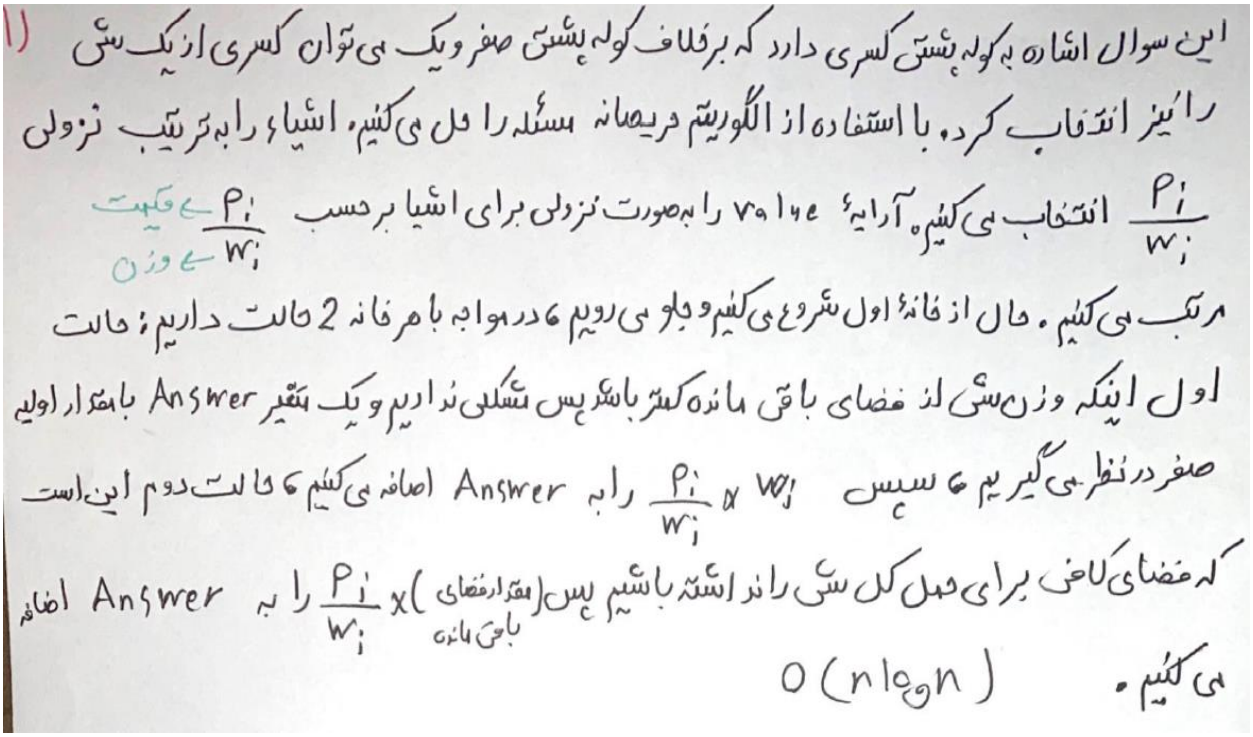
حل تکلیف پنجم – طراحی الگوریتم – سوال1و2و3

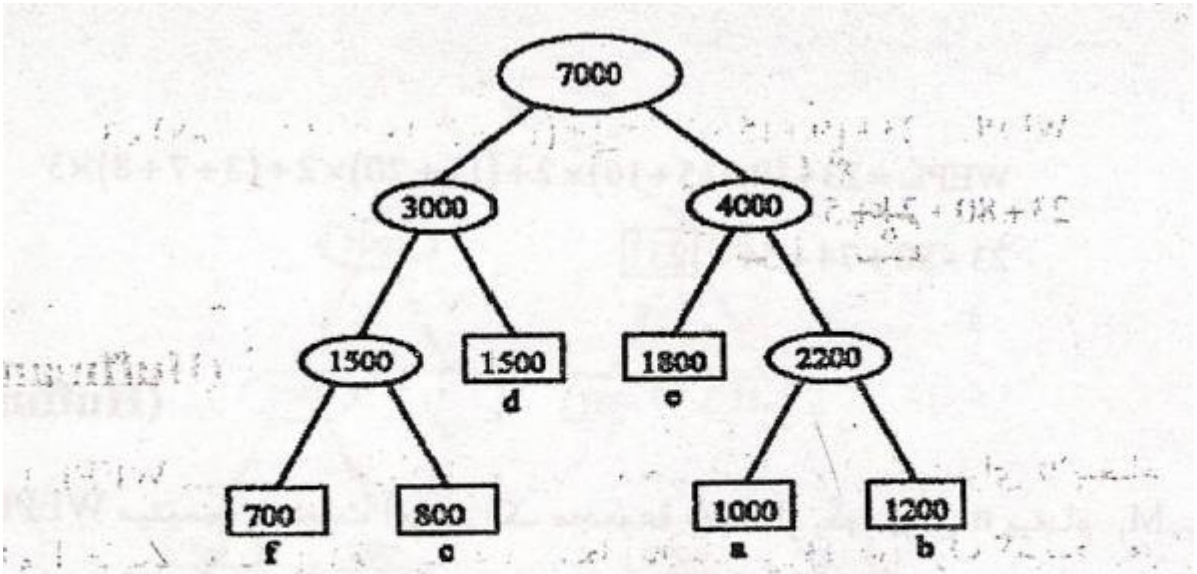
سوال1:



سوال2:

حل :

اگر این 7 حرف را مشابه هفت پیغام و دفعات تکرار آن‌ها را مشابه فرکانس نسبی مورد اشارهٔ بالا تعبیر کنیم، درخت دودویی با WEPL مینیمم به صورت زیر خواهد بود:



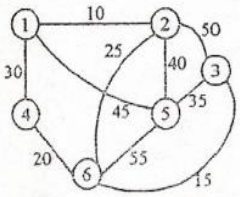
$$WEPL = 700\times3+800\times3+1500\times2+1800\times2+1000\times3+1200\times3 \Rightarrow$$

$$WEPL = \boxed{17700}$$

سوال3:

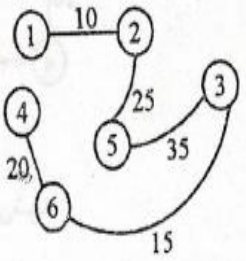
الگوریتم پریم: ایدهٔ این الگوریتم به این صورت است که از یک راس دلخواه شروع می‌کنیم (برای سهولت از راس 1 شروع خواهیم کرد)، این راس را به عنوان راس پردازش شده تلقی خواهیم کرد و در یک مرحلهٔ عمومی، راسی از بین رئوس پردازش نشده به گونه‌ای انتخاب خواهیم کرد که با حداقل هزینه به یکی از رئوس قبلا پردازش شده وصل شده باشد.

شکل‌های زیر مراحل مختلف انتخاب رئوس را برای گراف شکل مقابل نشان می‌دهند.



شکل ۱

مرحله	راس انتخاب شده	هزینهٔ یال اضافه شده به درخت	درخت در مرحلهٔ i
0	1	-	
1	2	10	
2	6	25	
3	3	15	
4	4	20	



5 5 35

و جمع هزینهٔ یال‌های انتخاب شده 105 است.

همان‌گونه که مشاهده شد، الگوریتم با درختی که فقط یک راس است (برای سهولت راس 1) شروع می‌شود. بعد n-1 یال درخت پوشای مینیمم، مرحله به مرحله به این درخت اضافه می‌شوند. یال بعدی {i ,j} که به این درخت باید اضافه شود به گونه‌ای است که راس i قبلاً در درخت بوده و j راسی است که در درخت نیست و cost[i ,j] از بین تمامی cost[k , l] ها ،که در آن k در درخت بوده و l در درخت نباشد، مینیمم است. اگر آرایه Near [1..n] را به صورت زیر تعریف کنیم:

Near [j]=0 اگر و تنها اگر راس j قبلا در درخت قرار داده شده و Near [j]=i ≠ 0 اگر و تنها اگر cost[i ,j] مینیمم باشد، در این صورت مطالب ارایه شده را می‌توان به صورت دقیق‌تر در الگوریتم زیر بیان کرد:

```
Algorithm Prim (cost , n , T , mincost)
mincost=0
for  i=2  to  n  do  Near [i]=1
Near [1]=0
for  i=1  to   n-1   do
{
j را به گونه‌ای انتخاب کنید که
        (Near[j]≠0) and (cost[j , Near [j]) مینیمم باشد)     (*)
```

```
        (T[i,1],T[i,2]) = (j, Near[j])
        min cost = min cost + cost[j, Near[j]]
        Near[j] = 0
    for  k = 2   to  n   do
        if (Near[k] ≠ 0) and (cost[k, Near[k]]) > cost[k, j]) then        (**)
            Near[k] = j
}
```

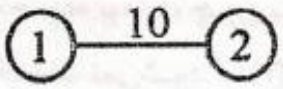
الگوریتم کراسکال

خلاصهٔ الگوریتم کراسکال به این صورت است. که ابتدای یال‌ها را به صورت صعودی هزینهٔ آن‌ها مرتب می‌کنیم.سپس با شروع از ابتدای این لیست مرتب شد، یالی را انتخاب خواهیم کرد که اضافه کردن آن به مجموعه یال‌های قبلاً انتخاب شده تشکیل دور ندهد. این عمل بعد از انتخاب (n-1) امین یال خاتمه خواهد پذیرفت. مراحل اجرای الگوریتم کراسکال برای گراف شکل 1، در اشکال زیر به تصویرکشیده شده‌اند:

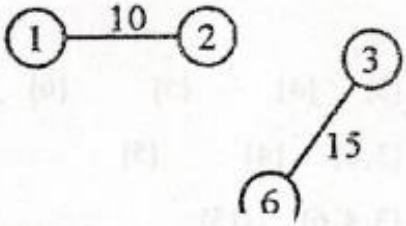
i	یال	هزینه	وضعیت انتخاب سیال
1	1,2	10	✓
2	3,6	15	✓
3	4,6	20	✓
4	2,6	25	✓
5	1,4	30	-
6	3,5	35	✓
7	2,5	40	
8	1,5	45	
9	2,3	50	
10	5,6	55	

لیست مرتب شده E

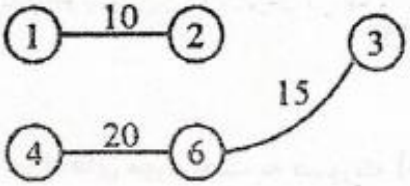
مرحلة ١



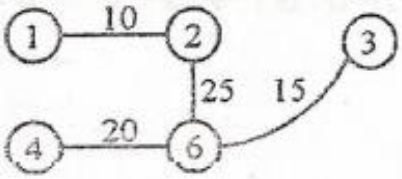
مرحلة ٢



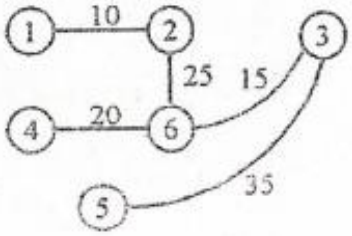
مرحلة ٣



مرحلة ٤



مرحلة ٥



با توجه به مطالب بالا الگوریتم کراسکال را می‌توان به صورت دقیقتر زیر نوشت:

برای این الگوریتم فرض شده است که نخست یال‌های گراف به صورت صعودی بر حسب هزینهٔ آن‌ها مرتب شده و در آرایهٔ E ذخیره شده‌اند.

```
Algorithm Kruskal (E , n , T , Mincost)
Mincost= 0  ,  N_edge = 0 , i=1  ,  T= ϕ
While   N_edge < (n-1)  do
{
    e= (u , v) = (E=[i , 1] , E [i , 2])
    if (اضافه کردن یال e به T تشکیل دور ندهد) then
    {
        T=TU{e}
        Mincost= mincost + cost [u,v]
        N_edge = N_edge +1
    }
      i= i+1
}
```

این توضیحات اضافه برای یادگیری بیشتر شما عزیزان نوشته شده، و نیازی به شرح دقیق آنها توسط شما نبوده است.

☺☺