VÕ TIẾN
Thảo luận kiến thức CNTT trường BK về KHMT(CScience), KTMT(CEngineering)
https://www.facebook.com/groups/khmt.ktmt.cse.bku

Cấu Trúc Dữ Liệu và Giải Thuật (DSA)

DSA3 - HK242

Cuối Kì

Thảo luận kiến thức CNTT trường BK
về KHMT(CScience), KTMT(CEngineering)
https://www.facebook.com/groups/khmt.ktmt.cse.bku

TP. HỒ CHÍ MINH, THÁNG 4/2025

Mục lục

1 Lý Thuyết Graph

1.1 Các loại biểu diễn đồ thị

1. Ma trận kề (Adjacency Matrix)

- Sử dụng ma trận vuông kích thước $n \times n$ với n là số đỉnh.
- Phần tử $a[i][j] = 1$ nếu tồn tại cạnh từ i đến j, ngược lại là 0.
- Với đồ thị có trọng số, $a[i][j]$ là trọng số cạnh hoặc vô cực nếu không có cạnh.

Ưu điểm:

- Truy xuất cạnh nhanh: $O(1)$.
- Dễ cài đặt.

Nhược điểm:

- Tốn bộ nhớ: $O(n^2)$.

2. Danh sách kề (Adjacency List)

- Mỗi đỉnh có một danh sách các đỉnh kề.
- Với đồ thị có trọng số: lưu cặp `(đỉnh kề, trọng số)`.

Ưu điểm:

- Tiết kiệm bộ nhớ với đồ thị thưa: $O(n + m)$.
- Duyệt các đỉnh kề nhanh.

Nhược điểm:

- Kiểm tra cạnh tồn tại chậm hơn: $O(\text{bậc của đỉnh})$.

3. Danh sách cạnh (Edge List)

- Lưu danh sách các cạnh dưới dạng các cặp (u, v) hoặc bộ ba (u, v, w) nếu có trọng số.

Ưu điểm:

- Đơn giản, dễ sử dụng cho các thuật toán như Kruskal.

Nhược điểm:

- Không tiện cho việc duyệt các đỉnh kề hoặc kiểm tra cạnh tồn tại.

So sánh tổng quan

Biểu diễn	Bộ nhớ	Duyệt đỉnh kề	Kiểm tra cạnh	Phù hợp với
Ma trận kề	$O(n^2)$	Trung bình	Rất nhanh ($O(1)$)	Đồ thị dày
Danh sách kề	$O(n + m)$	Nhanh	Trung bình	Đồ thị thưa
Danh sách cạnh	$O(m)$	Không tiện	Chậm ($O(m)$)	Kruskal, duyệt cạnh

Chọn biểu diễn tùy bài toán:

- Cần kiểm tra nhanh cạnh → Ma trận kề.
- Cần duyệt hàng xóm thường xuyên → Danh sách kề.
- Duyệt hoặc sắp xếp cạnh → Danh sách cạnh.

1.2 DFS và BFS

Ý tưởng DFS:

- DFS là thuật toán duyệt theo chiều sâu.
- Xuất phát từ một đỉnh, ta đi càng sâu càng tốt theo các đỉnh kề chưa được thăm.
- Khi không còn đường để đi, ta quay lui để tiếp tục duyệt các nhánh còn lại.

```
void DFS(int u) {
    visited[u] = true; // Đánh dấu đỉnh u đã được thăm
    for (int v : adj[u]) { // Duyệt các đỉnh kề với u
        if (!visited[v]) {
            DFS(v); // Gọi đệ quy DFS từ đỉnh v
        }
    }
}
```

Ý tưởng BFS:

- BFS là thuật toán duyệt theo chiều rộng.
- Xuất phát từ một đỉnh, ta lần lượt thăm các đỉnh kề trước (các đỉnh gần), sau đó mở rộng dần ra các đỉnh xa hơn.
- BFS thường sử dụng hàng đợi (queue) để lưu các đỉnh cần xử lý.

```
void BFS(int s) {
    queue<int> q;
    visited[s] = true;
    q.push(s); // Bắt đầu từ đỉnh s

    while (!q.empty()) {
        int u = q.front(); q.pop();
        for (int v : adj[u]) {
            if (!visited[v]) {
                visited[v] = true;
                q.push(v); // Thêm đỉnh kề vào hàng đợi
            }
        }
    }
}
```

1.3 Topo Sort

Ý tưởng DFS:

- Duyệt DFS từ các đỉnh chưa thăm.
- Khi rời khỏi đỉnh (hết nhánh), thêm vào danh sách kết quả.
- Đảo ngược danh sách kết quả để được thứ tự topo.

```cpp
vector<int> topo;
bool visited[MAX];

void dfs(int u) {
    visited[u] = true;
    for (int v : adj[u]) {
        if (!visited[v])
            dfs(v);
    }
    topo.push_back(u); // thêm sau khi duyệt xong
}

void topological_sort(int n) {
    for (int i = 1; i <= n; ++i) {
        if (!visited[i])
            dfs(i);
    }
    reverse(topo.begin(), topo.end()); // Đảo ngược để được topo đúng
}
```

Ý tưởng BFS:

- Tính bậc vào (in-degree) của mỗi đỉnh.
- Đưa các đỉnh có in-degree = 0 vào hàng đợi.
- Lặp lại: lấy đỉnh ra, giảm in-degree các đỉnh kề, nếu còn đỉnh có in-degree = 0 thì tiếp tục.

```cpp
vector<int> topo;
int indegree[MAX];

void topological_sort(int n) {
    queue<int> q;
    for (int i = 1; i <= n; ++i)
        if (indegree[i] == 0)
            q.push(i);

    while (!q.empty()) {
        int u = q.front(); q.pop();
        topo.push_back(u);

        for (int v : adj[u]) {
            indegree[v]--;
            if (indegree[v] == 0)
                q.push(v);
        }
    }
}
```

1.4 Câu khung nhỏ nhất và dijkstra

1. **Kruskal:**
   - Sắp xếp tất cả các cạnh theo thứ tự tăng dần trọng số.
   - Duyệt từng cạnh và thêm vào cây nếu **không tạo chu trình** (dùng Union-Find để kiểm tra).
   - Dừng khi có đủ `n - 1` cạnh.
2. **Prim:**
   - Xây dựng cây dần dần từ một đỉnh ban đầu.
   - Mỗi bước chọn cạnh có trọng số nhỏ nhất nối từ cây hiện tại đến đỉnh chưa thuộc cây.
   - Dùng hàng đợi ưu tiên để tối ưu hiệu suất.
3. **Dijkstra – Tìm đường đi ngắn nhất từ một đỉnh**
   - Khởi tạo khoảng cách từ nguồn đến mọi đỉnh là vô cùng (`INF`), riêng đỉnh nguồn là `0`.
   - Mỗi bước chọn đỉnh có khoảng cách nhỏ nhất chưa xử lý.
   - Cập nhật khoảng cách của các đỉnh kề nếu tìm được đường đi ngắn hơn.
   - Lặp lại cho đến khi xử lý hết các đỉnh.

2 Câu hỏi trong các đề thi

1. Cho G là một đồ thị đơn với 20 đỉnh và 8 thành phần liên thông. Nếu xóa một đỉnh của G thì số thành phần liên thông của G sẽ nằm trong đoạn . . . .

   a) 8 và 20 b) 8 và 19 c) 7 và 19 d) 7 và 20

   Làm 4 câu tiếp theo Cho đồ thị có hướng $G(V_x, E_d)$, với:

   $$V_x = \{A, B, C, D, E, F\}$$

   Các cạnh của đồ thị là:

   - $A \to B$
   - $A \to C$
   - $B \to E$
   - $C \to D$
   - $C \to E$
   - $D \to E$
   - $D \to F$
   - $E \to A$
   - $E \to B$

2. Ma trận kề E_d của đồ thị là:

   a) $\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$ b) $\begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$

   c) $\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$ d) $\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$

3. Danh sách lân cận Hỏi danh sách lân cận của đồ thị $G(V_x, E_d)$:

   a)
   $A \to \{B, C\}$
   $B \to \{E\}$
   $C \to \{D, E\}$
   $D \to \{E, F\}$
   $E \to \{A, B\}$
   $F \to \{\}$

   b)
   $A \to \{B, C, D\}$
   $B \to \{E\}$
   $C \to \{D, E\}$
   $D \to \{E, F\}$
   $E \to \{A, B, C\}$
   $F \to \{\}$

   c)
   $A \to \{B, C\}$
   $B \to \{E\}$
   $C \to \{D, E\}$
   $D \to \{F, E\}$
   $E \to \{B, A\}$
   $F \to \{\}$

   d)
   $A \to \{B, C\}$
   $B \to \{E\}$
   $C \to \{D, F\}$
   $D \to \{E, F\}$
   $E \to \{A, B\}$
   $F \to \{\}$

4. Duyệt DFS từ đỉnh A Hỏi thứ tự duyệt DFS từ đỉnh A:

   a) $A \to B \to E \to C \to D \to F$ b) $A \to C \to D \to F \to E \to B$
   c) $A \to C \to E \to B \to D \to F$ d) $A \to B \to C \to D \to E \to F$

5. Duyệt BFS từ đỉnh A Hỏi thứ tự duyệt BFS từ đỉnh A:

a) $A \to B \to C \to E \to D \to F$
b) $A \to C \to B \to D \to E \to F$
c) $A \to B \to C \to E \to F \to D$
d) $A \to C \to B \to E \to D \to F$

Làm câu tiếp theo Cho một đồ thị được biểu diễn dưới dạng ma trận kề như sau:

	U	P	B	S	T	J	O	H	W
U	0	1	0	0	0	0	0	1	0
P	0	0	1	0	0	0	0	1	0
B	0	0	0	0	0	1	0	0	0
S	0	0	1	0	1	0	0	0	0
T	0	0	0	0	0	1	0	0	0
J	0	0	0	0	0	0	0	0	0
O	0	0	0	0	0	0	0	1	0
H	0	0	0	0	0	0	0	0	0
W	0	0	0	0	0	0	0	0	0

6. Một thứ tự topo của đồ thị trên là:

a) $O, U, P, H, W, B, S, T, J$
b) $O, U, H, P, W, S, B, J, T$
c) $O, U, P, H, W, S, B, T, J$
d) $O, U, H, P, W, S, B, T, J$

7. Duyệt đồ thị theo thứ tự **DFS** bắt đầu từ đỉnh U. Thứ tự duyệt đúng là:

a) $U, P, B, J, S, T, H, W, O$
b) $U, P, B, J, H, O, S, T, W$
c) $U, P, H, W, S, B, T, J, O$
d) $U, P, S, B, J, T, H, O, W$

8. Duyệt đồ thị theo thứ tự **BFS** bắt đầu từ đỉnh U. Thứ tự duyệt đúng là:

a) $U, P, H, B, W, S, O, T, J$
b) $U, P, H, S, W, B, O, T, J$
c) $U, P, B, H, W, S, T, O, J$
d) $U, P, S, H, W, B, T, J, O$

Làm 5 câu tiếp theo Cho một đồ thị được biểu diễn dưới dạng danh sách kề như sau (các số là trọng số):

$$A : B(2), C(1)$$
$$B : C(2), D(3)$$
$$C : D(1), E(4)$$
$$D : E(1)$$
$$E : \varnothing$$

9. **Duyệt DFS từ đỉnh** A. Thứ tự duyệt đúng là:

a) A, B, C, D, E
b) A, C, D, E, B
c) A, B, D, E, C
d) A, C, B, E, D

10. **Duyệt BFS từ đỉnh** A. Thứ tự duyệt đúng là:

a) A, B, C, D, E
b) A, C, B, D, E
c) A, B, C, E, D
d) A, C, D, B, E

11. **Giá trị khung nhỏ nhất** (Minimum Spanning Tree) của đồ thị. Tìm tổng trọng số của khung nhỏ nhất:

a) 5
b) 6
c) 7
d) 8

12. **Thứ tự Topo Sort** của đồ thị là:

a) A, B, C, D, E
b) A, C, B, E, D
c) A, B, D, C, E
d) A, C, D, B, E

13. **Thứ tự của giải thuật Dijkstra** từ đỉnh A. Khoảng cách ngắn nhất từ A đến các đỉnh khác lần lượt là:

a) $A(0), B(2), C(1), D(2), E(3)$
b) $A(0), B(2), C(1), D(3), E(4)$
c) $A(0), B(3), C(1), D(2), E(5)$
d) $A(0), B(2), C(2), D(3), E(6)$

Cho 4 câu sau với đoạn code sau

```
class Graph {
private:
    int nover;        // Số đỉnh trong đồ thị
    int **wm;         // Ma trận trọng số

    void dfsUtil(int v, vector<bool> &visited, vector<int> &result)
     → {
        visited[v] = true;
        result.push_back(v);
        for (int i = 0; i < nover; ++i) {
            if (wm[v][i] != 0 && !visited[i]) {
                /*code1*/
            }
        }
    }
public:
   vector<int> dfs() {
        vector<bool> visited(nover, false);
        vector<int> result;
        for (int i = 0; i < nover; ++i) {
            if (/*code2*/) {
                dfsUtil(i, visited, result);
            }
        }
        return result;
    }
    vector<int> bfs() {
        vector<bool> visited(nover, false);
        vector<int> result;
        queue<int> q;

        for (int i = 0; i < nover; ++i) {
            if (!visited[i]) {
                q.push(i);
                /*code3*/
                while (!q.empty()) {
                    int v = q.front();
                    q.pop();
                    result.push_back(v);

                    for (int j = 0; j < nover; ++j) {
                        if (/*code4*/) {
```

```
                            q.push(j);
                            visited[j] = true;
                        }
                    }
                }
            }
        }
        return result;
    }
```

14. Hãy điền vào `Code1`:

    a) dfsUtil(i, visited, result);
    b) dfsUtil(i+1, visited, result);
    c) dfsUtil(i-1, visited, result);
    d) dfs();

15. Hãy điền vào `Code2`:

    a) visited[i] == false;
    b) visited[i] == true;
    c) visited[i - 1] == true;
    d) visited[i - 1] == false;

16. Hãy điền vào `Code3`:

    a) visited[i] = true;;
    b) visited[i] = false;
    c) visited[i-1] = true;
    d) visited[i-1] = false;

17. Hãy điền vào `Code4`:

    a) wm[v][j] != 0 && !visited[j];
    b) wm[v][j] == 0 && visited[j];
    c) wm[v][j] != 0;
    d) wm[v][j] == 0;

Cho 5 câu sau Cho lớp `Graph` được hiện thực dùng ma trận liền kề như sau:

```
class Graph {
private:
    int nover; // số đỉnh trong đồ thị
    int **wm;  // ma trận trọng số
    int minDistance(int dist[], bool sptSet[]) {
        int min = INT_MAX, min_index;
        for (int v = 0; v < nover; v++)
            if (sptSet[v] == false && dist[v] <= min)
                min = dist[v], min_index = v;
        return min_index;
    }
public:
    void dijkstra(int src) {
        int dist[nover];
        bool sptSet[nover];
        for (int i = 0; i < nover; i++)
            dist[i] = INT_MAX, sptSet[i] = false;
        dist[src] = 0;
        for (int count = 0; count < /*Code1*/; count++) {
            int u = minDistance(dist, sptSet);
            /*Code2*/ = true;
```

```
22              for (int v = 0; v < nover; v++)
23                  if (!sptSet[v] && dist[u] != INT_MAX && /*Code3*/)
24                      dist[v] = /*Code4*/;
25          }
26          // printSolution(dist);
27      }
28  };
```

18. Điền vào `Code1`:

    a) `nover` b) `nover - 1` c) `src` d) `src - 1`

19. Điền vào `Code2`:

    a) `sptSet[u]` b) `sptSet[src]` c) `sptSet[count]` d) `sptSet[nover]`

20. Điền vào `Code3`:

    a) `wm[u][v] < dist[v]` b) `dist[v] + wm[u][v] < dist[u]`
    c) `wm[u][v] < dist[u]` d) `dist[u] + wm[u][v] < dist[v]`

21. Điền vào `Code4`:

    a) `dist[u]` b) `wm[u][v]`
    c) `dist[u] + wm[u][v]` d) `dist[v] + wm[u][v]`

22. Với chức năng trả về vị trí của đỉnh có trọng số nhỏ nhất và sau đó loại bỏ đỉnh này khỏi danh sách ở mỗi lần lặp, nên áp dụng cấu trúc dữ liệu nào cho hàm `minDistance` thì thích hợp nhất để giảm độ phức tạp thời gian thực thi?

    a) Đống (*Heap*) b) Chồng (*Stack*)
    c) Danh sách liên kết đơn (*Singly Linked List*) d) AVL