EE3980 Algorithms

Homework 7. Grouping Friends

106061151 劉安得

1. Introduction

- Strongly Connected Components

  In this homework, we are going to use the concept of Strongly Connected Components to solve the problem. A strongly connected component is a maximal set of vertices such that for every pair of vertices u and v they are mutual reachable; that is, vertex u is reachable from v and vice versa.

- Depth First Search

  DFS can be used to solve Strongly Connected Components. Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.

- Graph

  DFS is perform on graph. A graph, G, consists of two sets V and E.

  - The set V is a finite, nonempty set of vertices.
  - The set E is a set of pairs of vertices; these pairs are called edges.
  - They are also denoted by V(G) and E(G).
  - And the graph is also denoted by G(V, E).

In this homework, we use ten communication lists as input data, record the CPU times and correlate the performance to analyses.

## 2. Approach

### A. DFS

```
// To find the strongly connected components of the graph G = (V, E).
// Input: graph G
// Output: strongly connected components.
Algorithm SCC(G)
{
    Construct the transpose graph GT;
    DFS_Call(G); // Perform DFS to get array f[1 : n].
    Sort V of GT in order of decreasing value of f[v].
    DFS_Call(GT); // Perform DFS on GT.
    Each tree of the resulting DFS forest is a strongly connected
component.
}

// Initialization and recursive DFS function call.
// Input: graph G
// Output: array f[|V|]: finish time.
Algorithm DFS_Call(G)
{
    for v := 1 to n do { // Initialize to not visited and no predecessor.
        visited[v] := 0;
        f[v] := 0;
    }
    time := 0; // Global variable to track time.
    N = 0;  // Global variable to record the # of trees
    for v := 1 to n do { // To handle forest case.
        if (visited[v] = 0) then DFS(G, v, N);
    }
}

// More sophisticated Depth first search starting from vertex v of graph
G.
// Input: starting node v
```

```
// Output: array f[|V|]: finish time.
Algorithm DFS(G, v, i)
{
    visited[v] := i;
    for each vertex w adjacent to v do {
        if (visited[w] = 0) then {
            DFS(G, w, i);
        }
    }
    time := time + 1;
    f[v] := time;
}

struct NODE {
    TYPE data; // for data storage
    struct NODE *link; // pointer to the next node
}
```

I. Time Complexity

n is the number of vertices unvisited, $e_v$ is the number of edge of vertex v

| Statement | s/e | Freq. | Total steps |
|---|---|---|---|
| **Algorithm DFS(G, v, i)** | 0 | - | 0 |
| **{** | 0 | - | 0 |
| **visited[v] := i;** | 1 | 1 | 1 |
| **for each vertex w adjacent to v do {** | 1 | $e_v$+1 | $e_v$+1 |
| **if (visited[w] = 0) then {** | 1 | $e_v$ | $e_v$ |
| **DFS(G, w, i);** | T(n-1) | 1 | T(n-1) |
| **}** | 0 | - | 0 |
| **}** | 0 | - | 0 |
| **time := time + 1;** | 1 | 1 | 1 |
| **f[v] := time;** | 1 | 1 | 1 |
| **}** | 0 | - | 0 |
| **Total** | | | T(n-1)+2$e_v$+4 |

$$T(n) = \mathrm{T}(\mathrm{n}-1) + 2\mathrm{e}_v + 4 = \sum_{v \in V} 2\mathrm{e}_v + 4\mathrm{n} = 4\mathrm{n} + 2\mathrm{e} = \Theta(\mathrm{n}+\mathrm{e})$$

n is the number of vertices, e is the number of edges,

$n_v$, $e_v$ is the number of vertices and edges of the tree which contains vertex v

| Statement | s/e |
|---|---|
| `Algorithm DFS_Call(G)` | 0 |
| `{` | 0 |
| `for v := 1 to n do {` | 1 |
| `visited[v] := 0;` | 1 |
| `f[v] := 0;` | 1 |
| `}` | 0 |
| `time := 0;` | 1 |
| `N = 0;` | 1 |
| `for v := 1 to n do {` | 1 |
| `if (visited[v] = 0)` | 1 |
| `then DFS(G, v, N);` | $T_{DFS}(n_v, e_v)$ |
| `}` | 0 |
| `}` | 0 |
| **Total** | |

$$T(n,e) = 7 + \sum_{v \in V}(4n_v + 2e_v) = 4n + 2e + 7 = \Theta(n+e)$$

n is the number of vertices, e is the number of edges,

| Statement | s/e |
|---|---|
| `Algorithm SCC(G)` | 0 |
| `{` | 0 |
| `Construct the transpose graph GT;` | Θ(n+e) |
| `DFS_Call(G);` | 4n+2e+7 |
| `Sort V of GT in order of decreasing value of f[v].` | Θ(n) |
| `DFS_Call(GT);` | 4n+2e+7 |
| `}` | 0 |
| **Total** | |

$$T(n,e) = \Theta(n+e) + \Theta(n) + 8n + 4e + 14 = \Theta(n+e)$$

II. Space Complexity

- $S_{DFS}(n, e)$

n+2e for adjacent lists, 2n for array *visited[]* and *f[]*, one for *time*

And for each recursive function call, one for each variable: *v*, *w*, and *i*

There is n stack, therefore

$S_{DFS}(n, e) = 3n + 3n + 2e + 2 = 6n + 2e + 2 = \Theta(n+e)$

- $S_{DFS\_Call}(n, e)$

  n+2e for adjacent lists, 2n for array *visited[]* and *f[]*, one for *time*, *v* and *N*

  $S_{DFS\_Call}(n, e) = 3n + 2e + 3 + S_{DFS}(n, e) = 3n + 2e + 3 + 3n = 6n + 2e + 3 = \Theta(n+e)$

- S(n, e)

  2n+4e for G and GT, 2n for array *visited[]* and *f[]*, one for *time* and *N*

  $S(n, e) = 4n + 4e + 3 + S_{DFS\_Call}(n, e) = 4n + 4e + 2 + 3n + 1 = 7n + 4e + 3 = \Theta(n+e)$

B. Main Function

```
// Driver function to measure SCC function.
// Input: communication list
// Output: CPU time used, the # of subgroups
Algorithm main(void)
{
    Read N & M and a list of names & records and store as G & GT;
    t0 := GetTime();
    SCC(G);
    t1 := GetTime();
    t := (t1 - t0);
    write(t, # of subgroups);
}
```

C. Keys of Implementation

- visited[]

  I use visited[] to record the index of different trees.

  0: unvisited; 1: tree 1; 2: tree 2; … ; i: tree i

And to check whether the node can still perform *visited[] == 0*

- Sort V of GT in order of decreasing value of f[v].

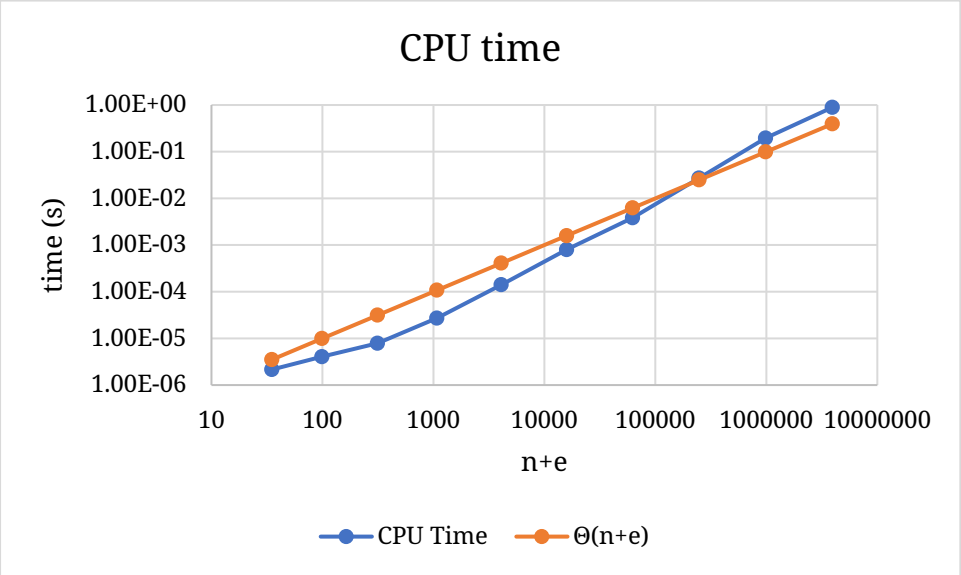
  Because f[v] is 1~n and doesn't repeat with each other. Therefore, it is unnecessary to use a comparison-based sorts. We can directly assign it to a array A which A[N-f[v]] = v.

  And I use an array v[] to record the DFS order of V. When DFS_call() is invoked, it will perform DFS according to v[].

3. Result & Observations

A. CPU Time (Units: s)

| N | M | CPU Time | Subgroup |
|---|---|---|---|
| **10** | 25 | 2.14577e-06 | 3 |
| **20** | 79 | 4.05312e-06 | 5 |
| **40** | 274 | 7.86781e-06 | 6 |
| **80** | 996 | 2.71797e-05 | 9 |
| **160** | 3926 | 0.000140905 | 14 |
| **320** | 15547 | 0.000795841 | 20 |
| **640** | 61786 | 0.00383401 | 30 |
| **1280** | 246515 | 0.027205 | 43 |
| **2560** | 984077 | 0.195953 | 63 |
| **5120** | 3934372 | 0.892375 | 92 |



B. Conclusions/ Observations

The result of analysis matches the plot, the time complexity of brute-force approach is O(n + e), the time complexity of Strongly Connected Components is as same as DFS.

| **Functions** | **Time Complexity** | **Space Complexity** |
|---|---|---|
| **DFS** | Θ(n+e) | Θ(n+e) |
| **DFS_call** | Θ(n+e) | Θ(n+e) |
| **SCC** | Θ(n+e) | Θ(n+e) |