Hints

vladrus13rus

2021

1 ППО

Тут были хинты, но я их стер

2 Теоркод

Давайте начнем издалека. ~~Что такое теоркод?~~

Вспомним, что такое порождающая матрица (будем называть ее G). Ну, видно по названию, она умеет порождать вектора (будем называть такие вектора закодированные вектора). Потом поймем, что по фактам закодированные вектора были бы никому не нужны, если бы не умели их раскодировать. Поэтому есть операция декодирования. Но опять же, зачем их кодировать, если можно не кодировать (кроме защиты)?

Ответ: для нестабильных соединений. У нас начинают как то зашумляться вектора, и нам нужно как то этому противостоять. Возможно, внесем небольшую избыточность, но код будет хорошо противостоять помехам. Круто, да? Нет, потому что нам дали лабу по этой фигне.

2.1 Labas. A. A?

Здесь нам дана порождающая матричка G, а мы должны написать декодер, кодер и так далее.

Нет, нас наебали, декодер писать не надо. Хех. Хех.

Итак начнем. Важные факты:

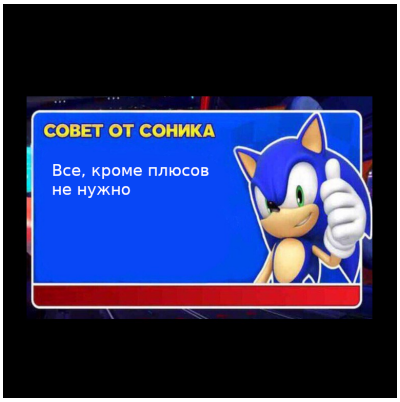
1. 

Рисунок 1. Важный совет от Соника
2. У нас есть файлы, а ввод и вывод можно ускорять (смотри тырнеты, как именно)

1. **Матрицы**. Начнем с самого просто - у нас есть матрицы. Создайте класс матриц, вам будет достаточно создать даже просто `vector<vector<bool> >`. Помните о том, что мы работаем в поле по модулю два - то есть сложение является xor, а умножение является and.

2. **Работа с матрицами**. Так же рекомендую заранее задуматься о том, как вы будете их перемножать (или хотя бы как умножить вектор на матрицу, полезно, можно проверить заранее, что оно работает)

3. **Кодирование**. Как же работает кодирование? Помним, что мы имеем порождающую матрицу, значит, при умножения вектора на порождающую матрицу мы получим закодированный вектор (нет, не матрицу на вектор, нет, не транспонированный вектор на транспонированную матрицу). Просто, не так ли? Это реально все, вы закодировали слово.

4. **ДеКодирование**. Вот теперь поговорим о веселом. Правда ли, что довольно скучно смотреть на матрицы? Поэтому придумали решетки - графы, которые содержат в себе закодированные слова. Очень полезно, не так ли?

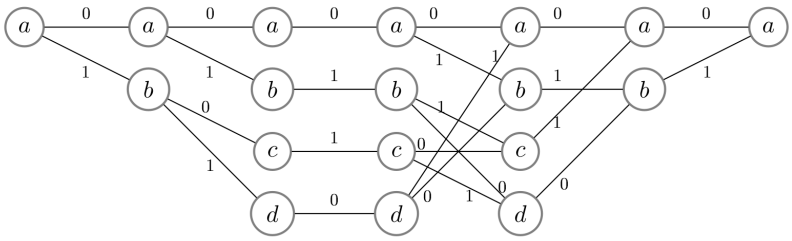


Рисунок 2. Решетка

Мы путешествуем только строго слева направо. Например, я могу собрать слово 101010, но не могу 001000.

Как же их строить? Пока мы не будем их строить, разберемся пока с другими приколами.

(a) **Минимальная Спеновая Форма**. *Это можно прочитать в учебнике, стр. 107* Что это вообще такое? Это такая порождающая матрица в форме такой, что все ее начала ненулевых строк и концы ненулевых строк различны. То есть если мы возьмем например матрицу

```
1 1 1 0 1
0 0 1 0 0
0 1 1 0 1
```

То спэны (начало, конец - 1) будут такие: (0, 3), (2, 1), (1, 3). Здесь совпадают концы строк 1 и 3, так что это не спеновая форма.

Как в нее вообще приходить?

Давайте условимся, что начала строк должны идти последовательно с увеличением номера строки.

Давайте просто пройдемся Гауссом - верхним проходом и снизу.

Верхний проход упорядочит начала по возрастанию и уникализирует их.

А проход снизу вверх сделаем чуть хитрее: будем смотреть сколько в рассмотренном столбце строк с 1. Если это одна, то будем спокойны, просто помечаем, что строка и столбец теперь зачеркнуты, и переходим к следующему столбцу. Если их ноль,

просто переходим к следующему столбцу. Если их больше, то вычитаем из всех, кроме последнего с 1, последний столбец с 1.

Пример:

```
1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 |
1 1 1 1 0 0 0 0 | 0 0 0 0 1 1 1 1 | 0 1 0 1 1 0 1 0 |
1 1 0 0 1 1 0 0 | 0 0 1 1 0 0 1 1 | 0 0 1 1 0 0 1 1 |
1 0 1 0 1 0 1 0 | 0 1 0 1 0 1 0 1 | 0 1 0 1 0 1 0 1 |

| 1 1 1 1 1 1 1 1 | 1 1 1 1 0 0 0 0
| 0 1 0 1 1 0 1 0 | 0 1 0 1 1 0 1 0
| 0 0 1 1 0 0 1 1 | 0 0 1 1 1 1 0 0
| 0 0 0 0 0 0 0 0 | 0 0 0 0 1 1 1 1
```

(b) **Решетка**. Теперь можем декодировать все в решетку, ура! Назовем строки активными в i, если i находится в промежутке спена ($start \leq i \leq finish$). Строим слой за слоем (помните, что граф у нас был слоеный?)

Каждый слой i будет иметь свои информационные символы: это все "номера"строк, что активны в i. То есть вершинки нашего графа — это все комбинации информационных символов (например, слой может быть задан 1 и 3 символом, и у нас могут быть вершинки 00, 01, 10, 11). Конечно же, лучше не генерировать их все, а по надобности, или можно нагенерить себе приколов (2^{64} вершиной выглядит крайне невкусно).

Хорошо, мы сделали вершинки по слоям, у них есть свои идентификаторы — комбинация информационных символов. Теперь надо провести реберки — Берем информационные символы для предыдущего слоя для каждой из вершинок (возможно, добавляем новый информационный символ, так как на новом слое могла появиться новая активная строка) и умножаем на столбец номера слоя в матрице. Берем xor по всем формулам - получаем либо 0, либо 1. Это и есть ребро. Если у нас появился новый информационный символ, то смотрим, куда вести ребро. То есть если у нас были информационные символы m_1, m_3, m_9, добавился m_{15}, то мы ведем из xyz слоя $i-1$ в xyk по ребру произведения (x, y, z, k) на $matrix[i][x], matrix[i][y], matrix[i][z], matrix[i][k]$.

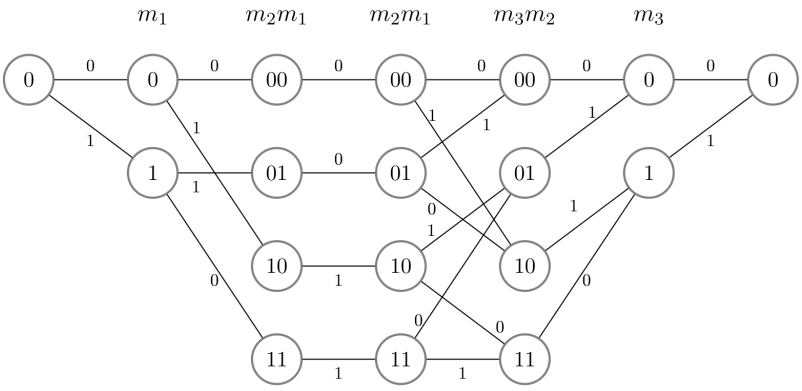


Рисунок 3. Из матрицы выше получается это

Теперь мы можем исправить то, что зашумлено! Зашумить можно так: добавить случайное число.

Это и есть алгоритм Витерби. Давайте просто для каждой вершины будем помечать, какое минимальное расстояние оно имеет от нашего входного. Например, расстояние

слова (1.5, 1, -1, -1, -1.05, 0) от слова (1, 1, -1, -1, -1, 1) будет равно $sum = abs(1.5-1) + abs(1-1) + abs(-1+1) + abs(-1+1) + abs(-1.05+1) + abs(0-1)$. Это стандартная динамика на вершинах, просто идем по слоям, и для каждого слоя все для всех вершин релаксируем значение по ребру со значением x как $relax_value = abs(x-a)$, где a - символ в строке по номеру символа. Не забываем смотреть, откуда мы получили то самое минимальное значение в вершине. Потом просто будем идти по родителям и получим ответ.

Так же, надо учесть, что в кодировщике нам даны символы по формуле $1 - 2x$ (то есть вместо 1 нам на вход дают -1, а вместо 0 дают 1). Это важно, когда мы будем брать разницу между ребром и текущим символом

5) **Симуляция**

Что мы делаем?

(a) Каждый шаг мы генерируем новое слово.

(b) Кодируем его.

(c) Переводим в вещественное по формуле $1 - 2x$ и добавляем случайный шум

(d) Декодируем

(e) Сравниваем с полученным на шаге б. Если ошибка, то прибавляем к количеству. Если ошибок уже больше чем max_error, то брейкаемся. Иначе идем в а)

(f) Выводим количество ошибок на количество **проведенных** итераций.

Немного про шум: это просто нормальное распределение (в `c++` есть класс `std::normal_distribution`, погуглите), $sigma^2 = 0.5 \cdot 10^{\frac{-OBSH}{10}} \cdot \frac{n}{k}$

6) **Важные моменты**

(a) Помните о том, что самая тяжелая часть программы по времени будет операция кодирования и декодирования. Жертва скоростью декодирования в пользу скорости создания графа — путь к провалу

(b) Таким образом, поиск ноды в динамике через map — очень долгая операция. Можно хранить ноду как ссылку, она не будет есть память, однако старайтесь, что бы поиск следующей ноды, предка, текущего оптимального значения для ноды и его обновление было за чистую $O(1)$ (не обязательно каждая, но чем быстрее делаются эти операции, тем быстрее (в разы) делается операция декодирования, это плюс для вас)

(c) Есть полезная штука, как inline функции

(d) Ускорение ввода так же неплохо ускорит вас (но это не точно)

7) **Приложение**

Приложение состоит из файлов, где номер - это номер теста, а имя это:

а) $input \rightarrow n\ k\ G$

б) $span \rightarrow$ матрица в МСФ, затем пары из спэнов

в) $graph \rightarrow$ перечислены все пути графа. Вершины - активные номера порядке возрастания. В конце даны все слои