Graph Implementation

การอบรมคอมพิวเตอร์โอลิมปิก สอวน. ค่ายที่ 2
ปีการศึกษา 2564

Pisit Makpaisit

COMPUTER SCIENCE

Adjacency Matrix vs Adjacency List

	Adjacency Matrix	Adjacency List
ตรวจสอบว่ามี edge จาก u -> v หรือไม่	O(1)	O(d)
หาจำนวน degree	O(n)	O(d)
Memory ที่ใช้ (Sparse graph)	$O(n^2)$	O(n + m)
Memory ที่ใช้ (Dense graph)	$O(n^2)$	O(n + m)
เพิ่ม/ลบ edge	O(1)	O(1) / O(d)
DFS/BFS	$O(n^2)$	O(n + m)

n = จำนวน vertex, m = จำนวน edge

Adjacency Matrix

```
int m[100][100];                    // เป็น 0 ทุกช่องเพราะประกาศแบบ Global

void addEdge(int u, int v) {        // undirected graph
    m[u][v] = m[v][u] = 1;          // ถ้ามี weight ให้ส่งค่า weight มา set แทน 1
}
```

Adjacency List

```
void addEdge(vector<vector<int>> &adj, int u, int v) {
    adj[u].push_back(v);
    adj[v].push_back(u);
}

int main() {
    vector<vector<int>> adj(n);     // กำหนดค่าให้มี n ตัว (n  = จำนวน vertex)
    ...
}
```

Basic Graph Algorithms

- Graph Traversal (DFS/BFS)
- Graph Coloring
- Cycle Detection
- Single Source Shortest Path (Dijkstra Algorithm)
- All-Pairs Shortest Path (Floyd–Warshall Algorithm)
- Minimum Spanning Tree (Prim / Kruskal Algorithm)
- Topological Sorting

Adjacency List (Weighted Graph)

```
void addEdge(vector<vector<pair<int, int>>> &adj, int u, int v, int weight) {
    adj[u].push_back(make_pair(v, weight));
    adj[v].push_back(make_pair(u, weight));
}

int main() {
    vector<vector<pair<int, int>>> m(adj);
    ...
}
```

DFS (Adjacency List)

```
bool visited[n];                    // n เป็นจำนวน vertex
vector<vector<int>> adj(n);

void dfs_rec(int u) {
    if (visited[u]) return;
    visited[u] = true;
    for ( auto v: adj[u] ) dfs_rec(v);
}

void dfs() {    // ถ้า graph มีหลาย component จะเรียก dfs recursive ครั้งเดียวไม่ได้
    for (int i = 0; i < n; i++) dfs_rec(i);
}
```

BFS (Adjacency List)

```
bool visited[n];                    // n เป็นจำนวน vertex
vector<vector<int>> adj(n);

void bfs(int u) {
    queue<int> q;       q.push(u);     visited[u] = true;
    while (!q.empty()) {
        int s = q.front();   q.pop();
        for (auto v: adj[s])  {
            if (!visited[v]) {
                q.push(v);    visited[v] = true;
            }
        }
    }
}
```

BFS + Save Level (Adjacency List)

```
bool visited[n];                // n เป็นจำนวน vertex
vector<int> level(n);
vector<vector<int>> adj(n);

void bfs(int u) {
   queue<int> q;      q.push(u);    visited[u] = true;    level[u] = 0;
   while (!q.empty()) {
      int s = q.front();   q.pop();
      for (auto v: adj[s])  {
         if (!visited[v]) {
            q.push(v);    visited[v] = true;   level[v] = level[s] + 1;
         }
      }
   }
}
```

Dijkstra's Algorithm

```
vector<vector<pair<int, int>>> adj;
```

ใช้ set แทน priority_queue

```
void dijkstra(int s, vector<int> & d, vector<int> & p) {
    int n = adj.size();    d.assign(n, INF);    p.assign(n, -1);

    d[s] = 0;
    set<pair<int, int>> q;  q.insert({0, s});

    while (!q.empty()) {
        int v = q.begin()->second;    q.erase(q.begin());
        for (auto edge : adj[v]) {
            int to = edge.first, len = edge.second;
            if (d[v] + len < d[to]) {
                q.erase({d[to], to});
                d[to] = d[v] + len, p[to] = v;
                q.insert({d[to], to});
            }
        }
    }
}
```