Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de la Universidad

Tarea 3

Ordenamiento topológico

Autor: Marco Gómez
Email: marcogomezf@gmail.com

Profesor: Patricio Poblete
Auxiliar: Matías Ramírez
Sven Reisenegger
Ayudantes: Nicolás Canales
Pedro Belmonte
Tomas Vallejos
Víctor Garrido

Fecha de entrega: 10 de mayo de 2019
Santiago, Chile

Resumen

En este informe se explica el proceso de como se abordó el problema que se plantea en la tarea. Parte describiendo como se separó el problema en partes, las estructuras de datos que se utilizaron y el invariante del algoritmo que se llevó a cabo. Muestra como se implementaron estas ideas en código java, describiendo brevemente las funciones auxiliares y como se implementan en el algoritmo en si. Finalmente se concluye y reflexiona sobre la tarea.

Índice de Contenidos

1. Diseño de la solución **1**
1.1. Descripción del Algoritmo 1
1.2. Invariante de cada paso 1

2. Implementación **3**
2.1. Estructuras de datos 3
2.2. Funciones auxiliares 3
2.3. Paso 1 3
2.4. Paso 2 4

3. Conclusiones **5**

1. Diseño de la solución

1.1. Descripción del Algoritmo

Para resolver el problema se separa el algoritmo en dos partes. Primero se leerá el input y lo organizara en estructuras de datos, luego se iterará sobre estas estructuras hasta que se resuelva. El algoritmo supone que las entradas son validas, tal como lo plantea el enunciado.

Estructuras de datos a utilizar:

1. Listas en lazadas (Nodo, Lista)
2. Nodos (Int, Lista)

Paso 1 : Crear un Arreglo de Listas con Nodos de los cursos ordenados.

El objetivo es leer el input de los cursos y entregar un arreglo, donde cada posición del arreglo es una lista de cursos con cantidad de requisitos igual a su posición, es decir los cursos en la lista Arreglo[0] tienen cero cursos como requisitos.

Lo primero que hace leer el input, crear un arreglo de listas que es lo que entrega esta función y una lista auxiliar que llamaremos Nodos Libres que utilice para guardar nodos que no sabia a priori en que posición del arreglo iban.

Paso 2 : Imprimir el Arreglo en orden.

En este paso el algoritmo recorre el Arreglo, creado en paso 1, imprimiendo los cursos en orden y bajando un nivel en el arreglo de los cursos que lo tenían como requisito.

1.2. Invariante de cada paso

Crear Arreglo ordenado:

1. Se toma el curso y se pregunta si esta en Nodos libres.
   - Si es así se saca y se agrega en la lista del Arreglo en la posición que corresponde.
   - Si no esta en Nodos Libres quiere decir que no existe el nodo de ese curso, por lo tanto se crea y se agrega a la lista del Arreglo correspondiente.
2. Se itera sobre cada curso que tiene como requisito, y les pregunta si ya esta en el Arreglo o en Nodos Libres. en caso de no tener requisitos no hace nada.
   - Si el Nodo requisito esta en Arreglo o Nodos Libres, se agrega el curso a su lista de quien es requisito.
   - Si no esta quiere decir que no existe, entonces se crea y se agrega a Nodos Libres
3. Avanza a la siguiente linea hasta terminar y entrega el Arreglo.

Imprimir cursos en orden:

1. Encuentra un nodo del arreglo[0] es decir un nodo sin requisitos e imprime su valor.
2. Itera sobre cada curso de que lo tenia es requisito
3. Lo busca en el Arreglo y lo mueve desde Arreglo[n] a Arreglo[n-1] ya que se aprobó un curso que tenia como requisito

2. Implementación

2.1. Estructuras de datos

a) Nodos:
Son objetos con un valor Int asignado, que representa al curso correspondiente. Y un puntero a una Lista enlazada que contiene los cursos del que el es requisito.

b) Listas enlazadas
Son objetos con un puntero a un Nodo (curso) Y un puntero a otra Lista enlazada.

Cada una con sus respectivos constructores.

2.2. Funciones auxiliares

i) buscarEn(int, Lista) - > Nodo:
Retorna un puntero al Nodo que posee a int como curso en la Lista.

ii) buscarEnArreglo(int, Lista[]) - > Nodo:
Retorna un puntero al Nodo que posee a int como curso en el arreglo de Listas.

iii) eliminarNodoDeLista(Nodo, Lista) -> Lista :
Retorna un puntero a la Lista sin Nodo.

iv) agregarNodoALista(Nodo, Lista) -> Lista
Retorna un puntero a Lista pero con Nodo al principio.

v) sacarUltimoDeArreglo0(Lista[]) -> Nodo
Elimina el ultimo Nodo del Arreglo[0] y lo retorna.

vi) sacarUltimoDeRequisitos(Nodo) -> Nodo
Elimina el ultimo curso del que del que Nodo era requisito y lo retorna.

vii) numeroDelArregloEnQueEsta(Nodo, Lista[]) -> Int
retorna en que posición del arreglo se encuentra la Lista que contiene a Nodo

2.3. Paso 1

Explicado cada paso en orden:

1. Recibe el input con Scanner(System.in) y se lee la primera linea asignando ese digito a la variable Int cantidadCursos.

2. Crea un Arreglo de tamaño cantidadCursos y una lista Nodoslibres=null;

3. Empieza un for que recorrera cada linea del input creando el string[] cursos como el arreglo de digitos de la linea.

4. Se recoge el valor del primer curso y se crea NodoCurso que será el nodo del curso si esta en NodosLibres y retorna null si no esta. "Nodo NodoCurso = buscarEn(curso, NodosLibres);"

5. Pregunta ¿NodoCurso == null? es decir "¿esta el curso en nodos libres?"
   - Si esta se elimina con NodosLibres = eliminarNodoDeLista(NodoCurso, NodosLibres);
   - Si no, se crea NodoCurso = new Nodo(curso,null);
6. Se agrega el curso a la lista correspondiente del Arreglo con
   Arreglo[cursos.length - 1] = agregarNodoAListA(NodoCurso, Arreglo[cursos.length - 1]);
7. se crea un segundo for para los requisitos del curso, se extrae el int en una variable requisitoz se pregunta si:
   - ¿Esta en NodosLibres? enLibres = buscarEn(requisito,NodosLibres);
   - ¿Ya esta en el arreglo? enArreglo = buscarEnArreglo(requisito, Arreglo);
     si esta en alguno de los dos se agrega NodoCurso a la lista de quien es requisito de enLibres o enArreglo, respectivamente.

     enLibres.apunta = agregarNodoAListA(NodoCurso,enLibres.apunta);
     enArreglo.apunta = agregarNodoAListA(NodoCurso,enArreglo.apunta);
8. if (enLibres==null and enArreglo == null)
   Es decir, no existe un nodo con int requisito, se crea uno y se agrega a NodosLibres
   Nodo nodoRequerido = new Nodo(requisito, new Lista(NodoCurso));
   NodosLibres = agregarNodoAListA(nodoRequerido, NodosLibres);
9. retornar Arreglo

2.4. Paso 2

Explicado cada paso en orden:

1. Es una iteración que termina cuando el Arreglo[0] este vacío. while(arreglo[0] != null)
   Si arreglo[0] es null, quiere decir que todo el arreglo esta vació y ya se imprimieron todos los cursos en orden.
2. Se recoge el un Nodo çurso"de la lista Arreglo[0] en este caso el ultimo y se imprime.
3. Empieza otra iteración con while que termina cuando la lista de los cursos que requerían a çurso.^{es}te vacía
4. Se elimina uno de esos cursos y calculamos en cual lista del arreglo esta
   Nodo esreq = sacarUltimoDeRequisitos(curso);

   int n= numeroDelArregloEnQueEsta(esreq,arreglo);
5. Lo elimina de Arreglo[n] y agrega a Arreglo[n-1] porque se aprobó un curso que tenia como requisito
6. Cuando estas dos iteraciones terminen estarán todos los cursos impresos en orden y se habrá resuelto el problema.

3. Conclusiones

Gracias a esta tarea podemos dar cuenta de la utilidad de las estructuras de datos, y distribuir los problemas en mas problemas pequeños, en lo personal me sirvió mucho para aprender a programar de forma ordenada cosa que antes no hacia, por lo mismo tuve muchos problemas en realizar la tarea. Al darme cuenta de que mi código estaba muy desordenado y no entendía nada empecé desde cero pero de forma mas ordenada, cosa que ayudo mucho para encontrar pequeños errores en el código y tener mas claridad de que era lo que estaba haciendo.