1. **Постановка задачи:**
   1. Генерация Лабиринта
   2. Нахождение выхода из него
2. **Генерация лабиринта**
   1. Генерация взвешенного графа размером NxM
   2. Нахождение минимального остова
   3. Преобразование взвешенного графа в матрицу
3. **Генерация взвешенного графа размером NxM**

На основе размера матрицы NxM генерируется взвешенный граф:  
1) Создается список смежность размером N\*M.

2) Алгоритм проходится по каждой вершине,   
проверяя соседей слева, сверху, справа, снизу.  
3) Добавляет к этой вершине соседей если они существуют.

**4)** **Алгоритма Прима O(M\*logN)**

Алгоритм Прима (англ. Prim's algorithm) — алгоритм поиска минимального остовного дерева (англ. minimum spanning tree, MST) во взвешенном неориентированном связном графе.

Искомый минимальный остов строится постепенно, добавлением в него рёбер по одному. Изначально остов полагается состоящим из единственной вершины (её можно выбрать произвольно). Затем выбирается ребро минимального веса, исходящее из этой вершины, и добавляется в минимальный остов. После этого остов содержит уже две вершины, и теперь ищется и добавляется ребро минимального веса, имеющее один конец в одной из двух выбранных вершин, а другой — наоборот, во всех остальных, кроме этих двух. И так далее, т.е. всякий раз ищется минимальное по весу ребро, один конец которого — уже взятая в остов вершина, а другой конец — ещё не взятая, и это ребро добавляется в остов (если таких рёбер несколько, можно взять любое). Этот процесс повторяется до тех пор, пока остов не станет содержать все вершины (или, что то же самое, n-1 ребро).

В итоге будет построен остов, являющийся минимальным. Если граф был изначально не связен, то остов найден не будет (количество выбранных рёбер останется меньше n-1).

**5) Преобразование взвешенного графа в матрицу**

1) Создание матрицы, состоящий из стен.  
2) Проход по каждой вершине матрицы, помечая путь от этой вершины до связанных с ней.

3) Добавление границ слева, сверху, справа, снизу.  
4) Добавление точек начала(слева сверху) и конца(справа снизу).

**6) Нахождение выхода из лабиринта**

Преобразование матрицы в граф

Нахождение пути выхода

Вывод лабиринта с путем выхода на экран

**7) Преобразование матрицы в граф**

1) Прохождение по каждой клетке.  
2) Если клетка является проходимой, то происходит поиск соседей, смежных с ней.  
3) Добавление соседей к данной вершине.

**8) Обход в ширину O(v + e)**

Поиск в ширину (обход в ширину, breadth-first search) — это один из основных алгоритмов на графах.

В результате поиска в ширину находится путь кратчайшей длины в невзвешенном графе, т.е. путь, содержащий наименьшее число рёбер.

Алгоритм работает за O (v+e), где v — число вершин, e — число рёбер.

На вход алгоритма подаётся заданный граф (невзвешенный), и номер стартовой вершины s. Граф может быть как ориентированным, так и неориентированным, для алгоритма это не важно.

Сам алгоритм можно понимать как процесс "поджигания" графа: на нулевом шаге поджигаем только вершину s. На каждом следующем шаге огонь с каждой уже горящей вершины перекидывается на всех её соседей; т.е. за одну итерацию алгоритма происходит расширение "кольца огня" в ширину на единицу (отсюда и название алгоритма).

Более строго это можно представить следующим образом. Создадим очередь q, в которую будут помещаться горящие вершины, а также заведём булевский массив \rm used[], в котором для каждой вершины будем отмечать, горит она уже или нет (или иными словами, была ли она посещена).

Изначально в очередь помещается только вершина s, и \rm used[s] = true, а для всех остальных вершин \rm used[] = false. Затем алгоритм представляет собой цикл: пока очередь не пуста, достать из её головы одну вершину, просмотреть все рёбра, исходящие из этой вершины, и если какие-то из просмотренных вершин ещё не горят, то поджечь их и поместить в конец очереди.

В итоге, когда очередь опустеет, обход в ширину обойдёт все достижимые из s вершины, причём до каждой дойдёт кратчайшим путём. Также можно посчитать длины кратчайших путей (для чего просто надо завести массив длин путей d[]), и компактно сохранить информацию, достаточную для восстановления всех этих кратчайших путей (для этого надо завести массив "предков" p[], в котором для каждой вершины хранить номер вершины, по которой мы попали в эту вершину).

**9) Вывод лабиринта с путем выхода на экран**

1) Прохождение по каждой клетке.  
2) Вывод стены или прохода.  
3) Если клетка является путем, то вывод кружка.

**10) Спасибо за внимание!**