Алгоритмы. Домашнее задание №2

Горбунов Егор Алексеевич

24 сентября 2015 г.

Задача №1 (Про вычёркивание k цифр из числа длины n за $\mathcal{O}(n)$)

$num[0..n-1]$ — входное число длины n. Рассмотрим $n[0..k]$ — первые $k+1$ цифру этого числа. Пусть i_m — это индекс первого вхождения $max(n[0..k])$ в $n[0..k]$. Нам выгодно жадно вычеркнуть все цифры до i_m-ой, благо их не больше k, т.к. $0 \le i_m \le k$. Выгодно нам это сделать потому, что положим, первая цифра, которую мы оставим в числе — это цифра с индексом $j < i_m$, но по определению i_m: $num[j] < num[i_m]$, но числа с большей цифрой в старшем разряде всегда больше чисел **той же длины** с меньшей цифрой в старшем разряде. Первой оставить цифру с индексом $> k$ мы не можем, т.к. придётся вычеркнуть более k цифр, что не позволительно по условию. Значит нам действительно выгодней всего вычеркнуть все цифры до i_m-ой! Таким образом мы вычеркнем m цифр и нам останется вычеркнуть $k-m$ в числе длины $n-m-1$, что решается аналогично =).

Вход: num — число длины n
k — сколько цифр нужно вычеркнуть, $k <= n$

```
q ← uber_queue()
from ← 0
to ← k
start :
for i from from to to do
    q.push(num[i])
max ← q.max()
while to − from ≥ 0 do
    from ← from + 1
    x ← q.pop()
    if x! = max then
        to_cross.add(x)
    else
        to ← to + 1
        break;
if to_cross.size() < k then
    goto start
```

Приведённый алгоритм будет работать за $\mathcal{O}(n)$, если очередь q реализовать на 2 стеках с поддержкой опреации максимума за амортизированную константу. Ни один элемент массива num не может побывать дважды в очереди q. num обходится последовательно слева направо.

Задача №2 (Совершенное паросочетание в дереве за $\mathcal{O}(n)$)

Дано дерево T. Известно, что в любом дереве найдётся хотя бы 2 вершины степени 1.

Вход: T — дерево

Выход: PM — множество рёбер, входящий в совершенное паросочетание (если его нет, то $\varnothing$)

```
 1: PM ← ∅
 2: S ← ∅
 3: mcnt ← 0
 4: for v ∈ V(T) do
 5:     if d(v) = 1 then
 6:         S.add(v)
 7: while S ≠ ∅ do
 8:     v ← S.pop()
 9:     u ← Γ(v).first()
10:     T.delete_edge((v, u))
11:     PM.add((v, u))
12:     for v ∈ Γ(u) do
13:         T.delete_edge((v, u))
14:         if d(v) = 1 then
15:             S.add(v)
16:     mcnt ← mcnt + 2
17: if mcnt < |V(T)| then
18:     PM ← ∅
19: return PM
```

$\Gamma(v)$ — список вершин, смежных с v.

Если $d(v) = 1$, то из v выходит единственное ребро $e \in E(T)$ и оно **обязано входить в совершенное паросочетание**, т.к. это единственная возможность покрыть v. После выполнения строки, в силу свойст дерева, 6 в S будет как минимум 2 вершины. В строке 8 происходит удаление висячей вершины v из s, а на следующей строке мы удаляем единственное ребро (v, u), выходящее из v. Теперь мы можем выкинуть их T вершины v и u т.к. они уже покрыты. Важно: после выкидывания этих вершин $T \backslash \{v, u\}$ останется деревом, а значит в нём так же будет как минимум 2 висячие вершины, некоторые из них уже могут лежать в S, а некоторые могли появиться после удаления вершины u, поэтому мы и проделываем поиск таких вершин в строках 12—15. S — может быть стэком, т.е. операции с ним реализуются за $\mathcal{O}(1)$. PM так же может быть стэком, т.е. операции с ним тоже за $\mathcal{O}(1)$. Удаление ребра из графа может быть реализовано за $\mathcal{O}(1)$ (помечать рёбра как удалённые). На каждой итерации цикла $While$ мы покрываем 2 вершинки, суммарное число покрытых вершин хранится в переменной $mcnt$.

Если к моменту выхода это число не равно $|V(T)|$, то совершенного паросочетания для T не существует.
Таким образом на каждой итерации цикла $While$ мы делаем $\mathcal{O}(d(u))$ операций, тело цикла $While$ отрабатывает столько раз, сколько вершин в T, а таким образом весь алгоритм отрабатывает за $\mathcal{O}(2|E(T)|) = \mathcal{O}(n)$ (по лемме о рукопожатиях), где n — число вершин (в дереве $|E(T)| = n - 1$). ■

Задача №3 (-)

Задача №4 (Запросы о сумме на отрезке за $\mathcal{O}(1)$)

Делаем следующий предподсчёт, очевидно, работающий за линейное время от размера входного массива:

Алгоритм 1 Предподсчёт

Вход: A — массив из n целых чисел
Выход: P — массив из $n+1$ целых чисел, что $P[0] = 0,\ P[k] = \sum_{i=0}^{k-1} A[i], k > 0$

$P \leftarrow fill(array[0..n], 0)$
for i **from** 0 **to** $n-1$ **do**
    $P[i+1] \leftarrow P[i] + A[i]$

Отвечаем на запрос за $\mathcal{O}(1)$ исходя из очевидного равенства: $\sum_{i=l}^{r} A[i] = \sum_{i=0}^{r} A[i] - \sum_{i=0}^{l-1} A[i]$

Алгоритм 2 Ответ на запрос

Вход: l, r — натуральные, такие, что $0 \le l \le r \le n-1$
Выход: целое число: $s = \sum_{i=l}^{r} A[i]$

return $P[r+1] - P[l]$

Задача №5 (Разложение числа в сумму за $\mathcal{O}(n)$)

Вход: $s, A[0..n-1]$ — целое положительное число и массив из целых положительных чисел
Выход: l, r такие, что $\sum_{i=l}^{r} A[i] = s$; или $l = r = -1$, если сумму составить невозможно

l ← 0
r ← 0
sum ← 0
while $r < n$ AND $l < n$ AND $s! = sum$ **do**
    if $sum < s$ **then**
        $sum \leftarrow sum + A[r]$
        $r \leftarrow r + 1;$
    else if $sum > s$ **then**
        $sum \leftarrow sum - A[l]$

$\quad l \leftarrow l + 1$

$r \leftarrow r + 1$

Этот алгоритм работает за $\mathcal{O}(n)$ т.к. $l \le r$ на протяжении всей его работы, т.к. перед главным циклом это верно, а далее l увеличивается лишь тогда, когда $sum > s$, а это верно лишь тогда, когда $l < r$, т.к. sum всегда хранит сумму чисел массива с индексами в полуинтервале $[l, r)$.
Пускай для некоторого числа s наименьший возможный по левой границе ответ на задачу — полуинтервал $[a, b)$, алгоритм подбирает ответ, начиная с $[0, 0)$. На каждой итерации двигается **на 1** либо левая граница, либо правая. Тогда рано или поздно либо l дойдёт до a, либо r дойдёт до b. Пусть l дошёл до a раньше, чем r до b, тогда $sum < s$, т.к. $l \le r \le b$ и числа в массиве положительны. Значит, исходя из алгоритма, далее мы будем двигать правую границу до тех пор, пока r не станет равным b и условие внешнего цикла сломается (накоец то)! Пускай r раньше дошёл до b, чем l дошёл до a, т.е. $l < a < b = r$. Но тогда $sum > s$ в силу натуральности чисел массива. А значит, анологично, мы будем двигать границу l вперёд до тех пор, пока l не станет равным a и условие цикла снова сломается. Значит алгоритм корректен. ∎

Задача №6 (Максимальная сумма в массиве над $\mathcal{Z}$ за $\mathcal{O}(n)$)

Вход: A — массив из n целых чисел
Выход: l, r — такие индексы массива, что: $\sum_{i=l}^{r} A[i]$ максимальна

$P \leftarrow fill(array[0..n], 0)$
for i **from** 0 **to** $n - 1$ **do**
$\quad P[i+1] \leftarrow P[i] + A[i]$
$(l, r) \leftarrow min\_max\_ind(P)$
$(l', r') \leftarrow max\_min\_ind(P)$
if $P[r] - P[l] > P[r'] - P[l']$ **then**
$\quad$ **return** $(l + 1, r)$
else
$\quad$ **return** $(l' + 1, r')$

$min\_max\_ind(P)$ — возвращает пару (l, r), что l — индекс минимального элемента в массиве P, а r — индекс максимального элемента в P, причём $r > l$ (r ищется после l).
$max\_min\_ind(P)$ — возвращает пару (l, r), что r — индекс максимального элемента в массиве P, а l — индекс минимального элемента в P, причём $l < r$ (l ищется после r).

Функции $min\_max\_ind(P)$ и $max\_min\_ind(P)$ очевидно могут быть реализованы за $\mathcal{O}(n)$ опреаций. Подсчёт сумм на префиксах так же занимает $\mathcal{O}(n)$. В конце мы получаем такую пару (l, r), что $\sum_{i=0}^{l-1} A[i]$ минимальна, а $\sum_{i=0}^{r} A[i]$ максимальна, причём из всех возможных $l \le r$. Но это значит, что $\sum_{i=l}^{r} A[i]$ максимальна! ∎