Разбор заданий олимпиады ЦПМ
Старшая-1

Введение

В данном материале вы сможете найти рекомендации к выполнению задачи, а также описание некоторых алгоритмов, которые можно использовать в её решении. Курсивом в тексте выделены ссылки, по которым вы сможете найти дополнительную, более подробную информацию по темам.

Основная идея

Будем работать с неориентированным взвешенным графом: узлы сетки - вершины, отрезки прямых - рёбра, веса рёбер сообщают в день состязаний. Тогда задача сводится к поиску кратчайшего пути в этом графе между вершиной A3 и закодированной в штрих-коде.

Разберёмся с определениями

Если вам совсем не понятно о чём идет речь, то, вероятно, вы ещё никогда не сталкивались с понятием графа.

Граф состоит из вершин и соединяющих их рёбер. Они бывают ориентированные и неориентированные. Также вершины или рёбра могут иметь дополнительный параметр - вес. Рассмотрим эти понятия на примере карты метро.

Станции - это вершины, а перегоны между станциями - рёбра. Весом ребра может быть время, которое поезд тратит на перемещение от одной станции к другой. Этот граф неориентированный, так как по любому перегону мы можем двигаться как в одну, так и в другую сторону.

Подробнее про графы можно почитать *здесь* или *здесь.*

Как хранить граф

У нас есть вершины и рёбра. Пронумеруем как-то вершины, а рёбра будем задавать парой чисел - номерами вершин, которые оно соединяет.

Обычно графы хранят одним из этих способов:

1) *Матрица смежности* - двумерный массив, в ячейках которого хранится информация про ребро между любой парой вершин. Например вес, или -1 если такого ребра нет.

2) *Список смежности* - двумерный массив, в котором для каждой вершины хранится список смежных вершин.

3) *Список рёбер* - двумерный массив, в котором в каждой строчке хранится информация о ребре.

Для неориетированных графов стоит отметить, что если в задаче дано ребро (v, u) веса w, то при добавлении его в одну из структур следует добавлять как пару (v, u), так и (u, v).

Как избавиться от двумерных массивов

В случае, если возникает проблема использования двумерных массивов, можно заменить их на одномерные. Рассмотрим, как это можно сделать для матрицы смежности и списка рёбер.

Пусть у нас есть граф на n вершинах и m рёбрах. (Например, в данной задаче $n = 35$, так как сетка имеет 35 узлов, а $m = 58$, так как между этими узлами 58 отрезков.)

Начнем с матрицы смежности - это двумерный массив $n \times n$, назовём его W. Тогда $W_{i,j}$ - вес ребра из вершины i в вершину j. Тот же массив мы можем хранить одномерно - выпишем строчки W подряд в одну длинную, получим одномерный массив H размером n^2. Тогда чтобы узнать $W_{i,j}$ необходимо обратиться к массиву H по индексу $i \cdot n + j$.

Теперь рассмотрим, как действовать в случае списка рёбер. Пусть в каждой строчке двумерного массива мы хранили начало, конец и вес ребра - получим массив L размером $m \times 3$. Превратим его в одномерный тем же способом - выпишем подряд в одну строку и получим массив T размером $3m$. Тогда чтобы узнать строчку L_i, необходимо обратиться к T по индексам с $3 \cdot i$ по $3 \cdot i + 2$.

Замечание к данной задаче

Граф в данном случае - координатная сетка. Для простоты поменяем систему координат - вместо [1, 2, 3, 4, 5] пронумеруем строки с нуля: [0, 1, 2, 3, 4]. Аналогично сделаем и для столбцов - вместо [а, б, в, г, д, е, ж] будем нумеровать [0, 1, 2, 3, 4, 5, 6]. Тогда номер узла в сквозной нумерации можно задать по координатам следующим образом: $i = y \cdot 7 + x$, где y - номер строки, а x - номер столбца.

Рассмотрим, что происходит, на примере стартовой вершины. Её начальные координаты - $(A, 3)$. Координаты после замены - $(0, 2)$. Тогда номер стартовой вершины это $st = 2 \cdot 7 + 0 = 14$.

Можно по-прежнему задавать вершины парой координат, однако ввести сквозную нумерацию будет удобнее, ведь так легче обращаться к рёбрам. Также стоит отметить, что существует много различных способов перенумеровать вершины - выберите тот, который вам удобнее и понятнее.

Как искать кратчайший путь

Существует много способов выполнить эту задачу, рассмотрим некоторые из них.

Первый - **алгоритм Дейкстры**. Это алгоритм поиска кратчайших путей от заданной вершины до любой другой. Он применим только если веса рёбер - неотрицательные числа.

Обозначим за d_v - текущую верхнюю оценку на кратчайший путь от st (заданная вершина) до вершины v. Изначально для всех вершин эта оценка представляет собой какое-то очень большое число, означающее недостижимость (например, можно взять 10^9, так как сумма любого пути по рёбрам будет точно меньше, чем это число).

Разделим вершины на 2 группы - в первой группе (назовем её S) будут вершины, для которых d_v является точной оценкой - весом кратчайшего пути, во второй (назовем её $V \setminus S$) - все остальные. Изначально все вершины лежат в $V \setminus S$. Для st мы знаем точную оценку: $d_{st} = 0$, так как мы уже находимся в стартовой вершине.

Итак, наша задача переместить все вершины из $V \setminus S$ в S. Будем выбирать в $V \setminus S$ вершину с минимальной оценкой, перемещать её в S и "релаксировать" рёбра из неё.

Под релаксацией ребра следует понимать попытку улучшить ответ через данное ребро. Например, если мы рассматриваем ребро $v \to u$ веса w, то мы попробуем улучшить оценку на кратчайший путь для вершины u через вершину v: $d_u = min(d_u, d_v + w)$. При перемещении v из $V \setminus S$ в S следует прорелаксировать все рёбра, выходящие из v.

Действуя таким образом, мы переместим все вершины в множество S, а узнать вес кратчайшего пути до нужной нам вершины сможем обратившись к массиву d.

Подробнее ознакомиться с этим алгоритмом и понять, почему он работает, можно *здесь* или *здесь*.

Ниже приведен псевдокод алгоритма:

```
INF = 1e9                          #"недостижимость"

def dijkstra(st):
    d = [INF for i in range (n)] #изначально знаем, что кратчайший путь < INF
    used = [0 for i in range (n)]#будем помечать 1 вершины, которые в S
    d[st] = 0                      #d для стартовой вершины уже знаем
    for i in range n:              #хотим переместить в S все n вершин
        min_d = INF                #минимальное значение d по вершинам вне S
        v = -1                     #индекс вершины со значением min_d
        for j in range n:          #выбираем минимальную вершину из V \ S
           if not used[j] and d[j] < min_d:
               min_v = j
               min_d = d[j]

        used[min_v] = 1            #добавляем выбранную вершину в S
                                   #ребра хранятся в матрице смежности w
                                   #ребра (i, j) нет, то w[i][j] = -1
        for u in range n:          #релаксируем ребра из выбранной вершины
           if w[min_v][u] > -1:    #проверяем, что ребро (min_v, j) существует
               d[u] = min(d[u], d[i] + w[i][u])
```

Второй - **алгоритм Форда-Беллмана.** Это алгоритм поиска кратчайших путей от заданной вершины до любой другой. Он применим даже если есть рёбра отрицательного веса.

Аналогично, обозначим за d_v - текущую верхнюю оценку на кратчайший путь от st до вершины v. Изначально для всех вершин это большое число, означающее недостижимость.

Так как в графе всего n вершин, то максимальная длина пути (количество рёбер в этом пути) между любыми 2 вершинами меньше n. Знаем, что $d_{st} = 0$. На каждой итерации будем релаксировать все рёбра графе, причем на i-той итерации пути длины $\leqslant i$ посчитаны верно, тогда достаточно проделать n итераций и пути до всех вершин будут посчитаны верно.

Подробнее ознакомиться с этим алгоритмом и понять, почему он работает, можно *здесь* или *здесь*.

Ниже приведен псевдокод алгоритма:

```
INF = 1e9                          #"недостижимость"
d = [INF for i in range (n)]       #изначально знаем, что кратчайший путь < INF

for i in range(1, n):              #совершаем n - 1 итерацию
   for e in edges:                 #просматриваем все ребра в графе
                                   #e.v и e.u - вершины ребра, e.w - его вес
      d[e.u] = min(d[e.u], d[e.v] + e.w)      #релаксируем ребро
```

Как восстановить путь

Просто узнать вес кратчайшего пути недостаточно. Необходимо востановить посследовательность вершин, по которым этот путь проходит. Заведём массив $prev$, в котором будем хранить номер предыдущей вершины в кратчайшем пути.

Для этого во время релаксации необходимо запоминать из какой вершины мы "пришли". Другими словами, если при релаксации ребра $v \to u$ веса w получается, что $d_v + w < d_u$, то $prev_u = v$

Зная $prev$ для любой вершины, легко восстановить весь путь от st до искомой вершины (назовем её t). Начиная с t будем идти по предкам, пока не дойдем до st.

Ниже приведен псевдокод:

```
path = []                  #здесь будет кратчайший путь
v = t                      #текущая вершина
while not v == st:         #пока не дойдем до стартовой
   path.append(v)          #добавим текущую вершину в путь
   v = prev[v]             #перейдем к предку
path.append(st)            #добавим стартовую
path.reverse()             #восстановим корректный порядок
```

Обработка штрих-кода

Считав данные со штрих-кода мы получаем двоичную последовательность. Чтобы определить закодированные координаты, необходимо перевести её в десятичную систему счисления. Как это сделать можно узнать *здесь.*

Функционал робота

Для того, чтобы описанные выше решения было легко реализовать, рекомендуется заранее написать функции движения: повороты, проезд до перекрёстка.

Зная верную последовательность вершин и используя эти функции несложно написать функцию, определяющую, как роботу действовать на каждом перекрёстке, чтобы добраться до нужной вершины.

Также в целом рекомендуется создавать функции для выполнения отдельных блоков задания - это структурирует программу и облегчает отладку.

Заключение

Предложенные решения не являются единственно верными. Однако предполагается, что они помогут вам подступиться к задаче или справиться с возникающими в ходе её решения трудностями.