Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

Practica 2

Servicio de mesajería

Algoritmia Para Problemas Dificiles

Aarón Ibáñez Espés, 779088
Fernando Gomez Osta, 774762

Índice

1. Resumen **2**

2. Desarrollo **2**

2.1. Descripción de la implementación 2

2.2. Pruebas realizadas 3

2.3. Reparto del trabajo 5

3. Anexo 1 **6**

1. Resumen

Este documento es una memoria de la práctica 2 de la asignatura de algoritmia para problemas difíciles.

El ella se plantea un problema en el cual se quiere elegir el servicio de mensajería disponible con menor tiempo de reparto posible de los dos disponibles en la ciudad. Ambos servicios tienen el mismo coste y su estrategia de reparto es la misma, cuando un conductor llega a una intersección, este elige aleatoriamente por que carretera seguir y si llega a la intersección en la que vives te entrega el paquete. Por ello se ha implementado un algoritmo probabilista que simula el comportamiento de un conductor para determinar cual es el tiempo medio de reparto y se han diseñado un conjunto completo de pruebas en el que se evalúa el algoritmo y sus resultados.

Para la implementación de la practica se ha decidido utilizar C++ ya que al ser un lenguaje compilado su eficiencia es mayor.

2. Desarrollo

2.1. Descripción de la implementación

Para el desarrollo de la practica se han implementado tres clases las cuales representan una carretera, una intersección y una ultima clase que representa el grafo de conexiones entre intersecciones.

- **Carretera:** Esta clase como su nombre indica representa una carretera. En ella se almacenan cinco valores relevantes, los identificadores de las intersecciones que une (u y v), el tiempo de recorrer la carretera (t_{uv}) y la probabilidad de ir de u a v (p_{uv}) y viceversa (p_{vu}). Además, esta clase cuenta con todos los métodos getters para cada uno de sus valores privados y una función utilizada para debugar el código (ver Figura 1).

- **Intersección:** Esta clase representa una intersección. En ella se almacena el identificador de la intersección, el numero de carreteras que contiene y un vector que apunta a las mismas. En esta clase se han implementado los métodos getters y setters, ya que al ser una intersección un objeto que se construye mientras se lee el fichero son necesarios dichos métodos. Además, se ha implementado una función *printInterseccion*, la cual muestra toda la información sobre la intersección y que se ha utilizado para el debug del programa. (ver Figura 2)

- **Mapa:** Esta clase es la clase principal utilizada para resolver el problema. En ella se almacenan el numero de intersecciones, el numero de carreteras, y las intersecciones donde se encuentran la vivienda, el almacén de la empresa A y el almacén de la empresa B, además de un vector que apunta a todas las intersecciones que puede haber en la ruta (ver Figura 3).

  Esta clase cuenta con un constructor en el cual se procesa el fichero de entrada para convertirlo en un grafo que la función camino pueda procesar de forma mas sencilla. Se ha tomado la decisión de representar la entrada en forma de grafo, en el cual cada intersección es un vértice y cada carretera representa una arista, ya que se ha considerado que era una manera sencilla e intuitiva de representar un camino en un mapa.

  Para ello en el constructor inicialmente se inicializa el vector de intersecciones y se crea un vector auxiliar de carreteras *carreteras*. Una vez se han creado estos se procesan las carreteras y se almacenan en *carreteras* para posteriormente, iterar sobre el vector de carreteras y terminar con un vector de intersecciones en el cual cada una de ellas apunte a las carreteras que la forman (ver Figura 4).

Para resolver la ruta desde cada uno de los almacenes a la vivienda se ha implementado la función *camino(string camino)* en la clase *Mapa*, la cual tiene un parámetro en el que se le debe indicar desde que almacén se quiere trazar la ruta. El funcionamiento de la misma se detalla a continuación.

- Inicialmente se establece la intersección donde se encuentra el almacén como inicio de la ruta.

- A partir de ahí se itera sobre las intersecciones evaluando en que intersección se encuentra el repartidor y eligiendo aleatoriamente por cual de las carreteras conectadas a esa intersección

debe continuar, hasta alcanzar la intersección donde se encuentra la vivienda o determinar que la vivienda es inalcanzable.

Para la elección aleatoria de carretera, se ha utilizado un generador de números reales aleatorios y se ha decidido por cual de ellas debe continuar en función de la probabilidad de cada una de ellas.

En el Anexo 1 se dispone de una captura del código (ver Figura 5).

2.2. Pruebas realizadas

Para la ejecución de las pruebas se ha optado por implementar en el *main* de la practica la evaluación de cada entrada, es decir, en el main se declara una clase de tipo *Mapa* y a partir de la cual se obtienen las soluciones. Para la correcta evaluación de cada uno de los ficheros que se han creado para las pruebas se han realizado 500 iteraciones del mismo y se ha calculado la media de tiempo en llegar desde cada uno de los almacenes a la vivienda, para posteriormente evaluar cual de los dos sistemas de mensajería es mejor elegir. Para calcular la media se ha contado el numero de veces en las cuales el repartidor encuentra la vivienda para cada caso y se ha dividido el total de la suma de todos los recorridos que han encontrado solución entre el numero de veces que se ha conseguido encontrar solución. El hecho de calcular la media, se debe a que en un algoritmo probabilista existe el caso de que la solución sea muy mala, en nuestro problema puede darse el caso de que el repartidor se pierda en un bucle y no consiga salir, lo que le llevaría a pasar por un numero de carreteras mucho mayor al óptimo, sin embargo, esto no pasa en todos los casos, solamente en los peores, por ello se calcula la media de varias iteraciones. Además, se dispone de un scritpt *pruebas.sh* el cual ejecuta los ficheros de prueba ubicados en *pruebasEntrega* automáticamente (ver Figuras 6 y 7).

Código disponible en el Anexo 1.

Los ficheros de prueba que se han generado abarcan los siguientes tipos.

- Pruebas creadas de forma manual. Estas pruebas son casos de prueba mas específicos y de menor tamaño pero de utilidad. Entre ellos se encuentran ficheros de prueba con con bucles entre carreteras lo que puede generar que el repartidor pese a ser un mapa pequeño este todo el rato dando vueltas en circulo y no encuentre solución o encuentre una solución de tamaño muy elevado. Mapas en los que la vivienda es inalcanzable desde ambos almacenes y mapas en los que la vivienda es inalcanzable desde el almacén de la empresa A y otros desde la B.

- También se ha creado un fichero *generadorPruebas*, en el que se han generado mapas de diferentes tamaños y densidades de forma automática. Estas pruebas son completamente aleatorias, desde las uniones que realizan las carreteras hasta las probabilidades de las mismas, siempre cumpliendo las restricciones especificadas, por lo que pueden darse casos en los que pese a que la solución sea alcanzable las probabilidades de que se llegue a ella sean demasiado bajas como para que en alguna de las 500 repeticiones de la búsqueda del camino que se hacen se encuentre.

  Cabe destacar que los tiempos de los ficheros generados aleatoriamente excepto de *pruebaTiempo4.txt* y *pruebaTiempo1.txt* se han generado con tiempo de recorrer la carretera 1, ya que el tiempo de recorrido no se ha considerado tan relevante como las probabilidades asociadas a las carreteras. Esta decisión se ha tomado, en parte, debido a que a la hora de observar los resultados medios de tiempo facilita el saber cuantas iteraciones del algoritmo se han realizado.

Para hacer que el algoritmo siempre devuelva solución y no se quede en un bucle infinito en caso de no encontrar un camino a la vivienda, se han establecido un numero de iteraciones máximo en función del tamaño de la entrada. Se han considerado ficheros de tamaño pequeño a aquellos de tamaño ≤ 50, a los que se les ha asignado un numero máximo de 2000 iteraciones, es decir, que se recorran como máximo 2000 carreteras y en caso de no encontrar solución devolver inalcanzable en esa iteración, y ficheros grandes a los de tamaño > 50, a los que se ha limitado a un total de 10000 iteraciones, es decir, recorrer un numero máximo de 10000 carreteras.

Fichero	Tamaño	Camino A	Camino B	Tiempo(ms)
inalcanzableAyB.txt	5	inalcanzable	inalcanzable	5036
inalcanzableA.txt	10	inalcanzable	2	2908
inalcanzableB.txt	10	10	inalcanzable	2829
bucle1.txt	10	inalcanzable	inalcanzable	5050
recorridoDens1_1.txt	10	5	1	0
recorridoDens1_2.txt	10	17	21	0
recorridoDens0.75_10.txt	10	1	15	0
pruebaTiempo4.txt	40	9438	10863	1046
recorridoDens0.75_50.txt	50	150	536	620
recorridoDens0.75_100.txt	100	189	154	86
recorridoDens1_3.txt	100	4589	4747	23486
pruebaTiempo1.txt	500	21875	21367	3044
aleatoria1.txt	40	98	100	29
aleatoria2.txt	80	80	97	22
aleatoria3.txt	160	1244	1170	2734
aleatoria4.txt	320	341	328	483
aleatoria5.txt	480	1237	1222	2708
aleatoria6.txt	640	4569	4767	24352

Cuadro 1: Pruebas realizadas

En el cuadro de la parte superior se pueden ver todos los resultados obtenidos para todos los casos de prueba que se han considerado relevantes.

- **inalcanzableAyB.txt:** Este fichero es una de las pruebas generadas de forma manual, en el se ha creado un grafo para el cual no existe camino desde ninguno de los almacenes hasta la intersección donde se debe entregar el paquete. Como el tamaño del grafo es 5 (tamaño pequeño), el numero máximo de carreteras recorridas antes de detener la ejecución es de 2000. Esto se ve reflejado en el tiempo total de ejecución, donde pese a ser un grafo de un tamaño muy reducido el tiempo de ejecución es muy elevado.
- **inalcanzableA.txt e inalcanzableB.txt:** Estos son otros de los dos ficheros generados de forma manual, en ellos la intersección de entrega es inalcanzable desde el almacén de la empresa A y desde el de la empresa B respectivamente. Con respecto al tiempo de ejecución se puede observar como se encuentra ligeramente por encima de la mitad del tiempo del fichero anterior, esto es debido a que para uno de los almacenes se esta llegando al limite de iteraciones en todo momento, sin embargo, para el otro se encuentra camino en un tiempo bastante reducido.
- **recorridoDens1_1.txt, recorridoDens1_2.txt y recorridoDens0.75_10.txt:** Estos ficheros son ficheros generados aleatoriamente con densidades altas y de tamaño reducido, por lo que se puede observar como el tamaño medio de la solución es en los 3 casos bastante reducido, siendo el caso del fichero **recorridoDens1_2.txt** el que mayor longitud de camino posee. El tiempo de ejecución de estos ficheros es muy reducido debido a que por su tamaño y densidad no suponen demasiada complejidad.
- **pruebaTiempo1.txt y pruebaTiempo4.txt:** Estos ficheros son pruebas de tamaños muy diferentes pero que comparten que los tiempos de recorrer la carretera se han establecido aleatoriamente, por lo que el resultado ya no depende tanto del numero de carreteras recorridas, sino también, del tiempo que le cuesta recorrer cada una de ellas. En el primer fichero se puede observar como el servicio de paquetería que tarda menos en entregar el paquete seria el de la empresa B, sin embargo, la diferencia es tan sutil que se podría elegir cualquiera de los 2. En el fichero **pruebaTiempo4.txt** si que se puede decir que el servicio de mensajería de la empresa A es mas rápido que el de la B.
- **aleatoria[1..6].txt:** Estos ficheros son pruebas generadas completamente de forma aleatoria, en ellos se puede observar como para tamaños relativamente pequeños el algoritmo encuentra solución en un numero razonable de pasos, incluso costándole menos tiempo a ficheros con mayor numero de intersecciones, debido a la aleatoriedad de su generación.

2.3. Reparto del trabajo

En cuanto al reparto del trabajo en el desarrollo de la practica, esta se ha desarrollado de manera conjunta repartiendo el trabajo de forma equitativa y trabajando mayoritariamente al mismo tiempo de forma telepresencial.

3. Anexo 1

```
/*
* Clase: Carretera
* Funcion: Representa cada una de las carreteras del mapa
* u,v: intersecciones que une
* t_uv: tiempo de recorrer la carretera en minutos
* p_uv: probabilidad de ir de u a v
* p_vu: probabilidad de ir de v a u
**/
class Carretera{
private:
    int u;
    int v;
    int t_uv;
    float p_uv;
    float p_vu;
```

Figura 1: Clase carretera

```
/*
* Clase: Interseccion
* Funcion: Representa cada una de las intersecciones del mapa
* idInterseccion: identificador de la interseccion
* nCruce: numero de carreteras que tiene el cruce
* carreteras: vector que apunta a todas las carreteras del cruce
**/
class Interseccion {
private:
    int idInterseccion;
    int nCruce;
    vector<Carretera*> carreteras;
```

Figura 2: Clase intersección

```
/*
* Clase: Mapa
* Funcion: Representa el mapa
* nIntersecciones: numero de intersecciones del mapa
* mCarreteras: numero de carreteras del mapa
* cVivienda: interseccion en la que se encuentra la vivienda
* almacenA: interseccion en la que se encuentra el almacen A
* almacenB: interseccion en la que se encuentra el almacen B
**/
class Mapa {
private:
    int nIntersecciones;
    int mCarreteras;
    int cVivienda;
    int almacenA;
    int almacenB;
    vector<Interseccion> intersecciones;
```

Figura 3: Clase mapa

```
//Constructor de la clase
Mapa(string path){
    ifstream f;
    f.open(path);
    vector<Carretera*> carreteras;
    if(f.is_open()){
        //Lee los datos iniciales
        f >> nIntersecciones >> mCarreteras >> cVivienda >> almacenA >> almacenB;

        //Inicializa las intersecciones
        for (int i = 0; i < nIntersecciones; i++){
            intersecciones.push_back(Interseccion(i+1));
        }
        int ini, fin, tiempo;
        float prob_uv, prob_vu;
        int indexRoad = 0;
        carreteras.clear();
        while(!f.eof()){

            //Leer las M carreteras
            f >> ini >> fin >> tiempo >> prob_uv >> prob_vu;
            carreteras.push_back(new Carretera(ini,fin,tiempo,prob_uv,prob_vu));

            //Crear las N intersecciones
            intersecciones[ini-1].addCarretera(carreteras[indexRoad]);
            intersecciones[fin-1].addCarretera(carreteras[indexRoad]);
            indexRoad++;
        }
        f.close();
    }
}
```

Figura 4: Constructor de la clase mapa

```
/*
* camino: calcula el camino desde el almacen especificado hasta la vivienda
**/
int camino(string almacen){
    int IDalmacen = elegirAlmacen(almacen);

    random_device num;
    uniform_real_distribution<double> generator(pow(10,-4),1);

    bool caminoEncontrado = false;
    int tiempo = 0, i = 0, maxIter = 0;

    Interseccion estoy = intersecciones[IDalmacen];
    //Se define un numero maximo de iteraciones
    if(nIntersecciones < 50){
        maxIter = 2000;
    }else {
        maxIter = 10000;
    }

    while(!caminoEncontrado && i < maxIter){
        float prob = generator(num);
        float acum = 0.0;
        vector<Carretera *> carrEstoy = estoy.getCarreteras();
        // Se decide a que interseccion debe desplazarse el repartidor
        for (Carretera *c : carrEstoy){
            if (c->getU() == estoy.getId() && prob < c->getPuv()+acum){
                estoy = intersecciones[c->getV()-1];
                tiempo += c->getTuv();
                break;
            }else if (c->getV() == estoy.getId() && prob < c->getPvu()+acum){
                estoy = intersecciones[c->getU()-1];
                tiempo += c->getTuv();
                break;
            }else if (c->getU() == estoy.getId()){
                acum += c->getPuv();
            }else if (c->getV() == estoy.getId()){
                acum += c->getPvu();
            }
        }
        if (estoy.getId() == cVivienda) {
            caminoEncontrado = true;
        }
        i++;
    }
    //Se comprueba si se ha llegado al limite de iteraciones
    if (i == maxIter){
        return 0;
    }else {
        return tiempo;
    }
}
```

Figura 5: Función camino

```cpp
int main(int argc, char *argv[]){
    if (argc != 2){
        cout << "USAGE ERROR" << endl;
        cout << "Usage: ./practica2 <path to file>" << endl;
        exit(1);
    }
    string path = argv[1];
    Mapa paqueteria = Mapa(path);

    int acumA = 0, timeA = 0,nExitoA = 0, tiempoA;
    int acumB = 0, timeB = 0, nExitoB = 0, tiempoB;
    chrono::steady_clock::time_point beginA, endA;
    chrono::steady_clock::time_point beginB, endB;

    //Se realizan 500 iteraciones de cada prueba
    for (int i = 0; i < 500; i ++){
        //Calculo del camino de A
        beginA = chrono::steady_clock::now();
        tiempoA = paqueteria.camino("almacenA");
        endA = chrono::steady_clock::now();
        timeA += chrono::duration_cast<chrono::milliseconds>(endA - beginA).count();
        acumA += tiempoA;
        if (tiempoA != 0) {
            nExitoA++;
        }
        //Calculo del camino de B
        beginB = chrono::steady_clock::now();
        tiempoB = paqueteria.camino("almacenB");
        endB = chrono::steady_clock::now();
        timeB += chrono::duration_cast<chrono::milliseconds>(endB - beginB).count();
        acumB += tiempoB;
        if (tiempoB != 0) {
            nExitoB++;
        }
    }
    if (nExitoA == 0){
        nExitoA = 1;
    }
    if (nExitoB == 0){
        nExitoB = 1;
    }
    cout << path << "," << paqueteria.numIntersecciones() << "," << acumA/nExitoA << "," << acumB/nExitoB << ","<< timeA+timeB << endl;
    return 0;
}
```

Figura 6: Función main

```bash
#!/bin/bash

PRUEBAS=$(ls pruebasEntrega/)

make

for PROG in $PRUEBAS; do
        ./practica2 "pruebasEntrega/"$PROG
done;

make clean
```

Figura 7: Script para la ejecución automática de pruebas