Теоретическое решение задачи B

Алгоритм решения и доказательство его правильности

Для решения данной задачи использовался ориентированный невзвешенный граф, как основной математический объект. Т.к. комнаты в лабиринте удобно представлять в виде множества вершин, а переходы в виде набора дуг и нам не интересно сколько будет стоить переход из одной вершины в другую. Переходы вида a=b эквивалентны петлям графа, где a – номер комнаты, из которой вышла крыса, b – номер комнаты, в которую пришла. Также заметим, что одинаковые пары чисел a b, обозначающие разные переходы-это кратные дуги. Используем именно ориентированный граф, т.к. в условии задачи не сказано, что если существует путь из a в b крыса сможет вернуться из b в a.

Будем хранить граф с помощью матрицы смежности g размером n*n, где n – кол-во вершин. Напомним, что для графа с кратными дугами в ячейке i,j будет храниться кол-во путей из i в j.Очевидно, что ответом для k = 1 будет сумма элементов первой строки(крыса начинает бежать из 1-ой комнаты) матрицы смежности($\sum_{i=1}^{n} g[1][i]$), где k – длина маршрута.

Мы уже выяснили как найти ответ для k = 1, покажем как искать ответ для матрицы k+1, имея матрицу k. Мы будем пытаться понять возможно ли попасть из вершины i в j через любую 3-ую вершину (назовем эту вершину p). Если из вершины i есть путь в p (длиной k), а из вершины p существует дуга в j => из i в j существует путь длиной k+1. Таких вершин p может быть n штук (петля так же учитывается). Справедлива следующая формула: $d_{k+1}[i][j] = \sum_{p=1}^{n} d_k\,[i][p]g[p][j]$, где d_k-посчитанная матрица ответов для k, d_{k+1}-матрица ответов для k+1. Умножение и будет проверкой на то, сколько путей существует из i в j через p для k+1.Матрица смежности g используется т.к. нам нужно просто узнать сколько существует дуг из p в j без учета длины пути и предыдущих результатов, а матрица d_k как раз будет хранить предыдущие результаты.И т.к. нам нужно рассмотреть все промежуточные вершины p-мы суммируем результаты умножения.

Заметим, что приведенная выше формула совпадает с формулой произведения двух матриц. Значит, мы можем записать $d_{k+1} = d_k g$, т.к. d_1по сути является g, тогда $d_k = g^k$,т.е. для получения нужной нам матрицы мы должны перемножать исходную матрицу смежности k раз.

Чтобы уменьшить количество умножений матриц воспользуемся алгоритмом бинарного возведения в степень. Единственное условие для этого алгоритма, а именно-ассоциативность операции умножения соблюдено (умножение матриц ассоциативно). Воспользуемся тождеством (для любой четной степени):$a^k = (a^{k/2})^2 = a^{k/2}a^{k/2}$.Мы уменьшили степень n всего за одну операцию умножения. Если же степень нечетная-пытаемся ее сделать четной: $a^k = a^{k-1}a$.Т.е. если n-четна мы переходим к k/2,в противном случае к k-1.В худшем случае будет $2\log k$ переходов(когда k-нечетное).Если реализовывать с помощью рекурсии при достижении k = 0(база рекурсии) возвращать E-единичную матрицу, при умножении на которую вторая матрица будет оставаться сама собой.

Обобщим все вышесказанное, будем возводить матрицу g в степень k с помощью алгоритма бинарного возведения в степень. Реализуем его самым очевидным способом-с помощью рекурсии: при достижении базы рекурсии-возвращаем единичную матрицу, когда k – четное возвращаем вызов функции возведения в степень, где матрицей будет результат перемножения g на g, а степень уменьшится вдвое:k/2.Когда k-нечетное возвращаем результат перемножения g на вызов функции, у которой в параметрах будет матрица g и k уменьшенная на единицу:k-1. Для умножения матриц с помощью двойного вложенного цикла будем проходить по всем элементам матрицы и применять формулу $d_{k+1}[i][j] = \sum_{p=1}^{n} d_k\,[i][p]g[p][j]\%mod$ для каждого элемента матрицы,mod-модуль 10^9+7. Заметим, что для результата всех операций в алгоритме будем брать остаток от ответа по модулю, т.к. ответ может быть очень большим. В конце для получения итогового ответа просуммируем все элементы первой строки получившейся матрицы:$\sum_{i=1}^{n} g[1][i]\%mod$

Временная сложность

1.Умножение квадратных матриц размером n совершается за O(n^3), т.к. алгоритм перемножения работает с тройным вложенным циклом, каждый из которых проходит n вершин.

2.Алгоритм бинарного возведения в степень как описано выше работает за O($2\log k$)=O($\log k$) умножений

Итоговая временная сложность-O($n^3 \log k$)

Затраты памяти

Для реализации описанного выше алгоритма, требуется только исходная матрица смежности размера n*n, а также константное число вспомогательных переменных.

Таким образом, итоговые затраты памяти-O(n^2).