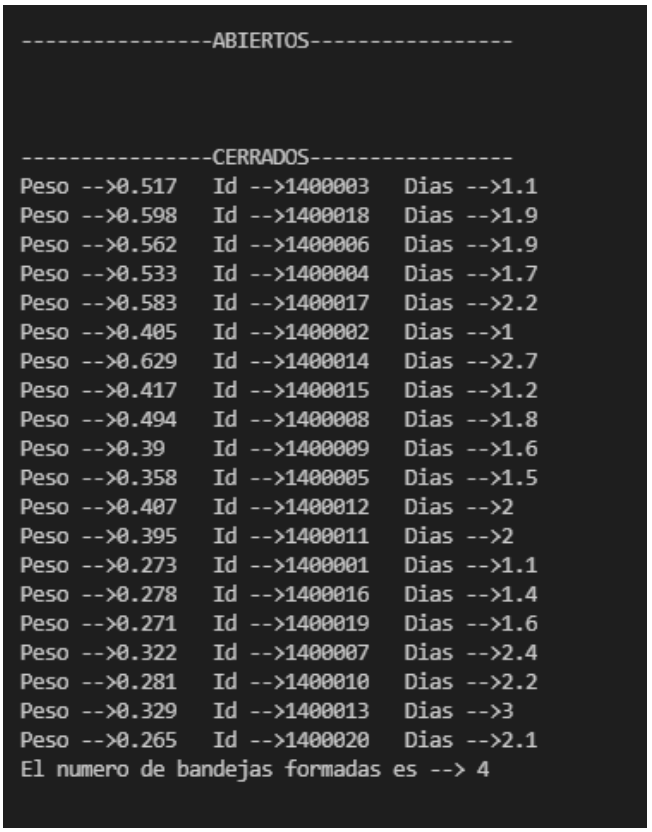
En esta parte del programa es donde vamos recorriendo toda la lista del almacén y asignando las bandejas. Nuestra condición de salida del bucle es que la longitud de elementos de la lista abiertos antes de ser tratada sea igual que la longitud de cerrados, en el caso de que esto se dé, significa que todas las frutas han sido consumidas y por tanto, una finalización exitosa de nuestro algoritmo.

Cuando un elemento es candidato para la bandeja, comprobamos que este elemento no esté en cerrados y si en abiertos, en caso afirmativo, pasamos a eliminar el elemento de la lista de abiertos y añadirlo a la de cerrados, calculamos de nuevo el pesoTotal y costeTotal de la bandeja e insertamos la fruta.

En caso de que esta llegue a los 2kg mínimos, añadimos la bandeja a una lista denominada *“listaBandejas”* , seteamos la bandeja a los valores iniciales y los valores de pesoTot y costeTot a 0.

Vamos actualizando al final del programa continuamente el valor de la longitud de cerrados.

Conjunto de abiertos y cerrados al final de la ejecución



Resultados bandeja 1:

```
Datos de la bandeja  1
El peso total es --> 2.21
El coste total es --> 5.45
La bandeja 1 contiene --> 4 frutas

Peso -->0.517   Id -->1400003   Dias -->1.1
Peso -->0.598   Id -->1400018   Dias -->1.9
Peso -->0.562   Id -->1400006   Dias -->1.9
Peso -->0.533   Id -->1400004   Dias -->1.7
```

Resultado bandeja 2:

```
 Datos de la bandeja  2
 El peso total es --> 2.034
 El coste total es --> 5.123
 La bandeja 2 contiene --> 4 frutas

 Peso -->0.583   Id -->1400017   Dias -->2.2
 Peso -->0.405   Id -->1400002   Dias -->1
 Peso -->0.629   Id -->1400014   Dias -->2.7
 Peso -->0.417   Id -->1400015   Dias -->1.2
```

Resultado bandeja 3:

```
Datos de la bandeja  3
El peso total es --> 2.044
El coste total es --> 5.333
La bandeja 3 contiene --> 5 frutas

Peso -->0.494   Id -->1400008   Dias -->1.8
Peso -->0.39    Id -->1400009   Dias -->1.6
Peso -->0.358   Id -->1400005   Dias -->1.5
Peso -->0.407   Id -->1400012   Dias -->2
Peso -->0.395   Id -->1400011   Dias -->2
```

Resultado bandeja 4:

```
 Datos de la bandeja  4
 El peso total es --> 2.019
 El coste total es --> 5.728000000000001
 La bandeja 4 contiene --> 7 frutas

 Peso -->0.273   Id -->1400001   Dias -->1.1
 Peso -->0.278   Id -->1400016   Dias -->1.4
 Peso -->0.271   Id -->1400019   Dias -->1.6
 Peso -->0.322   Id -->1400007   Dias -->2.4
 Peso -->0.281   Id -->1400010   Dias -->2.2
 Peso -->0.329   Id -->1400013   Dias -->3
 Peso -->0.265   Id -->1400020   Dias -->2.1
```

Diferencia entre Algoritmo A* y Prolog

El algoritmo utilizado para prolog no consideraba la posibilidad de que “sobraran” piezas, de ahí que en la salida salga un “no” al final, en cambio el A* está optimizado para que esta condición no suceda.

Otra diferencia notable es que el procedimiento de prolog es “añadir secuencialmente”, no se basa en ningún tipo de heurística a diferencia que el A*, el cual usa una función de evaluación con el objetivo de poder determinar qué fruta es la “mejor” en cada caso,

¿Qué problemas de índole práctico presenta la producción de bandejas obtenida con el Algoritmo A*?

El principal objetivo del algoritmo A* es que finalice correctamente ajustándose a la función de evaluación, ahí es donde reside la mayor complicación del algoritmo. Si se usa una función de evaluación “mala”, el resultado no será el óptimo e incluso que sobre alguna pieza. En nuestro caso, se hace un reparto equitativo de frutas por bandeja y consume todas las entradas como se muestra en la imagen de los conjuntos de abiertos y cerrados. Este sería el principal problema, definir una buena función de evaluación.