Разбор DP [6]

В целом это разбор прошедшего контеста. Но сначала несколько основных принципов динамического программирования.

Основы динамического программирования

Под динамическим программированием понимается подход, когда вы, имея какую-то задачу, разбиваете её на более мелкие подзадачи, которые вы можете проще рассчитать.

Например, в задаче вычисления n-го числа Фибоначчи вы разбиваете её на вычисление $n-1$ и $n-2$ числа. Но заметим, что до бесконечности разбивать не получится. Поэтому нам нужно указать какие-то минимальные решения, на которых можно останавливаться. Здесь возьмём первое и второе числа Фибоначчи, оба равны единице.

Заметим, что для вычисления $n-1$ числа Фибоначчи, нужно вычислить ещё раз $n-2$ и $n-3$. В динамическом программировании мы этого делать не будем, каждый раз когда мы вычисляем какое-то значение, мы запомним его, например, в какой-нибудь массив, чтобы при следующем обращении просто взять посчитанное значение.

Если мы заведём вектор dp для этой задачи длины $n+1$, то после всех вычислений ответ будет храниться в элементе $dp[n]$.

Таким образом, для решения задачи у нас есть:

1. Формула пересчёта текущего значения через более простые
2. Значения наиболее простых случаев
3. Структура для хранения полученных ответов
4. Понимание, где в нашей структуре хранится ответ на задачу

Разбор контеста DP [6]

A. Кузнечик

Условие.

Кузнечик Пётр живёт на числовой прямой и ему нужно попасть из точки 0 в точку n, он может прыгать только в сторону увеличения координат не более чем на k шагов, то есть первый прыжок он может осуществить только в точки $1, 2, \ldots, k$. Помогите ему определить сколькими путями он сможет это сделать, так как ответ может быть очень большой выведите его по модулю $10^9 + 7$.

Решение.

Количество путей добраться до точки i на числовой прямой у кузнечика равно сумме количества путей добраться до предыдущих k точек. Будем хранить эти значения в векторе dp длины $n + 1$. Укажем, что количество путей остаться на месте равно 1, $dp[0] = 1$.

$$dp[n] = \sum_{i=1}^{k} dp[i - k]$$

В коде у нас могло быть что-то такое:

```
vector<int> dp(n+1);
dp[0] = 1;
for (int i = 0; i <= n; i++) {
    for (int j = i - 1; j >= max(0, i - k); j--) {
        dp[i] = (dp[i] + dp[j]) % INF;
    }
}
```

B. Непростой кузнечик

Условие.

Кузнечик Пётр живёт на числовой прямой и ему нужно попасть из точки 1 в одну из точек интервала с l по r включительно, он может прыгать только в точки чьи координаты делятся на координаты его текущей точки, то есть из точки i прыжок он может осуществить только в точки $2 \cdot i, 3 \cdot i, 4 \cdot i, \ldots$. Помогите ему определить сколькими путями он сможет это сделать, так как ответ может быть очень большой выведите его по модулю $10^9 + 7$.

Решение.

Здесь нам уже не удобно считать текущее значение через предыдущие, так как придётся разбирать каждое встретившееся число на делители.

Поэтому заметим, что текущее посчитанное значение количества прыжков $dp[i]$ мы можем прибавлять к $i \cdot 2, i \cdot 3, \ldots$.

В коде это выглялит так:

```
for (int i = 1; i <= r; i++) {
    for (int j = 2 * i; j <= r; j += i) {
        dp[j] = (dp[i] + dp[j]) % INF;
    }
}
```

Выше в коде: const int INF = 1e9+7;

Также не забываем, что для ответа нужно просуммировать все $dp[i]$ для $i \in [l, r]$.

C. Кузнечик и опастности

Условие.

Кузнечик Пётр живёт на числовой прямой и ему нужно попасть из точки 0 в точку n, он может прыгать только в сторону увеличения координат не более чем на k шагов, то есть первый прыжок он может осуществить только в точки $1, 2, \ldots, k$. Однако не все места на числовой прямой безопасны, в некоторых точках зазевавшегося путника ждут неприятности. Гарантируется, что начало и конец пути безопасны. Помогите кузнечику определить сколькими путями он сможет добраться до пункта назначения не проходя через небезопасные точки, так как ответ может быть очень большой выведите его по модулю $10^9 + 7$.

Решение.

Здесь нам нужно запомнить, в каких координатах таится опасность. Если текущая координата опасна, будем оставлять значение в ней равным нулю.

В коде у нас например следующее:

```
for (int i = 1; i <= n; i++) {
    if (s.count(i)) {
        continue;
    }
    for (int j = i; j >= max(0, i - k); j--) {
        dp[i] = (dp[i] + dp[j]) % INF;
    }
}
```

Где s - *set* наших опасных координат.

D. Кузнечик и алчность

Условие.

Кузнечик Пётр живёт на числовой прямой и ему нужно попасть из точки 0 в точку n, он может прыгать только в сторону увеличения координат не более чем на k шагов, то есть первый прыжок он может осуществить только в точки $1, 2, \ldots, k$. Недавно была произведена распродажа чисел на числовой прямой и каждое из них теперь кому-то принадлежит, понятное дело, теперь все хотят как-то отбить свои вложения, поэтому за посещение каждого числа была установлена какая-то цена. Ваша задача помочь кузнечику потратить как можно меньше денег на своём пути.

Решение.

Теперь нам нужно пересчитывать текущее значение через предыдущие таким образом, чтобы до текущей клетки можно было добраться как можно дешевле. Предположим, что мы уже посчитали все предыдущие значения. Тогда самый дешёвый путь до текущего будеч через минимальное из k предыдущих посчитанных значений.

А, нам здесь нужно ещё выводить путь! Тогда при нахождении очередного минимального из предыдущих k значений, мы запоминаем его индекс в каком-нибудь массиве *path* (путь) для клетки, в которой стоим.

Затем нам достаточно попереходить по всем индексам, начиная с n и записать весь индексный путь в вектор ответа. Потом отразить его.

В коде это подобное:

```
1  vector<ll> path(n + 1, -1);
2  vector<ll> dp(n + 1);
3  for (int i = 1; i <= n; i++) {
4      ll mx = LINF;
5      for (int j = i-1; j >= max(0, i - k); j--) {
6          if (mx > dp[j]) {
7              mx = dp[j];
8              path[i] = j;
9          }
10     }
11     dp[i] = cost[i] + mx;
12 }
13 vector<int> ans;
14 for (int i = n; i >= 0; i = path[i])
15     ans.push_back(i);
16 reverse(all(ans));
```

запомнили -1, чтобы цикл на строке 14 завершился по достижении 0-й коорд.

↓

Что такое LINF?

Это просто константа, равная числу, которое никогда не будет достигнуто в векторе dp.

Почему переменная называется mx, а не min? ТАК ВЫШЛО :)

это в моём коде записывается на ans.begin(), ans.end()

"Зачем мы дали эту задачу?"

E. Стройся!

Условие.

Товарищ майор любит очень любит дисциплину, однако он также любит разнообразие. Под его командованием находится рота из n солдат. Каждому солдату майор присвоил номера от 1 до n в порядке убывания роста. Для того, чтобы ежедневное построение на плацу не наскучило ему, он придумал новые правила расположения солдат:

- Солдат с номером 1 стоит первым.
- Разность номеров соседних солдат не превышает 2.

Зная, что товарищ майор успокоится и отправит солдат в запас только когда попробует все возможные построения удовлетворяющие правилам, они хотят узнать оставшийся срок службы. Помогите им вычислить количество дней до возвращения домой.

Разбор. (Спасибо Сергею)

Для небольших n решение найти просто.

При $n = 1$ единственное расположение это (1), значит ответ 1.

При $n = 2$ единственное расположение это $(1, 2)$ здесь тоже ответ 1.

Для $n = 3$ уже два варианта: $(1, 2, 3)$ $(1, 3, 2)$ ответ 2.

Рассмотрим общий случай с достаточно большим n. По условию мы должны начать с 1. Какое число можем поставить следующим?

- Следующим поставим 2. Но что тогда мы можем заметить? Теперь необходимо решить задачу:

  Каким количеством способов можно расположить числа от 2 до n, со следующими условиями:

  - На первом месте стоит число 2 (ведь мы решили что сейчас будем брать его).
  - Соседние значения должны отличаться не более чем на 2.

  Знакомо не так ли? Это в точности исходная задача, но только ее размерность $n - 1$. Ведь нет разницы распределять числа от 2 до n или от 1 до $n - 1$.

- Можем взять число 3. Тогда следует подумать что же дальше.

  - Возьмем число 4. В итоге получается последовательность $(1, 3, 4, \dots)$. Можно заметить что число 2 в этом случае необходимо использовать прямо сейчас, иначе потом не будет выполнено

условие, однако после 2 поставить ничего тоже не сможем и это путь в никуда.

– Раз 4 никак не подходит, посмотрим что если взять 2. Получается $(1, 3, 2, \dots)$. Тогда следующим числом не подходит ничего кроме 4 и мы снова видим задачу идентичную исходной, но только начинаем с 4 и распределяем $n - 3$ числа.

– Оставшийся возможный вариант это попробовать 5. Начало тогда выглядит как $(1, 3, 5, \dots)$. Заметим, что если далее взять 4 или 6, то получим либо $(1, 3, 5, 4, \dots)$ либо $(1, 3, 5, 6, \dots)$ и в обоих этих случаях взяв число меньше, не сможем никогда взять бОльшие числа и наоборот, взяв бОльшие никак не возьмем меньшие.

  В самом деле $(1, 3, 5, 4, 2, \dots)$ не имеет продолжения, в $(1, 3, 5, 4, 6)$ число 2 невозможно будет дальше использовать. Так же и $(1, 3, 5, 6, 4, \dots)$ не позволяет уже брать числа 7 и более, в $(1, 3, 5, 6, 7, \dots)$ и $(1, 3, 5, 6, 8, \dots)$ числа 2 и 4 невозможно будет дальше поместить.

  Таким образом далее следует брать 7. Продолжая подобные рассуждения видим что продолжать всегда будем нечетными числами, до тех пор пока они $<= n$, а затем начиная с максимального четного числа $<= n$ идем по убыванию обратно в сторону числа 2. Получается последовательность вида $(1, 3, 5, \dots, n-1, n, n-2, n-4, \dots, 2)$ в случае четного n и $(1, 3, 5, \dots, n-2, n, n-1, n-3, \dots, 2)$ в случае нечетного n.

  И подобная последовательность только одна.

Таким образом решение исходной задачи размерности n сводится к решению двух задач меньшей размерности ($n - 1$ и $n - 3$) и одному особому случаю который добавляет 1 к ответу.

По сути это то же самое что вычисление чисел Фибоначчи, но чуть по другим правилам. Как это делать было рассказано в лекции. Итоговая сложность $O(n)$. По памяти $O(n)$ или $O(1)$ в зависимости от реализации ;-)

Для 111 ответ порядка 10^{18}, поэтому без *long long* не обойтись. Ну или писать на *python*, ограничения малы, поэтому проблем со временем не будет, да и код короче))

F. Черепашка и опасности

Условие.

Черепашка живёт на прямоугольном поле и хочет добраться из точки $(0, 0)$ в точку (n, m). Передвигается она таким образом, что из точки с координатами (i, j) может попасть только в точки $(i + 1, j)$ и $(i, j + 1)$. Однако не все места на поле безопасны, в некоторых точках зазевавшегося путника ждут неприятности. Гарантируется, что начало и конец пути безопасны. Помогите черепашке определить сколькими путями она сможет добраться до пункта назначения, не проходя через небезопасные точки. Так как ответ может быть очень большой выведите его по модулю $10^9 + 7$.

Решение.

О, это же кузнечик на плоскости! (В трёхмерном пространстве не будет)

(Чтобы отдельно не обрабатывать случаи выхода в отрицательные индексы, можем использовать индексацию с единицы)

Что нам нужно делать здесь? Будем хранить вектор векторов dp, в котором будем считать количество способов добраться до точки (i, j). Если точка является опасной, при считывании запишем $dp[i][j] = -1$.

Пусть мы стоим в безопасной точке (i, j). В эту точку мы можем попасть двумя способами: из $(i - 1, j)$ и из $(i, j - 1)$, если эти точки не являются опасными. Значит, будем прибавлять это количество способов к текущему значению $dp[i][j]$.

В коде можно написать что-то такое:

```
for (int i = 1; i < n + 2; i++) {
    for (int j = 1; j < m + 2; j++) {
        if (dp[i][j] == -1)
            continue;
        if (dp[i - 1][j] != -1)
            dp[i][j] = (dp[i][j] + dp[i - 1][j]) % INF;
        if (dp[i][j - 1] != -1)
            dp[i][j] = (dp[i][j] + dp[i][j - 1]) % INF;
    }
}
```

G. Диггер

Условие.

Когда гонорары известнейшего певца из КБ стали падать, он решил пойти подрабатывать диггером. У кого-то взял карту с отмеченными сокровищами, нашёл лопату. Но копать ему было невероятно лень, поэтому он мог пройти только по одному маршруту на карте. Помогите ему определить этот маршрут.

Карта представляет собой прямоугольник, разбитый на множество прямоугольников-секторов. Певец идет из верхнего левого угла в нижний правый, причем шагнуть может только вправо, вниз или вправо-вниз. Помогите ему найти как можно больше золота.

Решение.

Если вы поняли, как решать предыдущую задачу и задачу про алчного кузнечика, вам не составит труда придумать решение и для этой.

Поэтому расскажу кратко:

В $dp[i][j]$ будем хранить наибольшую сумму, которую может получить певец, добравшись из начальной координаты в (i, j). Пусть мы хотим найти решение для (i, j) клетки, в предположении, что значение для всех клеток левее и выше уже посчитаны. Зададимся вопросом, какая была предыдущая клетка, из которой певец пришёл в (i, j) клетку?

Ответ на него: из клетки, значение в которой является наибольшим. Действительно, при переходе ко всей сумме, которая накопилась за путь до предыдущей клетки, мы прибавим стоимость текущей клетки.

В итоге наша динамика пересчитывается так:

$$dp[i][j] = cost[i][j] + \max(dp[i-1][j], dp[i-1][j-1], dp[i][j-1])$$

Код предоставляется написать самостоятельно.

H. Количество чисел

Условие.

Посчитайте количество чисел с **не более** чем N разрядами, сумма цифр которых равна S. Так как ответ может быть очень большим, выведите его по модулю $10^9 + 7$.

Решение.

В задачах на динамическое программирование полезно пытаться выписать первые сколько-то значений, посмотреть как они зависят.

В этой задаче мы посмотрим зависимость искомого количества чисел от количества разрядов и суммы цифр. Сделаем табличку, где по горизонтале в каждом столбце сумма цифр чисел будет равна $0, 1, 2, 3, \ldots$, а по вертикале количество разрядов чисел будет равно $1, 2, 3, \ldots$.

В каждой ячейке (i, j) будем записывать количество чисел с i разрядами, сумма цифр которых равна j. Попробуйте сами заполнить её и увидеть закономерность, а потом читайте дальше.

Правда попробуйте, полезное упражнение

	0	1	2	3	4	5	6	7	8	9
1										
2										
3										

Заметим, что в первой строке с 0 по 9 будут стоять единицы, так как если сумма для одноразрядного числа равна, например, 5, то само число будет также 5, а больше нет. Начиная с 10-го элемента первой строки будут стоять все нули.

Также очевидно, что в первом столбце во всех строчках, кроме первой будут стоять нули.

Как образовывается ответ для (i, j) клетки в данной задаче? Все возможные числа, сумма цифр в которых равна 0, для i-го разряда можно образовать через $i - 1$ дописыванием после одной цифры после каждого числа.

То есть, если в мы хотим посчитать ответ для $(3, 8)$, мы можем дописать к числам, которые образуются в $(2, 8)$ и приписать к каждому из них в конце 0. Тогда сумма цифр в них не изменится, а количество разрядов увеличится на 1. Так же пересчитываем из $(2, 7)$, приписывая к каждому числу там в конец 1, для $(2, 5)$ припишем 2, и так далее. Помним, что больше 9 у нас нет цифр.

Вот такое будет заполнение таблицы:

	0	1	2	3	4	5	6	7	8	9
1	1	1	1	1	1	1	1	1	1	1
2	0	1	2	3	4	5	6	7	8	9
3	0	1	3	6	10	15	21	28	36	45

Значит, не учитывая ответ для 0 в первой строке, мы можем пользоваться следующей формулой:

$$dp[i][j] = \sum_{k=j-9}^{j} dp[i-1][k]$$

Заметим, что в данной задаче нам нужно было отвечать до 1000 раз, поэтому хорошим ходом будет один раз посчитать таблицу для всех возможных значений, а затем просто выводить на каждый запрос что-то типо $dp[n][s]$.

Теперь вот двумерная динамика

I. Космические бои

Условие.

Василий является стрелком на космическом корабле. В бою его задача нанести по кораблю противника несколько разрезов лазером в фиксированном порядке.

Разрез представляет собой отрезок на плоскости, по которому должен пройти луч лазера, направление прохода по отрезку не фиксировано. Лазер движется по плоскости с некоторой фиксированной скоростью, может мгновенно изменять направление своего движения, но не может телепортироваться из одной точки в другую, необходимо оптимизировать длину пройденного лазером пути. Найдите кратчайший путь лазера, включающий в себя все отрезки в заданном во входных данных порядке.

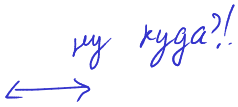
При необходимости лазер может проходить по одной точке неограниченное число раз.

Решение.

Так как порядок разрезов уже задан, можно посмотреть, как i-й разрез зависит от предыдущего.

Будем называть координаты первой точки разреза левыми координатами, а вторые правыми. Можно посчитать, какова будет длина пути лазера, если мы пойдём из левой координаты $i-1$ разреза в левую координату i разреза, или из правой координаты $i-1$ разреза в левую. Затем прибавить расстояние, необходимое для преодоления самого i-го разреза. Так же можно посмотреть и расстояния до правой координаты разреза i.

Значит, в динамике dp будет храниться расстояния пути лазера до текущих левой ($dp[i][0]$) и правой ($dp[i][1]$) точки отрезка. Пересчитываться они будут так:

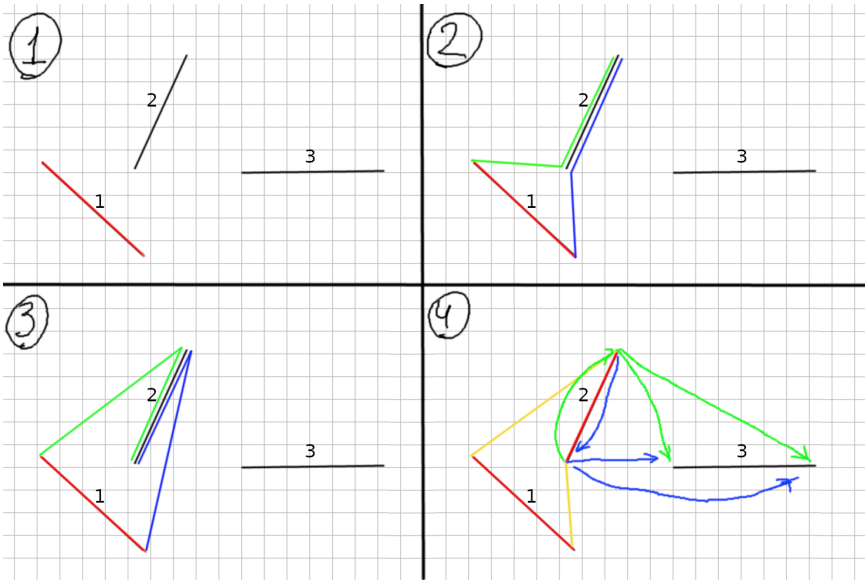


$$dp[i][0] = dist(l[i], r[i]) + \min(dist(l[i-1], r[i]) + dp[i-1][0], dist(r[i-1], r[i]) + dp[i-1][1]),$$

$$dp[i][1] = dist(l[i], r[i]) + \min(dist(l[i-1], l[i]) + dp[i-1][0], dist(r[i-1], l[i]) + dp[i-1][1]),$$

где $l[i]$ и $r[i]$ — это координаты i-го отрезка, $dist()$ — функция, вычисляющая расстояние между двумя точками на плоскости.

Посмотрим на рисунок, с ним должно быть понятнее.



В первой части мы устанавливаем значения $dp[0][0]$ и $dp[0][1]$ равными длине первого отрезка. Во второй части мы высчитываем расстояние до правой координаты через левую и правую предыдущего отрезка. В третьей части делаем то же самое для левой координаты. Четвёртая часть — это зарисовка того, что будет происходить дальше, для третьего отрезка, после того как мы посчитали кратчайшие расстояния до левой и правой координаты второго отрезка.

Вместо вывода:
Динамика просто так не даётся. Чтобы хорошо решать задачи на неё, нужно потренироваться ещё где-нибудь.
Например, на acmp.ru в архиве задач есть тема „Динамическое программирование"
Также много материала есть на informatics.mccme.ru