به نام خدا

# ساختمان داده ها

جلسه بیستم

**دانشگاه بوعلی سینا**

گروه مهندسی کامپیوتر

نیم سال دوم 98-1397

# گرافها

# Graphs

## مطالب این فصل

- مقدمه و تعاریف
- نمایش گراف
  - ماتریس مجاورتی
  - لیست مجاورتی
  - لیست مجاورتی چندگانه
  - یالهای وزن دار
- اعمال روی گرافها
  - جستجوی عمقی
  - جستجوی ردیفی
  - مولفه های همبند
  - درختهای پوشا
  - مولفه های دو اتصالی
- درخت پوشای کمترین هزینه
  - الگوریم راشال
  - الگوریتم پریم
  - الگوریتم سولین
- کوتاهترین مسیر
  - یک مبدا چند مقصد
  - بین دو زوج راس
  - بستار متعدی
- شبکه های فعالیت

# اعمال ابتدايي گراف

با توجه به گراف بدون جهت G(V,E) و راس v از V(G) می خواهیم رئوسی از G را که از v قابل دسترسی هستند، به دست آوریم( یعنی همه رئوس متصل به v) . برای این کار دو روش وجود دارد :

- **جستجوی عمقی** : روش عمقی تا حدودی شبیه پیمایش preorder یک درخت است.

- **جستجوی ردیفی** : این روش تا حدودی پیمایش ترتیب سطحی را مجسم می کند.

در پیمایش های عمقی و ردیفی ، فرض می کنیم که برای نمایش گراف ها از لیست مجاورتی پیوندی استفاده شده است.

# جستجوي عمقي (Depth First Search)

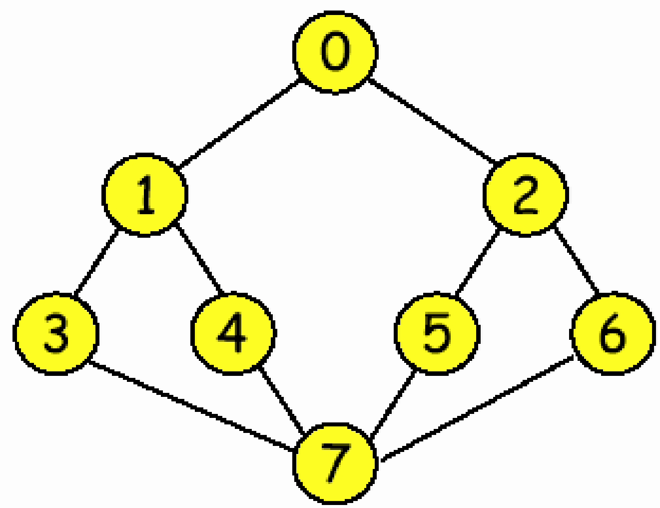
- در آغاز راس v را ملاقات مي كنيم.
- بعد راسي مانند w را كه قبلا ملاقات نشده و مجاور به v است را انتخاب كرده و روش جستجوي عمقي را با w دنبال مي كنيم.
- موقعيت جاري راس v در ليست مجاورتي با قرار دادن آن در يک پشته صورت مي گيرد.
- در نهايت ، جستجو به راسي مانند u خواهد رسيد كه فاقد هر گونه راس غيرملاقات شده در ليست مجاورتي باشد.
- در اين مرحله راسي از پشته انتخاب و حذف شده و فرآيند فوق به همين صورت ادامه پيدا مي كند.
- بر اساس اين روش ، رئوس ملاقات شده، خارج شده و رئوس ملاقات نشده ، داخل پشته قرار مي گيرند. جستجو زماني پايان مي پذيرد كه پشته تهي باشد.

# جستجوی عمقی

```
void Graph::DFS() {
  visited = new bool[n]; // n vertices from 0 to n-1
  for(int i = 0; i < n; ++i)
    visited[i] = false;
  DFS(0); // start search from Vertex 0
  delete [] visited;
}

void Graph::DFS(int v) {
  visited[v] = true;
  // pseudo code
  for( each vertex w adjacent to v)
    if(! visited[w])
      DFS(w);  // recursive call
}
```

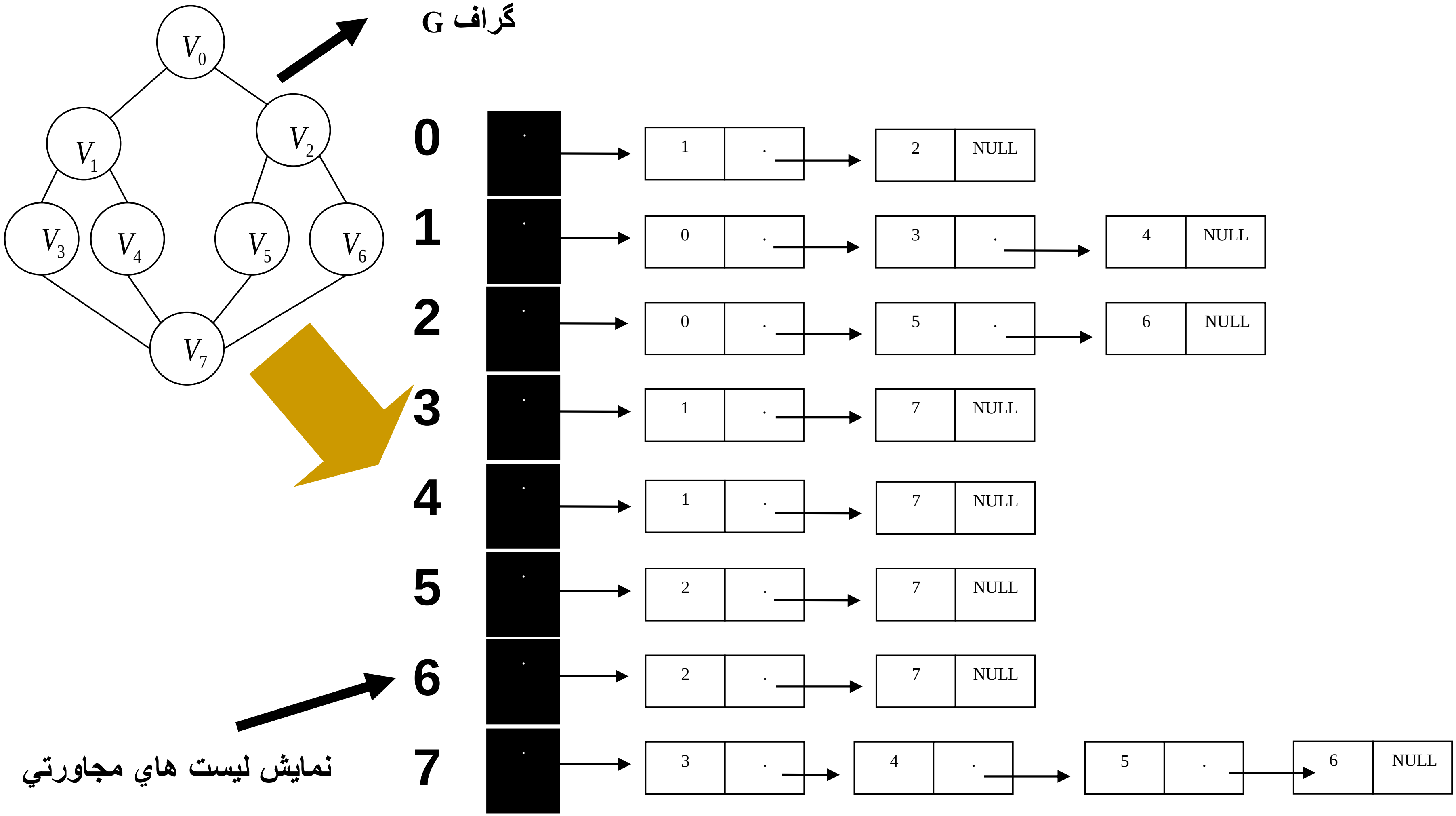
**Stack-Based**




If G is in adjacency list ➔ O(e)
If G is in adjacency matrix ➔ O($n^2$)

**Traversal Order:**
**0, 1, 3, 7, 4, 5, 2, 6**

# جستجوي عمقي(مثال)



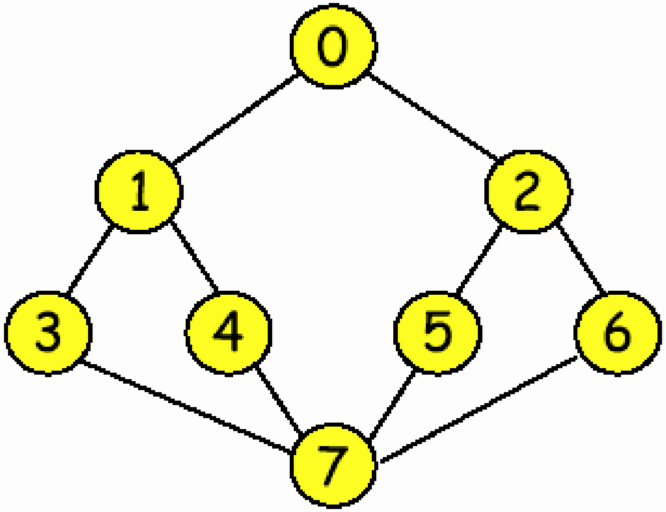
# جستجوي رديفي (Breadth First Search)

- پیمایش را با راس v شروع نموده ، پس از ملاقات راس مزبور ، آنرا علامت گذاری می کنیم و در صف قرار می دهیم.
- اول هر یک از رئوس مجاور به راس v را در لیست مجاورتی ملاقات می کنیم.
- مادامیکه هر راس ملاقات می شود ، آنرا در یک صف قرار می دهیم.
- کار هنگامی که لیست مجاورتی تمام شد ، راسی را از صف حذف و با تست هر یک از رئوس در لیست مجاورتی این فرآیند را ادامه می دهیم.
- رئوس ملاقات نشده ، ملاقات و سپس در صف قرار می گیرند. رئوس ملاقات شده نادیده گرفته می شوند.
- جستجو هنگامی که صف تهی گردد ، خاتمه می یابد.

# جستجوی ردیفی

```
void Graph::BFS(int v) { // starting from Vertex v
  visited = new bool[n]; // n vertices from 0 to n-1
  for(int i = 0; i < n; ++i)
    visited[i] = false;
  Queue<int> q;
  visited[v] = true;
  q.Insert(v);
  while(! q.IsEmpty()) {
    v = *q.Delete(v);
    for( each vertex w adjacent to v)
      if(! visited[w]) {
        visited[w] = true;
        q.Insert(w);
      }
  }
  delete [] visited;
}
```

**Queue-Based**



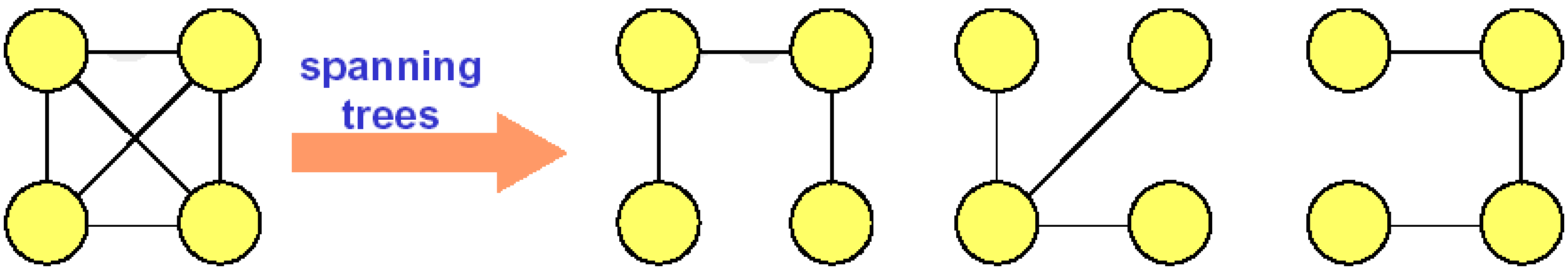

**If G is in adjacency list ➔ O(e)**
**If G is in adjacency matrix ➔ $O(n^2)$**

**Traversal Order:**
**0, 1, 2, 3, 4, 5, 6, 7**

# درختهای پوشا

درخت پوشای یک گراف G درختی است که :

- شامل همه راسهای گراف G باشد
- شامل لبه های گراف G باشد




A spanning tree consists of n vertices and n-1 edges

# درختهای پوشا

مقصود از زیر گراف حداقل ، یعنی زیرگرافی که تعداد لبه هایش کمترین باشد.

- هر گراف متصل با n راس ، بایستی حداقل n-1 لبه داشته باشد و همه گراف ها متصل با n-1 لبه ، درخت هستند.
- درخت پوشا دارای n-1 لبه می باشد.
- ایجاد زیرگراف های حداقل ، کاربردهای متعددی در طراحی شبکه های ارتباطی دارد.

مثال : اگر رئوس گراف G نماینده شهرها و لبه ها معرف جاده های ارتباطی بین شهرها باشد ، آنگاه حداقل تعداد خطوط مورد نیاز برای اتصال n شهر به یکدیگر n-1 خواهد بود.

# درختان پوشاي با حداقل هزينه

هزينه يک درخت پوشاي يک گراف جهت دار داراي وزن ، مجموع هزينه هاي( وزن هاي) لبه ها در درخت پوشا مي باشد.

درخت پوشاي حداقل هزينه ، درخت پوشايي است که داراي کمترين هزينه باشد.

براي به دست آوردن درخت پوشاي حداقل هزينه يک گراف جهت دارمتصل مي توان از سه الگوريتم متفاوت استفاده نمود :

الگوريتم راشال ، الگوريتم پريم ، الگوريتم سولين

هر سه روش از يک طراحي الگوريتمي به نام خط مشي greedy استفاده مي کنند.

# درختان پوشاي با حداقل هزينه

**براي درخت هاي پوشا از ملاک کمترين هزينه استفاده مي شود.**
**روش ما بايد داراي شرايط زير باشد :**

**1)بايد فقط از لبه هاي داخل گراف استفاده کنيم.**

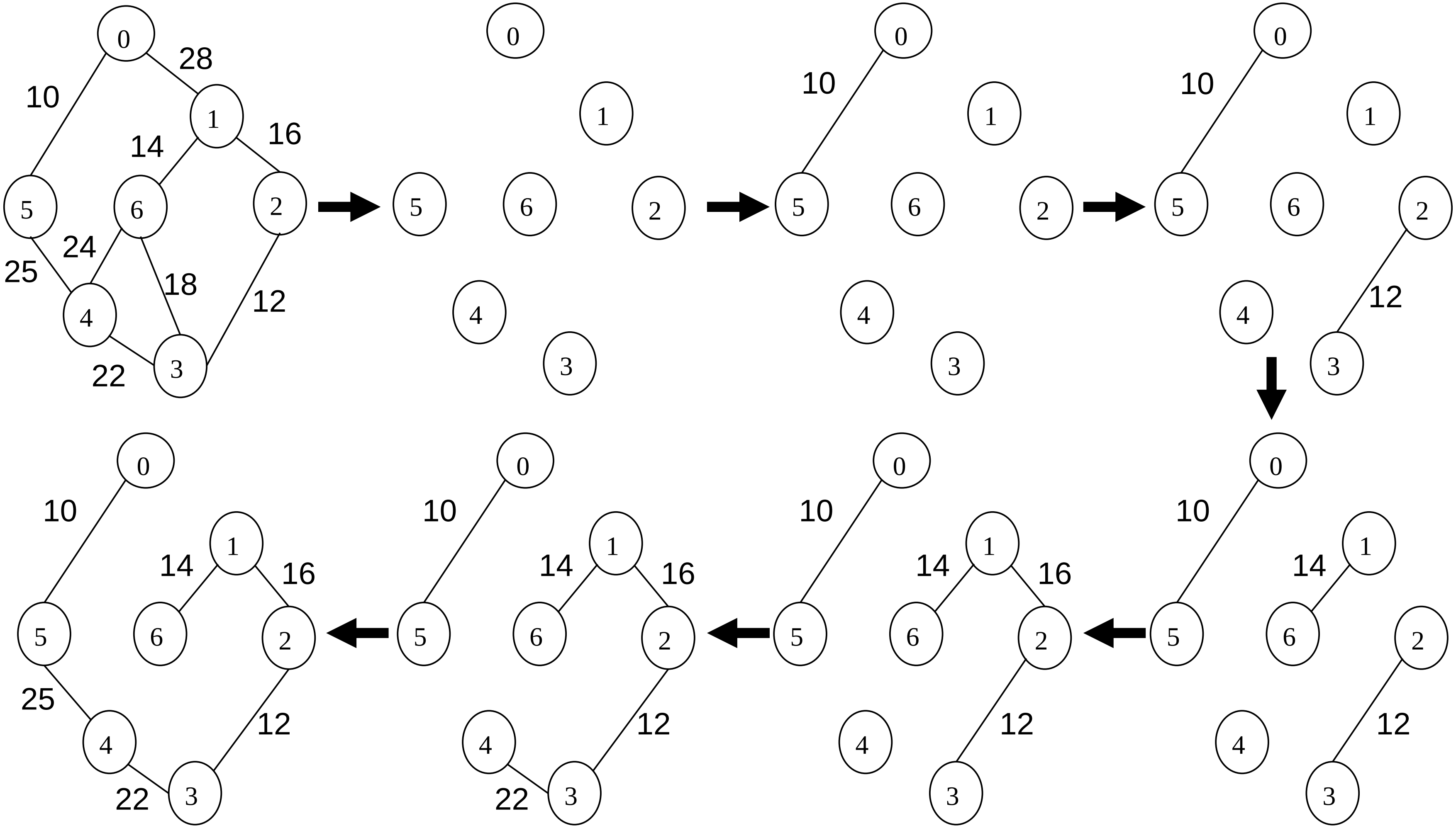
**2) بايد دقيقا از n-1 لبه استفاده کنيم.**

**3) نبايد از لبه هايي که ايجاد يک حلقه مي کنند ، استفاده کنيم.**

# الگوريتم راشال

**در اين روش ، درخت پوشاي با کمترين هزينه T ، لبه به لبه ساخته مي شود. لبه هاي مورد استفاده در T ، به ترتيب صعودي وزن ها مي باشد. يک لبه در T خواهد بود، اگر با لبه هاي قبل که در T بوده اند ، تشکيل يک حلقه ندهد چون G متصل است و داراي $n > 0$ راس است ، دقيقا $n - 1$ لبه براي T انتخاب مي شود.**

# الگوریتم راشال (مثال)



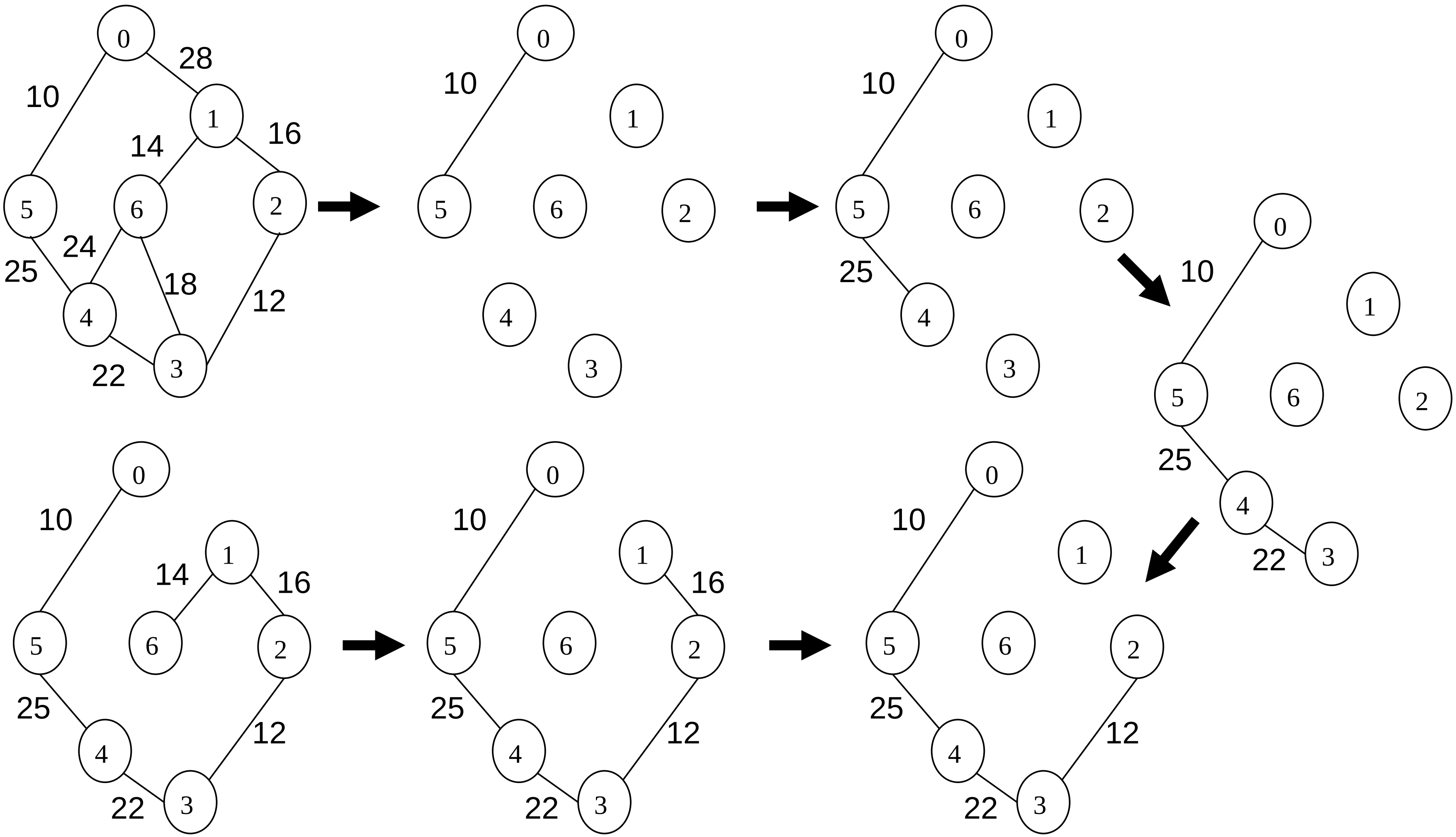
# الگوریتم راشال

```
// E is the set of all edges in G
T = ∅;
while((T contains less than n - 1 edges) && (E is not empty))
{
  choose an edge (v, w) from E of lowest cost;
  delete (v, w) from E;
  if((v, w) does not create a cycle in T)
    add (v, w) to T;
  else
    discard (v, w);
}
if( T contains few than n - 1 edges )
  cerr << "no spanning tree"; // graph is not connected
```

# الگوريتم پريم

**الگوريتم پريم مانند الگوريتم راشال ، در هر زمان يک لبه از درخت پوشاي حداقل هزينه را مي سازد. هر چند در هر مرحله الگوريتم ، مجموعه لبه ها انتخاب شده يک درخت را تشکيل مي دهند . در مقابل ، مجموعه لبه هاي انتخاب شده در الگوريتم راشال در هر مرحله يک جنگل را تشکيل مي دهند. الگوريتم پريم با يک درخت مانند T ، که تنها شامل يک راس است ، شروع مي کند. اين مي تواند هر يک از رئوس در گراف اصلي باشد. سپس يک لبه با کمترين هزينه مانند $(u,v)$ به T اضافه مي شود به نحوي که $T \cup \{(u,v)\}$ نيز خود يک درحت مي باشد. اين عمل را تا زماني که T شامل n – 1 لبه باشد ، ادامه مي دهيم.**

# الگوریتم پریم (مثال)



```
// assume that G has at least one vertex
// TV: vertex set of the spanning tree
// T: edge set of the spanning tree

TV = {0};  // start with Vertex 0 and no edges
for( T = ∅; T contains fewer than n-1 edges; ) {
  let (u, v) be a least-cost edge s.t. u ∈ TV and V ∉ TV;
  if( there is no such edge) break;  // G is not connected
  add v to TV;
  add (u, v) to T;
}
if( T contains few than n - 1 edges )
  cerr << "no spanning tree"; // graph is not connected
```

# الگوريتم سولين

بر خلاف الگوريتم پريم و راشال ، الگوريتم سولين چندين لبه را براي اضافه نمودن در هر مرحله انتخاب مي كند. در ابتدا يک مرحله ، لبه هاي انتخاب شده ، همراه با تمام n راس گراف ، تشكيل يک درخت پوشا را مي دهند. در خلال يک مرحله يک لبه براي هر درخت در جنگل انتخاب مي كنيم. اين لبه داراي حداقل هزينه بوده يعني دقيقا داراي يک راس در درخت مي باشد. از آنجا كه دو درخت در جنگل مي توانند يک لبه يكسان انتخاب كنند ، لذا مي توانيم كپي تكراري از لبه ها را حذف كنيم. در ابتداي مرحله اول ، مجموعه لبه هاي انتخاب شده خالي مي باشد. اين الگوريتم هنگامي به پايان مي رسد كه فقط يک درخت در انتهاي يک مرحله باقي و يا هيچ لبه اي براي انتخاب باقي نمانده باشد.

# الگوریتم سولین (مثال )