



دانشگاه علم و صنعت ایران

دانشکده مهندسی کامپیوتر

جزوه درس

ساختمان‌های داده

استاد درس: سید صالح اعتمادی

پاییز ۱۳۹۸

جلسه ۲۲

جدول هش

محتبی نافذ - ۱۳۹۸/۱۱/۹

جزوه جلسه ۲۲ ام مورخ ۱۳۹۸/۱۱/۹ درس ساختمان‌های داده تهیه شده توسط محتبی نافذ. در جهت مستند کردن مطالب درس ساختمان های داده، بر آن شدیم که از دانشجویان جهت مکتوب کردن مطالب کمک بگیریم. هر دانشجو می‌تواند برای مکتوب کردن یک جلسه داوطلب شده و با توجه به کیفیت جزوه از لحاظ کامل بودن مطالب، کیفیت نوشتار و استفاده از اشکال و منابع کمک آموزشی، حداکثر یک نمره مثبت از بیست نمره دریافت کند. خواهش مند است نام و نام خانوادگی خود، عنوان درس، شماره و تاریخ جلسه در ابتدای این فایل را با دقت پر کنید.

۱.۲۲ مقدمه ای بر تابع هش

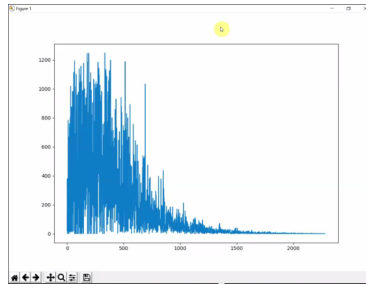
با توجه به مطالب جلسه گذشته این جلسه با ذکر چند نمونه تابع هش و مقایسه ی آن ها باهم مطالب را ادامه می‌دهیم
فرض کنید یک مجموعه از ۶۰۰ هزار رشته در اختیار داریم و می‌خواهیم یک تابع هش برای هش کردن این رشته ها پیاده سازی کنیم تا بتوانیم در یک جدول هش آن ها را دلخواه نگهداری کنیم
در اینجا چند تابع هش را بیان و نمودار مقدار هش (hash value) یکتا بر اساس تعداد collision هر hash value را رسم و توضیح می‌دهیم.

- یک ایده ی ساده جمع اعداد اسکی کاراکترهای موجود در رشته است

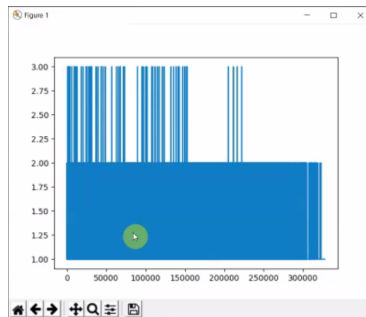
همانطور که می بینیم ۶۰۰ هزار رشته به حدود ۲۴۰۰ hash value یکتا متناظر شده که برای مثال در hash value ۵۰۰ حدود ۱۱۰۰ collision رخ داده

- یک ایده ی دیگر هش کردن به صورت hash value های رندوم میباشد

همانطور که می بینیم این هش تعداد collision به شدت کم و پخش شده ای دارد که این عالی است اما یک ایراد بزرگ دارد این مزیت را از بین میبرد
زمانی که دو بار یک رشته را به آن می‌دهیم دو hash value متفاوت به ما میدهد پس در جدول هش ما هر مقداری را Insert کنیم نمیتوانیم دوباره آن را بازیابی کنیم



شکل ۱.۲۲: نمودار هش جمع کد اسکی



شکل ۲.۲۲: نمودار هش کاملاً رندوم وار

آیا میتوانید یک تابع هش خوب برای رشته مثال بزنید؟

حال فرض کنید تعداد زیادی شماره تلفن داده شده و میخواهیم یک تابع هش برای متناظر نمودن آن ها به خانه های جدول هش پیاده سازی کنیم.

- میتوان سه رقم اول هر شماره را به عنوان مقدار هش در نظر گرفت.

اما در این صورت همه ی کسانی که در یک منطقه زندگی میکنند ده یک خانه متناظر میشوند

- یک ایده ی دیگر هش کردن به صورت hash value های رندوم میباشد

همانطور که اشاره کردیم در این صورت مقادیر قابل بازیابی نیستند.

آیا میتوانید یک تابع هش خوب برای اعداد مثال بزنید؟

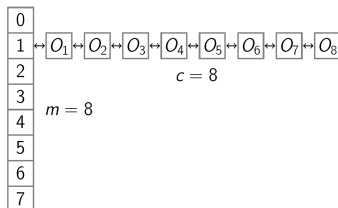
۲.۲۲ ویژگی های تابع هش خوب

- یک مقدار را همیشه به یک hash value متناظر کند

- مقدار هش سریع محاسبه شود
 - در تعداد مقادیر زیاد مقادیر را به صورت توزیع شده ای به هش های گوناگون متناظر کند
 - collision آن کم باشد
- نکته : باید توجه داشت که هر تابع هشی با توجه به یک مجموعه دیتا ممکن است خوب باشد یا بد اما ما باید طوری تابع هش پیاده سازی کنیم که عموماً خوب باشد

۳.۲۲ denied of service attack (DOS)

DOS یک نوع حمله سایبری برای از کار انداختن یا کند کردن پاسخگویی یک سرور است به طوری که کاربران به صفحه ی وب یا اینترنت یا ایمیل و ... دسترسی نداشته باشند. اساس کار این حمله ایجاد collision زیاد است. هر تابع هشی را میتوان ورودی هایی را داد که در نهایت همه ی آن ها به یک مقدار هش شوند شاید این کار سخت باشد اما امکان پذیر است



شکل ۳.۲۲: نوع collision برای dos attack

در پارس کردن فایل ها در سرور ها بسیار از فایل json استفاده میشود این فایل نوعی جدول هش است حمله کننده اگر بتواند تعداد زیادی collision پیدا کند میتواند در کار سرور اختلال ایجاد نماید سرور با جستجو کردن در این LinkList دچار مشکل خواهد شد. شرط این کار این است که حمله کننده تعداد زیاد رشته از قبل به سرور بدهد و خروجی هش را ببیند و بتواند تعدادی رشته ی collision دار پیدا نماید در سال ۲۰۱۱ شرکت مایکروسافت برای حل این مشکل امنیتی از UseRandomizedStringHashAlgorithm برای غیر قابل پیش بینی کردن collision ها در پیاده سازی جدول هش استفاده کرد. که در ادامه توضیح خواهیم داد.

۴.۲۲ Universal Family

ایده : یک مجموعه از توابع هش داریم و هر دفعه به طور رندوم یکی از این توابع را انتخاب میکنیم فرض کنید یک مجموعه ی U داریم که شامل $0, 1, 2, \dots, m-1$ است حال h را یک تابع هش در نظر بگیرید که هر ورودی را به عددی عضو U متناظر میکند. حال فرض کنید H یک مجموعه از این توابع هش h است یعنی:

$$\{ \{ 1-m, \dots, 2, 1, 0 \} \leftarrow U : h \} = H$$

H , Universal family است اگر برای دو ورودی مختلف احتمال collision کم تر از $1/m$ است

$$P[h(x)=h(y)] \leq 1/m$$

x, y عضو U و $x \neq y$

۵.۲۲ Load Factor

به نسبت تعداد مقادیر ذخیره شده در جدول هش به سائز کل جدول هش load factor می گویند
 $e = n / m$
 نکته : اگر جدول هش یک universal family بود میانگین زمان دسترسی به طولانی ترین زنجیره (chain) از
 order $O(1 + e)$ می باشد
 کاربرد دیگر load factor مدیریت حافظه است
 به این صورت که سعی میشود این نسبت بین ۵۰ تا ۹۰ باشد اگر بیشتر شد باید برای جلوگیری از
 collision سائز جدول هش را افزایش داد

۱ که شبه کد Rehash کردن در زیر آمده است.؟؟

```

Data: T
Result: new table
Rehash( T ):
loadFactor = T.numberOfKeys/T.size;
if not at end of this document then
    Create new Tnew of size, Tsize * 2;
    Choose hnew with cardinality of Tnew size;
    while T not empty do
        | Insert object in Tnew using new hnew;
    end
end
T = Tnew, h = hnew;
  
```

Algorithm 1: How to Rehash hash table?

مرتبه ی زمانی: $O(n)$

مرتبه زمانی تحلیل زمانی: $O(1)$

۶.۲۲ یک تابع هش universal family برای اعداد

$$\{ m \bmod (p \bmod (ax+b)) = hp(x) \} = Hp$$

p یک عدد رندوم است.

a, b هر عددی بین صفر تا $p-1$ هستند

مثال : $a = 34, b = 2, p = 10\ 000\ 019$ فرض کنید عدد مورد نظر $x = 1\ 482\ 567$ باشد

$$(2 + 1482567 * 34) \bmod 10000019 = 407185$$

^۱pseudocode

$$407185 \bmod 1000 = 185$$

$$185 = h(x)$$

۷.۲۲ یک تابع هش universal family برای رشته ها

تیپ کلی این مجموعه از توابع هش :

$$\{ p \bmod \sum(S[i] \text{pow}(x, i)) = hp(s) \} = Pp$$

i از صفر تا اندازه ی رشته منها ی یک می باشد. S نام رشته ورودی است
 p یک عدد اول ثابت و x بین یک تا $p-1$ است.
 این هش به Polynomial Hashing معروف است.

۲ شبه کد PolyHash ؟؟

```

Data: S,p,x
Result: hash
hash = 0;
for  $i$  from  $|S| - 1$  down to 0 do
  | hash = ( hash * x + S[i] ) mod p;
end
return hash;

```

Algorithm 2: How to implement PolyHash hash function ?

مثال: $3 = |S|$

$\bullet = \text{hash}$

$S[2] \bmod p = \text{hash}$

$S[2]x + S[1] \bmod p = \text{hash}$

$S[2] \text{pow}(x, 2) + S[1]x + S[0] \bmod p = \text{hash}$

۸.۲۲ یافتن یک زیر رشته در یک رشته

سوال : یک زیررشته P (pattern) و یک رشته ی T (Text) داده شده و باید تمام ایندکس هایی که زیررشته در رشته قرار دارد را خروجی دهید.

راه حل معمولی :

مرحله ی اول یک تابع برای تشخیص برابری دو رشته مینویسیم:

۳ شبه کد AreEqual ؟؟

pseudocode^۲

pseudocode^۳

```

Data: S1,S2
Result: boolean(true or false)
if |S1| != |S2| then
  | return False;
end
for i from 0 to |S1| - 1 do
  | if S1[i] != S2[i] then
  | | return False;
  | end
end
return hash;

```

Algorithm 3: Are Equal tow string ?

زمان اجرای کد بالا $O(|P|)$ است.
مرحله بعد پیاده سازی راه حل است :

شبه کد FindSubStringNative ?? ^۴

```

Data: T,P
Result: positions
positions = empty list;
for i from 0 to |T| - |P| do
  | if AreEqual(T[i..i+|P|-1], P) then
  | | positions.Append(i);
  | end
end
return positions;

```

Algorithm 4: positions of pattern in text

زمان اجرای کد بالا $O(|T||P|)$ است.
اما راه حل بهینه تر استفاده از هشینگ است.

۹.۲۲ الگوریتم RabinKarp

ایده : به جای مقایسه خود زیر رشته های رشته اصلی با الگو میدانیم اگر مقدار هش ان ها برابر نبود پس قطعاً باهم برابر نیستند.
ولی اگر برابر بودند چاره ای جز مقایسه کاراکتر به کاراکتر نیست.
الگوریتم اولیه :

شبه کد initial RabinKarp ^۵ ??

pseudocode^۴
pseudocode^۵

```

Data: T,P
Result: positions
p = big prime , x = random(1, p-1);
positions = empty list;
pHash = PolyHash(P, p, x);
for i from 0 to |T| - |P| do
    tHash = PolyHash( T[i..i+|P|-1], p, x);
    if pHash != tHash then
        | continue;
    end
    if AreEqual(T[i..i+|P|-1], P ) then
        | positions.Append(i);
    end
end
return positions;

```

Algorithm 5: positions of pattern in text

هشدار: زمانی که است که دو زیر رشته باهم برابر باشند که در این صورت زمان اجرا بیشتر میشود ولی یادمان باشد که تعداد collision در یک هش family universal به شدت کم است

نکته: زمان اجرای الگوریتم بالا هم تقریباً همان $O(|T||P|)$ است. برای سریع تر شدن چه کنیم؟

ایده RabinKarp زیر رشته های متوالی یک رشته مقدار هش شان بسیار شبیه هم بوده و میتوان ارتباطی برای زود تر حساب شدن هش ها پیدا کرد.

توجه کنید که تابع هش مورد استفاده PolyHash می باشد

$$Pp = \{ hp(s) = \sum(S[i] \text{pow}(x,i)) \bmod p \}$$

Consecutive substrings

$$\begin{aligned}
 T &= \text{b e a c h} \\
 \text{encode}(T) &= \begin{bmatrix} 1 & 4 & 0 & 2 & 7 \end{bmatrix} \quad |P| = 3 \\
 h(\text{"ach"}) &= 0 + 2x + 7x^2 \\
 &\quad \downarrow \times \downarrow \times \\
 h(\text{"eac"}) &= 4 + 0 + 2x^2 \\
 H[2] &= h(\text{"ach"}) = 0 + 2x + 7x^2 \\
 H[1] &= h(\text{"eac"}) = 4 + 0x + 2x^2 = \\
 &= 4 + x(0 + 2x) = \\
 &= 4 + x(0 + 2x + 7x^2) - 7x^3 = \\
 &= xH[2] + 4 - 7x^3
 \end{aligned}$$

شکل ۴.۲۲: ارتباط بین مقدار هش زیر رشته های متوالی

در نهایت رابطه ی بازگشتی زیر بین زیر رشته های متوالی برداشت میشود:

$$H[i] = x * H[i+1] + (T[i] - (T[i+|P|] * \text{pow}(x, |P|)) \bmod p)$$

که $\text{pow}(x, |P|)$ را میتوان یکبار حساب و بعداً فقط استفاده نمود
 برای شروع کد زنی باید قیل از حل تابعی پیاده سازی شود که با روش سریع و با استفاده از رابطه ی باگشتی

کل هش زیر رشته ها را محاسبه کند.

شبه کد PreComputeHashes ؟؟

```

Data: T,|P|,p,x
Result: H
H = array of length |T| - |P| + 1 ;
S = T[|T| - |P| ... |T|-1];
H[|T| - |P|] = PolyHash(S, p, x);
y = 1;
for i from 1 to |P| do
  | y = (y*x) mod p;
end
for i from |T|-|P|-1 down to 0 do
  | H[i] = (x*H[i+1]+T[i]-y*T[i+|P|]) mod p ;
end
return H;

```

Algorithm 6: hash of substring

حال ربین کارپ اصلی را بازنویسی میکنیم:

شبه کد main RabinKarp ؟؟

```

Data: T,P
Result: positions
p = big prime , x = random(1, p-1);
positions = empty list;
pHash = PolyHash(P, p, x);
H = PreComputeHashes(T, |P|,p,x);
for i from 0 to |T| - |P| do
  | if pHash != H[i] then
  | | continue;
  | end
  | if AreEqual(T[i..i+|P|-1],P) then
  | | positions.Append(i);
  | end
end
return positions;

```

Algorithm 7: positions of pattern in text

زمان اجرای الگوریتم بالا از مرتبه ی زمانی $O(|T|+(q+1)*|P|)$ است که چون معمولا q کوچک است پس مرتبه ی آن خیلی کم تر از $O(|T|^*|P|)$

[۱] [۲] [۳] [۴]

Bibliography

- [1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd ed., 2009.
- [2] “Dos (denial of service) attack tutorial: Ping of death, ddos.” <https://www.guru99.com/ultimate-guide-to-dos-attacks.html>.
- [3] “Effective denial of service attacks against web application platforms.” <https://fahrplan.events.ccc.de/congress/2011/Fahrplan/events/4680.en.html>.
- [4] “Hash tables and hash functions” https://www.youtube.com/watch?v=KyUTuwz_b7Q. Accessed: 10 January 2020.