



SCIENCE
SILPAKORN UNIVERSITY

ค่ายโอลิมปิก สอวน. 2

Problems (and Solutions)

ผศ.ดร.ภิญโญ แท้ประสาทสิทธิ์

pinyotae at gmail dot com

หัวข้อเนื้อหา

- ในวันนี้เราจะมาทำโจทย์และเรียนรู้เทคนิคต่าง ๆ ที่น่าสนใจ
 - Breadth-First Search (BFS)
 - Depth-First Search (DFS)
 - priority_queue

BFS (1): ประกาศโครงสร้างเบื้องต้น

```
#include <bits/stdc++.h>
using namespace std;
```

สมมติว่าโหนดในกราฟมีรหัสประจำตัวเป็น int

```
const int MAX_N = 1e5 + 5;
```

```
// adjacency list for storing edges
vector<int> adj[MAX_N];
```

```
// nodes should be visited once
bool visited[MAX_N];
```

- ในการเก็บการเชื่อมต่อแบบ adjacency list เราไม่ใช่ Linked List แต่ใช้ vector แทนแนวคิดของลิสต์
- เวลาที่เราเขียนว่า int A[10] มันหมายถึงอาเรย์เก็บ int สิบช่อง ถ้าเราอยากได้อาเรย์ของเวกเตอร์จำนวนเต็ม เราก็จะเขียนออกมาแบบนี้

โค้ดจากฟีดจอม (แต่ไม่รู้ว่าใครเป็นคนเขียน)

BFS (2): สร้างข้อมูลเส้นเชื่อม

```
int main() {  
    int n, m; // number of nodes and edges  
    cin >> n >> m;
```

ไม่มีอะไรพิเศษ แต่ตั้งสติกับนิยามของชื่อตัวแปรไว้

```
while (m--) {  
    int u, v;  
    cin >> u >> v; // nodes represent edge (u, v)  
  
    // undirected graph, push both u,v and v,u  
    adj[u].push_back(v);  
    adj[v].push_back(u);  
}
```

- เนื่องจากตัว adj มีความหมายว่า adjacency ของโหนด u ไปโหนดที่อยู่ในในเวกเตอร์ทั้งหมด เราจึงค่อย ๆ เติมโหนดลงในเวกเตอร์
- เพราะเป็นเส้นแบบไม่มีทิศทาง (ไม่มีหัวลูกศร) จึงเติมโหนดลงไปในลิสต์ของทั้งต้นทางที่เป็น u และ v

BFS (3): Traverse Graph by Using std::queue

```
queue<int> q;  
q.push(1);  
visited[1] = true;  
vector<int> order;
```

รายการโหนดที่จะ traverse ตามลำดับ

สมมติว่าโหนดหมายเลข 1 เป็นจุดเริ่มต้น

เก็บลำดับการเข้าถึงโหนด

```
while (!q.empty()) {  
    int u = q.front();  
    q.pop();  
  
    order.push_back(u);  
    for (auto v : adj[u]) {  
        if (visited[v] == false) {  
            visited[v] = true;  
            q.push(v);  
        }  
    }  
}
```

เอาโหนดใน q ออกมา traverse

เนื่องจากเราจะเอาโหนดที่ดึงค่ามาแล้วทิ้งไป เราจึงต้องทำการ pop ทิ้ง

หยิบเอารายการโหนดที่เชื่อมต่อกับโหนดปัจจุบันออกมา

ต้องเช็คทุกครั้งว่ายังไม่ visit ไม่อย่างนั้นจะวนไม่รู้จบ

BFS (4): Print Order of Traversal

```
for (auto v : order) cout << v << ' ';
```

เนื่องจาก order เก็บลำดับการเข้าถึงโหนดไว้ การพิมพ์ออกมาแบบนี้
จึงแสดงลำดับแบบ first in, first out เพราะเราใช้ queue จัดลำดับ
การ traverse

ถ้าเราใช้ stack เราจะได้ลำดับแบบอื่นออกมา แต่จะเป็นเลขชุด
เดียวกัน ทว่าในการทดสอบการเข้าถึงโหนดแบบนี้ queue มักจะ
ทำงานได้เร็วกว่า

หัวข้อเนื้อหา


- ในวันนี้เราจะมาทำโจทย์และเรียนรู้เทคนิคต่าง ๆ ที่น่าสนใจ
 - Breadth-First Search (BFS)
 - Depth-First Search (DFS)
 - priority_queue

DFS (1): Declaration

```
#include <bits/stdc++.h>
using namespace std;

const int MAX_N = 1e5 + 5;

// adjacency list for storing edges
vector<int> adj[MAX_N];
bool visited[MAX_N]; // nodes should be visited once
vector<int> order;
```



เนื่องจาก order อันนี้จะต้องถูกใช้ซ้ำในการเรียก recursive แต่ละครั้ง
ดังนั้นเราจึงยกออกมาเพื่อใช้ร่วมกันง่าย ๆ

DFS (2): Recursive Function

```
void dfs(int u) {  
    order.push_back(u);
```

โหนดที่จะ visit

บันทึกลำดับไว้ก่อนไปต่อ

```
    for (auto v : adj[u]) {  
        if (visited[v] == false) {  
            visited[v] = true;  
            dfs(v);  
        }  
    }  
}
```

ดึงเอาเส้นเชื่อมออกมา

ป้องกันการ visit ซ้ำ

Traverse node v ที่
ไม่เก็บไว้ในคิวหรือลิสต์

DFS (3): Recursive Function

ทำให้พิมพ์ผลลัพธ์ด้วย cout เร็วขึ้น

```
int main() {  
    cin.tie(nullptr)->sync_with_stdio(false);  
    int n, m; // number of nodes and edges  
    cin >> n >> m;  
  
    while (m--) {  
        int u, v;  
        cin >> u >> v; // nodes represent edge  
        // undirected graph, push both u,v and v,u  
        adj[u].push_back(v);  
        adj[v].push_back(u);  
    }  
  
    visited[1] = true;  
    dfs(1);
```

ดึงเอาเส้นเชื่อมออกมา

โดยรวมเหมือนเดิม ต่างกันตรงที่เรียก
เป็นฟังก์ชันแยกแบบระบุหมายเลข
โหนดเริ่มต้น

หัวข้อเนื้อหา

- ในวันนี้เราจะมาทำโจทย์และเรียนรู้เทคนิคต่าง ๆ ที่น่าสนใจ
 - Breadth-First Search (BFS)
 - Depth-First Search (DFS)
 - `priority_queue`