به نام خدا

آزمون عملی دوم

ساختمان‌داده‌ها و الگوریتم‌ها

۹ خرداد ۱۳۹۸

پاسخ‌نامه

دکترشریفیJR و حل معادله (دسته‌بندی: جست‌وجوی دودویی)

محدودیت زمان: ۱ ثانیه

محدودیت حافظه: ۱۰۰ مگابایت

اگر عبارت طرف چپ تساوی را به شکل یک تابع از x در نظر بگیریم؛ می‌توان دریافت که با توجه به مثبت بودن ضرایب و نیز اعداد l و r، این تابع اکیداً صعودیست پس برای این‌که مقدارش برابر با b بشود می‌توان از جست‌وجوی دودویی استفاده کرد. هزینه‌ی زمانی الگوریتم از $O(n \log(l-r))$ است. از آن‌جا که دقت ۶ رقم اعشار مدنظر است، به محض این‌که اختلاف دو سر بازه‌ی جست‌وجوی دودویی از 10^{-6} کمتر شد جواب را گزارش می‌کنیم.

کد این سوال در ادامه آمده است:

```
n = int(input())
a = list(map(float, input().split()))
b = float(input())
l, r = map(float, input().split())


def pol(x):
    res = 0
    c = 1
    for i in range(n+1):
        res += c * a[i]
        c *= x
    return res


while r - l > 0.0000001:
    m = (r + l) / 2
    if pol(m) > b:
        r=m
    else:
        l=m

print(l)
```

کیانوش در تیمارستان! (دسته‌بندی: درهم‌سازی)

محدودیت زمان: ۲/۵ ثانیه

محدودیت حافظه: ۵۰۰ مگابایت

برای حل زیرمسئله می‌توان دوبه‌دو همه‌ی رشته‌ها را با هم مقایسه کرد که این الگوریتم از $O(n^2k)$ است که k برابر با طول رشته‌هاست.

برای کسب نمره‌ی کامل باید به این نکته توجه کرد که اگر به ازای اندیسی مثل j، با حذف این اندیس دو رشته از رشته‌های ورودی با هم برابر شوند، بنابراین با توجه به متمایز بودن تمام رشته‌ها، این دو رشته فقط در اندیس j با هم تفاوت داشته‌اند. پس باید ابتدا تمام رشته‌ها را hash کرد؛ تابع درهم‌سازی ما می‌تواند به این صورت باشد:

$$f(s) = \sum_{i=0}^{k-1} s_i p^i$$

که s_j کد ascii حرف jام رشته (اندیس‌های رشته از صفر شروع می‌شوند) و p یک عدد اول مناسب است. حالا به ازای هر اندیس j باید hash تمام رشته‌ها را در صورتی که اندیس j از آن‌ها حذف شده باشد حساب کنیم؛ که طبق تابع بالا برابر می‌شود با:

$$g(s,j) = \sum_{i=0}^{j-1} s_i p^i + \sum_{i=j+1}^{k-1} s_i p^{i-1}$$

دقت کنید که $g(s,j)$ صرفاً برابر با $f(s) - s_j p^j$ نیست و با توجه به حذف یک حرف از رشته، توان p در اندیس‌های بعد از j باید یک واحد کم شود. بنابراین برای هر رشته دو آرایه‌ی هش جزئی (Partial Hash) را به ترتیب زیر می‌گیریم:

$$PartialHash1(s,j) = \sum_{i=0}^{j-1} s_i p^i \text{ و } PartialHash2(s,j) = \sum_{i=j+1}^{k-1} s_i p^{i-1}$$

که این دو آرایه را باید به ازای هر رشته به صورت پیش‌پردازش ساخت. هزینه‌ی این قسمت از الگوریتم از $O(nk)$ است.
کد این بخش به صورت زیر است:

```
codes = []
while True:
    x = input()
    if x != '$':
        codes.append(x)
    else:
        break

p = 863
n = len(codes)
mod = 10 ** 9 + 7
k = len(codes[0])

p_list = [1]
for i in range(k):
    p_list.append(p_list[-1] * p % mod)

partial_hash1 = [[0 for j in range(k)] for i in range(n)]
partial_hash2 = [[0 for j in range(k)] for i in range(n)]
```

```
for i in range(n):
    s = codes[i]
    for j in range(k):
        if j > 0:
            partial_hash1[i][j] = partial_hash1[i][j-1]
        partial_hash1[i][j] += ord(s[j]) * p_list[j]
        partial_hash1[i][j] %= mod
    for j in range(k-1, 0, -1):
        if j < k - 1:
            partial_hash2[i][j] = partial_hash2[i][j+1]
        partial_hash2[i][j] += ord(s[j]) * p_list[j-1]
        partial_hash2[i][j] %= mod
```

حال به ازای هر اندیس j باید روی تمام رشته‌ها حرکت کنیم و با استفاده از دو هش جزئی که ساختیم، هش رشته‌ها را بعد از حذف اندیس jام بیابیم و آن‌ها را در یک آرایه کوچک‌تر نگه داریم تا اگر دو هش با هم برابر شدند، تقلب را کشف کنیم!
کد این بخش به صورت زیر است:

```
cheat = [False for i in range(n)]

for j in range(k):
    hash_table = [None] * 611593
    for i in range(n):
        s = codes[i]
        hash_value = (partial_hash1[i][j-1] if j else 0) + (partial_hash2[i][j+1] if
j < k - 1 else 0)
        hash_value = hash_value % mod % len(hash_table)
        if hash_table[hash_value] == None:
            hash_table[hash_value] = i
        else:
            cheat[i] = True
            cheat[hash_table[hash_value]] = True

answer = [codes[i] for i in range(n) if cheat[i]]
answer.sort()
if len(answer):
    for s in answer:
        print(s)
else:
    print('all ok')
```

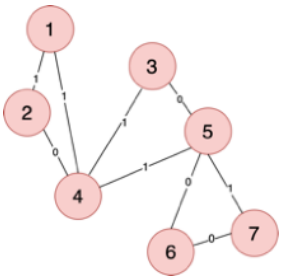
البته در آزمون با توجه به این که مرتب‌سازی سریع جزو سرفصل‌های آزمون به حساب می‌آمد، مجاز به استفاده از تابع sort نبودید و باید یک تابع مرتب‌سازی برای خود پیاده‌سازی می‌کردید. با توجه به هزینه‌ی مرتب‌سازی، هزینه‌ی زمانی این الگوریتم مجموعاً از $O(nk \log n)$ است.

عکساش همه هشتگ لاکچری تهرانه! (دسته‌بندی: الگوریتم‌های گراف)

محدودیت زمان: ۱ ثانیه

محدودیت حافظه: ۲۵۶ مگابایت

برای حل زیرمسئله می‌توان از الگوریتم دایکسترا با هزینه‌ی $O(n^2)$ بهره برد؛ اما برای کسب نمره‌ی کامل باید ایده‌ی الگوریتم BFS را اندکی تغییر داد.



با بررسی روی ورودی نمونه (شکل مقابل)، می‌توان دریافت که الگوریتم BFS پاسخ درست را نمی‌دهد؛ زیرا برای مثال وقتی راس شماره‌ی ۵ وارد صف شده، ممکن است راس ۷ را زودتر از راس ۶ وارد صف کند و بنابراین زودتر از آن از صف خارج کند درحالی که مسیر ۵،۶،۷ وزنی کمتر از مسیر ۵،۷ دارد.

برای حل این مشکل، باید به این نکته توجه داشت که وقتی دو راس u و v با یال به وزن صفر به هم متصل می‌شوند، طول کوتاه‌ترین مسیر از u به هر راس دیگر گراف با طول کوتاه‌ترین مسیر از v به آن راس برابر است؛ در واقع برای مسئله‌ی کوتاه‌ترین مسیر، عملاً این دو راس را می‌توان یک راس درنظر گرفت. پس می‌توان با تغییر اندکی در الگوریتم BFS سوال را حل کرد؛ به این ترتیب که به جای صف، یک صف دوطرفه (deque) می‌گیریم و هرگاه یال با وزن صفر دیدیم، سر دیگر یال را به ابتدای صف اضافه می‌کنیم تا زودتر از بقیه راس‌های صف دیده شود؛ زیرا عملاً باید دو سرِ یال صفر را یک راس در نظر بگیریم. هزینه‌ی زمانی این الگوریتم هم مثل BFS از $O(n+m)$ است. کد این سوال در ادامه آمده است:

```python
from collections import deque

n, m = map(int, input().split())
adlist = [[] for i in range(n)]

for i in range(m):
    u, v, w = map(int, input().split())
    u, v = u-1, v-1
    adlist[u].append([v, w])
    adlist[v].append([u, w])

dist = [10**9 for i in range(n)]
vis = [0 for i in range(n)]
dist[0] = 0

dq = deque()
dq.append(0)

while len(dq):
    x = dq.popleft()
    if vis[x]:
        continue
    vis[x] = True
    for y, w in adlist[x]:
        if not vis[y] and dist[y] > dist[x] + w:
            dist[y] = dist[x] + w
            if w:
                dq.append(y)
            else:
                dq.appendleft(y)

print(dist[n-1])
```