

SPb HSE, 1 курс, осень 2022/23

Практика по алгоритмам #8

Рекурсивный перебор

10 ноября

Собрано 17 ноября 2022 г. в 10:15

Содержание

1. Рекурсивный перебор	1
2. Разбор задач практики	3
3. Домашнее задание	7
3.1. Обязательная часть	7
3.2. Дополнительная часть	8

Рекурсивный перебор

1. Вас преследуют кучи

Дана бинарная min-куча. Найти k -ую статистику за:

- $\mathcal{O}(k \log n)$
- $\mathcal{O}(k^2)$
- $\mathcal{O}(k \log k)$

2. Го ветвиться!

```

1 void go(int i):
2     if (i == n):
3         int cnt = 0, s = 0;
4         for (int j = 0; j < n; j++) if (used[j]) cnt++, s += a[j];
5         if (cnt == k) best = max(best, s);
6         return;
7     go(i + 1);
8     used[i] = 1, go(i + 1), used[i] = 0;
9 go(0);

```

Что делает этот код? За сколько он работает? Как его ускорить не меняя идеи решения? Как решить ту же задачу жадно?

3. Задачи на перебор

- Посчитайте $\binom{n}{k}$, используя только сложения и вычитания. Воспользуйтесь рекурсией.
 - Добейтесь того, чтобы время работы было $\mathcal{O}(n^2)$.
 - Как написать без рекурсии?
- Дана строка длины $n \leq 20$.
Посчитайте число различных подпоследовательностей длины ровно k .
(*) Предложите технические оптимизации.
- Выведите все строки, состоящие из букв a и b длины ровно n , где нет подстроки $abab$.
- Выведите все возрастающие массивы длины k , состоящие из целых чисел от 1 до C .
- Выведите все массивы длины n из чисел 1, 2, 3, в которых двоек не меньше чем единиц.

4. Скобки

Придумайте перебор, перечисляющий все скобочные последовательности длины n из двух типов скобок. Пример для $n = 4$: $()()$, $()[]$, $[]()$, $[] []$, $([])$, $[(())]$, $(())$, $[[]]$.

- За $\mathcal{O}(3^n)$.
- За $\mathcal{O}(\text{answer})$. Поясните, почему $\text{answer} = \mathcal{O}(2.82^n)$.

5. Ограбление банка и странная функция

Даны n предметов. Придумайте перебор, который позволяет выбрать ровно k предметов, у которых максимально значение $(\sum a_i) \cdot (\sum b_i)$.

- За $\mathcal{O}(2^n)$.
- Пусть a_i и b_i целые $1 \leq C \leq 10$. Добавьте запоминание. За сколько работает теперь решение? 3. (**) Можно сделать крутое запоминание, чтобы получилось $\mathcal{O}(n^3 \cdot C)$. Как?

6. Обыкновенное ограбление банка

Даны n предметов их веса a_i , их стоимости b_i и число S .

Унести какие-то предметы, что $\sum a_i \leq C$ и при этом $\sum b_i \rightarrow \max$.

7. Расписание для студентов

Есть $n \leq 10$ аудиторий, m групп студентов, k преподавателей, $t \leq 10$ слотов времени.

Нужно провести w пар. Для каждой пары известны группа студентов, преподаватель, выбран слот времени. Для каждой группы известен размер, для каждой аудитории вместимость.

Когда у студентов окно, они уходят разлагаться в бесконечно большую студкомнату.

Ваша задача — распределить аудитории.

1. Провести все пары как-нибудь.

2. Провести все пары, минимизируя отвратительность расписания: суммарное количество перемещений между аудиториями.

3. Как добавить в перебор запоминание? За сколько теперь перебор работает?

8. Казалось бы, причём здесь рекурсивный перебор?

Даны три массива a , b , c . Длины до 10^6 . Рассмотрим все суммы $a_i + b_j + c_k$.

Найдите $k \leq 10^6$ минимальных сумм.

Указание: сделайте бинпоиск по ответу. Что дальше?

Решите для $m \leq 10$ массивов.

(*) За сколько точно это работает, если массивов m ?

(**) Решите, если массивов до 10^6 и их суммарная длина до 10^6 ?

9. Микросхемы

Придумайте любое корректное решение для задачи.

Даны $n \leq 10$ квадратных микросхем, их размеры — r_i .

Разместить микросхемы без наложений на квадратной плате минимального размера.

10. (*) Статистика в массиве

Найти k -ую статистику за $n + \mathcal{O}(k \log n)$ сравнений.

11. (*) Оптимизации куч

а) Есть куча, которая умеет `Add`, `Merge`, `ExtractMin` за $\mathcal{O}(\log n)$, `Build` за $\mathcal{O}(n)$.

На ее основе построить структуру данных, которая умеет все то же, но `Add` за $\mathcal{O}(1)$.

б) Есть куча, которая умеет `Add` за $\mathcal{O}(1)$, `Merge`, `ExtractMin` за $\mathcal{O}(\log n)$, `Build` за $\mathcal{O}(n)$.

На ее основе построить структуру данных, которая умеет все то же, но `Merge` за $\mathcal{O}(1)$.

Разбор задач практики

1. Вас преследуют кучи

- a) $\mathcal{O}(k \log n)$. k раз вынимаем минимум.
- b) $\mathcal{O}(k^2)$. Минимум в корне. Второй минимум в одном из детей корня. Третий либо в детях меньшего сына, либо в большем сыне.
Итого у нас на шаге i есть i кандидатов, выбираем минимального и делаем его детей кандидатами. Кандидаты хранятся в списке, а минимум мы ищем линейным проходом.
- c) $\mathcal{O}(k \log k)$. То же, что в (b), но храним кандидатов в (другой) куче вместо списка.

2. Го ветвиться!

Код перебирает все 2^n подмножеств, выбирает максимальной суммы размера ровно k .
Ускорение в n раз: насчитывать размер и сумму по ходу рекурсии.
Жадное решение за $\mathcal{O}(n)$: взять k максимальных.

3. Задачи на перебор

- a) Мы можем рекурсивно перебрать все подпоследовательности.
Каждый символ или берём в ответ, или нет, всего подпоследовательностей 2^n .
Чтобы проверять различность, добавим их все в `set`.

```

1 vector<int> a(n); // исходные данные
2 vector<int> res(k); // текущая подпоследовательность
3 set<vector<int>> s; // не паримся, кидаем вектора, ответ на задачу - s.size()
4 int calc(int ni, int ki):
5     if (ni == n):
6         // это место можно улучшить, строить так, чтобы к концу было ровно k, но рано
7         if (ki == k) s.insert(res);
8         return;
9     calc(ni+1, ki); // пропускаем
10    if (ki < k)
11        res[ki] = a[ni], calc(ni+1, ki+1); // берём

```

Знакомые с хешами могут `set<vector<int>>` превратить в `unordered_set<int64_t>`, хеш, конечно, пересчитывается за $\mathcal{O}(1)$ по ходу рекурсии.

- b) $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$. Считаем рекурсивно.

В рекурсии много раз вычисляем одно и то же \Rightarrow будем запоминать уже посчитанные результаты и проверять «если уже посчитано, можно сразу вернуть результат».
Для каждой пары $1 \leq i \leq n$, $1 \leq j \leq k$ считаем функции не более одного раза $\Rightarrow \mathcal{O}(nk)$.

```

1 int binom(int n, int k):
2     if (n < k) return 0;
3     if (n == 1) return 1;
4     if (result[n][k] != -1) return result[n][k];
5     return result[n][k] = binom(n-1, k-1) + binom(n-1, k); // так можно

```

Без рекурсии: два цикла `for` по возрастанию n и k , для меньших всегда уже посчитано.

- c) Рекурсивно перебираем строки, по ходу рекурсии сразу выходим, если суффикс = `abab`.
Решение оптимально, так как работает за $\mathcal{O}(\text{ответа})$. Быстрее вывести не получится.

d) Очень похоже на пункт (а) для $12 \dots n$, правда?

Будем внимательно следить, чтобы работало за $\mathcal{O}(\text{ответа})$.

Улучшение: переход «пропускаем» в коде выше только `if (ki + n-ni-1 >= k)`.

Для разнообразия напишем ещё и другую версию рекурсии.

```
1 vector<int> res(k); // найденные k чисел от 1 до C
2 int calc(int i): // перебираем очередное число, i-е
3     if (i == k) { cout << res; return; }
4     for (res[i] = !i ? 1 : res[i-1]+1; res[i] + (k-i-1) <= C; res[i]++)
5         calc(i+1);
```

e) Ещё одна рекурсия. Опять же важно написать за $\mathcal{O}(\text{ответа})$.

Отсечение: `if (cnt2 + n - i < cnt1) return;`

4. Скобки

Перебор: каждая следующая или открывающая (два варианта), или парная закрывающая (последней открытой (+1 вариант)). Итого 3^n , баланс проверяем по ходу рекурсии.

За $\mathcal{O}(\text{ответа})$: по ходу рекурсии следим, что $i + \text{balance} \leq n \Rightarrow$ всё успеем закрыть.

Число скобочных из одного типа скобок длины $2n$: $\frac{1}{n} C_{2n}^n$, длина n : $\frac{1}{n/2} C_n^{n/2} = \Theta(\frac{1}{n^{3/2}} 2^n)$.

Для каждой пары скобок нужно выбрать тип, 2 варианта $\Rightarrow \Theta(\frac{1}{n^{3/2}} 2^n \cdot 2^{n/2}) \approx 2.82^n$ вариантов.

5. Ограбление банка и странная функция

2^n . Рекурсивный перебор «берём/не берём i -й предмет».

По ходу рекурсии насчитываем $\sum a_i$ и $\sum b_i$.

```
1 int go(int i, int taken, int suma, int sumb):
2     if i==n: ... return
3     go(i+1, taken, suma, sumb)
4     go(i+1, taken+1, suma+a[i], sumb+b[i])
```

Запоминание: параметра рекурсии, $i, k, \sum a_i, \sum b_i \Rightarrow \text{map}<\text{tuple}<\text{int}, \text{int}, \text{int}, \text{int}>, \text{int}>>$.
 $\sum a_i \leq n \cdot C \Rightarrow$ время работы $\mathcal{O}(n \cdot n \cdot nC \cdot nC) = \mathcal{O}(n^4 C^2)$.

Крутое запоминание: для фиксированного i, k , и $\sum a_i$ нужны помнить не все варианты «какое $\sum b_i$ мы можем получить», а только максимальный. Подробнее в следующих сериях.

6. Обыкновенное ограбление банка

Если мы напишем обычную рекурсию, получим `go(i, sumA, sumB)` и

запоминание за $n \cdot S \cdot \max \sum b_i$. Можно умнее:

```
1 int maxSumB(int i, int S): // максимальная \sum b_j для j = i..n-1, если осталось S места
2     if (i == n) return 0
3     if (a[i] > S) return maxSumB(i+1, S)
4     return min(maxSumB(i+1, S), b[i]+maxSumB(i+1, S-a[i]))
```

Запоминание в таком перебор даст $\mathcal{O}(n \cdot C)$.

7. Расписание для студентов

Провести пары как-нибудь.

Жадность «самой большой группе студентов самую большую аудиторию».

Перебор. Перебираем все пары по возрастанию времени.

Для каждой аудитории помним, кто и когда последним там занимался.

Функция перебора возвращает минимальный суммарный штраф от момента i до конца дня.

```

1 int minFee(int i, vector<pair<int,int>> last):
2   if (i == w) return 0; // уже провели все пары
3   auto [students, teacher] = p[i]; // для кого и кто читает i-ую пару
4   int best = 0;
5   for (int where = 0; where < m; where++):
6     auto last_ = last;
7     last_[where] = {students, teacher}; // новая версия.
8     auto [s,t] = last[where]; // кто там сейчас
9     best = min(best, minFee(i+1, last_) + (students!=s) + (teacher!=t));
10  return best

```

Запоминание — $\text{map}\langle\text{pair}\langle\text{int},\text{vector}\langle\text{int}\rangle\rangle,\text{int}\rangle$ от i и $last$.

Оптимизация: можно было не создавать новый $last$, а делать $\text{swap}(last[where], p[i])$.

8. Казалось бы, причём здесь рекурсивный перебор?

Сделаем бинпоиск по ответу. Осталось научиться перебирать все числа $\leq x$ в любом порядке. Важно перебирать за $\mathcal{O}(\text{ответа})$ и сразу останавливаться, если ответ $\geq k$.

Отсортируем массивы. Пусть для удобства массивов не 3, а сразу m .

$\text{go}(i, x)$: перебрать все $a[i, ?] + a[i+1, ?] + a[i+2, ?] + \dots \leq x$.

\exists хотя бы один ответ iff сумма первых $\leq x \Rightarrow$ поддерживаем сумму первых на суффиксе.

Можно заранее упростить задачу, сразу вычесть $\forall i, j \ a[i, j] -= a[i, 0] \Rightarrow$

сумма минимальных = 0 \Rightarrow просто следим в рекурсии, что $x \geq 0$.

```

1 void go(int i, int x):
2   if answer >= k: return // как только нашли k меньших, выходим из рекурсии
3   if (i == m):
4     answer++
5     return
6   for (j = 0; j < len(a[i]) and a[j] <= x; j++)
7     go(i + 1, x - a[j])

```

Работает всё это удовольствие за $\mathcal{O}(km \log C)$, где $\log C$ от бинпоиска по ответу.

Сама рекурсия перебирает $\leq k$ вариантов, каждый за m спуска вглубь.

(**) Можно сделать $\mathcal{O}(k \log m \log C)$.

Нам хочется, чтобы рекурсия всегда ветвилась, тогда суммарное время рекурсии будет $\mathcal{O}(\text{числа листьев})$, а листьев $\leq k$. Сейчас $a[i, 0]=0$, обозначим $b[i]=a[i, 1]$.

Чтобы рекурсия разветвилась, спустимся сразу на такой уровень рекурсии j , что $b[j] \leq x$.

Итого задача про массив b : для данных i и x найти $\min j \geq i: b_j \leq x$.

Её можно решать за $\mathcal{O}(\log |b|)$ деревом отрезков, мы научимся во втором семестре.

9. Микросхемы

Будем прижиматься вправо-вниз. Кладя очередную, перебираем кого именно кладём и прижатое-вправо-вниз положение, куда мы её положим. Если уже положено i схем, крайних положений не больше $(i+1)^2$ и для x , и для y не более $i+1$ интересных вариантов.

Оптимизация: запоминание.

Не рабочая оптимизация: кладём микросхемы снизу-вверх ($y_{next} \geq y_{prev}$).

10. (*) Статистика в массиве

Решение #1. Построим дерево отрезков или weak кучу (доп задача из последнего дз) за ровно $n-1$ сравнение, затем k раз достанем минимум за $\log n$.

Решение #2. Сперва `random_shuffle`, затем идём и поддерживаем k минимумов.

Очередной элемент x_i сравниваем с худшим из k минимумов y . $\Pr[x_i \leq y] = \frac{k}{i}$, вероятность того, что среди первых i элементов один из k лучших стоит в конце.

Итого: мы всегда сделаем первое сравнение и ещё $\approx k \cdot (\ln n + \mathcal{O}(1))$ раз нужно что-то делать.

В общем случае первое решение кажется удобнее. Для $k = 2$ второе сильно проще.

11. (*) Оптимизации куч

a) `Add` за $\mathcal{O}(1)$. Добавляет новые элементы в отдельный список `freshItems`.

Если пришел `ExtractMin` или `Merge`, делаем `Merge(q, Build(freshItems))`.

Потенциал $\varphi = |\text{freshItems}|$.

b) Дана куча `Heap`, которая умеет `add` за $\mathcal{O}(1)$, `merge` и `extractMin` за $\mathcal{O}(\log n)$.

Создадим новую рекурсивную кучу `NewHeap` по схеме «куча куч» [\[wiki\]](#) [\[Brodal,p17\]](#):

```
1 struct NewHeap<T>:
2     T min;
3     Heap<NewHeap<T>> q;
```

`Heap` мы можем создать \forall типа с оператором «<>», `NewHeap<T>` сравниваются по `T min`.

`NewHeap.add` сводится к `Heap.add`, $\mathcal{O}(1)$.

`NewHeap.merge` сводится к `Heap.add`, $\mathcal{O}(1)$.

`NewHeap.extractMin` можно за $\mathcal{O}(\log n)$ сделать следующим образом:

```
1 void extractMin(NewHeap heap): // O(log n)
2     result = heap.min
3     child = extractMin(heap.q) // метод Heap, O(log n)
4     heap.min = child.min
5     heap.q = merge(heap.q, child.q) // метод Heap, O(log n)
```

Домашнее задание

3.1. Обязательная часть

1. (3) Числа Стирлинга

Если интересно, числа Стирлинга второго рода – количество способов разбиения множества из n элементов на k непустых подмножеств. Сейчас нам важно, что считать их можно по формуле:

$$S(n, k) = \begin{cases} kS(n-1, k) + S(n-1, k-1) & \text{для } n > 0, k > 0 \\ 1 & \text{для } n = k = 0 \\ 0 & \text{иначе} \end{cases}$$

Ваша задача – написать функцию `int S(int n, int k)`, которая считает число Стирлинга для $n, k \leq 100$, предполагая, что ответ помещается в `int`.

2. (2) Плюс-минус халява

Выведите за \mathcal{O} (размера ответа) все последовательности из n плюс-минус единиц, суммы которых лежат в $[L, R]$.

3. (3) Перебор разбиений

Предложите алгоритм, который выводит в произвольном порядке все разбиения множества $\{1, 2, \dots, n\}$ на подмножества за \mathcal{O} (длины ответа).

Пример: $n = 3 \rightarrow \{\{1, 2, 3\}\}, \{\{1, 2\}, \{3\}\}, \{\{1, 3\}, \{2\}\}, \{\{1\}, \{2, 3\}\}, \{\{1\}, \{2\}, \{3\}\}$.

4. (3) Статистика на суммах подмножеств

Дано множество из n положительных элементов.

Найдите k -ую порядковую статистику среди сумм всех 2^n подмножеств. $n \leq 100, k \leq 10^5$.

(+1) $n \leq 10^5$.

Указание: посмотрите решение похожей задачи из практики.

5. (1.5) Древний вступительный тест

Сколько способов из клетки 0 дойти до клетки n , на каждом шаге, прыгая вперёд на 1, 3 или 4 клетки?

Обозначим такое количество за f_n и заметим, что $f_0 = 1$, а $\forall n > 0, f_n = f_{n-1} + f_{n-3} + f_{n-4}$.

Найти f_{30} . Ответ на задачу — число. Текст решения — одно число.

В отличии от констестов у вас нет 100 попыток... что же делать?

3.2. Дополнительная часть

1. (3) Допинг

Дано $n \leq 60$ допингов, i -й допинг увеличивает силу на a_i . Допинги можно смешивать.

Дано $k \leq 60$ спортсменов, которых нужно победить и k наших спортсменов.

У каждого известна сила. Суммарная сила всех соперников, которых нужно победить ≤ 60 .

Сколько способов распределить **все** допинги между нашими спортсменами так, чтобы можно было так назначить соперников, чтобы в каждой паре наш спортсмен победил?

p.s. Если не любите допинги/спортсмены, представляйте артефакты/герои.

2. (3) Длинные доминошки

Посчитайте за $\mathcal{O}^*(3^n)$ количество замощений грида $n \times n$ с дырками фигурами 1×3 . $\mathcal{O}^*(3^n)$ означает, что нам не важна полиномиальная часть времени работы, только экспонента.

3. (4) Парковка

Нужно на парковке $n \times n$ расставить максимальное число машинок 1×2 , чтобы до каждой машины можно было дойти из входа в парковку, клетки 1,1, только по пустым клеткам.

Эта задача имеет конструктивное решение. Нам оно не интересно.

Задача на тему «рекурсивный перебор с запоминанием».

Полный балл получит любое решение за $2^{\mathcal{O}(n)}$.