Informe de Laboratorio 08

Tema: Búsqueda en profundidad

Nota

Estudiante	Escuela	Asignatura
Reyser Julio Zapata Butron rzapata@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Análisis Y Diseño de Algoritmos Semestre: IV Código: 1702231

Laboratorio	Tema	Duración
08	Búsqueda en profundidad	02 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2024 - B	03 diciembre 2024	03 diciembre 2024

1. Código base

El siguiente código presentado, es la base para la realización de los ejerción propuestos

```cpp
#include <iostream>
#include <vector>

using namespace std;

typedef int vertex;

struct Graph
{
  int V;
  vector<vector<int>> adj;
};

static int cnt;
static int pre[1000];

static void dfsR(Graph &G, vertex v, int depth, vector<int> &order);

void GRAPHdfs(Graph &G, vector<int> &order)
{
  cnt = 0;
  for (vertex v = 0; v < G.V; ++v)
    pre[v] = -1;

  for (vertex v = 0; v < G.V; ++v)
```

```
26      if (pre[v] == -1)
27        dfsR(G, v, 0, order);
28  }
29
30  static void dfsR(Graph &G, vertex v, int depth, vector<int> &order)
31  {
32    // Imprime la indentaci n correspondiente a la profundidad
33    for (int i = 0; i < depth; ++i)
34      cout << ". ";
35
36    // Imprime la llamada a dfsR para el v rtice actual
37    cout << v << " dfsR(G," << v << ")\n";
38
39    pre[v] = cnt++;
40    order.push_back(v);
41
42    for (int w = 0; w < G.V; ++w)
43    {
44      if (G.adj[v][w] == 1)
45      {
46        for (int i = 0; i < depth; ++i)
47          cout << ". ";
48
49        if (pre[w] == -1)
50        {
51          cout << v << "-" << w << " dfsR(G," << w << ")\n";
52          dfsR(G, w, depth + 1, order);
53        }
54        else
55        {
56          cout << v << "-" << w << "\n";
57        }
58      }
59    }
60
61    for (int i = 0; i < depth; ++i)
62      cout << ". ";
63    cout << v << "\n";
64  }
65
66  int main()
67  {
68    Graph G;
69    G.V = 12;
70    G.adj = vector<vector<int>>(G.V, vector<int>(G.V, 0));
71
72    G.adj[0][1] = 1;
73    G.adj[0][4] = 1;
74    G.adj[1][2] = 1;
75    G.adj[1][5] = 1;
76    G.adj[2][3] = 1;
77    G.adj[3][7] = 1;
78    G.adj[4][8] = 1;
79    G.adj[5][4] = 1;
80    G.adj[6][5] = 1;
81    G.adj[6][10] = 1;
82    G.adj[6][2] = 1;
83    G.adj[7][11] = 1;
84    G.adj[7][6] = 1;
85    G.adj[8][9] = 1;
86    G.adj[9][5] = 1;
87    G.adj[9][8] = 1;
88    G.adj[10][9] = 1;
89    G.adj[11][10] = 1;
90
```

```
91    vector<int> order;
92
93    GRAPHdfs(G, order);
94
95    // Imprime el orden de visita y los n meros de orden
96    cout << "\nw ";
97    for (size_t i = 0; i < order.size(); ++i)
98      cout << order[i] << " ";
99    cout << "\npre[w] ";
100   for (size_t i = 0; i < order.size(); ++i)
101     cout << pre[order[i]] << " ";
102   cout << "\n";
103
104   // Imprime el vector pre[] ordenado por v rtice
105   cout << "\nv ";
106   for (int v = 0; v < G.V; ++v)
107     cout << v << " ";
108   cout << "\npre[v] ";
109   for (int v = 0; v < G.V; ++v)
110     cout << pre[v] << " ";
111   cout << "\n";
112
113   return 0;
114 }
```

Listing 1: base.cpp

2. Ejercicios Propuestos

2.1. Casos extremos. ¿Cuál es el resultado de GRAPHdfs(G) cuando G.adj es 0? ¿Y cuando G.V es 1?

Cuando `G.adj` es 0

Si la matriz de adyacencia `G.adj` está completamente llena de ceros, esto indica que el grafo no tiene ninguna arista; es decir, todos los vértices son aislados. En este escenario, la función `GRAPHdfs(G)` realizará lo siguiente:

- **Inicialización:** El contador `cnt` se establece en 0 y todos los elementos del arreglo `pre[]` se inicializan a -1, indicando que ningún vértice ha sido visitado.
- **Recorrido de Vértices:** La función iterará sobre cada vértice del grafo. Dado que no hay aristas, la condición `G.adj[v][w] == 1` nunca se cumplirá, por lo que no se realizarán llamadas recursivas a `dfsR()`.
- **Asignación de Números de Orden:** Cada vértice se considerará una componente conectada por sí mismo. Por lo tanto, `dfsR()` se llamará para cada vértice individualmente, asignando un número de orden secuencial a cada uno.
- **Resultado Final:** El vector `order` contendrá todos los vértices en orden ascendente desde 0 hasta `G.V - 1`, y el arreglo `pre[v]` reflejará el orden en que fueron visitados, que coincidirá con su índice debido a la ausencia de aristas.

Cuando `G.V` es 1

Si el grafo tiene únicamente un vértice (`G.V = 1`), la función `GRAPHdfs(G)` operará de la siguiente manera:

- **Inicialización:** Similar al caso anterior, `cnt` se establece en 0 y `pre[0]` se inicializa a -1.

- **Recorrido de Vértices:** Solo hay un vértice que considerar. La condición `pre[v] == -1` se cumple para `v = 0`, por lo que se llamará a `dfsR(G, 0, 0, order)`.
- **Ejecución de `dfsR()`:**
  - Se asigna `pre[0] = 0` y `order` se actualiza para incluir el vértice 0.
  - Se intenta iterar sobre posibles adyacencias, pero dado que `G.adj[0][w]` es 0 para cualquier `w`, no se realizarán más llamadas recursivas.
  - La función termina después de asignar el número de orden al único vértice.
- **Resultado Final:** El vector `order` contendrá solo el vértice 0, y `pre[0]` será 0, reflejando que el único vértice ha sido visitado primero.

2.2. Base de la recursión. ¿Cuál es la base de la recursión en la función dfsR()?

La base de la recursión en la función `dfsR()` se alcanza cuando se intenta explorar un vértice que ya ha sido visitado o cuando no hay más vértices adyacentes no visitados para explorar desde el vértice actual. Específicamente:

- **Vértice ya visitado:** Si durante la exploración de los adyacentes de un vértice `v`, se encuentra un vértice `w` tal que `pre[w] != -1`, esto indica que `w` ya ha sido visitado. En este caso, no se realiza una llamada recursiva y se continúa con el siguiente vértice adyacente.
- **Sin vértices adyacentes no visitados:** Si un vértice `v` no tiene vértices adyacentes `w` tales que `G.adj[v][w] == 1` y `pre[w] == -1`, la función `dfsR()` termina su ejecución para `v`, retornando al vértice desde el cual se realizó la llamada recursiva.

2.3. Realiza una búsqueda en profundidad en el grafo dado por las siguientes listas de adyacencia. Haga un rastreo de la búsqueda.

0	1, 4
1	2, 5
2	3
3	7
4	8
5	4
6	5, 10, 2
7	11, 6
8	9
9	5, 8
10	9
11	10

Ejecución del código

```
> .\base.exe
0 dfsR(G,0)
0-1 dfsR(G,1)
. 1 dfsR(G,1)
. 1-2 dfsR(G,2)
. . 2 dfsR(G,2)
. . 2-3 dfsR(G,3)
. . . 3 dfsR(G,3)
. . . 3-7 dfsR(G,7)
. . . . 7 dfsR(G,7)
. . . . 7-6 dfsR(G,6)
. . . . . 6 dfsR(G,6)
. . . . . 6-2
. . . . . 6-5 dfsR(G,5)
. . . . . 5 dfsR(G,5)
. . . . . 5-4 dfsR(G,4)
. . . . . 4 dfsR(G,4)
. . . . . 4-8 dfsR(G,8)
. . . . . 8 dfsR(G,8)
. . . . . 8-9 dfsR(G,9)
. . . . . 9 dfsR(G,9)
. . . . . 9-5
. . . . . 9-8
. . . . . 9
. . . . . 8
. . . . . 4
. . . . . 5
. . . . . 6-10 dfsR(G,10)
. . . . . 10 dfsR(G,10)
. . . . . 10-9
. . . . . 10
. . . . . 6
. . . . 7-11 dfsR(G,11)
. . . . . 11 dfsR(G,11)
. . . . . 11-10
. . . . . 11
. . . . 7
. . . 3
. . 2
. 1-5
. 1
0-4
0

w 0 1 2 3 7 6 5 4 8 9 10 11
pre[w] 0 1 2 3 4 5 6 7 8 9 10 11

v 0 1 2 3 4 5 6 7 8 9 10 11
pre[v] 0 1 2 3 7 6 5 4 8 9 10 11
```

3. Repositorio de Github

- Repositorio de Github donde se encuentra el actual laboratorio
  `https://github.com/ReyserLynnn/ada-lab-b-24b/tree/main/laboratorio08/src`
- Repositorio de Github donde se encuentran los laboratorios del curso
  `https://github.com/ReyserLynnn/ada-lab-b-24b.git`