موسسه آموز ش عالی ماد

تحلیل و طراحی الگوریتم ها

فصل سوم:
الگوریتم های حریصانه
Greedy algorithms

سید جبار بارخدا
barkhoda@gmail.com

مهر ۹۲

مطالب فصل

- معرفی روش حریصانه
- حل مسائل نمونه و بررسی کارایی آنها
  - مساله کوله پشتی کسری(غیر صفر و یک)
  - کمینه سازی زمان کل در سیستم
  - زمان بندی با مهلت معین (scheduling with deadline)
  - کد هافمن
  - ادغام فایل ها
  - الگوریتم های پیداکردن درخت پوشای مینیمم(MST) در گراف
  - الگوریتم های پیداکردن کوتاه ترین مسیر در گراف

روش حریصانه (Greedy)

- الگوریتم حریصانه، به ترتیب عناصر را گرفته، هر بار آن عنصری را که طبق معیاری مشخص "بهترین" به نظر می رسد، بدون توجه به انتخاب هایی که قبلا انجام داده یا در آینده انجام خواهد داد، بر می دارد.
- الگوریتم حریصانه ، کار را با یک مجموعه تهی آغاز کرده به ترتیب عناصری به مجموعه اضافه می کند تا این مجموعه، حلی برای نمونه ای از یک مسئله را نشان دهد.

روش حریصانه (Greedy)

- غالباً برای حل مسائل بهینه سازی به کار می رود.
- برای هر مسئله ای نمی تواند پاسخ بهینه را به دست آورد.
  - باید اثبات کرد که پاسخ همواره بهینه است یا خیر

- **در روش حریصانه تقسیم به نمونه های کوچکتر صورت نمی پذیرد.**
- با انجام یک سری انتخاب که در هر لحظه بهترین به نظر می رسد عمل می کند.
  - انتخاب بهینه در هر لحظه (بهینه محلی - Local Optimum)
  - تصمیم درباره انتخاب یا رد یکی از داده های ورودی غیرقابل برگشت است.

مثال (مسئله خرد کردن پول)

- هدف برگرداندن باقیمانده پول با حداقل تعداد سکه ها می باشد.
- حل با روش حریصانه
  - در آغاز هیچ سکه ای در مجموعه نداریم.
  - سکه با ارزش بیشتر انتخاب می شود. **(روال انتخاب)**
  - باید بررسی کنیم با افزودن این سکه به بقیه پول، جمع کل آنها از چیزی که باید باشد بیشتر می شود یا خیر.**(بررسی امکان سنجی)**
  - اگر با افزودن این سکه، بقیه پول از میزان لازم بیشتر نشود، این سکه به مجموعه اضافه می شود.
  - بررسی می شود که آیا تمام پول پرداخت شده است یا نه **(بررسی راه حل)**
  - اگر هنوز مقداری مانده بود مجدداً از مرحله **انتخاب سکه**، فرآیند تکرار می شود.
- این تکرار تا زمانی انجام می شود که با سکه های موجود بقیه پول به طور کامل پرداخت گردد و یا اینکه امکان پرداخت پول با این سکه ها وجود نداشته باشد.

مثال (مسئله خرد کردن پول)

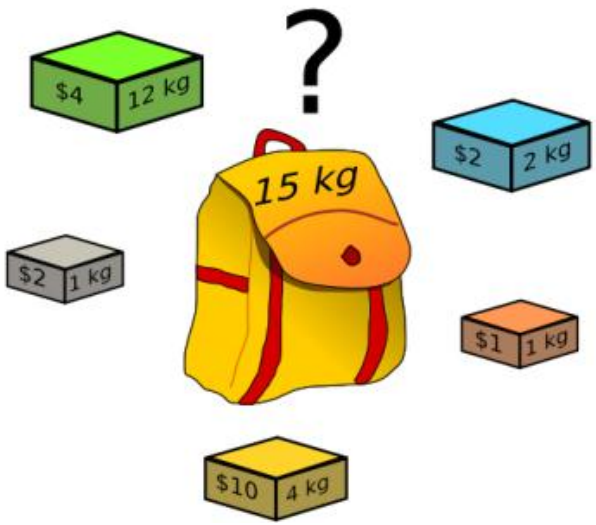
- زمانی که این الگوریتم کار نمی کند:
  - پرداخت ۱۸ سنت با سکه های ۱ ، ۶ و ۷ سنتی
    - حریصانه: دو سکه ۷ سنتی و چهار سکه ۱ سنتی
    - بهینه: سه تا سکه ۶ سنتی
  - پرداخت ۱۶ سنت با سکه های ۱۲، ۱۰، ۵ و ۱ سنتی
    - حریصانه: یک سکه ۱۲ سنتی و چهار سکه ۱ سنتی
    - بهینه: یک سکه ۱۰ سنتی، یک سکه ۵ سنتی و یک سکه ۱ سنتی

- این الگوریتم همواره جواب بهینه را نمی دهد.

روش حریصانه

- هر دور تکرار الگوریتم حریصانه شامل مراحل زیر است:
  - **روال انتخاب:** از بین عناصر، بهترین عنصر درنظر گرفته می شود. این انتخاب براساس یک معیار حریصانه که به طور محلی بهترین جواب را در هر لحظه انتخاب می کند شکل می گیرد.
  - **بررسی امکان سنجی:** بررسی می شود که آیا با انتخاب آن مولفه امکان رسیدن به جواب وجود دارد یا خیر
    - درصورت مثبت بودن پاسخ، آن مولفه را به مجموعه جواب اضافه می کنیم
    - در صورت منفی بودن پاسخ، آن مولفه را برای همیشه کنار می گذاریم.
  - **بررسی جواب:** با افزودن هر عنصر به مجموعه جواب، بررسی می کنیم اگر جواب مسئله حاصل شده است. جواب به دست آمده بهینه فرض خواهد شد و کار تمام است.

مساله کوله پشتی کسری(غیر صفر و یک)



- دراین مسئله امکان انتخاب کسری از هر شی وجود دارد. اگر کسر $0 \le x_i \le 1$ از شی i انتخاب شود ارزش $p_i\ x_i$ بدست می آید.
- هدف:

بیشینه کردن رابطه مقابل $\sum_{i=1}^{n} p_i x_i$

به شرط اینکه: $\sum_{i=1}^{n} w_i x_i < M$

مساله کوله پشتی کسری(غیر صفر و یک)

- راه حل های حریصانه:

۱. مرتب کردن اشیا به ترتیب بیشترین ارزش و انتخاب آنها

۲. مرتب کردن اشیا به ترتیب کمترین وزن و انتخاب آنها

۳. مرتب کردن اشیا به ترتیب بیشترین مقدار ارزش واحد وزن (p_i / w_i) و انتخاب آنها

مثال

- ظرفیت کوله پشتی ۲۰ می باشد.

وزن	۱۰	۱۵	۱۸
ارزش	۱۵	۲۴	۲۵

- جدول زیر نتیجه سه راه حل مختلف بیان شده را نشان می دهد:

	(x_1, x_2, x_3)	$\Sigma w_i x_i$	$\Sigma p_i x_i$
1	(1, 2/15, 0)	20	28.2
2	(0, 2/3, 1)	20	31
3	(0, 1, 1/2)	20	**31.5**

- روش سوم یعنی انتخاب براساس بیشترین ارزش هر واحد وزن شیئ، جواب بهینه را می دهد.

مسئله زمان بندی (scheduling)

- دو نوع مسئله زمان بندی داریم:

۱. **کمینه سازی زمان کل در سیستم** برای انتظار کشیدن و سرویس دهی (زمان بودن در سیستم)
   - مثال: آرایشگاه

۲. **زمان بندی با مهلت معین (scheduling with deadline)**

مساله کمینه سازی زمان کل در سیستم

مثال:

شماره کار	۱	۲	۳
زمان سرویس دهی	۵	۱۰	۴

کار	زمان بودن در سیستم
۱	۵ (زمان سرویس دهی)
۲	۵ (انتظار برای انجام کار ۱) + ۱۰ (زمان سرویس دهی)
۳	۵ (انتظار برای انجام کار ۱) + ۱۰ (انتظار برای انجام کار ۲) + ۴ (زمان سرویس دهی)

زمان کل بودن در سیستم برای زمان بندی فوق

$$5 + (5 + 10) + (5+ 10 + 4) = 39$$

مساله کمینه سازی زمان کل در سیستم

- لیست تمام زمان بندی های ممکن

زمان بندی	کل زمان بودن در سیستم
[1, 2, 3]	$5 + (5 + 10) + (5 + 10 + 4) = 39$
[1, 3, 2]	$5 + (5 + 4) + (5 + 4 + 10) = 33$
[2, 1, 3]	$10 + (10 + 5) + (10 + 5 + 4) = 44$
[2, 3, 1]	$10 + (10 + 4) + (10 + 4 + 5) = 43$
[3, 1, 2]	$4 + (4 + 5) + (4 + 5 + 10) = 32$
[3, 2, 1]	$4 + (4 + 10) + (4 + 10 + 5) = 37$

- زمان بندی [3,1,2] بهینه می باشد.
- مرتبه الگوریتم برابر با تعداد حالات جایگشت کارها می باشد که از درجه **فاکتوریل** است.

روش حریصانه برای کمینه سازی زمان کل در سیستم

- ملاک انتخاب کارها:
  - ابتدا کاری انتخاب شود که زمان سرویس دهی کمتری داشته باشد.

شماره کار	۱	۲	۳
زمان سرویس دهی	۵	۱۰	۴

[3 , 1 , 2]

  - این انتخاب باعث می شود کارهای بعدی کمتر منتظر بمانند و در نتیجه زمان کل بودن آنها در سیستم کاهش می یابد.

الگوریتم روش حریصانه برای کمینه سازی زمان کل در سیستم

- برای یافتن زمان بندی بهینه، کارها را بر اساس زمان سرویس دهی آنها به ترتیب صعودی **مرتب** می کنیم.
- کارها را به ترتیب زمان سرویس کمتر، انتخاب می کنیم.
- پیچیدگی الگوریتم: $O(n \lg n)$
- این روش همواره جواب بهینه را می هد.
- قضیه: تنها زمان بندی که کل زمان بودن در سیستم را کمینه می کند، زمان بندی ای است که در آن کارها برحسب افزایش زمان سرویس مرتب می شوند.

مساله زمان بندی با مهلت معین

- در این حالت، اجرای هر کاری باید تا زمان مشخصی شروع شود در غیر این صورت آن کار دیگر قابل اجرا نخواهد بود.
- مدت زمان انجام کارها برابر است.
- **مسئله:** تعیین زمان بندی با سود کل بیشینه، با این فرض که هر کاری دارای سودی است و فقط وقتی قابل حصول است که آن کار در مهلت مقررش شروع شود.

- مثال: (زمان سرویس برای هر کار ۱ واحد است)

کار	مهلت	سود
۱	۲	۳۰
۲	۱	۳۵
۳	۲	۲۵
۴	۱	۴۰

همه زمان بندی های ممکن

- زمان بندی [1, 2] غیر ممکن است.

کار	مهلت	سود
۱	۲	۳۰
۲	۱	۳۵
۳	۲	۲۵
۴	۱	۴۰

زمان بندی	سود کل
[1, 3]	30 + 25 = 55
[2, 1]	35 + 30 = 65
[2, 3]	35 + 25 = 60
[3, 1]	25 + 30 = 55
[4, 1]	**40 + 30 = 70**
[4, 3]	40 + 25 = 65

اصطلاحات

- **دنباله امکان پذیر (feasible)**: دنباله ای از کارها که در آن همه کارها بتوانند به ترتیب و در مهلت مقرر خود آغاز شوند.
  - مثال: دنباله [1 , 4] امکان پذیر ولی [4 ,1] امکان پذیر نمی باشد.
- **مجموعه امکان پذیر**: مجموعه ای از کارها که برای آن حداقل یک دنباله امکان پذیر وجود داشته باشد.
  - مثال: مجموعه {1, 4} امکان پذیر ولی {2, 4} امکان پذیر نمی باشد.
- **دنباله بهینه**: یک دنباله امکان پذیر با حداکثر سود
  - مثال: دنباله [4, 1] دنباله بهینه است.

بررسی امکان پذیری

- فرض کنید S مجموعه ای از کارها باشد. در این صورت مجموعه S امکان پذیر خواهد بود اگر و فقط اگر دنباله حاصل از مرتب شدن کارهای S بر اساس مهلت های **غیر نزولی**، امکان پذیر باشد.
  - مثال آیا مجموعه { ۱, ۲, ۴, ۷ } امکان پذیر می باشد.

کار	مهلت	سود
۱	3	40
۲	1	35
۳	1	30
4	3	25
5	1	20
6	3	15
7	2	10

[2, 7, 1, 4]
↑ ↑ ↑ ↑
1 2 3 3

- خیر، زیرا کار ۴ در مهلتش نمی تواند زمان بندی شود.

الگوریتم حریصانه

sort the jobs in ***non-increasing*** order by ***profit***;

S = ϕ;

while (the instance is not solved){

select next job; //selection procedure

if (S is feasible with this job added) //feasible check

add this job to S;

if (there are no more jobs) //solution check

the instance is solved;

}

Sort the elements within S, in non-decreasing order by deadline;

مثال

کار	مهلت	سود
۱	3	40
۲	1	35
۳	1	30
4	3	25
5	1	20
6	4	15
7	2	10

- $S = \Phi$
- $S = \{1\}$
- $S = \{1,2\}$ ([2, 1] امکان پذیر)
- $\{1, 2, 3\}$ رد می شود.
- $S = \{1,2, 4\}$ ([2, 4, 1] امکان پذیر)
- $\{1, 2, 4, 5\}$ رد می شود.

- $S=\{1, 2, 4, 6\}$ ([2, 4, 1, 6] امکان پذیر)
- $\{1, 2, 4,6, 7\}$ رد می شود.

$S = \{1,2,4,6\} \Longrightarrow$ [2, 1, 4, 6] , [2, 4, 1, 6]

کد هافمن(Huffman Code)

- **کدگذاری هافمن** یک الگوریتم کدگذاری برای فشرده‌سازی اطلاعات است.
- در کدگذاری هافمن، از روشی خاص برای انتخاب نحوه نمایش هر نویسه استفاده می‌شود.
- برای هر نویسه یک کد باینری با طول متغیر استفاده می شود.
- در این روش، رشته‌ای که نشان دهنده یک نویسه خاص است هیچ گاه پیشوند رشته دیگر که نمایانگر نویسه دیگر است، نمی‌باشد.
- در این روش نویسه‌های پرکاربردتر(یا با تکرار بیشتر) با رشته‌های بیتی کوتاهتری نسبت به آن‌هایی که کاربردشان کمتر است، نشان داده می‌شوند.

کدینگ هافمن

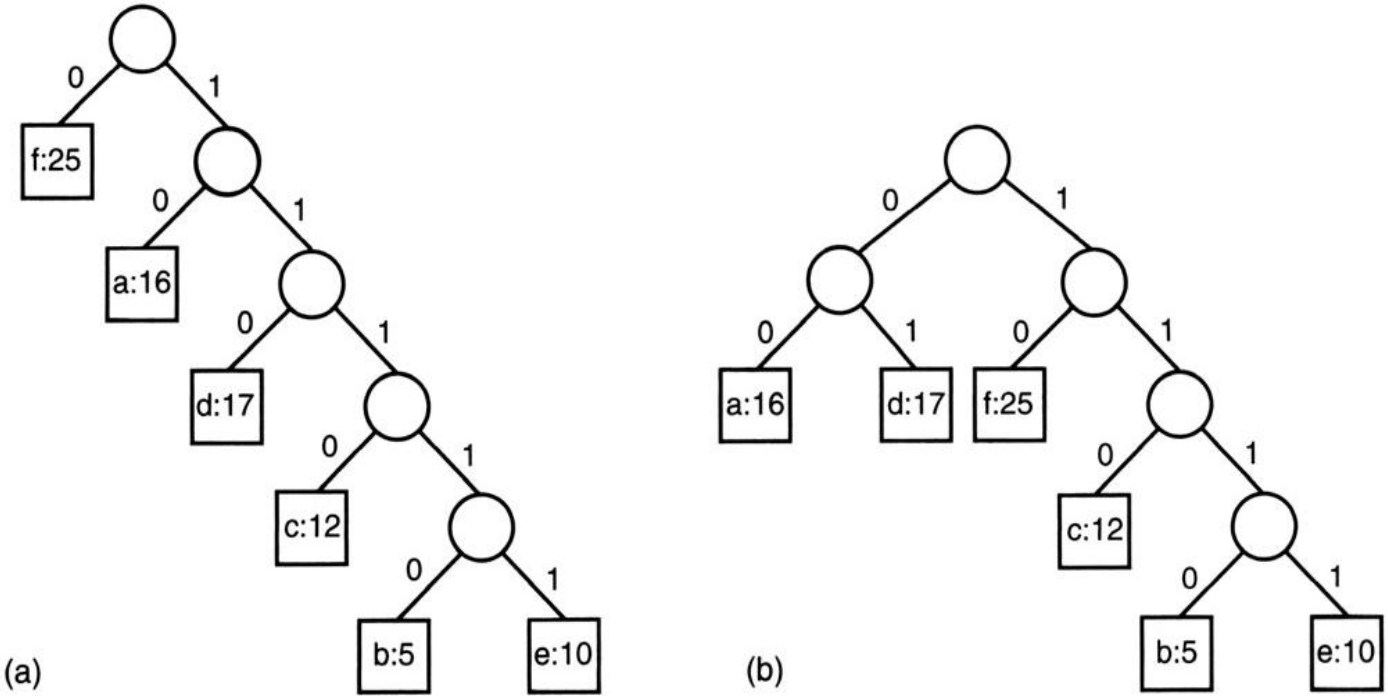
کاراکتر	فراوانی	C1(طول ثابت)	C2	هافمن(C3)
a	۱۶	000	10	00
b	۵	001	11110	1110
c	۱۲	010	1110	110
d	۱۷	011	110	01
e	۱۰	100	11111	1111
f	۲۵	101	0	10

Bits(C1)=16*3+5*3+12*3+17*3+10*3+25*3=255

Bits(C2)=16*2+5*5+12*4+17*3+10*5+25*1=231

Bits(C3)=16*2+5*4+12*3+17*2+10*4+25*2=212

کد پیشوندی با طول متغیر(Variable Length Prefix Code)



Bits(C1)=16(3)+5(3)+12(3)+17(3)+10(3)+25(3)=255

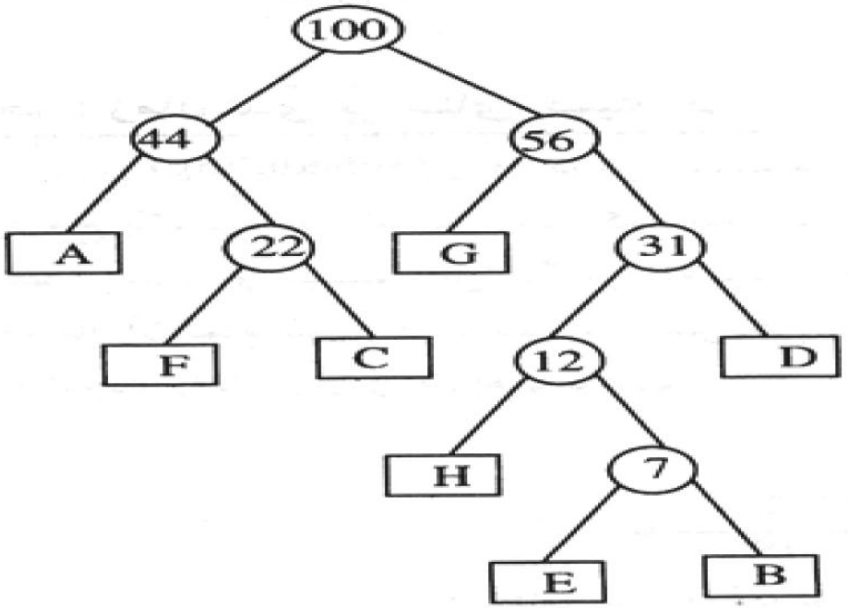
Bits(C2)=16(2)+5(5)+12(4)+17(3)+10(5)+25(1)=231

Bits(C3)=16(2)+5(4)+12(3)+17(2)+10(4)+25(2)=212

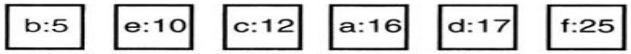
کد هافمن– مثال

- فرض کنیم A,B,C,D,E,F,G,H ، ۸ عنصر اطلاعاتی با وزن های مشخص شده هستند:

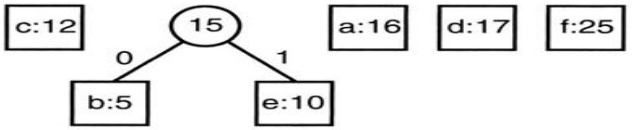
عنصر اطلاعاتی	A	B	C	D	E	F	G	H
وزن	22	5	11	19	2	11	25	5



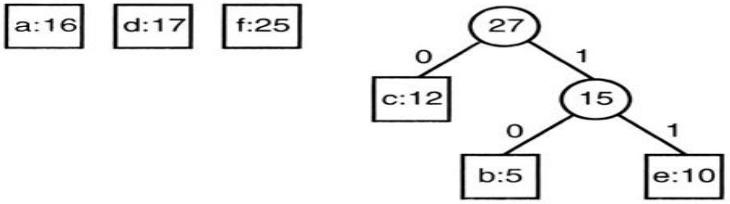
کدینگ هافمن بهینه(Optimal)



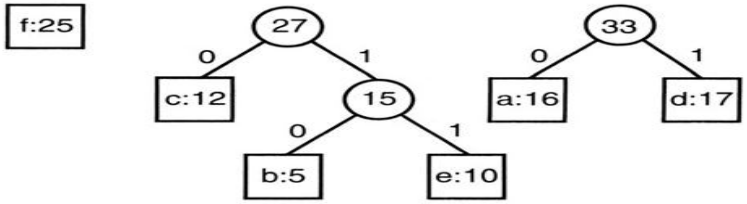
(0)



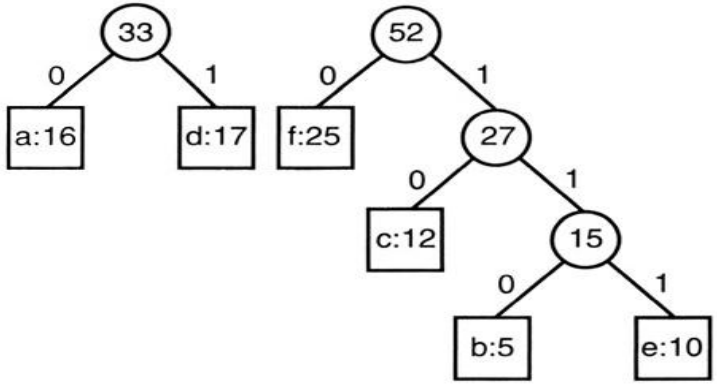
(1)



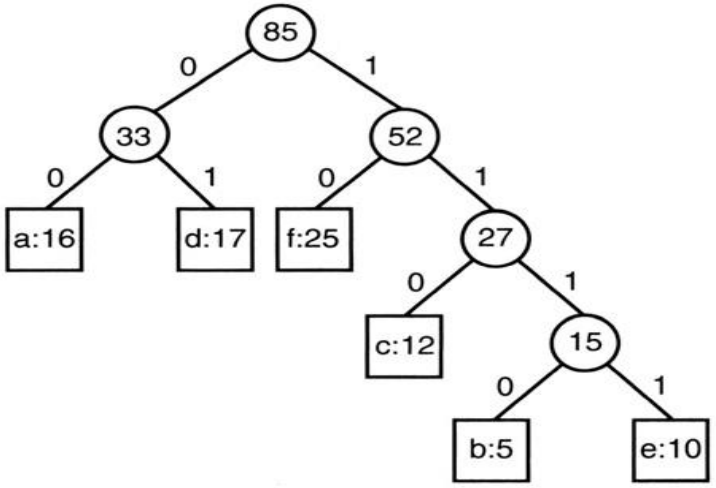
(2)



(3)



(4)



(5)

الگوریتم هافمن

Priority queue : Highest priority (lowest frequency)
Element is removed first

```
for(i=1; i<=n-1; i++){
        remove(PQ,p);
        remove(PQ,q);
        r=new nodetype;
        r->left=p;
        r->right=q;
        r->frequency=p->frequency + q ->frequency;
        insert(PQ, r);
}
 remove(PQ, r)
 return r;
```

Priority queue (heap) Initialization O(n)
Each heap operation O(logn)
Huffman algorithm complexity O(nlogn).

ادغام دودویی و بهینه فایل ها

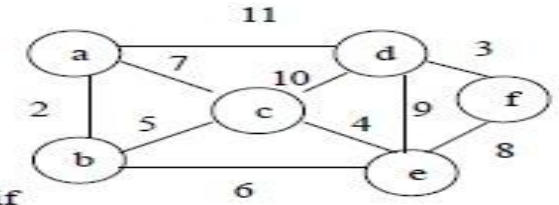
- تعدادی فایل با تعداد عناصر مشخص داریم، می خواهیم آنها را با هم ادغام کنیم. در نهایت یک فایل خواهیم داشت.
- عناصر هر فایل مرتب هستند.
- باید هزینه ادغام بهینه باشد.
- راه حل حریصانه
  - در هر مرحله دو فایل با تعداد رکورد های کمتر ادغام می شوند.

الگوریتم های درخت پوشای مینیمم(MST)

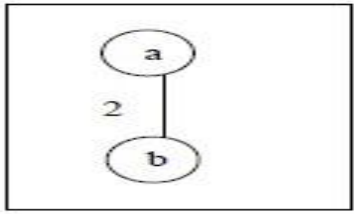
- گراف، درخت، درخت پوشا، درخت پوشای کمینه
- الگوریتم های تولید درخت پوشای کمینه
  - الگوریتم پریم
  - الگوریتم کراسکال
- نکته: الگوریتم های کروسکال و پریم همواره درخت های پوشای کمینه را ایجاد می کنند.

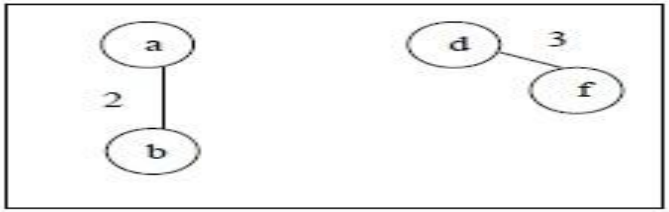
کراسکال

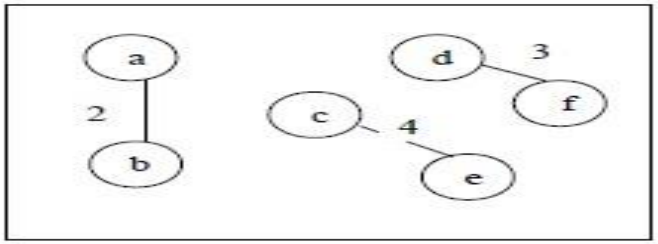
```
Procedure kruskal
   T  —  ∅
   While ( |T| < n-1 ) and ( |E| ≠ 0 )  do
      Select  edge e   from E with minimum cost
      Delete  e  from  E
      If  e  does not  create a cycle in T then
          add e  to T
      endif
   repeat
   if |T| < n-1   then write ('no spanning Tree ') endif
end.
```

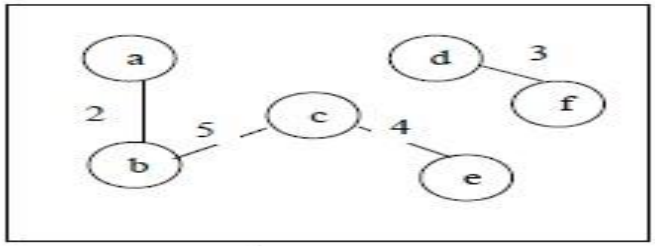


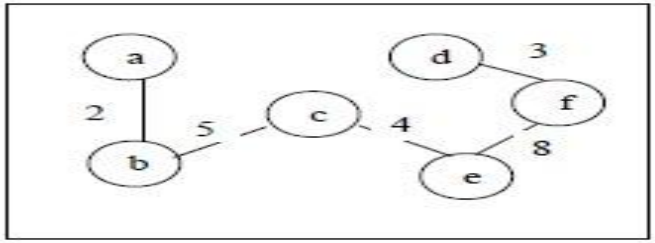
مراحل ساخت درخت:











الگوریتم های کوتاهترین مسیر در گراف

- دایجسترا
  - کوتاهترین مسیرها در گراف از یک راس به بقیه رئوس را می دهد.
  - با یال های منفی جواب نمی دهد.
  - به روش حریصانه عمل می کند.
  - single-source shortest path
- بلمن فورد
  - تک مبدا
  - با یال منفی جواب می دهد ولی با وزن منفی جواب نمی دهد.
- الگوریتم فلوید
  - به روش برنامه نویسی پویا عمل می کند.
  - کوتاهترین مسیرها از یک راس به بقیه رئوس را می دهد.
  - با یال منفی مشکلی ندارد ولی با دور منفی کار نمی کند.