Первый курс, осенний семестр
Практика по алгоритмам #2b

Линейные структуры данных

14 сентября

Собрано 22 сентября 2015 г. в 18:37

Содержание

1. Новые задачи

1. **Стандартные задачи на стек**
   a) Проверить правильность скобочной последовательности с несколькими типами скобок.
   b) Посчитать значение арифметического выражения, содержащего числа и операции `+`, `-`, `*`.
   c) Даны два выражения с целыми числами и операциями `+`, `-`, `*`. Суммарная длина не превосходит 10^6. Проверить, равны ли значения выражений.

2. **Частичные суммы**
   a) Много раз сделать `+=` на отрезке, в конце один раз вывести массив.
   b) Сперва много раз `+=` на отрезке, затем много раз запрос суммы на отрезке.
   c) (*) В каждой целой точке x числовой прямой есть $f[x]$, изначально равная нулю. Те же запросы, что и в предыдущем пункте. Координаты запросов целые от 0 до 10^{18}.

3. **Амортизация**
   Придумайте потенциал и докажите, что рассказанная на теории очередь с минимумом работает за амортизированное $\mathcal{O}(1)$.

4. **Оптимальный подотрезок**
   За линейное время найти подотрезок массива.
   a) С максимальной суммой (два решения).
   b) С максимальной суммой, длины от L до R.
   c) С максимальной суммой, содержащий не менее k различных чисел.
   Будем считать, что числа не превосходят по модулю 10^5.

5. **Задачи на стек**
   a) В массиве найти для каждого элемента ближайших меньших соседей слева и справа за $\mathcal{O}(n)$.
   b) Дана матрица из нулей и единиц. Найти наибольший по площади подпрямоугольник, состоящий только из нулей за $\mathcal{O}(n^2)$.
   c) (*) Обобщите предыдущий пункт до 3D.
   d) (*) Решить пункт `(1)` за один проход вперед по массиву (справа ищем меньше либо равный).

6. (*) **Сравнение выражений**
   Даны два выражения с целыми числами, **переменными**, операциями `+`, `-`, `*` и **скобками**. Суммарная длина не превосходит 10^6. Проверить, равны ли тождественно эти выражения (то есть совпадают при любых значениях переменных).

7. (*) **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. $\mathcal{O}(n)$.
   a) Скобки только одного типа.
   b) k типов скобок.

2. Разбор задач практики

1. • **Простые задачи на стек**
   a) **Правильность скобочной последовательности с несколькими типами скобок.**
   Открывающие скобки кладем в стек, встречая закрывающую, сравниваем ее тип с открывающей на вершине стека и делаем `pop`.
   b) **Значение арифметического выражения.**
   Два стека `numbers` и `operators`. Под словами «применение оператора» подразумевается последовательность действий: вынуть два числа y, x из `numbers` и верхнюю операцию $\circ$ из `operators`, положить $x \circ y$ в `numbers`.
   Идем по выражению слева направо. Встречая число x, кладем его в `numbers`. Встречая оператор $\circ$, пока приоритет `operators.top()` выше или равен, применяем `operators.top()`, в конце кладем $\circ$ в `operators`.
   Еще надо выполнить все операторы, оставшиеся в стеке после всего прохода по выражению. Окончательный результат будет на вершине `numbers`.
   c) **Проверить равенство двух длинных выражений.**
   Проверим, что их значения равны по модулю $P = 10^9 + 7$. Для большей верности можно проверить по модулю нескольких случайных простых чисел в пределах $2 \cdot 10^9$. Вероятность того, что выражения не равны, а остатки по модулю P равны: $\frac{1}{P}$.

2. • **Частичные суммы**
   a) **Много раз сделать `+=` на отрезке, в конце один раз вывести массив.**
   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.**
   Выполняем прибавление и восстанавливаем массив, как в предыдущем пункте. Считаем префиксные суммы. Ответ на запрос: `pref[r + 1] - pref[l]`.
   c) **Координаты запросов до 10^{18}.**
   Отсортируем концы l и $r + 1$ всех запросов по координате. Координаты концов запросов lи и $r + 1$ заменим на их позиции в отсортированном порядке. Этот прием называется “сжатие координат”. Операции `+=` и восстановление массива делаются так же. При подсчете префиксных сумм не забыть, что между элементами `i` и `i + 1` на самом деле целый отрезок `[x[i], x[i + 1])`, у всех элементов которого значение равно `value[i]`. То есть `pref[i + 1] = pref[i] + (x[i] - x[i - 1]) * value[i - 1]`, тогда `pref[i + 1] =` $\sum_{z \in [0, x[i])}$ `f[z]`.

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

3. • **Амортизация**
   Опишем очередь на массиве с поддержкой минимума.
   Будем хранить m_1 – позицию минимума на всей очереди, m_2 – позицию минимума среди элементов правее m_1 и так далее:
   $a[m_1] = \min a[L..R], a[m_{i+1}] = \min a[m_i + 1..R]$.

При этом $m_i < m_{i+1}, a[m_i] \leqslant a[m_{i+1}]$. Будем хранить m_i в деке `mins`. В коде `L` и `R` указывают на начало и конец очереди включительно.

```
get_min() {
  return mins.front();
}
pop() {
  if (L == mins.front()) mins.pop_front();
  return queue.pop();
}
push(x) {
  while (!mins.empty() && a[mins.back()] <= x) mins.pop_back();
  mins.push_back(R);
  queue.push(x);
}
```

Нам подойдёт потенциал $\varphi =$ `mins.size()`.
`pop()` и `get_min()` делают $\mathcal{O}(1)$ реальных операций и на $\mathcal{O}(1)$ меняют потенциал.
Пусть `push()` сделал k операций, тогда $\varphi_i - \varphi_{i-1} = -k$, итого амортизированное время $\mathcal{O}(1)$.

4. • **Оптимальный подотрезок.**
За линейное время найти подотрезок массива.

   a) **С максимальной суммой.**
   • *Способ #1.*
   ```
   l = 0, sum = 0;
   for (r = 0; r < n; r++)
     if ((sum += a[r]) < 0)
       l = r + 1, sum = 0;
   ```
   • *Способ #2.*
   ```
   l = 0, r = 0, sum = 0;
   while (r <= n)
     if (sum >= 0)
       sum += a[r++];
     else
       sum -= a[l++];
   ```
   Для каждого i ищем отрезок с максимальной суммой, оканчивающийся на i. Пусть для $i-1$ оптимальным является отрезок $[l_{i-1}, i-1]$. Тогда если для i оптимален отрезок $[l_i, i]$, то либо $l_i = l_{i-1}$, либо $l_i = i$. Предположим обратное. Тогда $\texttt{sum}([l_i, i-1]) + a[i] = \texttt{sum}([l_i, i]) > \texttt{sum}([l_{i-1}, i]) = \texttt{sum}([l_{i-1}, i-1]) + a[i] \Rightarrow \texttt{sum}([l_i, i-1]) > \texttt{sum}([l_{i-1}, i-1])$, противоречие.

   • *Способ #3.* Насчитаем частичные суммы `pref[]`. Сумма на отрезке `[l, r]` равна `pref[r + 1] - pref[l]`, при фиксированном `r` нужно минимизировать `pref[l]`. Перебираем `r` в порядке возрастания. Поддерживаем минимум `pref[l]` по всем пройденным `l` $\leqslant$ `r`.

   b) **С максимальной суммой, длины от L до R.**
   Теперь нас интересует минимум среди `pref[l]` на отрезке $[\texttt{r} - R, \texttt{r} - L]$. Поддерживаем очередь с минимумом, на каждом шаге один элемент оттуда уходит, и один приходит.

   c) **С максимальной суммой, содержащий не менее k различных чисел, числа небольшие.**

Теперь нас интересует минимум среди `pref[l]` на отрезке `[1, s]`, где `s` – последняя позиция, такая, что `[s + 1, r]` содержит хотя бы k различных чисел.
Заметим, что при увеличении `r` позиция `s` только растет, так что ее можно продолжать двигать со значения с прошлой итерации, тогда суммарно по всем итерациям ее поиск займет $\mathcal{O}(n)$.
Осталось научиться проверять, можно ли подвинуть `s` вперед. Для этого поддерживаем массив-счетчик каждого числа. При сдвиге `r` делаем `count[a[r]]++`, при сдвиге `s` делаем `count[a[s]]--`. Также поддерживаем число ненулевых ячеек в `count`, меняя его каждый раз, когда только что измененная ячейка начала или перестала быть нулем.
Заметим, что с использованием хэш-таблицы вместо массива можно решить эту задачу и для больших чисел.

5. • **Задачи на стек.**
   a) **Для каждого элемента ближайших меньших соседей слева и справа за $\mathcal{O}(n)$.**
   ```
   for (int i = 0; i < n; ++i) {
     while (!stack.empty() && a[i] <= a[stack.top()])
       right[stack.pop()] = i;
     if (!stack.empty())
       left[i] = stack.top();
     stack.push(i);
   }
   ```
   Инвариант: для каждого элемента стека перед ним в стеке идет ближайший к нему слева меньший.

   b) **Найти наибольший подпрямоугольник из нулей за $\mathcal{O}(n^2)$.**
   Найдем для каждой клетки длину отрезка $h[i][j]$ только из нулей, идущего вверх из этой клетки (0, если $a[i][j] == 1$, иначе $h[i-1][j] + 1$).
   Теперь при фиксированных нижней i, левой l и правой r максимально возможная верхняя граница равна $\min h[i][l..r]$. Переберем нижнюю строку и позицию минимума, для подсчета максимальной ширины при фиксированном минимуме $h[i][j]$ находим ближайший слева и справа элементы $< h[i][j]$.
   c) **(*) Обобщите предыдущий пункт до 3D.**
   ?

6. • **(*) Проверить равенство двух длинных выражений с переменными.**
Несколько раз подставим случайные числа в переменные и сравним значения выражений. Нам не повезет только, если мы попали в общий корень выражений, а их не очень много.

7. • **(*) Правильные скобочные подстроки**
   a) **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:
   - `balance[`i`] == balance[`$i+k$`]`,
   - $\forall j \in [i..i+k]$ `balance[`j`] >= balance[`i`]`. То есть, минимиальный баланс на отрезке

$[i..i+k]$ не меньше `balance[`i`]`.

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

b) • k **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 $\leqslant r$, минимальный $\geqslant l$.

3. Домашнее задание

1. Обязательная часть

1. **(1) Максимальное среднее арифметическое**
   Найти подотрезок с максимальным средним арифметическим элементов за O(n)
2. **(2) Оптимальный отрезок**
   Найти подотрезок с максимальной суммой, длины от L до R, содержащий от A до B различных чисел за $\mathcal{O}(n)$.
3. **(2) Максимизация числа**
   Дано число, представленное n цифрами в десятичной записи без ведущих нулей. Из числа требуется вычеркнуть ровно $0 \leqslant k < n$ цифр так, чтобы полученное число из $n-k$ цифр было бы максимальным. Задачу требуется решить за линейное от n время.
4. **(2) ПСП-подстрока**
   Найти максимальную по длине подстроку, являющуюся правильной скобочной последовательностью за $\mathcal{O}(n)$. Один тип скобок. **+1 балл** за решение, работающее для k типов скобок.

2. Дополнительная часть

1. **(3) Число прямоугольников**
   Посчитать число подпрямоугольников, состоящих только из нулей, за $\mathcal{O}(n^2)$.
2. **(3) Избавление от амортизации**
   Придумайте, как в очереди с минимумом победить амортизацию. То есть, придумайте аналогичную структуру данных с временем работы $\mathcal{O}(1)$ в худшем случае на каждую операцию.

4. Разбор домашнего задания

1. Обязательная часть

1. **(1) Максимальное среднее арифметическое**
   Среднее не превосходит максимальный элемент. Значит, задача сводится к поиску максимума в массиве.

2. **(2) Оптимальный отрезок**
   Это обобщение задачи, которую разбирали в классе: для каждой правой границы v, нужно найти такую границу u, что $sum[0, u)$ — минимально.
   Тогда $sum[u, v) = sum[0, v) - sum[0, u)$ — максимально.

   При этом, частичные суммы $sum[0, u)$ следует учитывать только для тех u, для которых $L \leqslant v - u \leqslant R$, и на $[u, v)$ от A до B различных чисел. Будем хранить их в очереди.

   Алгоритм: Для каждого v слева направо делаем следующее:
   1) кладём в конец очереди элементы до $v - L$ невключительно (границы подходят по длине).
   2) вынимаем из очереди элементы до $v - R$ (границы не подходят по длине).
   3) вынимаем из очереди элементы пока различных чисел больше, чем B.
   4) вынимаем из очереди элементы до максимального в очереди, или пока различных чисел не станет меньше, чем A.
   Вынутыми на шаге 4) числами обновляем ответ — величину максимальной суммы.

   Корректность: При передвижении u вправо (изъятия из очереди), количество различных чисел убывает, а длина интервала $[u, v)$ уменьшается.

   Поэтому рассматривать элементы, выкинутые на шагах 2 и 3, а также всех левее них (выкинутые ранее, при других v) бессмысленно.

   Рассматривать элементы правее максимального (или того, на котором станет слишком мало различных элементов) тоже бессмысленно.

   Поэтому для каждой правой границы v не будут рассмотрены только «бесполезные» левые границы.

3. **(2) Максимизация числа**
   Алгоритм:
   1) Бежим слева направо, если предыдущая цифра меньше, чем текущая удаляем её. Для этого все уже рассмотренные цифры кладём в стек.

   2) Если удаления кончились (k штук), заканчиваем алгоритм.

   3) Если прошли по всему числу, а удаления остались — удаляем несколько цифр с конца.

   Корректность: Очевидно, что порядок удалений не важен, их все можно было делать слева направо. Обозначим удалённые слева направо цифры в оптимальном ответе $d_1, d_2, \ldots, d_k$.

   Очевидно, что последовательность цифр до цифры d_1 убывает. Иначе можно было бы увеличить более старший разряд и получить ответ лучше.

   После удаления числа d_1 последовательность цифр до цифры d_2 будет убывать, и так далее.

Если прошли по всему числу, то последовательность цифр всего числа убывающая, надо удалить самые маленькие цифры в младших разрядах.

4. **(2) ПСП-подстрока**
Алгоритм: Давайте запустим обычный алгоритм проверки СП на правильность, но в случае, если нашли отрицательный баланс (или непарную закрывающую скобку), мы пометим её как плохую и, сделав вид, что её не было, продолжим алгоритм.

Скобки, которые останутся на стеке, тоже пометим как плохие.

После прохода по строке для каждого символа будет написано плохой он или хороший. Ответом будет являться непомеченный отрезок максимальной длины.

Корректность:
1) Предположим, что подстрока s является ПСП, и она не делится на две ПСП.
Наш алгоритм положит в стек первую её скобку, потом корректно отработает, очистив стек от её внутренностей, а потом получит парную закрывающуюся скобку.

Аналогично, если $s = ab$, где a, b являются ПСП.

Таким образом, алгоритм не пометит ни одного символа внутри s и приведёт стек в исходное состояние.

2) Допустим, некоторая закрывающая скобка помечена как плохая. Тогда она не входит ни в одну подстроку, являющуюся ПСП.

Действительно, если существует $[a, b]$ — ПСП: $s \in [a, b]$, то баланс $[a, s] \geqslant 0$. Но баланс $[s, s] = -1$, значит между a и s есть c, такой что баланс $[c, s] = 0$ (баланс меняется на ± 1). Значит, алгоритм не пометил бы символ s.

2. Дополнительная часть

1. **(3) Число прямоугольников**
Как на практике, найдем высоты столбцов из нулей $h[i][j]$ и ближайший минимальный слева к каждому.
Считаем $\texttt{ans}[i][j]$ – число прямоугольников, у которых нижний правый угол на позиции (i, j).
Рассмотрим ближайшее меньшее к $h[i][j]$ слева число $h[i][k]$.
Прямоугольники высотой с $h[i][k]+1$ по $h[i][j]$ могут иметь любую левую границу из $[k+1, i]$ и только оттуда, итого их $(i-k)(h[i][j]-h[i][k])$.
Прямоугольников с меньшей высотой будет $\texttt{ans}[i][k]$: любому прямоугольнику, у которого правый нижний угол (i, k), можно взаимно однозначно сопоставить прямоугольник с правым нижним углом (i, j) высоты $\leqslant h[i][k]$, просто продлив его вправо до j.
Итого $\texttt{ans}[i][j] = (i-k)(h[i][j]-h[i][k]) + \texttt{ans}[i][k]$.

2. **(3) Избавление от амортизации**
Смотрите лекцию.