به نام خالق زیبایی ها

تمرین سری چهـارم طراحی الگـوریتم ها – استاد مرضیه رحیمی

مصطفی فضلی شهری – 9822803

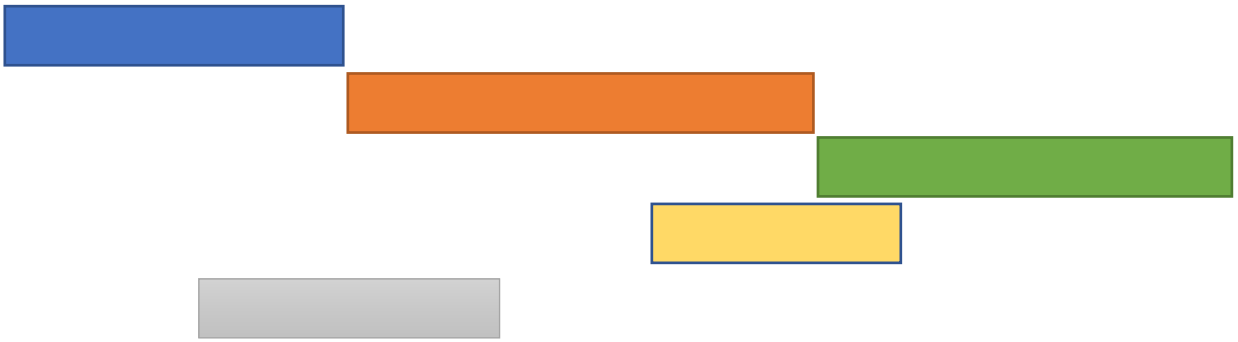
الگوریتم های حریصانه Greedy Algorithms :

1 - برای هریک از روش‌های حریصانه زیر، یک مثال نقض ارئه دهید که نشان دهد، نمی‌تواند پاسخ بهینه را برای مسئله انتخاب فعالیت‌ها بیابد .

- انتخاب کوتاه‌ترین فعالیت سازگار در هر مرحله

اگر فرآیند های بلند ولی سازگار داشته باشیم که پشت سر هم و بدون وقفه اجرا شوند و بهینه ترین حالت برای انجام یک فعالیت باشند، اما فعالیت های کوتاه تنها سازگاری داشته باشند و بهینه نباشند، آنگاه استفاده از کوتاه ترین فرآیند های سازگار جواب بهینه ای برای این حالت نمی باشد.

برای توصیف بهتر می توان نمودار فرآیندی مانند نمودار زیر مثال زد :

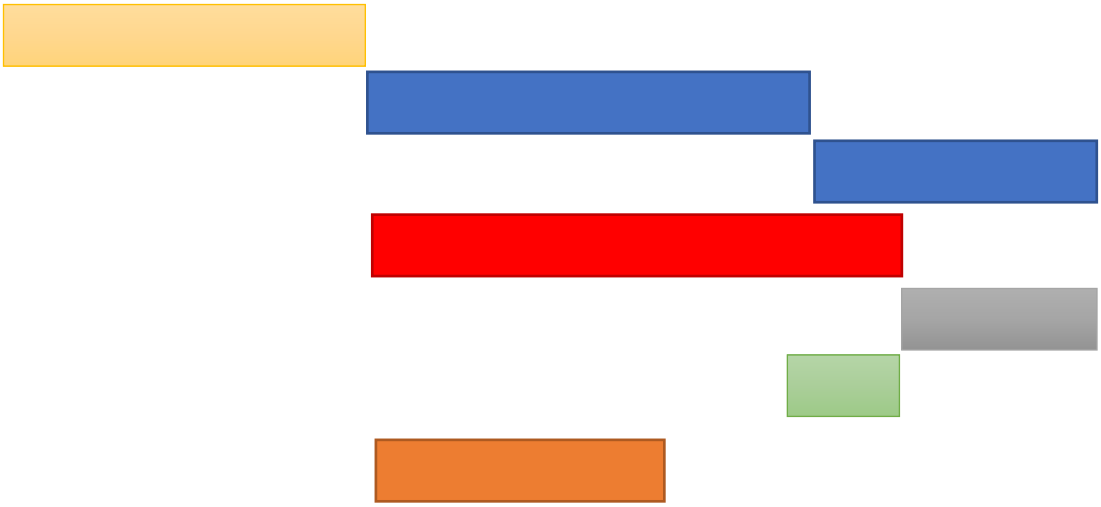


در این نمودار رسم شده بهینه ترین حالت به ترتیب استفاده از نمودار های آبی، نارنجی، سبز است که کوتاه ترین فرآیند ها نمی باشند.

- انتخاب فعالیت سازگاری که کمترین تعداد همپوشانی را با فعالیتهای باقیمانده دارد.

اگر در ابتدا فعالیت های سازگار را انتخاب کنیم و سپس تعداد همپوشانی را با بقیه فعالیت ها مقایسه کنیم و مجبور باشیم فعالیتی را انتخاب کنیم که کمترین تعداد همپوشانی را بقیه فرآیندها را داشته باشد، به تداخل برمیخوریم؛ زیرا که ممکن است فعالیت های سازگار و بهینه لزوما کمترین همپوشانی را با بقیه فرآیند ها نداشته باشد.

برای توصیف بهتر می توان نمودار زیر را مثال زد:

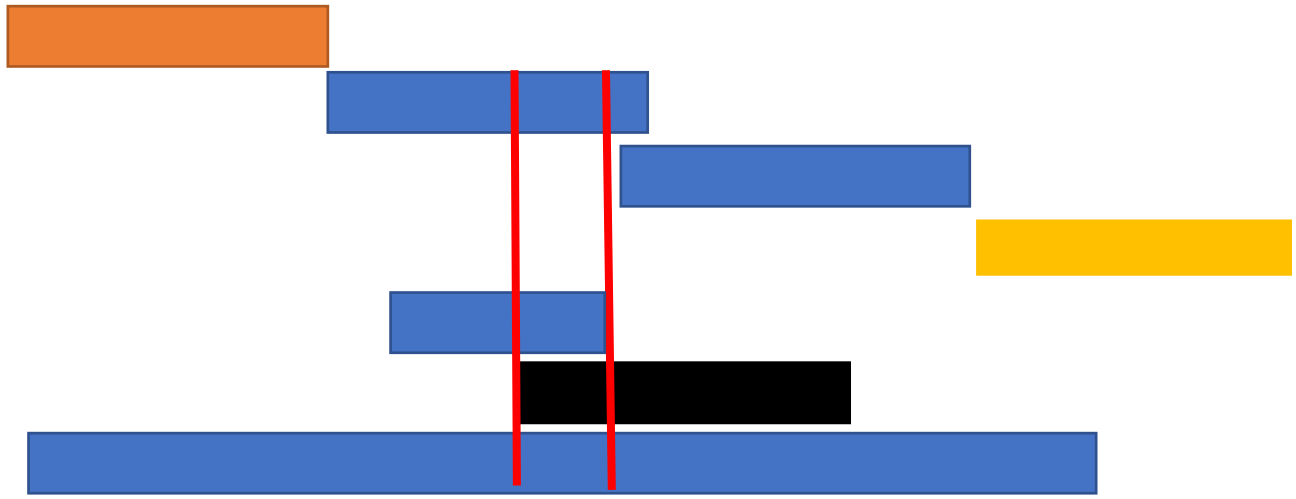


در این نمودار که رسم شده است با توجه به متن سوال، بهینه ترین حالت به ترتیب نموداری با شکل های زرد، نارنجی، سبز و خاکستری می شود که بین دو رنگ نارنجی و سبز فضای زیادی هدر می رود، اما می دانیم که بهینه ترین حالت در شکل می تواند فعالیت های سبز و آبی باشند که به صورت پیاپی اجرا شده ولی با بقیه فرآیندها همپوشانی بیشتری دارد.

2- فرض کنید تعداد زیادی اتاق کنفرانس در اختیار داریم و تعدادی فعالیت که بعضی از آنها با هم همپوشانی دارند. هر فعالیت با زمان شروع و پایانش مشخص شده است. میخواهیم، فعالیتها را به گونهای در اتاقهای کنفرانس برگزار نماییم که حداقل تعداد اتاقها را اشغال نماییم. بدیهی است، فعالیتهایی که با هم همپوشانی دارند نمیتوانند در یک اتاق برگزار شوند. راه حلی حریصانه برای این مسئله ارائه دهید.

برای رسیدن به بهترین و بهینه ترین جواب باید در هر مرحله ابتدا طولانی ترین فعالیت سازگار را انتخاب و آن را با توجه به طولانی ترین فعالیت های سازگار باقی مانده جدا کنیم و برای آن یک اتاق در نظر بگیریم، بدین صورت برای فرآیند های دیگر همین روند را انجام می دهیم و طولانی ترین فرآیندها را در ابتدا جدا کرده و سپس فعالیت های سازگار با آن را جدا می کنیم و سوال را حل می کنیم.

برای توصیف بهتر می توانیم به نمودار زیر توجه کنیم :



حال مراحل را به ترتیب نوشته شده در بالا انجام می دهیم :

بدین صورت و به سادترین شکل می توان بهینه ترین حالت را برای تعداد اتاق ها و تعداد فعالیت ها در نظر گرفت.

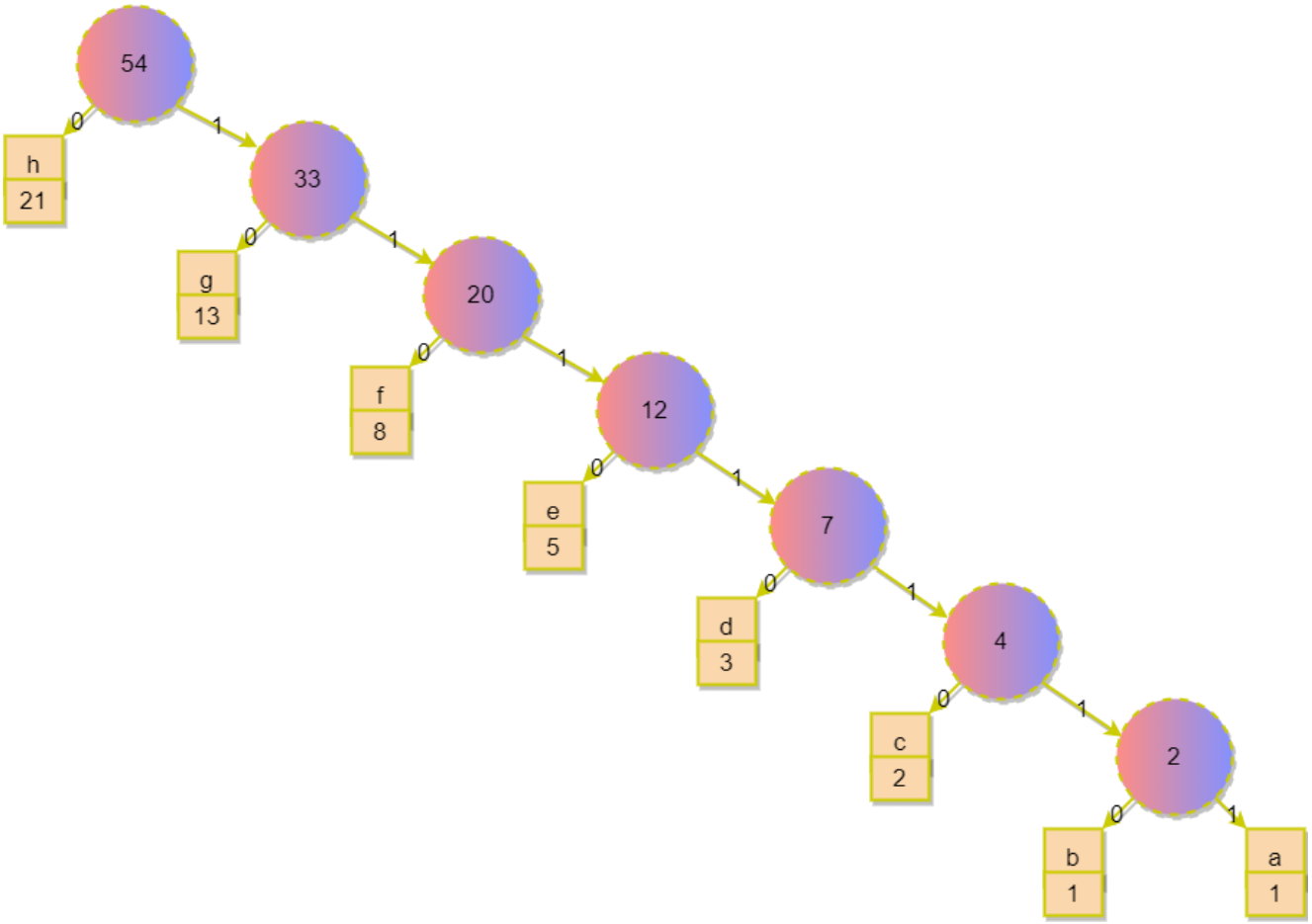
در نمدار بالا می توان دریافت با توجه به طولانی ترین فرآیند ها و سازگاری ها، حداقل به 4 اتاق نیاز داریم تا بتوانیم هر کلاس را به صورت کامل و بدون تداخل برگزار کنیم.

3- فرض کنید فرکانس رخداد کاراکترهای یک متن، اعداد فیبوناچی و به صورت زیر هستند:

a: 1, b: 1, c: 2, d: 3, e: 5, f: 8, g: 13, h: 21

- کد هافمن را برای این مجموعه کاراکتر تولید نمایید.

برای توضیح بهتر طریقه به دست آوردن کد هافمن می توانیم از نمودار آن استفاده کنیم:



برای رسم این نمودار ابتدا آرایه را به صورت صعودی مرتب میکنیم، سپس با استفاده از الگوریتم هافمن، کوچک ترین عدد را با عدد موجود در آرایه بعدی جمع می نمائیم.

- n کاراکتری که فرکانس آنهاn آیا می توانید پاسخ خود را به صورت عمومی برای هر مجموعه عنصر اول فیبوناچی است، بیان نمایید؟

عدد در نظر گرفته شده برای هر حرف را بررسی میکنیم :

- a : 1111111
- b : 1111110
- c : 111110
- d : 11110
- e : 1110

- f : 110
- g : 10
- h : 0

با توجه به تعداد یک و صفرها می‌توان دنباله ای را در آن پیدا کرد، بدین صورت که:

a : 1(N-1)1

b : 1(N-1)0

c : 1(N-2)0

d : 1(N-3)0

e : 1(N-4)0

f : 1(N-5)0

g : 1(N-6)0

h : 1(N-7)0

- برای ذخیره داده ها با کد مربوطه چه میزان فضا احتیاج است. این مقدار را با فضای لازم در صورت استفاده از کد اسکی مقایسه نمایید.

برای ذخیره سازی کاراکتر ها با استفاده از کد هافمن داریم:

a : 7 bit
b : 7 bit
c : 6 bit
d : 5 bit
e : 4 bit
f : 3 bit
g : 2 bit
h : 1bit

که سه حالت داریم:
بدترین حالت ذخیره سازی که همه کاراکتر ها a یا b باشند که 7 بیتی هستند، آنگاه داریم : 7*n

حالت میانگین که به صورت متفاوت از هر یک از کاراکتر ها استفاده شده باشد، تقریبا :
$\frac{1+2+3+4+5+6+7+7}{8}$ = 4.37 * n

بهترین حالت که همه کاراکتر ها h باشد : 1 * n

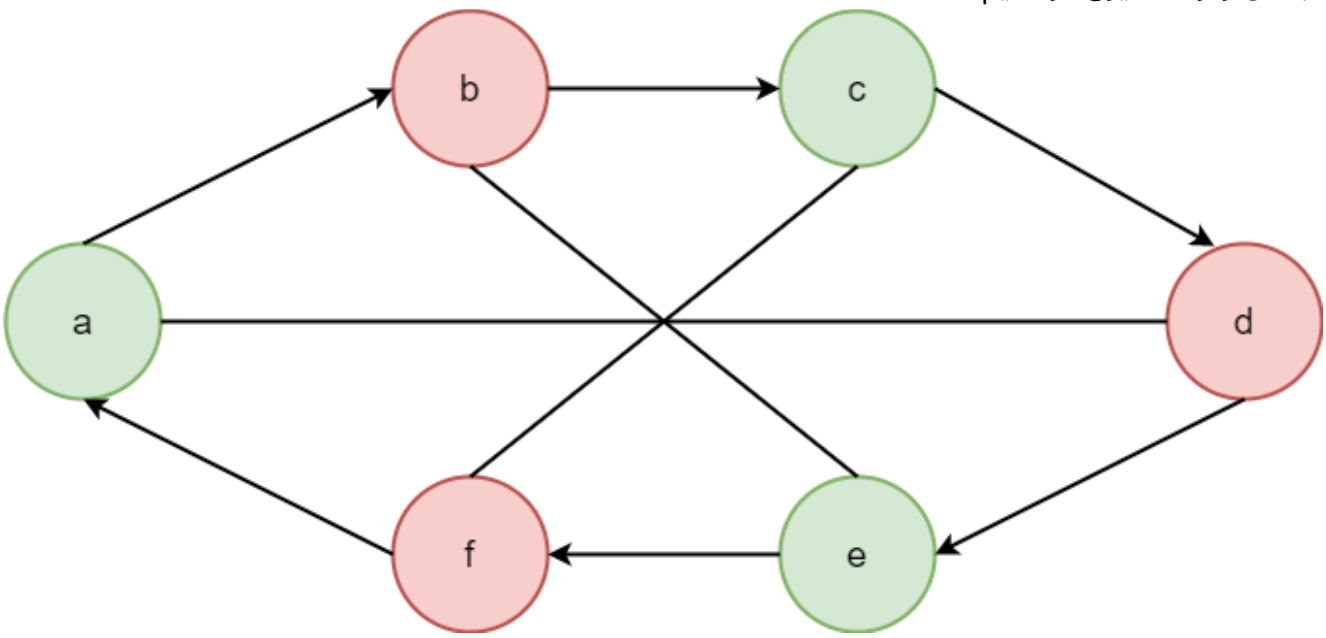
حال اگر برای این کاراکترها از کد اسکی استفاده کنیم، داریم : 8 * n

پس به ازای هر کاراکتر به تقریب می توان گفت که 3.63 = 8-4.37 بیت کمتر مصرف می شود و روش هافمن بهینه تر و فشرده تر از کد اسکی عمل می‌کند.
حال اگر برای مثال، یک عبارتی 8 حرفی شامل تمام حروف ذکر شده داشتیم باشم :
ASCII : 8 * 8 = 64 bit
Huffman = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 7 =35 bit

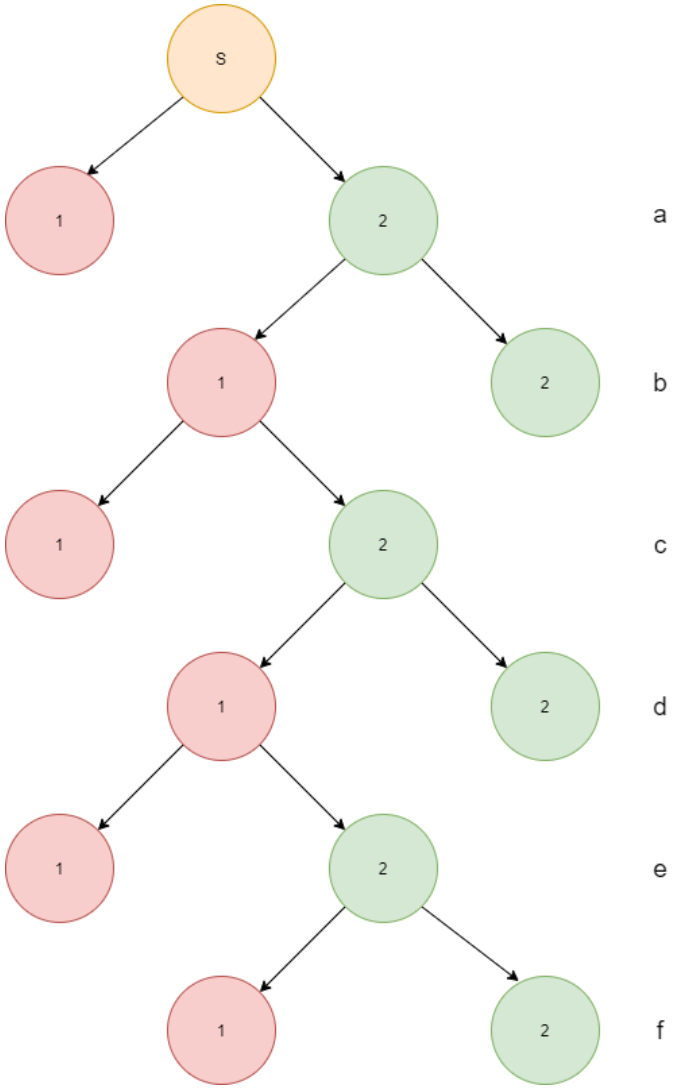
مسائل پس گرد :

1- درخت پسگرد را برای مسئله رنگ آمیزی گراف زیر با 2 رنگ رسم نمایید.

برای حل این مسئله می توانیم از یک نود شروع کرده و به دلخواه در مرحله اول به ان یک رنگ بدهیم، برای مثال در اولین گام به آن رنگ سبز بدهیم، سپس می توانیم به صورت ساعتگرد یا پادساعتگرد یا تصادفی به یکی از نود های مجاور آن برویم و آن را با رنگی غیر از سبز، مثلا قرمز رنگ بزنیم، سپس همین کار را برای نود مجاور دیگر که آن را رنگ نزده بودیم انجام دهیم و این کار را برای تمامی نود های گراف انجام دهیم تا همه نود ها به صورت کامل رنگ آمیزی شوند.
برای این منظور پرش تصادفی بین نودها پیشنهاد نمی شود زیرا ممکن است چندین بار یک نود را مشاهده کنیم که قبلا آن را رنگ آمیزی کرده ایم.



بدین صورت همانند فلش های رسم شده نمودار به سادگی و در سریع ترین زمان رنگ می شود بدون آن که هیچ دو نود مجاوری با یکدیگر همرنگ باشند.



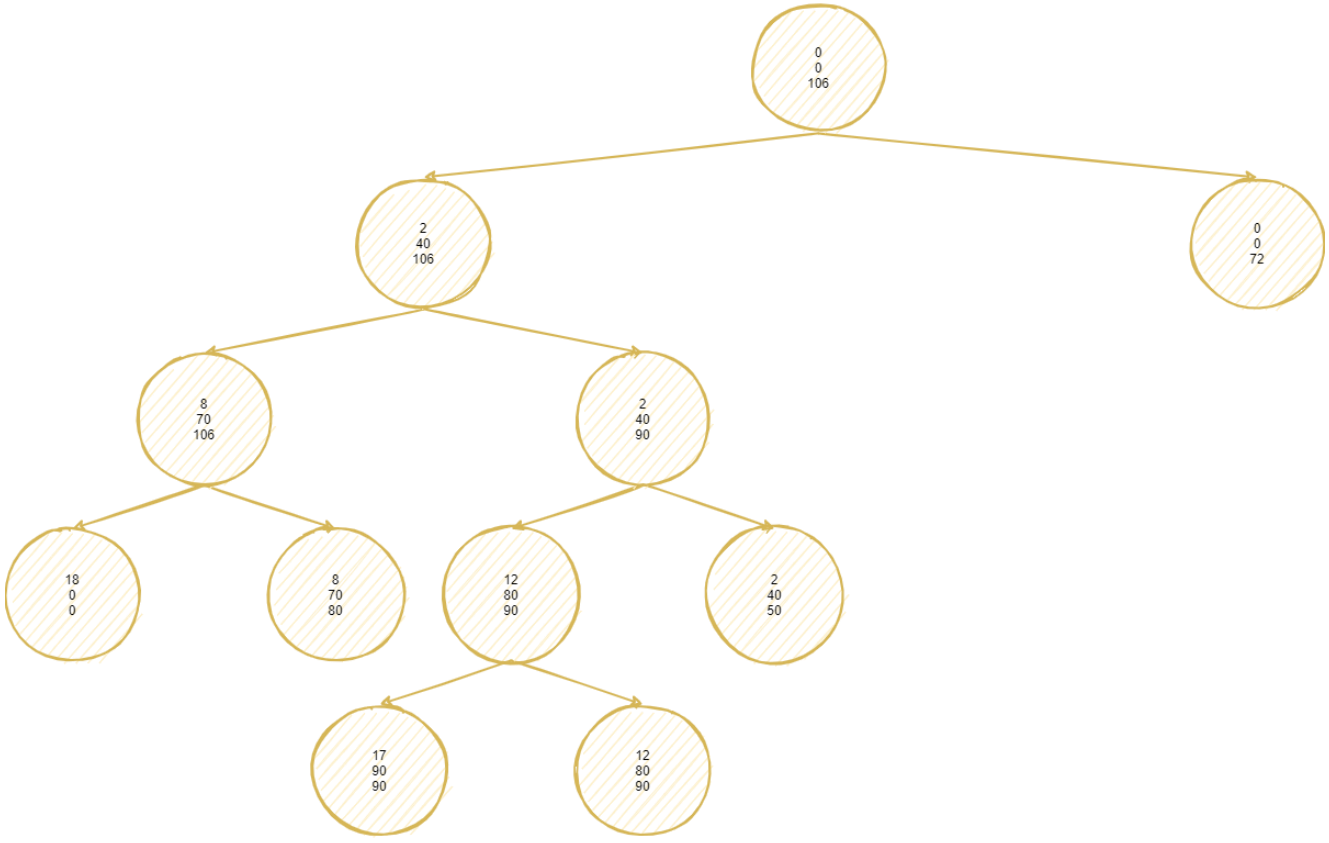
بدین صورت نیز میتوان با استفاده از نمودار این شکل را رنگ آمیزی کرد یا نشان داد که هر بار در خانه مجاور به رنگ دیگر رفته ایم.

2- مسئله زیر یک مسئله کوله پشتی 1-0 است. این مسئله را به هر دو روش پویا و پسگرد، حل نمائید.

w_i	p_i
2	40
6	30
10	40
5	10

W = 17

روش پسگرد یا حریصانه :



در روش پسگرد یا حریصانه آیتم ها را مطابق جدول رو برو به ترتیب ارزش بر وزن می نویسیم :

Item	w_i	p_i	P_i/W_i
1	2	40	20
2	6	30	5
3	10	40	4
4	5	10	2

حال در مرحله اول بیشترین ظرفیتی که می توان از آن وزن برداشت را کسری شدن بر میداریم :

2 + 6 + 9 = 17

40 + 30 + 36 = 106

و طبق توضیحات ارائه شده هر بار با کشیدن و امتحان کردن حالت درخت را رسم می کنیم تا به بهینه ترین جواب برای این سوال برسیم .

روش پویا :

طبق فرمول زیر شروع به رسم حالات از k برابر 4 تا k برابر 2 می کنیم تا به بهینه ترین جواب ممکن برسیم :

$$v[k,w] = \begin{cases} v[k-1,w] & \text{if wk > w} \\ max = \{v[k\text{-}1,w], v[k\text{-}1,w\text{-}wk] + bk\} & \text{else} \end{cases}$$

در هر مرحله، برای به دست آوردن مقادیر مجهول باید مقادیر دیگری را محاسبه نمود که این روش را وابسته به حالت های دیگر می کند و باعث بوجود آمدن پویایی در الگوریتم می شود :

V[4, 17] = max (V[3, 17] , V[3, 12] + 10)

V[3, 17] = max (V[2, 17] , V[2, 7] + 40)

V[3, 12] = max (V[2, 12] , V[2, 2] + 40)

V[3, 12] = max (V[2, 12] , V[2, 2] + 40) = max(70, 80) = 80

V[3, 17] = max (V[2, 17] , V[2, 7] + 40) = max(70, 80) = 80

V[2, 7] = max (V[1, 7] , V[1, 1] + 30) = max(40, 30) = 40

V[2, 12] = max (V[1, 12] , V[1, 6] + 30) = max(40, 70) = 70

V[2, 17] = max (40, V[1, 11] + 30) = max(40 , 70) = 70

جواب نهایی با استفاده از محاسبه حالات مختلف بدین صورت بدست می آید :

- **V[4, 17] = max (V[3, 17] , V[3, 12] + 10) = max(80, 90) = 90**