Дистанционные туры, осень 2016/17
Разбор тура #3

Собрано 9 ноября 2016 г. в 22:51

Содержание

A. Билеты в кино. Рюкзак.

Казалось бы обычный рюкзак:
$is[i, x]$ – можно ли посадить первых i групп так, чтобы в первом ряду было x человек.
Что делать с возможностью сажать чётную группу в два ряда?
Те, группы, которые садятся в один ряд, сажаем с одного края, те,
которые садятся в два ряда, сажаем с другого края.
Итого переходы:

$$is[i, x] \Rightarrow \begin{cases} is[i+1, x] & \text{всех во второй} \\ is[i+1, x+a_i] & \text{всех в первый} \\ is[i+1, x+\frac{a_i}{2}] & \text{пополам, к другому краю; делаем этот переход для чётного } a_i \end{cases}$$

Наивная реализация требует $\mathcal{O}(nS)$ времени и $\mathcal{O}(S)$ памяти, где S – суммарное число людей.
Ответ равен $\min\limits_{is[n,x]=1} \max(x, S-x)$. Можно соптимизировать в 64 раза, использовав `bitset`.

B. Дана строка. Динамика и строки.

Решение #1, динамика + z-функция.
$f[i, j]$ – сколько раз подстрока $s[i..j]$ встречается на суффиксе $s[i..n)$.
Ответ на задачу – максимум по всем $f[i, j]$.
Пересчёт: нужно найти m_j, позицию первого вхождения $s[i..j]$ на суффиксе $s[j{+}1..n)$.
Зафиксируем i, насчитаем z-функцию на суффиксе $s[i..n)$ и пойдём двумя указателями j и m_j, внутри будем за $\mathcal{O}(1)$ сравнивать строки на равенство, используя насчитанную z-функцию.
Получили решение за $\mathcal{O}(n^2)$ времени с маленькой константой.
Для хранения динамики нужно $\Theta(n^2)$ памяти $10\,000 \times 10\,000 \times$ `sizeof(short)` $<$ 200mb.

Решение #2, хеши + хеш-таблица.
Переберём длину ответа k. Пройдём по всем подстрокам длины k, для каждой насчитаем полиномиальный хещ и в хеш-таблицы для хеша будем хранить пару – количество непересекающихся вхождений, правый конец последнего.
Это решение использует $\Theta(n)$ памяти.
Реализация с хеш-таблицей использует $\Theta(n^2)$ памяти с большой константой.
Можно хеш-таблицу заменить на сортировку, тогда будет $\Theta(n^2 \log n)$ с маленькой константой.

C. Пекарня. `multiset<pair<int,int>>`

Идём по дням, едим самые дешёвые буханки хлеба, храним список предложений
в формате "$\langle cost_i, k_i \rangle$ – есть ещё k_i буханок стоимости $cost_i$ в запасе".
Храним, конечно, в `multiset`, упорядоченном по $cost_i$.

1. Добавим произведённую партию $\langle f_i, c_i \rangle$.
2. Съедим самые дешёвые пары суммарного объёма d_i.
3. Оставим самые дешёвые пары суммарного объёма g_i.
4. Увеличим все $cost_j$ в `multiset` на e_i.

Последнюю операцию можно делать за $\mathcal{O}(1)$: храним переменную `add` – сколько мы хотим ко всем добавим. Теперь, чтобы положить в `multiset` x, нужно положить $x -$ `add`.

Время работы $\mathcal{O}(n \log n)$, используемая память $\mathcal{O}(n)$.

D. Созвездие краба. Перебор с отсечениями.

Идея #1: можно округлить e_i до точности $\frac{1}{x}$, привести к целым от 0 до x и написать ужасную динамику по дереву, в которой будет $n^2k(nx)$ состояний, где nx – максимальное значение $\sum_i e_i$. $f[v, k, m, sum]$ – минимальное $\prod_j m_j$ в поддереве v, если отрезали уже k рёбер, обрубок с корнем в v содержит m вершин, $\sum_i e_i = sum$.

Идея #2: перебор с отсечениями.
Перебираем рёбра дерева снизу вверх, каждое или отрезаем, или нет.
Чтобы быстро пересчитывать ответ, для каждой v поддерживаем размер не отрезанной части поддерева $size[v]$ и общее количество неотрезанных вершин $rest$.

```
1  // answer -- текущий оптимальный ответ
2  // i -- номер текущего ребра, которое мы или отрезаем, или нет
3  // k -- количество компонент, на которые нужно разбить rest вершин
4  // sum -- сумма весов уже отрезанных рёбер
5  // product -- произведение размеров уже отрезанных компонент
6  void go(int i, int k, double sum, double product) {
7    // Самое главное отсечение: по ответу, для этого нужно оценить, какой ответ мы получим
8    if (answer + ε >= (sum + min_weight[i] * (k-1)) * product * pow(rest / k, k)))
9      return;
10   ...
11 }
```

Далее делаем два рекурсивных вызовов. Отсечение по ответу работает лучше, если у нас уже дана хорошая оценка $answer$. Можно написать бинпоиск по ответу, внутри которого проверять, “можно ли получить ответ больше x”.