فصل چهارم
رهیافت حریصانه

سید ناصر رضوی
E-mail: razavi@Comp.iust.ac.ir

۱۳۸۵

ایده

- رهیافت اسکروج
- عناصر داده ای را به ترتیب انتخاب کن، هر بار «بهترین» انتخاب را انجام بده، بدون توجه به انتخاب های قبلی و انتخاب هایی که در آینده انجام خواهند گرفت.
- اغلب در مسائل بهینه سازی بکار می رود. (مانند برنامه نویسی پویا)
- باید مشخص شود که با دنباله ای از راه حل های بهینه محلی، یک راه حل بهینه سراسری بدست می آید.

مثال: باقیمانده پول

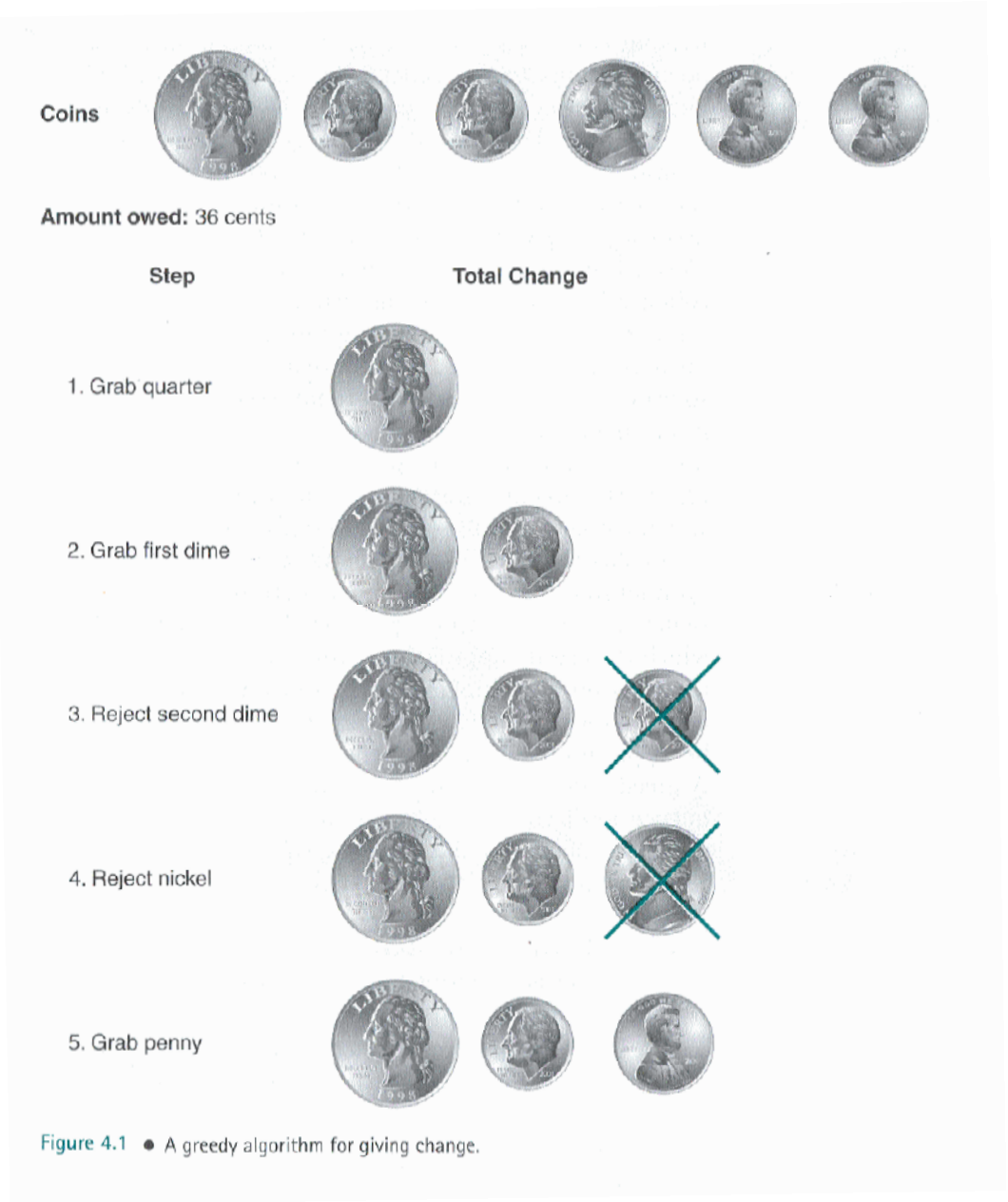


Figure 4.1 • A greedy algorithm for giving change.

الگوریتم

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

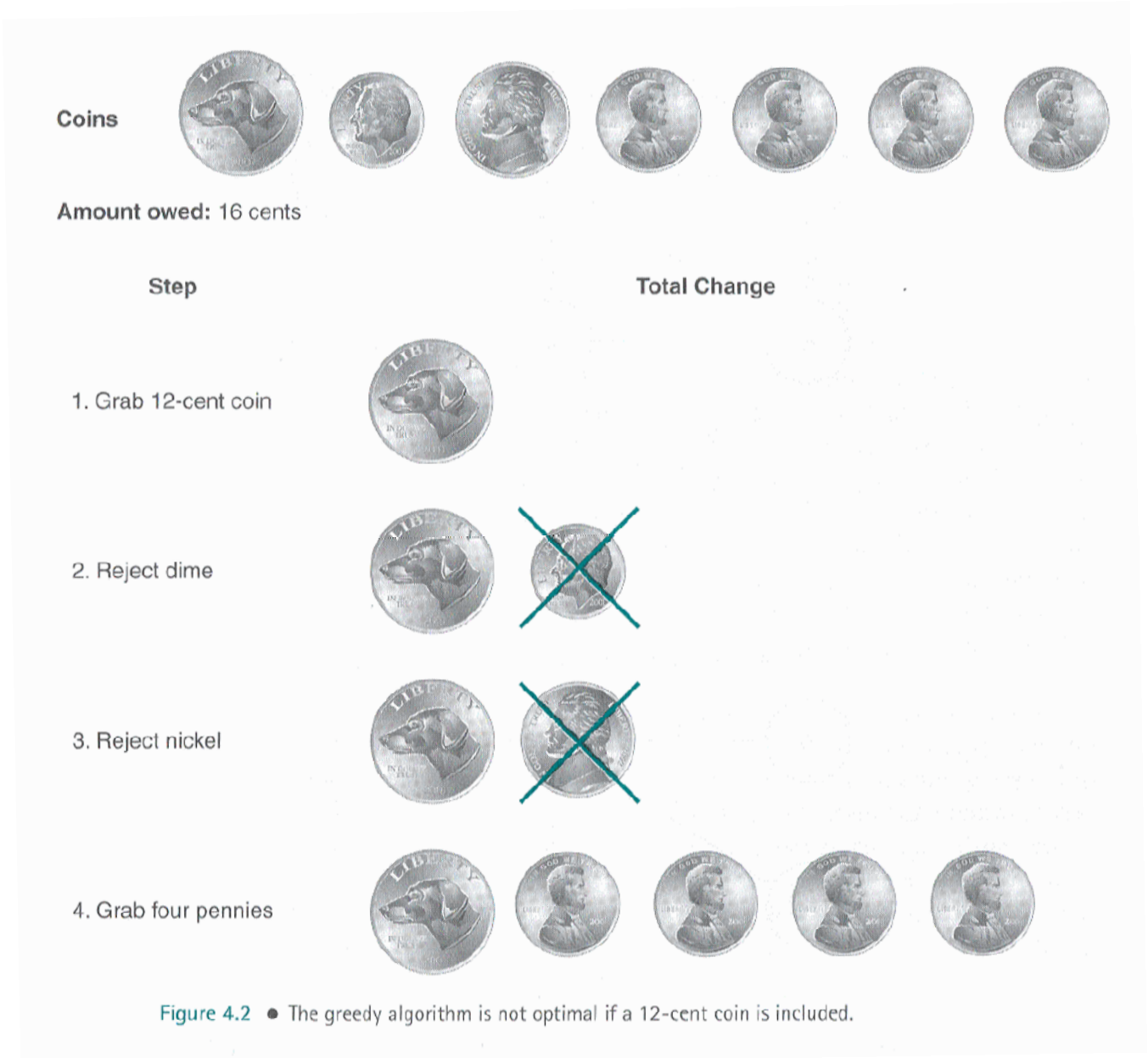


Figure 4.2 • The greedy algorithm is not optimal if a 12-cent coin is included.

شکست
رهیافت
حریصانه

اضافه کردن یک عنصر به مجموعه

- **روال انتخاب**، عنصر بعدی را که باید به مجموعه اضافه شود، انتخاب می کند. انتخاب براساس یک ملاک حریصانه انجام می شود که برخی شرایط بهینه محلی را در زمان انتخاب برآورده می سازد.
- **بررسی امکان سنجی**، تعیین می کند که آیا مجموعه جدید برای رسیدن به حل نمونه عملی است یا خیر.
- **بررسی راه حل**، بررسی می کند که آیا مجموعه جدید، راه حلی برای نمونه مساله می باشد یا خیر.

درخت های پوشای کمینه (*MST*)

- برخی اصطلاحات:
  - گراف بدون جهت
  - مسیر
  - گراف همبند
  - دور ساده
  - گراف بدون دور
  - درخت (درخت آزاد)
  - درخت ریشه دار

مثال ها

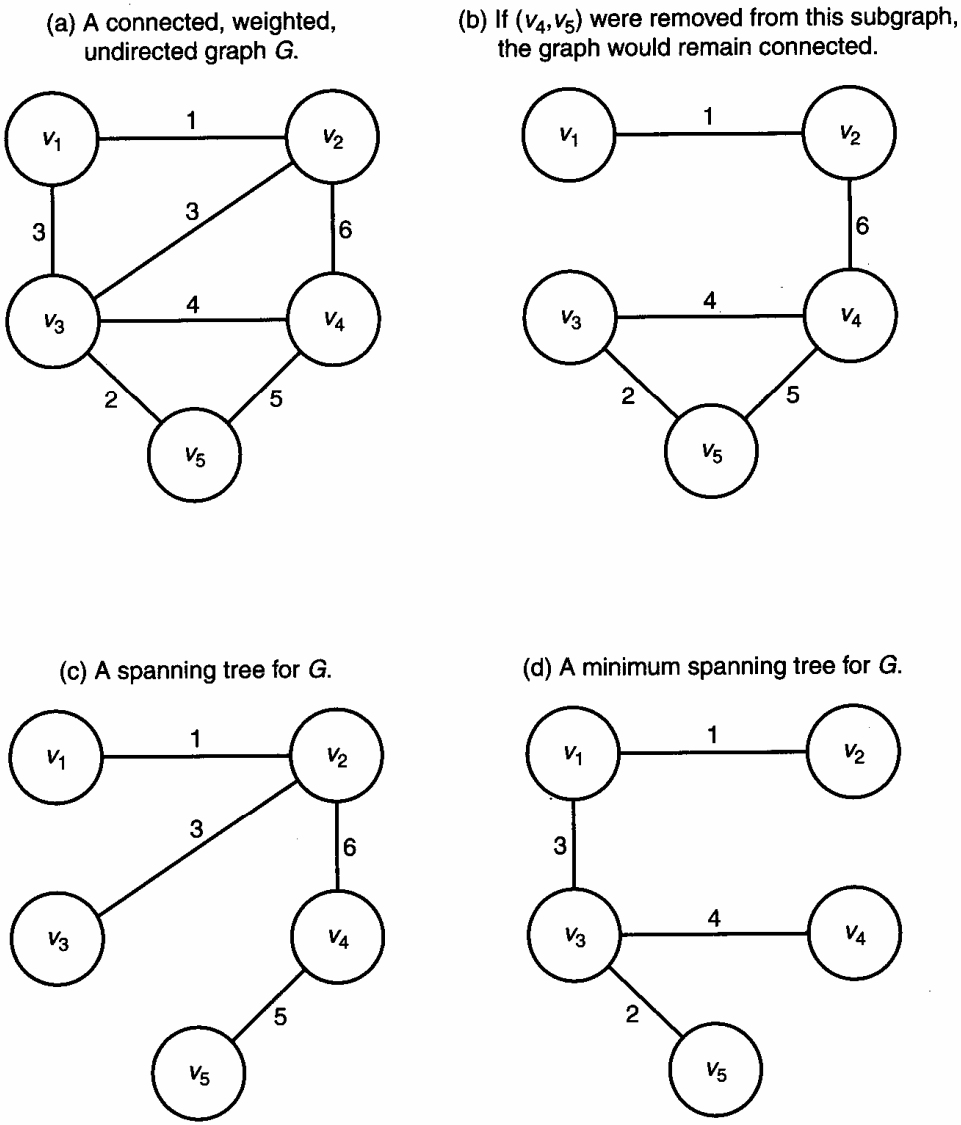


Figure 4.3 • A weighted graph and three subgraphs.

درخت پوشای کمینه

- درخت پوشا
- درخت پوشای کمینه
- تعریف رسمی گراف بدون جهت

تعریف:

یک **گراف بدون جهت** G شامل یک مجموعه متناهی و غیر تهی V می باشد که عناصر آن را رئوس گراف G می نامیم، به همراه مجموعه E که شامل مجموعه ای از زوج رئوس (یال) در V می باشد.

$$G = (V, E)$$

یافتن درخت پوشای کمینه

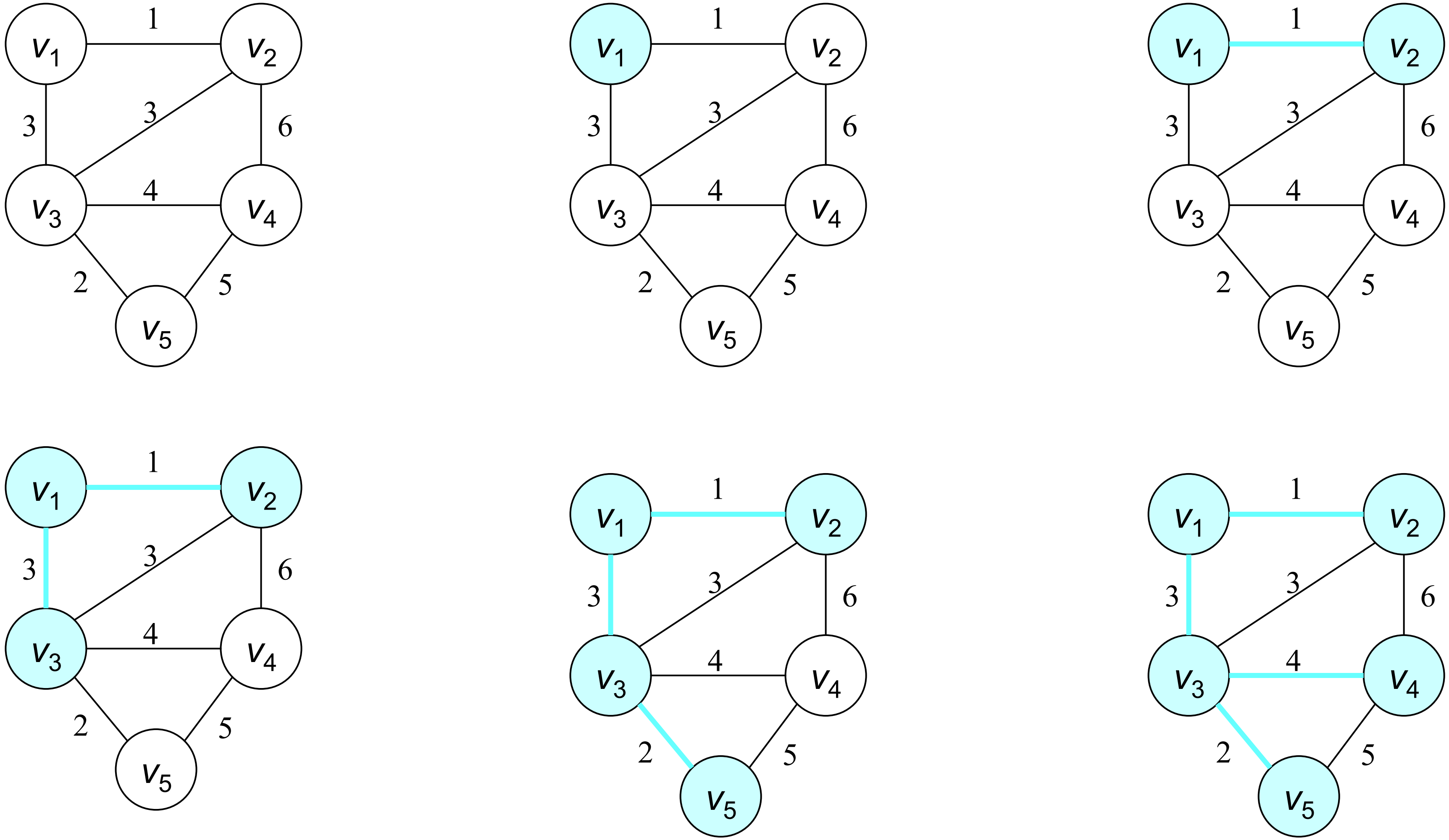
- برای یافتن $T = (V, F)$ کمینه برای $G = (V, E)$

```
F = ∅                                                    // Initialize set of
                                                         // edges to empty.
while (the instance is not solved){

  select an edge according to some locally
  optimal consideration;                                 // selection procedure

  if (adding the edge to F does not create a cycle)
  add it;                                                // feasibility check

  if (T = (V,F) is a spanning tree)                      // solution check
  the instance is solved;
}
```

الگوریتم پریم

```
F = ∅;                                  // Initialize set of edges
                                        // to empty.
Y = {v1};                               // Initialize set of vertices to
                                        // contain only the first one.
while (the instance is not solved){

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

  add the vertex to Y;
  add the edge to F;

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

یک مثال


v_1 v_2 v_3 v_4 v_5
1 3 3 6 4 2 5
v_1 v_2 v_3 v_4 v_5
1 3 3 6 4 2 5
v_1 v_2 v_3 v_4 v_5
1 3 3 6 4 2 5
v_1 v_2 v_3 v_4 v_5
1 3 3 6 4 2 5
v_1 v_2 v_3 v_4 v_5
1 3 3 6 4 2 5
v_1 v_2 v_3 v_4 v_5
1 3 3 6 4 2 5

ماتریس مجاورتی

$$W[i][j] = \begin{cases} \text{weight on edge} & \text{if there is an edge between } v_i \text{ and } v_j \\ \infty & \text{if there is no edge between } v_i \text{ and } v_j \\ 0 & \text{if } i = j. \end{cases}$$

	1	2	3	4	5
1	0	1	3	∞	∞
2	1	0	3	6	∞
3	3	3	0	4	2
4	∞	6	4	0	5
5	∞	∞	2	5	0

Figure 4.5 ● The array representation *W* of the graph in Figure 4.3(a).

الگوریتم پریم

$nearest[i]$ = index of the vertex in Y nearest to v_i

$distance[i]$ = weight on edge between v_i and the vertex indexed by $nearest[i]$

▶ Algorithm 4.1 **Prim's Algorithm**

Problem: Determine a minimum spanning tree.

Inputs: integer $n \geq 2$, and a connected, weighted, undirected graph containing n vertices. The graph is represented by a two-dimensional array W, which has both its rows and columns indexed from 1 to n, where $W[i][j]$ is the weight on the edge between the ith vertex and the jth vertex.

Outputs: set of edges F in a minimum spanning tree for the graph.

```
void prim (int n,
           const number W[][],
           set_of_edges& F)
{
  index i, vnear;
  number min;
  edge e;
  index nearest[2..n];
  number distance[2..n];

  F = ∅;
  for (i = 2; i <= n; i++){
     nearest[i] = 1;          // For all vertices, initialize v1
     distance[i] = W[1][i];   // to be the nearest vertex in
  }                           // Y and initialize the distance
                              // from Y to be the weight
                              // on the edge to v1.
```

الگوریتم پریم (ادامه)

```
repeat (n - 1 times){
                                          // Add all n - 1 vertices to Y.
    min = ∞;
    for (i = 2; i <= n; i++)              // Check each vertex for
        if (0 ≤ distance[i] < min){       // being nearest to Y.
            min = distance[i];
            vnear = i;
        }
    e = edge connecting vertices indexed
        by vnear and nearest[vnear];
    add e to F;
    distance[vnear] = -1;                 // Add vertex indexed by
    for (i = 2; i <= n; i++)              // vnear to Y.
        if (W[i][vnear] < distance[i]){   // For each vertex not in
            distance[i] = W[i][vnear];    // Y, update its distance
            nearest[i] = vnear;           // from Y.
        }
  }
}
```

پیچیدگی زمانی برای همه حالات

- عمل اصلی: در حلقه *repeat* دو حلقه وجود دارد که هر یک از آنها $(n-1)$ بار تکرار می شود، اجرای دستورات داخل هریک از آنها را می توان به عنوان یک بار اجرای عمل اصلی در نظر گرفت.
- اندازه ورودی: n، تعداد رئوس
- پیچیدگی زمانی:

چون حلقه *repeat* به تعداد $(n-1)$ بار تکرار می شود، پیچیدگی زمانی برابر است با:

$$T(n) = 2(n-1)(n-1) = \Theta(n^2)$$

اثبات

▲ **Lemma 4.1** Let $G = (V, E)$ be a connected, weighted, undirected graph; let F be a promising subset of E; and let Y be the set of vertices connected by the edges in F. If e is an edge of minimum weight that connects a vertex in Y to a vertex in $V - Y$, then $F \cup \{e\}$ is promising.

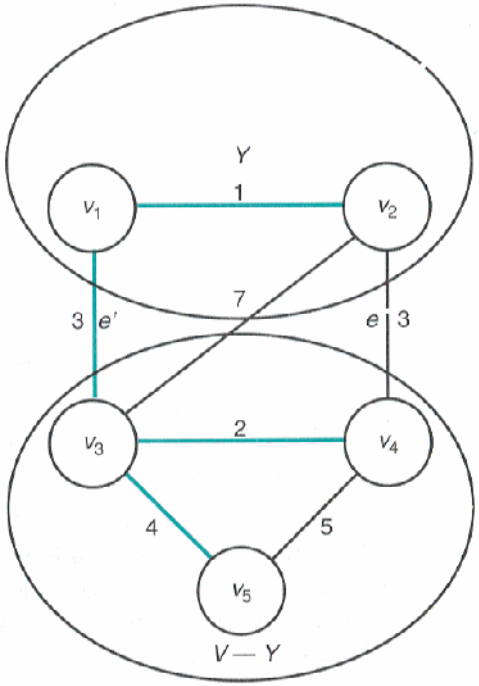


Figure 4.6 ● A graph illustrating the proof in Lemma 4.1. The edges in F' are shaded in color.

قضیه ۴-۱

◀ قضیه ۴-۱ – الگوریتم پریم همواره یک درخت پوشای کمینه ایجاد می کند.

اثبات: برای آنکه نشان دهیم F پس از هر بار تکرار حلقه *repeat* امید بخش است از استقراء استفاده می کنیم.

مبنای استقراء: واضح است که مجموعه $\varnothing$ امید بخش است.

فرض استقراء: فرض کنید پس از یک بار تکرار حلقه *repeat* مجموعه یال های بدست آمده F امید بخش است.

گام استقراء: باید نشان دهیم مجموعه $F \cup \{e\}$ که e یال انتخابی در تکرار بعدی است، امید بخش است. چون یال e کمینه است و یک راس از V را به راسی در $V - F$ متصل می کند، طبق لم 4.1 نتیجه می گیریم که $F \cup \{e\}$ امید بخش می باشد.

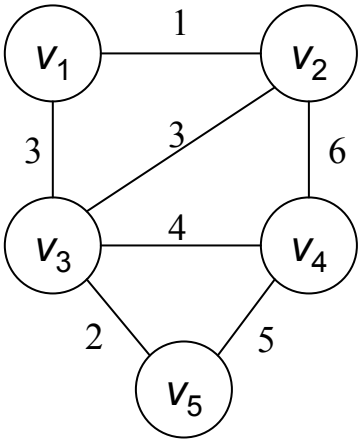
بنابراین مجموعه نهایی F امید بخش می باشد و چون شامل یال های یک درخت می باشد، آن درخت باید یک درخت پوشای کمینه باشد.

الگوریتم کروسکال

```
F = ∅;                                          // Initialize set of
                                                // edges to empty.
create disjoint subsets of V, one for each
vertex and containing only that vertex;

sort the edges in E in nondecreasing order;


while (the instance is not solved){

  select next edge;                             // selection procedure

  if (the edge connects two vertices in         // feasibility check
                    disjoint subsets){
      merge the subsets;
      add the edge to F;
  }
  if (all the subsets are merged)               // solution check
     the instance is solved;
}
```

یک مثال

۱. مرتب سازی یال ها بر حسب طول



(v_1, v_2) 1

(v_3, v_5) 2

(v_1, v_3) 3

(v_2, v_3) 3

(v_3, v_4) 4

(v_4, v_5) 5

(v_2, v_4) 6

یک مثال

۲. ایجاد مجموعه های مجزا

(v_1, v_2) 1

(v_3, v_5) 2

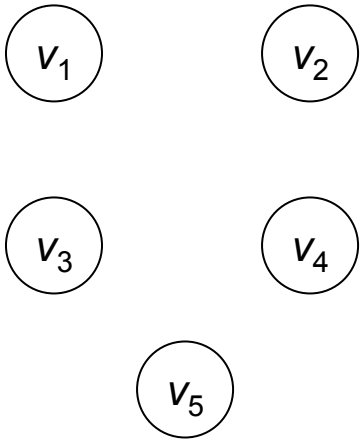
(v_1, v_3) 3

(v_2, v_3) 3

(v_3, v_4) 4

(v_4, v_5) 5

(v_2, v_4) 6



یک مثال

۳. انتخاب یال (v_1, v_2)

(v_1, v_2) 1

(v_3, v_5) 2

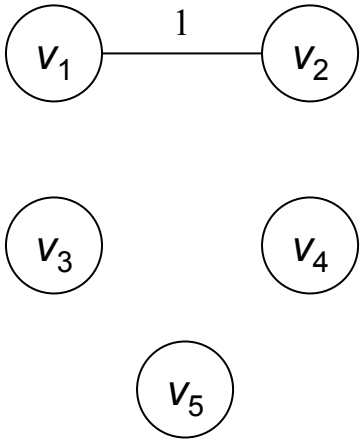
(v_1, v_3) 3

(v_2, v_3) 3

(v_3, v_4) 4

(v_4, v_5) 5

(v_2, v_4) 6



یک مثال

۴. انتخاب یال (v_3, v_5)

(v_1, v_2) 1

(v_3, v_5) 2

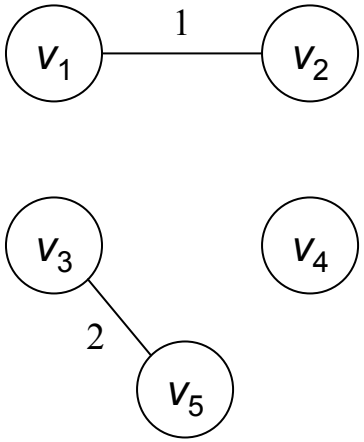
(v_1, v_3) 3

(v_2, v_3) 3

(v_3, v_4) 4

(v_4, v_5) 5

(v_2, v_4) 6



یک مثال

5. انتخاب یال (v_1, v_3)

(v_1, v_2) 1

(v_3, v_5) 2

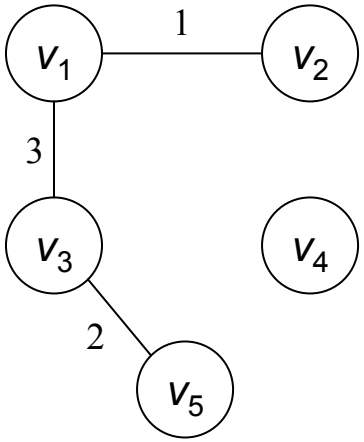
(v_1, v_3) 3

(v_2, v_3) 3

(v_3, v_4) 4

(v_4, v_5) 5

(v_2, v_4) 6



یک مثال

6. انتخاب یال (v_2, v_3)

(v_1, v_2) 1

(v_3, v_5) 2

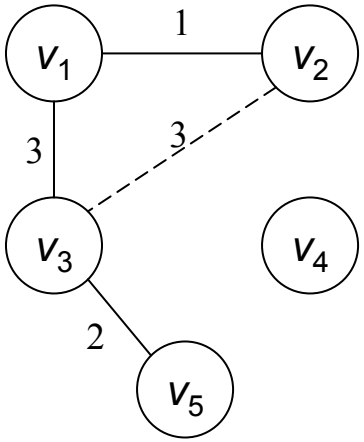
(v_1, v_3) 3

(v_2, v_3) 3

(v_3, v_4) 4

(v_4, v_5) 5

(v_2, v_4) 6



یک مثال

7. انتخاب یال (v_3, v_4)

(v_1, v_2) 1

(v_3, v_5) 2

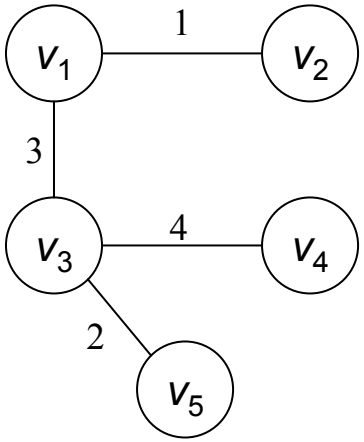
(v_1, v_3) 3

(v_2, v_3) 3

(v_3, v_4) 4

(v_4, v_5) 5

(v_2, v_4) 6



انواع داده ای و عملیات

- انواع داده ای:
  - index i;
  - set_pointer p, q;
- عملیات:
  - $initial(n)$: n زیرمجموعه مجزا ایجاد می کند که هر یک حاوی دقیقا یکی از اندیس های ۱ تا n می باشد.
  - $p = find(i)$: باعث می شود که p به مجموعه حاوی اندیس i اشاره کند.
  - $merge(p, q)$: دو مجموعه ای را که p و q به آنها اشاره می کنند، ادغام می کند.
  - $equal(p, q)$: اگر p و q هر دو به یک مجموعه اشاره کنند $true$ برمی گرداند.

الگوریتم کروسکال

Algorithm 4.2 Kruskal's Algorithm

Problem: Determine a minimum spanning tree.

Inputs: integer $n \geq 2$, positive integer m, and a connected, weighted, undirected graph containing n vertices and m edges. The graph is represented by a set E that contains the edges in the graph along with their weights.

Outputs: F, a set of edges in a minimum spanning tree.

```
void kruskal (int n, int m,
              set_of_edges E,
              set_of_edges& F)
{
  index i, j;
  set_pointer p, q;
  edge e;

  Sort the m edges in E by weight in nondecreasing order;
  F = ∅;
  initial(n);                    // Initialize n disjoint subsets.
  while (number of edges in F is less than n - 1){
    e = edge with least weight not yet considered;
    i, j = indices of vertices connected by e;
    p = find(i);
    q = find(j);
    if (! equal(p, q)){
       merge(p, q);
       add e to F;
    }
  }
}
```

پیچیدگی زمانی در بدترین حالت

- عمل اصلی: یک دستورالعمل مقایسه
- اندازه ورودی: n، تعداد رئوس و m تعداد یال ها
- تحلیل:
  - زمان لازم برای مرتب سازی یال ها: $W(m) \in \Theta(m\lg m)$
  - زمان در حلقه while: $W(m) \in \Theta(m\lg m)$
  - زمان لازم برای ایجاد n مجموعه مجزا: $T(n) \in \Theta(n)$
  - در کل:

$$W(m, n) \in \Theta(m\lg m) = \Theta(n^2 lgn) \text{ (when } m = n(n\text{-}1)/2)$$

اثبات

▲ لم 4.2 فرض کنید $G = (V, E)$ یک گراف بدون جهت ، همبند و وزن دار باشد، فرض کنید F یک زیر مجموعه امید بخش از E باشد و فرض کنید e یالی با وزن کمینه در $E - F$ باشد به طوری که $F \cup \{e\}$ فاقد دور باشد. در آن صورت $F \cup \{e\}$ امید بخش است.

◄ قضیه 4.2 الگوریتم کروسکال همواره یک درخت پوشای کمینه ایجاد می کند.

مقایسه الگوریتم های پریم و کروسکال

- الگوریتم پریم: $T(n) \in \Theta(n^2)$
- الگوریتم کروسکال: $W(m, n) \in \Theta(m\lg m) = \Theta(n^2 lgn)$
- در هر انتهای بازه زیر

$$n - 1 \leq m \leq n(n-1)/2$$

- اگر تعداد یال ها نزدیک به کرانه پایینی باشد، الگوریتم کروسکال $\Theta(nlgn)$ است و باید سریعتر از پریم باشد.
- اگر تعداد یال ها نزدیک به کرانه بالایی باشد، الگوریتم کروسکال $\Theta(n^2 lgn)$ است ، یعنی الگوریتم پریم باید سریعتر باشد.

الگوریتم دیکسترا برای کوتاه ترین مسیر تک مبدا

```
Y = {v1};
F = ∅;

while (the instance is not solved){

  select a vertex v from V − Y, that has a    // selection
  shortest path from v1, using only vertices  // procedure and
  in Y as intermediates;                      // feasibility check

  add the new vertex v to Y;
  add the edge (on the shortest path) that touches v to F;

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

مثال

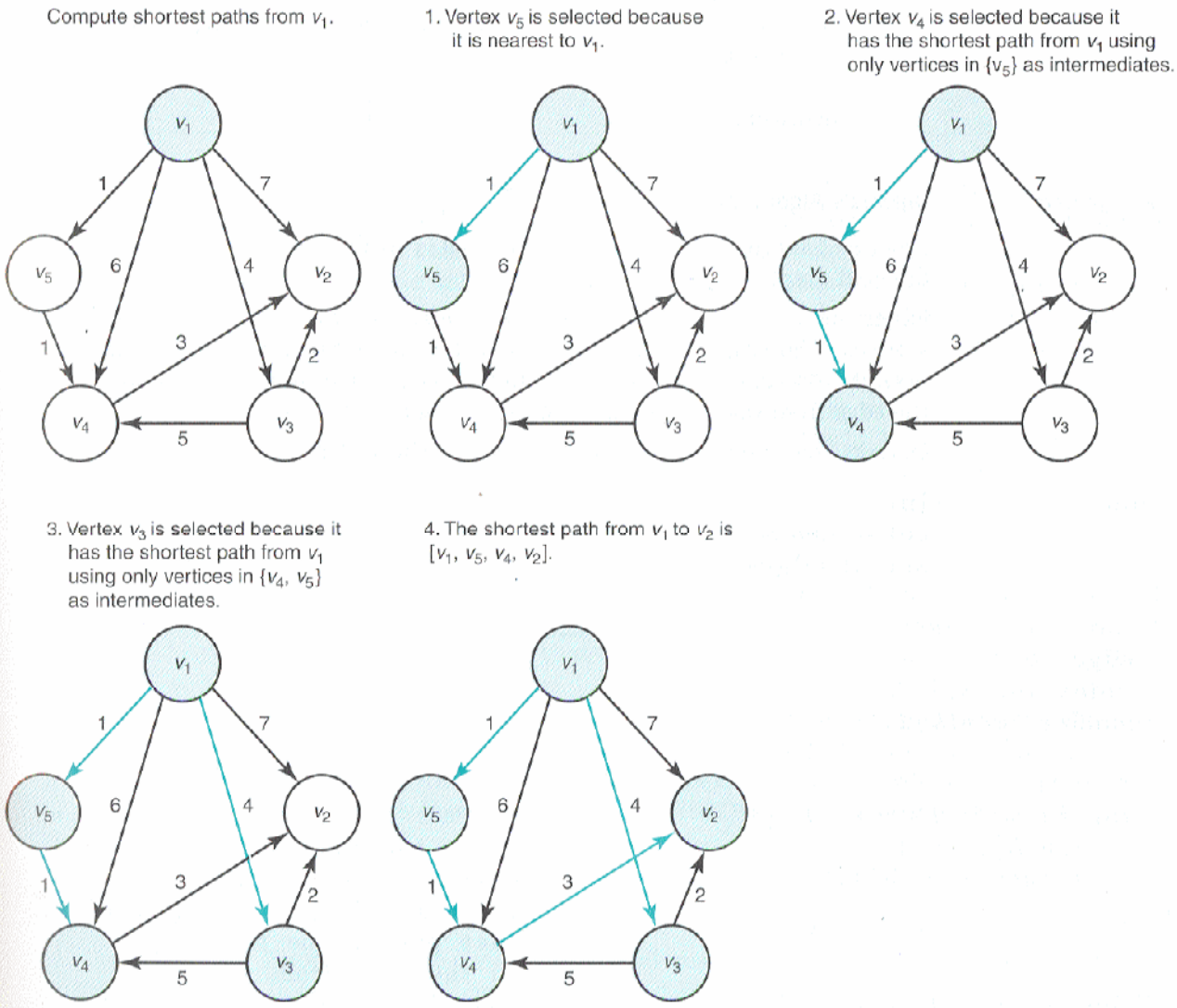


Figure 4.8 • A weighted, directed graph (in upper-left corner) and the steps in Dijkstra's algorithm for that graph. The vertices in Y and the edges in F are shaded in color at each step.

آرایه های کمکی

- $Touch\ [i]$ =

اندیس راس v در Y به طوری که یال $<v, v_i>$ آخرین یال روی کوتاهترین مسیر فعلی از v_1 به v_i که فقط از رئوس Y به عنوان رئوس میانی استفاده می کند، می باشد.

- $Length\ [i]$ =

طول کوتاهترین مسیر فعلی از v_1 به v_i که فقط از رئوس Y به عنوان رئوس میانی استفاده می کند، می باشد.

الگوریتم

► Algorithm 4.3 **Dijkstra's Algorithm**

Problem: Determine the shortest paths from v_1 to all other vertices in a weighted, directed graph.

Inputs: integer $n \geq 2$, and a connected, weighted, directed graph containing n vertices. The graph is represented by a two-dimensional array W, which has both its rows and columns indexed from 1 to n, where $W[i][j]$ is the weight on the edge from the ith vertex to the jth vertex.

Outputs: set of edges F containing edges in shortest paths.

```
void dijkstra (int n,
               const number W[][],
               set_of_edges& F)
{
  index i, vnear;
  edge e;
  index touch[2..n];
  number length[2..n];

  F = ∅;
  for (i = 2; i <= n; i++){          // For all vertices, initialize v1
     touch[i] = 1;                   // to be the last vertex on the
     length[i] = W[1][i];            // current shortest path from
  }                                  // v1, and initialize length of
                                     // that path to be the weight
                                     // on the edge from v1.
  repeat (n - 1 times){              // Add all n - 1 vertices to Y.
     min = ∞;
     for (i = 2; i <= n; i++)        // Check each vertex for
        if (0 ≤ length[i] < min){    // having shortest path.
           min = length[i];
           vnear = i;
        }
     e = edge from vertex indexed by touch[vnear]
         to vertex indexed by vnear;
     add e to F;
     for (i = 2; i <=n; i++)
        if (length[vnear] + W[vnear][i] < length[i]){
           length[i] = length[vnear] + W[vnear][i];
           touch[i] = vnear;         // For each vertex not in Y,
        }                            // update its shortest path.

     length[vnear] = -1;             // Add vertex indexed by vnear
  }                                  // to Y.
}
```

پیچیدگی زمانی برای همه حالات

$$T(n) = 2(n\text{-}1)^2 \in (n^2)$$

زمانبندی - *scheduling*

- دو گونه زمان بندی:
  - کمینه سازی کل زمان برای انتظار کشیدن و سرویس دهی (زمان بودن در سیستم– *time in the system*)
    - مثال: آرایشگاه
  - زمانبندی با مهلت معین – *scheduling with deadlines*

کمینه سازی کل زمان بودن در سیستم

- **مثال 4.2**

$$t_1 = 5,\ t_2 = 10,\ \text{and}\ t_3 = 4$$

کار	زمان بودن در سیستم
۱	۵ (زمان سرویس دهی)
۲	۵ (انتظار برای انجام کار ۱) + ۱۰ (زمان سرویس دهی)
۳	۵ (انتظار برای انجام کار ۱) + ۱۰ (انتظار برای انجام کار ۲) + ۴ (زمان سرویس دهی)

کل زمان بودن در سیستم برای زمان بندی فوق:

$$\underbrace{5}_{\text{زمان مربوط به کار ۱}} + \underbrace{(5 + 10)}_{\text{زمان مربوط به کار ۲}} + \underbrace{(5 + 10 + 4)}_{\text{زمان مربوط به کار ۳}} = 39$$

لیست همه زمان بندی های ممکن

زمان بندی	کل زمان بودن در سیستم
[1, 2, 3]	$5 + (5 + 10) + (5 + 10 + 4) = 39$
[1, 3, 2]	$5 + (5 + 4) + (5 + 4 + 10) = 33$
[2, 1, 3]	$10 + (10 + 5) + (10 + 5 + 4) = 44$
[2, 3, 1]	$10 + (10 + 4) + (10 + 4 + 5) = 43$
[3, 1, 2]	$4 + (4 + 5) + (4 + 5 + 10) = 32$
[3, 2, 1]	$4 + (4 + 10) + (4 + 10 + 5) = 37$

نکات:

۱. زمان بندی [3, 1, 2] بهینه می باشد.

۲. زمان الگوریتمی که بخواهد همه زمان بندی ها را در نظر بگیرد به صورت فاکتوریل می باشد.

۳. یک زمان بندی بهینه هنگامی بدست می آید که در آن اول کارهایی که زمان سرویس کوچکتری دارند زمان بندی شوند.

الگوریتم

```
sort the jobs by service time in nondecreasing order;

while (the instance is not solved){

  schedule the next job;              // selection procedure and
                                      // feasibility check

  if (there are no more jobs)         // solution check
     the instance is solved;
}
```

اثبات

- پیچیدگی زمانی: $W(n) \in \Theta(n\lg n)$

◄ قضیه 3.4 تنها زمان بندی که کل زمان بودن در سیستم را کمینه می کند، زمان بندی ای است که در آن کارها بر حسب افزایش زمان سرویس مرتب می شوند.

تعمیم الگوریتم برای چند سرویس دهنده

- سرویس دهنده ۱ به کارهای:

$1, (1+m), (1+2m), (1+3m), \ldots$

- سرویس دهنده ۲ به کارهای:

$2, (2+m), (2+2m), (3+3m), \ldots$

...

- سرویس دهنده i به کارهای:

$i, (i+m), (i+2m), (i+3m), \ldots$

...

- سرویس دهنده m به کارهای:

$m, (m+m), (m+2m), (m+3m), \ldots$

- سرویس می دهد.

زمان بندی با مهلت معین

- بیشینه سازی سود کل
- زمان بندی های غیر ممکن – [1, 2]
- مثال

کار	مهلت	سود
۱	۲	۳۰
۲	۱	۳۵
۳	۲	۲۵
۴	۱	۴۰

همه زمان بندی های ممکن

زمان بندی	سود کل
[1, 3]	30 + 25 = 55
[2, 1]	35 + 30 = 65
[2, 3]	35 + 25 = 60
[3, 1]	25 + 30 = 55
[4, 1]	40 + 30 = 70
[4, 3]	40 + 25 = 65

برخی اصطلاحات

- دنباله امکان پذیر: دنباله ای از کارها که در آن همه کارها بتوانند به ترتیب و در مهلت مقرر خود آغاز شوند.
  - مثال: دنباله [1, 4] امکان پذیر ولی [4, 1] امکان پذیر نمی باشد.
- مجموعه امکان پذیر: مجموعه ای از کارها که برای آن حداقل یک دنباله امکان پذیر وجود داشته باشد.
  - مثال: مجموعه {1, 4} امکان پذیر ولی {2, 4} امکان پذیر نمی باشد.
- دنباله بهینه: یک دنباله امکان پذیر با حداکثر سود
- مجموعه بهینه از کارها: مجموعه کارها در دنباله بهینه

الگوریتم

```
sort the jobs in nonincreasing order by profit;

S = ∅;

while (the instance is not solved){

  select next job;                                  // selection procedure

  if (S is feasible with this job added)            // feasibility check
     add this job to S;
  if (there are no more jobs)                       // solution check
     the instance is solved;
}
```

مثال 4.4

کار	مهلت	سود
۱	3	40
۲	1	35
۳	1	30
4	3	25
5	1	20
6	3	15
7	2	10

۱. $S = \varnothing$

۲. $S = \{1\}$

۳. $S = \{1, 2\}$ ([2, 1] امکان پذیر)

۴. $\{1, 2, 3\}$ رد می شود.

۵. $S = \{1, 2, 4\}$ ([2, 1, 4] امکان پذیر)

۶. $\{1, 2, 4, 5\}$ رد می شود.

۷. $\{1, 2, 4, 6\}$ رد می شود.

۸. $\{1, 2, 4, 7\}$ رد می شود.

$$S = \{1, 2, 4\} \Rightarrow [2, 1, 4], [2, 4, 1]$$

بررسی امکان پذیری

▲ لم 4.3 فرض کنید S مجموعه ای از کارها باشد. در این صورت S امکان پذیر خواهد بود اگر و فقط اگر ترتیب حاصل از مرتب شدن کارهای S بر اساس مهلت های غیر نزولی، امکان پذیر باشد.

مثال آیا مجموعه $\{1, 2, 4, 7\}$ امکان پذیر می باشد.

$$[2, \quad 7, \quad 1, \quad 4]$$
$$\uparrow \quad \uparrow \quad \uparrow \quad \uparrow$$
$$1 \quad 2 \quad 3 \quad 3$$

خیر، زیر کار ۴ در مهلتش نمی تواند زمان بندی شود.

الگوریتم رسمی

▶ Algorithm 4.4 **Scheduling with Deadlines**

Problem: Determine the schedule with maximum total profit given that each job has a profit that will be obtained only if the job is scheduled by its deadline.

Inputs: n, the number of jobs, and array of integers *deadline*, indexed from 1 to n, where *deadline*[i] is the deadline for the ith job. The array has been sorted in nonincreasing order according to the profits associated with the jobs.

Outputs: an optimal sequence J for the jobs.

```
void schedule (int n,
               const int deadline[],
               sequence_of_integer& j)
{
  index i;
  sequence_of_integer K;

  J = [1];
  for (i = 2; i <= n; i++){
     K = J with i added according to nondecreasing values of
                                                  deadline[i];
     if (K is feasible)
        J = K;
  }
}
```

مثال 4.6

کار	۱	۲	۳	٤	٥	٦	۷
مهلت	۳	۱	۱	۳	۱	۳	۲

1. J = [1]
2. K = [2, 1] , J = [2, 1] because K is *feasible*
3. K = [2, 3, 1] is not *feasible*
4. K = [2, 1, 4], J = [2, 1, 4] because K is *feasible*
5. K = [2, 5, 1, 4] is not *feasible*
6. K = [2, 1, 6, 4] is not *feasible*
7. K = [2, 7, 1, 4] is not *feasible*

مقدار نهایی J = [2, 1, 4]

پیچیدگی زمانی در بدترین حالت

- عمل اصلی: دستورالعمل مقایسه
- اندازه ورودی: n، تعداد کارها
- پیچیدگی زمانی:
  - زمان لازم برای مرتب سازی کارها: $\Theta(n\lg n)$
  - تعداد مقایسه ها در حلقه *for-i*

$$\sum_{i=2}^{n}[(i-1)+i] = n^2 - 1 \in \Theta(n^2)$$

  - در کل: $W(n) \in \Theta(n^2)$

قضیه ۴-۴

◄ قضیه ۴-۴ الگوریتم ۴-۴ همواره یک مجموعه بهینه از کارها را ایجاد می کند.

اثبات: از طریق استقراء روی تعداد کارها صورت می پذیرد.

کد هافمن

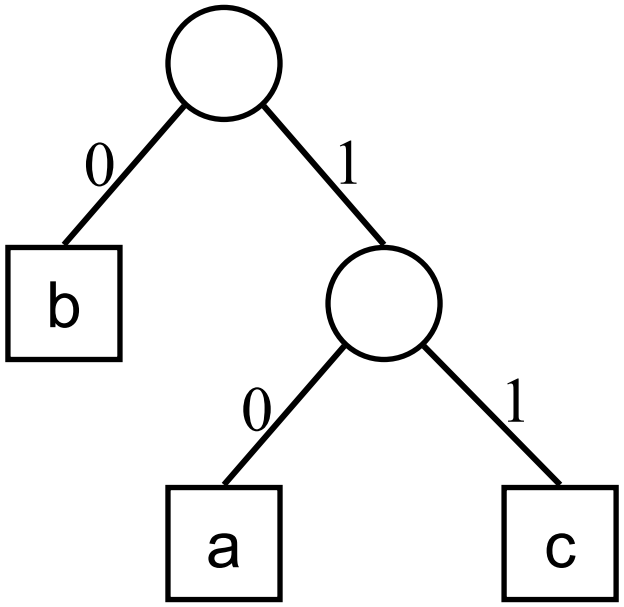
- فشرده سازی داده ها
- کد هافمن
- کد دودویی
- کلمه کد
- کد دودویی با طول ثابت
- کد دودویی با طول متغیر

مثال

- فایل : ababcbbbc
- دو طرح کد گذاری
  - کد دودویی با طول ثابت (کد 4.1)
    - a: 00, b: 01, c: 11
    - فایل کد شده: 000100001110101011 1
  - کد دودویی با طول متغیر (کد 4.2)
    - a: 10, b: 0, c: 11
    - فایل کد شده: 1001001100011

کدهای پیشوندی

- در کد پیشوندی، هیچ کلمه کد مربوط به یک کاراکتر، پیشوند کلمه کد کاراکتر دیگری نمی باشد.
- درخت دودویی مربوط به کد 4.2



مثال

Character	Frequency	C1(fixed-length)	C2	C3(Huffman)
a	16	000	10	00
b	5	001	11110	1110
c	12	010	1110	110
d	17	011	110	01
e	10	100	11111	1111
f	25	101	0	10

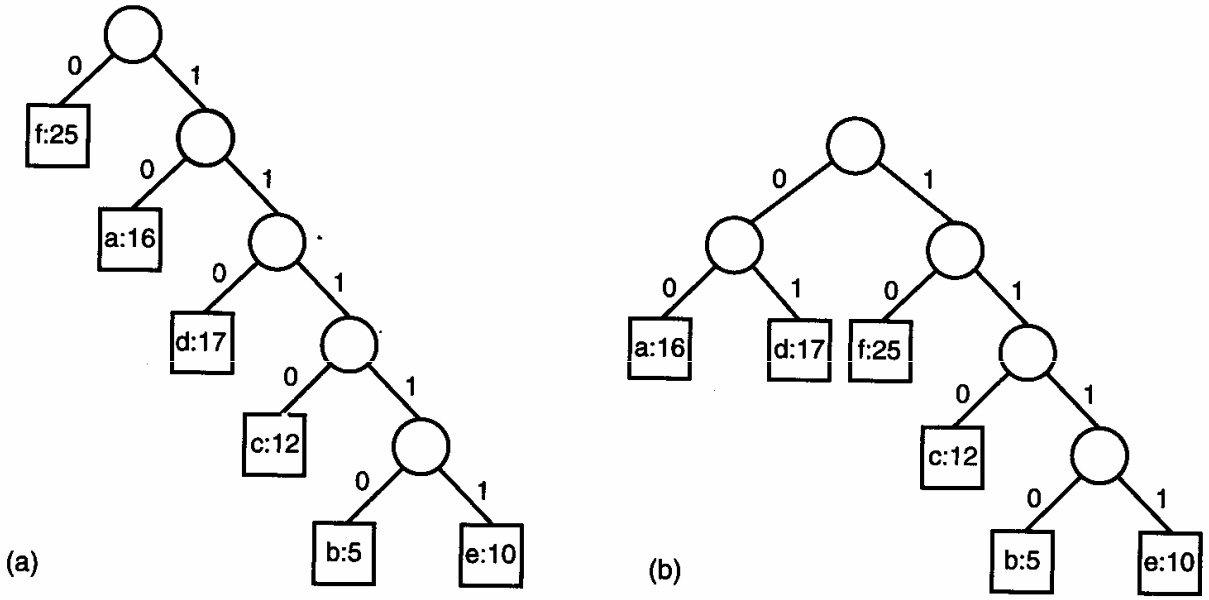


Figure 4.10 ● The binary character code for Code C2 in Example 4.7 appears in (a), while the one for Code C3 (Huffman) appears in (b).

تعداد بیت های لازم برای کد گذاری یک فایل

$$bits(T) = \sum_{i=1}^{n} frequency(v_i) depth(v_i)$$

- مثال : هر یک از موارد زیر را محاسبه کنید:
  - $bits(C1) = (16 + 5 + 12 + 17 + 10 + 25) * 3 = 255$
  - $bits(C2) = 16 * 2 + 5 * 5 + 12 * 4 + 17 * 3 + 10 * 5 + 25 * 1 = 231$
  - $bits(C3) = 16 * 2 + 5 * 4 + 12 * 3 + 17 * 2 + 10 * 4 + 25 * 2 = 212$

الگوریتم هافمن

- ساختار داده ای

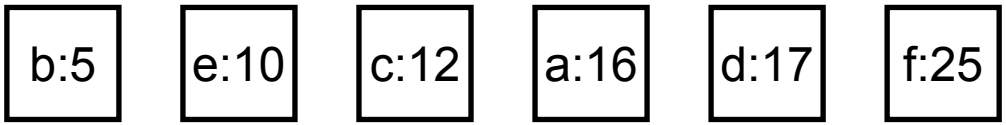
```
struct nodetype
{
  char symbol;          // The value of a character.
  int frequency;        // The number of times the character
                        // is in the file.
  nodetype* left;
  nodetype* right;
};
```

صف اواویت PQ

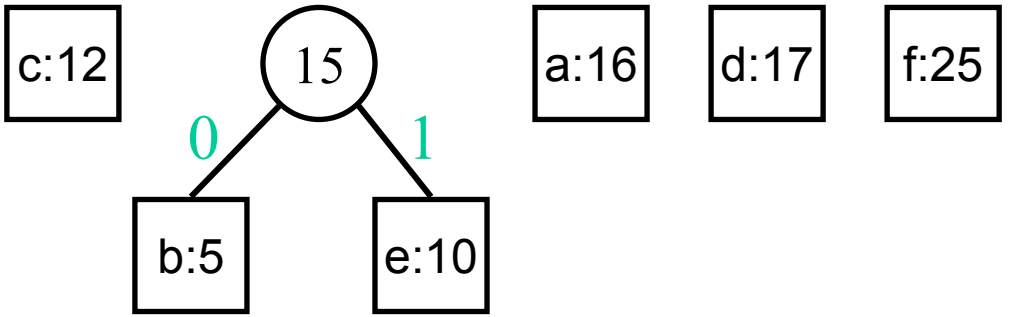
- n اشاره گر به رکورد های $nodetype$ را در صف اولویت PQ به گونه ای تنظیم کن که:
  - p -> $symbol$ = یک کاراکتر متفاوت در فایل
  - p -> $frequency$ = فراوانی آن کاراکتر در فایل
  - p -> $left$ = p -> $right$ = NULL

```
for (i=1; i <= n-1; i++) {   // There is no solution check; rather,
    remove(PQ, p);           // solution is obtained when i = n - 1.
    remove(PQ, q);           // Selection procedure.
    r = new nodetype;        // There is no feasibility check.
    r->left = p;
    r->right = q;
    r->frequency = p->frequency + q->frequency;
    insert(PQ, r);
}
remove(PQ, r);
return r;
```

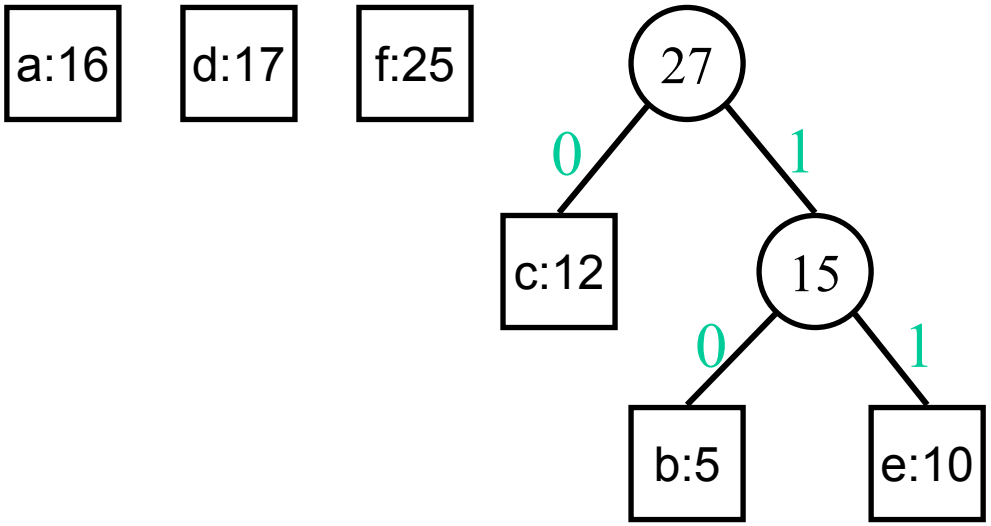
مثال ۴-۸



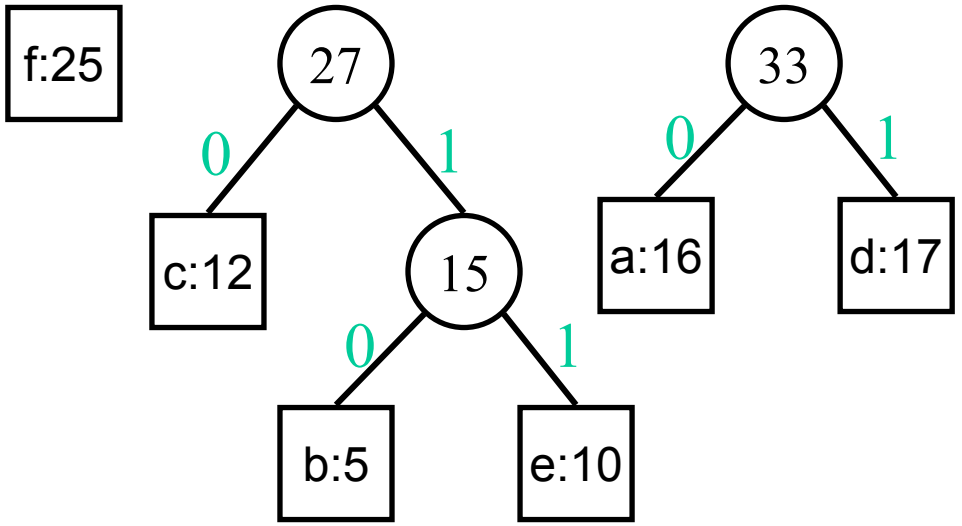
مثال ۸-۴



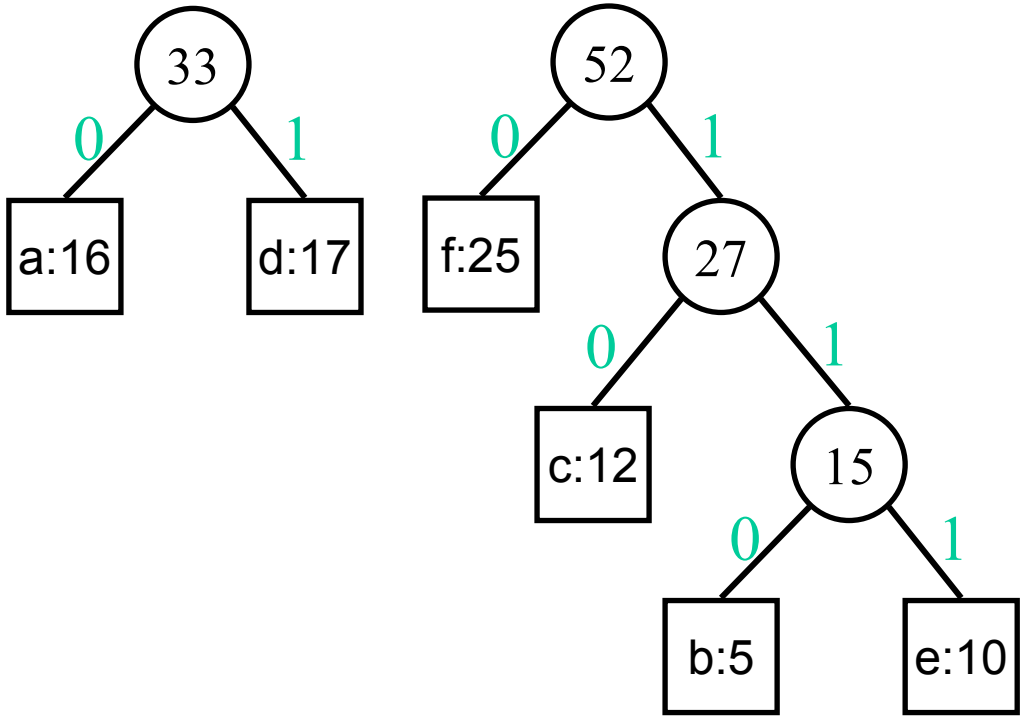
مثال ۸-۴



مثال ۸-۴

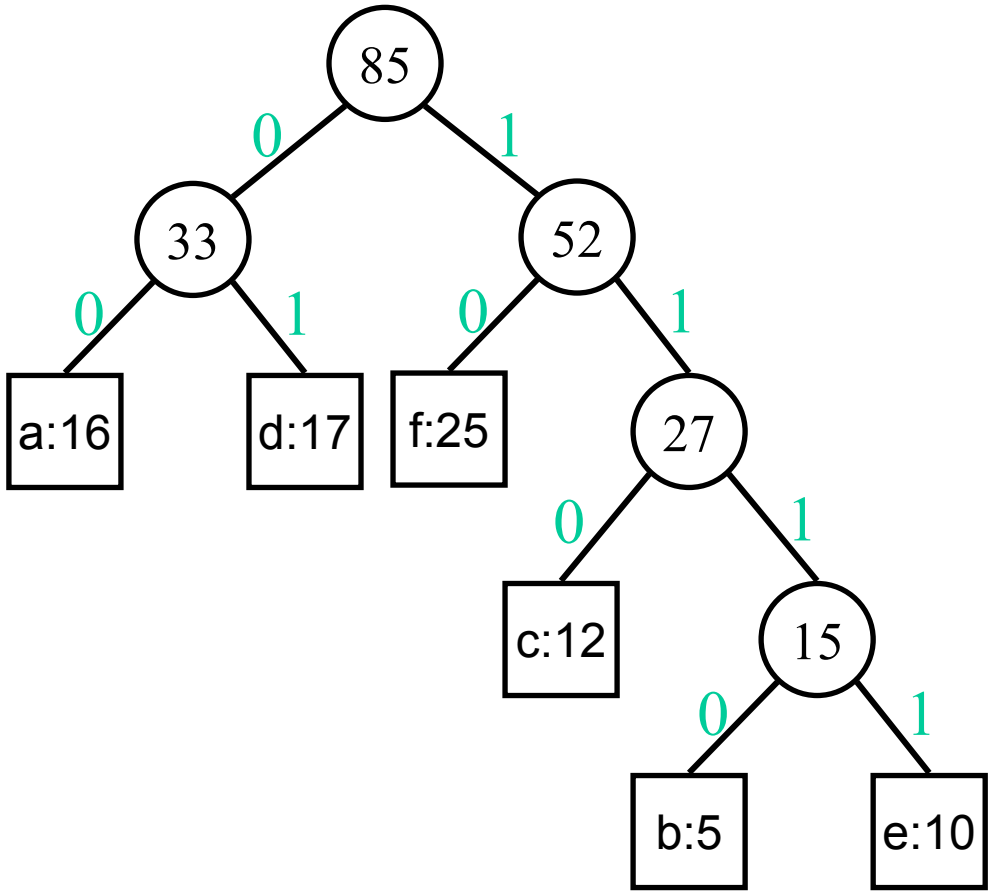


مثال ۸-۴



مثال ۸-۴

کد	کاراکتر
00	*a*
1110	*b*
110	*c*
01	*d*
1111	*e*
10	*f*



اثبات

▲ لم ۴-۴ درخت دودویی متناظر با یک کد پیشوندی بهینه، کامل است. یعنی، هر گره غیر برگ دارای دو فرزند می باشد.

◄ قضیه ۴-۵ الگوریتم هافمن یک کد دودویی بهینه تولید می کند.

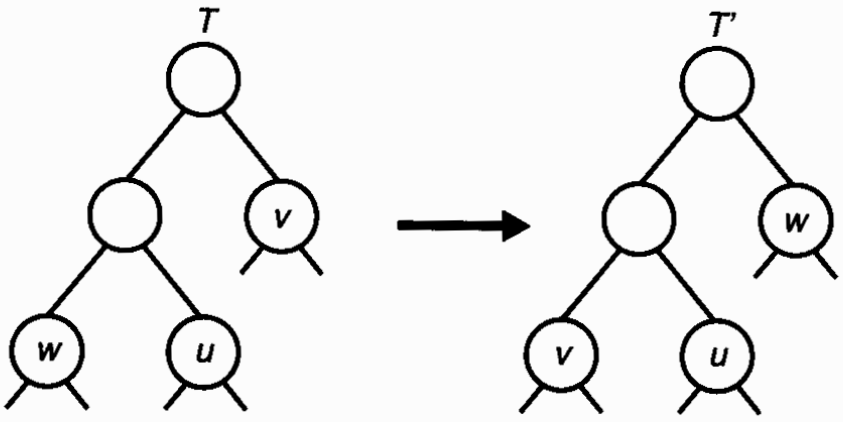


Figure 4.12 ● The branches rooted at *v* and *w* are swapped.

رهیافت حریصانه در مقابل برنامه نویسی پویا: مساله کوله پشتی

- کارآیی:
  - رهیافت حریصانه اغلب ساده تر و کارآتر می باشد.
- اثبات بهینگی:
  - برنامه نویسی پویا: اصل بهینگی
  - رهیافت حریصانه: یک اثبات که معمولا پیچیده تر می باشد.

رهیافت حریصانه برای مساله کوله پشتی 0-1

- مساله کوله پشتی 0-1
  - S = $\{item_1, item_2, \ldots, item_n\}$
  - w_i = weight of $item_i$
  - p_i = profit of $item_i$
  - W = maximum weight the knapsack can hold
- تعیین یک زیر مجموعه مانند A از S به طوری که:

$$\sum_{item_i \in A} p_i \quad \text{is maximized subject to} \quad \sum_{item_i \in A} w_i \le W$$

– الگوریتم $Brute\ force$: $O(2^n)$ – زیرا 2^n زیر مجموعه وجود دارد.

شکست رهیافت حریصانه

Item	item1	$item_2$	$item_3$
weight	25	10	10
profit	10	9	9

$W = 30$

Item	$item_1$	$item_2$	$item_3$
weight	10	10	25
profit	5	5	15

۱) ابتدا عناصر با بیشترین سود را در کوله پشتی قرار بده

۲) ابتدا سبک ترین عناصر را در کوله پشتی قرار بده

۳) ابتدا عناصری را که دارای سود بیشتری در واحد وزن می باشند در کوله پشتی قرار بده

شکست رهیافت حریصانه(ادامه)

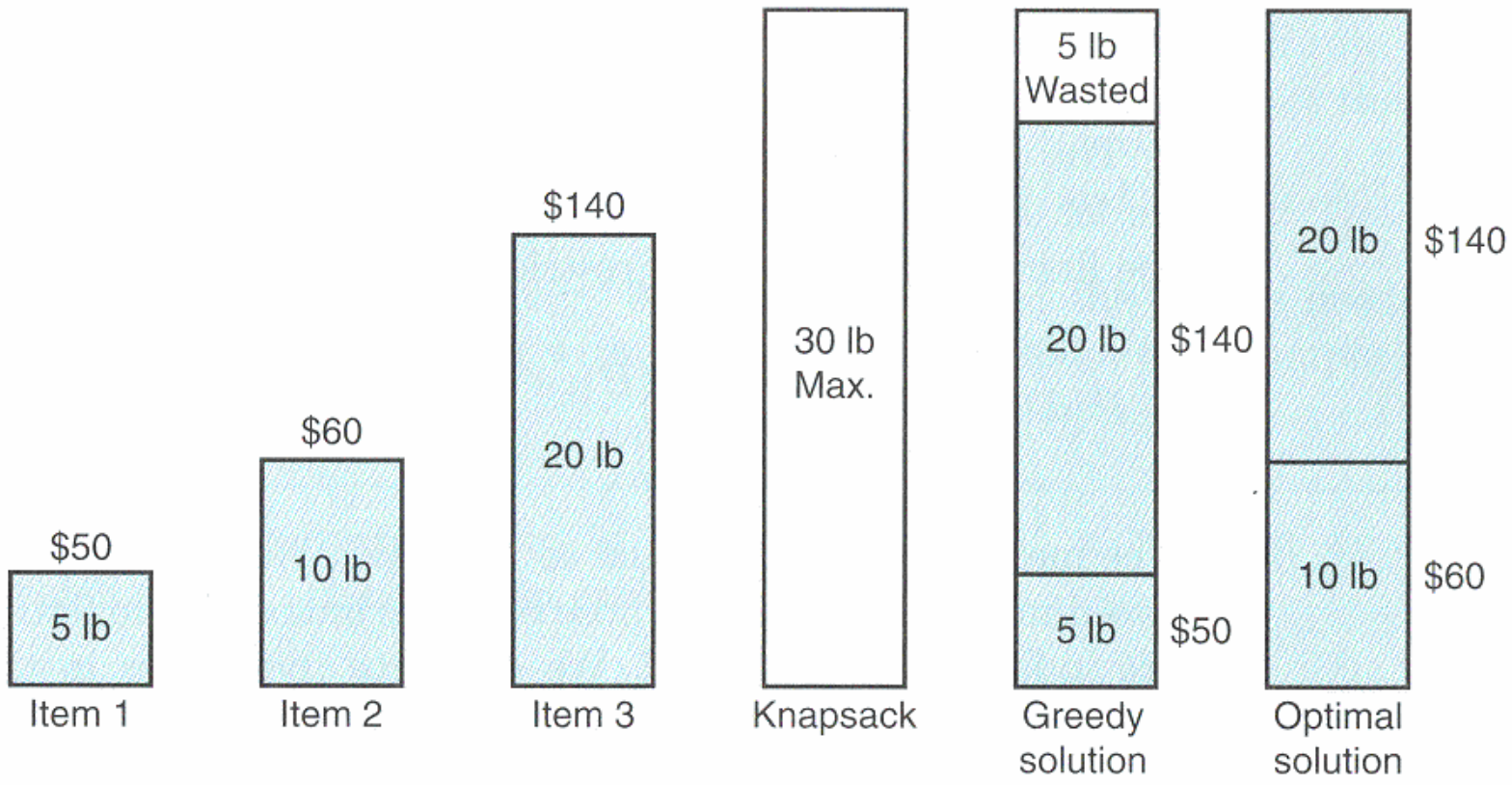


Figure 4.13 ● A greedy solution and an optimal solution to the 0-1 Knapsack problem.

رهیافت حریصانه برای مساله کوله پشتی کسری

- سود کل در مثال قبل

$$\$50 + \$140 + (5/10)(\$60) = \$220$$

رهیافت برنامه نویسی پویا برای مساله کوله پشتی 0-1

- الگوریتم:

$$P[i][w] = \begin{cases} maximum\left(P[i-1][w], p_i + P[i-1][w-w_i]\right) & \text{if } w_i \le w \\ P[i-1][w] & \text{if } w_i > w. \end{cases}$$

- حداکثر سود = $P[n][W]$
- با استفاده از آرایه $P[0..n][0..W]$
- پیچیدگی زمانی:

تعداد درایه های محاسبه شده برابر nW : $\Theta(nW)$

شکل بهتر الگوریتم برنامه نویسی پویا برای مساله کوله پشتی 1-0

- برگشت به عقب به منظور تعیین مداخل مورد نیاز به دلیل

$$P[n][W] = \begin{cases} maximum\,(P[n-1][W], p_n + P[n-1][W-w_n]) & \text{if } w_n \le W \\ P[n-1][W] & \text{if } w_n > W, \end{cases}$$

- مداخل مورد نیاز در سطر n-1، برابر $P[n\text{-}1][W]$ و $P[n\text{-}1][W\text{-}w_n]$ می باشند
- به طور کلی می توانیم از این حقیقت بهره ببریم که $P[i][w]$ از روی $P[i\text{-}1][w]$ و $P[i\text{-}1][w\text{-}w_i]$ محاسبه شده است.
- مگر اینکه $n = 1$ و یا $w \le 0$

یک مثال

- $W = 30$

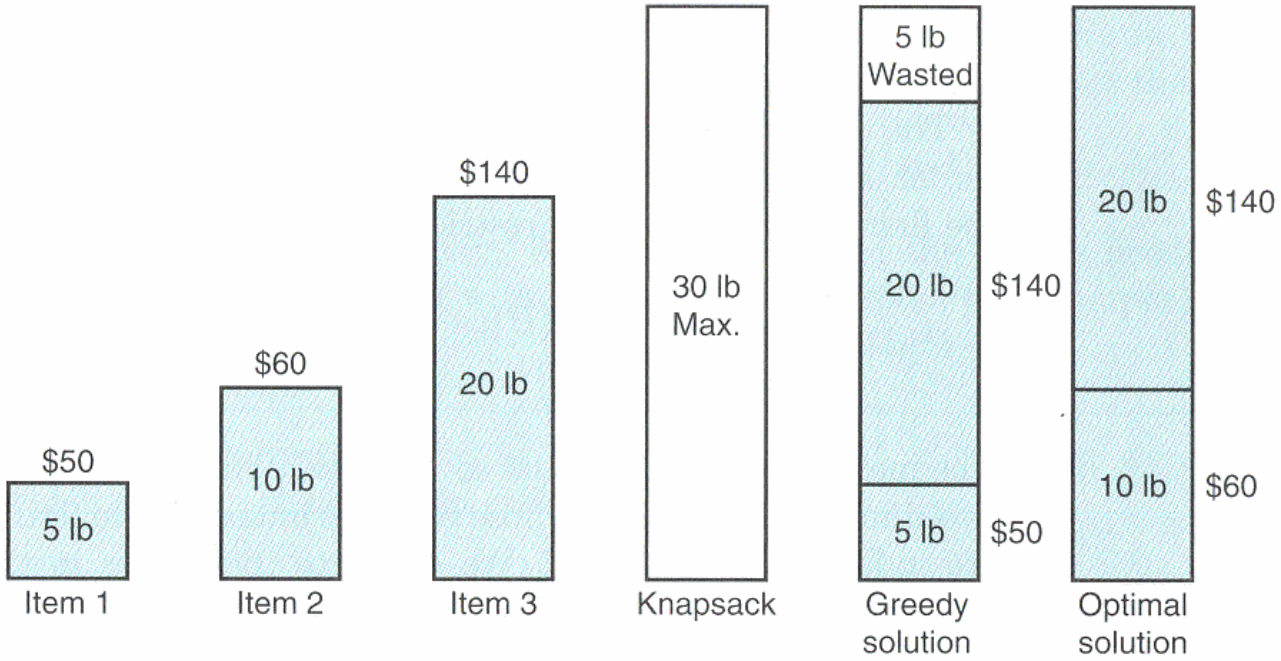
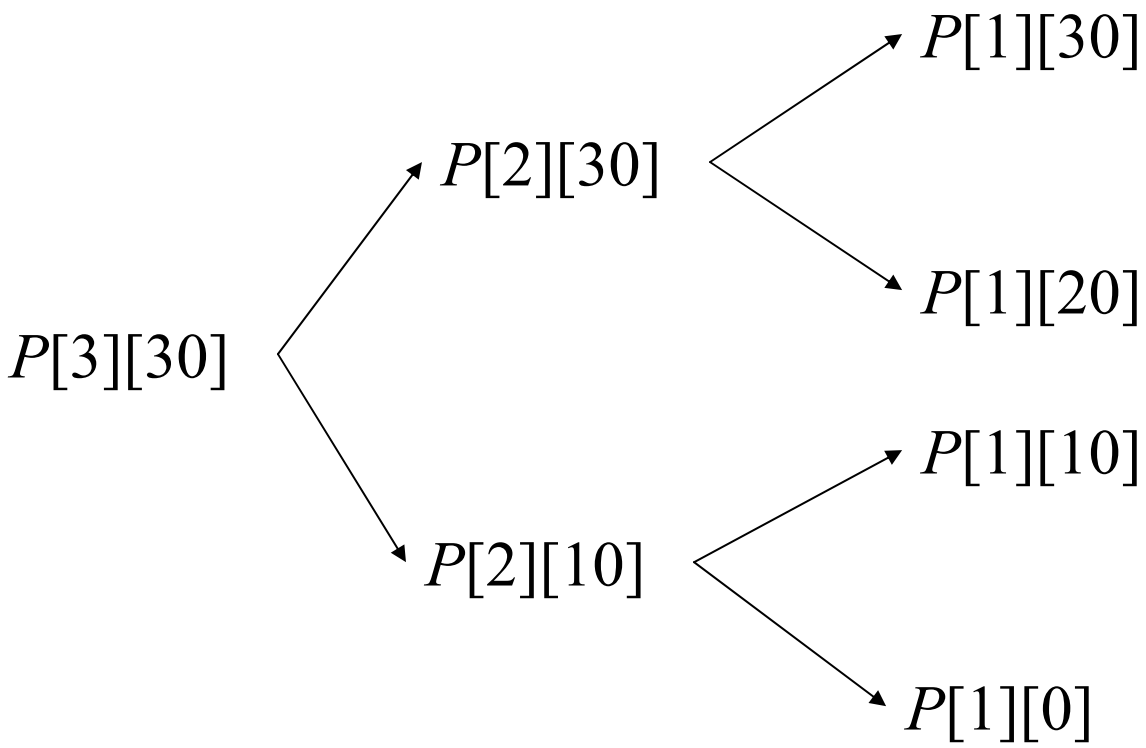


Figure 4.13 ● A greedy solution and an optimal solution to the 0-1 Knapsack problem.

مثال ۴-۹: تعیین عناصر مورد نیاز

عناصر مورد نیاز سطر ۱ | عناصر مورد نیاز سطر ۲ | عناصر مورد نیاز سطر ۳



مثال ۴-۹: محاسبه عناصر

- محاسبه سطر ۱:

$$P[1][w] = \begin{cases} \text{maximum}(P[0][w],\ \$50 + P[0][w-5]) & 5 \le w \\ P[0][w] & 5 > w \end{cases}$$

$$= \begin{cases} \$50 & 5 \le w \\ 0 & 5 > w \end{cases}$$

- بنابراین:

- $P[1][0] = \$0$
- $P[1][10] = \$50$
- $P[1][20] = \$50$
- $P[1][30] = \$50$

مثال ۴-۹: محاسبه عناصر

- محاسبه سطر ۲:

$$P[2][10] = \begin{cases} \text{maximum}(P[1][10],\ \$60 + P[1][0]) & 10 \le 10 \\ P[1][10] & 10 > 10 \end{cases}$$
$$= \$60$$

$$P[2][30] = \begin{cases} \text{maximum}(P[1][30],\ \$60 + P[1][20]) & 10 \le 30 \\ P[1][30] & 10 > 30 \end{cases}$$
$$= \$60 + \$50 = \$110$$

مثال ۴-۹: محاسبه عناصر

- محاسبه سطر ۳:

$$P[3][30] = \begin{cases} \text{maximum}(P[2][30],\ \$140 + P[2][10]) & 20 \le 30 \\ P[2][30] & 20 > 30 \end{cases}$$
$$= \$140 + \$60 = \$200$$

پیچیدگی زمانی

- حداکثر 2^i درایه در سطر $(n-i)$ محاسبه می شود.

- تعداد کل درایه های محاسبه شده حداکثر برابر است با:

$$1 + 2 + 2^2 + \ldots + 2^{n-1} = 2^n - 1 = \Theta(2^n)$$

- نتیجه: تعداد عناصر محاسبه شده در بدترین حالت:

$$O(\text{minimum}(2^n, nW))$$

تمرینات

- بخش ۴-۱
  - ۲، ۶ و ۹
- بخش ۴-۲
  - ۱۱
- بخش ۴-۳
  - ۱۷، ۲۰ و ۲۳
- بخش ۴-۴
  - ۲۴، ۲۵ و ۲۸
- بخش ۴-۵
  - ۳۴، ۳۵، ۳۶
- تمرینات اضافی
  - ۴۴ و ۴۷