توضیح الگوریتم هافمن

در کدگذاری هافمن از روش خاصی برای تولید کد مخصوص هر نماد استفاده می شود. در این روش که به روش کد های پیشوندی هم معروف است رشته ای که نماینده یک کاراکتر است **هیچ گاه** پیشوند رشته دیگر که نماینده یک کاراکتر دیگر است نمی باشد.

مثال) فرض کنید ۴ کاراکتر بصورت a ، b ، c و d داریم و کد متناظر به هر یک از این کاراکتر ها به ترتیب بصورت 00 ، 01 ، 0 و 1 باشند. این شیوه کدگذاری منجر به ابهام می شود چرا که کد متناظر با کاراکتر c پیشوند کد های متناظر با a و b است. اگر رشته کدگذاری شده بصورت 0001 داشته باشیم رشته اصلی آن می تواند cccd یا ccb یا acd یا ab باشد.

در کدگذاری هافمن دو بخش اصلی وجود دارد :

۱. درخت هافمن را با استفاده از کاراکتر های ورودی بسازیم.
۲. درخت ساخته شده را پیمایش کنیم و کد های مربوط به هر کاراکتر را معین کنیم.

ساخت درخت هافمن

ورودی آرایه ای از کاراکتر های یکتا به همراه تعداد تکرار آنها (frequency) و خروجی درخت هافمن

۱. برای هر یک از کاراکتر های یکتا یک نود بسازیم و هیپ مینیمم همه این نود ها را تشکیل می دهیم (هیپ مینیمم بعنوان صف اولویت استفاده می شود و تعداد تکرار کاراکتر ها (frequency) برای مقایسه دو نود در هیپ مینیمم استفاده می شود. لذا نود یا کاراکتر با کمترین frequency در ریشه قرار می گیرد.)
۲. دو نود با فرکانس کمتر را از هیپ استخراج می کنیم.
۳. یک نود دیگر میسازیم که فرکانس آن برابر است با مجموع فرکانس های دو نودی که در مرحله قبل استخراج کردیم. اولین نودی که استخراج کردیم را به عنوان فرزند چپ و دومین نود را به عنوان فرزند راست این نود قرار می دهیم. سپس این نود را در هیپ مینیمم قرار می دهیم.
۴. مرحله ۲ و ۳ را ادامه می دهیم تا جایی که فقط یک نود در هیپ باقی بماند که این نود ریشه درخت است و درخت هافمن تکمیل شده است.

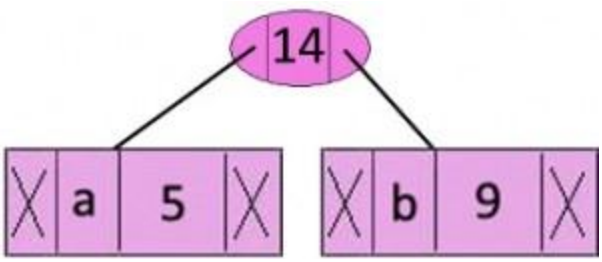
مثال)

ورودی :

character	Frequency
a	5
b	9
c	12
d	13
e	16
f	45

مرحله ۱) هیپ مینیمم را که شامل ۶ نود که هرکدام ریشه درختی با یک نود هستند را میسازیم.

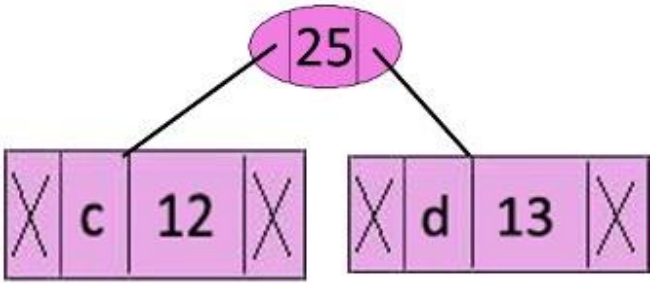
مرحله ۲) دو نود با فرکانس کمتر را از هیپ استخراج می کنیم و نود جدید را میسازیم.



اکنون هیپ شامل ۵ نود است که ۴ نود آن ریشه درخت با یک عضو هستند و یک نود آن ریشه درختی با ۳ عضو است.

character	Frequency
c	12
d	13
Internal Node	14
e	16
f	45

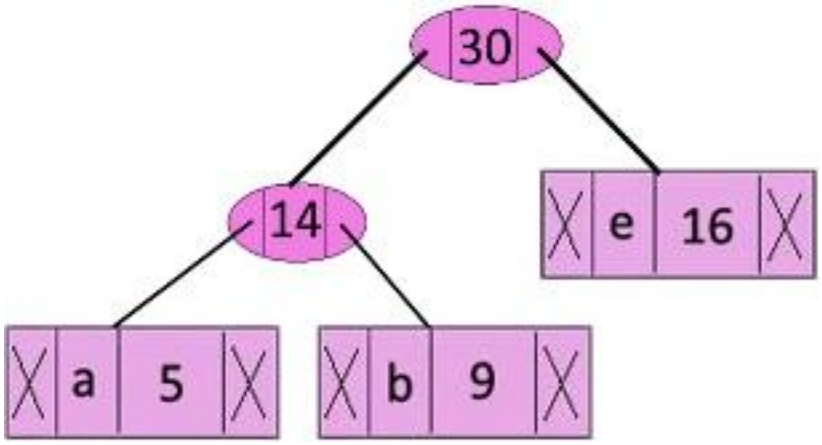
مرحله ۳) دو نود با فرکانس کمتر را استخراج کرده و نود درونی (internal node) جدید با فرکانس ۱۲ + ۱۳ = ۲۵ را میسازیم.



اکنون هیپ شامل ۴ نود است که ۲ نود ریشه درخت با یک عنصر و ۲ نود دیگر ریشه درختی با بیشتر از یک عضو هستند.

character	Frequency
Internal Node	14
e	16
Internal Node	25
f	45

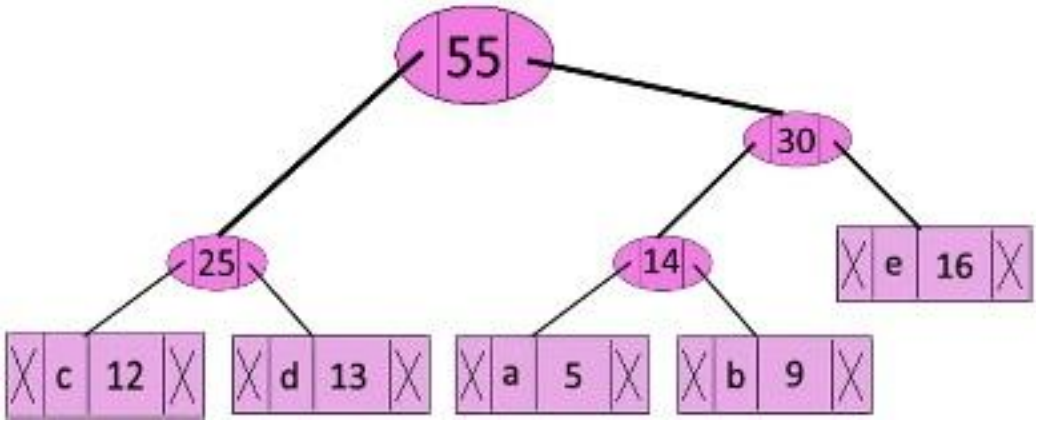
مرحله ۴) دو نود با فرکانس کمتر را استخراج می کنیم. یک نود درونی جدید با فرکانس ۱۴ + ۱۶ = ۳۰ میسازیم



اکنون هیپ شامل ۳ نود است

character	Frequency
Internal Node	25
Internal Node	30
f	45

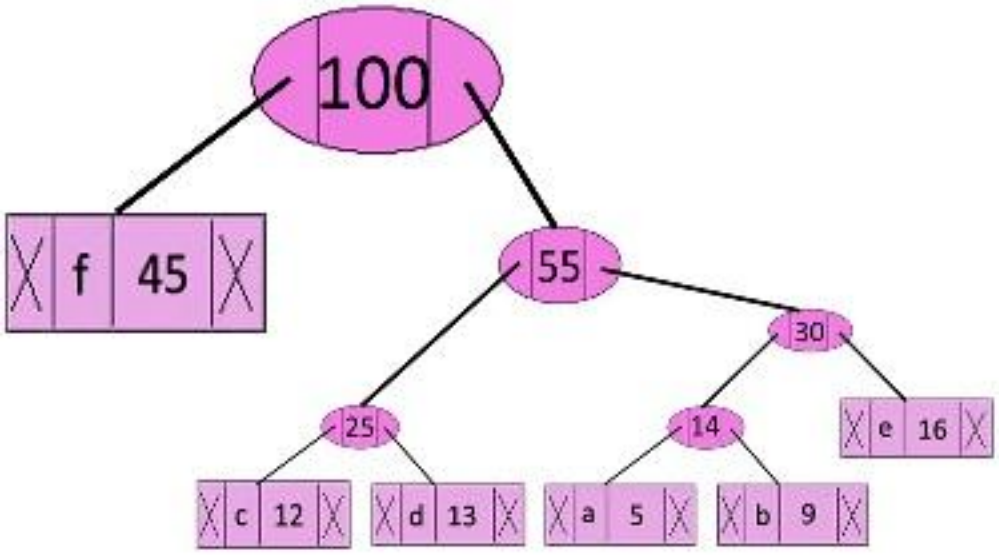
مرحله ۵) دو نود با فرکانس کمتر را استخراج کرده و نود درونی جدید با فرکانس ۲۵ + ۳۰ = ۵۵ را میسازیم



اکنون هیپ شامل ۲ نود است

character	Frequency
f	45
Internal Node	55

مرحله ٦) دو نود با فرکانس کمتر را استخراج کرده و نود درونی را میسازیم



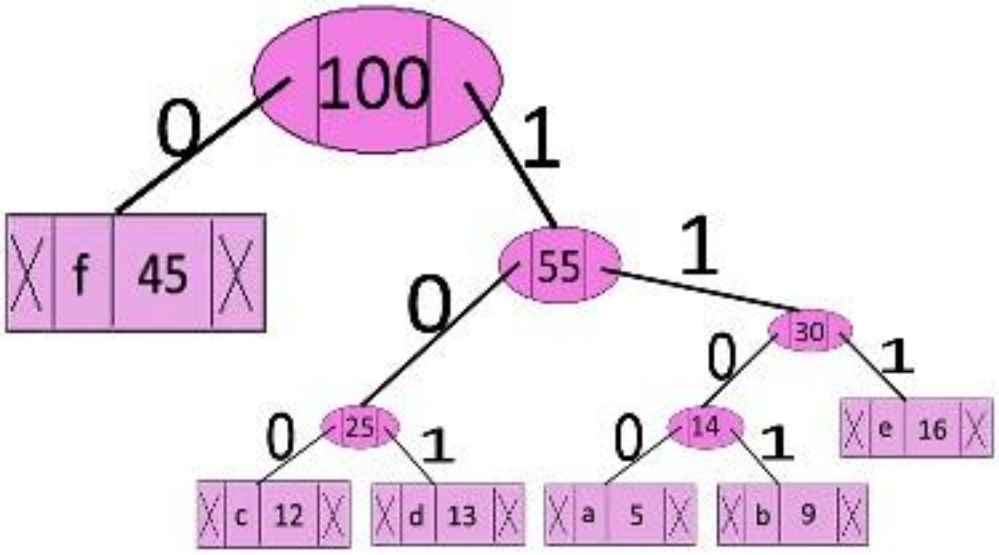
اکنون هیپ تنها شامل یک نود است

```
character       Frequency
Internal Node    100
```

از آنجایی که تنها یک نود در هیپ باقی مانده این الگوریتم در اینجا به پایان می رسد.

تعیین کد از طریق درخت هافمن

درخت را پیمایش می کنیم. از یک آرایه کمکی استفاده می کنیم. هنگامی که به سمت فرزند چپ حرکت میکنیم 0 را در آرایه مینویسیم و هنگامی که به سمت فرزند راست حرکت می کنیم 1 را در آرایه مینویسیم. هنگامی که به برگ رسیدیم آرایه را چاپ می کنیم.



کد هر کاراکتر بصورت زیر است

```
character   code-word
    f          0
    c          100
    d          101
    a          1100
    b          1101
    e          111
```

تا به اینجا مربوط به Huffman Coding می شود اما بحث دیگری که مطرح می شود Huffman Decoding است که به decode کردن کد هایی که در قسمت قبل در مورد شیوه ساختن آنها بحث کردیم می پردازد.

برای decode کردن ما به درخت هافمن نیاز داریم. برای پیدا کردن کاراکتر منتاظر با هر کد از مراحل زیر بهره می بریم :

۱. از ریشه آغاز میکنیم و مراحل زیر را انجام می دهیم تا به برگ برسیم.
۲. اگر بیت جاری (یعنی بیتی که در حال حاضر آنرا خواندیم) 0 بود به نود سمت چپ (فرزند چپ) می رویم.
۳. اگر 1 بود به نود سمت راست می رویم.
۴. اگر در طی پیمایش به برگ برخورد کردیم کاراکتر آن برگ را چاپ می کنیم و سپس به کار خود از مرحله یک ادامه میدهیم.

البته در پیاده سازی که در این پروژه انجام شده است از روش ساده تری استفاده شده است. و توجه به این نکته که در هنگام decode کردن یک فایل ما درخت هافمنی که هنگام encode کردن فایل ابتدایی ساخته شده است را در اختیار نداریم حائز اهمیت است. به همین دلیل ما مجبوریم اطلاعات لازم برای decode کردن را در فایل encode شده قرار بدهیم. (روش استفاده از header)

همچنین باید توجه داشت که ورودی اولیه ما در این پروژه فایل متنی است. و در این فایل اطلاعاتی از کاراکتر ها و تعداد تکرار (فرکانس) آنها در اختیار نیست. لذا قبل از اجرای الگوریتم بر روی رشته متن ورودی باید یکبار کاراکتر های یکتا و تعداد تکرار هر یک از آنها را محاسبه کرده سپس از طریق آن درخت هافمن را تشکیل بدهیم و در آخر با استفاده از درخت متن را کدگذاری کنیم.

مقایسه سایز ورودی با خروجی

مثال) ورودی :

```
Hello World!
```

خروجی)
توجه داشته باشید که برای ذخیره هر کاراکتر از ۸ بیت استفاده می شود.

```
characters    |    frequency      |    binary Huffman value

                   1                   000
!                  1                   1100
H                  1                   001
W                  1                   100
d                  1                   1101
e                  1                   1110
l                  3                   01
o                  2                   101
r                  1                   1111

Encoded Huffman data:
001111001011010001001011111011011100
Input Size:
12 character occurrences * 8 bits = 96 bits or 12 bytes

Output size:
 : 1 occurrence * 3 bits = 3 bits
!: 1 occurrence * 4 bits = 4 bits
H: 1 occurrence * 3 bits = 3 bits
W: 1 occurrence * 3 bits = 3 bits
d: 1 occurrence * 4 bits = 4 bits
e: 1 occurrence * 4 bits = 4 bits
l: 3 occurrences * 2 bits = 6 bits
o: 2 occurrences * 3 bits = 6 bits
r: 1 occurrence * 4 bits = 4 bits
Total Sum: 37 bits approx. 5 bytes
```