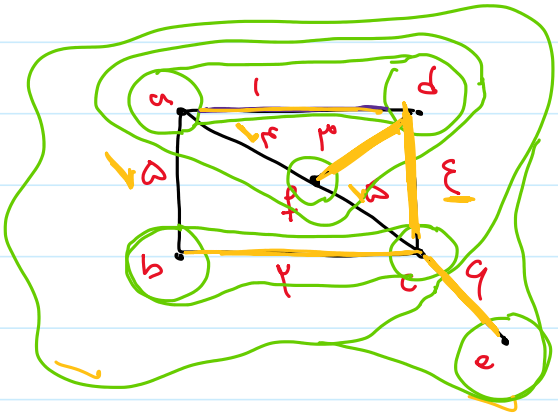
جلسه بیست و هشتم

13:15 Sunday, January 2, 2022

موضوع : مجموعه‌های جزا . disjoint Set



union_find : داده ساختاری که ۳ عمل زیر را انجام دهد:

Makeset(x) : یک مجموعه شامل عنصر x ایجاد می‌کند.

find(x) : مجموعه‌ای که شامل عنصر x است را برمی‌گرداند.

union(x,y) : مجموعه‌هایی که شامل x و y هستند را با هم ادغام می‌کند.

{1} {2} {3} {4} {5}

{1,2} {3} {4} {5}

union(1,2) →

union(4,3) → {1,2} {3,4} {5}

union(1,4) → {1,2,3,4} {5}

find(2) → 1

*هر مجموعه با عنصر نماینده‌اش مشخص می‌شود.

۱- Quick-find فرض کنید عناصر ما ۱ تا n هستند

Set | ۱ | ۲ | ۳ | ۴ | ۵ | ۶ | .... | n |

{1} {2} ---- {n}

find(1) : 1

union(1, 2)

union(3, 4)

اول کار: به ازای هر i، set[i]=i

```
int find(i) {
    return(set[i])
}
```

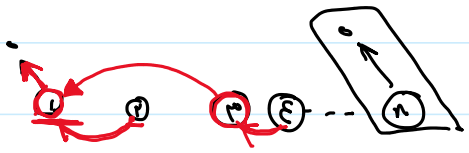
```
union(i, j){
    x = set[i]
    y = set[j]
    for(l: 1 → n)
        if (set[l]==x)
            set[l]=y
}
```

O(1) : find

O(n) : union

۲- Quick union - پیاده سازی درختی.

ایده: استفاده از اشاره‌گرها



{1} {2} ... {n}

find(1) = 1

union(1, 2)

union(3, 4)

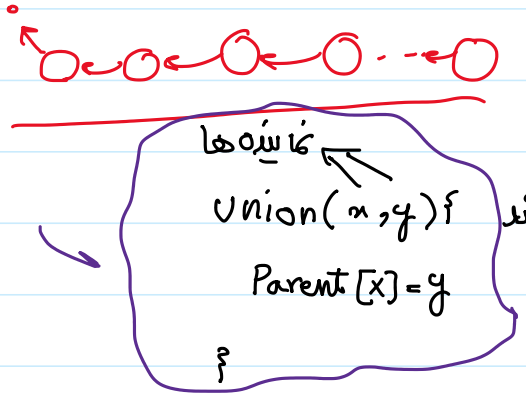
union(2, 4) →

```
int find(i) {
    if(Parent[i]==0)
```

```
union(i, j) {
    x = find(i)
```

```
if(Parent[i]==0)
    return(i)
return find(Parent[i])
}
```

```
( x=find(i)
( y=find(j)
Parent[x]=y
}
```



زمان اجرا : find(i) : $O(n)$

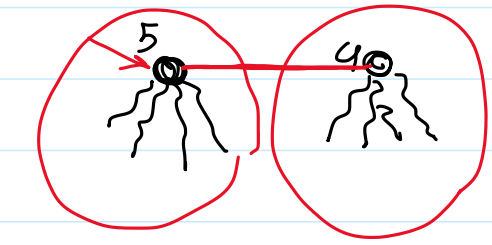
ادغام : $O(n)$

```
union(x,y){
    Parent[x]=y
}
```

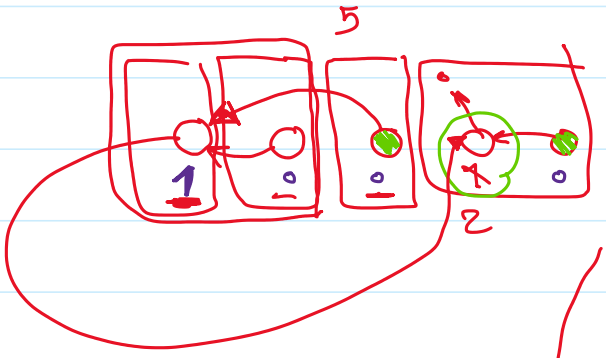
تابع نمایندگی

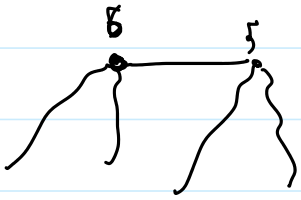
ادغام درصورتی که ورودی نمایندگان باشد $O(1)$.

بهبود ۱ : هر بار ریشه درخت با ارتفاع کمتر را زیر ریشه درخت با ارتفاع بیشتر قرار دهیم.



ارتفاع : rank





نماینده

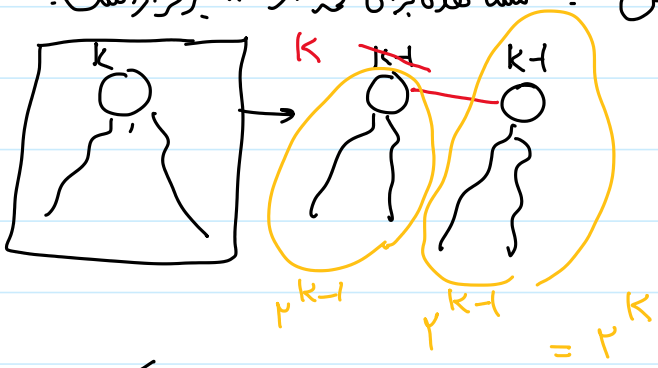
```
union(x,y){
    if rank[x]>rank[y]
        Parent[y]=x
    else if rank[y]>rank[x]
        Parent[x]=y
    else
        Parent[y]=x
        rank[x]++
```

؟

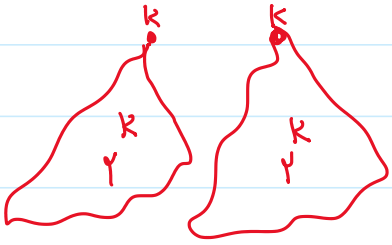
** مشاهده : اگر مقدار رنک یک راس برابر k باشد، زیر درخت شامل آن راس حداقل 2^k راس دارد.

استقرا: $k=0$ ← 1 راس

فرض : مشاهده برای کمتر از k برقرار است.



* نتیجه : تعداد راس های با رنک k حداکثر $\frac{n}{2^k}$ است.

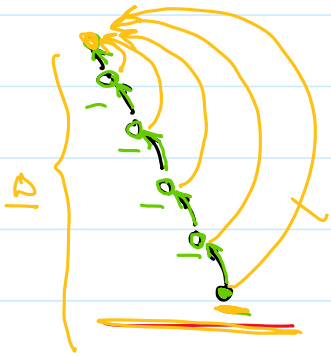


* نتیجه : ارتفاع درختها حداکثر $\log n$ است. *

* زمان find به $O(\log n)$ کاهش پیدا کند.

زمان union : $O(1)$

پیاده سازی ۲:




```
find(x){
  if (parent(x)==0)
    return(x)
  else
    Parent[x]=find(Parent[x])
    return(Parent[x])
}
```

نتیجه بهبود: union : $O(1)$ ، find : $O(\alpha(n))$

$O(\log^* n)$ ← k ← $2^{2^{2^{\cdots^2}}}$ ← $n = 2^{2^{2^{2^{2}}}}$ ← $\log^*(n) < 6$

$\log^* 4 = 2$

$\log^* 8 = 3$

$\log^* 2^{16} = 4$ ← 65536

$\log^* 2^{65536} = 5$

* هزینه سرشکن هر عملیات find برابر $O(\log^* n)$ است.

مشاهده ۳) با حرکت از هر راسی به سمت ریشه، مقدار رنک راس‌ها زیادی شود.

مشاهده ۴) اگر زمانی یک راس از ریشه بودن درآمده، دیگر رنک آن تغییر نمی‌کند.

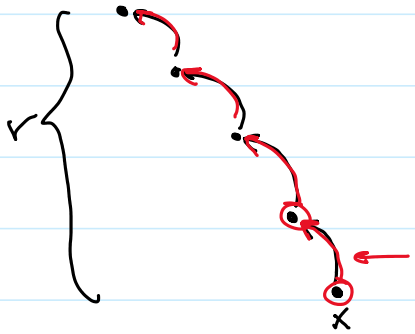
حال: راس‌ها را براساس رنک در $\log^*$ گروه مختلف قرار می‌دهیم

[0 1] [2 2] [3 4] [5 16] [17 65536] [65537 2^{65536}]

گروه ۱ | گروه ۲ | ... | $[k+1 \; 2^k]$

$2 \uparrow k$ ← $2^{2^{\cdots^2}}$ ← k

$\log^*(2 \uparrow k) = k$



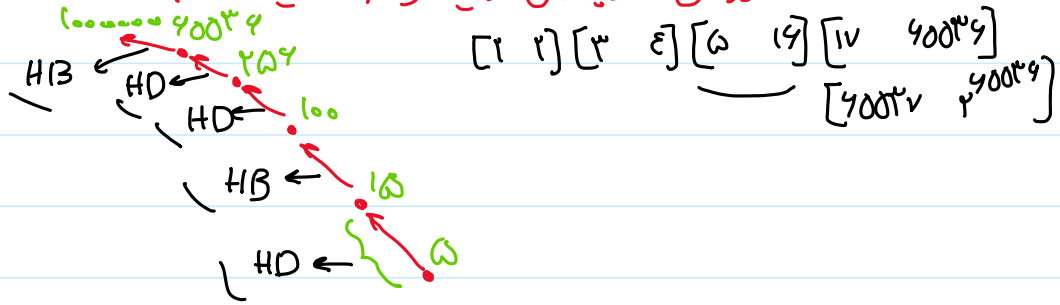
هزینه این find = ۲

اگر ۲ بار با گروه، هزینه find برابر ۲ است. این هزینه را به ۲ قسمت

اگر ۲ بار با ۵ برویم، هزینه find برابر ۲ است. این هزینه را به ۲ قسمت تقسیم می‌کنیم:

HB: اگر پدر راس v ریشه است یا Parent[v] در گروه دیگری نیست به v قرار دارد، یک واحد به HB اضافه می‌کنیم.

HD: اگر راس v و پدرش در یک گروه بودند یک واحد به HD اضافه می‌کنیم

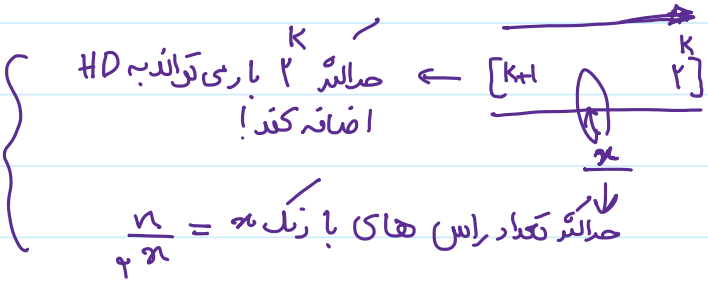


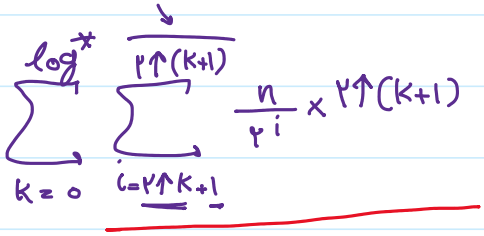
*بعد از n عملیات، مقدار HD و HB چقدر است؟

اول HB: به ازای هر find مقدار HB حداکثر $\log^* n$ تغییر می‌کند؟

← بعد از n عملیات HB حداکثر $n\log^* n$.

دوم HD:





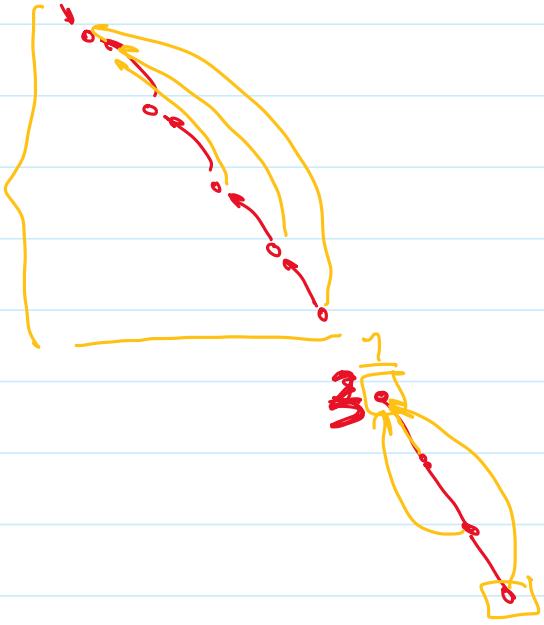
$$\sum_{i=2\uparrow k+1} \frac{n \times 2\uparrow(k+1)}{2 \times 2^{2\uparrow k}} = \frac{n \times 2\uparrow(k+1)}{2 \times 2\uparrow(k+1)} = \frac{n}{2}$$

$$i = 2\uparrow k+2 \quad = \frac{n}{4}$$

$$i = 2\uparrow k+3 \quad = \frac{n}{8}$$

$$\sum_{k=0}^{\log^* n} n = n \log^* n \quad \square$$

$\therefore O(n \log^* n) = HB + HD$



```
union(x, y){
  Paren[x] = y
}
```

```
union(i, j){
  x = find(i)
  y = find(j)
  union'(x, y)
}
```