BFS vs DFS vs Dijkstra vs Bellman ford vs Floyd warshall

DFS (Depth-First Search)

- **إيه هي؟** :دي خوارزمية بتستخدم للبحث في الجراف بتبدأ من نود معينة وتستكشف أقصى العمق في كل فرع قبل ما ترجع للخلف.
- **استخداماتها** :البحث، التحقق من وجود مسار، اكتشاف الدورات.
- **التعقيد الزمني**O(V + E) :، حيث V هو عدد النود و E هو عدد الأضلاع.

BFS (Breadth-First Search)

- **إيه هي؟** :دي خوارزمية بتستخدم للبحث في الجراف بتبدأ من نود معينة وتستكشف كل النود في نفس المستوى قبل ما تنزل للمستوى اللي بعده.
- **استخداماتها** :البحث، إيجاد أقصر مسار في جراف غير موجه بدون أوزان.
- **التعقيد الزمني**.(O(V + E :

Dijkstra

- **إيه هي؟** :دي خوارزمية بتستخدم لإيجاد أقصر مسار من نود معينة لكل النود الأخرى في الجراف مع أوزان غير سالبة.
- **استخداماتها** :إيجاد أقصر مسار في جرافات مع أوزان موجبة.
- **التعقيد الزمني** O(V^2) :مع مصفوفة أو O(E + V log V) مع كومة أولويات.(Priority Queue)

Bellman-Ford

- **إيه هي؟** :دي خوارزمية بتستخدم لإيجاد أقصر مسار من نود معينة لكل النود الأخرى وبتشتغل حتى لو كان في أوزان سالبة.
- **استخداماتها** :إيجاد أقصر مسار في جرافات فيها أوزان سالبة، والتحقق من وجود دورات سالبة.
- **التعقيد الزمني**.(O(V * E :

Floyd-Warshall

- **إيه هي؟** :دي خوارزمية بتستخدم لإيجاد أقصر مسار بين كل زوج من النود في الجراف.
- **استخداماتها** :إيجاد أقصر مسار بين كل النود في جراف كثيف.

- **التعقيد الزمني**.O(V^3) :

مقارنة بسيطة:

- **DFSو :BFS** بيستخدموا للبحث والتحقق من المسارات، DFSبيستكشف العمق الأول وبعدين يتراجع، و BFSبيستكشف المستوى الأول قبل ما ينزل للي بعده.
- :**Dijkstra**بيستخدم لإيجاد أقصر مسار في جرافات بأوزان موجبة.
- :**Bellman-Ford**بيستخدم في الجرافات اللي ممكن يكون فيها أوزان سالبة.
- :**Floyd-Warshall**بيستخدم لإيجاد أقصر مسار بين كل زوج من النود في الجراف الكثيف.

كل خوارزمية لها استخداماتها الخاصة وتعقيدها الزمني، وده بيساعدك تختار الأنسب لمشكلتك

Templets\algorithmes.cpp

```cpp
#include <iostream>
#include <vector>
#include <queue>
#include <climits>
using namespace std;

// بيانات الجراف
const int MAX = 1005;
vector<pair<int, int>> adj[MAX]; // قائمة الجوار

// خوارزمية البحث في العمق
void DFS(int node, vector<bool> &visited)
{
    visited[node] = true;
    cout << node << " ";

    for (auto neighbor : adj[node])
    {
        int nextNode = neighbor.first;
        if (!visited[nextNode])
        {
            DFS(nextNode, visited);
        }
    }
}

// خوارزمية البحث في العرض
void BFS(int start)
{
    vector<bool> visited(MAX, false);
    queue<int> q;
    q.push(start);
    visited[start] = true;

    while (!q.empty())
    {
        int node = q.front();
        q.pop();
        cout << node << " ";

        for (auto neighbor : adj[node])
        {
            int nextNode = neighbor.first;
            if (!visited[nextNode])
            {
                q.push(nextNode);
```

```
                visited[nextNode] = true;
            }
        }
    }
}

// خوارزمية Dijkstra
void Dijkstra(int start)
{
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>
> pq;
    vector<int> dist(MAX, INT_MAX);
    dist[start] = 0;
    pq.push({0, start});

    while (!pq.empty())
    {
        int node = pq.top().second;
        int distance = pq.top().first;
        pq.pop();

        for (auto neighbor : adj[node])
        {
            int nextNode = neighbor.first;
            int weight = neighbor.second;

            if (dist[nextNode] > dist[node] + weight)
            {
                dist[nextNode] = dist[node] + weight;
                pq.push({dist[nextNode], nextNode});
            }
        }
    }
}

// خوارزمية Bellman-Ford
bool BellmanFord(int start, int n)
{
    vector<int> dist(MAX, INT_MAX);
    dist[start] = 0;

    for (int i = 1; i <= n - 1; ++i)
    {
        for (int u = 1; u <= n; ++u)
        {
            for (auto neighbor : adj[u])
            {
                int v = neighbor.first;
                int weight = neighbor.second;
```

```
                if (dist[u] != INT_MAX && dist[u] + weight < dist[v])
                {
                    dist[v] = dist[u] + weight;
                }
            }
        }
    }

    // التحقق من وجود دورات سالبة
    for (int u = 1; u <= n; ++u)
    {
        for (auto neighbor : adj[u])
        {
            int v = neighbor.first;
            int weight = neighbor.second;
            if (dist[u] != INT_MAX && dist[u] + weight < dist[v])
            {
                return false; // يوجد دورة سالبة
            }
        }
    }
    return true; // لا يوجد دورات سالبة
}

// خوارزمية Floyd-Warshall
void FloydWarshall(int n)
{
    vector<vector<int>> dist(MAX, vector<int>(MAX, INT_MAX));

    // تعيين الأوزان المبدئية
    for (int u = 1; u <= n; ++u)
    {
        dist[u][u] = 0;
        for (auto neighbor : adj[u])
        {
            int v = neighbor.first;
            int weight = neighbor.second;
            dist[u][v] = weight;
        }
    }

    // خوارزمية الأساس
    for (int k = 1; k <= n; ++k)
    {
        for (int i = 1; i <= n; ++i)
        {
            for (int j = 1; j <= n; ++j)
            {
                if (dist[i][k] != INT_MAX && dist[k][j] != INT_MAX &&
```

```
                    dist[i][k] + dist[k][j] < dist[i][j])
                {
                    dist[i][j] = dist[i][k] + dist[k][j];
                }
            }
        }
    }
}

int main()
{
    // مثال لاضافة الأضلاع في الجراف
    adj[1].push_back({2, 5});
    adj[1].push_back({3, 3});
    adj[2].push_back({3, 2});
    adj[2].push_back({4, 6});
    adj[3].push_back({4, 7});

    // مثال لاستخدام الخوارزميات
    cout << "DFS: ";
    vector<bool> visited(MAX, false);
    DFS(1, visited);
    cout << endl;

    cout << "BFS: ";
    BFS(1);
    cout << endl;

    cout << "Dijkstra: ";
    Dijkstra(1);
    cout << endl;

    cout << "Bellman-Ford (Negative Cycle?): ";
    if (BellmanFord(1, 4))
    {
        cout << "No";
    }
    else
    {
        cout << "Yes";
    }
    cout << endl;

    cout << "Floyd-Warshall: ";
    FloydWarshall(4);
    // اطبع النتيجة هنا
    return 0;
}
```