Санкт-Петербургский политехнический университет Петра Великого
Институт машиностроения, материалов и транспорта

Курсовая работа

Дисциплина: Объектно-ориентированное программирование

Тема: Обход графа в ширину и в глубину

Выполнил студент группы 3331506/90401: Ильясов А.Е.

Преподаватель: Ананьевский М.С.

«___»_____2022 г.

Санкт-Петербург

2022

# 1. Введение

Существует ряд задач, где нужно обойти некоторый граф в глубину или в ширину, так, чтобы посетить каждую вершину один раз. При этом посетить вершины дерева означает выполнить какую-то операцию. Обход графа — это поэтапное исследование всех вершин графа.

Для решения таких задач используются два основных алгоритма:

- Поиск в ширину (*breadth-first search* или *BFS*)
- Поиск в глубину (*depth-first search* или *DFS*)

# 2. Описание алгоритма поиска в ширину

Поиск в ширину подразумевает поуровневое исследование графа:

1. Вначале посещается корень – произвольно выбранный узел.
2. Затем – все потомки данного узла.
3. После этого посещаются потомки потомков и т.д. пока не будут исследованы все вершины.

Вершины просматриваются в порядке роста их расстояния от корня.

Алгоритм поиска в ширину работает как на ориентированных, так и на неориентированных графах.

Для реализации алгоритма удобно использовать очередь.

Рассмотрим работу алгоритма на примере графа на рисунке 1.

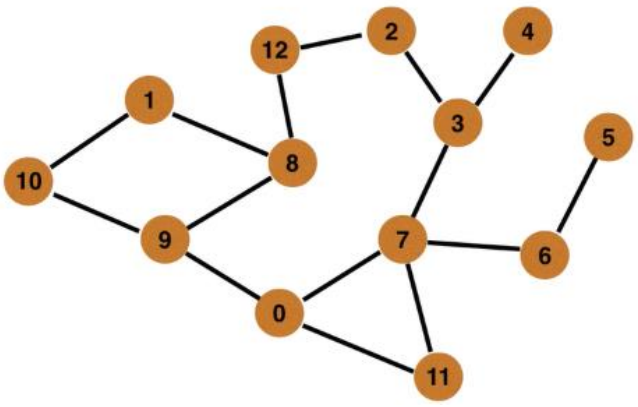


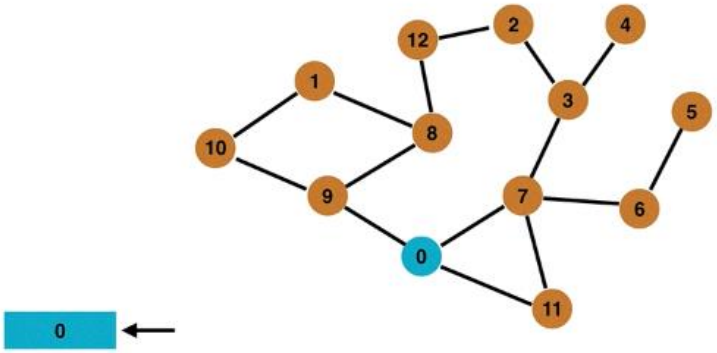

Рисунок 1. Граф для обхода

Каждая вершина может находиться в одном из 3 состояний:

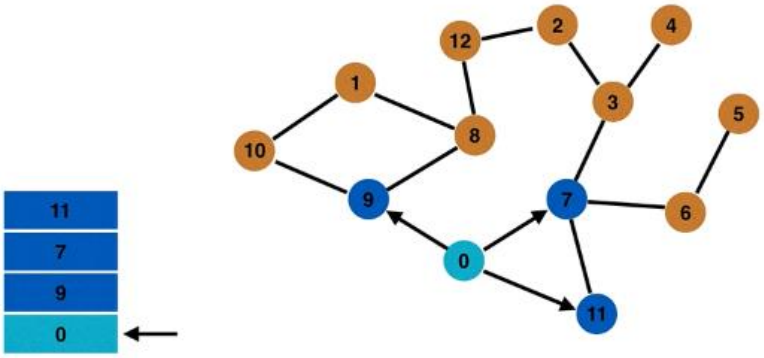
- 0 — коричневый – необнаруженная вершина;
- 1 — синий – обнаруженная, но не посещенная вершина;
- 2 — серый – обработанная вершина.

Голубой – рассматриваемая вершина.

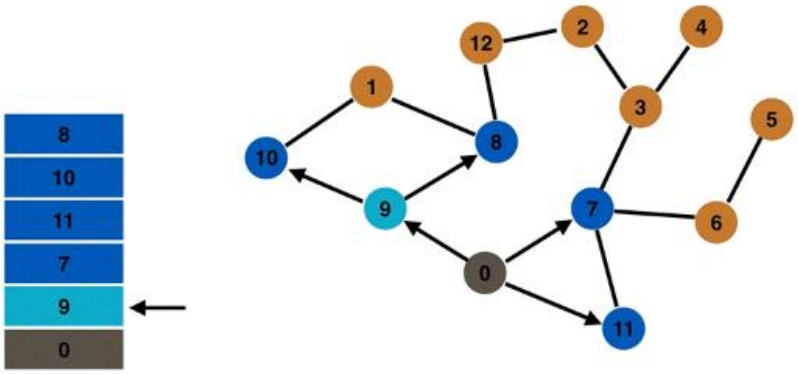
Шаг 1. Добавляем в очередь нулевую вершину.



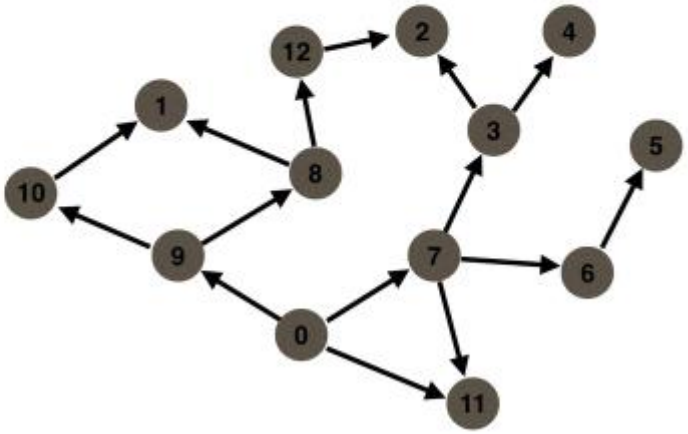

Шаг 2. Добавляем в очередь все вершины, смежные с нулевой вершиной.




Шаг 3. Добавляем в очередь все вершины, смежные с вершиной, находящейся следующей в очереди.



Шаг 4 и далее. Повторить шаг 3 до тех пор, пока в очереди есть непосещенные вершины.


В результате работы алгоритма получаем просмотр каждой вершины графа один раз.

Применения алгоритма поиска в ширину

- Поиск кратчайшего пути в невзвешенном графе (ориентированном или неориентированном).
- Поиск компонент связности.
- Нахождения решения какой-либо задачи (игры) с наименьшим числом ходов.
- Найти все рёбра, лежащие на каком-либо кратчайшем пути между заданной парой вершин.
- Найти все вершины, лежащие на каком-либо кратчайшем пути между заданной парой вершин.

Псевдокод алгоритма поиска в ширину:

```
BFS(start_node) {
 for(all nodes i) visited[i] = false; // изначально список посещённых узлов
                                         пуст
 queue.push(start_node);              // начиная с узла-источника
 visited[start_node] = true;
 while(! queue.empty() ) {            // пока очередь не пуста
  node = queue.pop();                 // извлечь первый элемент в очереди
  foreach(child in expand(node)) {    // все преемники текущего узла
   if(visited[child] == false) {      // ... которые ещё не были посещены
    queue.push(child);                // ... добавить в конец очереди...
    visited[child] = true;            // ... и пометить как посещённые
   }
  }
 }
}
```

## 3. Исследование алгоритма поиска в ширину

Время выполнения BFS составляет O(V+E), а посколькумы используем очередь, вмещающую все вершины, его пространственная сложность составляет O(V). V — общее количество вершин. E — общее количество граней (ребер).

## 4. Описание алгоритма поиска в глубину

Стратегия поиска в глубину, как и следует из названия, состоит в том, чтобы идти «вглубь» графа, насколько это возможно.

1. Двигаемся из начальной вершины.
2. Движемся в произвольную смежную вершину.
3. Из этой вершины обходим все возможные пути до смежных вершин.
4. Если таких путей нет или мы не достигли конечной вершины, то возвращаемся назад к вершине с несколькими исходящими ребрами и идем по другому пути.
5. Алгоритм повторяется, пока не будут исследованы все вершины.

Алгоритм поиска в глубину работает как на ориентированных, так и на неориентированных графах.

Для реализации алгоритма удобно использовать стек или рекурсию.

Рассмотрим работу алгоритма на примере графа на рисунке 2.

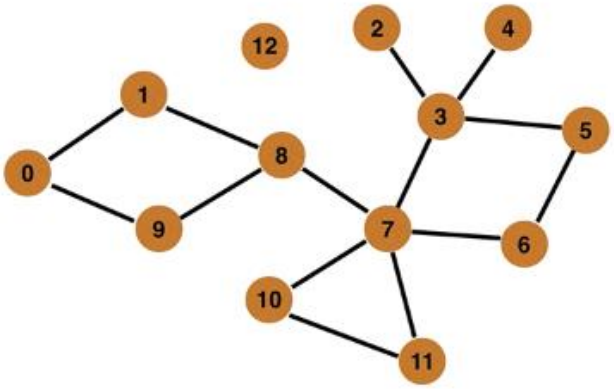


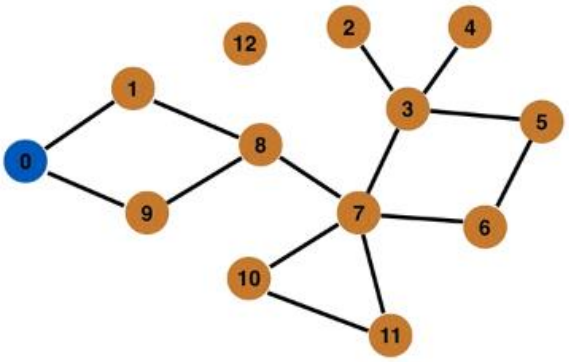

Рисунок 2. Граф для обхода

Каждая вершина может находиться в одном из 3 состояний:

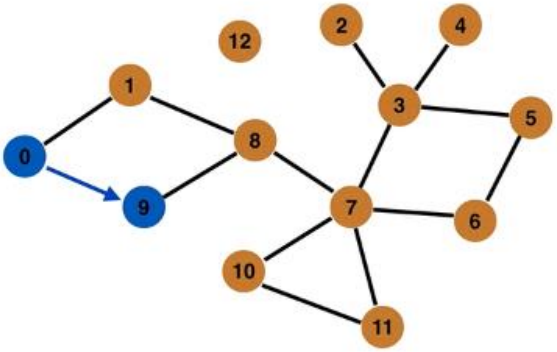
- 0 — коричневый – необнаруженная вершина;
- 1 — синий – обнаруженная, но не посещенная вершина;
- 2 — серый – обработанная вершина.

Голубой – рассматриваемая вершина.

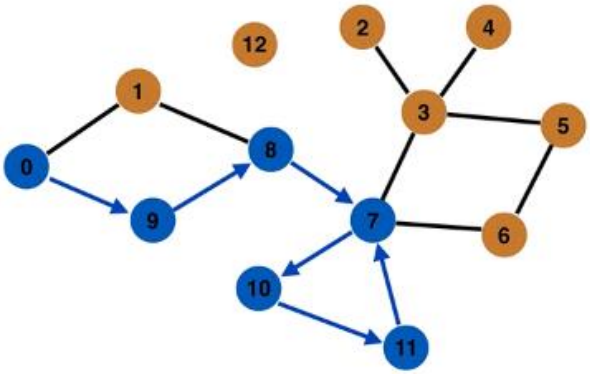
Шаг 1. Начинаем поиск с произвольной (нулевой) вершины.



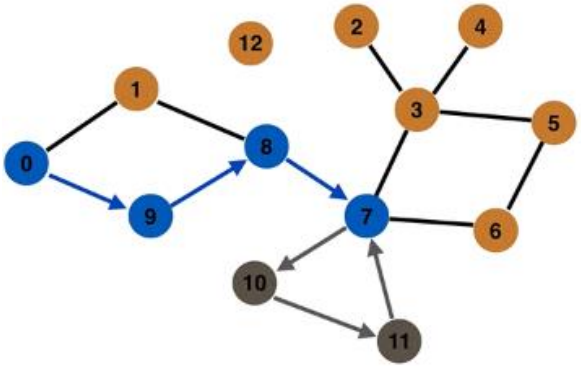
Шаг 2. Переходим к смежной ближайшей вершине.



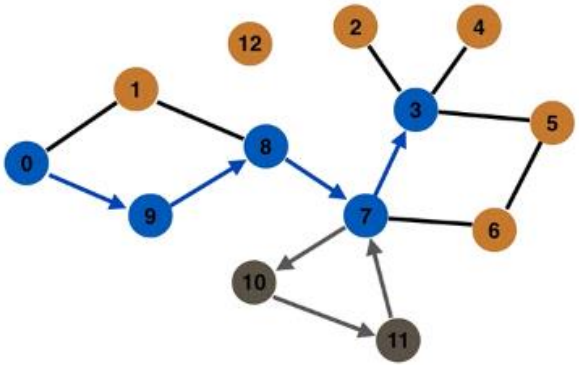

Шаг 3 – Шаг 6. Повторяем шаг 2 до тех пор, пока есть куда двигаться



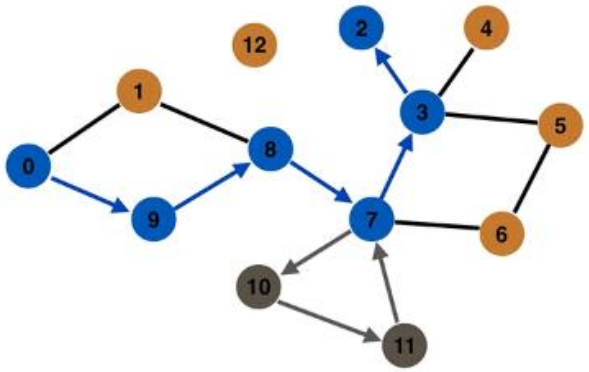

Шаг 7. Возвращаемся в ближайшую вершину с разветвлениями.



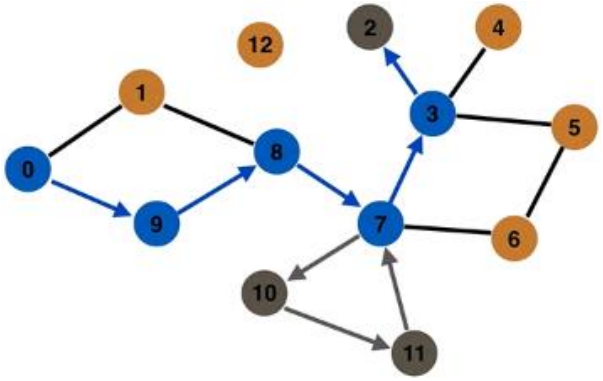

Шаг 8. Переходим к смежной ближайшей вершине (по другому пути).



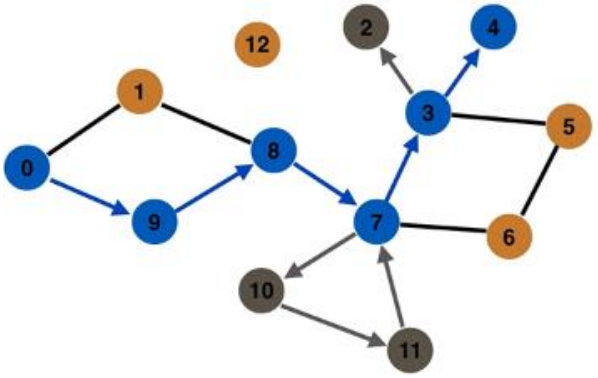
Шаг 9. Повторяем шаг 8 до тех пор, пока есть куда двигаться



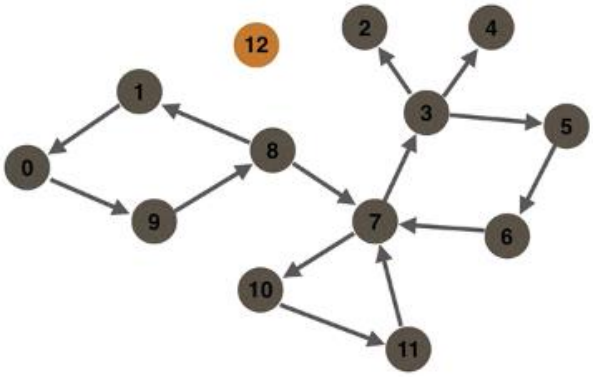

Шаг 10. Возвращаемся в ближайшую вершину с разветвлениями.




Шаг 11. Переходим к смежной ближайшей вершине (по другому пути).




Шаг 11. Повторяем алгоритм до тех пор, пока есть непосещенные вершины.



В результате работы алгоритма получаем просмотр каждой вершины графа один раз.

Применения алгоритма поиска в глубину:

- Поиск любого пути в графе.
- Поиск лексикографически первого пути в графе.
- Проверка, является ли одна вершина дерева предком другой.
- Поиск наименьшего общего предка.
- Топологическая сортировка.
- Поиск компонент связности.

Псевдокод алгоритма поиска в глубину:

```
function doDfs(G[n]: Graph): // функция принимает граф G с количеством вершин n и выполняет обход в глубину во всем графе
   visited = array[n, false]  // создаём массив посещённых вершины длины n, заполненный false изначально

   function dfs(u: int):
      visited[u] = true
      for v: (u, v) in G
         if not visited[v]
            dfs(v)

   for i = 1 to n
      if not visited[i]
         dfs(i)
```

## 5. Исследование алгоритма поиска в глубину

Оценим время работы обхода в глубину. Процедура dfs вызывается от каждой вершины не более одного раза, а внутри процедуры рассматриваются все ребра. Всего таких ребер для всех вершин в графе O(E), следовательно, время работы алгоритма оценивается как O(V+E).

Список литературы


1. Хайнеман, Д. Алгоритмы. Справочник. С примерами на C, C++, Java иPython /Д. Хайнеман, Г. Поллис, С. Селков. – Вильямс, 2017.
2. Седжвик Роберт. Фундаментальные алгоритмы на C++. Анализ/Структуры данных/Сортировка/Поиск: Пер. с англ./Роберт Седжвик.- Издательство «ДиаСофт», 2001.
3. https://prog-cpp.ru/data-graph/

Приложение 1. Поиск в ширину (BFS)

```cpp
#include <iostream>
#include <queue> // очередь
using namespace std;

int main()
{
    queue<int> Queue;
    int arr[7][7] = { { 0, 1, 1, 0, 0, 0, 1 }, // матрица смежности
                      { 1, 0, 1, 1, 0, 0, 0 },
                      { 1, 1, 0, 0, 0, 0, 0 },
                      { 0, 1, 0, 0, 1, 0, 0 },
                      { 0, 0, 0, 1, 0, 1, 0 },
                      { 0, 0, 0, 0, 1, 0, 1 },
                      { 1, 0, 0, 0, 0, 1, 0 } };
    int nodes[7] = { 0 }; // вершины графа (0 - все вершины не рассмотрены)
    Queue.push(0); // помещаем в очередь первую вершину
    while (!Queue.empty()) // пока очередь не пуста
    {
        int node = Queue.front(); // извлекаем вершину
        Queue.pop();
        nodes[node] = 2; // отмечаем ее как посещенную
        for (int j = 0; j < 7; j++) // проверяем для нее все смежные вершины
        {
            if (arr[node][j] == 1 && nodes[j] == 0) // если вершина смежная и не обнаружена
            {
                Queue.push(j); // добавляем ее в очередь
                nodes[j] = 1; // отмечаем вершину как обнаруженную
            }
        }
        cout << node << endl; // выводим номер вершины
    }
    return 0;
}
```

Приложение 2. Поиск в глубину (DFS) с использованием рекурсии

```
#include <iostream>
using namespace std;

int mas[7][7] = { { 0, 1, 1, 0, 0, 0, 1 }, // матрица смежности
                  { 1, 0, 1, 1, 0, 0, 0 },
                  { 1, 1, 0, 0, 0, 0, 0 },
                  { 0, 1, 0, 0, 1, 0, 0 },
                  { 0, 0, 0, 1, 0, 1, 0 },
                  { 0, 0, 0, 0, 1, 0, 1 },
                  { 1, 0, 0, 0, 0, 1, 0 } };

int nodes[7] = { 0 }; // вершины графа (исходно все вершины непосещенные)

void search(int st, int n)
{
    cout << st << " ";
    nodes[st] = 1;
    for (int r = 0; r < n; r++)
        if ((mas[st][r] == 1) && (nodes[r] == 0))
            search(r, n);
}

int main()
{
    search(0, 7);
    return 0;
}
```