Обход в ширину (BFS)

Denis Bakin

Мотивация: зачем нужен BFS?

Задача: дан невзвешенный граф и стартовая вершина s. Найти кратчайшие расстояния от s до всех остальных вершин

Мотивация: зачем нужен BFS?

Задача: дан невзвешенный граф и стартовая вершина s. Найти кратчайшие расстояния от s до всех остальных вершин

Может ли DFS решить эту задачу?

- DFS идёт «вглубь» — может найти длинный путь раньше короткого
- DFS **не гарантирует** кратчайшие пути
- Нужен другой порядок обхода

BFS: обход «по слоям»

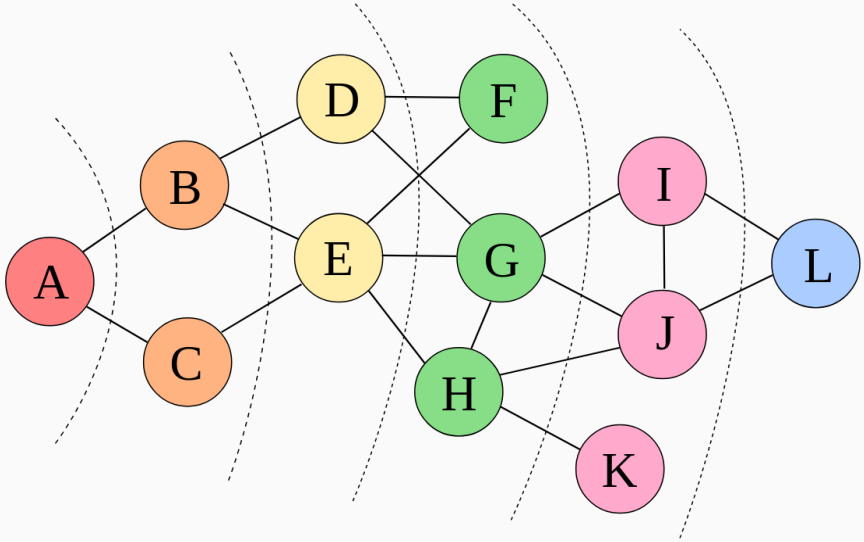
Поиск в ширину (BFS, Breadth-First Search) — алгоритм обхода графа, который посещает вершины **по увеличению расстояния** до стартовой вершины

BFS: обход «по слоям»

Поиск в ширину (BFS, Breadth-First Search) — алгоритм обхода графа, который посещает вершины **по увеличению расстояния** до стартовой вершины

Расстояние	Вершины
0	s
1	все соседи s
2	все соседи соседей s (ещё не посещённые)
...	...
k	все вершины на расстоянии k от s

BFS: обход «по слоям»



BFS: реализация

1. Создаём очередь, помещаем в неё стартовую вершину s
2. Инициализируем массив расстояний: $d[s] = 0$, остальные $d[v] = -1$

BFS: реализация

1. Создаём очередь, помещаем в неё стартовую вершину s
2. Инициализируем массив расстояний: $d[s] = 0$, остальные $d[v] = -1$
3. Пока очередь не пуста — достаём вершину v из головы очереди и просматриваем всех её соседей u. Если $d[u] = -1$, то присваиваем $d[u] = d[v] + 1$ и добавляем u в конец очереди

BFS: реализация

1. Создаём очередь, помещаем в неё стартовую вершину s
2. Инициализируем массив расстояний: $d[s] = 0$, остальные $d[v] = -1$
3. Пока очередь не пуста — достаём вершину v из головы очереди и просматриваем всех её соседей u. Если $d[u] = -1$, то присваиваем $d[u] = d[v] + 1$ и добавляем u в конец очереди

Значение $d[v] = -1$ одновременно означает «не посещена», поэтому отдельный массив `visited` не нужен

BFS: реализация

```
std::vector<std::vector<int>> adj(n);
std::vector<int> dist(n, -1);  // -1 означает «не посещена»

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

Пример работы BFS

Сложность BFS

Сложность BFS

Временная сложность: $O(n + m)$

Сложность BFS

Временная сложность: $O(n + m)$

- Каждая вершина добавляется в очередь **ровно один раз** (проверка `dist[u] == -1`)
- Каждое ребро просматривается не более двух раз (для неориентированного графа)

Сложность BFS

Временная сложность: $O(n + m)$

- Каждая вершина добавляется в очередь **ровно один раз** (проверка `dist[u] == -1`)
- Каждое ребро просматривается не более двух раз (для неориентированного графа)

Пространственная сложность: $O(n)$

Сложность BFS

Временная сложность: $O(n + m)$

- Каждая вершина добавляется в очередь **ровно один раз** (проверка `dist[u] == -1`)
- Каждое ребро просматривается не более двух раз (для неориентированного графа)

Пространственная сложность: $O(n)$

- Массив `dist` размера n
- Очередь содержит не более n вершин

Восстановление кратчайшего пути

Восстановление кратчайшего пути

Заведём массив предков `parent`:

- `parent[v]` — вершина, из которой мы пришли в v при BFS
- `parent[s] = -1` (у стартовой вершины нет предка)

При обнаружении нового соседа u из вершины v:

$$parent[u] = v$$

Рёбра на кратчайшем пути

Задача: найти все рёбра, лежащие на каком-либо кратчайшем пути между a и b

Рёбра на кратчайшем пути

Задача: найти все рёбра, лежащие на каком-либо кратчайшем пути между a и b

Алгоритм:

1. Запустить BFS из $a \to$ массив расстояний d_a
2. Запустить BFS из $b \to$ массив расстояний d_b

Рёбра на кратчайшем пути

Задача: найти все рёбра, лежащие на каком-либо кратчайшем пути между a и b

Алгоритм:

1. Запустить BFS из $a \to$ массив расстояний d_a
2. Запустить BFS из $b \to$ массив расстояний d_b
3. Ребро (u, v) лежит на кратчайшем пути $\iff$

$$d_a[u] + 1 + d_b[v] = d_a[b]$$

Рёбра на кратчайшем пути

Задача: найти все рёбра, лежащие на каком-либо кратчайшем пути между a и b

Алгоритм:

1. Запустить BFS из $a \to$ массив расстояний d_a
2. Запустить BFS из $b \to$ массив расстояний d_b
3. Ребро (u, v) лежит на кратчайшем пути $\iff$

$$d_a[u] + 1 + d_b[v] = d_a[b]$$

Сложность: $O(n + m)$ — два запуска BFS

Неявные графы: мотивация

Граф не всегда задан списком смежности. Часто вершины и рёбра определяются **правилами**

Неявные графы: мотивация

Граф не всегда задан списком смежности. Часто вершины и рёбра определяются **правилами**

Пример: булева матрица $n \times m$

- Клетка (x, y) свободна или занята
- За один шаг можно перейти в соседнюю свободную клетку (по горизонтали или вертикали)
- **Задача:** найти кратчайший путь от (x_s, y_s) до (x_t, y_t)

Неявные графы: мотивация

Граф не всегда задан списком смежности. Часто вершины и рёбра определяются **правилами**

Пример: булева матрица $n \times m$

- Клетка (x, y) свободна или занята
- За один шаг можно перейти в соседнюю свободную клетку (по горизонтали или вертикали)
- **Задача:** найти кратчайший путь от (x_s, y_s) до (x_t, y_t)

Можно построить явный граф (пронумеровать клетки как $x \cdot m + y$), но проще адаптировать сам BFS

0-1 BFS: мотивация

Задача: граф с рёбрами веса 0 или 1. Найти кратчайшие расстояния от s

0-1 BFS: мотивация

Задача: граф с рёбрами веса 0 или 1. Найти кратчайшие расстояния от s

Обычный BFS считает все рёбра одинаковыми — не подходит

0-1 BFS: мотивация

Задача: граф с рёбрами веса 0 или 1. Найти кратчайшие расстояния от s

Обычный BFS считает все рёбра одинаковыми — не подходит

Ключевое наблюдение: если от a до b ведёт путь из нулевых рёбер, то $d[a] = d[b]$

0-1 BFS: идея

Если в графе оставить только 0-рёбра, он распадётся на **компоненты связности** с одинаковым расстоянием

0-1 BFS: идея

Если в графе оставить только 0-рёбра, он распадётся на **компоненты связности** с одинаковым расстоянием

Вернём 1-рёбра — они соединяют эти компоненты

$\rightarrow$ задача сводится к обычному BFS по компонентам

0-1 BFS: идея

Если в графе оставить только 0-рёбра, он распадётся на **компоненты связности** с одинаковым расстоянием

Вернём 1-рёбра — они соединяют эти компоненты

$\rightarrow$ задача сводится к обычному BFS по компонентам

Реализация: заменим очередь на **дек** (deque)

- Ребро веса 0 $\rightarrow$ сосед добавляется в **начало** дека
- Ребро веса 1 $\rightarrow$ сосед добавляется в **конец** дека

Это гарантирует обработку вершин в порядке неубывания расстояния

Итоги: алгоритм BFS

- Обход графа в ширину с помощью **очереди**
- Посещает вершины в порядке неубывания расстояния от старта
- Находит **кратчайшие расстояния** в невзвешенном графе
- Сложность: $O(n + m)$