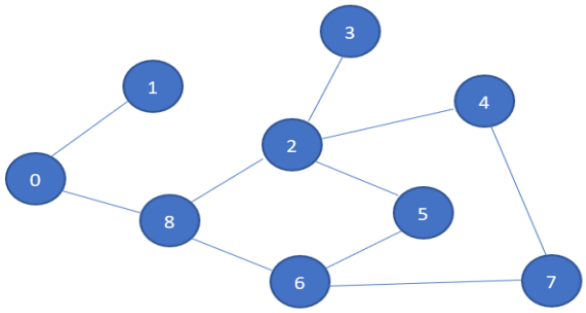
Amirali Farazmand

99522329

الف) برای گراف زیر الگوریتم DFS را پیاده سازی کنید . دقت کنید که لازم است مرحله به مرحله الگوریتم را توضیح دهید و محتوای ساختمان داده ای که در حل سوال از آن استفاده می‌کنید را مرحله به مرحله مشخص کنید . عملیات را از راس شماره صفر آغاز کنید .



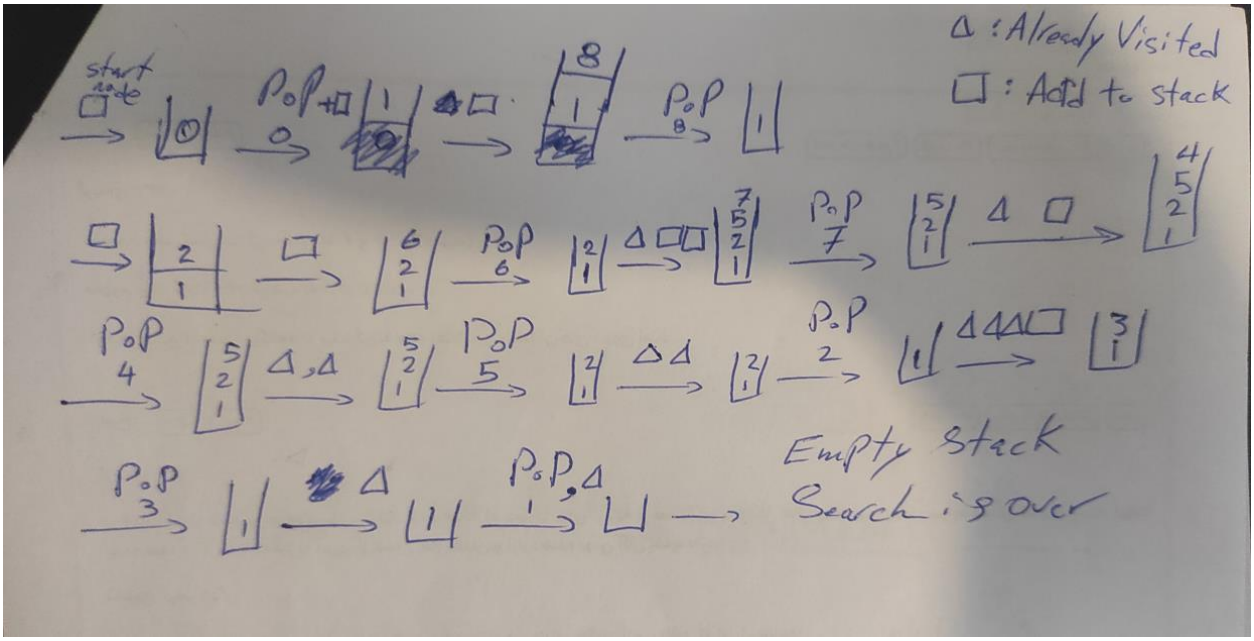
از iterative DFS استفاده شده است که در آن 0 را در استک خود میگذاریم.همچنین آرایه ای بطول نود هایمان داریم که اگر نودی را ویزیت کردیم 1 میگذاریم وگرنه 0.

در یک لوپ باید از استک نود وردارایم.ویزیت مربوط به آن را 1 میکنیم و از همه همسایه هایش اگر تا حالا ویزیت نکرده باشیمش در استک میگذاریم وگرنه کاری رویشان انجام نمیدهیم. هر وقت که همه همسایه های نودی را طی کردیم به سراغ نود بعدی داخل استک میرویم. لوپ تا موقعی که استک خالی نشده باشد باید ادامه پیدا کند.

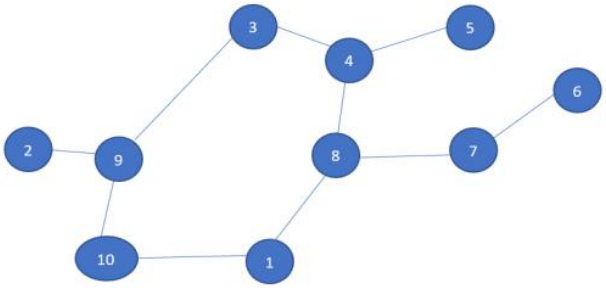
مرحله به مرحله در عکس مشخص کردم.

```cpp
struct Edge
{
    int src, dest;
};

class Graph
{
public:
    vector<vector<int>> adjList;

    Graph(vector<Edge> const &edges, int n)
    {
        adjList.resize(n + 1);

        for (auto &edge : edges)
        {
            adjList[edge.src].push_back(edge.dest);
            adjList[edge.dest].push_back(edge.src);
        }
    }
};
```

```cpp
void iterativeDFS(Graph const &graph, int node, vector<bool> &is_visited)
{
    stack<int> node_stack;
    node_stack.push(node);

    while (!node_stack.empty())
    {
        int node = node_stack.top();       You, 1 second ago • Uncommitted changes
        node_stack.pop();
        if (is_visited[node])
            continue;
        else if (!is_visited[node])
        {
            cout << node +1<< "\n";
            is_visited[node] = true;
        }
        for (int n : graph.adjList[node])
        {
            if (!is_visited[n])
            {
                node_stack.push(n);
            }
        }
    }
}
```


Δ : Already Visited
□ : Add to stack
start node
Pop+□
Pop
Empty Stack
Search is over

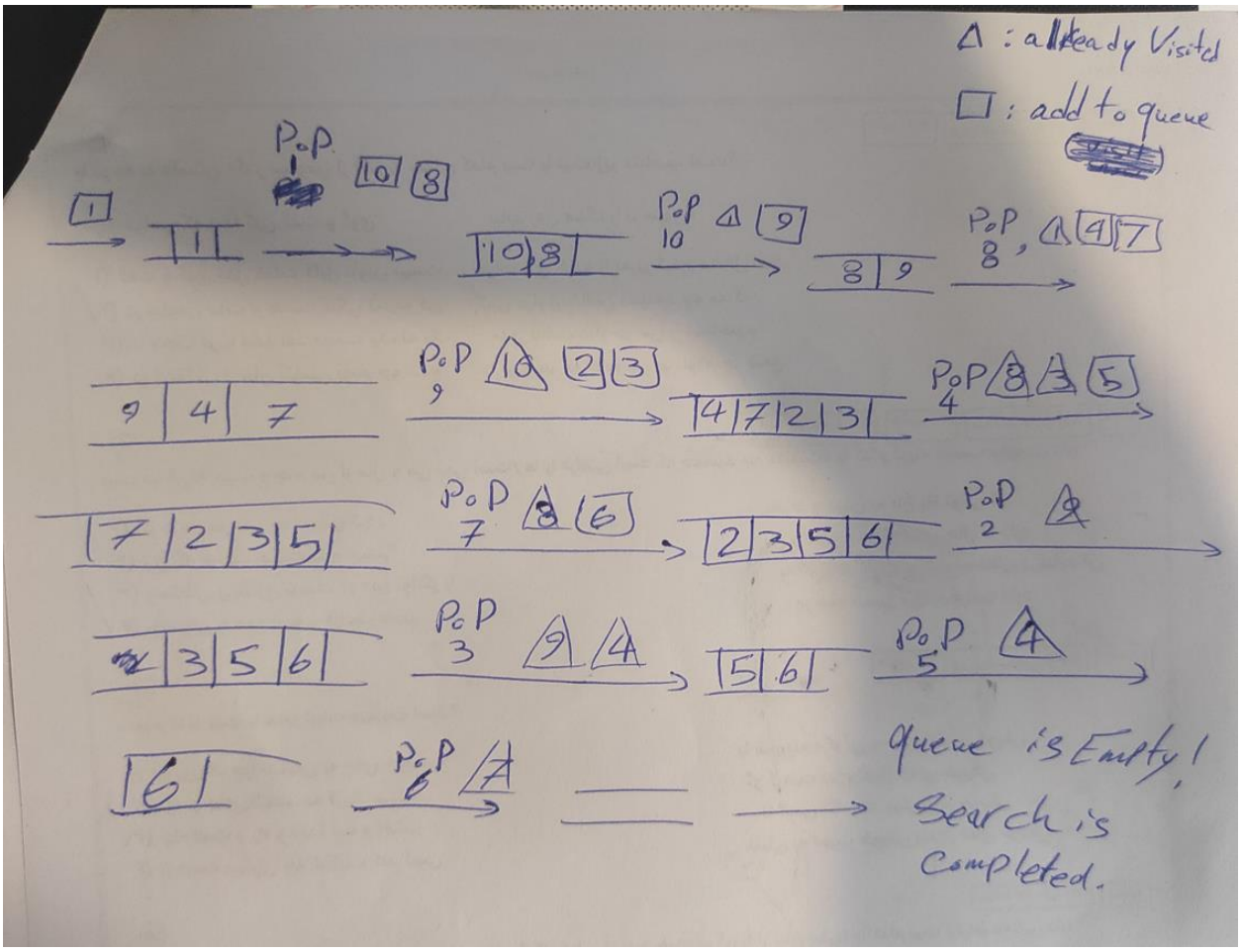
ب) این بار برای گراف زیر الگوریتم BFS را پیاده سازی کنید . تمامی نکاتی که در بخش قبل گفته شده بود ، برای این بخش نیز باید رعایت شوند . عملیات را از راس شماره ۱ آغاز کنید .



در این بخش از iterative BFS استفاده شده است. از ساختار صف استفاده میکنیم.مثل بخش قبل در آرایه ای بولین ذخیره میکنیم که نود ها را ویزیت کردیم یا نه. نود 1 را داخل صف میگذاریم.در یک لوپ در هر مرحله نود را از صف بیرون میکشیم و برای همه ی همسایه هایش چک میکنیم که اگر قبلا ویزیت شدند کاریشان نمیکنیم ولی اگر نشده باشند خانه مربوطه آن را در آرایه درست میکنیم و به صف اضافه اش میکنیم. این کار را تا وقتی صف خالی شود ادامه میدهیم، وقتی صف خالی باشد یعنی ما همه نود ها را بررسی کردیم.

در شکل مراحل این مثال نشان داده شده اند.

```
void iterativeBFS( int start_node, int v)
{
    queue<int> q;
    vector<bool> visited(v + 1, false);

    q.push(start_node);
    visited[start_node] = true;
    while (!q.empty())
    {
        int node = q.front();
        q.pop();
        cout << node << endl;
        for (int n : neighbours_vector[node])
        {
            if (!visited[n])
            {
                visited[n] = true;
                q.push(n);
            }
        }
    }
      You, 1 second ago • Uncommitted changes
}
```


△ : already Visited
□ : add to queue
P.P 1
10 8
1
11
10 8
P.P 10
△ 9
8 9
P.P 8, △ 4 7
9 4 7
P.P 9
△ 10 2 3
4 7 2 3
P.P 4
△ 8 △ 5
7 2 3 5
P.P 7
△ 8 6
2 3 5 6
P.P 2
△
3 5 6
P.P 3
△ 9 △ 4
5 6
P.P 5
△ 4
6
P.P 6
△
Queue is Empty!
Search is completed.

ج) پیچیدگی زمانی و پیچیدگی حافظه دو الگوریتم بالا را بیان کنید . همچنین تفاوت این الگوریتم‌ها از نظر ساختمان داده برای پیاده‌سازی و دلیل آن (برای مثال اگر از صف استفاده می‌شود ، چرا از پشته استفاده نمی‌شود و ...) بیان کنید . اگر بدانیم راس نهایی که دنبال آن می‌گردیم به راس شروع نزدیک‌تر است ، استفاده از کدام روش را پیشنهاد می‌کنید؟ در حالت برعکس چه طور (یعنی بدانیم راس نهایی در فاصله نسبتا زیادی نسبت به راس اولیه قرار دارد) ؟ دلیل خود را بیان کنید .

اگر از صف استفاده شود یعنی ما نودی که زودتر دیده ایم را زودتر میرویم سراغش و همسایه های او را بررسی میکنیم. اما در استک وقتی ما نودی را زودتر از همسایه های دیگرش میبینیم دیرتر می اییم سراغش و وقتی کار همه ی آنها تمام شد این نود بررسی میشود. در مثال گفته شده که راس نهایی نزدیک راس شروع است بهتر است bfs و در حالت مخالفش بهتر است dfs زد چرا که bfs نود ها را گویا به ترتیب عمق نسبت به نقطه شروع میپیماید، برخلافش در dfs از نود اولیه تا جایی که عمق دارد هم چه بسا پیش میرود.

مرتبه زمانی هردو برابر با هم است چرا که ما نود ها را ذخیره میکنیم.

O(V) .

همچنین مرتبه زمانی هردو برابر با O(V+E) هست به ازای V برابر تعداد راس و E برابر تعداد یال.