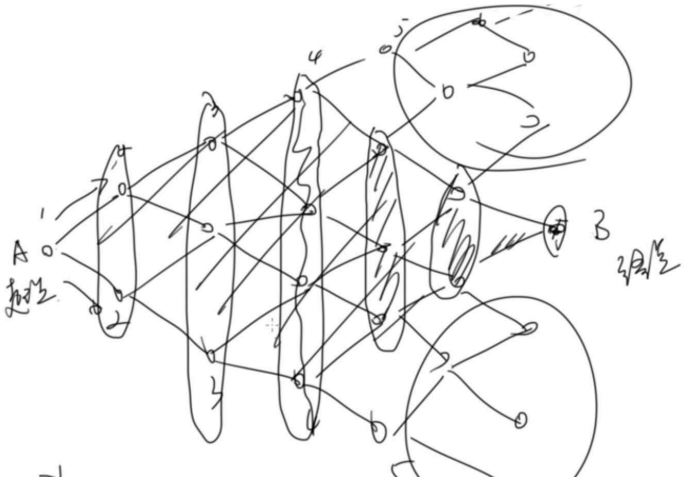

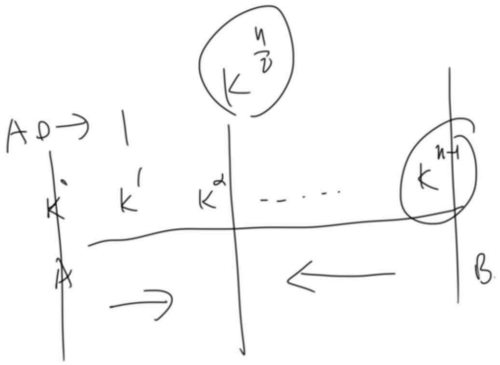

双向宽度优先搜索算法

主讲人 令狐冲

双向宽度优先搜索代码模板


```
初始化 forward_queue 和 forward_set，加入起点
初始化 backward_queue 和 backward_set，加入终点

distance = 0
while forward_queue 和 backward_queue 都非空
    distance += 1
    for 所有 forward_queue 里的点
        拓展出下一层的点放到 forward_queue 和 forward_set 里
        如果碰到了 backward_set 里的点则 return distance
    distance += 1
    for 所有 backward_queue 里的点
        拓展出下一层的点放到 backward_queue 和 backward_set 里
        如果碰到了 forward_set 里的点则 return distance

return 找不到
```

跳马问题

https://www.lintcode.com/problem/knight-shortest-path

马从棋盘上的起点跳到终点需要最少花多少步？

我们带着大家一起敲一下代码

Python 代码 - 入口函数


```python
def shortestPath(self, grid, source, destination):
    if not grid or not grid[0]:
        return -1
    if grid[destination.x][destination.y]:
        return -1
    if (source.x, source.y) == (destination.x, destination.y):
        return 0

    forward_queue = deque([(source.x, source.y)])
    forward_set = set([(source.x, source.y)])
    backward_queue = deque([(destination.x, destination.y)])
    backward_set = set([(destination.x, destination.y)])

    distance = 0
    while forward_queue and backward_queue:
        distance += 1
        if self.extend_queue(grid, forward_queue, forward_set, backward_set):
            return distance
        distance += 1
        if self.extend_queue(grid, backward_queue, backward_set, forward_set):
            return distance

    return -1
```

```python
DIRECTIONS = (
    (1, 2),
    (-1, 2),
    (2, 1),
    (-2, 1),
    (-1, -2),
    (1, -2),
    (-2, -1),
    (2, -1),
)
```

Python 代码 - extend_queue and is_valid


```python
def extend_queue(self, queue, visited, opposite_visited, grid):
    for _ in range(len(queue)):
        x, y = queue.popleft()
        for dx, dy in DIRECTIONS:
            new_x, new_y = (x + dx, y + dy)
            if not self.is_valid(new_x, new_y, grid, visited):
                continue
            if (new_x, new_y) in opposite_visited:
                return True
            queue.append((new_x, new_y))
            visited.add((new_x, new_y))

    return False
```

```python
def is_valid(self, x, y, grid, visited):
    if x < 0 or x >= len(grid):
        return False
    if y < 0 or y >= len(grid[0]):
        return False
    if grid[x][y]:
        return False
    if (x, y) in visited:
        return False
    return True
```

Java 代码


```java
int[][] DIRECTIONS = {
    {1, 2},
    {-1, 2},
    {2, 1},
    {-2, 1},
    {-1, -2},
    {1, -2},
    {-2, -1},
    {2, -1},
};

public int shortestPath(boolean[][] grid,
                        Point source,
                        Point destination) {
    if (grid == null || grid.length == 0) {
        return -1;
    }
    if (grid[0] == null || grid[0].length == 0) {
        return -1;
    }
    if (source.x == destination.x && source.y == destination.y) {
        return 0;
    }
    if (grid[destination.x][destinatio.y]) {
        return -1;
    }

    Queue<Point> forwardQueue = new LinkedList<Point>();
    Queue<Point> backwardQueue = new LinkedList<Point>();
    int n = grid.length;
    int m = grid[0].length;
    boolean[][] forwardSet = new boolean[n][m];
    boolean[][] backwardSet = new boolean[n][m];
```

```java
forwardQueue.offer(new Point(source.x, source.y));
backwardQueue.offer(new Point(destination.x, destination.y));
forwardSet[source.x][source.y] = true;
backwardSet[destination.x][destination.y] = true;

int distance = 0;
while (!forwardQueue.isEmpty() && !backwardQueue.isEmpty()) {
    distance++;
    if (extendQueue(forwardQueue, forwardSet, backwardSet, grid)) {
        return distance;
    }

    distance++;
    if (extendQueue(backwardQueue, backwardSet, forwardSet, grid)) {
        return distance;
    }
}

return -1;
```

Java 代码 - extendQueue & isValid


```java
boolean extendQueue(Queue<Point> queue,
                    boolean[][] visited,
                    boolean[][] oppositeVisited,
                    boolean[][] gird) {
    int queueLength = queue.size();
    for (int i = 0; i < queueLength; i++) {
        int x = queue.peek().x;
        int y = queue.poll().y;
        for (int j = 0; j < 8; j++) {
            int newX = x + DIRECTIONS[j][0];
            int newY = y + DIRECTIONS[j][1];
            if (!isValid(newX, newY, gird, visited)) {
                continue;
            }
            if (oppositeVisited[newX][newY]) {
                return true;
            }
            queue.offer(new Point(newX, newY));
            visited[newX][newY] = true;
        }
    }

    return false;
}
```

```java
boolean isValid(int x,
                int y,
                boolean[][] grid,
                boolean[][] visited) {
    if (x < 0 || x >= grid.length) {
        return false;
    }
    if (y < 0 || y >= grid[0].length) {
        return false;
    }
    if (grid[x][y]) {
        return false;
    }
    if (visited[x][y]) {
        return false;
    }

    return true;
}
```

跳马问题（二）

https://www.lintcode.com/problem/knight-shortest-path-ii

马从棋盘上的起点跳到终点需要最少花多少步？

这次马只能向右走

Python 代码


```python
def shortestPath2(self, grid):
    if not grid or not grid[0]:
        return -1

    n, m = len(grid), len(grid[0])
    if grid[n - 1][m - 1]:
        return -1
    if n * m == 1:
        return 0

    forward_queue = collections.deque([(0, 0)])
    forward_set = set([(0, 0)])
    backward_queue = collections.deque([(n - 1, m - 1)])
    backward_set = set([(n - 1, m - 1)])

    distance = 0
    while forward_queue and backward_queue:
        distance += 1
        if self.extend_queue(forward_queue, FORWARD_DIRECTIONS, forward_set, backward_set, grid):
            return distance

        distance += 1
        if self.extend_queue(backward_queue, BACKWARD_DIRECTIONS, backward_set, forward_set, grid):
            return distance

    return -1
```

```python
FORWARD_DIRECTIONS = (
    (1, 2),
    (-1, 2),
    (2, 1),
    (-2, 1),
)

BACKWARD_DIRECTIONS = (
    (-1, -2),
    (1, -2),
    (-2, -1),
    (2, -1),
)
```

Python 代码 - extend_queue and is_valid


```python
def extend_queue(self, queue, directions, visited, opposite_visited, grid):
    for _ in range(len(queue)):
        x, y = queue.popleft()
        for dx, dy in directions:
            new_x, new_y = (x + dx, y + dy)
            if not self.is_valid(new_x, new_y, grid, visited):
                continue
            if (new_x, new_y) in opposite_visited:
                return True
            queue.append((new_x, new_y))
            visited.add((new_x, new_y))

    return False
```

```python
def is_valid(self, x, y, grid, visited):
    if x < 0 or x >= len(grid):
        return False
    if y < 0 or y >= len(grid[0]):
        return False
    if grid[x][y]:
        return False
    if (x, y) in visited:
        return False
    return True
```

Java 代码


```java
int[][] FORWARD_DIRECTIONS = {
    {1, 2},
    {-1, 2},
    {2, 1},
    {-2, 1}
};
int[][] BACKWARD_DIRECTIONS = {
    {-1, -2},
    {1, -2},
    {-2, -1},
    {2, -1}
};

public int shortestPath2(boolean[][] grid) {
    if (grid == null || grid.length == 0) {
        return -1;
    }
    if (grid[0] == null || grid[0].length == 0) {
        return -1;
    }
    int n = grid.length;
    int m = grid[0].length;
    if (grid[n - 1][m - 1] == true) {
        return -1;
    }
    if (n * m == 1) {
        return 0;
    }
```

```java
Queue<Point> forwardQueue = new LinkedList<Point>();
Queue<Point> backwardQueue = new LinkedList<Point>();
boolean[][] forwardSet = new boolean[n][m];
boolean[][] backwardSet = new boolean[n][m];

forwardQueue.offer(new Point(0, 0));
backwardQueue.offer(new Point(n - 1,m - 1));
forwardSet[0][0] = true;
backwardSet[n - 1][m - 1] = true;

int distance = 0;
while (!forwardQueue.isEmpty() && !backwardQueue.isEmpty()) {
    distance++;
    if (extendQueue(forwardQueue, FORWARD_DIRECTIONS, forwardSet, backwardSet, grid)) {
        return distance;
    }

    distance++;
    if (extendQueue(backwardQueue, BACKWARD_DIRECTIONS, backwardSet, forwardSet, grid)) {
        return distance;
    }
}

return -1;
```

Java 代码 - extendQueue & isValid


```java
boolean extendQueue(Queue<Point> queue,
                    int[][] directions,
                    boolean[][] visited,
                    boolean[][] oppositeVisited,
                    boolean[][] grid) {
    int queueLength = queue.size();
    for (int i = 0; i < queueLength; i++) {
        Point head = queue.poll();
        int x = head.x;
        int y = head.y;

        for (int j = 0; j < 4; j++) {
            int newX = x + directions[j][0];
            int newY = y + directions[j][1];
            if (!isValid(newX, newY, grid, visited)) {
                continue;
            }
            if (oppositeVisited[newX][newY] == true) {
                return true;
            }

            queue.offer(new Point(newX,newY));
            visited[newX][newY] = true;
        }
    }

    return false;
}
```

```java
boolean isValid(int x,
                int y,
                boolean[][] grid,
                boolean[][] visited) {
    if (x < 0 || x >= grid.length) {
        return false;
    }
    if (y < 0 || y >= grid[0].length) {
        return false;
    }
    if (grid[x][y]) {
        return false;
    }
    if (visited[x][y]) {
        return false;
    }

    return true;
}
```

单词阶梯

https://www.lintcode.com/problem/word-ladder/

给一个起始单词和终止单词

问起始单词通过几次变换能够变成终止单词

一次变换定义为改变单词一个字母

且变换过程中的单词需要在词典中出现

Python 代码


```python
def ladderLength(self, start, end, wordSet):
    if start == end:
        return 1

    wordSet.add(start)
    wordSet.add(end)
    graph = self.construct_graph(wordSet)

    forward_queue = collections.deque([start])
    forward_set = set([start])
    backward_queue = collections.deque([end])
    backward_set = set([end])

    distance = 1
    while forward_queue and backward_queue:
        distance += 1
        if self.extend_queue(graph, forward_queue, forward_set, backward_set):
            return distance
        distance += 1
        if self.extend_queue(graph, backward_queue, backward_set, forward_set):
            return distance

    return -1
```

```python
def extend_queue(self, graph, queue, visited, opposite_visited):
    for _ in range(len(queue)):
        word = queue.popleft()
        for next_word in graph[word]:
            if next_word in visited:
                continue
            if next_word in opposite_visited:
                return True
            queue.append(next_word)
            visited.add(next_word)
    return False
```

Python 代码 - 构图


```python
def construct_graph(self, wordSet):
    graph = {}
    for word in wordSet:
        graph[word] = self.get_next_words(word, wordSet)
    return graph

def get_next_words(self, word, wordSet):
    next_word_set = set()
    for i in range(len(word)):
        prefix = word[:i]
        suffix = word[i + 1:]
        chars = list('abcdefghijklmnopqrstuvwxyz')
        chars.remove(word[i])
        for char in chars:
            next_word = prefix + char + suffix
            if next_word in wordSet:
                next_word_set.add(next_word)
    return next_word_set
```

Java 代码 - BFS


```java
public int ladderLength(String start, String end, Set<String> wordSet) {
    if (start.equals(end)) {
        return 1;
    }

    HashMap<String, Set<String>> graph;
    Queue<String> forwardQueue = new LinkedList<String>();
    Queue<String> backwardQueue = new LinkedList<String>();
    Set<String> forwardSet = new HashSet<String>();
    Set<String> backwardSet = new HashSet<String>();

    wordSet.add(start);
    wordSet.add(end);
    graph = constructGraph(wordSet);
    forwardQueue.offer(start);
    backwardQueue.offer(end);
    forwardSet.add(start);
    backwardSet.add(end);

    int distance = 1;
    while (!forwardQueue.isEmpty() && !backwardQueue.isEmpty()) {
        distance++;
        if (extendQueue(graph, forwardQueue, forwardSet, backwardSet)) {
            return distance;
        }
        distance++;
        if (extendQueue(graph, backwardQueue, backwardSet, forwardSet)) {
            return distance;
        }
    }

    return -1;
}
```

```java
boolean extendQueue(HashMap<String, Set<String>> graph,
                    Queue<String> queue,
                    Set<String> visited,
                    Set<String> oppositeVisited) {
    int queueLength = queue.size();
    for (int i = 0; i < queueLength; i++) {
        String word = queue.poll();
        Set<String> nextWordSet = graph.get(word);
        for (String nextWord : nextWordSet) {
            if (visited.contains(nextWord)) {
                continue;
            }
            if (oppositeVisited.contains(nextWord)) {
                return true;
            }

            queue.offer(nextWord);
            visited.add(nextWord);
        }
    }

    return false;
}
```

Java 代码 - 构图


```java
HashMap<String, Set<String>> constructGraph(Set<String> wordSet) {
    HashMap<String, Set<String>> graph = new HashMap<String, Set<String>>();
    for (String word : wordSet) {
        graph.put(word, getNextWords(word, wordSet));
    }

    return graph;
}

Set<String> getNextWords(String word, Set<String> wordSet) {
    Set<String> nextWordSet = new HashSet<String>();
    int wordLength = word.length();
    for (int i = 0; i < wordLength; i++) {
        String prefix = word.substring(0, i);
        String suffix = word.substring(i + 1);
        char[] chars = ("abcdefghijklmnopqrstuvwxyz").toCharArray();
        for (int j = 0; j < 26; j++) {
            if (word.charAt(i) == chars[j]) {
                continue;
            }
            String nextWord = prefix + chars[j] + suffix;
            if (wordSet.contains(nextWord)) {
                nextWordSet.add(nextWord);
            }
        }
    }

    return nextWordSet;
}
```

120. Word Ladder - LintCode