

SPb HSE, 1 курс, осень 2021/22

Практика по алгоритмам #3

Структуры данных, стек

17 сентября

Собрано 18 сентября 2021 г. в 09:45

Содержание

1. Структуры данных, стек	1
1.1. Задачи про суммы	1
1.2. Изучаем STL	1
1.3. Структуры данных, стек	1
2. Разбор задач практики	3
2.1. Задачи про суммы	3
2.2. Изучаем STL	3
2.3. Структуры данных, стек	3
3. Домашнее задание	8
3.1. Обязательная часть	8
3.2. Дополнительная часть	8

1.1. Задачи про суммы

Этот блок нужен, чтобы размяться и решить задачу 8, которая пригодится в кучах.

1. $\sum_{k=0}^{\infty} \frac{n}{2^k}$

4. $\sum_{k=1}^n \left(\frac{1}{k} - \frac{1}{k+1} \right)$

7. $\prod_{k=1}^n (2 \cdot 4^k)$

2. $\sum_{k=0}^{\infty} \frac{n}{1.01^k}$

5. $\prod_{k=1}^n \frac{k}{k+1}$

8. $\sum_{k=0}^{\infty} \frac{k}{2^k}$

3. $\sum_{k=1}^n \frac{1}{2^{k-1}}$

6. $\prod_{k=2}^n (1 - k^{-2})$

1.2. Изучаем STL

Проведём эксперимент.

Подумаем про аллокацию памяти и её перегрузку.

```
1 queue<int> q;
2 list<int> l;
3 deque<int> d;
4 vector<int> v;
5 stack<int> s;
6 queue<int, list<int>> q2;
```

- Пушбеки в какую структуру будут работать быстро/долго?
- Как ускорить?
- У какой из структур есть `operator[]`?

1.3. Структуры данных, стек

1. Очередь с минимумом, часть 2

Поймите код. Что делает, за сколько? Как доказать амортизацию?

```
1 struct Item { int value, id; };
2 struct Qmin {
3     int q_front, q_back; deque<Item> mins;
4     void pop() { if (mins.front().id == q_front++) mins.pop_front(); }
5     void push(int x) {
6         while (!mins.empty() && mins.back().value >= x) mins.pop_back();
7         mins.push_back({x, q_back++});
8     }
9     int get_min() { return mins.empty() ? INT_MAX : mins.front().value; }
10 };
```

2. Амортизация

Придумайте потенциал и докажите амортизированное время работы.

- Очередь с минимумом через два стека работает за $\mathcal{O}(1)$.
- Дек, использующий утроение массива, работает за $\mathcal{O}(1)$.

- с) Обычно вектор при `pop_back` память не освобождает. Давайте сделаем, чтобы вектор на n элементах всегда весил $\Theta(n)$. Придумайте. Докажите амортизированное время $\mathcal{O}(1)$, используйте идею с монетками.

3. Двусвязный список.

Придумайте структуру данных.

Она должна уметь перебирать элементы в обоих направлениях, *если идти с конца* но хранить меньше, чем `struct Node { Node *l, *r; int x; }; Node *head, *tail;`.

4. Частичные суммы

Придумайте для каждого пункта структуру данных, которая умеет

- Много раз сделать `+=` на отрезке, в конце один раз вывести массив.
- Сперва много раз `+=` на отрезке, затем много раз запрос суммы на отрезке.

5. Оптимальный подотрезок

За линейное время найти подотрезок массива.

- С максимальной суммой (два решения).
- С максимальной суммой, длины от A до B .
- С максимальной суммой, содержащий не менее k различных чисел.

6. Задачи на стек

- В массиве найти для каждого элемента ближайший \leq слева за $\mathcal{O}(n)$.
- Дан массив. Найти отрезок $[l, r]: (\min_{i \in [l..r]} a_i) \cdot (r - l + 1) \rightarrow \max$.
- Дана матрица из нулей и единиц. Найти наибольший по площади подпрямоугольник, состоящий только из нулей за $\mathcal{O}(n^2)$.

7. (*) Дек и стеки

Придумайте дек с минимумом со всеми операциями за $\mathcal{O}(1)$. Докажите время работы.

8. (*) Частичные суммы

В каждой целой точке x числовой прямой есть $f[x]$, изначально равная нулю.

Запросы сперва `+=(1, r)`, затем `sum(1, r)`. Координаты запросов целые от 0 до 10^{18} .

9. (*) Стек и ближайшие

- За один проход со стеком найти ближайшие " $<$ " справа и " \leq " слева.
- А теперь **меньший** справа и **меньший** слева.

10. (*) Правильные скобочные подстроки

Дана строка s из скобок, а также число k . Для всех i проверить, что подстрока $s[i..i + k]$ является правильной скобочной последовательностью. $\mathcal{O}(n)$.

- Скобки только одного типа.
- m типов скобок.

11. (**) Стек и 3D

Дана матрица из нулей и единиц в 3D. Найти наибольший белый подпараллелепипед.

Разбор задач практики

2.1. Задачи про суммы

$$1. \sum_{k=0}^{\infty} \frac{n}{2^k} = n \sum_{k=0}^{\infty} \frac{1}{2^k} = 2n.$$

$$2. \sum_{k=0}^{\infty} \frac{n}{1.01^k} = n \cdot \frac{1}{1-1.01} = 101n.$$

$$3. \text{Сумма нечетных} - \text{сумма всех} \text{ минус} \text{сумма четных. } \sum_{k=1}^n \frac{1}{2k-1} = \sum_{k=1}^{2n} \frac{1}{k} - \sum_{k=1}^n \frac{1}{2k} = \ln 2n - \frac{1}{2} \ln n + \Theta(1) = \ln \sqrt{n} + \Theta(1).$$

$$4. \sum_{k=1}^n \left(\frac{1}{k} - \frac{1}{k+1} \right) = 1 - \frac{1}{n+1}$$

$$5. \prod_{k=1}^n \frac{k}{k+1} = \frac{n!}{(n+1)!} = \frac{1}{n+1}$$

$$6. \prod_{k=2}^n (1 - k^{-2}) = \prod_{k=2}^n \frac{(k-1)(k+1)}{k^2} = \prod_{k=2}^n \frac{k-1}{k} \cdot \prod_{k=2}^n \frac{k+1}{k} = \frac{n+1}{2n}$$

$$7. \prod_{k=1}^n (2 \cdot 4^k) = 2^n \cdot 2^{2 \sum_{k=1}^n k} = 2^{n+n(n+1)} = 2^{n(n+2)}$$

$$8. \text{Пусть } S = \sum_{k=0}^{\infty} \frac{k}{2^k}, \text{ тогда } S = \sum_{k=1}^{\infty} \frac{k}{2^k} = \frac{1}{2} \sum_{k=1}^{\infty} \frac{k}{2^{k-1}} = \frac{1}{2} \sum_{k=0}^{\infty} \frac{k+1}{2^k} = \frac{1}{2} \left(S + \sum_{k=0}^{\infty} \frac{1}{2^k} \right) = \frac{1}{2} (S + 2) \Rightarrow S = \frac{1}{2} (S + 2) \Rightarrow S = 2.$$

$$\begin{aligned} \text{Способ \#2: } \sum_{k=0}^{\infty} \frac{k}{2^k} &= \begin{array}{cccccc} \frac{1}{2} & + \frac{1}{4} & + \frac{1}{8} & + \frac{1}{16} & + \frac{1}{32} & + \dots \\ & + \frac{1}{4} & + \frac{1}{8} & + \frac{1}{16} & + \frac{1}{32} & + \dots \\ & & + \frac{1}{8} & + \frac{1}{16} & + \frac{1}{32} & + \dots \\ & & & + \frac{1}{16} & + \frac{1}{32} & + \dots \\ & & & & + \frac{1}{32} & + \dots \\ & & & & & \dots \end{array} \\ &= 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots = 2 \end{aligned}$$

2.2. Изучаем STL

`vector`, `list`, `deque` – структуры данных.

`queue` и `stack` – обёртки над ними.

Дольше всего из трёх структур работает `list<int>`, т.к. на каждый `push` вызывается `operator new`. Раз в 10 дольше.

У `deque` операции `push` чуть быстрее чем у `vector`.

Чтобы ускорить `list`, нужно задать свой аллокатор. Самый простой способ – переопределить `operator new`.

Время работы `queue` и `stack` зависит от того, через что они реализованы. По умолчанию через `deque`.

У дека и вектора есть `operator []`, у остальных нет.

2.3. Структуры данных, стек

1. Очередь с минимумом, часть 2

`mins.front()` хранит минимум в очереди и его индекс m_1 . После него в `mins` лежит минимум среди элементов очереди с индексами ($m_1..q_back$) и его индекс m_2 , и так далее. Элементы `mins` упорядочены по возрастанию и значений, и индексов.

Делая `pop`, смотрим, не вынули ли индекс минимума. Если вынули, ничего, после него в `mins` новый минимум.

Делая `push`, обновляем минимум во всё большем хвосте очереди, пока новый элемент не больше последнего минимума.

Время работы линейно, так как каждый элемент не более одного раза добавлен и удалён из `mins`, других операций нет.

Потенциалом: $\varphi = |\text{mins}|$. $a(\text{push}) = t(\text{push}) + \Delta\varphi = \mathcal{O}(1)$, сколько вынули, на столько уменьшился φ .

Заметим, что если явно хранить всю очередь `q`, то можно проверять не индексы, а сами значения. Но данная реализация использует в среднем $\mathcal{O}(\log n)$ памяти на случайных данных.

2. Амортизация

a) **Очередь с минимумом.** $\varphi = |s_{\text{push}}|$.

$$a(\text{reverse}) = t(\text{reverse}) + \Delta\varphi = |s_{\text{push}}| + (0 - |s_{\text{push}}|) = 0 \Rightarrow a_i = \mathcal{O}(1).$$

φ всегда неотрицателен.

b) **Дек.** $\varphi = -3\text{capacity}$.

$$a(\text{rebuild}) = t(\text{rebuild}) + \Delta\varphi = 3\text{capacity} + 3(\text{capacity} - 3\text{capacity}) \leq 0 \Rightarrow a_i = \mathcal{O}(1).$$

$$\varphi_0 - \varphi_n \leq 3n.$$

c) **Вектор.** Когда уменьшать память в `pop_back` в 2 раза? Когда элементов в 4 раза меньше. За `push_back` кладем 2 монетки: одну себе и одну в первую половину. Тогда хватит монет на копирование при увеличении.

За `pop_back` кладем 1 монетку в первую четверть. Тогда хватит монет на копирование при уменьшении.

3. Двусвязный список

Храним `sum = (size_t)l ^ (size_t)r` вместо `l, r`.

Когда приходим в `v` из `prev`, имеем `next = (Node*)(v->sum ^ (size_t)prev)`.

Новая структура умеет все, что обычный двусвязный список, кроме вставки и удаления из середины за $\mathcal{O}(1)$.

4. Частичные суммы

a) **Много раз сделать += на отрезке, в конце один раз вывести массив.**

Прибавление на отрезок – это два прибавления на суффикс. К элементу `a[i]` прибавляется все, что прибавлено к суффиксу, начинающемуся с `j < i`. Итого:

Прибавление: `a[l] += x, a[r + 1] -= x`

Восстановление: `for i: a[i] += a[i - 1]`

b) **Сперва много раз += на отрезке, затем много раз запрос суммы на отрезке.**

Вычисляем массив, как в предыдущем пункте. Считаем префиксные суммы. Ответ на запрос: `pref[r + 1] - pref[l]`.

Можно просто два раза подряд вызвать стандартную функцию `partial_sum(a + 1, a + n + 1, a + 1)`, лежит в `<numeric>`.

5. Оптимальный подотрезок

а) С максимальной суммой.

Способ #1.

```

1 l = 0, sum = 0, answer = 0;
2 for (r = 0; r < n; r++) {
3     if ((sum += a[r]) < 0)
4         l = r + 1, sum = 0;
5     answer = max(answer, sum);
6 }

```

Способ #2.

```

1 l = 0, r = 0, sum = 0; // [l; r)
2 while (r <= n)
3     if (sum >= 0)
4         answer = max(answer, sum += a[r++]);
5     else
6         answer = max(answer, sum -= a[l++]);

```

Для каждого r ищем отрезок с максимальной суммой, оканчивающийся на r .

Пусть для $r - 1$ оптимален отрезок $[l_{r-1}, r - 1]$. Тогда для r либо $l_r = l_{r-1}$, либо $l_r = r$.

Будь для r оптимально другое начало, оно было бы оптимально и для $r - 1$, сумма на этих отрезках отличается на $a[r]$, не зависящее от l .

Способ #3. Найдем частичные суммы $\text{pref}[]$. Сумма на отрезке $[1, r]$ равна $\text{pref}[r + 1] - \text{pref}[1]$, при фиксированном r нужно минимизировать $\text{pref}[1]$. Перебираем r в порядке возрастания. Поддерживаем минимум $\text{pref}[1]$ по всем пройденным $1 \leq r$.

б) С максимальной суммой, длины от L до R .

Теперь нас интересует минимум среди $\text{pref}[1]$ на отрезке $[r - R, r - L]$. Поддерживаем очередь с минимумом, на каждом шаге один элемент оттуда уходит, и один приходит.

в) Содержащий не менее k различных чисел, числа небольшие.

Теперь нас интересует минимум среди $\text{pref}[1]$ на отрезке $[0, s-1]$, где s – последняя позиция, что $[s, r]$ содержит хотя бы k различных чисел.

При увеличении r позиция s только растёт, её можно продолжать двигать со значения s прошлой итерации, тогда суммарно по всем итерациям выйдет $\mathcal{O}(n)$.

Как проверять, можно ли подвинуть s вперед? Поддерживаем массив-счетчик каждого числа.

После $r++$ делаем $\text{count}[a[r]]++$, перед $s++$ делаем $\text{count}[a[s]]--$.

Также храним число ненулевых ячеек count , меняя его каждый раз, когда только что измененная ячейка начала или перестала быть нулем.

С использованием хеш-таблицы вместо массива можно делать это и для больших чисел.

6. Задачи на стек

a) Для каждого элемента ближайший меньший слева за $\mathcal{O}(n)$.

```

1 for (int i = 0; i < n; ++i) {
2     while (!stack.empty() && a[i] <= a[stack.top()]) stack.pop();
3     if (!stack.empty()) left[i] = stack.top();
4     stack.push(i);
5 }

```

В стеке и `left` хранятся индексы элементов.

Инвариант: для каждого элемента стека перед ним в стеке идет ближайший к нему слева меньший. Когда видим элемент $a[i]$, на вершине стека лежит $a[i - 1]$. Либо он сам меньше $a[i]$, либо нам нужен элемент $a[j] < a[i] \leq a[i - 1]$ левее, начинаем поиск с ближайшего меньшего к $a[i - 1]$, он как раз дальше по стеку. И так далее.

b) $(\min_{i \in [l..r]} a_i) \cdot (r - l + 1) \rightarrow \max$

Переберем позицию минимума. Пусть a_i будет минимумом на отрезке. Выгодно расширить его как можно сильнее влево и право, то есть до ближайших $< a_i$ слева и справа.

c) **Найти наибольший подпрямоугольник из нулей за $\mathcal{O}(n^2)$.**

Найдем для каждой клетки длину столбика $h[i][j]$ только из нулей, идущего вверх из этой клетки (0, если $a[i][j] == 1$, иначе $h[i-1][j] + 1$).

Прямоугольник с нижней границей i , левой l и правой r имеет верхнюю границу не более $\min h[i][l..r]$. Переберем нижнюю границу и минимальный столбик прямоугольника $h[i][j]$. Выгодно расширить по максимуму влево и вправо, то есть до ближайших слева и справа элементов $< h[i][j]$.

7. (*) Дек и стеки

Дек через два стека. Если один из стеков кончился, то просто разделим второй пополам на два стека.

После разделения стека размера $2k$ нам нужно минимум k операций `pop`, чтобы снова нужно было делать разделение. Если класть по монетке за каждый `pop` и `push`, то их хватит на разделение не опустевшего стека.

Еще можно ввести потенциал: модуль разности размеров стеков.

8. (*) Частичные суммы, координаты запросов до 10^{18}

Отсортируем концы l и $r + 1$ всех запросов по координате. Координаты концов запросов l и $r + 1$ заменим на их позиции в отсортированном порядке. Этот прием называется «сжатие координат». Операции `+=` и восстановление массива делаются так же.

Надо учесть, что между элементами i и $i + 1$ на самом деле целый отрезок $[x[i], x[i + 1])$, у всех его элементов значение равно $value[i]$.

Т.е. $pref[i + 1] = pref[i] + (x[i + 1] - x[i]) * value[i] \Rightarrow pref[i] = \sum_{z \in [0, x[i])} f[z]$.

Заметим, что можно отсортировать все запросы и `+=`, и суммы, тогда мы сразу будем знать сжатые координаты запросов суммы. А можно сжать координаты только для `+=`, тогда нужно будет искать бинпоиском позиции концов запросов суммы.

9. (*) **Стек и ближайшие**

```

1 for (int i = 0; i < n; ++i) {
2   while (!stack.empty() && a[i] < a[stack.top()]) right[stack.pop()] = i;
3   if (!stack.empty()) left[i] = stack.top();
4   stack.push(i);
5 }

```

Найдем ближайший меньший слева. Либо ближайший \leq уже меньше, либо нас интересует уже посчитанный ближайший $<$ к нему.

10. (*) **Правильные скобочные подстроки**а) **Один тип скобок.**

Считаем баланс скобок (число открывающих минус число закрывающих) на каждом префиксе. Подстрока – правильная, если в ней поровну открывающих и закрывающих скобок, и у нее нет префикса, на котором закрывающих больше, чем открывающих. То есть $(i..i+k]$ – хороший, если:

- $\text{balance}[i] == \text{balance}[i+k]$,
- $\forall j \in (i..i+k] \text{balance}[j] \geq \text{balance}[i]$. То есть, минимальный баланс на отрезке $(i..i+k]$ не меньше $\text{balance}[i]$ (точнее, равен).

Поддерживаем значения баланса на отрезке $(i..i+k]$ в очереди с минимумом и проверяем два указанных выше условия.

б) **m типов скобок.**

Запустим алгоритм проверки скобочной последовательности на правильность. Если для какой-то скобки нет парной, запомним это. Так же для каждой парной запомним позицию пары. Ответ на запрос $[l, r]$: проверить, что все скобки на $[l, r]$ имеют пару в $[l, r]$. То есть максимальный номер пары скобки из этого отрезка $\leq r$, минимальный $\geq l$.

11. (*) **Стек и 3D**

Блин как купаца

Домашнее задание

3.1. Обязательная часть

1. **(2) Сумма $\frac{k^2}{2^k}$**

Найдите сумму $\sum_{k=0}^{\infty} \frac{k^2}{2^k}$.

2. **(1) Максимальное среднее арифметическое**

Найти подотрезок с максимальным средним арифметическим элементов за $\mathcal{O}(n)$.

3. **(1) Оптимальный отрезок (easy)**

Найти самый короткий подотрезок, в котором хотя бы k различных чисел. $\mathcal{O}(n)$.

4. **(2) Оптимальный отрезок (medium)**

Найти подотрезок с максимальной суммой, длины от L до R , содержащий от A до B различных чисел, за $\mathcal{O}(n)$.

5. **(2) Максимизация числа**

Дано число, представленное n цифрами в d -ичной системе счисления без ведущих нулей. Из числа требуется вычеркнуть ровно $0 \leq k < n$ цифр так, чтобы полученное число из $n-k$ цифр было бы максимальным. Задачу требуется решить за линейное от n время при $d = 10$. **(+1)** за решение за $\mathcal{O}(n)$ для произвольного d .

6. **(2) ПСП-подстрока**

Найти максимальную по длине подстроку, являющуюся правильной скобочной последовательностью за $\mathcal{O}(n)$. Один тип скобок.

(+1) за решение, работающее для k типов скобок.

(+1) посчитать число подстрок, являющихся ПСП (k типов скобок).

3.2. Дополнительная часть

1. **(2) Приключения на окружности**

Дана окружность и число d . Даны $n \leq 10^6$ точек на окружности единичного радиуса. Каждая точка задана углом “радиус-вектора из центра окружности”. Для каждой из n точек найти ближайшую в порядке по часовой стрелке на евклидовом расстоянии **не менее** d .

2. **(3) Число прямоугольников**

Посчитать число подпрямоугольников, состоящих только из нулей, за $\mathcal{O}(n^2)$.

3. **(3) Избавление от амортизации**

Придумайте, как в очереди с минимумом через два стека победить амортизацию. То есть, придумайте аналогичную структуру данных с временем работы $\mathcal{O}(1)$ в худшем случае на каждую операцию.

Внимание: задача будет разобрано на лекции, коммитить можно **только** до лекции