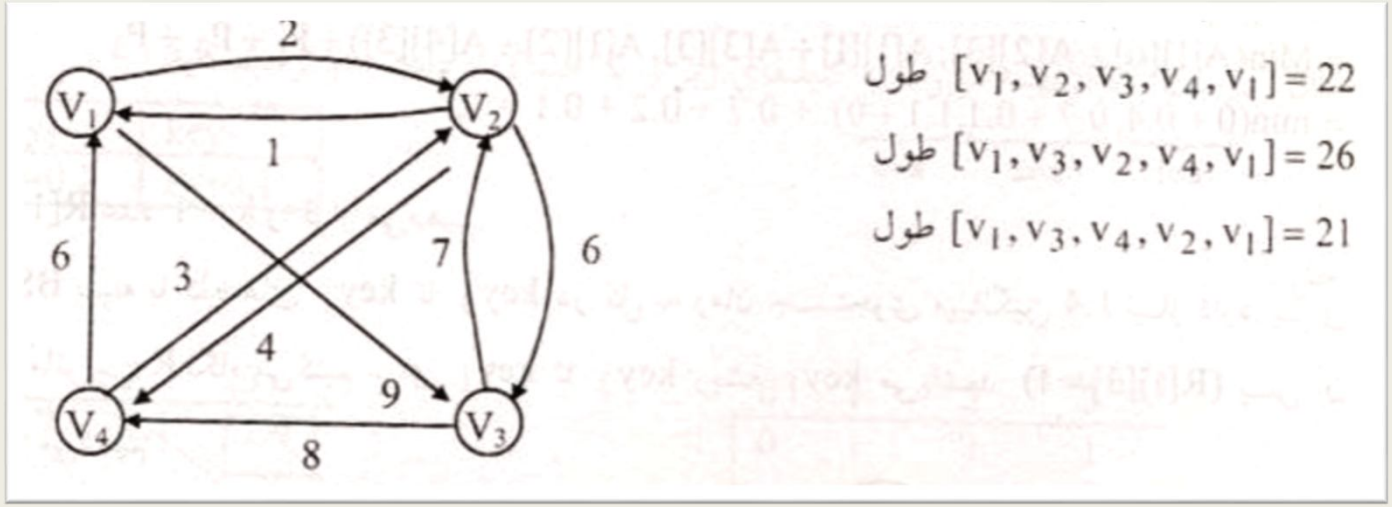
فروشنده دوره گرد

- فروشنده ای می خواهد از تعدادی شهر که توسط جاده هایی به یکدیگر متصل هستند عبور کند و در انتها به شهر اولیه خود برگردد .
- هدف یافتن کوتاه ترین مسیر برای فروشنده است به نحوی که از تمام شهرها دقیقاً یک بار عبور کند و این مساله استاندارد فروشنده دوره گرد است.
- در یک گراف جهت دار یک تور ، که به آن مدار هامیلتون نیز گفته می شود عبارت از مسیری از یک راس به خودش است که از تمام رئوس دیگر دقیقاّ یک بار عبور می کند . منظور از یک تور بهینه در گراف جهت دار وزن دار مسیری از یک نوع است که طول آن حداقل می باشد .

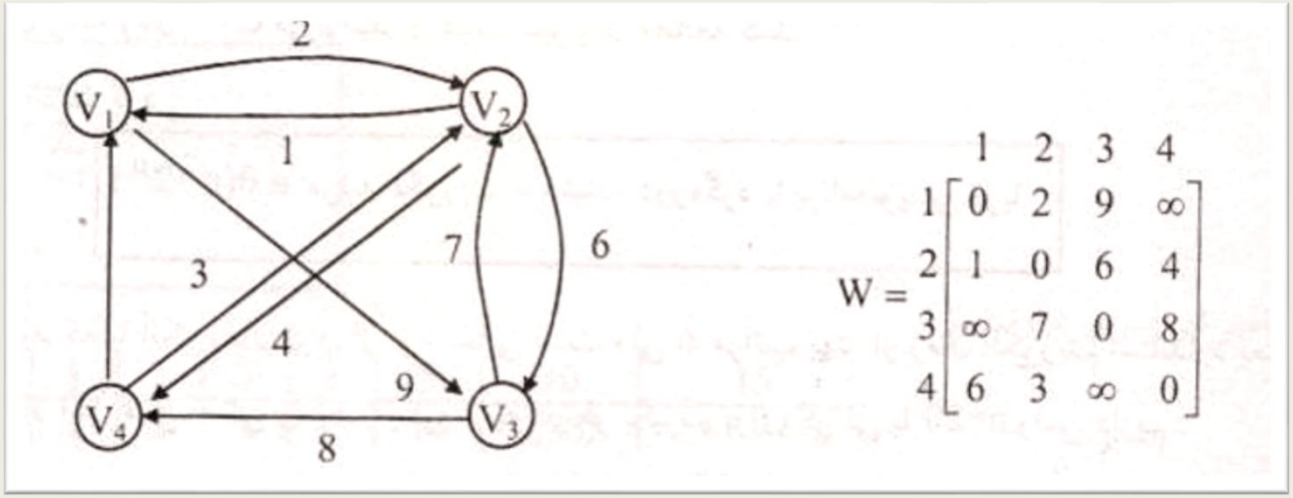
فروشنده دوره گرد

- بنابراین مسئله فروشنده دوره گرد در واقع پیدا کردن یک تور بهینه برای یک گراف جهت دار موزون است.
- توجه کنید که ممکن است گرافی اصلاً تور نداشته باشد. همچنین طول تور بهینه وابسته به انتخاب راس آغازین نیست.
- مثال : در گراف شکل زیر ۳ تور با طول های تعیین شده عبارت اند از :



فروشنده دوره گرد – مثال

- **مثال : در گراف جهت دار موزون زیر با ماتریس هم جواری W داده شده ، طول تور بهینه را به دست آورید.**



$$W = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 2 & 9 & \infty \\ 1 & 0 & 6 & 4 \\ \infty & 7 & 0 & 8 \\ 6 & 3 & \infty & 0 \end{bmatrix} \end{array}$$

- **نکاتی قبل از آغاز حل مثال:**

$$D[i][\phi] = W[i][1]$$

$$D[i][A] = min(W[i][j] + D[v_j][A - \{v_j\}])$$

فروشنده دوره گرد – مثال

حالت اول: مجموعه A تک عضوی

$$D[v_4][\{v_2\}] = W[4][2] + D[v_2][\phi] = 3 + 1 = 4$$

$$D[v_2][\{v_3\}] = W[2][3] + D[v_3][\phi] = 6 + \infty = \infty$$

$$D[v_4][\{v_3\}] = W[4][3] + D[v_3][\phi] = \infty + \infty = \infty$$

$$D[v_2][\{v_4\}] = W[2][4] + D[v_4][\phi] = 4 + 6 = 10$$

$$D[v_3][\{v_4\}] = W[3][4] + D[v_4][\phi] = 8 + 6 = 14$$

فروشنده دوره گرد – مثال

حالت دوم: مجموعه A دو عضوی

$$D[v_4][\{v_2, v_3\}] = \min_{v_i \in \{v_2, v_3\}} (W[4][j] + D[v_j][\{v_2, v_3\} - \{v_j\}])$$

$$= min(W[4][2] + D[v_2][\{v_3\}], W[4][3] + D[v_3][\{v_2\}])$$

$$= min(3 + \infty, \infty + 8) = \infty$$

$$D[v_3][\{v_2, v_4\}] = min(W[3][2] + D[v_2][\{v_4\}], W[3][4]$$

$$+ D[v_4][\{v_2\}]) = min(7 + 10, 8 + 4) = 12$$

$$D[v_2][\{v_3, v_4\}] = min(W[2][3] + D[v_3][\{v_4\}], W[2][4]$$

$$+ D[v_4][\{v_3\}]) = min(6 + 14, 4 + \infty) = 20$$

فروشنده دوره گرد – مثال

حالت سوم: مجموعه A سه عضوی

$$D[v_1][\{v_2, v_3, v_4\}]$$

$$= \min_{v_i \in \{v_2, v_3, v_4\}} (W[1][j] + D[v_j][\{v_2, v_3, v_4\} - \{v_j\}])$$

$$= min \begin{pmatrix} W[1][2] + D[v_2][\{v_3, v_4\}], \\ W[1][3] + D[v_3][\{v_2, v_4\}], \\ W[1][4] + D[v_4][\{v_2, v_3\}] \end{pmatrix}$$

$$= min(2 + 20, 9 + 12, \infty + \infty) = 21$$

پس طول تور بهینه برابر ۲۱ می باشد.

فروشنده دوره گرد –الگوریتم برنامه نویسی پویا

```
void tarvel ( int , n
                const number W[ ][ ] ,
                index  P[ ] [ ] ,
                number & minlength)
    {
        index  i , j , k ;
        number D[1..n][subset of  V -  { v1 } ] ;
        for ( i = 2 ; i ≤ n ; i ++)
```

فروشنده دوره گرد –الگوریتم برنامه نویسی پویا

D[i] [Ø] = W [i] [1] ;

for (k = 1 ; k ≤ n - 2 ; k ++)

for (all subsets A ⊂ V – { v1 } containing k vertices)

for (i such that j≠1 and v_i is not in A){

D[i] [A] = minimum (W [i] [j] + D [v_j] [A – {v_j}]);

P [i] [A] = value of j that gave the minimum;}

D [1] [V – { v_i }] = minimum (W [1] [j] +

D [v_j] [V – { v_1}]);

P [1] [V – { v_1 }] = value of j that gave the minimum;

minlength = D [1] [V – { v_1 }];

}

فروشنده دوره گرد – مثال

تحلیل پیچیدگی فضا و زمان در حالت معمول برای الگوریتم برنامه نویسی پویا برای مسئله فروشنده دوره گرد

عمل اصلی: زمان در هر دو حلقه ی اول و آخر ، در مقایسه با زمان در حلقه میانی چشمگیر نیست، زیرا حلقه میانی حاوی سطوح گوناگون تو در تویی است . دستورات اجرا شده برای هر مقدار V_j را می توان عمل اصلی در نظر گرفت.

اندازه ورودی : n، تعداد رئوس موجود در گراف.

$$T(n) = n^2 2^n \in \theta(n^2 2^n)$$

فروشنده دوره گرد – مثال

توجه کنید با آنکه زمان فوق از نوع نمایی است ولی به مراتب بهتر از زمان الگوریتم استاندارد یعنی (n-1)! است . مثلاً اگر عمل اصلی را یک میکرو ثانیه در نظر بگیریم برای گرافی با n =20 راس داریم

الگوریتم استاندارد $= (20-1)!\ \mu sec = 3857\ year$

الگوریتم برنامه نویسی پویا $= (20^2 \times 2^{20})\mu sec = 7\ min$

البته الگوریتم $\theta(n^2 2^n)$ نیز هنگامی عملی است که n نسبتاً کوچک باشد چرا که مثلاً برای n=70 نیز این الگوریتم سال ها وقت نیاز دارد.

نکته : می توان اثبات کرد حافظه مورد نیاز این الگوریتم از مرتبه $\theta(n2^n)$ است.

بزرگ ترین زیر دنباله (رشته) مشترک (LCS)

- دو رشته X=<A,B,C,B,D,A,B> و Y=<B,D,C,A,B,A> را در نظر بگیرید. Z= <B,C,A> یک زیر رشته مشترک برای این دو رشته است .
- توجه کنید که <B,C,A> لازم نیست دقیقاً پشت سر هم در دو رشته X و Y ظاهر شده باشد بلکه تنها لازم است توالی ظاهر شدن (اول B، بعد Cو بعد A) در دو رشته وجود داشته باشد.

بزرگ ترین زیر دنباله (رشته) مشترک (LCS)

- به عبارت دقیق تر رشته $Z = <z_1 , z_2, \ldots z_k>$ یک زیر رشته برای رشته $X=<x_1 , x_2,\ldots,x_n>$, $k \geq n$ است هر گاه یک رشته اکیدا صعودی مثل $<i_1,i_2,\ldots,i_n>$ از اندیس های x وجود داشته باشد به گونه ای که برای همه $j= 1,2,\ldots,k$ رابطه $x_{i_j} = z_j$ برقرار باشد.
- در مثال فوق زیر رشته <B,C,A> به طول ۳ طـــولانی تـــرین زیر رشته مشـــترک در X و Y نیست. بلکه رشته <B,C,B,A> به طول ۴ طولانی ترین زیر دنباله مشترک آن ها است.

بزرگ ترین زیر رشته مشترک (LCS)–الگوریتم

- فرض کنید دو رشته به صورت a و b با طول های n و m داریم و a_i یعنی کاراکتر i ام رشته a .
- آرایه LCS را به این صورت تعریف می کنیم که هر خانه i,j از آرایه نشان گر اندازه بزرگترین زیر دنباله مشترکی باشد که می توان تنها با استفاده از i کاراکتر اول رشته a و j کاراکتر اول رشته b ساخت.
- هرخانه i,j را می توان با توجه با خانه های کوچکتر به دست آورد به صورتی که
  - اگر ai=bj طول بزرگترین زیر دنباله مشترک i,j برابر با طول بزرگترین زیر دنباله مشترک i-1,j-1 به اضافه ی ۱ است .
  - و اگر ai != bi طول بزرگترین زیر دنباله مشترک آنها برابر با حداکثر طول بزرگترین زیر دنباله i-1,j و i,j-1 است .
- در نهایت مقدار خروجی برابر با مقدار درایه n,m از آرایه LCS است .

بزرگ ترین زیر رشته مشترک (LCS)–الگوریتم

- آرایه‌ی d یک آرایه‌ی دو بعدی به ابعاد $(m+1)\times(n+1)$ است که در آن n و m طول دو رشته ورودی اند. مقدار $d_{i,j}$ برابر طول بزرگ‌ترین زیردنباله‌ی مشترک بین i خانه‌ی اول از سری اول و j خانه‌ی اول سری دوم است. a_i و b_i را به ترتیب عنصر i ام دنباله‌ی اول و دوم در نظر بگیرید. پس جواب مسئله برابر $d_{n,m}$ است.
- مقدار اولیه: مقدار تمام خانه‌های $d_{i,0}$ و $d_{0,j}$ برابر صفر است. چون طول بزرگ‌ترین زیردنباله‌ی مشترک بین دنباله‌ی تهی با هر دنباله‌ی دیگری برابر صفر است.

بزرگ ترین زیر رشته مشترک (LCS)–الگوریتم

- برای به دست آوردن مقدار $d_{i,j}$ خود این بزرگ‌ترین زیردنباله‌ی مشترک (که طولش $d_{i,j}$ است) را در نظر بگیرید. این زیردنباله :
- یا هر دوی a_i و b_j را شامل می‌شود که این حالت فقط در صورتی می‌تواند اتفاق بیافتد که $a_i = b_j$ باشد. در این حالت جواب برابر $d_{i-1,j-1}+1$است.
- یا شامل a_i نیست. درنتیجه مقدارش برابر $d_{i-1,j}$است.
- یا شامل b_j نیست. درنتیجه مقدارش برابر $d_{i,j-1}$است.

بزرگ ترین زیر رشته مشترک (LCS)–الگوریتم

- پس اگر $a_i = b_j$:

$$d_{i,j} = \max(d_{i-1,j-1}+1, d_{i-1,j}, d_{i,j-1})$$

- در غیر اینصورت:

$$d_{i,j} = \max(d_{i-1,j}, d_{i,j-1})$$

بزرگ ترین زیر رشته مشترک (LCS)–الگوریتم

```
for i from 0 to n
        d[i][0] = 0
for j from 0 to m
        d[0][j] = 0
for i from 1 to n
        for j from 1 to m
                d[i][j] = max( d[i-1][j] , d[i][j-1] )
                if   a[i] == b[j]
                        d[i][j] = max( d[i][j] , d[i-1][j-1] + 1 )
```

بزرگ ترین زیر رشته مشترک (LCS)

تحلیل پیچیدگی فضا و زمان در حالت معمول برای الگوریتم برنامه نویسی پویا برای مسئله LCS

عمل اصلی: پیدا کردن حداکثر مقدار در داخلی ترین حلقه

اندازه ورودی : دو رشته به طول های n و m

$$T(n) = n^2 \in \theta(n^2)$$

فصل چهارم

روش حریصانه

مقدمه

- الگوریتم حریصانه ، به ترتیب عناصر را گرفته ، هر بار آن عنصری را که طبق ملاکی معین "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، همانند برنامه نویسی پویا غالبا برای حل مسائل بهینه سازی به کار می روند، ولی روش حریصانه صراحت بیشتری دارد.

مقدمه

- الگوریتم حریصانه با انجام یک سری انتخاب، که هر یک در لحظه ای خاص، بهترین به نظر می رسد عمل می کند، یعنی انتخاب در جای خود بهینه است. امید این است که یک حل بهینه سرتاسری یافت شود، ولی همواره چنین نیست.
- برای یک الگوریتم مفروض باید تعیین کرد که آیا حل همواره بهینه است یا خیر.

مقدمه

- در روش حریصانه، تقسیم به نمونه های کوچک تر صورت نمی پذیرد.
- 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.

مقدمه

- نتیجه نهایی مجموعه ای از داده ها است که ممکن است ترتیب آنها نیز اهمیت داشته باشد.
- مجموعه جواب به صورت مرحله ای است و در هر مرحله یک مولفه از جواب حاصل می شود.
- جواب نهایی باید تابع هدف را بهینه کند(ماکزیمم یا مینیمم)
- تصمیم نهایی در مورد انتخاب یا عدم انتخاب توسط روال Select جواب قطعی و غیر قابل بازگشت می باشد.

مقدمه

- هر دور تکرار، شامل مولفه های زیر است:

1. روال انتخاب(Select): عنصر بعدی را که باید به مجموعه اضافه شود،انتخاب می کند.انتخاب طبق یک ملاک حریصانه است.
2. بررسی امکان سنجی(Feasible): تعیین می کند که آیا مجموعه جدید برای رسیدن به حل،عملی است یا خیر.
3. بررسی راه حل(Solution): تعیین می کند که آیا مجموعه جدید ، حل نمونه را ارائه می کند یا خیر.
4. یک تابع هدف : هدف بهینه کردن این تابع است.

مقدمه

```
function greedy(A, n)
        solution←Φ
        for i ←1 to n do
                x← select(A)
                if feasible(x, solution) then
                        solution ← union(solution, x)
                endif
        repeat
        return solution
end.
```

خرد کردن پول

- فرض کنید می‌خواهیم مبلغ c واحد پول را با n سکه $a_1<a_2...<a_n$ خرد کنیم، چگونه می‌توانیم با استفاده از تعدادی دلخواه از سکه‌های $a_1...,a_n$ مبلغ c را تشکیل دهیم، به صورتی که تعداد سکه‌های استفاده شده تا حد امکان کم باشد؟

خرد کردن پول - الگوریتم حریصانه

- در آغاز هیچ سکه ای در مجموعه جواب نداریم.
- بزرگترین سکه(از لحاظ ارزش) را پیدا می کند. یعنی ملاک برای تعیین بهترین سکه(بهینه محلی) ارزش سکه است.[روال انتخاب]
- آیا با افزودن این سکه به بقیه پول، جمع کل آنها از مبلغ مورد نظر بیشتر می شود یا خیر؟[تحقیق عملی بودن]
- اگر می توان سکه را افزور، به مجموعه جواب اضافه می شود. سپس بررسی می شود آیا مبلغ مورد نظر تامین شده یا خیر؟ [تحقیق حل شدن]
- اگر پاسخ منفی است، با استفاده از روال انتخاب سکه دیگری انتخاب و فرآیند تکرار می گردد.

خرد کردن پول – الگوریتم حریصانه

```
While (there are more coins and the instance is not solved){
  Grab the largest remaining coin; // solution procedure
  if(adding the coin makes the change exceed the amount owed)
     reject the coin;    // feasibility check
  else add coin to the change;
  if(the total value of the change equals the amount owed)   //solution check
     the instance is solved;
}
```

خرد کردن پول - مثال الگوریتم حریصانه

- فرض کنید فروشنده سکه های زیر را دارد:

سکه ۲۵ تومانی	۱ عدد
سکه ۱۰ تومانی	۲ عدد
سکه ۵ تومانی	۱ عدد
سکه ۱ تومانی	۲ عدد

- حال می خواهد باقی مانده پول مشتری که ۳۶ تومان است را بپردازد.

خرد کردن پول - مثال الگوریتم حریصانه

- برای حل مسئله به روش حریصانه چنین عمل می کنیم:

1. یک سکه ۲۵ تومانی می گیریم، کمتر از ۳۶ است پس به مجموعه جواب اضافه می کنیم.
2. یک سکه ۱۰ تومانی می گیریم، مجموع ۳۵ می شود و کمتر از ۳۶، پس به مجموعه جواب اضافه می کنیم.
3. سکه ۱۰ تومانی دوم را می گیریم، چون مجموع ۴۵ می شود و از ۳۶ بزرگتر است، آن را کنار می گذاریم.
4. سکه ۵ تومانی را می گیریم، چون مجموع ۴۰ می شود و از ۳۶ بزرگتر است، آن را کنار می گذاریم.
5. سکه ۱ تومانی را می گیریم، چون مجموع ۳۶ می شود و بیشتر از ۳۶ نیست، پس آن به مجموعه جواب اضافه می کنیم. مجموع سکه ها برابر با میزان لازم است، حل مسئله پایان می یابد.

درخت پوشای کمینه– مقدمه

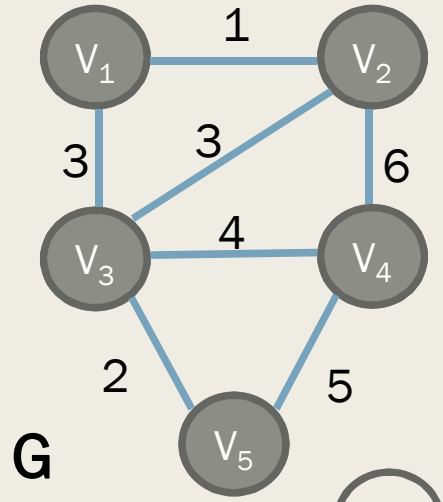
- در درس ساختمان داده ها خوانده اید که درخت یک گراف بدون جهت، متصل و بی چرخه است.
- به عبارتی دیگر درخت یک گراف بدون جهت است که در آن بین هر جفت از رئوس فقط و فقط یک مسیر وجود دارد.
- توجه کنید در این تعریف هیچ گره ای را به عنون ریشه در نظر نمی گیریم ولی در درخت ریشه دار (rooted tree) یکی از راس ها ریشه می باشد و اغلب منظور از درخت، درخت ریشه دار است. البته در این قسمت فرض می کنیم با درخت هایی سر و کار داریم که ریشه آن ها برای ما مهم نیست .

درخت پوشای کمینه– مقدمه

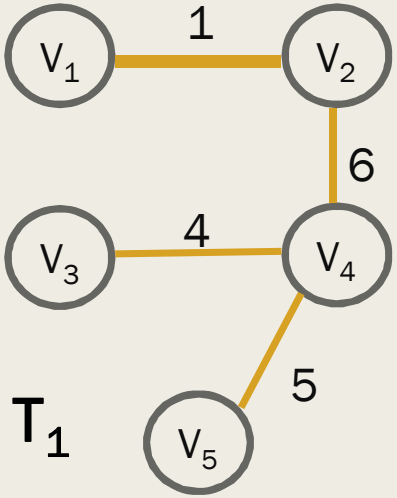
- درخت پوشا (spanning tree) برای یک گراف G، یک زیر گراف متصل می باشد که حاوی همه راس های گراف G بوده و همچنین یک درخت است به عبارت دیگر درخت پوشا شامل همه رئوس و برخی یال های گراف است به نحوی که متصل بوده و چرخه نیز ندارد.

درخت پوشای کمینه– مثال

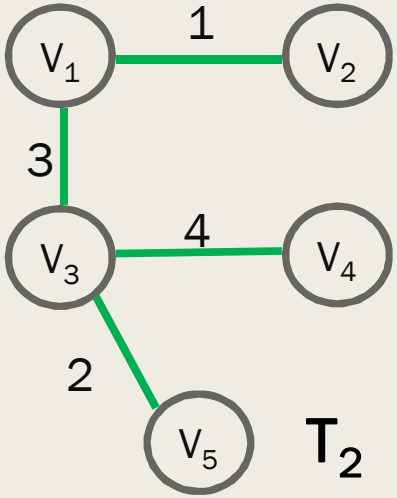
- مثلاً برای گراف بدون جهت وزن دار G سه درخت پوشا T_1، T_2 و T_3 رسم کرده ایم.



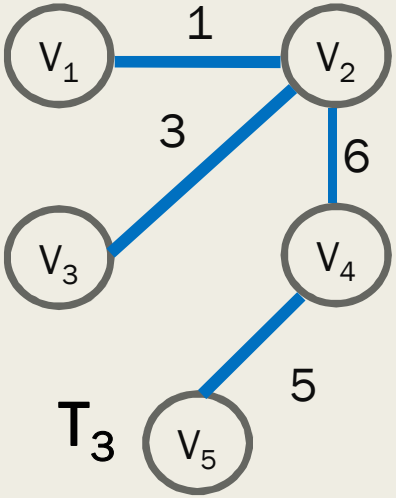
G



T_1



T_2



T_3

درخت پوشای کمینه– مثال

- همان طور که دیدید یک گراف ممکن است چندین درخت پوشا داشته باشد. گراف G درخت های پوشای بیشتری از آن چه رسم کرده ایم دارد . وزن کلی درخت های T_1، T_2 و T_3 عبارت اند از :

$T_1=1+4+5+6=16$

$T_2=1+2+3+4=10$

$T_3=1+3+5+6=15$

درخت پوشای کمینه– مثال

- همان طور که می دانید یک گراف بدون جهت را می توان به صورت دو مجموعه E و V نشان داد G=(V , E) که V مجموعه رئوس و E مجموعه یال ها می باشند . مثلاً برای گراف قبلی داریم :

$V=\{v_1,v_2,v_3,v_4 ,v_5\}$

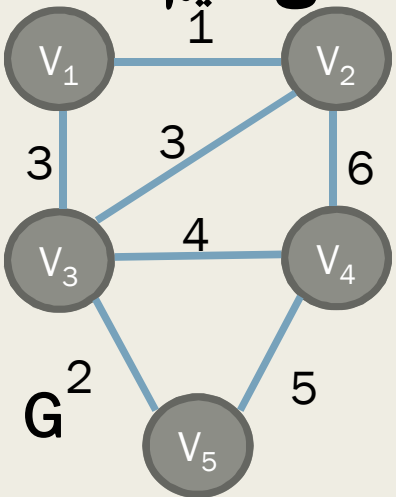
$E=\{(v_1,v_2),(v_1,v_3),(v_2,v_3) ,(v_2,v_4) ,(v_3,v_4) ,(v_3,v_5) ,(v_4,v_5)\}$

- در گراف بدون جهت (V_2,V_1) با (V_1,V_2) یکسان است ولی قرارداد می کنیم که هنگام نوشتن ابتدا راسی که اندیس کوچکتر دارد بیاوریم. درخت پوشای T برای گراف G= (V , E) شامل همه رئوس V می باشد ولی برخی از یال های Eرا دارد که آنها را با Fنمایش می دهیم . پس:

$T = (V , F) , F \in \dot{\subseteq} E$

درخت پوشای کمینه– الگوریتم پریم

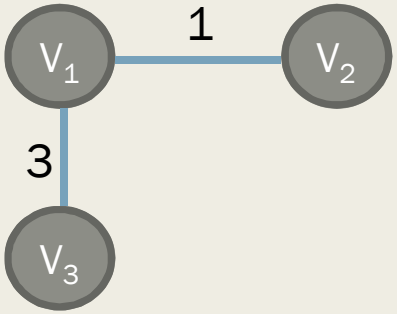
- در الگوریتم پریم(prime) ابتدا مجموعه یال های F را برابر تهی و مجموعه رئوس Y را نیز برابر یک راس دلخواه قرار می دهیم. سپس نزدیک ترین راس به Y را از مجموعه V-Y انتخاب می کنیم. بدیهی است که این راس توسط یالی با کمترین وزن به راسی از مجموعه Y متصل است. مثلاً در گراف G v_2 را انتخاب کرده که کمترین وزن (برابر ۱) را دارد . بدین ترتیب v_2 را به Y و (v_1,v_2) را به F اضافه می کنیم:



$F=\{(v_1,v_2)\}$, $Y=\{v_1,v_2\}$

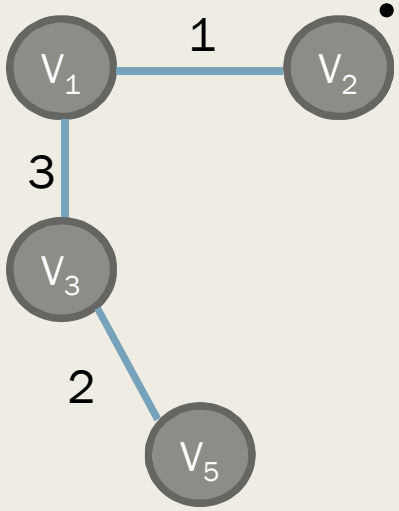
درخت پوشای کمینه– الگوریتم پریم

- در قدم بعدی v_3 را به مجموعه Y و (v_1,v_3) را به F اضافه می کنیم:



$$F=\{(v_1,v_2),(v_1,v_3)\}\ ,\ Y=\{v_1,v_2,v_3\}$$

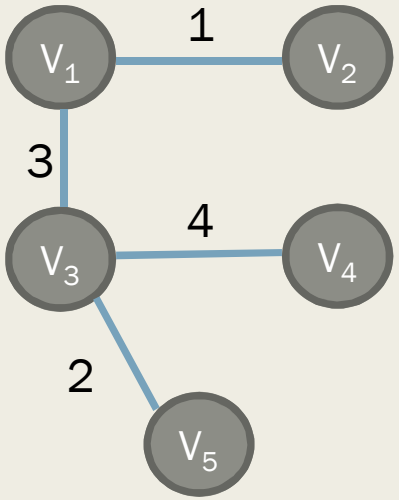
- این عملیات افزودن نزدیک ترین رئوس را آن قدر تکرار می کنیم تا Y=V شود. البته عدم ایجاد چرخه نیز بررسی می گردد :



در این مرحله فاصله $V-Y=\{v_4,v_5\}$ را با گره های موجود در $Y=\{v_1,v_2,v_3\}$ مقایسه کردیم و دیدیم که فاصله v_5 با گره v_3 از همه کمتر است لذا v_5 را به Y و (v_3,v_5) را به F اضافه کردیم .

درخت پوشای کمینه– الگوریتم پریم

- در مرحله آخر فاصله v_4 را با گره های موجود در $Y=\{v_1,v_2,v_3,v_5\}$
- مقایسه کرده و گره v_4 را به Y و (v_3,v_4) را به F اضافه می کنیم و درخت پوشای کمینه به دست می آید.



درخت پوشای کمینه– الگوریتم پریم

F=Φ

Y={v_1};

While (the instance is not solved)

{

select a vertex in V-Y that is nearest to Y; //selection procedure and feasibility check

add the vertex to Y;

add the edge to F;

if (Y==V) the instance is solved; //solution check

}

درخت پوشای کمینه– الگوریتم پریم

- البته در هنگام انتخاب راس از V- Y باید دقت کرد که حلقه ایجاد نشود.

نکته: در یک گراف کامل K_n با n راس به تعداد n^{n-2} درخت پوشا وجود دارد .

- از آنجا که در الگوریتم پریم در هر مرحل فاصله هر گره با گره های قبلی مقایسه می شود پس بدیهی است که از مرتبه $\theta(n^2)$ می باشد که n تعداد رئوس گراف است.

$\theta(n^2)$:مرتبه اجرای الگوریتم پریم

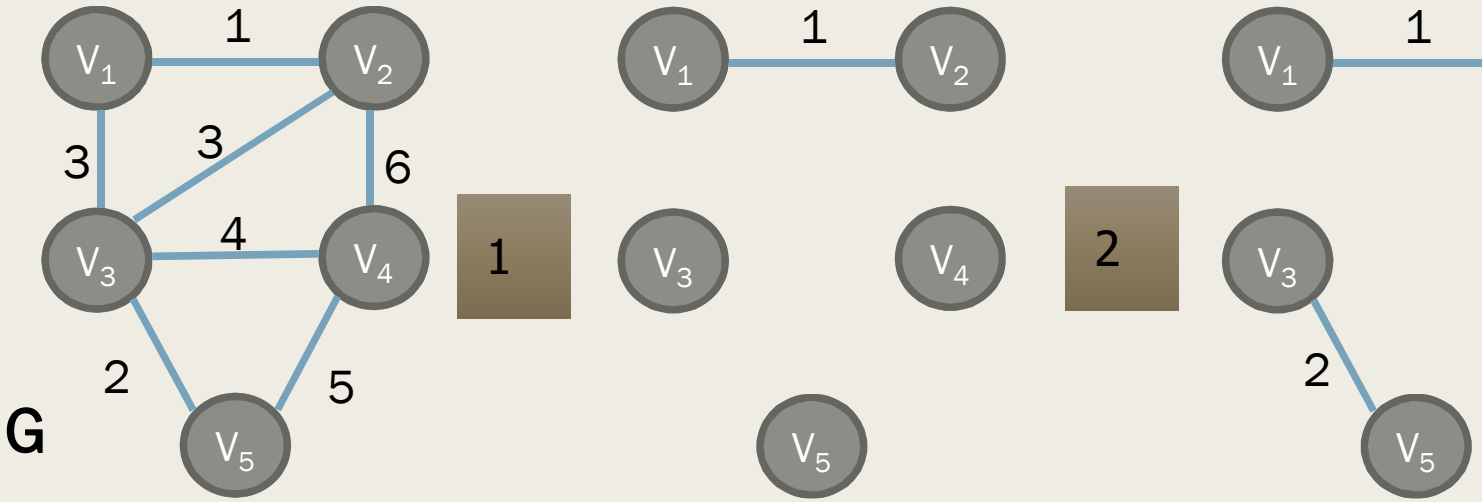
- هر چند که الگوریتم های حریصانه اغلب ساده تر از الگوریتم های پویا هستند ولی معمـــولاً تعییـــن اینکه آیا این الگوریتم حریصانه همواره جواب بهینه را می دهد دشوار بوده و نیاز به اثبات رسمی دارد . اثبات می شود که الگوریتم پریم همواره یک درخت پوشای کمینه را تولید می کند.

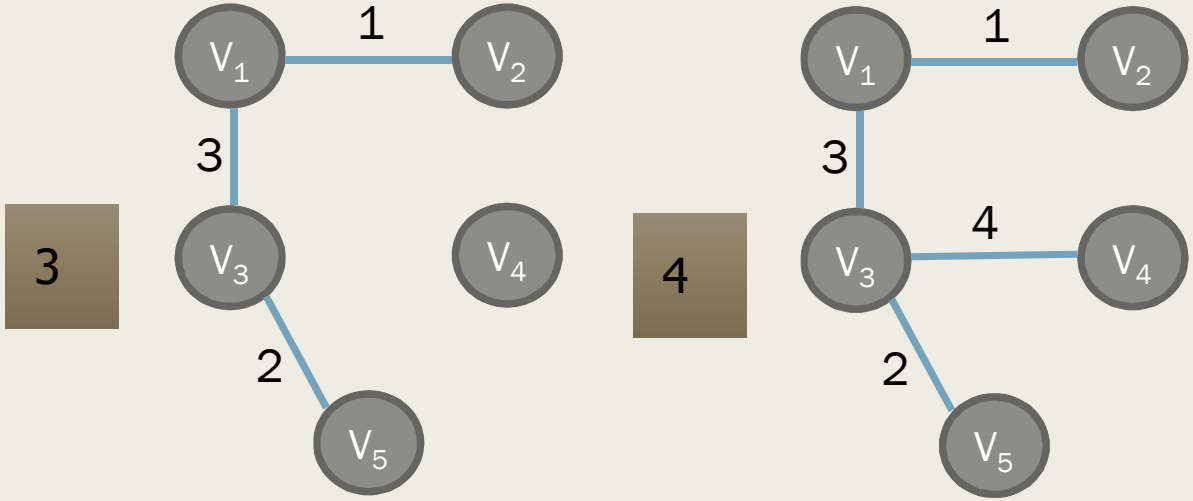
درخت پوشای کمینه– الگوریتم کراسکال(kruskal)

- این الگوریتم نیز مشابه الگوریتم پریم برای یافتن درخت پوشای کمینه یک گراف به کار می رود. در این الگوریتم ابتدا یال ها از کمترین وزن به بیشترین وزن مرتب می شوند سپس یال ها به ترتیب انتخاب شده و اگر یالی ایجاد حلقه کند کنار گذاشته می شود .
- عملیات هنگامی خاتمه می یابد که تمام راس ها به هم وصل شوند یا این که تعداد یال های موجود در F برابر n-1 شود که تعداد راس ها است.
- در برخی کتاب ها این روش با نام راشال مطرح شده است .

درخت پوشای کمینه– الگوریتم کراسکال(kruskal)

- مراحل الگوریتم برای یک گراف G





درخت پوشای کمینه– الگوریتم کراسکال(kruskal)

- بیشتر زمان در الگوریتم کراسکال مربوط به مرتب سازی یال هاست پس اگر تعداد یال e باشد زمان این الگوریتم از مرتبه θ(elge)خوهد بود.
- ممکن است به نظر برسد که این زمان بستگی به n (تعداد رئوس) ندارد. ولی همان طور که می دانید در بهترین حالت تعداد یال ها برابر n-1 و در بدترین حالت که گراف کامل باشد و بین هر دو راس یک مسیر مستقیم داشته باشیم تعداد یال ها برابر $\frac{n(n-1)}{2}$ خواهد بود یعنی :

$$n-1 \leq e \leq \frac{n(n-1)}{2}$$

درخت پوشای کمینه– الگوریتم کراسکال(kruskal)

- پس اگر e نزدیک به کران پایین باشد یعنی گراف نسبتاً خلوت بوده و یال کمی داشته باشد، الگوریتم کراسکال از مرتبه رو به رو است :

$$\theta(elge) = \theta(nlgn)$$

- و اگر e نزدیک به کران بالا باشد یعنی گراف نسبتاً پر باشد و یال های زیادی داشته باشد :

$$\theta(elge) = \theta(n^2lgn^2)=\theta(n^2.2lgn) = \theta(n^2lgn)$$

- خلاصه آنکه

پیچیدگی الگوریتم پریم=$\theta(n^2)$

پیچیدگی الگوریتم کراسکال= $\theta(elge) = \begin{cases} \theta(nlgn) & \text{برای گراف خلوت} \\ \theta(n^2lgn) & \text{برای گراف شلوغ} \end{cases}$

- پس اگر گرافی یال های کمی دارد بهتر است از روش کراسکال و اگر یال های زیادی دارد بهتر است از روش پریم استفاده کنیم.

درخت پوشای کمینه– الگوریتم کراسکال(kruskal)

- می توان اثبات کرد که الگوریتم کراسکال همواره یک درخت پوشای کمینه ایجاد می کند.

```
sort the edges in  by weight in ascending order;
F=Φ
While (number of edges in F is less than n-1){
   e = edge with least weight noy yet considered;
   i,j= indices of vertices connected by e;
   p = find(i);
   q= find(j)
   if(!equal(p,q){
      merge(p,q)
      adde to F;
   }
}
```

یافتن کوتاهترین مسیر–الگوریتم دایکسترا(Dijkstra)

- در روش برنامه نویسی پویا الگوریتم فلوید با مرتبه زمانی $\theta(n^3)$ برای یافتن کوتاهترین مسیر از هر راس به همه رئوس دیگر در یک گراف موزون و بدون جهت ارائه دادیم
- حال از روش حریصانه استفاده کرده، یک الگوریتم $\theta(n^3)$ برای مسئله طراحی می کنیم که دایکسترا نام دارد. فرض الگوریتم این است که از راس مورد نظر به هر یک از رئوس دیگر مسیری وجود داشته باشد.
- این الگوریتم همانند الگوریتم پریم (برای تعیین و ایجاد درخت پوشای کمینه) می باشد.

یافتن کوتاهترین مسیر –الگوریتم دایکسترا

- برای مقداردهی اولیه به مجموعه Y راسی که باید کوتاهترین مسیرها از آن تعیین شوند را قرار می دهیم و آن راس را v_1 در نظر می گیریم.
- به مجموعه F مقدار اولیه تهی را می دهیم. نخست یک راس v را انتخاب می کنیم که از همه به v_1 نزدیک تر است و آن را به Y و $<v, v_1>$ را به F اضافه می کنیم. واضح است که آن یال کوتاه ترین مسیر میان v_1، v است. سپس مسیرهایی از v_1 به رئوس موجود در V-Y را چک می کنیم که فقط رئوس موجود در Y را به عنوان رئوس مجاز می شمارند. یکی از این مسیرها، کوتاهترین مسیر است، راسی که در انتهای چنین مسیری باشد به Y و یالی که آن راس را در بر دارد به F افزوده می شود. این روال ادامه می یابد تا Y با V برابرشود. در انتها F حاوی یال های موجود در کوتاهترین مسیر است.

یافتن کوتاهترین مسیر –الگوریتم دایکسترا

F=Φ

Y={v_1};

While (the instance is not solved)

{

  select a vertex in v from V-Y that that has a ; //selection procedure

  shortest path from v_1, using only vertices; //and feasibility check

  in Y as intermediates;

  add the new vertex v to Y;

  add the edge(on theshortest path) that touches v to F;

  if (Y==V)

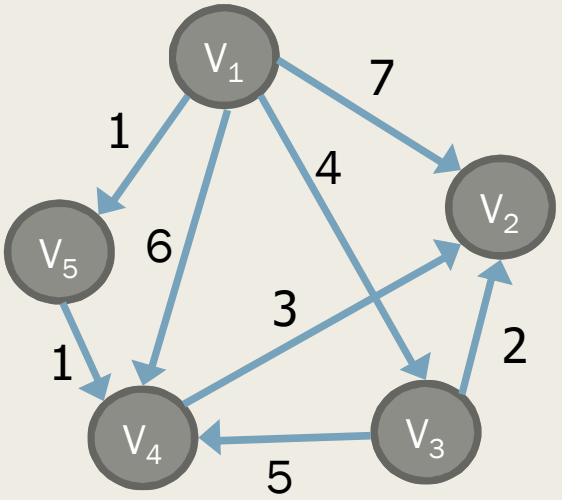
    the instance is solved; //solution check

}

یافتن کوتاهترین مسیر –الگوریتم دایکسترا

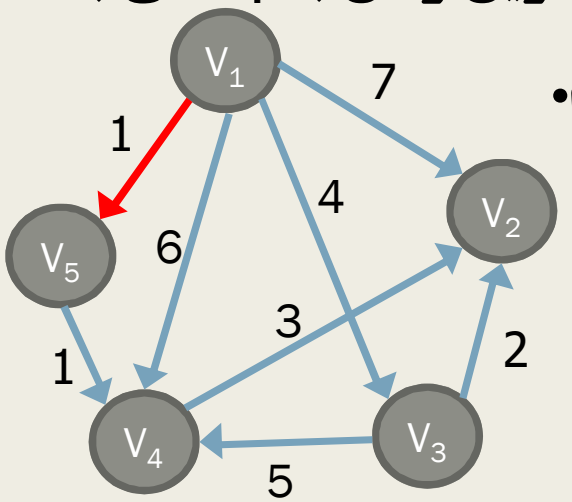
- مراحل الگوریتم برای یک گراف جهت دار G

1. محاسبه کوتاه ترین مسیر از v_1



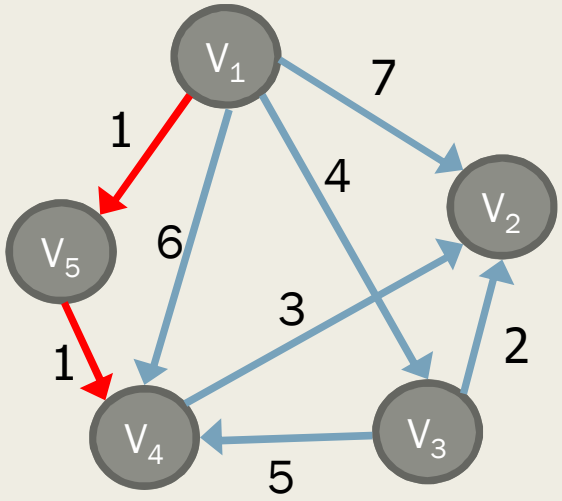
$F=\Phi$, $Y=\{v_1\}$

2. ابتدا راس v_5 را انتخاب می کنیم زیرا نزدیک ترین راس به v_1 می باشد و v_5 را به Y و $<v_1, v_5>$را به F اضافه ی کنیم.



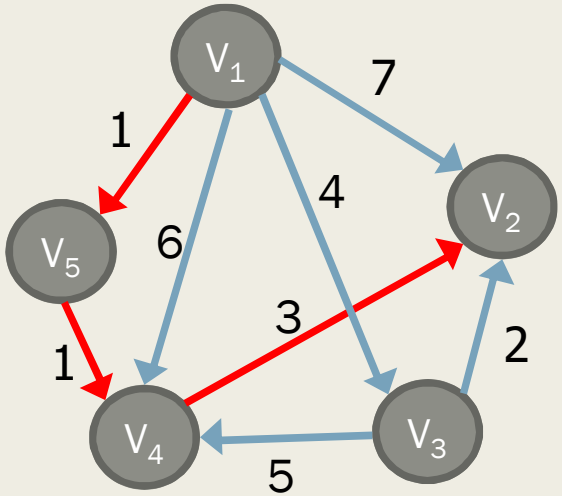
$F=\{<v_1, v_5>\}$, $Y=\{v_1, v_5\}$

یافتن کوتاهترین مسیر –الگوریتم دایکسترا



3. حال با در نظر گرفتن گره v_5 به عنوان واسطه گره v_4 را انتخاب می کنیم چرا که v_4 کوتاه ترین مسیر تا v_1 را (به کمک گره v_5) دارد.

$F=\{<v_1, v_5>, <v_5, v_4>\}$,
$Y=\{v_1, v_5, v_4\}$

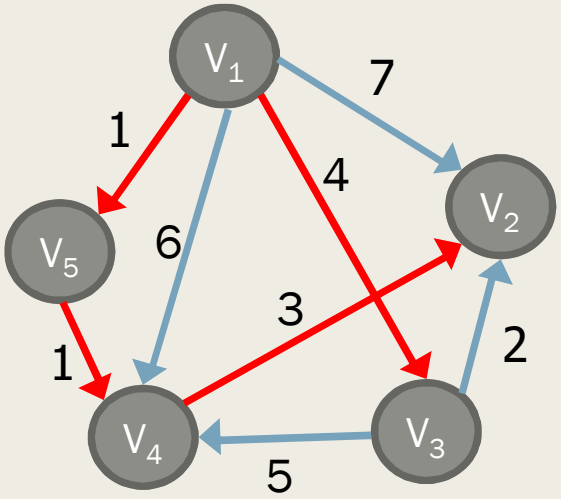


4. راس v_2 را انتخاب می کنیم چرا که با واسطه گری $\{v_5, v_4\}$ کوتاه ترین مسیر از v_1 را دارد.

$F=\{<v_1, v_5>, <v_5, v_4>, <v_4, v_2>\}$,
$Y=\{v_1, v_5, v_4, v_2\}$

یافتن کوتاهترین مسیر –الگوریتم دایکسترا

5. در آخر نیز v_3 را انتخاب می کنیم و می بینیم که کوتاه ترین مسیر از v_1 به v_3 مسیر $[v_1,v_3]$ است. لذا v_3 را به Y و آخرین یال مسیر مذکور یعنی$<v_1,v_3>$ را به F اضافه می کنیم.



$F=\{<v_1,v_5>, <v_5,v_4>, <v_4,v_2>, <v_1,v_3>\}$,
$Y=\{v_1,v_5,v_4,v_2,v_3\}$

الگوریتم کد هافمن

- فرض می کنید می خواهید n عنصر اطلاعاتی $A_1, A_2,\ldots,A_n$ را به وسیله رشته هایی از بیت ها به صورت کد درآوریم. یک راه ساده برای این منظور آن است که هر عنصر را به وسیله یک رشته r بیتی کد گذاری کنیم که در آن $2^{r-1} < n < 2^r$
- در این حالت طول کد همه عناصر با هم یکسان است
- مثال : برای کد گذاری ۴۸ کاراکتر حداقل به چند بیت نیاز داریم ؟

$$32 = 2^5 < 48 \leq 2^6 = 64 \rightarrow r = 6$$

- پس حداقل به ۶ بیت نیاز داریم .

الگوریتم کد هافمن

- حال فرض کنید عنصر های اطلاعاتی با احتمال مساوی اتفاق نمی افتد آنگاه می توان با استفاده از رشته های با طول متغیر در مصرف حافظه صرفه جویی کرد.
- به این ترتیب که عناصری که اغلب ظاهر می شوند با رشته های کوتاه تر و عناصری که به ندرت ظاهر می شوند با رشته های بزرگ تر نشان داده شوند. برای پیدا کردن این کدهای با طول متغیر می توان از الگوریتم هافمن استفاده کرد.
- الگوریتم هافمن را با مثال شرح می دهیم.

الگوریتم کد هافمن

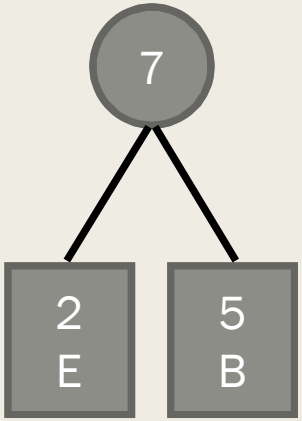
- فرض کنید A,H,G,F,E,D,C,B,A عنصر اطلاعاتی با وزن های مشخص شده هستند.

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

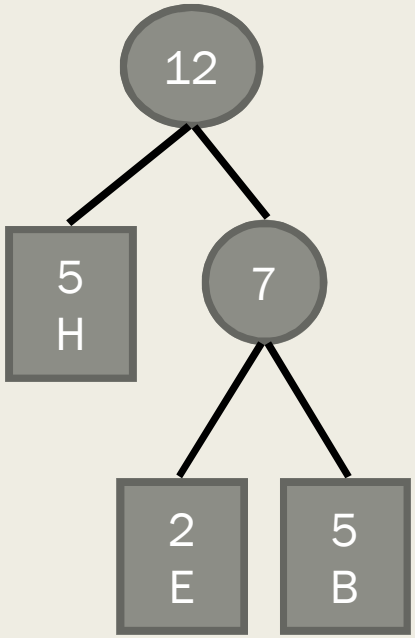
- می خواهیم درخت با حداقل طول مسیر وزن داده شده را با استفاده از اطلاعات بالا و الگوریتم هافمن ترسیم کنیم.
- شکل زیر مراحل کار را نشان می دهد . در هر مرحله دو درختی که ریشه مینیمم دارند را با هم ترکیب می کنیم (وزن آن ها را با هم جمع می کنیم) . گره با وزن کمتر سمت چپ قرار می گیرد.

الگوریتم کد هافمن

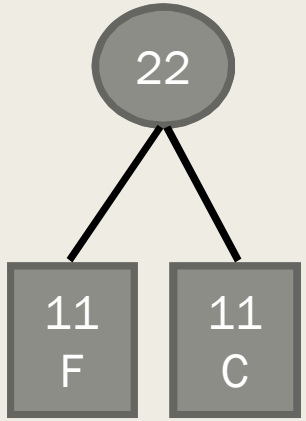
22	5	11	19	2	11	25	5
A	B	C	D	E	F	G	H



مرحله اول

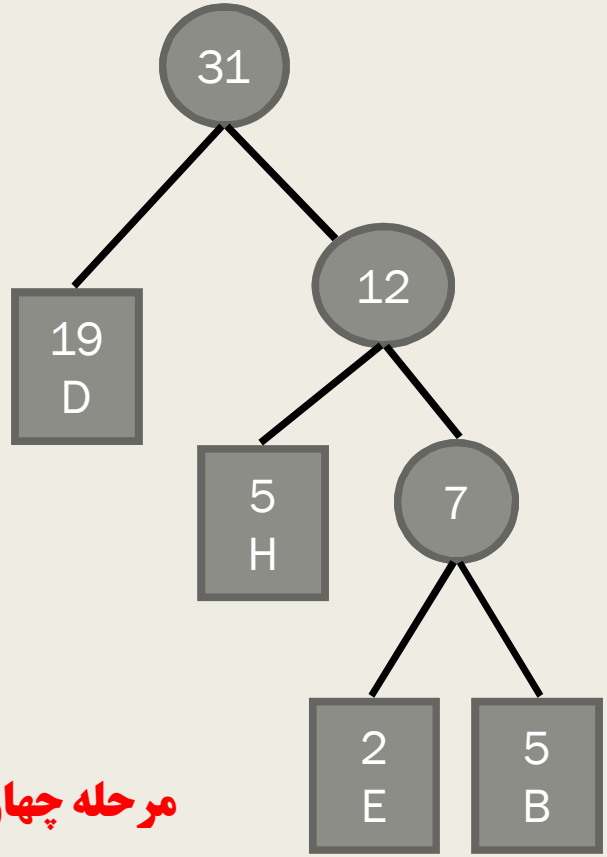


مرحله دوم

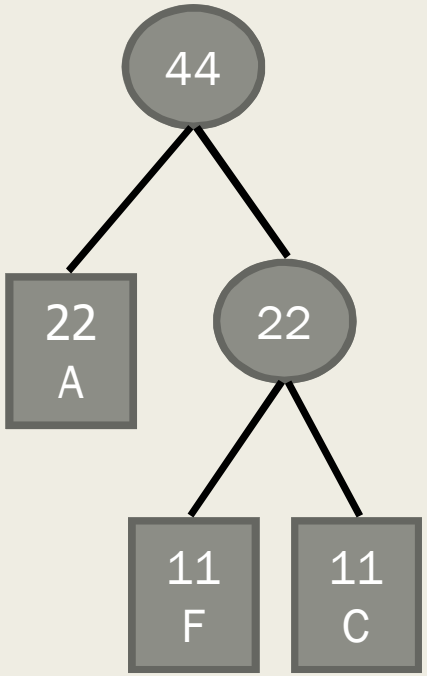


مرحله سوم

الگوریتم کد هافمن

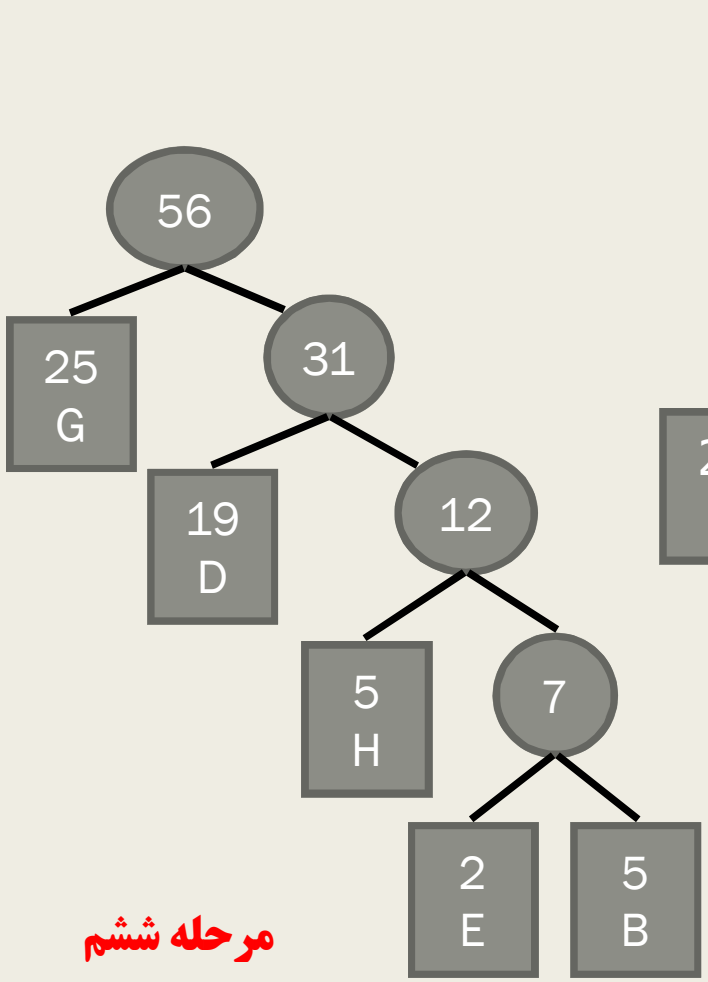


مرحله چهارم

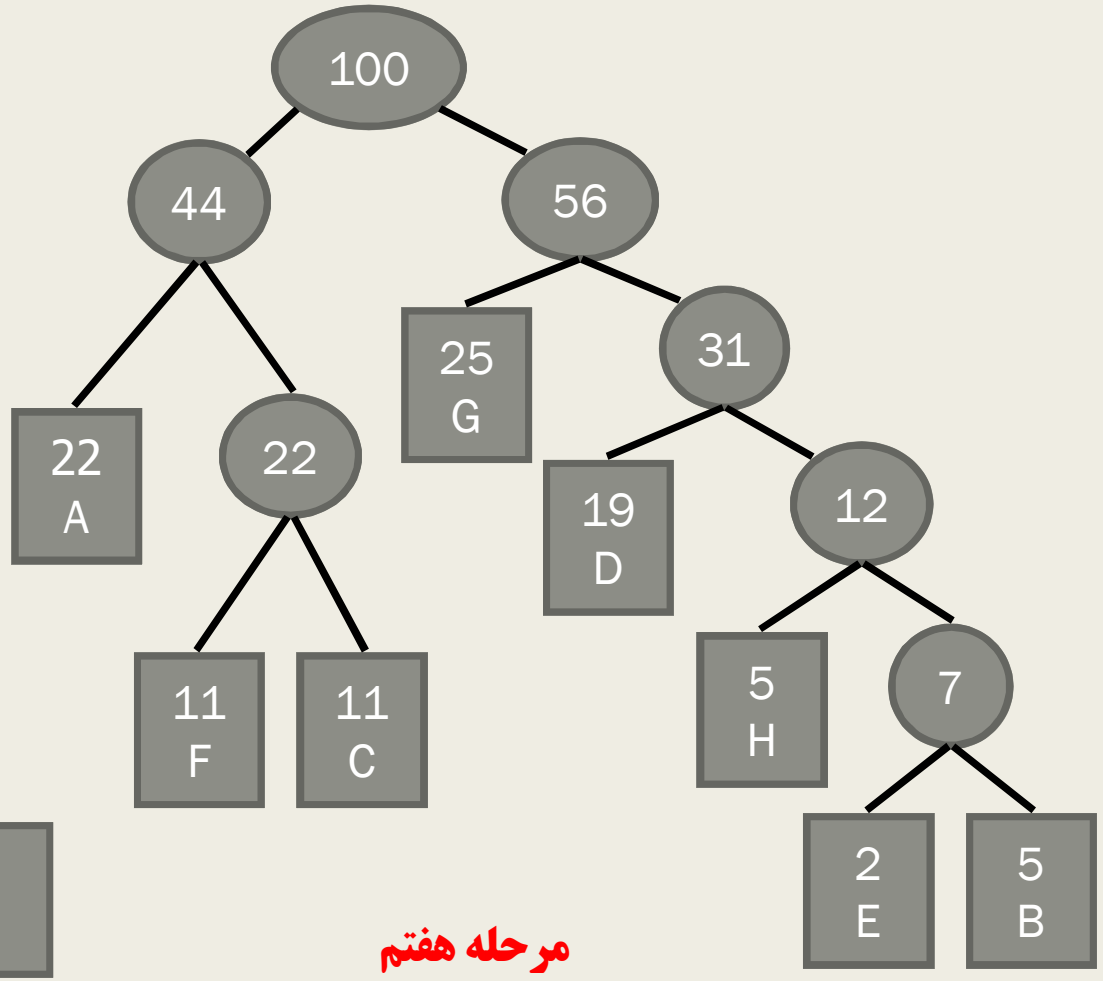


مرحله پنجم

الگوریتم کد هافمن

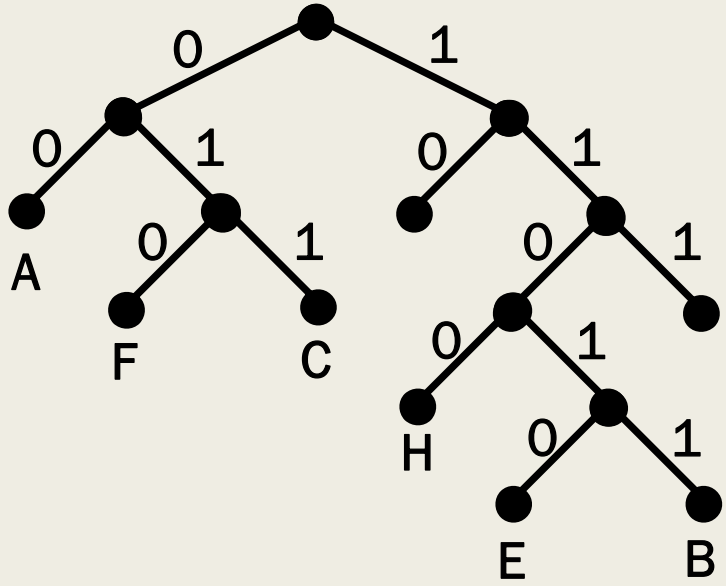


مرحله ششم



مرحله هفتم

الگوریتم کد هافمن



A = 00
B = 11011
C = 011
D = 111
E = 11010
F = 010
G = 10
H = 1100

الگوریتم کد هافمن

پیچیدگی زمانی: O(n lg n)

```
HUFFMAN(C)
  n ← |C|
  Q ← C      //O(lg n)
  for i 1 to n - 1
      do allocate a new node z
        left[z] ← x ← EXTRACT-MIN (Q)        //O(lg n)
        right[z] ← y ← EXTRACT-MIN (Q)       //O(lg n)
        f [z] ← f [x] + f [y]
        INSERT(Q, z)         //O(lg n)
  return EXTRACT-MIN(Q)   //Return the root of the tree.
```

مسئله کوله پشتی(knapsack)

- دزدی با کوله پشتی خود وارد جواهر فروشی می شود . کوله پشتی او تحمل حداکثر وزن M را دارد و بیشتر از آن را پاره می شود.
- هر قطعه جواهر وزن (Wi) و ارزش (Pi) معینی دارد. مساله انتخاب قطعاتی است که حداکثر ارزش را داشته باشد و نیز جمع وزن آنها از حداکثر M بیشتر نشود.
- یک راه حل ساده و غیر هوشمندانه آن است که همه زیر مجموعه های ممکن از n قطعه را در نظر بگیریم و با مقایسه کردن همه آنها حالتی را در نظر بگیریم که بیشترین ارزش را داشته و جمع وزن آن ها M باشد. از آنجا که تعداد این زیر مجموعه ها 2^n است لذا این الگوریتم از مرتبه نمایی است و ما دنبال راه حل سریع تری هستیم.

مسئله کوله پشتی

- ممکن است به نظر برسد یک راه حل حریصانه آن است که قطعاتی با ارزش بیشتر را زودتر برداریم تا هنگامی که جمع وزن آن ها به M برسد. ولی با مثال های ساده ای می توان نشان داد که این روش غلط است. مثلاً سه قطعه با وزن و ارزش های زیر را در نظر بگیرید :

قطعه	1	2	3
وزن قطعه = W_i	25	10	11
ارزش قطعه = P_i	10	9	8

فرض کنید حداکثر M=30 باشد. در این حالت دزد فقط قطعه اول را برمی دارد که ارزش ۱۰ دارد. در حالی که اگر قطعات دوم و سوم را برمی داشت ارزش ۱۷ به دست می آمد.

مسئله کوله پشتی

- یک راه حل حریصانه دوم آن است که قطعاتی با وزن کمتر را زودتر برداریم تا هنگامی که جمع وزن آن ها به M برسد. ولی با مثال های ساده ای می توان نشان داد که این روش نیز مناسب نیست. مثلاً سه قطعه با وزن و ارزش های زیر را در نظر بگیرید:

قطعه	1	2	3
W_i = وزن قطعه	9	15	21
P_i = ارزش قطعه	2	9	18

فرض کنید حداکثر M=30 باشد. در این حالت دزد قطعات اول و دوم را برمی دارد که ارزش ۱۱ دارند. در حالی که اگر قطعه اول را برمی داشت ارزش ۱۸ به دست می آمد.

مسئله کوله پشتی

- یک راه حل حریصانه پیچیده تر آن است که قطعاتی با بیشترین ارزش به ازای واحد وزن را زودتر برداریم تا هنگامی که جمع وزن آن ها به M برسد.مثلاً سه قطعه با وزن و ارزش های زیر را در نظر بگیرید:

قطعه	1	2	3
W_i = وزن قطعه	5	10	20
P_i = ارزش قطعه	50	60	140
P_i/W_i= نسبت ارزش به وزن قطعه	10	6	7

فرض کنید حداکثر M=30 باشد. در این حالت قطعات اول و سوم انتخاب می شود که ارزش ۱۹۰ دارند. البته بهینه ترین حالت انتخاب قطعات دوم و سوم با ارزش ۲۰۰ است.

مسئله کوله پشتی

در حالت کلی تر که امکان انتخاب کسری از قطعات نیز وجود داشته باشد داریم:

مساله: حداکثر نمودن سود حاصل از انتخاب اجناس $(Max \sum_{i=1}^{n} x_i.p_i)$

اگر کسر P_i **از شی** i **انتخاب شود سود حاصل ارزش** $x_i.p_i$ **کسب می شود**

ورودی:

- **n: تعداد کیسه ها**
- P_i**: سود حاصل از انتخاب کل قطعه** i **ام**
- W_i**: وزن کل قطعه** i **ام**
- **M: گنجایش کوله پشتی**

مسئله کوله پشتی

شرایط مساله: وزن همه کیسه ها روی هم از وزن کوله پشتی بیشتر است زیرا اگر کمتر باشد می توان همه را برداشت انتخابی وجود ندارد ($\sum_{i=1}^{n} w_i > M$). همچنین مجموع وزن اجسام انتخابی نباید از ظرفیت وزنی کوله پشتی بیشتر شود($\sum_{i=1}^{n} x_i.w_i > M$).

خروجی:

کسری از کیسه iام که انتخاب شده است داخل عنصر iام آرایه جواب قرار می گیرد.

مسئله کوله پشتی

```
Void greedy(w,n) {
        Sort(p,w);          // (pi/wi)>=(pi+1/wi+1)
        For(i=0;i<n;i++)
                X[i]=0;
                U=w;
                For(i=0;i<n;i++) {
                        if(w[i]>u)
                                break;
                        x[i]=1; u=u-w[i];
                }
                If(i<n)
                        x[i]=u/w[i];
}
```

پیچیدگی زمانی: O(n lg n)

مسئله کوله پشتی – مثال

- سه قطعه با وزن و ارزش های زیر را در نظر بگیرید:

قطعه	1	2	3
وزن قطعه = W_i	5	10	20
ارزش قطعه = P_i	50	60	140
نسبت ارزش به وزن قطعه = P_i/W_i	10	6	7

اگر ظرفیت برابر ۳۰ باشد(M=30) و اینکه بتوان بخشی از یک قطعه را برداشت، بر اساس روش حریصانه، قطعات اول، سوم و نیمی از قطعه دوم با ارزش ۲۲۰ انتخاب می شوند.