طراحی الگوریتم ها

روش حریصانه

استاد درس: مهدی جبل عاملی

مساله خرد کردن پول

$\{1, 2, 5, 10, 25\}$

73

مساله خرد کردن پول

```
Coin (A) {
C = {1, 2, 5, 10, 25};
S = {}, Sum = 0 ;
while (sum ≠ A)
{
      x ← بزرگترین سکه      شروط | sum + x ≤ A ;
      if (چنین سکه‌ای وجود نداشته باشد)
            return "No Solution"
      sum += x ;
      S = S ∪ {x};
}
return S; }
```

مساله خرد کردن پول

- روش حریصانه (Greedy)

```
Coin (A) {
c = {1, 2, 5, 10, 25} ;
S = {}, Sum = 0 ;
while (sum ≠ A)
{
        x ← بزرگترین سکه    شرط | sum + x ≤ A ;
        if (چنین سکه‌ای وجود نداشته باشد)
                return "No Solution"

        sum += x ;
        S = S U {x} ;
}
return S ; }
```

روش حریصانه (Greedy)

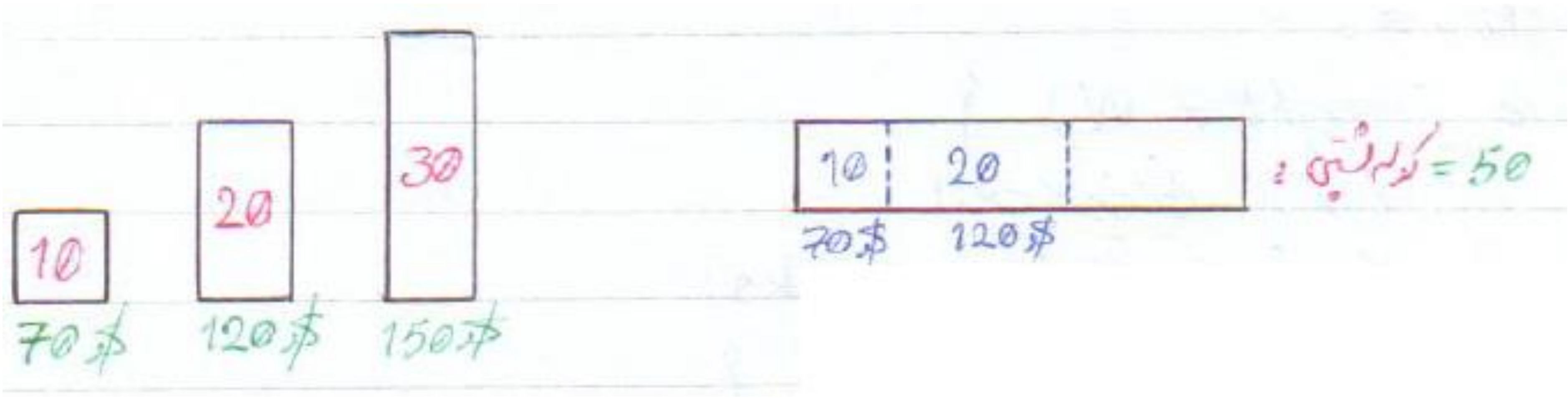
```
Greedy (C) {
    S = {};
    while  C ≠ {}  and  Not Solution(S)
    {
        x = Select (C) ;
        C = C - {x} ;
        if  Feasible (S U {x})
                S = S U {x} ;
    }
    if  Solution (S)
        return  Objective (S) ;
    else
        return  "No solution" ;
}
```

کوله پشتی 0-1

- معیار انتخاب شیء
  - شیء با ارزش بالاتر
  - شیء با وزن کمتر
  - شیء با نسبت ارزش به وزن بیشتر

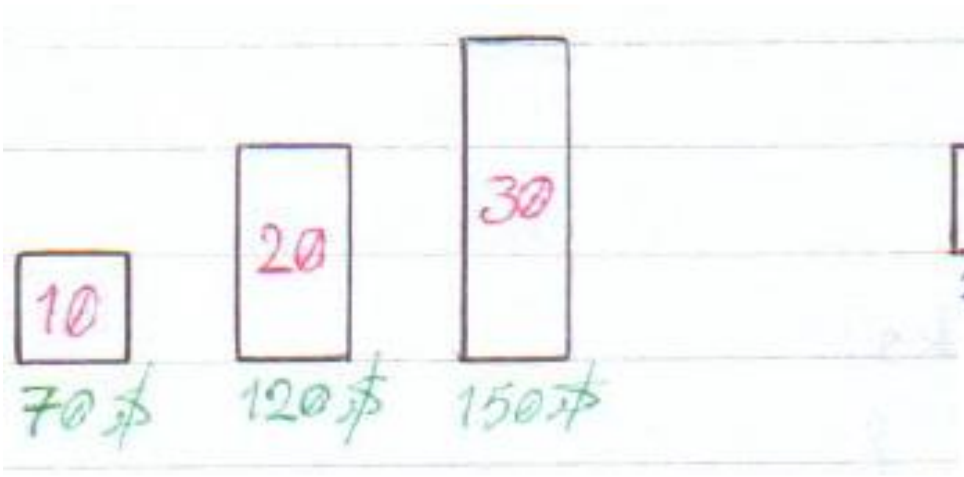
کوله پشتی 0-1

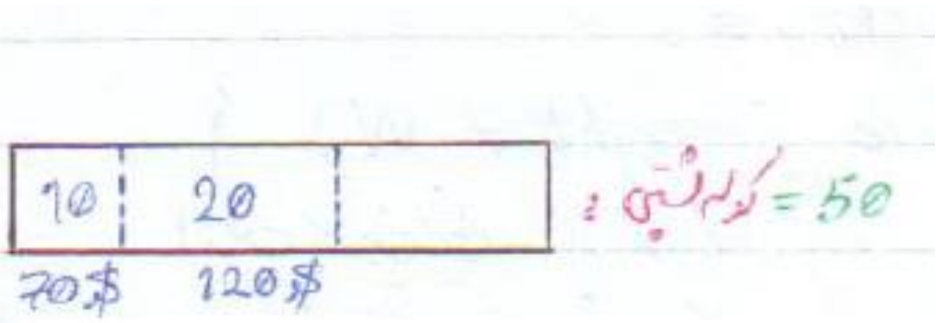
- معیار انتخاب شیء
  - شیء با ارزش بالاتر
  - شیء با وزن کمتر
  - شیء با نسبت ارزش به وزن بیشتر



کوله پشتی 0-1

- معیار انتخاب شیء
  - شیء با ارزش بالاتر
  - شیء با وزن کمتر
  - شیء با نسبت ارزش به وزن بیشتر

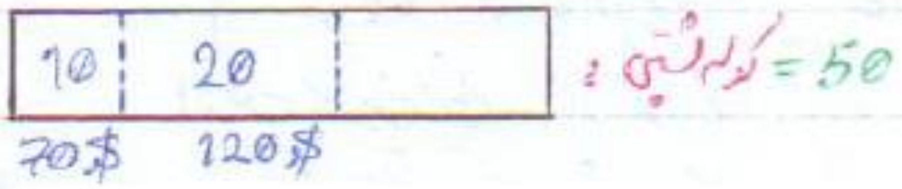




- متأسفانه ایده سوم هم جواب نمی دهد

کوله پشتی 0-1

- معیار انتخاب شیء
  - شیء با ارزش بالاتر
  - شیء با وزن کمتر
  - شیء با نسبت ارزش به وزن بیشتر



- متأسفانه ایده سوم هم جواب نمی دهد

نتیجه: هیچ ایده حریصانه ای برای این مساله وجود ندارد.

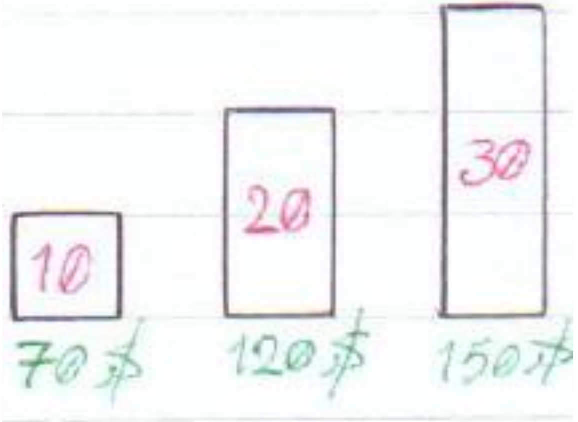
کوله پشتی کسری (Fractional Knapsack)

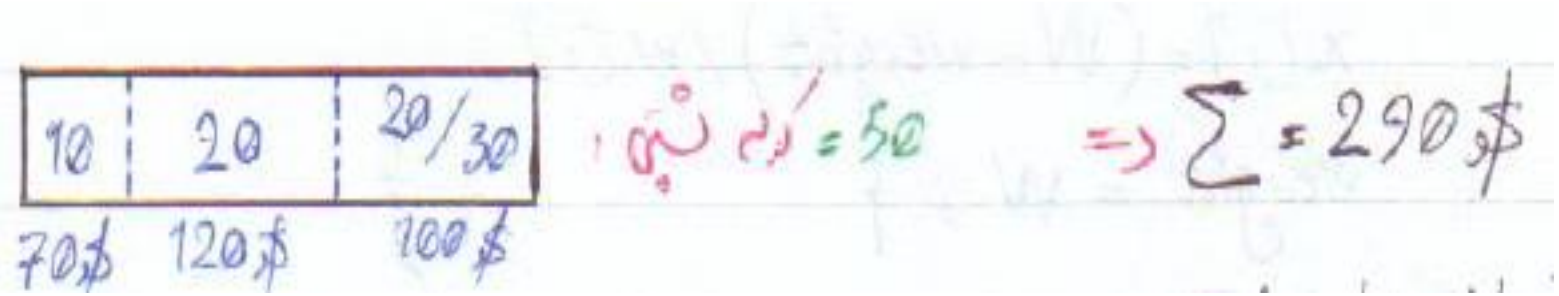
- ورودی ها مشابه کوله پشتی 0-1 است. تنها تفاوت در این است که $0 \leq x_i \leq 1$

کوله پشتی کسری (Fractional Knapsack)

$0 \leq x_i \leq 1$

- ورودی ها مشابه کوله پشتی 0-1 است. تنها تفاوت در این است که
- معیار انتخاب شیء: شیء با نسبت ارزش به وزن بیشتر

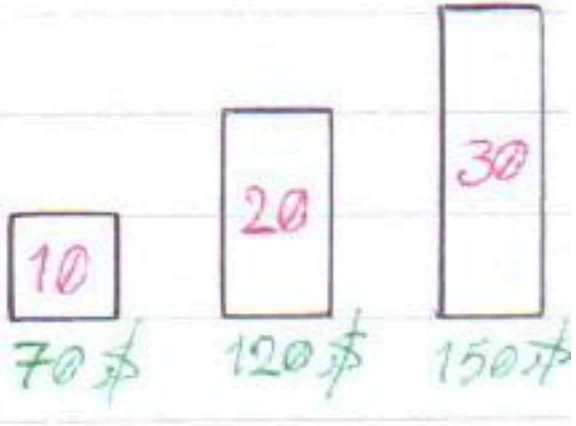


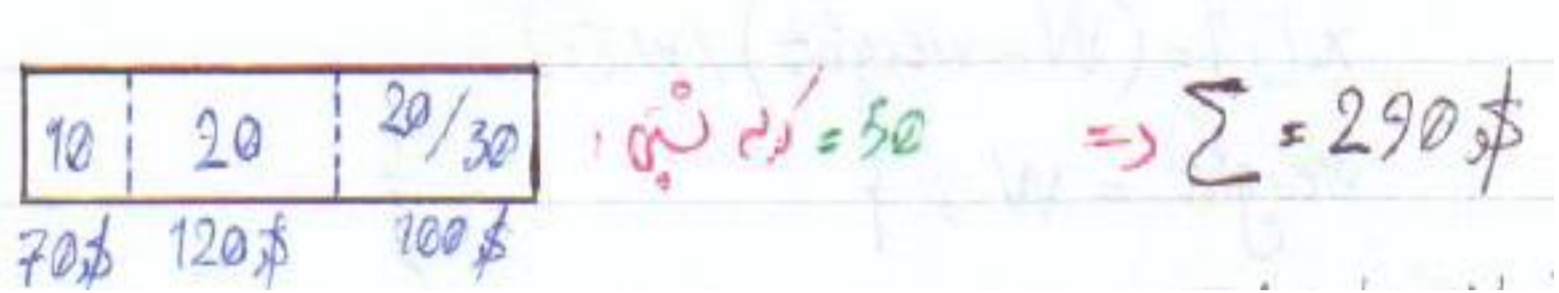


کوله پشتی کسری (Fractional Knapsack)

$0 \leq x_i \leq 1$

- ورودی ها مشابه کوله پشتی 0-1 است. تنها تفاوت در این است که
- معیار انتخاب شیء: شیء با نسبت ارزش به وزن بیشتر





- اثبات:

کوله پشتی کسری

```
Fractional-knapsack (n, w[1..n], p[1..n], W) {
    for i=1 to n do
        x[i] = 0 ;
    weight = 0 ;
    while (weight ≠ W) {
        i = ارزنده‌ترین شیء
        if شیء دیگری وجود نداشت break ;
        if w[i] + weight ≤ W {
            x[i] = 1 ;
            weight += w[i] ; }
        else {
            x[i] = (W - weight) / w[i] ;
            weight = W ; }
    }
    return x ; }
```

کوله پشتی کسری

- زمان اجرا:

```
Fractional-knapsack (n, w[1..n], p[1..n], W) {
    for i=1 to n do
        x[i] = 0 ;
    weight = 0 ;
    while (weight ≠ W) {
        i = اندیس بهترین شیء
        if شیء دیگری وجود نداشت break ;
        if w[i] + weight ≤ W {
            x[i] = 1 ;
            weight += w[i] ; }
        else {
            x[i] = (W - weight) / w[i] ;
            weight = W ; }
    }
    return x ; }
```

کوله پشتی کسری

```
Fractional-knapsack (n, w[1..n], p[1..n], W) {
    for i=1 to n do
        x[i] = 0;
    weight = 0;
    while (weight ≠ W) {
        i = اندیس بهترین شیئی
        if شیئی دیگری وجود نداشت break;
        if w[i] + weight ≤ W {
            x[i] = 1;
            weight += w[i]; }
        else {
            x[i] = (W - weight) / w[i];
            weight = W; }
    }
    return x; }
```

- زمان اجرا:

$) = n \times n \in O(n^2)$

کوله پشتی کسری

• بهبود زمان اجرا:

```
Fractional-knapsack (n, w[1..n], p[1..n], W) {
    for  i=1  to  n  do
        x[i] = 0 ;
    weight = 0 ;
    while (weight ≠ W)  {
        i =   اندیس بهترین نسبت
        if  شیئی دیگری وجود نداشت   break ;
        if  w[i] + weight ≤ W  {
                x[i] = 1 ;
                weight += w[i] ; }
        else {
                x[i] = (W - weight) / w[i] ;
                weight = W ; }
    }
    return  x ; }
```

کوله پشتی کسری

```
Fractional-knapsack (n, w[1..n], p[1..n], W) {
    for  i=1  to  n  do
        x[i] = 0 ;
    weight = 0 ;
    while (weight ≠ W) {
        i =   از بین بهترین شیء
        if  شیء دیگری وجود نداشت   break ;
        if  w[i] + weight ≤ W {
                x[i] = 1 ;
                weight += w[i] ; }
        else {
                x[i] = (W - weight) / w[i] ;
                weight = W ; }
    }
    return  x ; }
```

- بهبود زمان اجرا:

به شرط مرتب سازی اشیاء

$\in \theta(n \log n)$

ذخیره کردن فایل ها روی نوار

ذخیره کردن فایل ها روی نوار

- ورودی : n تعداد فایل ها

l_i طول فایل **i** ام

هدف: تعیین نحوه ذخیره فایلها روی نوار به طوری که میانگین زمان بازیابی حداقل باشد.

ذخیره کردن فایل‌ها روی نوار

مثال : $n=3$ $l_1=10$ $l_2=15$ $l_3=5$

<123>

ذخیره کردن فایل ها روی نوار

مثال : $n=3 \qquad l_1=10 \qquad l_2=15 \qquad l_3=5$

$\langle 123 \rangle \quad [10+(10+15)+(10+15+5)]/3 = 65/3 = 21.\bar{6}$

ذخیره کردن فایل‌ها روی نوار

مثال: $n=3 \quad l_1=10 \quad l_2=15 \quad l_3=5$

$\langle 123 \rangle \; [10+(10+15)+(10+15+5)]/3 = 65/3 = 21.\overline{6}$

$\langle 312 \rangle \; = 50/3 = 16.\overline{6}$

$\langle 132 \rangle \; = 60/3 = 20$

$\langle 321 \rangle \; = 65/3 = 21.\overline{6}$

$\langle 213 \rangle \; = 75/3 = 25$

$\langle 231 \rangle \; = 70/3 = 23.\overline{3}$

ذخیره کردن فایل ها روی نوار

- ایده حریصانه: فایل ها را به ترتیب اندازه فایل به صورت صعودی ذخیره کنیم.
- اثبات درستی ایده حریصانه:

ذخیره کردن فایل ها روی نوار

- ایده حریصانه: فایل ها را به ترتیب اندازه فایل به صورت صعودی ذخیره کنیم.
- اثبات درستی ایده حریصانه:

$P = \langle P_1 P_2 P_3 \cdots P_n \rangle$ فرض

$L_{P_i} = S_i$ تعریف

ذخیره کردن فایل ها روی نوار

- ایده حریصانه: فایل ها را به ترتیب اندازه فایل به صورت صعودی ذخیره کنیم.
- اثبات درستی ایده حریصانه:

فرض $P = \langle P_1 P_2 P_3 \cdots P_n \rangle$

تعریف $L_{P_i} = S_i$

میانگین زمان بازیابی $= \dfrac{S_1 + (S_1 + S_2) + (S_1 + S_2 + S_3) + \cdots + (S_1 + S_2 + \cdots + S_n)}{n}$

ذخیره کردن فایل ها روی نوار

- ایده حریصانه: فایل ها را به ترتیب اندازه فایل به صورت صعودی ذخیره کنیم.
- اثبات درستی ایده حریصانه:

$P = \langle P_1 P_2 P_3 \cdots P_n \rangle$ فرض

$L_{P_i} = S_i$ تعریف

$$\text{میانگین زمان بازیابی} = \frac{S_1 + (S_1 + S_2) + (S_1 + S_2 + S_3) + \cdots + (S_1 + S_2 + \cdots + S_n)}{n}$$

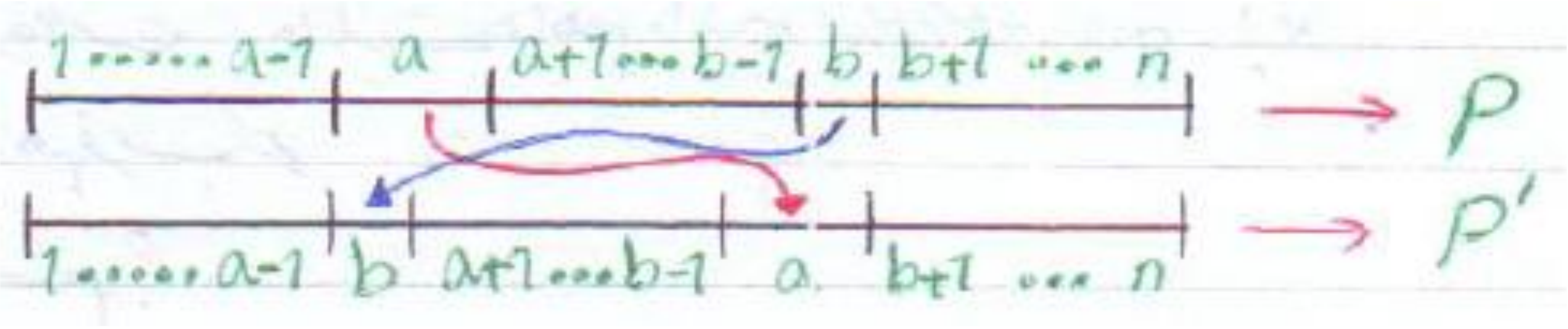
$$\text{صورت} = \sum_{i=1}^{n} (n-i+1) S_i = S(P)$$

ذخیره کردن فایل ها روی نوار

- اثبات درستی ایده حریصانه با برهان خلف:
- فرض کنید جواب P’ با میانگین زمان اجرای کمتر از P وجود داشته باشد.
- در این صورت P و P’ در حداقل دو فایل مثل a و b اختلاف دارند.

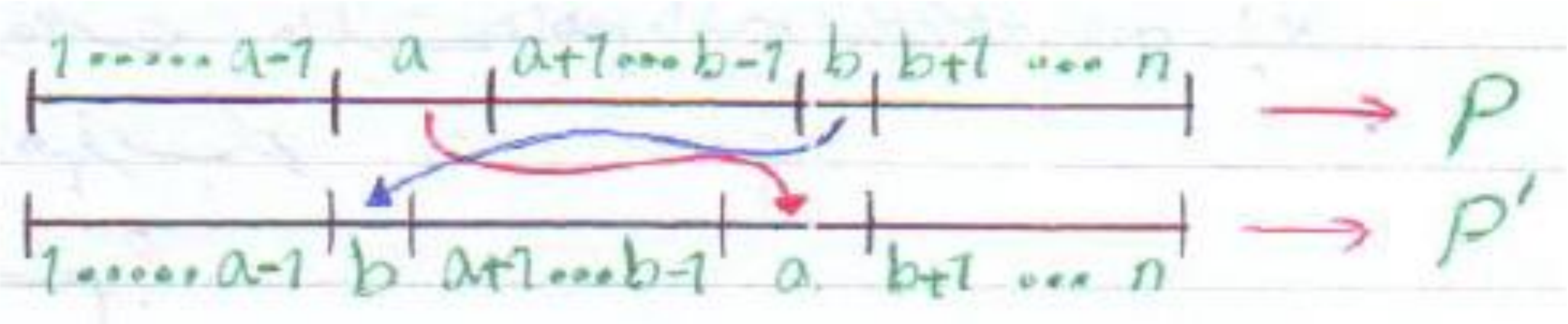
ذخیره کردن فایل ها روی نوار

- اثبات درستی ایده حریصانه با برهان خلف:
- فرض کنید جواب P’ با میانگین زمان اجرای کمتر از P وجود داشته باشد.
- در این صورت P و P’ در حداقل دو فایل مثل a و b اختلاف دارند(a<b)



ذخیره کردن فایل ها روی نوار

- اثبات درستی ایده حریصانه با برهان خلف:
- فرض کنید جواب ’P با میانگین زمان اجرای کمتر از P وجود داشته باشد.
- در این صورت P و ’P در حداقل دو فایل مثل a و b اختلاف دارند(a<b)



$$S(P') = \sum_{\substack{i=1 \\ , i \neq a,b}}^{n} (n-i+1) S_i + (n-b+1) S_a + (n-a+1) S_b$$

ذخیره کردن فایل ها روی نوار

- اثبات درستی ایده حریصانه با برهان خلف:

$$S(P') - S(P) = (b-a)(s_b - s_a)$$

ذخیره کردن فایل ها روی نوار

- اثبات درستی ایده حریصانه با برهان خلف:

$$s(p') - s(p) = (b-a)(s_b - s_a)$$

$$a < b \rightarrow s_a \leq s_b$$

ذخیره کردن فایل ها روی نوار

- اثبات درستی ایده حریصانه با برهان خلف:

$$S(P') - S(P) = (b-a)(S_b - S_a)$$

$a < b$ طبق فرض $\Rightarrow b - a \geq 0$

$S_a < S_b$ طبق فرض $\Rightarrow S_b - S_a \geq 0$

ضرب ۲ عدد مثبت $\Rightarrow \geq 0$

ذخیره کردن فایل ها روی نوار

- اثبات درستی ایده حریصانه با برهان خلف:

$$S(p') - S(p) = (b-a)(S_b - S_a)$$

$a < b$ طبق $\Rightarrow b - a \geq 0$

$S_a < S_b$ طبق $\Rightarrow S_b - S_a \geq 0$

ضرب ۲ عدد مثبت $\Rightarrow \geq 0$

$$S(p') - S(p) \geq 0$$

ذخیره کردن فایل ها روی نوار

- اثبات درستی ایده حریصانه با برهان خلف:

$$S(P') - S(P) = (b-a)(S_b - S_a)$$

$$\left.\begin{array}{l} a < b \text{ طبق} \Rightarrow b-a \geq 0 \\ S_a < S_b \text{ طبق} \Rightarrow S_b - S_a \geq 0 \end{array}\right\} \xrightarrow{\text{ضرب ۲ عدد مثبت}} \geq 0$$

$$S(P') - S(P) \geq 0$$

$$S(P') \geq S(P)$$

ذخیره کردن فایل ها روی نوار

- اثبات درستی ایده حریصانه با برهان خلف:

$$S(p') - S(p) = (b-a)(S_b - S_a)$$

$$\left.\begin{array}{l} a < b \text{ طبق فرض} \Rightarrow b - a \geq 0 \\ S_a < S_b \text{ طبق فرض} \Rightarrow S_b - S_a \geq 0 \end{array}\right\} \xrightarrow{\text{ضرب ۲ عدد مثبت}} \geq 0$$

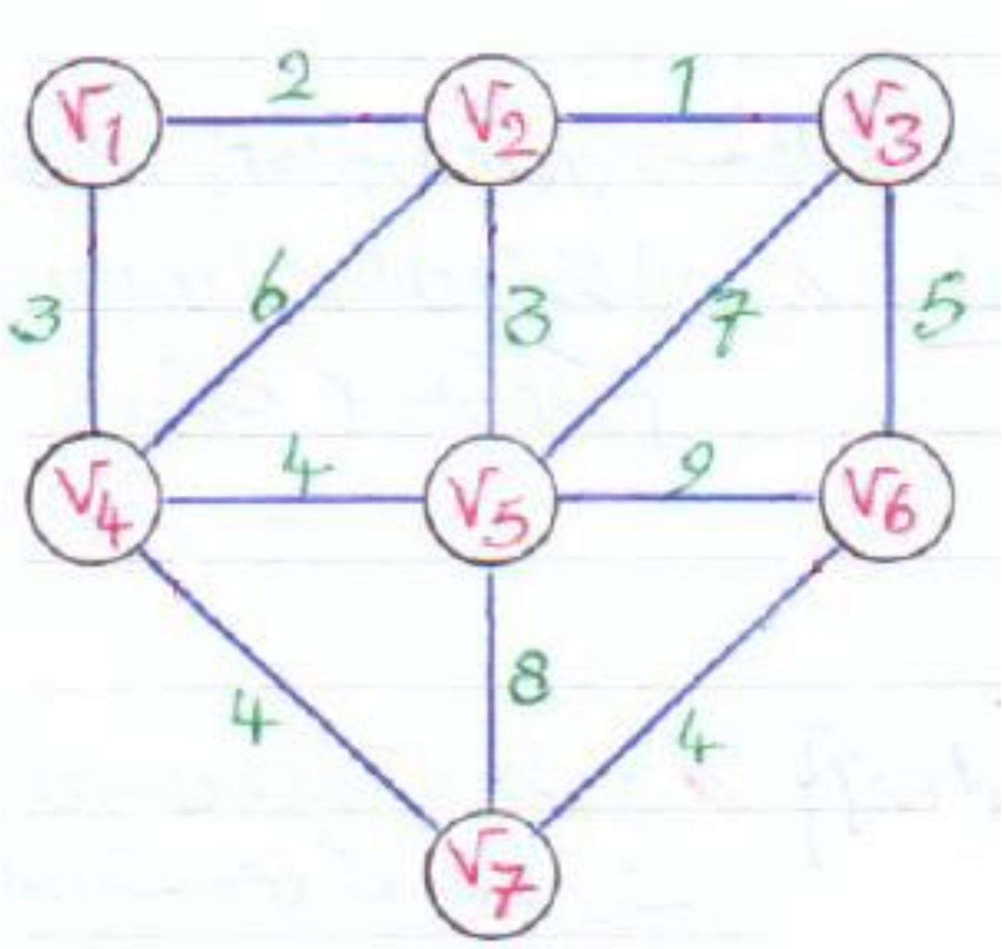
$$S(p') - S(p) \geq 0$$

$$S(p') \geq S(p)$$

- زمان اجرا: $\in \theta(n \log n)$

درخت پوشای کمینه (Minimum Spanning Tree)

درخت پوشای کمینه (Minimum Spanning Tree)



درخت پوشای کمینه (Minimum Spanning Tree)

تعریف) گراف وزن دار G متصل است که در آن V مجموعه رئوس و E مجموعه یال هاست $G=(V,E)$

و T زیر مجموعه ای از یال هاست $(T \subset E)$ T را مجموعه امید بخش می گویند اگر با اضافه کردن یال به مجموعه T

یک درخت پوشای کمینه بدست آید

درخت پوشای کمینه (Minimum Spanning Tree)

تعریف وزن‌دار $G=(V,E)$ مفروض است. اگر e یک یال باشد $(e \in E)$ و B زیرمجموعه‌ای از رئوس $(B \subset V)$
می‌گوییم e مجموعهٔ B را ترک می‌کند (leave) اگر تنها یک سر یال e در مجموعهٔ B باشد.

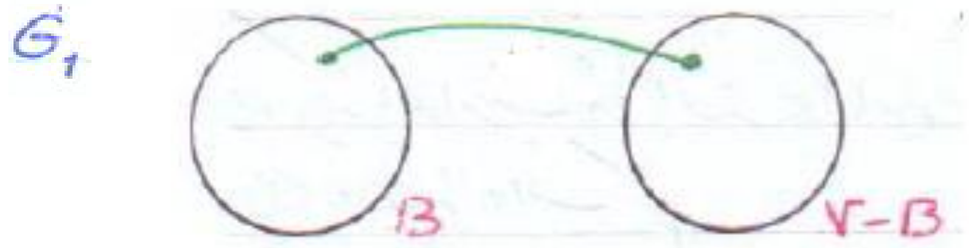
درخت پوشای کمینه (Minimum Spanning Tree)

لم: فرض کنید $G(V,E)$ موزون است. اگر B یک زیرمجموعه‌ای از رئوس باشد و T یک زیرمجموعه‌ای از یال‌ها و امیدبخش، به طوری که هیچ یالی از T مجموعه B را ترک نکند.
حال فرض کنید e کوچکترین یالی باشد که از مجموعه B خارج می‌شود (مجموعه B را ترک می‌کند)
در این صورت $T \cup \{e\}$ نیز امیدبخش است.

درخت پوشای کمینه (Minimum Spanning Tree)

لم: گراف $G(V,E)$ مفروض است. اگر B یک زیرمجموعه‌ای از رئوس باشد و T یک زیرمجموعه‌ای از یال‌ها و امیدبخش، به‌طوری‌که هیچ یالی از T مجموعه B را ترک نکند.
حال فرض کنید e کوچکترین یالی باشد که از مجموعه B خارج می‌شود (مجموعه B را ترک می‌کند)
در این صورت $T \cup \{e\}$ نیز امیدبخش است

اثبات:

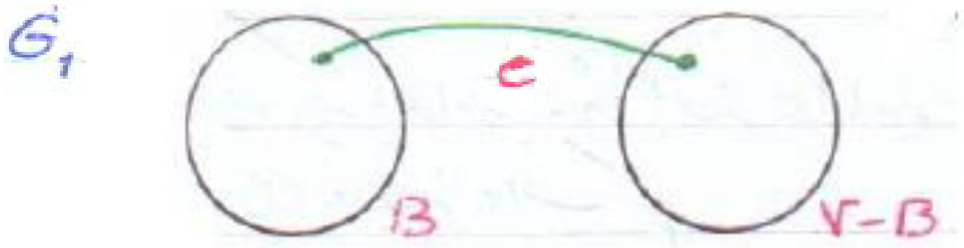


درخت پوشای کمینه (Minimum Spanning Tree)

لم: گراف $G(V,E)$ متصل و موزون است. اگر B یک زیرمجموعه‌ای از رئوس باشد و T یک زیرمجموعه‌ای از یال‌ها و امیدبخش، به‌طوری‌که هیچ یالی از T مجموعه B را ترک نکند.
حال فرض کنید e کوچکترین یالی باشد که از مجموعه B خارج می‌شود (مجموعه B را ترک می‌کند)
در این صورت $T \cup \{e\}$ نیز امیدبخش است.

اثبات:

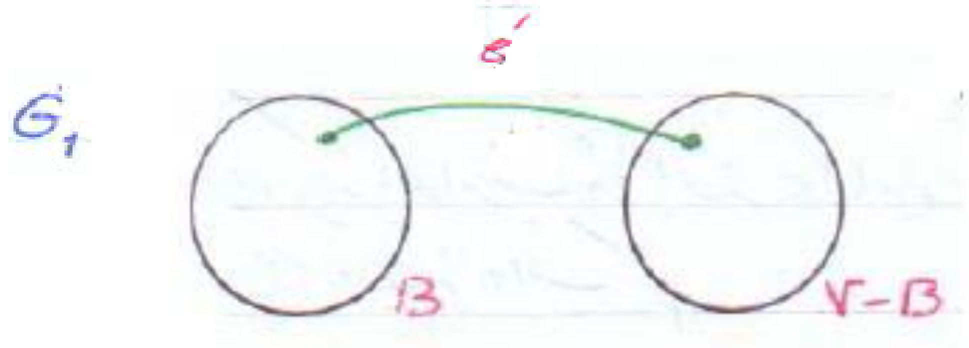
الف- اگر یال بین B و V-B همان e باشد.



درخت پوشای کمینه (Minimum Spanning Tree)

لم: فرض کنید $G(V,E)$ مفروض است. اگر B یک زیرمجموعه‌ای از رئوس باشد و T یک زیرمجموعه‌ای از یال ها و امیدبخش، بطوریکه هیچ یالی از T مجموعه B را ترک نکند.
حال فرض کنید e کوچکترین یالی باشد که از مجموعه B خارج می‌شود (مجموعه B را ترک می‌کند)
در اینصورت $T \cup \{e\}$ نیز امیدبخش است

اثبات:



ب- اگر یال بین B و V-B، یال دیگری مثل e′ باشد.

درخت پوشای کمینه (Minimum Spanning Tree)

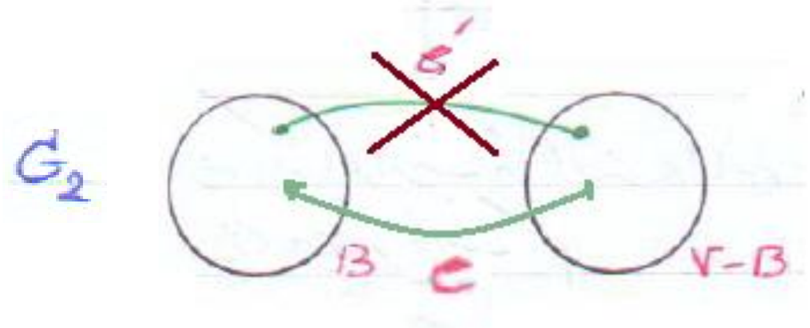
لم:

فرض کنید $G(V,E)$ موزون است. اگر B یک زیرمجموعه‌ای از رئوس باشد و T یک زیرمجموعه‌ای از یال‌ها و امیدبخش، به‌طوریکه هیچ یالی از T مجموعه B را ترک نکند.

حال فرض کنید e کوچکترین یالی باشد که از مجموعه B خارج می‌شود (مجموعه B را ترک می‌کند)

در اینصورت $T \cup \{e\}$ نیز امیدبخش است

اثبات:

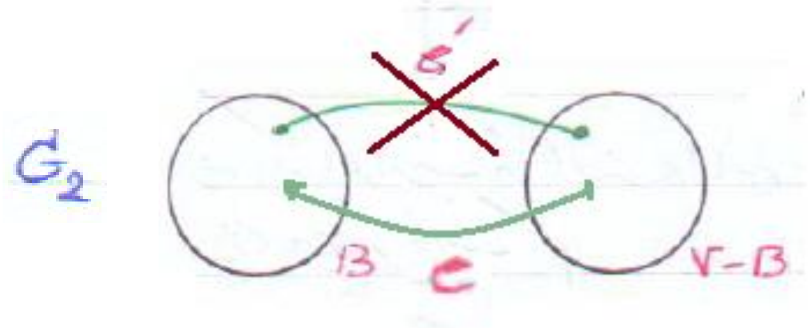


ب- اگر یال بین B و V-B، یال دیگری مثل ’e باشد.

درخت پوشای کمینه (Minimum Spanning Tree)

لم: گراف $G(V,E)$ مفروض است. اگر B یک زیرمجموعه‌ای از رئوس باشد و T یک زیرمجموعه‌ای از یال ها و امیدبخش، بطوریکه هیچ یالی از T مجموعه B را ترک نکند. حال فرض کنید e کوچکترین یالی باشد که از مجموعه B خارج می شود (مجموعه B را ترک می کند) در اینصورت $T \cup \{e\}$ نیز امیدبخش است

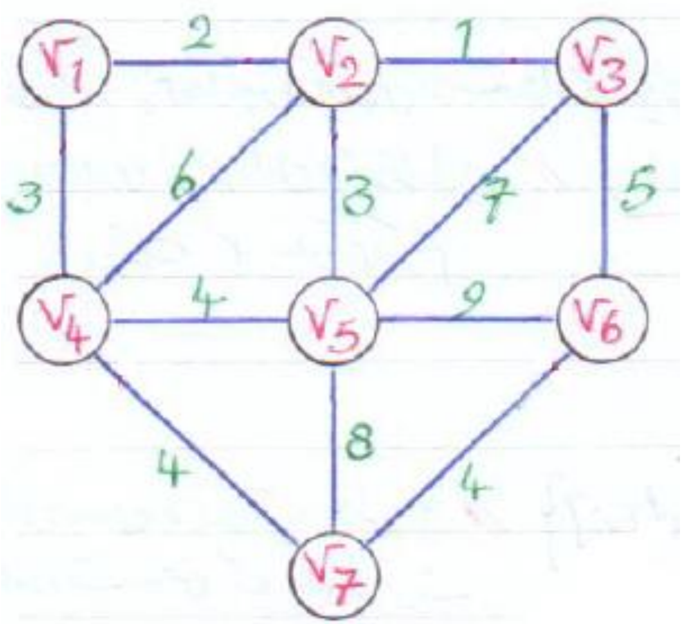
اثبات:



ب- اگر یال بین B و V-B، یال دیگری مثل e’ باشد.
چون وزن e از e’ بیشتر نیست پس وزن G2 از G1 بیشتر نیست
درنتیجه G2 یک درخت پوشای کمینه است که $T \cup \{e\}$
را شامل می شود.

پس $T \cup \{e\}$ امیدبخش است

درخت پوشای کمینه (Minimum Spanning Tree)



مرحله نخستین	—	$\{V_1\}\{V_2\}\{V_3\}\{V_4\}\{V_5\}\{V_6\}\{V_7\}$
مرحله اول	(V_2, V_3) ✓	$\{V_1\}\{V_2, V_3\}\{V_4\}\{V_5\}\{V_6\}\{V_7\}$
مرحله دوم	(V_1, V_2) ✓	$\{V_1, V_2, V_3\}\{V_4\}\{V_5\}\{V_6\}\{V_7\}$
مرحله سوم	(V_1, V_4) ✓	$\{V_1, V_2, V_3, V_4\}\{V_5\}\{V_6\}\{V_7\}$
مرحله چهارم	(V_2, V_5) ✓	$\{V_1, V_2, V_3, V_4, V_5\}\{V_6\}\{V_7\}$
مرحله پنجم	(V_4, V_5) ✗	بدون تغییر
مرحله ششم	(V_4, V_7) ✓	$\{V_1, V_2, V_3, V_4, V_5, V_7\}\{V_6\}$
مرحله هفتم	(V_6, V_7) ✓	$\{V_1, V_2, V_3, V_4, V_5, V_6, V_7\}$

الگوریتم Kruskal

```
kruskal ( G = (V,E)) {
    n = |V| ;  T = {} ;
    partition V into n Sets
    while |T| ≠ n-1
    {
        e = (u,v) ;  // کوچکترین یال
        if یال دیگری موجود نباشد return "No Solution"
        S1 = findSet(u);
        S2 = findSet(v);
        if S1 ≠ S2
        {
            T = T ∪ {e} ;
            MergeSet (S1,S2) ;
        }
    }
return T ;
}
```

الگوریتم Kruskal

- زمان اجرا:

```
kruskal (G=(V,E)) {
    n=|V| ; T={} ;
    partition V into n Sets
    while |T| ≠ n-1
    {
        e=(u,v) ; // کمترین یال
        if یال دیگری موجود نباشد return "No Solution"
        S1=findSet(u);
        S2=findSet(v);
        if S1 ≠ S2
        {
            T=T∪{e};
            MergeSet(S1,S2);
        }
    }
return T;
}
```

الگوریتم Kruskal

```
kruskal ( G=(V,E)) {
    n=|V| ;  T={}  ;
    partition V into n Sets
    while |T| ≠ n-1
    {
        e=(u,v) ; // کوچکترین یال
        if یال دیگری موجود نباشد return "No Solution"
        S1 = findSet(u);
        S2 = findSet(v);
        if S1 ≠ S2
        {
            T = T ∪ {e} ;
            MergeSet (S1,S2) ;
        }
    }
return T ;
}
```

- زمان اجرا:

$\in \Theta(a^2)$ if $|E|=a$

الگوریتم Kruskal

- بهبود زمان اجرا:

```
kruskal (G=(V,E)) {
    n=|V| ; T={} ;
    partition V into n Sets
    while |T| ≠ n-1
    {
        e=(u,v) ; // کوچکترین یال
        if یال دیگری موجود نباشد return "No Solution"
        S1=findSet(u);
        S2=findSet(v);
        if S1 ≠ S2
        {
            T=T∪{e};
            MergeSet(S1,S2);
        }
    }
return T;
}
```

الگوریتم Kruskal

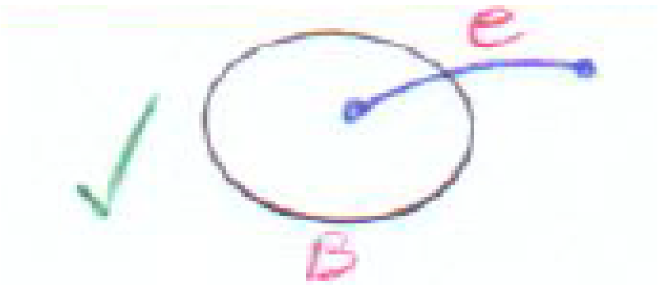
```
kruskal (G=(V,E)) {
    n=|V| ; T={} ; sort(E) ;
    partition V into n sets
    while |T| ≠ n-1
    {
        e=(u,v) ; // یال بعدی
        if یال دیگری موجود نباشد return "No Solution"
        S1=findSet(u);
        S2=findSet(v);
        if S1 ≠ S2
        {
            T=T∪{e};
            MergeSet(S1,S2);
        }
    }
return T;
}
```

- بهبود زمان اجرا:

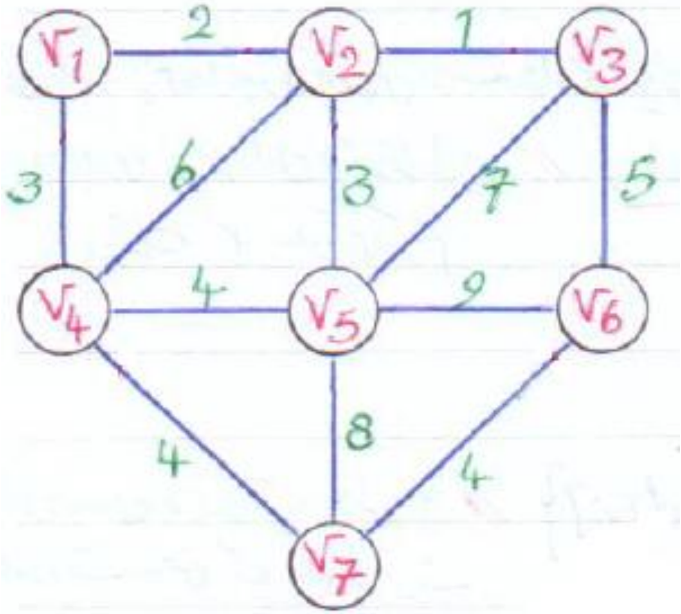
$\in \theta(a \log a)$

درخت پوشای کمینه (Minimum Spanning Tree)

الگوریتم پریم (Prim)



الگوریتم پریم



		B
مرحلهٔ اولیه	—	$\{V_1\}$
مرحلهٔ اول	(V_1, V_2) ✓	$\{V_1, V_2\}$
مرحلهٔ دوم	(V_2, V_3) ✓	$\{V_1, V_2, V_3\}$
مرحلهٔ سوم	(V_1, V_4) ✓	$\{V_1, V_2, V_3, V_4\}$
مرحلهٔ چهارم	(V_2, V_5) ✓	$\{V_1, V_2, V_3, V_4, V_5\}$
مرحلهٔ پنجم	(V_4, V_7) ✓	$\{V_1, V_2, V_3, V_4, V_5, V_7\}$
مرحلهٔ ششم	(V_6, V_7) ✓	$\{V_1, V_2, V_3, V_4, V_5, V_6, V_7\}$

الگوریتم پریم

```
Prim (G = (V,E)) {
  n = |V| ; T = {} ; B = {v1}
  Repeat n-1 Times
  {
      e = (u,v)     کوچکترین یال | u∈B , v∈V-B
      B = B U {v}
      T = T U {e}
  }
  return T;
}
```

الگوریتم پریم

Prim ($G = (V, E)$) {

$n = |V|$; $T = \{\}$; $B = \{V_1\}$

Repeat $n-1$ Times

{

$V_k =$ راسی که کمترین فاصله را از مجموعه B دارد

$B = B \cup \{V_k\}$

$T = T \cup \{e\}$

}

return T;

}

الگوریتم پریم

```
Prim ( G = (V,E)) {
n = |V| ; T = {} ; B = {V1}
Repeat n-1 Times
{
    Vk = رأسی که کمترین فاصله را از مجموعه B دارد

    B = B U {Vk}
    T = T U { یالی که باعث شده رأس Vk کمترین فاصله را از مجموعه B داشته باشد
}
return T;
}
```

الگوریتم پریم

MinDist[i] → کمترین فاصله راس i از مجموعه B

Nearest[i] → راسی از مجموعه B که کمترین فاصله را با راس i دارد

الگوریتم پریم

```
Prim (G = (V, E)) {
n = |V| ; T = {} ; B = {V1}
Repeat n-1 Times
{
    Vk = رأسی که کمترین فاصله را از مجموعه B دارد
    B = B U {Vk}
    T = T U {(Nearest[k], k)}
}
return T;
}
```

الگوریتم پریم

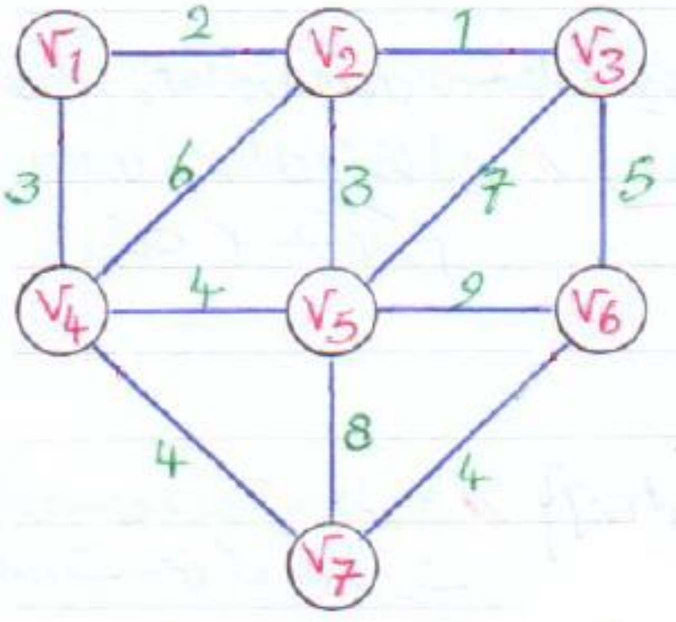
- مقادیر اولیه

$$Nearest[i] = V_1$$

$$MinDist[i] = L[1][i]$$

الگوریتم پریم(قسمت اول)

```
Prim ( n , L[1..n][1..n] ) {
  T = {} ;
  for  i=2  to  n  do
  {
      Nearest [i] = 1 ;
      MinDist [i] = L[1][i] ;
  }
  Repeat (n-1) Times
  {
      Min = ∞
      for  i=2  to  n  do
           if      MinDist [i] < Min
           {
                 Min = MinDist [i] ;
                 k = i ;
           }
      T = T ∪ { (Nearest [k] , k) }
```



```
Prim ( n , L[1..n][1..n]) {
  T = {} ;
  for  i=2  to  n  do
  {
       Nearest [i] = 1 ;
       MinDist [i] = L[1][i];
  }
  Repeat  (n-1)  Times
  {
       Min = ∞
       for   i=2   to   n   do
               if        MinDist [i] < Min
               {
                         Min = MinDist [i] ;
                         k = i ;
               }
       T = T ∪ { (Nearest [k] ,k) }

       for   i=2   to   n   do
               if  L[i][k] < MinDist[i]
               {
                         MinDist [i] = L[i][k] ;
                         Nearest [i] = k ;
               }
  }
  return  T ;
}
```

```
Prim ( n , L[1..n][1..n]) {
  T = {} ;
  for  i=2  to  n  do
  {
       Nearest [i] = 1 ;
       MinDist [i] = L[1][i];
  }
  Repeat  (n-1)  Times
  {
       Min = ∞
       for   i=2   to   n   do
            if  0 < Min Dist [i] < Min
            {
                 Min = MinDist [i];
                 k = i;
            }
       T = T ∪ { (Nearest [k] ,k) }
       MinDist [k] = -1 ;
       for   i=2   to   n   do
            if  L[i][k] < MinDist [i]
            {
                 MinDist [i] = L[i][k] ;
                 Nearest [i] = k ;
            }
  }
  return  T;
}
```

```
Prim ( n , L[1..n][1..n]) {
  T = {} ;
  for  i=2  to  n  do
  {
       Nearest [i] = 1 ;
       MinDist [i] = L[1][i];
  }
  Repeat  (n-1)  Times
  {
       Min = ∞
       for   i=2   to   n   do
             if   0 < Min Dist [i] < Min
             {
                   Min = MinDist [i] ;
                   k = i ;
             }
       T = T ∪ {(Nearest [k], k)}
       MinDist [k] = -1 ;
       for    i=2    to   n   do
             if  L[i][k] < MinDist [i]
             {
                   MinDist [i] = L[i][k] ;
                   Nearest [i] = k ;
             }
  }
  return  T ;
}
```

- زمان اجرا:

```
Prim ( n , L[1..n][1..n]) {
  T = {} ;
  for  i=2  to  n  do
  {
        Nearest [i] = 1 ;
        MinDist [i] = L[1][i];
  }
  Repeat  (n-1)  Times
  {
        Min = ∞
        for  i=2  to  n  do
             if  0 < Min Dist [i] < Min
             {
                  Min = MinDist [i] ;
                  k = i ;
             }
        T = T ∪ { (Nearest [k], k) }
        MinDist [k] = -1 ;
        for  i=2  to  n  do
             if  L[i][k] < MinDist [i]
             {
                  MinDist [i] = L[i][k] ;
                  Nearest [i] = k ;
             }
  }
  return  T ;
}
```

- زمان اجرا:

$$(n-1)+(n-1)\left[(n-1)+(n-1)\right]$$

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

مقایسه Kruskal و Prim

- Kruskal $\in \theta(a \log a)$

- Prim $\in \theta(n^2)$

مقایسه Kruskal و Prim

- Kruskal $\in \theta(a \log a)$
- Prim $\in \theta(n^2)$

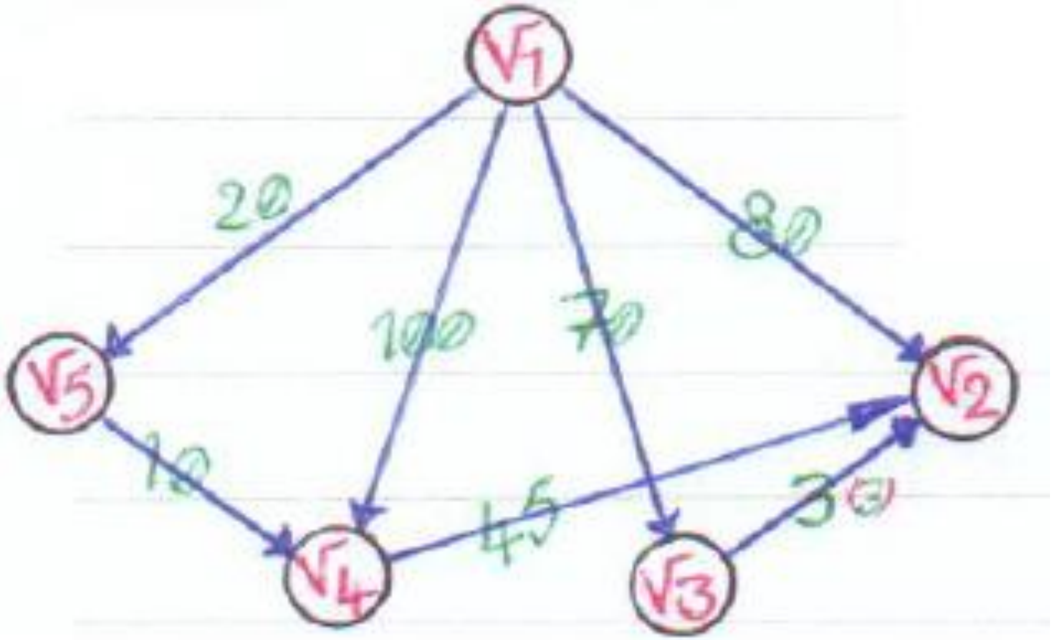
$$n-1 \leq a \leq \frac{n(n-1)}{2}$$

تمرین

- الگوریتم Kruskal را روی یک گراف با ۸ راس اجرا نمایید و درخت پوشای کمینه را مشخص کنید.

- الگوریتم Prim با جزییات پیاده سازی را روی یک گراف با ۸ راس اجرا نمایید و درخت پوشای کمینه را مشخص کنید.

Single Source shortest path : کوتاهترین مسیرها برای یک مبدأ



Dijkstra (G = <V, E>) {

$B = \{V_1\}$; $T = \{\}$

Repeat $(n-1)$ Times

{

V_k = رأسی که کمترین فاصله را از V_1 دارد با کمک رئوس مجموعه B

$B = B \cup \{V_k\}$

$T = T \cup \{$ یالی که باعث شد رأس V_k از طریق مجموعه B در کمترین باشد $\}$

}

return T;

}

len[i] → طول کوتاه‌ترین مسیر از v_1 به v_i با کمک مجموعه B

Touch[i] → یک راس است از مجموعه B که در مسیر از v_1 به v_i درست قبل از v_i قرار می‌گیرد

```
Dijkstra (G = <V, E>) {
  B = {V1} ; T = {}
  Repeat (n-1) Times
  {
        Vk = رأسی که کمترین فاصله را از V1 دارد با کمک رئوس مجموعه B
        B = B ∪ {Vk}
        T = T ∪ (Touch[k], Vk)
  }
  return T;
}
```

- مقدار اولیه

```
len[i] = L[1][i];
Touch[i] = 1;
```

الگوریتم Dijkstra (بخش اول)

```
Dijkstra ( n, L[1..n][1..n] ) {
  T = {};
  for  i=2  to  n  do
  {
        len[i] = L[1][i];
        Touch[i] = 1;          }
  Repeat (n-1) Times
  {
        Min = ∞
        for   i=2   to   n   do
                if      len[i] < Min
                {
                        Min = len[i] ;
                        k = i ;
                }
        T = T ∪ {(Touch[k], k)}
```

```
Dijkstra (n, L[1..n][1..n]) {
   T = {};
   for i=2 to n do
   {
        len[i] = L[1][i];
       Touch[i] = 1;          }
   Repeat (n-1) Times
   {
        Min = ∞
        for i=2 to n do
             if     len[i] < Min
             {
                      Min = len[i];
                      k = i;
             }
       T = T ∪ {(Touch[k], k)}
        for i=2 to n do
             if len[k]+L[k][i] < len[i]
             {
                  len[i] = len[k]+L[k][i]
                  Touch[i] = k
             }
   }
   return T;
}
```

```
Dijkstra ( n, L[1..n][1..n] ) {
  T = {};
  for  i=2  to  n  do
  {
        len[i] = L[1][i];
        Touch[i] = 1;          }
  Repeat (n-1) Times
  {
        Min = ∞
        for  i=2  to  n  do
              if  0 < len[i] < Min
              {
                    Min = len[i] ;
                    k = i ;
              }
        T = T ∪ {(Touch[k], k)}
        for  i=2  to  n  do
              if  len[k] + L[k][i] < len[i]
              {
                    len[i] = len[k] + L[k][i]
                    Touch[i] = k
              }
        len[k] = -1;
  }
  return T;
}
```

```
Dijkstra(n, L[1..n][1..n]) {
    T = {};
    for i=2 to n do
    {
        len[i] = L[1][i];
        Touch[i] = 1;
    }
    Repeat (n-1) Times
    {
        Min = ∞
        for i=2 to n do
            if 0 < len[i] < Min
            {
                Min = len[i];
                k = i;
            }
        T = T ∪ {(Touch[k], k)}
        for i=2 to n do
            if len[k] + L[k][i] < len[i]
            {
                len[i] = len[k] + L[k][i]
                Touch[i] = k
            }
        len[k] = -1;
    }
    return T;
}
```

- زمان اجرا:

$$(n-1)+(n-1)\left[(n-1)+(n-1)\right] \in \theta(n^2)$$

تمرین

- الگوریتم Dijkstra با جزییات پیاده سازی را روی یک گراف با 6 راس اجرا نمایید و یک یا دو مسیر بهینه را مشخص کنید.