A. Степінь двійки

Автори задачі: Денис Остапенко (ідея), Володимир Мшанецький (підготовка)
Кількість команд, що здали: 18 з 38 (47%)

Дана задача розв'язується за допомогою методу «прекалк» (від англійського «precalculate»). Цей метод полягає в тому, що спочатку пишеться допоміжна програма, яка запускається на комп'ютері учасника та знаходить відповіді на всі можливі варіанти вхідних даних задачі, після чого пишеться нова програма-розв'язок, в яку всі знайдені допоміжною програмою відповіді вставляються як константи (разом із відповідними їм вхідними даними). При цьому програма-розв'язок лише читає вхідні дані, знаходить їх серед відомих їй та одразу виводить відповідну відповідь.

Метод «прекалк» можна застосувати в цій задачі завдяки тому, що множина можливих вхідних даних в ній досить обмежена: існує лише $2^{10}-1=1023$ різних непустих підмножини десяткових цифр (не враховуючи те, що цифри можуть бути перелічені у вхідних даних у довільному порядку). Використання цього способу є доцільним в даній задачі, тому що написання програми, що «чесно» знаходить відповідь по заданим вхідним даним за дозволену в обмеженнях 1 секунду, принаймні спочатку виглядає нетривіальною. При цьому на допоміжну програму обмеження в 1 секунду, звісно, на накладається, бо учасник запускає її самостійно на своєму комп'ютері.

Алгоритм роботи допоміжної програми, що шукає відповіді на всі можливі вхідні дані, може бути наступним. Переберемо в циклі всі показники степеня E від 0 до $2\cdot10^5$. Для кожного E обчислимо відповідний йому степінь $P=2^E$. Затим побудуємо десятковий запис P (наприклад, за допомогою конвертації P у рядок) і знайдемо множину різних десяткових цифр D_E, що зустрічаються в ньому. Після цього перевіримо, чи вже зустрічалась нам множина D_E раніше. Якщо так, то продовжуємо пошук. Якщо ні, то це означає, що для вхідних даних, що дорівнюють D_E, відповіддю є поточний показник степеня E. Виводимо D_E та E, запам'ятовуємо множину D_E як таку, що вже зустрічалася, і також продовжуємо пошук. Коли цикл по E буде завершено, стане відомо, що для всіх множин десяткових цифр, які не було виведено, відповідь дорівнює -1. Варто зазначити, що в цілях оптимізації та спрощення реалізації слід не обчислювати P окремо на кожній ітерації, а зберігати значення з попередньої ітерації та домножувати його на 2.

Максимальне значення P в наведеному алгоритмі буде $2^{200\,000}$. У стандартні цілочисельні типи даних такі числа не вміщуються, а використовувати числа з рухомою комою не можна, бо вони можуть зберігати не всі значущі цифри. Тому для реалізації обчислення P в наведеному алгоритмі слід використовувати довгу арифметику. Це означає, що число зберігається не в числовій змінній, а, наприклад, в масиві, де в кожному елементі знаходиться окремий десятковий розряд числа. При цьому всі операції над таким числом виконуються «вручну», тобто за допомогою власноруч реалізованого алгоритму, що виконує операції як у стовпчик. В даній задачі найбільш оптимально зберігати розряди в масиві саме в десятковому вигляді, бо на кожному кроці алгоритму виконується переведення P в десятковий вигляд. Також можна використати готову реалізацію довгої арифметики, що існує в деяких мовах програмування. Наприклад в Java/Kotlin є клас `BigInteger`, а в Python вбудовані числові змінні автоматично перемикаються в режим довгої арифметики, коли число, що зберігається в них, стає занадто великим. Але така готова довга арифметика в даній задачі працює значно повільніше. В авторів задачі допоміжна програма, написана на C++ зі власноруч написаною довгою арифметикою відпрацьовує приблизно за 30 секунд, а аналогічні програми з використанням готової довгої арифметики на Kotlin та Python працюють близько 20 та 60 хвилин відповідно.

Фінальна програма-розв'язок, як було описано вище, має містити всі знайдені пари D_E та E у вигляді констант. При цьому множини D_E найбільш зручно вставити у вихідний код як рядкові константи у тому самому вигляді, у якому подається підмножина десяткових цифр у вхідних даних. Але треба звернути увагу на те, що цифри у вхідних даних можуть бути у довільному порядку. Один з найпростіших способів врахувати це — відсортувати символи у вхідному рядку за зростанням одразу після його зчитування. Відповідно, константи D_E також мають містити символи

в порядку за зростанням. Отже, доцільно при реалізації допоміжної програми зробити так, щоби вона одразу виводила множини D_E в такому вигляді.

Таким чином програма-розв'язок має зчитати вхідний рядок, відсортувати його символи, а далі може бути послідовність виразів if-elseif-elseif-elseif-..., які порівнюють отриманий рядок із константою та, якщо вони збігаються, виводить відповідну відповідь. Також можна використати вираз `switch` (або `when`), константний масив пар, map, тощо. У будь-якому разі, якщо відсортований вхідний рядок не було знайдено серед відомих, потрібно вивести -1.

Слід зазначити, що після того, як допоміжна програма закінчить виконання, можна помітити, що на практиці найбільша можлива відповідь в цій задачі значно менше $2 \cdot 10^5$. Настільки менше, що, якщо замінити в допоміжній програмі верхню границю циклу по E на це число, то вона буде відпрацьовувати значно швидше, ніж за 1 секунду, і все одно знаходити всі відповіді. Отже, можна не витрачати час на написання окремої програми-розв'язку, а трошки модифікувати допоміжну програму й відправити її як розв'язок.

Також, варто окремо розглянути одну з популярних помилок учасників — неправильне розуміння виразу «множина десяткових цифр певного числа». В множині елементи не повторюються, тобто, наприклад, множина десяткових цифр числа 111223 дорівнює $\{1,2,3\}$. Отже, десяткові цифри у вхідному рядку не повторюються, і кожну з них можна «використовувати» у степені декілька разів. Наприклад, для вхідних даних «356» відповідь — 16 ($2^{16}=65536$).

B. Від'ємна маса

Автор задачі: Володимир Мшанецький
Кількість команд, що здали: 34 з 38 (89%)

В даній задачі потрібно було обчислити значення виразу $-\lceil\sqrt{D}\rceil$, тобто, обчислити квадратний корінь з D, округлити догори та взяти зі знаком мінус. Все це можна зробити за допомогою математичних функцій, які були у всіх доступних на змаганні мовах програмування. Наприклад, код може бути наступним:

- C++: `-std::ceil(std::sqrt(D))`
- Python: `-math.ceil(math.sqrt(D))`
- Java: `-Math.ceil(Math.sqrt(D))`
- Kotlin: `-ceil(sqrt(D.toDouble()))`

При реалізації програми потрібно звернути увагу на два моменти. По-перше, для числа D потрібно використовувати 64-бітну змінну (`long long` або `long`), бо в змінні меншого розміру не вміщуються числа порядку 10^{18}. По-друге, потрібно вивести результат у цілочисельному форматі, в той час як функції `sqrt()` і `ceil()` повертають `double`. Отже, результат потрібно конвертувати у ціле число перед виводом.

Крім того, деякі команди здали обчислення квадратного кореня за допомогою бінарного пошуку та методу Ньютона. Також можна було реалізувати обчислення відповіді циклом `m = -1; while (m * m < D) m--;`. Це дуже неефективно, але також мало проходити.

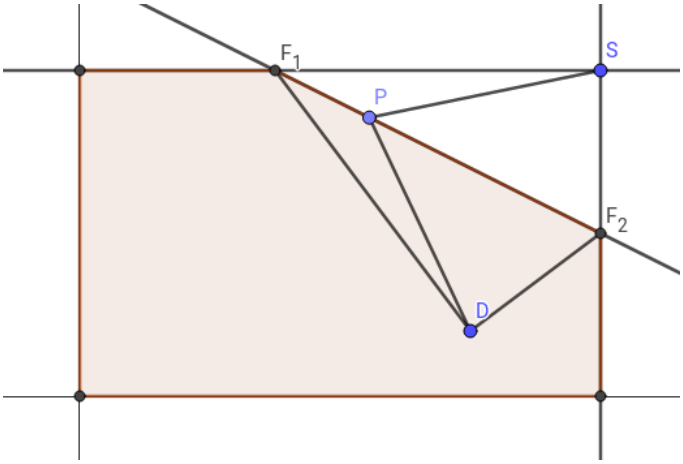
C. Согнуть лист бумаги

Автор задачі: Денис Остапенко
Кількість команд, що здали: 4 з 38 (11%)

Дана задача є суто математичною, тобто, в ній потрібно математично вивести формули на папірці, після чого реалізувати програму, що просто виконує обчислення за цими формулами. Розглянемо один з можливих розв'язків.

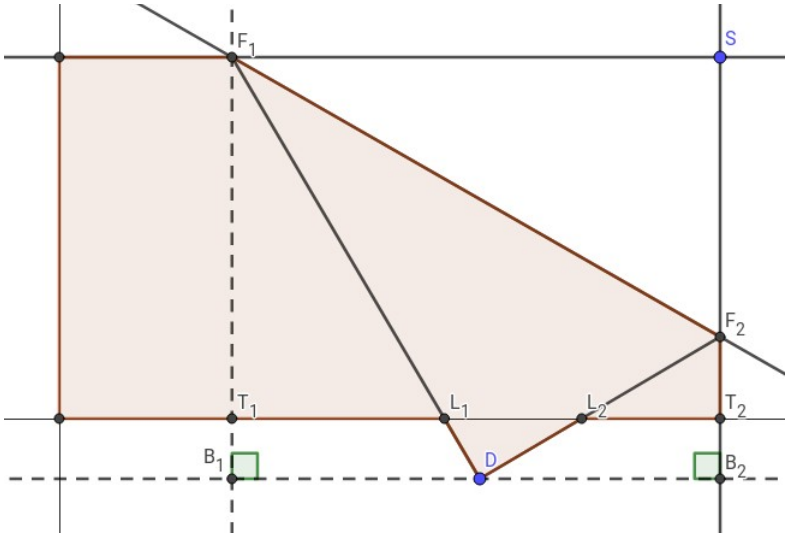
У випадку, коли $X>W$ або $Y>H$, очевидно, що перемістити кут таким чином, щоб не розірвати аркуш, неможливо. У випадку $X=W$ та $Y=H$ очевидно, що відповідь буде $W\cdot H$. У випадках $X=W$ та $Y<H$, а також $X<W$ та $Y=H$ також очевидно, що перемістити кут неможливо. Тепер розглянемо випадок $0\le X<W$ та $0\le Y<H$.

Позначимо точку (W,H) як S, а точку (X,Y) як D. Також позначимо точки перегину аркуша як F_1 і F_2. Очевидно, що в цьому випадку відповідь дорівнює площі прямокутника мінус площі трикутника SF_1F_2. Щоб знайти його площу, потрібно знайти довжини сторін SF_1 і F_2S, а для цього — координати точок F_1 та F_2.



Розглянемо довільну точку P на прямій F_1F_2. Трикутники SF_1F_2 та DF_1F_2 є рівними за умовою задачі. Завдяки цьому можна довести, що трикутники SPF_2 і DPF_2 також є рівними, а отже $SP=DP$. Це дозволяє нам записати рівняння прямої F_1F_2 як $distance(S,P)=distance(D,P)$. По черзі підставивши в це рівняння відомі координати $y_{F_1}=H$ та $x_{F_2}=W$, ми можемо знайти невідомі x_{F_1} і y_{F_2}. Якщо виявиться, що $x_{F_1}<0$ або $y_{F_2}<0$, то це також означає, що перемістити кут аркушу з точки S у точку D неможливо. В протилежному випадку ми можемо обчислити відповідь.

Тепер розглянемо випадок $0\le X<W$ та $Y<0$. В цьому випадку відповідь розраховується аналогічно попередньому випадку, але до неї необхідно додати площу трикутника DL_1L_2. Для того, щоб знайти її, необхідно знайти координати точок L_1 і L_2.



Розглянемо трикутники F_1B_1D та $F_1T_1L_1$. Можна довести, що вони — подібні, а отже $\frac{T_1L_1}{B_1D}=\frac{F_1T_1}{F_1B_1}$. Довжини відрізків B_1D, F_1T_1 і F_1B_1 можна легко записати по вже відомом координатам точок F_1 та D. Таким чином, ми можемо знайти довжину T_1L_1, а від неї — координату $x_{L_1}=x_{F_1}+T_1L_1$.

Аналогічним чином можна знайти координату x_{L_2}, розглянувши подібні трикутники F_2B_2D і $F_2T_2L_2$. Вона буде дорівнювати $x_{L_2}=x_S-T_2L_2$. Після цього ми можемо обчислити площу трикутника DL_1L_2, а отже, відповідь задачі.

Тепер розглянемо випадок $X<0$ та $0 \le Y < H$. В цьому випадку можна вивести формули аналогічно попередньому випадку. Або можна звести його до попереднього випадку, віддзеркаливши малюнок відносно прямої $y=x$, тобто, помінявши місцями значення W і H, а також X і Y.

Залишився лише випадок $X<0$ та $Y<0$, але очевидно, що загнути аркуш таким чином неможливо. Перевіряти цей випадок окремо непотрібно, бо в ньому ми й так отримаємо $x_{F_1}<0$ або $y_{F_2}<0$.

Таким чином, алгоритм роботи програми буде наступним:

1. Опрацювати випадки, коли $X \ge W$ і/або $Y \ge H$
2. Обчислити x_{F_1} і y_{F_2}, та перевірити, що $x_{F_1}>0$ та $y_{F_2}>0$
3. Обчислити $S_1 = W \cdot H - S_{SF_1F_2}$
4. Якщо $X \ge 0$ і $Y \ge 0$, то відповіддю є S_1
5. Обчислити координати точок L_1 і L_2
6. Обчислити та вивести $S_2 = S_1 + S_{DL_1L_2}$

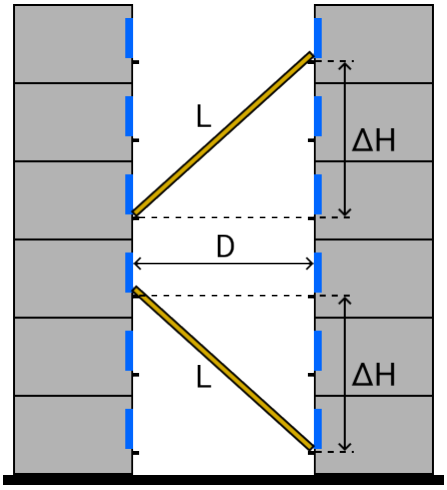
D. Problem authors

Автор задачі: Дмитро Садовий
Кількість команд, що здали: 33 з 38 (87%)

Давайте порахуємо, скільки задач мінімально можна підготувати для змагання — це роздати кожному автору мінімальну кількість задач, які він хоче підготувати, тобто $\sum_{1}^{N} X_i$. Також дізнаємось, скільки максимум можна підготувати задач — це роздати кожному автору максимальну кількість задач, яку він готовий підготувати, тобто $\sum_{1}^{N} Y_i$. Тепер очевидно, що якшо кількість задач олімпіади менша за $\sum_{1}^{N} X_i$, чи якщо це число більше за $\sum_{1}^{N} Y_i$ — то олімпіаду неможливо буде підготувати. Інакше відповідь «так».

E. Сто тисяч поверхів

Автор задачі: Володимир Мшанецький
Кількість команд, що здали: 8 з 38 (21%)



Хоча дану задачу розв'язало відносно небагато команд, вона насправді досить проста. Для початку розглянемо, як драбина кладеться між будинками. Якщо $D>L$, тобто, відстань між будинками більше ніж довжина драбини, то покласти драбину неможливо, а отже, відповідь буде -1. Якщо $D=L$, то драбину можна покласти лише горизонтально на два підвіконня, що знаходяться на одному рівні, тобто, на поверхах з однаковими номерами. Якщо $D<L$, то драбину можна покласти по діагоналі одним з двох способів, показаних на малюнку. В цьому випадку визначимо різницю в номерах поверхів у двох будинках, тобто, на скільки поверхів буде підійматися чи спускатися драбина. Оскільки висота всіх поверхів дорівнює 3 метрам, різницю можна обчислити як $\Delta F=\Delta H\div 3$. Значення ΔH можна обчислити за теоремою Піфагора: $\Delta H=\sqrt{L^2-D^2}$. Якщо отримане таким чином значення ΔF не є цілим, це означає, що не існує такої пари підвіконь, відстань між якими в точности дорівнює L, а отже, драбину покласти неможливо і відповідь буде -1.

Зауважте, що, якщо розрахувати ΔF за наведеною вище формулою у випадку $D=L$, то ми отримаємо $\Delta F=0$. Наведений нижче алгоритм працюватиме коректно у випадку $\Delta F=0$. Таким чином, опрацьовувати випадок $D=L$ окремо не потрібно.

Тепер знайдемо пару вікон, що підходять. Позначимо кількість поверхів у будинках як N_1 і N_2. Переберемо в циклі номер поверху в першому будинку F_1 від 1 до N_1. Якщо в першому будинку на поверсі F_1 немає вікна, то переходимо до наступного F_1. Інакше, переберемо номери поверхів другого будинку F_2, на які можна перелізти з поточного поверху F_1: $F_2=F_1+\Delta F$ і $F_2=F_1-\Delta F$. Для кожного F_2 перевіримо, що $1\le F_2\le N_2$ та що на цьому поверсі є вікно. Якщо це так, то ми знайшли пару поверхів, між якими можна перелізти. У цьому випадку потрібно обчислити, на скільки поверхів буде потрібно спуститися та піднятися ($(N_1-F_1)+(N_2-F_2)$), порівняти нову знайдену пару (F_1,F_2) з найкращою знайденою раніше, та оновити найкращу пару, якщо нова пара краща за неї згідно критеріїв, наведених в умові задачі. Після цього продовжуємо пошук з наступного F_1. Коли цикл закінчиться, найкраща знайдена пара (F_1,F_2) буде відповіддю задачі. Якщо жодної пари не було знайдено, то відповідь буде -1.

При реалізації програми єдиним неочевидним моментом є перевірка, що ΔF є цілим. В даній задачі, ймовірно через невеликі числа, працював майже будь-який спосіб перевірки, включаючи ті, в яких виконувалося порівняння чисел з рухомою комою на рівність. Але зазвичай таке порівняння не працює через похибки, тому більш безпечним було би виконувати перевірку, наприклад, за допомогою нерівності $|round(\Delta F)-\Delta F|<10^{-9}$. Також можна конвертувати ΔH у ціле число й після цього перевірити, що виконується рівність $D^2+(\Delta H)^2=L^2$.

F. Доїхати на футбол

Автор задачі: Дмитро Садовий
Кількість команд, що здали: 19 з 38 (50%)

Давайте розглянемо задачу, так, що стадіон знаходиться лише з одної сторони від усіх хлопців. Порахуємо яку мінімальну кількість машин нам потрібно, щоб умістити усіх хлопців. Це буде $X = \lceil N \div 5 \rceil$. І можна доказати, що нам не потрібно буде жодної додаткової машини, якщо відповідь справді існує. Для цього давайте попросимо X хлопців, що живуть дальше за всіх від стадіону, взяти машини. Всі інші туди точно помістяться, при умові що будуть в машинах вільні місця. Тоді нам залишається лише перевірити, що всі хлопці без машин дійсно помістяться в машини, які будуть їхати через них. Для цього відсортуємо хлопців за їх координатою, та пройдемо їх від того, що живе далі за всіх від стадіону, до того, що живе ближче за всіх до стадіону. Кожного разу, коли ми зустрічаємо хлопця, в якого є машина, ми будемо збільшувати кількість вільних місць на 4, а кожного разу, коли ми зустрічаємо хлопця без машини, ми будемо зменшувати цю кількість на 1. Якщо в жоден момент кількість місць не стане від'ємною, значить всі хлопці помістяться в машини.

Щоб вирішити задачу повністю, просто перевіримо і порахуємо це для двох окремих відрізків — ліворуч від стадіону та праворуч від стадіону.

G. Таємний маршрут патрулювання

Автор задачі: Володимир Мшанецький
Кількість команд, що здали: 1 з 38 (3%)

Дана задача може виглядати дуже складною (особливо, якщо подивитися анімацію з третього прикладу), але насправді вона розв'язується як звичайний лабіринт за допомогою пошуку в ширину. Потрібно тільки перебудувати лабіринт на тривимірний, щоби він став статичним.

Побудуємо новий лабіринт розміру $D \times H \times W$. Перший вимір буде позначати момент часу, тобто, поточний крок робота-охоронця, а інші два — рядок та стовпець оригінального лабіринту. Спочатку перенесемо всі статичні перешкоди (клумби з робо-квітами) в новий лабіринт. Для цього відмітимо відповідні клітини (або «кубики») нового лабіринту як блоковані у всі моменти часу. Після цього пройдемо циклом по всьому маршруту робота-охоронця та відмітимо всі клітини, які блокуються лазером, як блоковані в новому лабіринті у відповідний момент часу. Відмітити клітини можна, запустивши в кожному з чотирьох напрямків по циклу від поточної клітини, де знаходиться в цей момент часу робот-охоронець, до першої вже блокованої клітини, що зустрінеться. Таким чином ми побудуємо новий тривимірний лабіринт, в якому кожен зріз по фіксованій першій координаті буде збігатися зі станом оригінального лабіринту у відповідний момент часу. При цьому перший вимір у цьому лабіринті слід вважати зацикленим, тобто, за моментом часу $D-1$ знову слідує момент часу 0, тому що маршрут робота-охоронця є циклічним, а отже стани оригінального лабіринту повторюються через кожні D одиниць часу.

Стартовою клітиною в тривимірному лабіринті буде клітина з координатами $(0, R_S, C_S)$, де (R_S, C_S) — координати стартової клітини в оригінальному двовимірному лабіринті. Замість однієї фінішної клітини в тривимірному лабіринті буде цілий стовпчик фінішних клітин (t, R_F, C_F) для всіх $0 \le t \le D-1$, де (R_F, C_F) — координати фінішної клітини в оригінальному двовимірному лабіринті. Можливі ходи герою (Микити) в тривимірному лабіринті будуть такими самими, як у двовимірному, тільки з урахуванням того, що в кожному ході (вліво, вправо, вгору, вниз, стояти на місці) герой також переміщується на одну одиницю часу вперед у майбутнє. Тобто, доступні ходи будуть $(+1, 0, -1)$, $(+1, 0, +1)$, $(+1, -1, 0)$, $(+1, +1, 0)$ і $(+1, 0, 0)$.

Тепер ми можемо знайти найкоротший шлях від старту до одного з фінішів у тривимірному лабіринті за допомогою пошуку в ширину аналогічно тому, як це робиться в класичній задачі про двовимірний лабіринт. Опис використання алгоритму пошуку в ширину для розв'язання лабіринту виходить за рамки цього розбору, але його можна легко знайти в інтернеті. Зауважимо, що, хоча в оригінальному двовимірному лабіринті може знадобитися двічі заходити в одну клітину, в тривимірному лабіринті ніколи немає сенсу двічі заходити в одну тривимірну клітину («кубик») на шляху, бо, якщо ми вже були в деякій клітині оригінального двовимірного лабіринту в деякий момент часу, то, якщо ми зайдемо в неї знову в той самий момент часу, нічого нового нам це не дасть. При цьому заходити в декілька клітин (t, r, c) з різними t і однаковими r та c можна, і це як раз відповідає декільком відвідуванням клітини (r, c) в оригінальному двовимірному лабіринті в різні моменти часу.

При реалізації програми слід звернути особливу увагу на обмеження на використання оперативної пам'яті в задачі. Враховуючи максимальні обмеження на W, H і D, розмір тривимірного масиву 32-бітних цілих розмірністю $D \times H \times W$ вже буде займати більше 200 МБ з доступних 256 МБ. Тому необхідно вмістити майже всі дані, які потрібні під час роботи алгоритму, в цей один масив. Наприклад, для відвіданих клітин елементи масиву можуть зберігати довжину шляху до цієї клітини, невідвідані клітини можуть позначатися якимось дуже великим числом (наприклад, `INT_MAX`), а блоковані клітини можна позначати числом -1. Оскільки в деяких тестах (наприклад, повністю пусте поле з відповіддю -1) може знадобитися відвідати майже всі клітини тривимірного лабіринту для того, щоби визначити відповідь, спроби оптимізувати використання пам'яті за допомогою використання `map` замість масивів будуть мати зворотній ефект і не пройдуть.

H. Labyrinth

Автор задачі: Олександр Нестерюк
Кількість команд, що здали: 13 з 38 (34%)

Завдання пошуку шляху в лабіринті може бути зведена до завдання пошуку шляху на графі, для вирішення якої можна скористатися одним із відомих методів рішення. Для ситуації конкретної задачі можна застосувати алгоритм Дейкстри знаходження найкоротшого шляху. Цей алгоритм передбачає роботу з матрицею суміжності досліджуваного графа. Таким чином, розв'язання задачі зводиться до наступного:

1. Перетворення лабіринту на відповідний йому граф;
2. Побудова матриці суміжності одержаного графа;
3. Знаходження найкоротшого шляху між початковою та кінцевою вершинами отриманого графа, причому сам шлях можна не обчислювати, обмежившись знаходженням його довжини (часом проходження лабіринту).

Перший етап рішення може бути з'єднаний з другим — одразу побудувати матрицю суміжності необхідного графа. Для отримання графа потрібно послідовно пронумерувати всі клітини лабіринту, які при цьому стають вершинами графа, що формується, і захопити двовимірний квадратний масив пам'яті для зберігання матриці суміжності розмірності $(M \cdot N) \times (M \cdot N)$ (де M і N — розмірності поля лабіринту). Значення елементів матриці визначаються можливістю переходу між двома сусідніми вершинами графа (тобто між двома сусідніми клітинами лабіринту) і становлять ту ціну (тобто час), яку необхідно витратити на здійснення даного переходу, якщо даний перехід можливий, або 0 в іншому випадку. Оскільки перехід можливий тільки між сусідніми клітинами, що знаходяться попереду, ззаду, ліворуч або праворуч і тільки якщо різниця висот між ними і поточною клітиною (вершиною) не перевищує 1, то програма обчислення елементів матриці C виглядає таким чином:

```
for(int i = 0; i < N; i++) {
    for(int j = 0; j < M; j++) {
    if(i > 0) C[(i-1)*M+j][i*M+j] = C[i*M+j][(i-1)*M+j] =
abs(A[i][j]-A[i-1][j]) <= 1 ?  (abs(A[i][j]-A[i-1][j]) ? K * L :
K) : 0;
    if(j > 0) C[i*M+j-1][i*M+j] = C[i*M+j][i*M+j-1] = abs(A[i]
[j]-A[i][j-1]) <= 1 ?  (abs(A[i][j]-A[i][j-1]) ? K * L : K) : 0;
    if(i < N - 1) C[(i+1)*M+j][i*M+j] = C[i*M+j][(i+1)*M+j] =
abs(A[i][j]-A[i+1][j]) <= 1 ?  (abs(A[i][j]-A[i+1][j]) ? K * L :
K) : 0;
    if(j < M - 1) C[i*M+j+1][i*M+j] = C[i*M+j][i*M+j+1] =
abs(A[i][j]-A[i][j+1]) <= 1 ?  (abs(A[i][j]-A[i][j+1]) ? K * L :
K) : 0;
    }
}
```

Всі інші елементи матриці C дорівнюватимуть 0. Як видно з наведених формул, отримана матриця C є квадратною матрицею з нульовими елементами на головній діагоналі і елементи над головною діагоналлю є відображенням елементів під головною діагоналлю (оскільки граф не є спрямованим, тобто переміщення можливі з однаковою ціною у будь-якому напрямку по дугах). Звідси випливає, що кількість елементів, що зберігаються, може бути скорочена як мінімум в 2 рази, що також є необхідним виходячи з встановлених обмежень на обсяг пам'яті, що використовується: $(100 \cdot 100) \cdot (100 \cdot 100) \cdot 4$ перевищує її ліміт. Також для подальшого скорочення обсягу зайнятої пам'яті можна зберігати тільки ненульові елементи матриці C (їх всього максимум 4 для кожної клітини лабіринту), або взагалі відмовитися від їх зберігання, розраховуючи щоразу динамічно заново.

Далі застосовується алгоритм Дейкстри, який полягає у знаходженні мінімальної ціни, яку необхідно заплатити задля досягнення кожної з вершин. Для цього захоплюється одномірний масив D розмірністю, що дорівнює кількості вершин ($M \cdot N$), елементи якого дорівнюватимуть ціні

на поточний крок алгоритму. Спочатку всі елементи даного масиву дорівнюють максимально можливому значенню цілого числа `MAX_INT`, крім нульового, що відповідає початковій клітині лабіринту, який дорівнює `0`. При такій побудові кінцевий елемент масиву `D` відповідає останній клітині лабіринту, в якій знаходиться вихід і результатом завдання можна вважати значення, отримане в останньому елементі масиву `D`. На початковому етапі нульова вершина розглядається як поточна. Ще один масив такої ж розмірності знадобиться для зберігання інформації про те, чи було розглянуто («відвідано») відповідну вершину графа на попередніх кроках алгоритму.

Кількість кроків алгоритму визначається кількістю зв'язків між вершинами. Крок алгоритму полягає у виборі поточної вершини з найменшим значенням `D[i]` з тих, які мають зв'язки з уже «відвіданими» на попередніх кроках та переборі всіх «невідвіданих» вершин, пов'язаних із нею. Після закінчення кроку поточна вершина позначається як «відвідана».

На кожному кроці алгоритму розраховуються нові значення елементів масиву `D`. Значення елементів `D[i]` для вершини `i`, пов'язаної з поточною вершиною `j`, виходить рівним значенню `D[j] + C[i][j]`. Тобто, обчислення організуються всіма ненульовими елементами `i`-го рядка матриці `C`. При обчисленні нових значень елементів масиву `D` вибирається мінімальне значення з наявного і нового. Наприклад, якщо вершину `i` можна потрапити двома способами, витративши 5 чи 8 одиниць (старе значення 5, а нове 8), то значення елемента `D[i] = 5`. Також необхідно надходити, якщо попереднє значення елемента `D[i] = MAX_INT` (тобто, у цю вершину ще не було знайдено шляху з вихідної і новий шлях є єдиним).

Алгоритм закінчується тоді, коли не залишилося жодного не розглянутого зв'язка між усіма вершинами, тобто, всі вершини відзначені як «відвідані» або зв'язків, що залишилися, не існує.

Після закінчення роботи алгоритму залишається перевірити останній елемент масиву `D` і вивести його значення або, якщо воно дорівнює `MAX_INT`, тоді «-1», оскільки шляхи з вихідної точки лабіринту в кінцеву точку не існує.