Дистанционные туры, осень 2016/17
Разбор тура #2

Собрано 8 ноября 2016 г. в 19:34

Содержание

A. Лорел Крик. Вера в себя + bfs

Напишем ровно то, что описано в условии в виде поиска в ширину. Состояние (вершина графа) – где мы стоим, бревно какой длины мы держим в руках, какова сейчас карта болота. Реализовать это можно, например, так:

```
struct state {
  char s[maxn][maxn]; // карта болота
  int length, x, y;
};
bool operator < ( const state &a, const state &b ) {
  return memcmp(&a, &b, sizeof(a)) < 0;
}
map<state, int> distance; // для каждого состояние храним расстояние от начального
```

Понятно, почему решение корректно: bfs по таким состояниям делает ровно то, что написано в условии. К сожалению, непонятно, как оценить число состояний, но сабмит разберётся.

Что нужно было бы делать, если бы решение не проходило по времени?

1. Meet-In-The-Middle: запустить один bfs из начального состояния,
   параллельно запустить второй bfs из конечно состояния, ждать, когда они встретятся.
2. Жадные отсечения, запретить некоторые ходы.

B. Дождь. Бинпоиск + динамика + дерево отрезков

Внутри бинпоиска по ответу максимальная кислотность равна x, нужно проверить, можно ли как-нибудь расположить отрезки.
Заметим, что все клетки делятся на хорошие ($\leq x$) и плохие ($> x$).
Хорошие клетки образуют отрезки, найдём их длины $b_1, b_2, \cdots, b_m$.

Будем прижимать все отрезки максимально влево.
Идём слева направо, считаем динамику $f[n, k]$:
если из первых n отрезков выкинуть k, то их можно расположить в первых $f[n, k]$ клетках.

Переход: n-й отрезок можно пропустить, а можно начать в самой левой клетке от $f[n, k]$ и правее, чтобы было не менее $A[n]$ подряд идущих хороших клеток. Для этого в массиве b нужно на суффиксе найти самый левый элемент $\geq A[n]$. Это умеет одномерное дерево отрезков.

C. Цифровая задача. Динамика

Два случая. Первый: если доступна ровно одна цифра d,
жадно на каждом шаге вычитаем из x число максимальной длины вида $ddd\ldots d$.

Второй: доступны хотя бы две цифры. Самым важным было заметить, что ответ маленький.
Интуиция, почему ответ маленький. Пусть нам доступны цифра 0 и цифра d.
Тогда $x = \sum_i (d10^i a_i)$, где a_i – сколько будет цифр d в i-м разряде слагаемых.
Заметим, что $\frac{x}{d} = \sum_i a_i 10^i$ – разложение числа $\frac{x}{d}$ в 10-ной системе счисления. Поэтому минимальное число слагаемых равно $\max a_i \leq 9$. Число x могло не поделиться на d. Если были доступны другие цифры, то мы можем за не более d операций вида "вычесть из x одну цифру" сделать $d \bmod x = 0$ и ответ будет не более $9 + d$.

Считаем, что ответ не более k=20, длина числа x равна n, у нас (m=10)-ичная система счисления. При сложении суммарный перенос ($carry$) в следующий разряд не более $(m-1)k$.
Строим ответ начиная со старших слагаемых, состояние $0 \leq i \leq n, 0 \leq carry \leq (m-1)k$.
Цифры слагаемых в старших i разрядах уже зафиксированы, $carry$ – какой нужен перенос из младших слагаемых, чтобы в старших i разрядах сошлась сумма. Функция динамики – минимальное число слагаемых в старших разрядах, чтобы получить такое состояние.
Пересчёт динамики: по одной будем фиксировать цифры в i-м разряде слагаемых,
пусть первые $0 \leq j \leq k$ уже зафиксировали.
Всегда можем перебрать +1 цифры, сделать переход в j+1.
Если $j \leq f[i, carry, j]$, можем сделать переход в следующий разряд i+1.
Количество состояний $\mathcal{O}(n(mk)k)$, из каждого m+1 переход, итого $\texttt{Time} = \mathcal{O}(nm^2k^2)$.

D. Подпоследовательности. Жадность, hard

Если вы владете техникой `mincost flow`, то легко решили бы эту задачу за $\mathcal{O}(n^3)$. Как заранее понять, сколько баллов набирает это решение, если в условии ничего об этом не сказано? Заслать решение, проходящее первый тест и содержащее `assert(n ≤ 500)`.

1. Ребро $i \to i+n$ веса -1 и пропускной способности 1.
2. Ребро $i+n \to j$ есть, если $a_i < a_j$, $i < j$, вес ребра равен 0.
3. Исток $\to i$; $i+n \to$ Сток. Оба ребра пропускной способности 1.
4. Ищем Mincost-k-flow на построенном графе за $\mathcal{O}(kn^2)$.

Решение для $K = 1$ является знакомой вам "наибольшей возрастающей подпоследовательностью", которую можно найти за $\mathcal{O}(N \log N)$, используя динамику `minEnd[length]`.

Решение за $O(NK \log N)$ без восстановления ответа получается из предыдущего следующей жадностью. Будем хранить K подпоследовательностей в форме `minEnd[i][length]`. Функция `Insert(i, x)` вставляет в дерево подпоследовательностей `minEnd[i]` число x.

```
1 void Insert(int i, int x) {
2   auto &a = minEnd[i];
3   int j = lower_bound(a, a+n) - a; // бинпоиск из предыдущего решения
4   if (a[j] != +∞) Insert(i+1, a[j]); // новая жадная идея
5   a[j] = x; // пересчёт minEnd из предыдущего решения
6 }
```

Ответ к задаче получается так:

$$ans = \sum_{i=1}^{k} (\max j : \texttt{minEnd[i][j]} \neq +\infty)$$

Это решение работает достаточно быстро при $k \leq \sqrt{n}$. Оно находит только величину ответа, восстановить ответ так просто не получится.

Далее нам понадобится, чтобы все элементы были различны. Пусть это не так, $a_i = a_j, i < j$, тогда будем считать $a_i > a_j$, таким образом мы ввели полный порядок на всех элементах.
Заметим, что массив `minEnd`, который строит наш алгоритм является таблицей Юнга. Каждое число исходного массива является одной из клеток. При вставке очередного числа одна из клеток первой строки выбивается во вторую, из второй в третью и т.д. При этом в каждый момент времени все строки и столбцы таблицы строго возрастают. Наш алгоритм за $\mathcal{O}(n^{3/2} \log n)$ может построить первые $\sqrt{n}$ строк таблицы Юнга.
Построим первые $\sqrt{n}$ столбцов, решив такую же задачу для перевёрнутого массива. Таким образом мы построили всю таблицу, осталось вывести суммарную длину первых k строк.

Восстановление ответа является более сложной задачей. Мы не будем описывать его.