Поиск выхода из лабиринта

Все задачи из этого раздела делаются в одном и том же файле (дописываются от простого к сложному). Конечной целью будет написание программы, которая ищет выход из лабиринта, заданного в файле следующим образом: прямоугольная таблица, в которой символ `#` означает стенку, пробел — коридор, буква `s` — начальную позицию. Необходимо найти кратчайший путь из начальной позиции до «дырки» в любой из границ лабиринта, если такой путь существует.
Пример лабиринта, заданного в файле, и кратчайшего пути к выходу для этого лабиринта:

```
############                ############
# # #    #                 # # #    #xx
# #   #### #                 # #xxx####x#
### #    # #                 ###x#xxxx#x#
#   # ##   #                 #xxx# ##xxx#
# # # #### #                 #x# # #### #
#s# #      #                 #s# #      #
############                ############
```

Для хранения символов используем тип `char` (`char A[100][100]` — двумерный массив символов), для чтения символа из файла `f` наиболее привычно будет использовать `fscanf(f, "%c", &c);` где `c` — переменная типа `char`. Чтение повторяйте, пока `fscanf` возвращает 1 (количество прочитанных значений), то есть примерно так:

```
char c;
while (fscanf(f, "%c", &c) == 1)
/* читаем по одному символу до конца файла */
{
    ...
}
```

Вам понадобятся функции работы с очередью `enqueue`, `dequeue`, `is_empty`, написанные ранее.

43	**Чтение лабиринта из файла**	**3 балла**

Пусть в файле записана схема лабиринта в таком же виде, как в примере выше (слева). Необходимо считать её в двумерный массив `char`

`A[100][100]` и при чтении подсчитать реальное количество строк `N` и столбцов `M` таблицы. Символ конца строки обозначается `'\n'`, остальные используемые в примере символы — `'#'`, `'s'`, `' '`. Для проверки того, что программа работает правильно, напечатайте на экран лабиринт из таблицы `char A[100][100]` после того, как чтение закончено.
При чтении определите координаты клетки, в которой находится начальная позиция `s`, и напечатайте их.

44	Обход в ширину	5 баллов

Используя функции `enqueue`, `dequeue`, `is_empty`, выполните обход прочитанного лабиринта в ширину. Напечатайте `YES` и завершите алгоритм, если в процессе обхода вы попали в клетку (i, j), удовлетворяющую условию

```
if (i == 0 || j == 0 || i == N - 1 || j == M - 1)
```

— это условие описывает клетки, лежащие на границе лабиринта. Попав в такую клетку, можно считать, что выход из лабиринта успешно найден. При реализации алгоритма обхода придётся сохранять в очереди пары чисел (координаты клеток). Для сохранения пары чисел можно просто записать их в очередь подряд (дважды вызвав функцию `enqueue`) и аналогичным образом извлекать эти пары чисел из очереди:

```
enqueue(i, j);
...
i = dequeue();
j = dequeue();
```

Чтобы отметить клетку как пройденную, поставьте в неё какой-либо символ, отличный от `' '` и `'#'`. В конце алгоритма напечатайте массив, чтобы посмотреть, какие клетки были посещены при обходе.

45	Поиск пути	5 баллов

Добавьте к поиску в ширину сохранение информации о том, **из какой** клетки мы пришли в текущую (можно хранить обе координаты или же просто номер позиции в очереди). Используя эту информацию, отметьте в массиве кратчайший путь к выходу символами `'x'` и напечатайте массив (должно получиться нечто похожее на правый рисунок).

46	**Обход в глубину**	**5 баллов**

Решите задачу при помощи обхода в глубину (рекурсивная функция).

47	**Сравнение обходов**	**1 балл**

Приведите пример лабиринта, для которого обход в глубину найдёт не кратчайший путь. Всегда ли обход в ширину находит кратчайший путь?