

Universidad de Concepción
Facultad de Ingeniería
Departamento de Ing. Informática y
Cs. de la Computación

Informe proyecto 2

Análisis de Algoritmos (503309-1)
Profesora Cecilia Hernández

Alumnos:

Nicolás Araya

Martina Cádiz

Ayudante:

Manuel Díaz

Fecha:

05/07/21

Descripción del informe

El presente informe tiene el objetivo de describir la implementación de algoritmos aleatorizados, amortizados y diseño greedy en la resolución de los problemas planteados. Para la implementación de los algoritmos se trabajó utilizando C++, se implementaron distintas clases, métodos y estructuras para llevar a cabo las soluciones.

Los puntos se presentan en el siguiente orden:

1. Descripción de las soluciones
 - Requerimiento A
 - Requerimiento B
 - Requerimiento C
 - Requerimiento D
 - Requerimiento E
2. Ejecución del código
 - Ejemplo

Descripción de las soluciones

A continuación, se describirán las soluciones en el orden de aparición del listado entregado.

Requerimiento A

Para la implementación de un generador de Grafos de grilla se definieron la clase Nodo, la clase Grafo y la estructura Edge.

La clase "Nodo" permite almacenar el valor del Nodo, y además guarda una referencia a los Nodos a su alrededor.

La estructura Edge nos sirve para definir las aristas, permite almacenar el peso de la arista y una referencia a los dos Nodos que están unidos.

La clase Grafo aprovecha la clase Nodo y crea una matriz de Nodos de NxM, y un Map para guardar las aristas, el cual dado dos Nodos permite obtener la arista correspondiente.

Al momento de instanciar la clase Grafo, se le debe pasar como parámetro las dimensiones del grafo de grilla, e indicar si se trata de un grafo con pesos o no. El constructor de la clase Grafo, se encarga de crear los NxM Nodos y actualizar los punteros correspondientes a cada Nodo. También establece las aristas de cada par de Nodo y las almacena en el Map.

La clase Grafo, también permite obtener un Nodo utilizando el método getVertice(int, int) pasando como parámetros las coordenadas del Nodo como si fuese una matriz.

Finalmente se puede imprimir el Grafo, llamando al método print().

Requerimiento B

```
1. Pseudocódigo función random
random(a, b):
    if (a==1 and b == 0) or (a==0 and b == 1) then:
        return rand()&1
    max = a
    min = b
    if max < b then:
        max = b
        min = a
    res=INT_MAX
    dif = max - min
    while (res > dif) then:
        res = random(0,1)
        for(i in range(0, log2(dif)+1):
            res = res<<1
            x = random(0,1)
            res = res^x

    return res + min
```

Para la función solo se consideran los números enteros positivos, es decir 4 bytes o 32 bits, y se parte con la base de que si se pide ingresar un 1 o 0 la función retorna un 1 o un 0 utilizando `rand()&1`, el cual obtiene un número random y realiza un AND lógico con un 1, es decir, nos quedamos con el primer bit del número que produce la función `rand()`.

Cuando los números ingresados son distintos de 0 o de 1, la función primero recibe a y b, y encuentra cual es el número mayor, y el número menor. Luego calcula la diferencia que hay entre estos y lo almacena en la variable `dif`. La función `random` ahora lo que intentará es generar un número que está entre 0 y `dif`. Para esto se parte se parte con un número 0 o 1, el cual se guarda en `res`. Posterior a esto se entra a un ciclo `for` el cual recorre desde el bit 0 hasta la posición del bit más significativo de `dif` (Esto se hace para evitar para acortar tiempo, no es necesario recorrer los 32 bits). En cada iteración del `for`, se hace un shift lógico hacia la izquierda en `res`, para luego obtener un valor `x` llamando a `random(0,1)`, con este valor obtenido se usa para realizar un XOR lógico con `res`. Básicamente por cada bit se llama a la función `random(0,1)` para elegir el valor el cual guardar en el bit correspondiente. Si al final del `for` no se obtiene un valor entre 0 y `dif`, se vuelve a generar un número con el ciclo `for`, hasta salir del `while`. Cuando se genera un número correctamente, es retornado adicionando el valor `min`, para así retornar un número entre a y b.

La implementación de la función `random` utiliza un algoritmo Las Vegas, ya que nunca debería dar una respuesta incorrecta, o en este caso un valor incorrecto.

Análisis:

Sea x_i un número, y $P[x_i]$ la probabilidad de obtener x_i está dada por:

$$P[x_i] = 1/n$$

Con $n = 2^{\log_2(dif)+1}$, dado por las veces que itera el ciclo `for`.

Se define el suceso X : obtener un número entre 0 y `dif`.

*`dif` = `abs(a-b)`

Luego,

$$P[X] = \sum_{i=0}^{dif} P[X_i] = \sum_{i=0}^{dif} \frac{1}{n} = \frac{dif}{n}$$

$$E[X] = 1 + P[X]^c \cdot E[X]$$

$$P[X]^c = 1 - P[X] = 1 - \frac{dif}{n} = \frac{n - dif}{n}$$

$$\Rightarrow E[X] = 1 + \left(\frac{n - dif}{n} \right) \cdot E[X]$$

$$\Rightarrow E[X] \left(1 + \frac{n - dif}{n} \right) = 1$$

$$\Rightarrow E[X] \left(\frac{dif}{n} \right) = 1$$

$$\Rightarrow E[X] = \frac{n}{dif}$$

$$\Rightarrow E[X] = \frac{2^{\log_2(dif)+1}}{dif}$$

Aplicando el metodo de limite con $f(dif) = E[X]$ y $g(dif) = \log_2(dif)$

$$\lim_{dif \rightarrow \infty} \left(\frac{2^{\log_2(dif)+1}}{dif \log_2 dif} \right) = 0$$

$$\therefore E[x] = O(\log_2(dif))$$

Requerimiento C

Aprovechando la implementación de la función random, se utiliza en la clase Grafo para asignar valores a las aristas, para esto se debe indicar que se quiere crear un Grafo con pesos, y pasar por parámetro los valores a y b que representan el intervalo de valores que podrían tomar las aristas.

Requerimiento D

La estructura UnionFind fue implementada con las heurísticas de comprensión de ruta y uso de rank, para utilizarla se creó una clase UnionFind. Los métodos relevantes de esta clase son MakeSet(int), Find(int) y Union(int,int).

Con el objetivo de explicar la implementación de las heurísticas, se mostrarán los pseudocódigos de los métodos mencionados anteriormente:

Cabe destacar que la clase posee un único atributo privado que corresponde a un vector llamado "coleccion" de tipo "Node", "Node" es una estructura que posee dos atributos: parent y rank. Rank viene inicializado en cero, parent no.

1. Pseudocódigo método MakeSet

```
MakeSet(x):  
    inicio <- variable tipo Node  
    inicio.parent <- x  
    coleccion <- añadir la variable inicio al vector coleccion
```

2. Pseudocódigo método Find

```
Find(x): //método que trabaja con comprensión de ruta  
    if coleccion[x].parent != x then:  
        coleccion[x].parent = Find(coleccion[x].parent) //llamada  
        recursiva hasta encontrar el id del nodo padre  
    return coleccion[x].parent
```

3. Pseudocódigo método UnionFind

```
UnionFind(x,y): //método que trabaja con rank  
    parent_x <- Find(x) //busca el padre del subset donde esta x  
    parent_y <- Find(y) //busca el padre del subset donde esta y  
  
    //compara los ranks de los nodos padres  
    if coleccion[parent_x].rank > coleccion[parent_y].rank then:  
        coleccion[parent_y].parent = parent_x  
    else coleccion[parent_x].parent = parent_y  
  
    //condición que considera caso que ambos rank posean el mismo valor  
    if coleccion[parent_x].rank == coleccion[parent_y].rank then:  
        coleccion[parent_y].rank++ //se elige aumentar el valor del rank  
        del padre del subset de y
```

Para encontrar el árbol de cobertura mínima se implementó una función llamada "AlgoritmoKruskal" y la estructura principal que utiliza es UnionFind. El parámetro que recibe la función es de tipo Grafo* y corresponde al grafo de grilla en el que se buscará el árbol de cobertura mínima, cabe destacar que este grafo viene inicializado previamente, es decir posee pesos en sus aristas y vértices.

Para resolver el problema, lo primero que se hace en la función es realizar subsets de todos los vértices del grafo, en otras palabras, se inicializa una estructura UnionFind y se insertan los vértices llamando al método MakeSet. Luego se ordenan las aristas del grafo en orden creciente, para esto se creó una función auxiliar "OrdenarPeso" que ayuda a ordenar el vector que contiene las aristas. Por último, se crea un ciclo for que recorra el vector ordenado de aristas, estas contienen dos atributos de tipo nodo, los cuales corresponden a los nodos que comparten esa arista. Dentro del ciclo, se verifica la condición si los nodos poseen el mismo padre, con el fin de determinar si pertenecen al mismo subset. En caso

de no compartir padre, se unen ambos subsets con el método Union() de UnionFind. Esta acción termina cuando se unen todos los subsets, es decir todos los nodos poseen un mismo padre. Gracias a esto se obtiene el árbol de menor cobertura, ya que las aristas recorridas en el ciclo for están en orden creciente. La función imprime el camino de cobertura mínima del grafo y también el valor total de este.

Para facilitar la comprensión de la solución, puede visualizar el pseudocódigo:

```
4. Pseudocódigo función AlgoritmoKruskal
AlgoritmoKruskal(Grafo):
    UF <- estructura union find
    v <- cantidad total de nodos del grafo
    aristas <- vector de de tipo Edge que contiene las aristas del grafo
    min_path <- 0
    not_ans <- vector de tipo Edge que está vacío por el momento

    //crea subset para todos los nodos del grafo
    for i in range (0,v):
        UF.Makeset(i)
    //ordena las aristas en orden creciente gracias a la función OrdenaPeso
    sort(aristas, OrdenaPeso)

    for para cada arista del vector arista:
        if UF.Find(.nodo1) != UF.Find(nodo2) then: //condición que compara
            los parents de los subsets de cada nodos que comparten arista
            min_path += peso //peso de la arista
            UF.Union(nodo1,nodo2) //unión de los subsets
        else not_ans <- insertar arista al vector

    for para cada arista en el vector not_ans:
        grafo <- ocultar las aristas que no pertenecen al árbol de
            cobertura min.

    imprimir grafo //imprime el camino de cobertura mínima del grafo
    imprimir min_path // imprime el valor total del camino mínimo

    return
```

Requerimiento E

Para resolver el problema del laberinto, se implementaron dos funciones, una llamada "CreacionLaberinto" y la otra "Ciclo". La idea de la solución consiste en utilizar un grafo de grilla que tenga las aristas de su perímetro con peso 1 y las aristas internas con peso -1, la única arista del perímetro con peso -1 corresponderá a la que está conectada con el primer nodo (0,0) y el nodo que le sigue abajo. La idea de esto es que el perímetro no forme un ciclo. Luego este grafo ingresa a un bloque while, donde primero se seleccionará de manera aleatoria un nodo que este dentro del grafo y un nodo externo, si estos poseen la arista en común y además esta arista tiene valor -1 se cambiará el peso de la arista a uno. Para comprobar que no se forme ningún ciclo se llamará a la función "Ciclo", la cual recibirá el grafo con el peso actualizado y entregará un booleano "True" si es que se forma un ciclo y "False" en caso contrario. Si se forma un ciclo, se cambiará el peso de la arista seleccionada a su valor original (-1) y continuará de manera iterativa el ciclo while hasta que todos los nodos tengan estén asociados a una arista con peso 1.

En este problema, se utiliza un grafo con pesos -1 o 1, ya que este funciona como bandera para saber si seleccionarla o no. La estructura UnionFind es utilizada en la función ciclo para comprobar si existe uno en el grafo, cabe destacar que ignoran las aristas de peso -1 (se toma que no existen aún) y solo se comprueba entre las aristas de peso 1.

En el párrafo anterior la descripción de la solución fue hecha a grandes rasgos, para mayor detalle analizar el pseudocódigo:

1. Pseudocódigo función Ciclo

```
Ciclo(Grafo):  
  UF <- estructura union find  
  v <- cantidad total de nodos del grafo  
  aristas <- vector de de tipo Edge que contiene las aristas del grafo  
  aristas_nuevas <- vector de de tipo Edge que contiene las aristas de peso=1  
  
  //crea subset para todos los nodos del grafo  
  for i in range (0,v):  
    UF.Makeset(i)  
  //busca las aristas de peso 1  
  for para cada arista del vector arista:  
    if aristas.peso == 1 then:  
      aristas_nuevas <- añadir arista  
  
  aristas <- aristas_nuevas  
  
  for para cada arista del vector arista:  
    if UF.Find(arista.nodo1) == UF.Find(arista.nodo2) then:  
      return True //encontró un ciclo, subsets comparten padre  
    UF.Union(UF.Find(arista.nodo1), UF.Find(arista.nodo2))  
  return False //no encontró ciclo
```

2. Pseudocódigo función CreacionLaberito

CreacionLaberinto(Grafo,n,m):

v <- cantidad total de nodos del grafo
aristas <- vector de de tipo Edge que contiene las aristas del grafo
esta <- vector de par entero que guardará la posición del nodo en la grilla,
por el momento está vacío
no_esta <- vector de par entero que guardará la posición del nodo en la
grilla, por el momento está vacío

for para rellenar vector esta y no_esta:

//en esta sección se encuentra un ciclo for anidado que buscará los nodos
que están en el perímetro (serán los nodos añadidos al vector esta) y los
que están en el interior del grafo de grilla (serán los nodos añadidos al
vector no_esta). Para simplificar la comprensión del pseudocódigo no se
añadirá esta sección ya que es únicamente para llenar los vectores esta y
no_esta. //

for para preparar el grafo:

//Debido a que el grafo posee todas sus aristas con valor 1, en esta parte
se añade un for que cambie los valores de las aristas internas a -1, no se
añade al pseudocódigo por la misma razón anterior//

grafo.setEdge(nodo(0,0), nodoInferior(0,0)) <- -1 //entrada del laberinto

while True:

if esta == v then: //caso en el que el vector esta posea todos los
break nodos, se rompe el while

i <- número random entre 0 y el tamaño del vector esta

j <- número random entre 0 y el tamaño del vector no_esta

n <- nodo del grafo que se encuentre en la posición i en vector esta

m <- nodo del grafo que se encuentre en la posición j en vector
no_esta

//si es -2 getEdge es porque no existe una arista entre n y m

if grafo.getEdge != 1 and grafo.getEdge !=2 then:

grafo.setEdge(n,m,1) //cambia el valor del peso de la arista
a 1, la arista es la que comparta nodo n y m

if Ciclo(grafo) is True:

grafo.setEdge(n,m,-1)

else:

esta <- añadir el nodo en la posición j del vector no_esta

no_esta <- eliminar el nodo en la posición j

//el vector no_esta posee todos los nodos que no están
conectados a una arista con peso 1

grafo.setEdge(nodo(n-1,m-1), nodoSuperior(n-1,m-1)) <- -1 //crea la salida
del laberinto

imprimir grafo //imprime el laberinto

return

Ejecución del código

Al momento de ejecutar el código, este deberá entregar un grafo de grilla NxM con los pesos formados de manera aleatoria, luego deberá mostrar el árbol de cobertura mínima del grafo que se formó. Además deberá aparecer un laberinto NxM.

El código posee la libertad de poder escoger las dimensiones del laberinto y del grafo de grilla. Estas dimensiones se escogen escribiendo argumentos al momento de ejecutar el a.out.

Para ejecutar el código, se recomienda lo siguiente:

Paso 1. Abrir la carpeta Proyecto2

Paso 2. Para compilarlo, escribir en consola lo siguiente: `g++ *.cpp`

Paso 3. Para ejecutarlo, escribir en consola la siguiente línea: `./a.out 6 6 5 15 6 6`

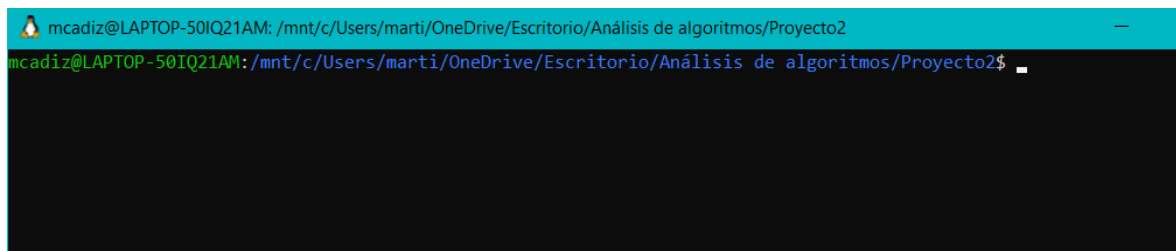
En este caso el grafo y el laberinto tendrán dimensiones 6x6, y el laberinto tendrá aristas entre 5 y 15. El ejecutable asigna los siguientes valores `./a.out N M X Y N M`, el primer par es para el grafo y el segundo es para los valores de las aristas y el tercero para el laberinto.

Se recomienda escoger 4x4, 6x6, para visualizar los cambios.

Paso 4. Ver resultados en consola.

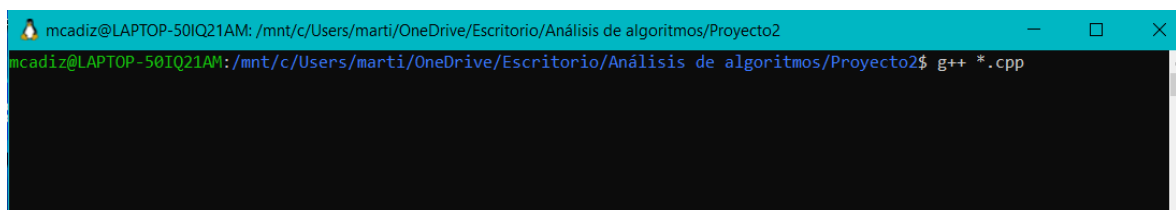
Ejemplo

Paso 1.



```
mcadiz@LAPTOP-50IQ21AM: /mnt/c/Users/marti/OneDrive/Escritorio/Análisis de algoritmos/Proyecto2
mcadiz@LAPTOP-50IQ21AM:/mnt/c/Users/marti/OneDrive/Escritorio/Análisis de algoritmos/Proyecto2$
```

Paso 2.



```
mcadiz@LAPTOP-50IQ21AM: /mnt/c/Users/marti/OneDrive/Escritorio/Análisis de algoritmos/Proyecto2
mcadiz@LAPTOP-50IQ21AM:/mnt/c/Users/marti/OneDrive/Escritorio/Análisis de algoritmos/Proyecto2$ g++ *.cpp
```

Paso 3.

```
[11]~ stopped g11 lepp
nicolas@DESKTOP-2KISC6S:/mnt/c/Users/nicolas/Desktop/Proyecto2$ ./a.out 6 6 5 15 4 4
El largo del grafo ingresado fue: 6
El ancho del grafo ingresado fue: 6
Con los datos ingresados se forma el siguiente grafo de grilla:

*****
|00|<-11->|01|<-10->|02|<-07->|03|<-12->|04|<-15->|05|
^          ^          ^          ^          ^
12         13         8         15         13         5
v          v          v          v          v          v
|06|<-12->|07|<-15->|08|<-09->|09|<-08->|10|<-05->|11|
^          ^          ^          ^          ^          ^
15         15         5         12         8         6
v          v          v          v          v          v
|12|<-10->|13|<-05->|14|<-15->|15|<-13->|16|<-06->|17|
^          ^          ^          ^          ^          ^
11         11         7         12         7         7
v          v          v          v          v          v
|18|<-08->|19|<-13->|20|<-14->|21|<-06->|22|<-12->|23|
^          ^          ^          ^          ^          ^
12         9         9         9         8         7
v          v          v          v          v          v
|24|<-05->|25|<-14->|26|<-14->|27|<-09->|28|<-08->|29|
^          ^          ^          ^          ^          ^
```

Árbol de cobertura mínima del grafo de grilla es:

```
*****
|00|<-01->|01|<-02->|02|<-02->|03|
^          ^          ^          ^
4          2          2          4
v          v          v          v
|04|        |05|        |06|        |07|
^          ^          ^          ^
5          9          6          7
v          v          v          v
|08|        |09|        |10|        |11|
^          ^          ^          ^
9          11         14
v          v          v
|12|<-12->|13|        |14|        |15|

*****
Valor total del camino: 90
```

El largo del laberinto ingresado fue: 4
El ancho del laberinto ingresado fue: 4
Creación laberinto aleatorio:

```
*****
|00|-----|01|-----|02|-----|03|
|                                     |
|04|      |05|      |06|      |07|
|         |         |         |         |
|08|      |09|-----|10|-----|11|
|         |         |         |         |
|12|-----|13|-----|14|-----|15|
*****
```