SPb HSE, ПАДИИ, 1 курс, весна 2023/24
Практика по алгоритмам #6

Динамика

22 февраля

Собрано 22 февраля 2024 г. в 12:24

Содержание

Динамика

1. **Какая-то линейная динамика**

   Представить число n в виде суммы min числа кубов. $\mathcal{O}(n)$.

   Какое состояние, какие переходы? Начальное и конечное состояния? Как сделать dp вперёд? Как назад? Как лениво? По какому графу мы ходим?

2. **Ацикличный граф.**

   В DAG-е найти самый длинный путь.

3. **Рюкзак с ценами**

   Даны n предметов. У каждого есть цена v_i и вес w_i.
   Найти max стоимость, которую можно набрать предметами суммарного веса $\leqslant S$.

   (a) Время $\mathcal{O}(nS)$, память $\mathcal{O}(S)$.
   (b) Два рюкзака размера S. Время $\mathcal{O}(nS^2)$, память $\mathcal{O}(S^2)$.

4. **Ленивость**

   Есть $n \leqslant 10^{18}$ котят. Нужно посчитать число групп, на которые поделятся котята, если они делятся на группы $\lfloor \frac{n}{2} \rfloor$ и $\lceil \frac{n}{2} \rceil$ при $n > k$ и не делятся при $n \leqslant k$.
   Доказать, что при решении ленивой динамикой будет $\mathcal{O}(\log n)$ состояний.

5. **Разбиение на палиндромы**

   Разбить строку на минимальное число палиндромов за $\mathcal{O}(n^2)$ времени и $\mathcal{O}(n)$ памяти.

6. **Редакционное расстояние**

   Даны две строки. За минимальное число операций «удаления одного символа», «вставки одного символа» и «замены символа на другой символ» получить из первой строки вторую. $\mathcal{O}(nm)$ времени и памяти с восстановлением, где n и m – размеры строк.

7. **Наибольшая общая возрастающая подпоследовательность**

   Даны a, b. Найти наибольшую общую возрастающую подпоследовательность a и b.

   a) $\mathcal{O}(n^4)$.
   b) $\mathcal{O}(n^3)$.
   c) `(*)` $\mathcal{O}(n^2)$.

Разбор задач практики

1. **Какая-то линейная динамика**

   Состояние: x — какое число хотим собрать.

   Храним $f[x]$ — минимальное число кубов в разбиении x.

   Переходы: $x \to x + i^3$.

   Начальное 0, конечное n.

   Назад: `for x : f[x] = 1 +` $\min_{i^3 \leqslant x}$`f[x-i^3]`

   Вперёд: `for x : for i : relax(f[x+i^3], f[x]+1)`

   Ленивая: `dfs(x)` : если ещё не считали, `f[x] = 1+`$\min_{i^3 \leqslant x}$`dfs(x-i^3)`

2. **Ацикличный граф.**

   `f[v] = 1 +` $\max\limits_{x \text{ сосед } v}$ `f[x]`. Порядок пересчёта: ленивая dp.

3. **Рюкзак с ценами**

   `f[j]` – максимальная стоимость в рюкзаке веса j.
   В `f[j]` учитываются первые i предметов.

```
for (int i = 0; i < n; ++i)
    for (int j = S; j >= w[i]; --j)
        f[j] = max(f[j], f[j - w[i]] + v[i]);
answer = f[S];
```

   Если есть два рюкзака, то будет `f[`j_1`,`j_2`]`.
   И три перехода – скипнуть, кинуть в первый, кинуть во второй:
   `f[`j_1`,`j_2`] = max(f[`j_1`,`j_2`], f[`j_1`-w`$_i$`,`j_2`] + v`$_i$`, f[`j_1`,`j_2`-w`$_i$`] + v`$_i$`);`

4. **Ленивость**

   Решение = рекурсия с запоминанием! Состояний будет $2 \log n$. Доказательство:
   На каждом уровне рекурсии работаем с числами из отрезка $[L_i, R_i]$. $L_1 = R_1 = n$.
   Утверждение: $R_i - L_i \leqslant 1$. Переход: $[2k, 2k{+}1] \to [k, k{+}1]$, $[2k{-}1, 2k] \to [k-1, k]$.

5. **Разбиение на палиндромы**

   Решение за $\mathcal{O}(n^2)$ времени.
   $f[i]$ – кол-во палиндромов, на которые можно разбить префикс.
   $f[i] = \min\limits_{j<i, [j,i) \text{ – палиндром}} (f[j] + 1)$.

   Есть два способа быстро проверять, является ли строка палиндромом:

   a) Предподсчитать для всех пар $[j, i)$ за $\mathcal{O}(n^2)$, но это требует квадрат памяти.
   b) Для каждого i найти $r_0[i]$, $r_1[i]$ – длины максимальных палиндромов чётной и нечётной длины с центром в i.

6. **Редакционное расстояние**

`f[i,j]` — редакционное расстояние между `s[0:i)` и `t[0:j)`

Переходы: `[i,j]` → `[i+1,j]`, `[i,j+1]`, `[i+1,j+1]`, в последнем проверяем $s_i = t_j$.

7. **Наибольшая общая возрастающая подпоследовательность (НОВП)**

- Простейшее решение: $f[i,j]$ – max длина НОВП, заканчивающейся ровно в $a[i]$ и $b[j]$.
Переход динамики вперёд: выбрать $i_1 > i$ и $j_1 > j$ такие, что $a[i_1] = b[j_1]$, $a[i_1] > a[i]$.
С ходу получили $\mathcal{O}(n^4)$.
Можно соптимизировать до $\mathcal{O}(n^3)$: делать переход в два этапа $f[i,j] \to g[i_1,j] \to f[i_1,j_1]$.

- Идея на $\mathcal{O}(n^2)$ – правильно выбрать состояние.
$ans[j,i]$ – длина НОВП, кончающейся **ровно** в $a[i]$, и **строго раньше** $b[j]$.
Два перехода в динамике вперёд: пытаемся взять или не взять $b[j]$.
Если не берём, попадаем в $ans[j{+}1,i]$, если берём, попадаем в $ans[j{+}1, next(i,b[j])]$,
где $next(i,b[j])$ – ближайший за $a[i]$ элемент, равный $b[j]$.
Функция $next$ для каждого $b[j]$ при увеличении i насчитывается вторым указателем.

- Можно последнюю идею написать ещё проще динамикой назад.
Будем пересчитывать по слоям: $ans[j] \to ans[j{+}1]$.
Изначально $ans[j{+}1] = ans[j]$, добавляем $b[j]$. Перебираем $i = 0..|a|{-}1$, когда приходим в $a[i] = b[j]$, чтобы получить $ans[j{+}1,i]$ нужно помнить $F = \max\limits_{k:\, a[k]<b[j], k<i} ans[j,k]$.

```
1 F = 0, ans[j+1] = ans[j]
2 for i=0..|a|-1:
3     if a[i] < b[j]: F = max(f, ans[j,i])
4     if a[i] = b[j]: ans[j+1,i] = max(ans[j+1,i], F+1)
```

Ну и напоследок заметим: достаточно хранить только одну строку.

Домашнее задание

Это задачи на тему «динамическое программирование».

Во всех задачах *обязательно* описывайте, что есть состояние, какую функцию считаем для каждого состояния, какие переходы, начальное, конечное, порядок перебора состояний.

3.1. Обязательная часть

1. **(2) Сводный брат Фибоначчи**

   Мы стоим в точке 0, за ход можем сместиться вправо на любое из чисел $1, 2, \ldots, k$.
   Сколько способов прийти в точку n?

   a) **(2)** $\mathcal{O}(nk)$
   b) **(+1)** $\mathcal{O}(n)$

2. **(2) Равномерное разбиение массива**

   Разбить массив длины $n \leqslant 50$ на $k \leqslant 50$ отрезков: сумма квадратов сумм отрезков $\to \min$.
   Пример: $n = 4, k = 3, a = \{7,2,1,4\} \to \{7\} + \{2,1\} + \{4\} \to 7^2 + (2+1)^2 + 4^2 = 74$.

3.2. Дополнительная часть

1. **(3+1) Почти рюкзак**

   У нас есть бесконечное количество предметов каждого типа, всего n типов предметов, i-й тип имеет вес a_i. Можно ли собрать предметы суммарного веса S? Мы уже умеем решать похожую задачу за $\mathcal{O}(nS)$.

   (1) Решить эту за $\mathcal{O}(nS)$.

   (2) Решить эту за $\mathcal{O}(n \min a_i \log n)$.