BRANCH & BOUND

ANÁLISIS Y DISEÑO DE ALGORITMOS.

Práctica Final

Memoria

Pedro Giménez Aldeguer, 15419933C

INDICE

1. Estructuras de datos

```
priority_queue<vector<bool>> q;
q.push(inicial);

while(!q.empty()) {
  vector <bool> vecCola = q.top();
  q.pop();
```

1.1 Nodo

He utilizado un vector de tipo booleano como nodo, indicando true, las ciudades que tienen gasolinera y false, las que no. La posición en la que se encuentra el valor corresponde con la ciudad a la que hace referencia. Además, para sacar el coste, he tenido que utilizar un método que me transforma el vector de booleanos a un vector de enteros, ya que mi método "coste" trabaja con vector de enteros.

```
vector <int> Bool_Int (vector<bool> inicial) {
  vector <int> vec;
  for (size_t i=0; i<inicial.size(); i++) {
    if (inicial[i]){
      vec.push_back (i);
    }
  }
  return vec;
}
```

1.2 Lista de nodos vivos

Uso una cola de prioridad donde se van almacenando los nodos vivos, aquellos con posibilidades de ser expandidos, pero no he podido utilizar esta propiedad ya que no he sabido como hacerlo a pesar de haberme leído la teoría y haber ido a clase.

2. Mecanismos de poda

```
if(cantTrue(inicial) < g){
  vectorPesimista = vec_pesimista (inicial, g, c);
  pesimista = coste (c, Bool_Int(vectorPesimista), v);
  if(pesimista < current_best){
    emp = Bool_Int(vectorPesimista);
    current_best = pesimista;
  }

  vectorOptimista = vec_optimista (inicial, g, c, v);
  optimista = coste (c, Bool_Int(vectorOptimista), v);
  if(optimista < current_best){
    q.push(inicial);
  }
}
```

2.1 Poda de tuplas no factibles

Los nodos no factibles los descarto al no crearlos, no permito crear nodos con mayores gasolineras de las que debemos colocar. Este tipo de poda es muy necesaria, es inútil e ineficiente calcular estos costes, ya que no podrían ser una solución factible.

2.2 Poda de tuplas no prometedoras

No he sabido extraer el nodo más prometedor. Trata de no seguir por las ramas de los nodos que sabes que van a tener peor coste, se pasa por los nodos en los que vas a conseguir mejores resultados.

3. Cotas pesimistas y optimistas

```
vector<bool> vec_pesimista (vector<bool> inicial, size_t cantGasolineras, size_t cantCiudades) {
  size_t g = cantGasolineras;
  vector <bool> vec = inicial;
  size_t posInicial = 0;
  size_t auxG;

  if (inicial.size()-1 >= 0){
    posInicial = inicial.size()-1;
  }

  for (size_t i=0; i<inicial.size(); i++){
    if (inicial[i]){
      g--;
    }
  }

  auxG = g;

  for (size_t i=posInicial; auxG>0; i++){
    vec.push_back (true);
    auxG--;
  }

  posInicial += g;

  for (size_t i=posInicial; i<cantCiudades-1; i++){
    vec.push_back (false);
  }
  return vec;
}
```

```
vector<size_t> vec_posicionesG (size_t g, vector<int> &v, vector<bool> inicial) {
  vector <size_t> vecPos, respuesta;

  while(vecPos.size() < v.size()){
    size_t mayor = 0;
    bool esta = false;
    size_t pos = 0;

    for(size_t i=0; i < v.size(); i++){
      esta = false;

      for(size_t j=0; j < vecPos.size(); j++){
        if(i == vecPos[j]){
          esta = true;
        }
      }

      if(esta == false){
        if(v[i] > (int)mayor){
          mayor = v[i];
          pos = i;
        }
      }
    }

    vecPos.push_back(pos);
  }
```

```
vector<bool> vec_optimista (vector<bool> inicial, size_t g, size_t cantCiudades, vector<int> &veh ) {
  vector <bool> vec = inicial;
  vector <size_t> pos;

  size_t posInicial = 0;

  if (inicial.size()>0){
    posInicial = inicial.size();
  }

  for (size_t i=0; i<inicial.size(); i++) {
    if (inicial[i]){
      g--;
    }
  }

  for (size_t i=posInicial; i<cantCiudades; i++){
    vec.push_back (false);
  }

  pos = vec_posicionesG(g, veh, inicial);

  for (size_t i=0 ; i<pos.size(); i++){
    vec[pos[i]] = true;
  }
  return vec;
}
```

3.1 Cota pesimista y optimista del nodo inicial

La cota pesimista del nodo inicial, calculando el coste a partir del vector pesimista, rellenado a true las n primeras posiciones.

La cota optimista del nodo inicial, calculando el coste a partir del vector optimista, rellenado a true las n posiciones de cuyas ciudades tengan más vehículos.

(n como el número de gasolineras)

3.2 Cota pesimista y optimista de los demás nodos

He utilizado el mismo método anterior para los demás nodos, pero teniendo en cuenta que no debo modificar las posiciones de los vectores que he extraído de la cola y los tomo como base.

El vector se rellena con todas las posiciones que faltan hasta que el tamaño de este sea igual a la cantidad de ciudades con el mismo método explicado anteriormente. Hay que tener en cuenta que la cantidad de gasolineras que debo de poner son n menos las que ya haya en el vector que tomo como base.

(n como el número de gasolineras)

4. Otros medios empleados para acelerar la búsqueda

No he empleado ningún otro mecanismo para acelerar la búsqueda.

NO IMPLEMENTADO

5. Estudio comparativos de distintas estrategias de búsqueda

No he podido hacer el estudio comparativo.

NO IMPLEMENTADO

6. Soluciones y tiempos de ejecución.

Fichero 01a-01g.p: Solución: 0; Tiempo de proceso: 0.070005 ms

```
./mca_bb -f 01a-01g.p
Bb: 0
Emplacements: 0
CPU time (ms): 0.070005
Iterations of loop while: 1
```

Fichero 07a-03g.p: Solución: 207.5; Tiempo de proceso: 0.263072 ms

```
./mca_bb -f 07a-03g.p
Bb: 207.5
Emplacements: 3 4 5
CPU time (ms): 0.263072
Iterations of loop while: 65
```

Fichero 10a-02g.p: Solución: 11485; Tiempo de proceso: 0.258831 ms

```
./mca_bb -f 10a-02g.p
Bb: 11485
Emplacements: 3 7
CPU time (ms): 0.258831
Iterations of loop while: 79
```

Fichero 10a-05g.p: Solución: 3506.75; Tiempo de proceso: 0.624112 ms

```
./mca_bb -f 10a-05g.p
Bb: 3506.75
Emplacements: 1 2 3 4 7
CPU time (ms): 0.624112
Iterations of loop while: 180
```

Fichero 20a-10g.p: Solución: 6225.16; Tiempo de proceso: 513.303 ms.

```
./mca_bb -f 20a-10g.p
Bb: 6225.16
Emplacements: 4 5 6 7 9 13 14 15 17 19
CPU time (ms): 513.303
Iterations of loop while: 53801
```

Fichero 30a-12g.p: Solución: 6675.52; Tiempo de proceso: 225533 ms.

```
./mca_bb -f 30a-12g.p
Bb: 6675.52
Emplacements: 5 6 10 13 14 15 16 17 19 20 21 24
CPU time (ms): 225533
Iterations of loop while: 9631335
```

Fichero 30a-25g.p: Solución: 758.106; Tiempo de proceso: 315.218 ms.

```
g++ -Wall -O3 -std=c++0x -o mca_bb mca_bb.cc
./mca_bb -f 30a-25g.p
Bb: 758.106
Emplacements: 1 3 4 5 6 7 9 10 11 12 13 14 15 16 17 18 19 20 21 22 24 25 27 28 29
CPU time (ms): 315.218
Iterations of loop while: 10287
```

Fichero 40a-10g.p: Solución: ¿?

```
./mca_bb -f 40a-10g.p
ERROR: can't open file: 40a-10g.p.
Usage:
```

Fichero 40a-30g.p: Solución: ¿?

```
mca_greedy -f file./mca_bb -f 40a-30g.p
ERROR: can't open file: 40a-30g.p.
Usage:
```