به نام خدا — خلاصه‌ی کلاس حل تمرین — بررسی fft

در این مسأله هدف محاسبه‌ی $F(x)$ به ازای مقادیر $\omega_0, \ldots, \omega_{n-2}, \omega_{n-1}$ که در آن $\omega_n = 1$ و $\omega_{\frac{n}{2}} = -1$ است.

$\boxed{\omega_n \text{ یعنی } \omega^n \text{ است}}$

فرض می‌کنیم که برای ω ، $F(\omega)$ را داریم و می‌خواهیم $F(\omega_2)$ را پیدا کنیم. داریم:

a_i در اینجا ضرایب چندجمله‌ای هستند

ω: $\quad a_0\omega_0 \quad a_1\omega_1 \quad a_2\omega_2 \quad \ldots \quad a_{n-1}\omega_{n-1}$

ω_2: $\quad a_0\omega_0 \quad a_1\omega_2 \quad a_2\omega_4 \quad \ldots \quad a_{n-1}\omega_{2n-2}$

اگر $2n$ را n بنامیم آنگاه سری دوم به صورت زیر در می‌آید:

$$a_0\omega_0 \quad a_1\omega_2 \quad a_2\omega_4 \quad \ldots \quad a_{\frac{n}{2}-1}\omega_{n-2} \quad a_{\frac{n}{2}}\omega_0 \quad \ldots \quad a_{n-1}\omega_{n-2}$$

در حالت کلی $F(x)$ را می‌توان به صورت مجموع زیر نوشت:

$$F(x) = \sum_{i=0}^{\frac{n}{2}-1} a_{2i}x^{2i} + \sum_{i=0}^{\frac{n}{2}-1} a_{2i+1}x^{2i+1} =$$

$$\underbrace{\sum_{i=0}^{\frac{n}{2}-1} a_{2i}x^{2i}}_{\text{بخش زوج } F(x^2)} + x\underbrace{\sum_{i=0}^{\frac{n}{2}-1} a_{2i+1}x^{2i}}_{\text{بخش دارای ضرایب فرد از } F(x^2)}$$

$$= F_e(x^2) + xF_o(x^2)$$

از طرفی $\underbrace{\omega_{j+\frac{n}{2}}}_{x} = \underbrace{-\omega_j}_{-x}$ یعنی برای محاسبه‌ی اندیس‌های بزرگتر از $\frac{n}{2}$ نیز می‌توان $-\omega_j$ را در عبارت بالا جایگذاری کرد.

$$\boxed{F(-x) = F_e(x^2) - xF_o(x^2)}$$

در نتیجه شبه کد این الگوریتم به صورت زیر در می آید :

```
recursive FFT(A, n)                                   ⟶ T(n)
    if (n = 1)
        return A
    else
                                  ماتریس ضرایب زوج
        Y_e ← recursive FFT(A_E, n/2)                 ← T(n/2)
        Y_o ← recursive FFT(A_o, n/2)                 ← T(n/2)
                                  ماتریس ضرایب فرد
        for (i = 0, i < n/2, i++) {
            Y_i       ← Y_e[i] + ω * Y_o[i]    }
            Y_{i+n/2} ← Y_e[i] − ω * Y_o[i]    }      ← O(n)
            ω = ω * ω_i^n
        }
        return Y
}
```

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

$$\Longrightarrow T(n) = \Theta(n\log n)$$

fibunacci heap

هیپ فیبوناچی

تفاوت های هیپ فیبوناچی و هیپ معمولی

function	min_heap	fibunacci heap
add	O(log n)	O(1)
delete	O(log n)	O(log n)
get min	O(1)	O(1)
delete min	O(log n)	O(1)

کارهایی از این هیپ که مرتبه‌ای متفاوت با هیپ معمولی دارند،

به صورت سرشکن محاسبه شده‌اند

سرشکن Amorthized :

فرض کنید داده‌ی ساختاری داریم که در آن یک عمل به صورت زنجیره‌ای از اعمال کوچکتر تعریف شده و هر یک از این اعمال از مرتبه‌ی $O(k_i)$ اجرا می‌شود آنگاه $O(F)$ برای این عمل زنجیره‌ای به صورت زیر تعریف می‌شود که به آن مرتبه‌ی سرشکن می‌گویند:

$$O(F) = \frac{\sum O(k_i)}{i}$$

خلاصه‌ی کلاس حل تمرین   درس آنالیز الگوریتم

الگوریتم Prim را برای یافتن MST شرح می‌دهیم:

ابتدا همه‌ی رأس‌ها را در یک آرایه قرار می‌دهیم و برای هر کدام، یک متغیر بولی اختصاص می‌دهیم. True به معنی قرار گرفتن رأس مورد نظر در درخت ساخته شده تاکنون و false یعنی اینکه این رأس در درخت قرار ندارد. R را مجموعه‌ی یال‌های انتخاب نشده‌ی مجاور با رأس‌های درخت در نظر می‌گیریم.

V	F		..	T		F

اولین رأس را به طور تصادفی انتخاب می‌کنیم (s')

E بار تکرار می‌شود

1) add all edges from s' to R

2) while (R not empty) {

3) e ← extract min from R ; → با استفاده از heap از $O(\log(E))$ انجام می‌شود

4) if(V[e.f] != V[e.s]) { → $O(V)$

رأس انتهای یال   رأس ابتدایی

5) $O(1)$ ← assume that e.s is out of V_T

6) $O(1)$ ← V[e.s] ← T

7) $E\log(E)$ ← add all edges from e.s to R

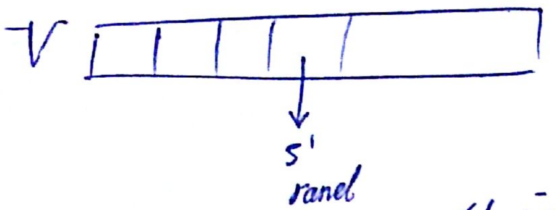
8) $O(1)$ ← add e to tree

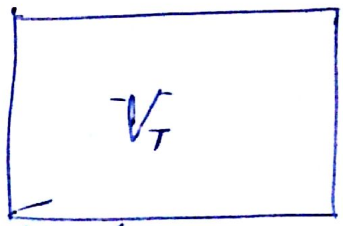
در نتیجه کل این الگوریتم از مرتبه‌ی $O(V+E\log E)$ است.

الگوریتم Prim را با استفاده از این هرم جدید می‌توان بهبود بخشید :

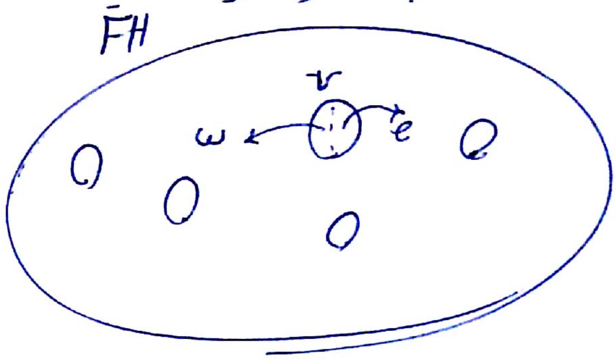
اگر این بار به جای این که یال‌ها را در هرم قرار دهیم، به ازای هر رأس دو field ، e و w را هم تعریف کرده و

هر رأس V_i را در FH قرار دهیم و بر اساس w ، min را پیدا کنیم ، داریم :





به ازای هر رأس بیرونی، کمترین یال به داخل را در اختیار داریم



در ابتدا وزن یال‌های از s' به سایر رأس‌ها را در داخل FH می‌گذاریم

1) Init weights in FH with edges from s to any of them

2) while (R not empty) $\longrightarrow O(v)$

3) e ← extract min $\longrightarrow O(\log v)$ — FH $O(1)$

4) add e to tree $\longrightarrow O(1)$ — overall $O(v \log v)$

5) delete V_e from FH — $O(\log v)$

for each edge from V_e to FH vertices : e' $\longrightarrow O(E)$
if needed Decrease key $V_{e'}$ $\xrightarrow{FH} O(1)$ — or $O(E)$

کل این الگوریتم در مرتبه‌ی $O(v \cdot \log v + E)$ اجرا می‌شود که بهبودی نسبت به حالت قبل است