HEAPSORT

- Giải thuật sắp xếp (sorting algorithm)
- Heaps
- Thuật giải Heapsort
- Hàng đợi ưu tiên (priority queue)

GIẢI THUẬT SẮP XẾP

- Input: một dãy n số (a_1, a_2, ..., a_n)
- Output: một hoán vị của input (a'_1, a'_2, ..., a'_n) sao cho

  $a'_1 \leq a'_2 \leq .... \leq a'_n$

HEAPS

- Đó là một mảng các đối tượng được biểu diễn bởi một cây nhị phân có thứ tự và cân bằng
- Mỗi nút tương ứng với một phần tử của mảng, gốc ứng với phần tử đầu tiên của mảng

HEAPS

- Cây được lấp đầy trên tất cả các mức, ngoại trừ mức thấp nhất được lấp đầy từ bên trái sang (có thể chưa lấp đầy)
- Một heap biểu diễn một mảng *A* có hai đặc tính:
  - *length*[*A*], là số phần tử của mảng
  - *heap-size*[*A*], là số phần tử của *A* được biểu diễn bởi heap

HEAPS

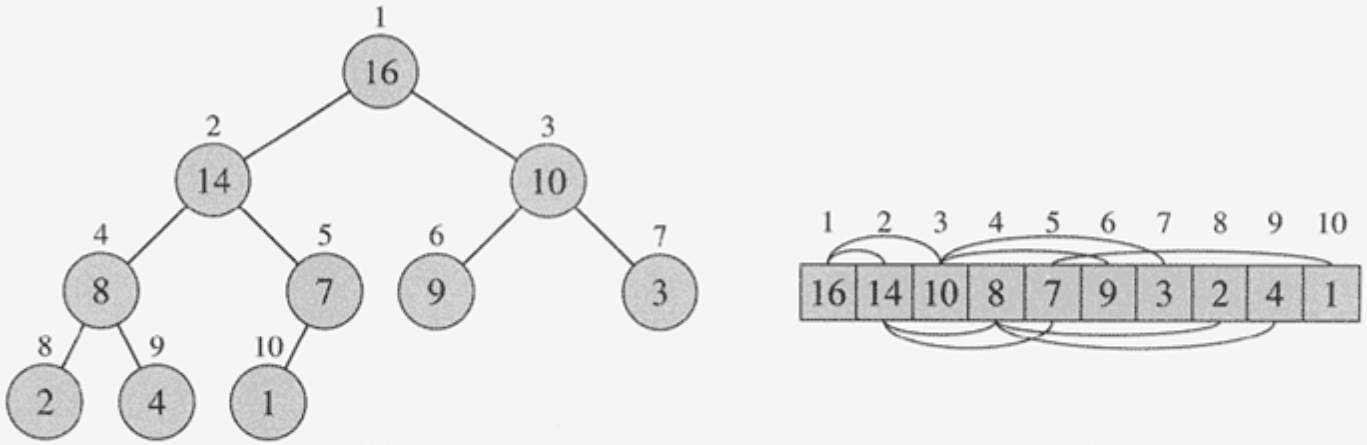


Figure 6.1 A max-heap viewed as **(a)** a binary tree and **(b)** an array. The number within the circle at each node in the tree is the value stored at that node. The number above a node is the corresponding index in the array. Above and below the array are lines showing parent-child relationships; parents are always to the left of their children. The tree has height three; the node at index 4 (with value 8) has height one.

HEAPS

- Chỉ số của cha, con trái và con phải của nút *i* có thể tính:
  - PARENT(*i*)

    return $\lfloor i/2 \rfloor$
  - LEFT(*i*)

    return $2i$
  - RIGHT(*i*)

    return $2i + 1$

HEAPS

- Có hai loại heap nhị phân, max-heap và min-heap
  - Trong max-heap $A[\text{PARENT}(i)] \geq A[i]$ với mọi nút i khác gốc
    $\Rightarrow$ phần tử lớn nhất được lưu trữ tại gốc
  - Trong min-heap $A[\text{PARENT}(i)] \leq A[i]$ với mọi nút i khác gốc
    $\Rightarrow$ phần tử nhỏ nhất được lưu trữ tại gốc

HEAPS

- Các thủ tục trên max-heap dùng cho sắp xếp
  - MAX-HEAPIFY tạo một max-heap có gốc tại nút *i*
  - BUILD-MAX-HEAP xây dựng một max-heap từ một mảng không thứ tự
  - HEAPSORT sắp xếp một mảng

MAX-HEAPIFY

- Đầu vào là một mảng (heap) A và chỉ số i trong mảng
- Các cây nhị phân được định gốc tại LEFT(i) và RIGHT(i) là các max-heap nhưng $A[i]$ có thể nhỏ hơn các con của nó
- MAX-HEAPIFY đẩy giá trị $A[i]$ xuống sao cho cây con định gốc tại $A[i]$ là một max-heap

MAX-HEAPIFY

```
MAX-HEAPIFY(A, i)
 1  l ← LEFT(i)
 2  r ← RIGHT(i)
 3  if l ≤ heap-size[A] and A[l] > A[i]
 4      then largest ← l
 5      else largest ← i
 6  if r ≤ heap-size[A] and A[r] > A[largest]
 7      then largest ← r
 8  if largest ≠ i
 9      then exchange A[i] ↔ A[largest]
10           MAX-HEAPIFY(A, largest)
```

MAX-HEAPIFY

- Thời gian chạy của MAX-HEAPIFY từ dòng 1 đến 8 là O(1)
- Mỗi cây con có kích thước lớn nhất là $2n/3$ nếu heap có n nút vì vậy thời gian chạy của MAX-HEAPIFY là

$$T(n) \leq T(2n/3) + O(1)$$

- Giải hệ thức này ta có $T(n) = O(\lg n) = O(h)$ (h là chiều cao cây)

BUILD-MAX-HEAP

- Các nút có chỉ số $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \ldots, n$ trong $A[1..n]$ là các lá của cây, mỗi nút như vậy là một max-heap
- BUILD-MAX-HEAP áp dụng MAX-HEAPIFY cho các nút con khác lá của cây từ dưới lên gốc bắt đầu từ nút $\lfloor n/2 \rfloor$
- Kết quả là một max-heap tương ứng với $A[1..n]$

BUILD-MAX-HEAP

```
BUILD-MAX-HEAP(A)
1  heap-size[A] ← length[A]
2  for i ← ⌊length[A]/2⌋ downto 1
3      do MAX-HEAPIFY(A, i)
```

BUILD-MAX-HEAP

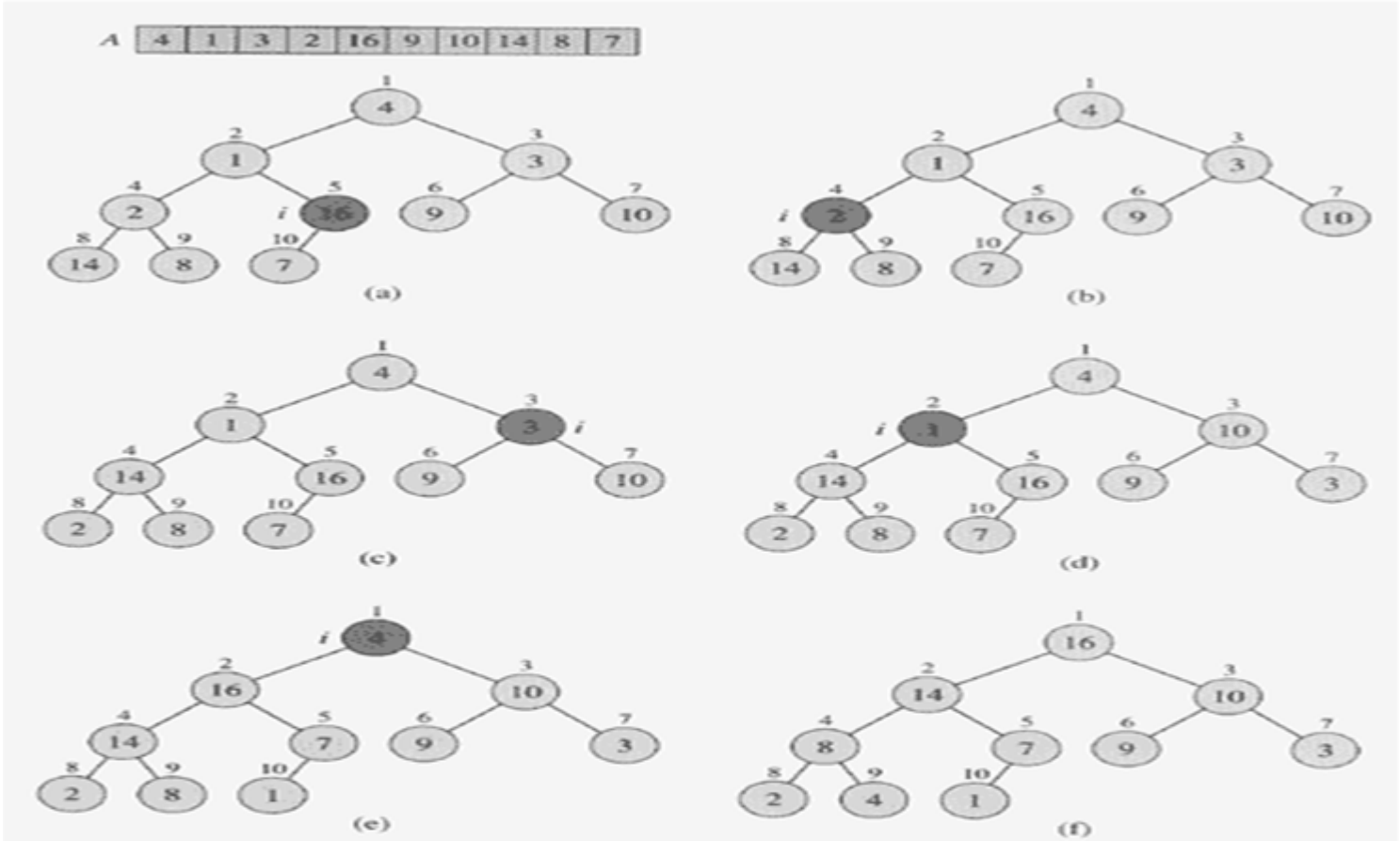


Figure 6.3 The operation of BUILD-MAX-HEAP, showing the data structure before the call to MAX-HEAPIFY in line 3 of BUILD-MAX-HEAP. **(a)** A 10-element input array A and the binary tree it represents. The figure shows that the loop index i refers to node 5 before the call MAX-HEAPIFY(A, i). **(b)** The data structure that results. The loop index i for the next iteration refers to node 4. **(c)–(e)** Subsequent iterations of the **for** loop in BUILD-MAX-HEAP. Observe that whenever MAX-HEAPIFY is called on a node, the two subtrees of that node are both max-heaps. **(f)** The max-heap after BUILD-MAX-HEAP finishes.

BUILD-MAX-HEAP

- Bất biến vòng lặp: Tại điểm bắt đầu của mỗi lần lặp của vòng lặp 2-3, mỗi nút i+1, i+2,..., n là gốc của một max-heap
- Bất biến này đúng trước lần lặp đầu tiên, sau đó duy trì cho mỗi lần lặp tiếp theo

BUILD-MAX-HEAP

- **Khởi đầu**: $i = \lfloor n/2 \rfloor$, mỗi nút $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \ldots, n$ là một lá, chúng là gốc của một max-heap
- **Duy trì**: MAX-HEAPIFY(A, i) đảm bảo nút i và các con của nó là các gốc của các max-heap, bất biến vòng lặp thỏa khi i giảm và trở về đầu vòng lặp
- **Kết thúc**: Khi $i = 0$, mỗi nút 1, 2,..., n là gốc của một max-heap

BUILD-MAX-HEAP

- Thời gian chạy của BUILD-MAX-HEAP là O($n$lg$n$), vì có n lần gọi MAX-HEAPIFY, mỗi lần chi phí lg$n$
- Thực sự thời gian chạy của BUILD-MAX-HEAP là O(n)

HEAPSORT

- Heapsort sử dụng BUILD-MAX-HEAP để xây dựng một max-heap trên mảng input $A[1..n]$
- Hoán đổi giá trị $A[1]$ với $A[n]$
- Loại nút n ra khỏi heap và chuyển $A[1..(n-1)]$ thành một max-heap
- Lặp lại các bước trên cho đến khi heap chỉ còn một phần tử

HEAPSORT

```
HEAPSORT(A)
1  BUILD-MAX-HEAP(A)
2  for i ← length[A] downto 2
3       do exchange A[1] ↔ A[i]
4          heap-size[A] ← heap-size[A] − 1
5          MAX-HEAPIFY(A, 1)
```

HEAPSORT

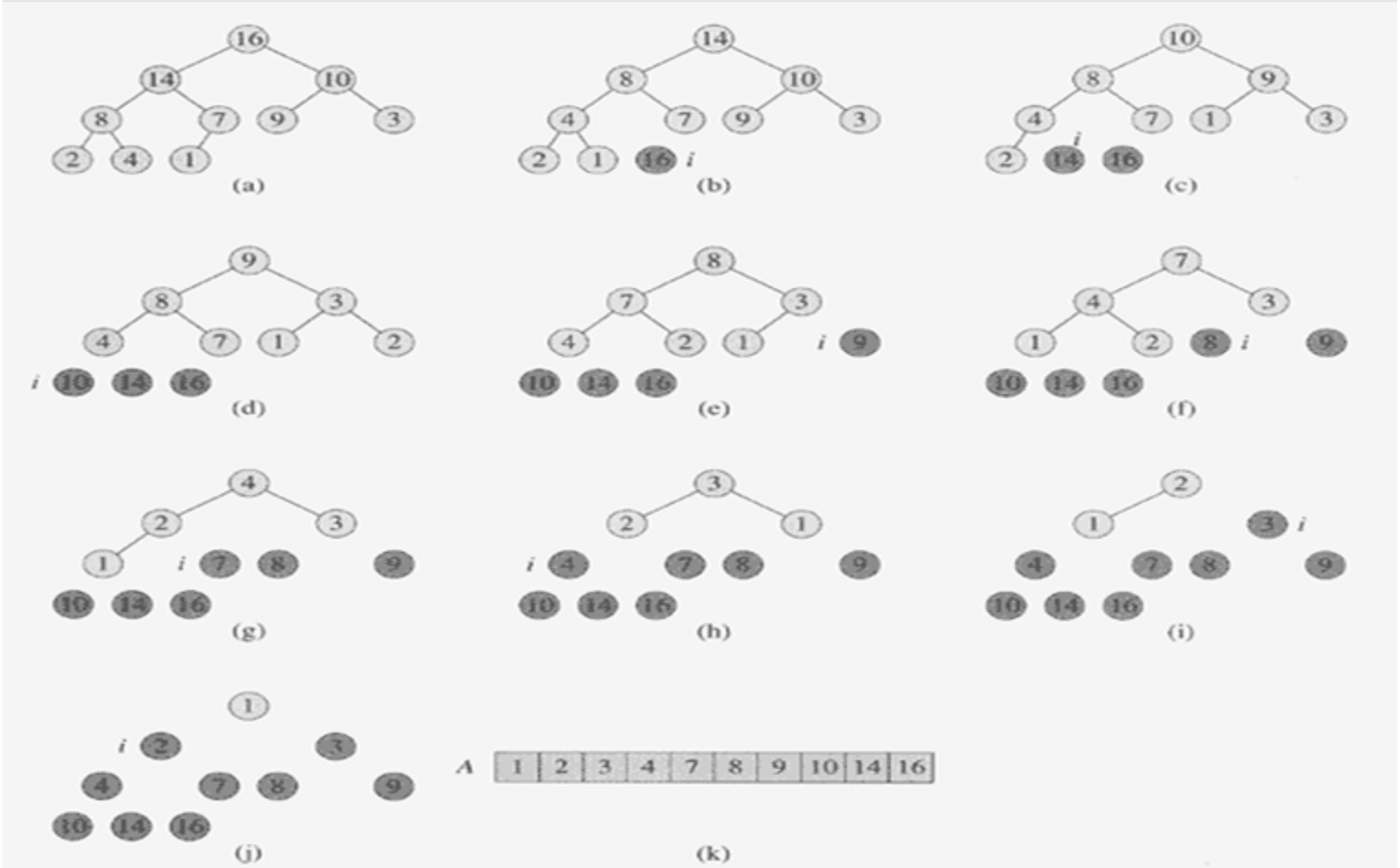


Figure 6.4 The operation of HEAPSORT. **(a)** The max-heap data structure just after it has been built by BUILD-MAX-HEAP. **(b)–(j)** The max-heap just after each call of MAX-HEAPIFY in line 5. The value of i at that time is shown. Only lightly shaded nodes remain in the heap. **(k)** The resulting sorted array A.

HEAPSORT

- Chi phí của BUILD-MAX-HEAP là $O(n)$
- Có n-1 lời gọi MAX-HEAPIFY, mỗi lời gọi chi phí $O(\lg n)$
- Vậy tổng chi phí của HEAPSORT là $O(n \lg n)$

HÀNG ĐỢI ƯU TIÊN

- Hàng đợi ưu tiên (priority queue) gồm một tập đối tượng trong đó đối tượng có khoá ưu tiên được xử lý trước
- Dùng max-heap để biểu diễn hàng đợi ưu tiên theo khóa lớn hơn
- Dùng min-heap để biểu diễn hàng đợi ưu tiên theo khóa nhỏ hơn

HÀNG ĐỢI ƯU TIÊN

- Thao tác trên hàng đợi ưu tiên
  - MAX-HEAP-INSERT(*A*, *x*)
  - HEAP-EXTRACT-MAX(*A*)
  - HEAP-MAXIMUM(*A*)
  - HEAP-INCREASE-KEY(*A*, *x*, *k*)

HEAP-EXTRACT-MAX

- HEAP-EXTRACT-MAX(*A*) loại phần tử được ưu tiên nhất ra khỏi hàng đợi *A*

HEAP-EXTRACT-MAX

```
HEAP-EXTRACT-MAX(A)
1  if heap-size[A] < 1
2      then error "heap underflow"
3  max ← A[1]
4  A[1] ← A[heap-size[A]]
5  heap-size[A] ← heap-size[A] − 1
6  MAX-HEAPIFY(A, 1)
7  return max
```

HEAP-EXTRACT-MAX

- Thời gian chạy của HEAP-EXTRACT-MAX(A) là $O(\lg n)$ trên một heap n phần tử

HEAP-INCREASE-KEY

- HEAP-INCREASE-KEY(*A*, *i*, *key*) tăng khoá tại nút *i* lên thành khoá *key*
- Đi chuyển phần tử có khóa *key* từ nút *i* hướng đến gốc để tìm nơi chính xác cho nút nhận khoá này

HEAP-INCREASE-KEY

```
HEAP-INCREASE-KEY(A, i, key)
1  if key < A[i]
2     then error "new key is smaller than current key"
3  A[i] ← key
4  while i > 1 and A[PARENT(i)] < A[i]
5        do exchange A[i] ↔ A[PARENT(i)]
6           i ← PARENT(i)
```

HEAP-INCREASE-KEY

- Thời gian chạy của HEAP-INCREASE-KEY tối đa là $O(\lg n)$ trên một heap n phần tử

HEAP-INCREASE-KEY

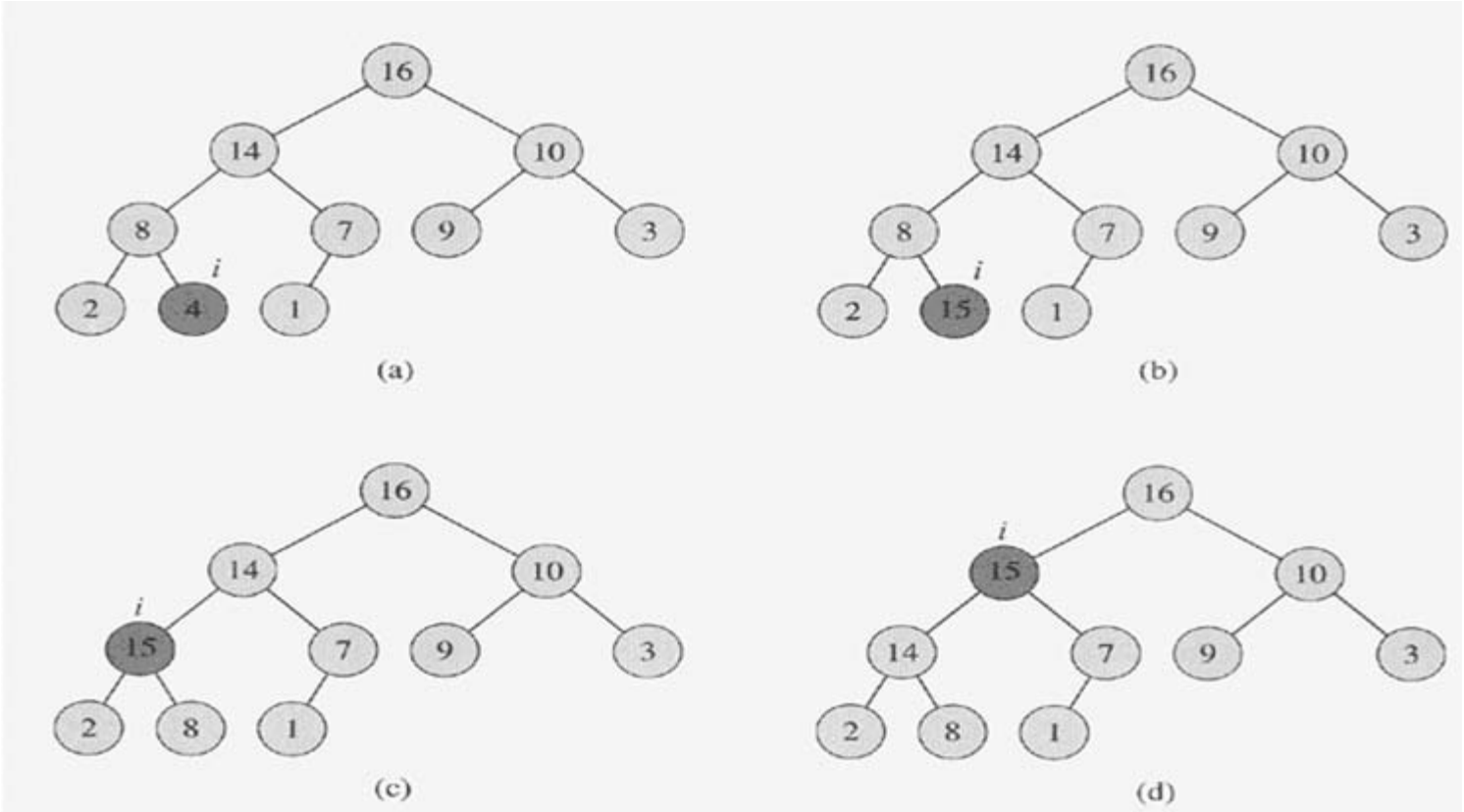


Figure 6.5 The operation of HEAP-INCREASE-KEY. **(a)** The max-heap of Figure 6.4(a) with a node whose index is i heavily shaded. **(b)** This node has its key increased to 15. **(c)** After one iteration of the **while** loop of lines 4–6, the node and its parent have exchanged keys, and the index i moves up to the parent. **(d)** The max-heap after one more iteration of the **while** loop. At this point, $A[\text{PARENT}(i)] \geq A[i]$. The max-heap property now holds and the procedure terminates.

MAX-HEAP-INSERT

- MAX-HEAP-INSERT(*A*, *key*) chèn một phần tử có khoá *key* vào một max-heap
- Đầu tiên, mở rộng heap bằng cách thêm vào một lá mới
- Áp dụng HEAP-INCREASE-KEY để tăng khóa *key* cho nút lá này

MAX-HEAP-INSERT

```
MAX-HEAP-INSERT(A, key)
1  heap-size[A] ← heap-size[A] + 1
2  A[heap-size[A]] ← −∞
3  HEAP-INCREASE-KEY(A, heap-size[A], key)
```

MAX-HEAP-INSERT

- Thời gian chạy của MAX-HEAP- INSERT tối đa là $O(\lg n)$ trên một heap n phần tử

QUICKSORT

- Mô tả Quicksort
- Giải thuật Quicksort
- Hiệu suất Quicksort

MÔ TẢ QUICKSORT

- Do C. A. R Hoare công bố năm 1962
- Là giải thuật tốt, được ứng dụng nhiều trong thực tế

MÔ TẢ QUICKSORT

- Được thiết kế dựa trên kỹ thuật chia để trị (divide-and-conquer):
  - **Divide**: Phân hoạch $A[p..r]$ thành hai mảng con $A[p..q-1]$ và $A[q+1..r]$ có các phần tử tương ứng nhỏ hơn hoặc bằng $A[q]$ và lớn hơn $A[q]$
  - **Conquer**: Sắp xếp hai mảng con $A[p..q-1]$ và $A[q+1..r]$ bằng lời gọi đệ qui

GIẢI THUẬT QUICKSORT

```
QUICKSORT(A, p, r)
1   if p < r
2       then q ← PARTITION(A, p, r)
3            QUICKSORT(A, p, q − 1)
4            QUICKSORT(A, q + 1, r)
```

PARTITION

```
PARTITION(A, p, r)
1   x ← A[r]
2   i ← p − 1
3   for j ← p to r − 1
4         do if A[j] ≤ x
5               then i ← i + 1
6                    exchange A[i] ↔ A[j]
7   exchange A[i + 1] ↔ A[r]
8   return i + 1
```

PARTITION

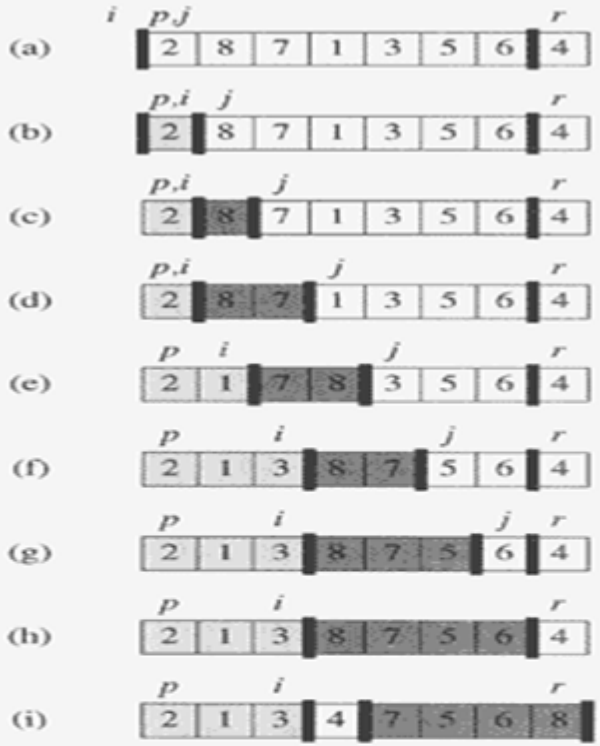


Figure 7.1 The operation of PARTITION on a sample array. Lightly shaded array elements are all in the first partition with values no greater than x. Heavily shaded elements are in the second partition with values greater than x. The unshaded elements have not yet been put in one of the first two partitions, and the final white element is the pivot. **(a)** The initial array and variable settings. None of the elements have been placed in either of the first two partitions. **(b)** The value 2 is "swapped with itself" and put in the partition of smaller values. **(c)–(d)** The values 8 and 7 are added to the partition of larger values. **(e)** The values 1 and 8 are swapped, and the smaller partition grows. **(f)** The values 3 and 8 are swapped, and the smaller partition grows. **(g)–(h)** The larger partition grows to include 5 and 6 and the loop terminates. **(i)** In lines 7–8, the pivot element is swapped so that it lies between the two partitions.

PARTITION

- PARTITION luôn chọn phần tử $x = A[r]$ làm phần tử chốt (pivot) để phân hoạch mảng $A[p..r]$
- Khi partition đang thực hiện mảng bị phân hoạch thành bốn vùng

PARTITION

- Tại điểm bắt đầu của vòng lặp **for** dòng 3-6 mỗi vùng thoả các tính chất sau đây (bất biến của vòng lặp)
  - Nếu $p \le k \le i$, thì $A[k] \le x$ (1)
  - Nếu $i+1 \le k \le j-1$, thì $A[k] > x$ (2)
  - Nếu $k = r$, thì $A[k] = x$ (3)

PARTITION

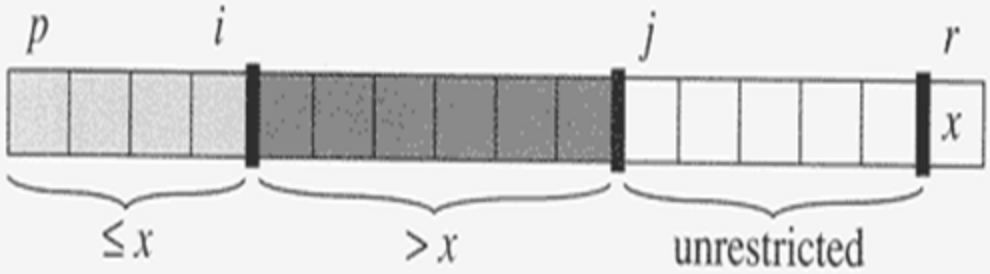


Figure 7.2 The four regions maintained by the procedure PARTITION on a subarray $A[p..r]$. The values in $A[p..i]$ are all less than or equal to x, the values in $A[i+1..j-1]$ are all greater than x, and $A[r] = x$. The values in $A[j..r-1]$ can take on any values.

PARTITION

- Khởi đầu
  - Trước lần lặp đầu tiên, $i = p - 1$ và $j = p$, không có giá trị nào giữa p và i và không có giá trị nào giữa $i + 1$ và $j - 1$
  - Các bất biếnvòng lặp thỏa

PARTITION

- Duy trì
  - Nếu $A[j] > x$, thao tác duy nhất trong vòng lặp là tăng j lên 1, điều kiện 2 thoả cho $A[j-1]$ và tất các mục khác không thay đổi
  - Nếu $A[j] \leq x$, i được tăng lên 1, A[i] và $A[j]$ được tráo đổi sau đó j tăng lên 1, hệ quả $A[i] \leq x$ và $A[j-1] > x$ các bất biến thỏa

PARTITION

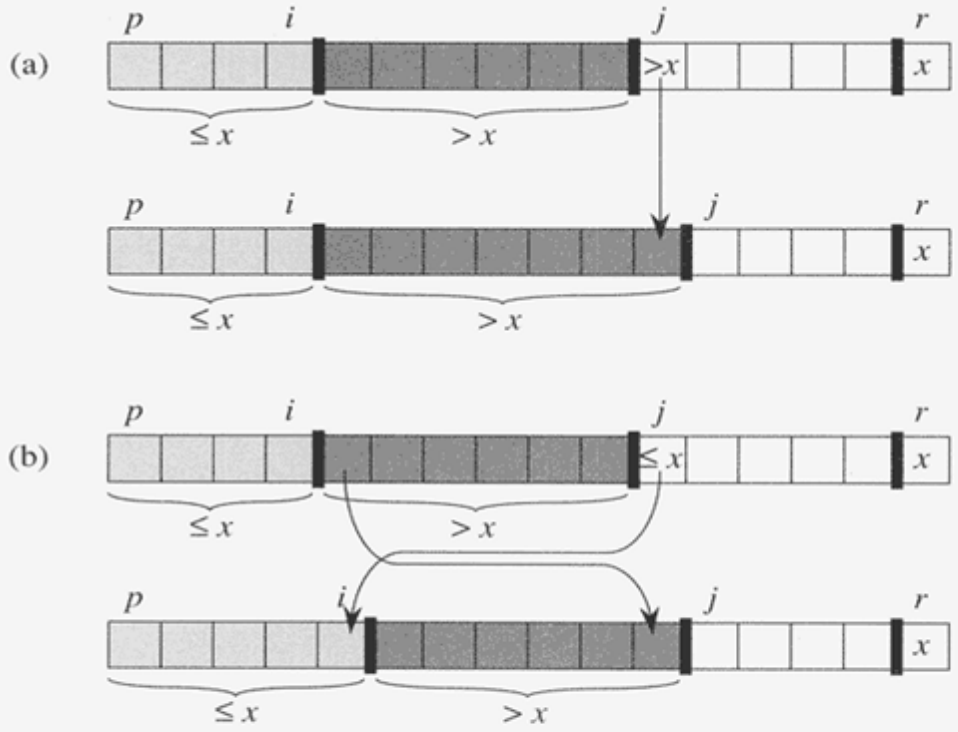


Figure 7.3 The two cases for one iteration of procedure PARTITION. **(a)** If $A[j] > x$, the only action is to increment j, which maintains the loop invariant. **(b)** If $A[j] \leq x$, index i is incremented, $A[i]$ and $A[j]$ are swapped, and then j is incremented. Again, the loop invariant is maintained.

PARTITION

- Kết thúc
  - Khi $j = r$, các bất biến vòng lặp thỏa và mảng đã phân hoạch thành ba phần, nhỏ hơn hoặc bằng x, lớn hơn x và phần cuối chỉ chứa $A[r] = x$
  - Hai lệnh kết thúc partition hoán đổi $A[r]$ với phần tử trái nhất lớn hơn x (vị trí $q = i + 1$)

PARTITION

- Gọi $n = r - p + 1$ là kích thước đầu vào của PARTITION trên mảng A[p..r]
- Thời gian chạy của PARTITION là O(n)

HIỆU SUẤT CỦA QUICKSORT

- Thời gian chạy của Quicksort phụ thuộc vào partition
- Nếu phân hoạch là cân bằng, Quicsort chạy nhanh ít nhất như Heapsort
- Trường hợp xấu nhất, thời gian chạy của Quicksort là $O(n^2)$

HIỆU SUẤT CỦA QUICKSORT

- Trường hợp xấu nhất (worst-case), hai mảng $A[p..q-1]$ và $A[q+1, r]$ có thước $n-1$ và 0
- Chi phí cho PARTITION là $O(n)$
- Vì vậy, thời gian chạy của Quicksort là

$$T(n) = T(n-1) + T(0) + O(n) = O(n^2)$$

HIỆU SUẤT CỦA QUICKSORT

- Trường hợp tốt nhất (best-case), hai mảng $A[p..q-1]$ và $A[q+1, r]$ có thước là $\lfloor n/2 \rfloor$ và $\lceil n/2 \rceil - 1$
- Chi phí cho PARTITION là $O(n)$
- Vì vậy, thời gian chạy của Quicksort là

$$T(n) \leq 2T(n/2) + O(n) = O(n \lg n)$$

HIỆU SUẤT CỦA QUICKSORT

- Phân hoạch cân bằng (balanced partitioning), hai mảng $A[p..q-1]$ và $A[q+1, r]$ có thước xấp xỉ $9n/10$ và $n/10$
- Chi phí cho PARTITION là $O(n)$
- Thơi gian chạy của Quicksort là

$$T(n) \leq T(9n/10) + T(n/10) + O(n) = O(n \lg n)$$

HIỆU SUẤT CỦA QUICKSORT

Cây đệ qui phân hoạch cân bằng

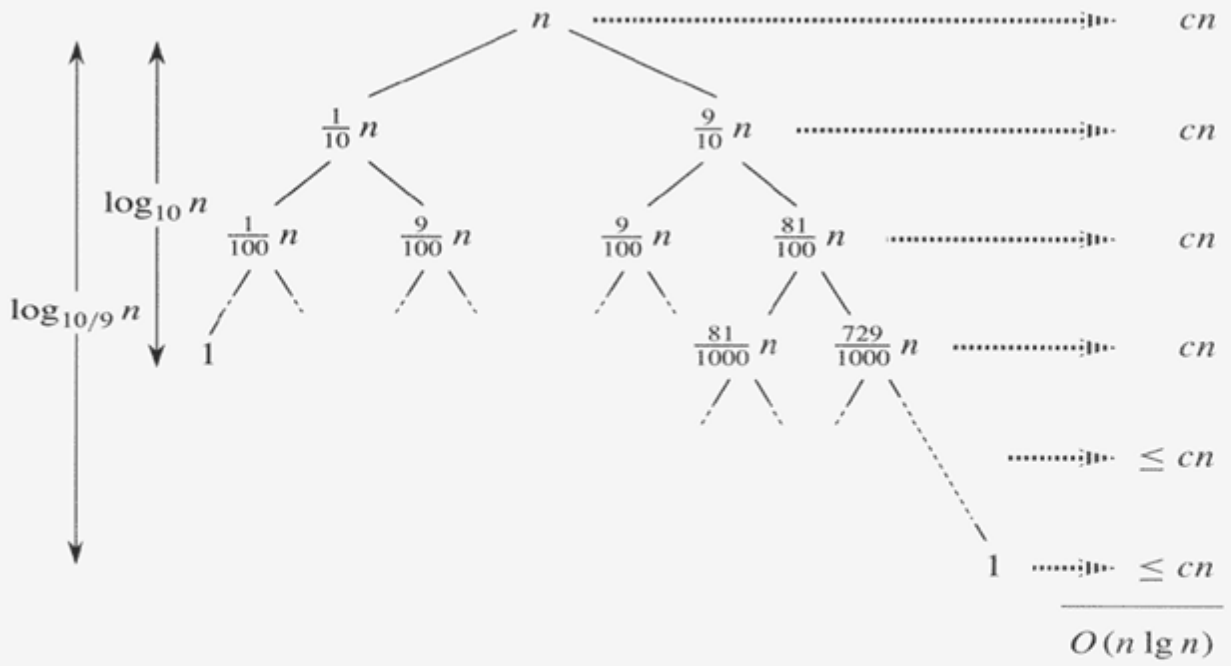


Figure 7.4 A recursion tree for QUICKSORT in which PARTITION always produces a 9-to-1 split, yielding a running time of $O(n \lg n)$. Nodes show subproblem sizes, with per-level costs on the right. The per-level costs include the constant c implicit in the $\Theta(n)$ term.

HIỆU SUẤT CỦA QUICKSORT

- Trường hợp trung bình (average case), Quicksort chạy nhanh gần với trường hợp tốt nhất

$$T(n) = O(n \lg n)$$

HIỆU SUẤT CỦA QUICKSORT

Hai mức của cây đệ qui cho trường hợp trung bình

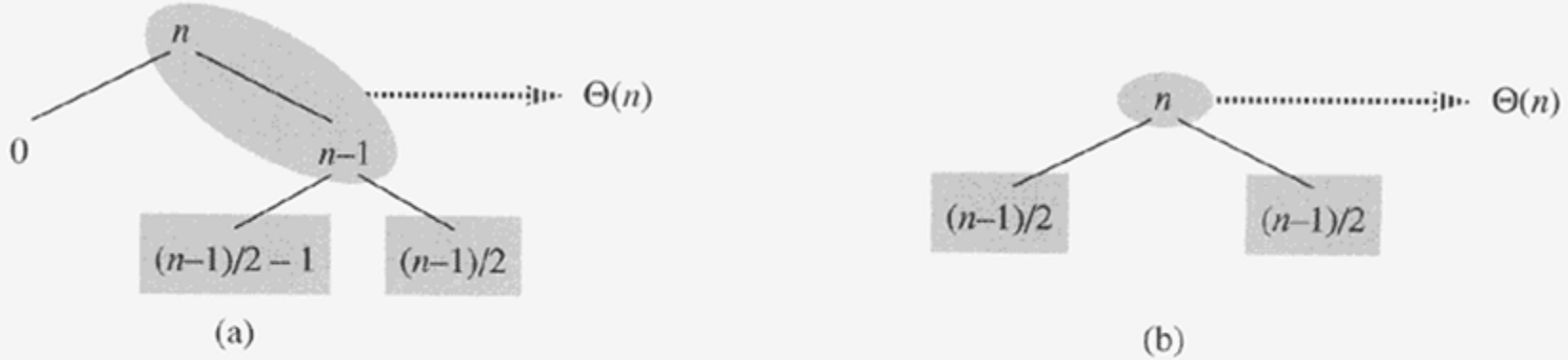


Figure 7.5 **(a)** Two levels of a recursion tree for quicksort. The partitioning at the root costs n and produces a "bad" split: two subarrays of sizes 0 and $n - 1$. The partitioning of the subarray of size $n - 1$ costs $n - 1$ and produces a "good" split: subarrays of size $(n - 1)/2 - 1$ and $(n - 1)/2$. **(b)** A single level of a recursion tree that is very well balanced. In both parts, the partitioning cost for the subproblems shown with elliptical shading is $\Theta(n)$. Yet the subproblems remaining to be solved in (a), shown with square shading, are no larger than the corresponding subproblems remaining to be solved in (b).

SẮP XẾP THỜI GIAN TUYẾN TÍNH

- Khái niệm
- Sắp xếp bằng đếm
- Sắp xếp theo lô

KHÁI NIỆM

- Giải thuật sắp xếp thời gian tuyến tính là giải thuật có thời gian chạy O(n)
- Các giải thuật tốt như Heapsort, Quicksort có thời gian chạy O($n$lg$n$)

KHÁI NIỆM

- Các giải thuật Heapsort, Quicksort dùng phương pháp so sánh, hoán đổi để sắp xếp
- Các giải thuật tuyến tính dựa trên thông tin của các phần tử để sắp xếp nên giảm được bậc của độ phức tạp

SẮP XẾP BẰNG ĐẾM

- Cho *k* là một số nguyên, sắp xếp bằng đếm (counting sort) giả sử mỗi một phần tử trong dãy input là một số nguyên trong miền từ 0 đến *k*

SẮP XẾP BẰNG ĐẾM

- Ý tưởng là đếm số phần tử nhỏ hơn phần tử *x* trong mảng nhập để đặt *x* trực tiếp vào vị trí của nó trong mảng xuất
- Chẳng hạn, nếu có 17 phần tử nhỏ hơn hoặc bằng *x* thì *x* được đặt vào ví trí 17

SẮP XẾP BẰNG ĐẾM

```
COUNTING-SORT(A, B, k)   // B là mảng xuất kết quả
 1  for i ← 0 to k
 2       do C[i] ← 0          // C là mảng chứa quan hệ các phần tử của A
 3  for j ← 1 to length[A]
 4       do C[A[j]] ← C[A[j]] + 1
 5  ▷ C[i] now contains the number of elements equal to i.
 6  for i ← 1 to k
 7       do C[i] ← C[i] + C[i − 1]
 8  ▷ C[i] now contains the number of elements less than or equal to i.
 9  for j ← length[A] downto 1
10       do B[C[A[j]]] ← A[j]
11          C[A[j]] ← C[A[j]] − 1
```

SẮP XẾP BẰNG ĐẾM

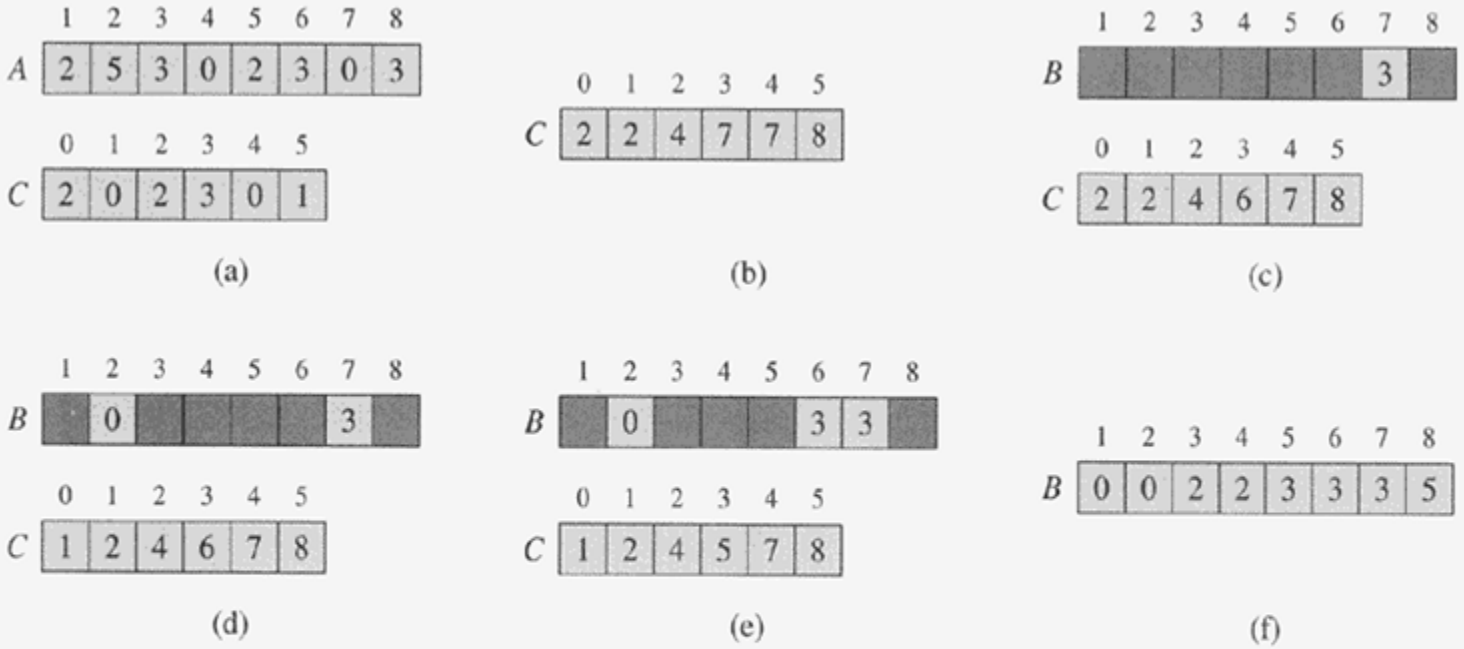


Figure 8.2 The operation of COUNTING-SORT on an input array $A[1..8]$, where each element of A is a nonnegative integer no larger than $k = 5$. **(a)** The array A and the auxiliary array C after line 4. **(b)** The array C after line 7. **(c)–(e)** The output array B and the auxiliary array C after one, two, and three iterations of the loop in lines 9–11, respectively. Only the lightly shaded elements of array B have been filled in. **(f)** The final sorted output array B.

SẮP XẾP BẰNG ĐẾM

- Dòng 1-2 khởi tạo các C[i] = 0
- Dòng 3-4 xác định số phần tử có giá trị là $i = A[j]$ trong A
- Dòng 6-7 xác định số phần tử trong A nhỏ hơn hoặc bằng i, đó là tổng của $C[i]$ và $C[i-1]$

SẮP XẾP BẰNG ĐẾM

- Dòng 9-10 đặt $A[j]$ vào trong vị trí được sắp chính xác của nó trong mảng B căn cứ vào số phần tử nhỏ hơn hoặc bằng $A[j]$ trong $C[A[j]]$
- Giảm $C[A[j]]$ đi 1 trong dòng 10 để các phần tử còn lại bằng $A[j]$ sẽ được đặt chính xác vào mảng B lần lặp sau

SẮP XẾP BẰNG ĐẾM

- Chi phí cho lệnh 1-2 là O(k)
- Chi phí cho lệnh 3-4 là O(n)
- Chi phí cho 6-7 là O(k)
- Chi phí cho 9-11 là O(n)
  - Vì vậy tổng chi phí thời gian là O($k + n$)
  - Nếu $k = O(n)$ thì tổng chi phí là O(n).

SẮP XẾP BẰNG ĐẾM

- COUNTING-SORT chạy thời gian tuyến tính và hiệu quả hơn các giải thuật sắp xếp bằng so sánh
- COUNTING-SORT chỉ sắp xếp các phần tử có khoá trong một miền nhất định (nhỏ hơn hoặc bằng k cho trước)
- COUNTING-SORT phải sử dụng thêm các mảng trung gian

SẮP XẾP THEO LÔ

- Sắp xếp theo lô (Bucket sort) giả sử input là một mảng *n* số không âm nhỏ hơn 1

SẮPP XẾP THEO LÔ

- Ý tưởng của Bucketsort
  - Phân bố mảng input vào *n* khoảng con (lô) của khoảng [0, 1)
  - Sắp xếp các phần tử trong mỗi lô và nối các lô để có mảng được sắp

SẮP XẾP THEO LÔ

```
BUCKET-SORT(A)    // A là mảng mà 0 ≤ A[i] <1
1  n ← length[A]
2  for i ← 1 to n
3       do insert A[i] into list B[⌊nA[i]⌋]      // B chứa các lô
4  for i ← 0 to n − 1
5       do sort list B[i] with insertion sort
6  concatenate the lists B[0], B[1], . . . , B[n − 1] together in order
```

SẮP XẾP THEO LÔ

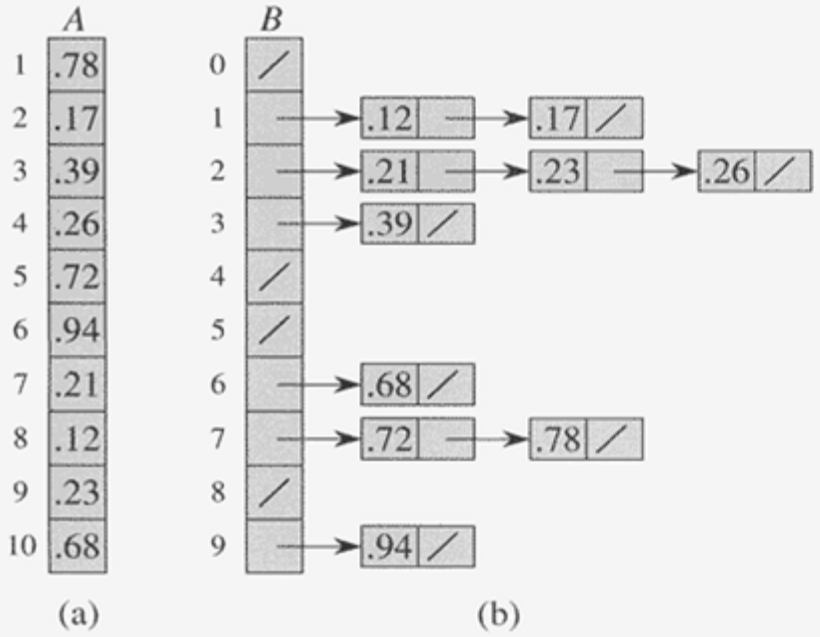


Figure 8.4 The operation of BUCKET-SORT. **(a)** The input array $A[1..10]$. **(b)** The array $B[0..9]$ of sorted lists (buckets) after line 5 of the algorithm. Bucket i holds values in the half-open interval $[i/10, (i + 1)/10)$. The sorted output consists of a concatenation in order of the lists $B[0], B[1], \ldots, B[9]$.

SẮP XẾP THEO LÔ

- Xét hai phần tử $A[i]$ và $A[j]$
  - Nếu $A[i]$ và $A[j]$ cùng rơi vào một lô, chúng có thứ tự nhờ giải thuật chèn trực tiếp
  - Ngược lại, gọi các lô tương ứng của $A[i]$ và $A[j]$ là $B[i']$ và $B[j']$, nếu $i' < j'$ thì lô $B[i']$ được nối trước lô $B[j']$ và khi đó $A[i] \leq A[j]$

SẮP XẾP THEO LÔ

- Thật vậy, giả sử ngược lại $A[i] \geq A[j]$ thì
  - $i' = \lfloor nA[i] \rfloor \geq \lfloor nA[j] \rfloor = j'$
  - Điều này mâu thuẫn với $i' < j'$, nghĩa là $A[i] \leq A[j]$
- Như vậy, giải thuật đảm bảo thứ tự của mảng output

SẮP XẾP THEO LÔ

- Do phân bố ngẩu nhiên *n* phần tử vào *n* khoảng con nên trung bình mỗi lô có 1 phần tử, vì vậy thời gian sắp xếp chèn là O(1)
- Từ đó, chi phí toàn bộ của giải thuật là O(*n*)

SẮP XẾP THEO LÔ

- BUCKET-SORT chạy thời gian tuyến tính và hiệu quả hơn các giải thuật sắp xếp bằng so sánh
- BUCKET-SORT chỉ sắp xếp các phần tử có khoá trong khoảng [0, 1)
- Không phải mọi phân bố sẽ cho mỗi lô chứa 1 phần tử

CÁC THUẬT TOÁN ĐỒ THỊ CƠ BẢN

- Các khái niệm và thuật ngữ
- Biểu diễn đồ thị
- Tìm kiếm theo chiều rộng
- Tìm kiếm theo chiều sâu

KHÁI NIỆM VÀ THUẬT NGỮ

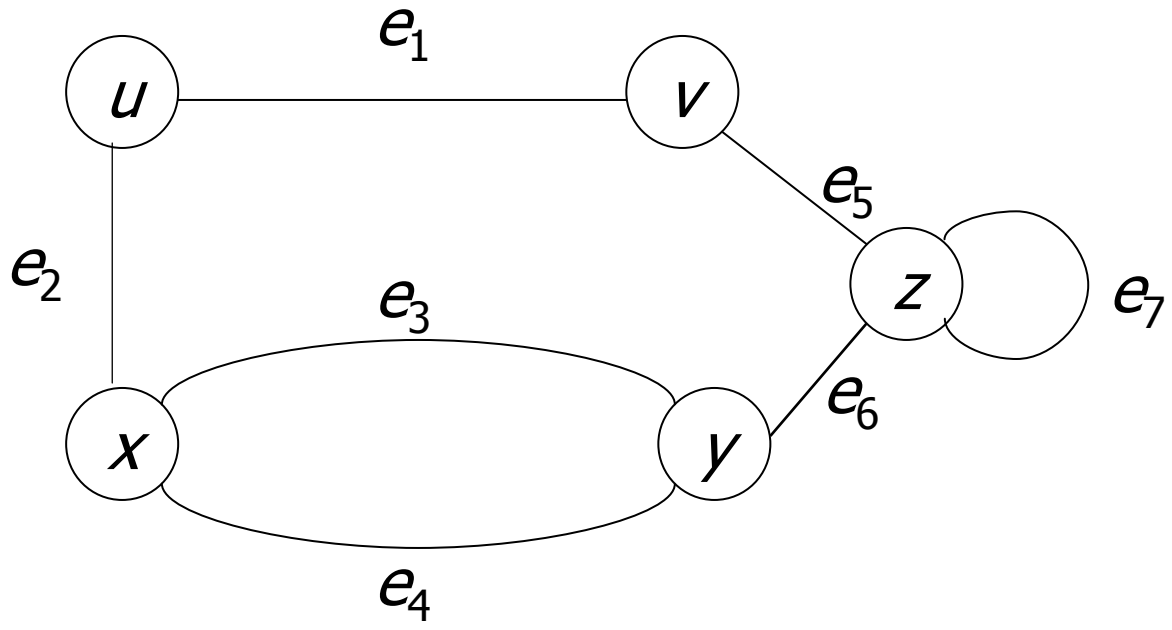
- Đồ thị vô hướng (undirected graph) $G = (V, E)$, gồm một tập V các đỉnh (vertice) và một tập E các cạnh (edge), mỗi cạnh $e = (u, v) \in E$ ứng với một cặp không có thứ tự các đỉnh $u, v \in V$
- Đồ thị có hướng (directed graph) $G = (V, E)$, gồm một tập V các đỉnh và một tập E các cạnh, mỗi cạnh $e = (u, v) \in E$ ứng với một cặp có thứ tự các đỉnh $u, v \in V$

KHÁI NIỆM VÀ THUẬT NGỮ

Ví dụ 1: Đồ thị vô hướng: $V = \{u, v, x, y, z\}$

$E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$

$e_3=(x, y)$
$e_4=(x, y)$
$e_7=(z, z)$

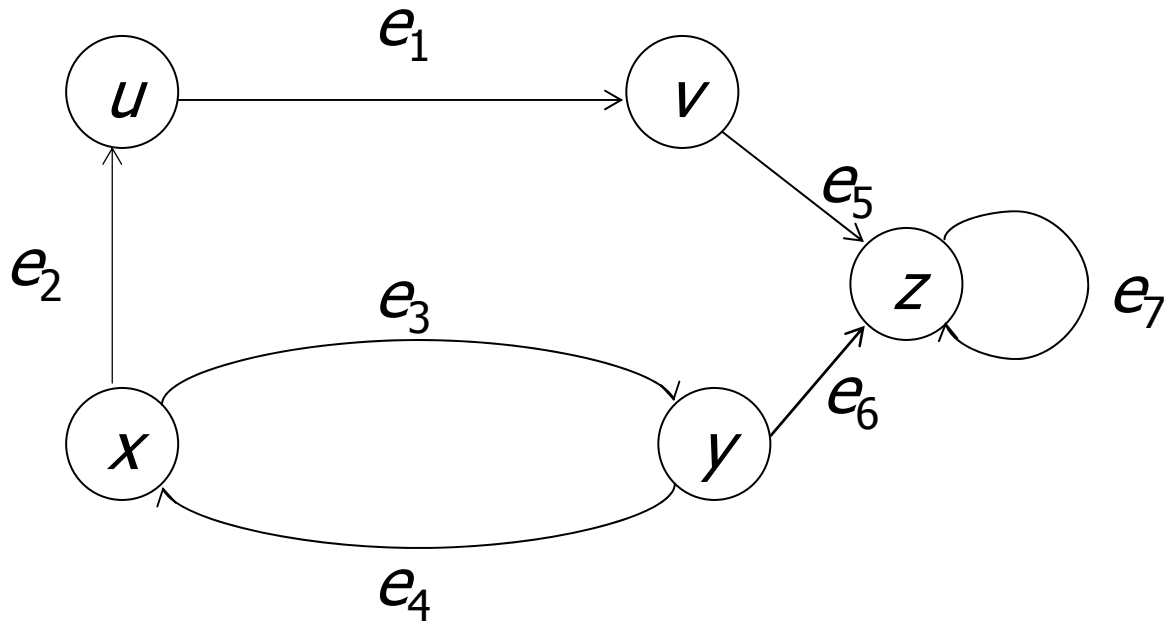


KHÁI NIỆM VÀ THUẬT NGỮ

Ví dụ 2: Đồ thị có hướng: $V = \{u, v, x, y, z\}$

$E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$

$e_3=(x, y)$
$e_4=(y, x)$
$e_7=(z, z)$



KHÁI NIỆM VÀ THUẬT NGỮ

- e_7 = (z, z) là cạnh khuyên
- e_3= (x, y) và e_4=(x, y) là hai cạnh song song
- Một đồ thị không có cạnh khuyên hoặc cạnh song song gọi là đơn đồ thị (simple graph), ngược lại gọi là đa đồ thị (multigraph)

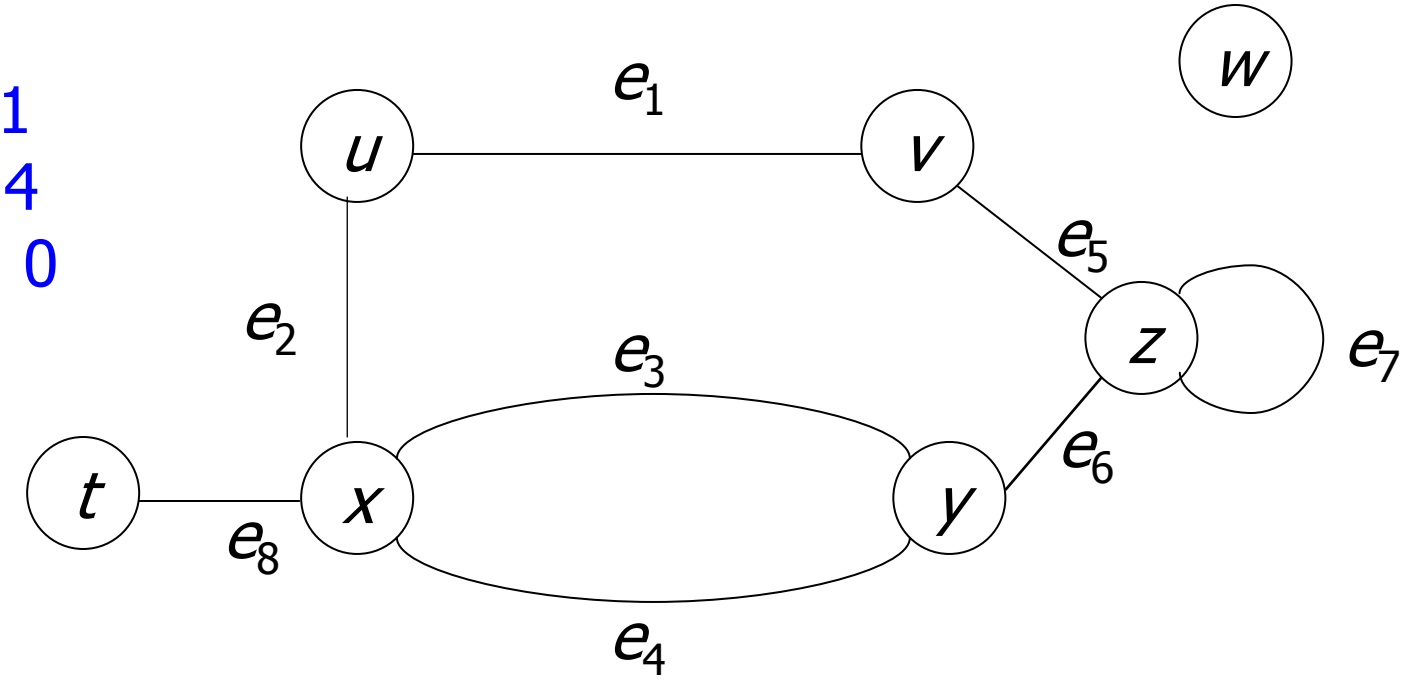
KHÁI NIỆM VÀ THUẬT NGỮ

- Đỉnh u và v là kề nhau (adjacent) nếu có cạnh $e = (u, v)$, cạnh e gọi là liên thuộc với u và v
- Bậc (degree) của đỉnh v trong đồ thị vô hướng là số cạnh liên thuộc với nó, ký hiệu deg(v), đỉnh bậc 0 gọi là đỉnh cô lập, đỉnh bậc 1 gọi là đỉnh treo
- Bán bậc ra (bán bậc vào) của đỉnh v trong đồ thị có hướng là số cạnh đi ra khỏi nó (đi vào nó) và ký hiệu $deg^+(v)$ ($deg(v)$)

KHÁI NIỆM VÀ THUẬT NGỮ

Ví dụ 3: Bậc của các đỉnh đồ thị vô hướng

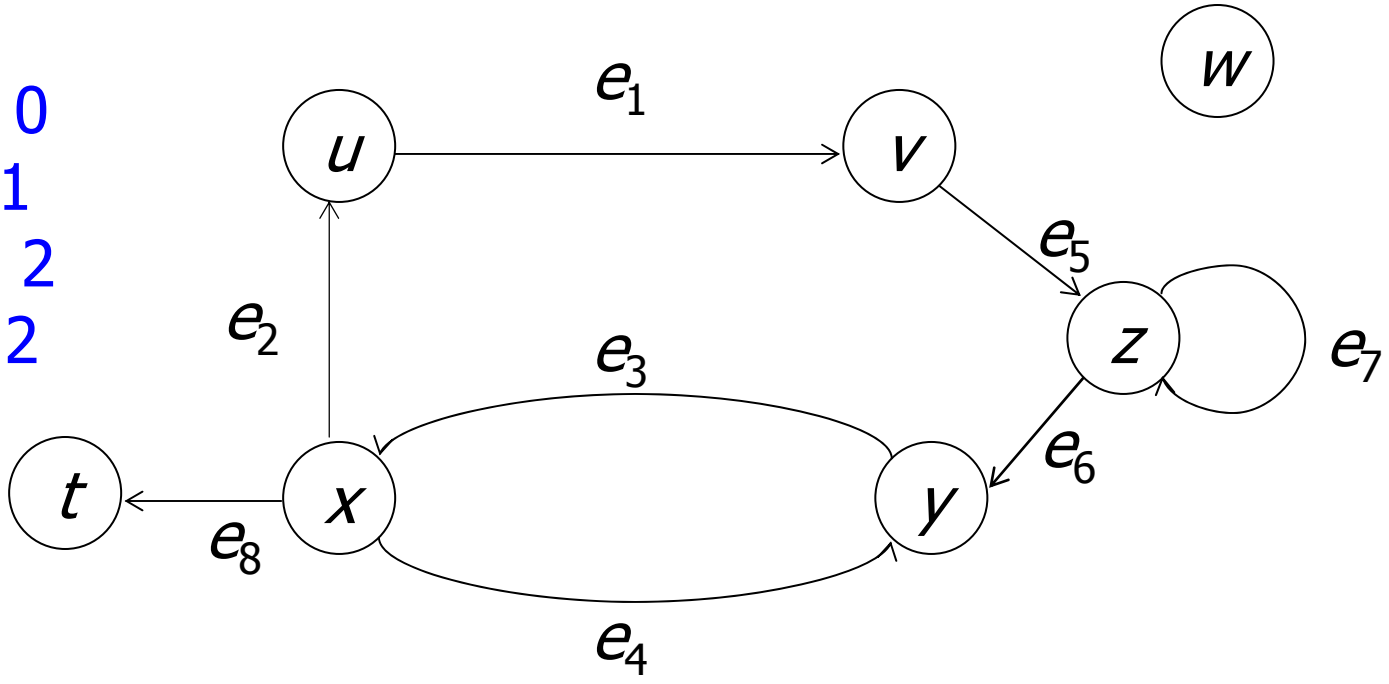
deg(*t*)= 1
deg(z)= 4
deg(w)= 0



KHÁI NIỆM VÀ THUẬT NGỮ

Ví dụ 4: Bán bậc của các đỉnh đồ thị có hướng

$deg^+(t) = 0$
$deg^-(t) = 1$
$deg^+(z) = 2$
$deg^-(z) = 2$

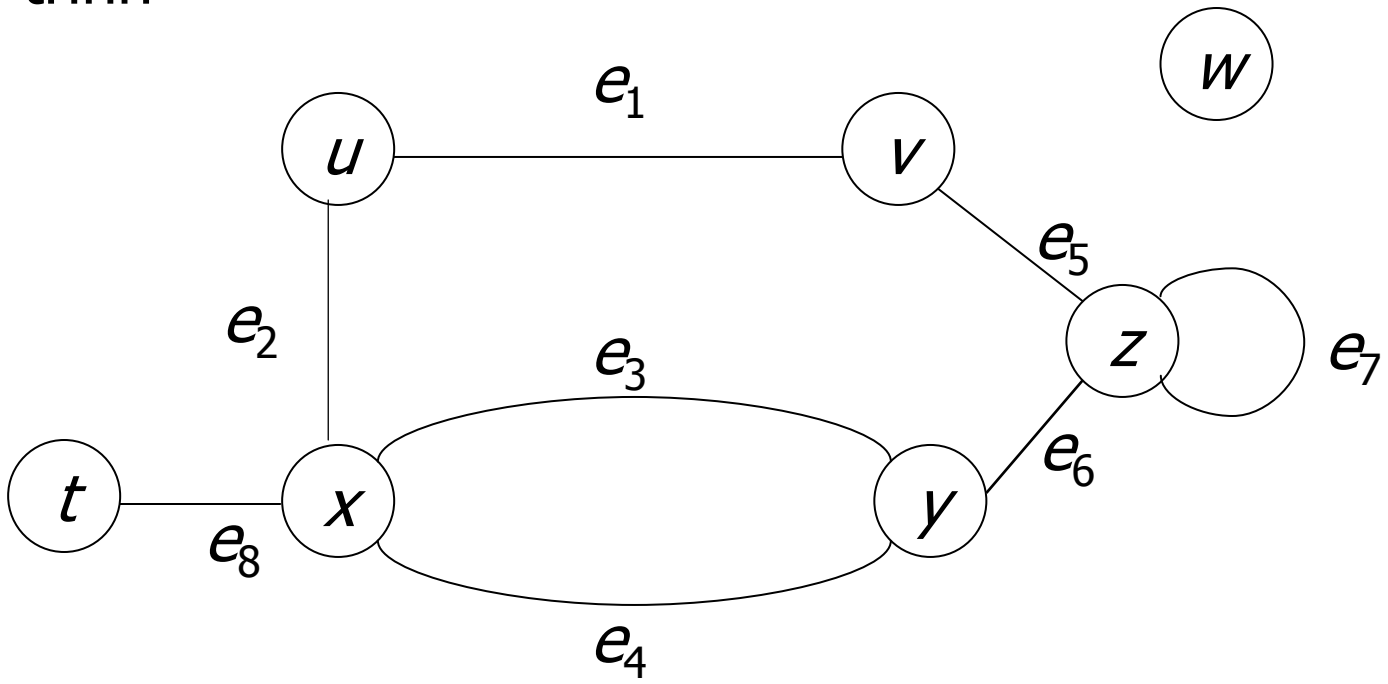


KHÁI NIỆM VÀ THUẬT NGỮ

- Đường đi độ dài n từ đỉnh x_0 đến đỉnh x_n trong một đồ thị là dãy $P = x_0, x_1, \ldots, x_n$ trong đó mỗi (x_i, x_{i+1}) là một cạnh
- Đường đi có đỉnh đầu x_0 trùng với đỉnh cuối x_n gọi là chu trình
- Đường đi hay chu trình gọi là đơn nếu không có đỉnh lặp lại (trừ đỉnh đầu và cuối nếu là chu trình)

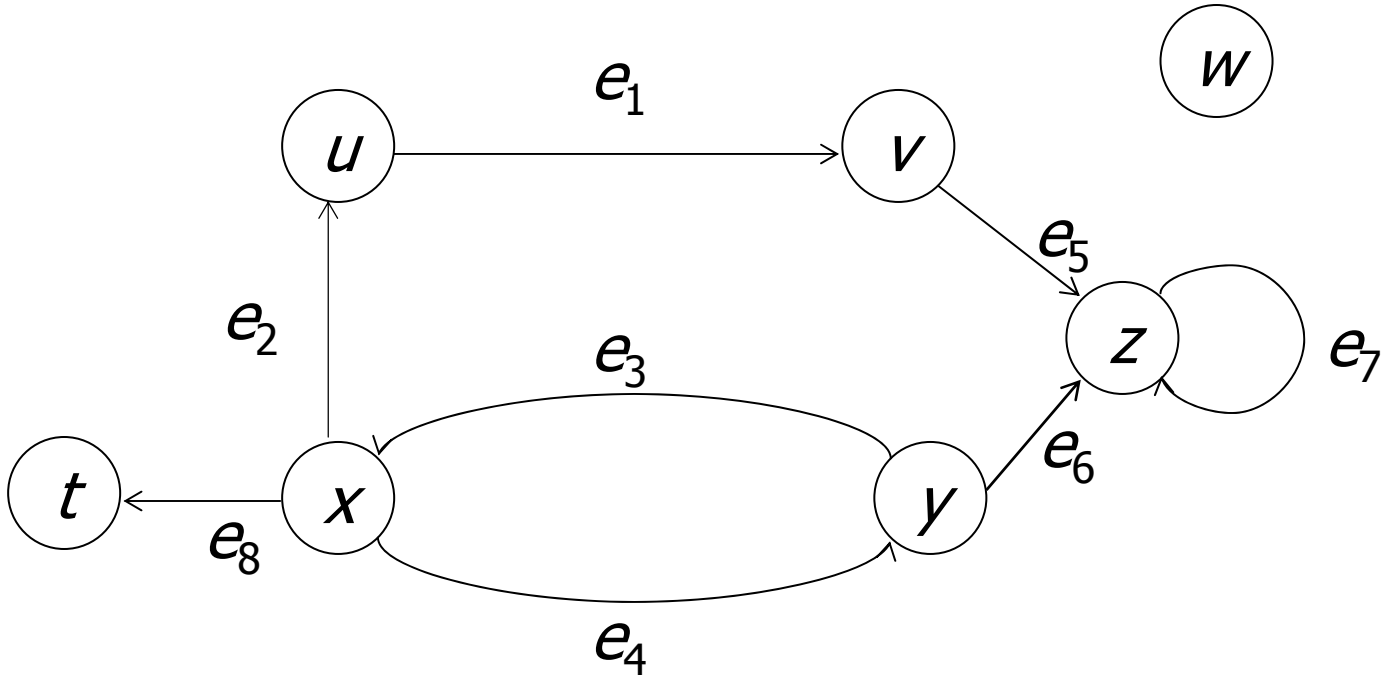
KHÁI NIỆM VÀ THUẬT NGỮ

Ví dụ 5: $P = u, v, z, y$ là một đường đi và $C = u, v, z, y, x, u$ là một chu trình



KHÁI NIỆM VÀ THUẬT NGỮ

Ví dụ 6: $P = x, u, v, z$ là một đường đi và $C = x, y, x$ là một chu trình

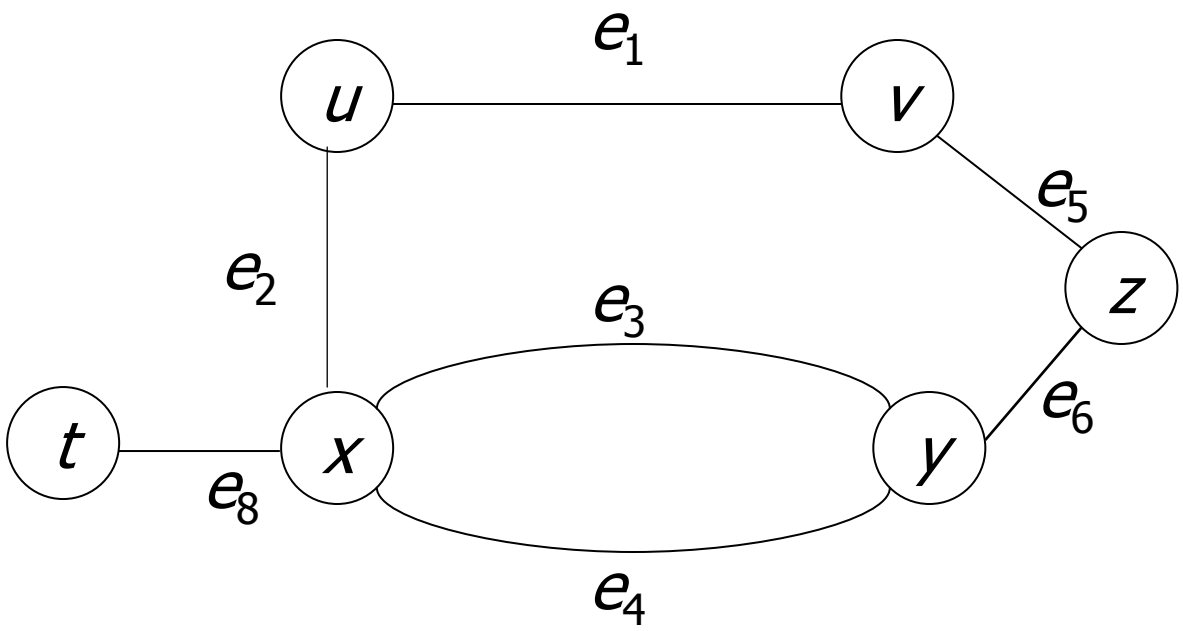


KHÁI NIỆM VÀ THUẬT NGỮ

- Một đồ thị được gọi là liên thông nếu luôn tìm được đường đi giữa hai đỉnh bất kỳ của nó

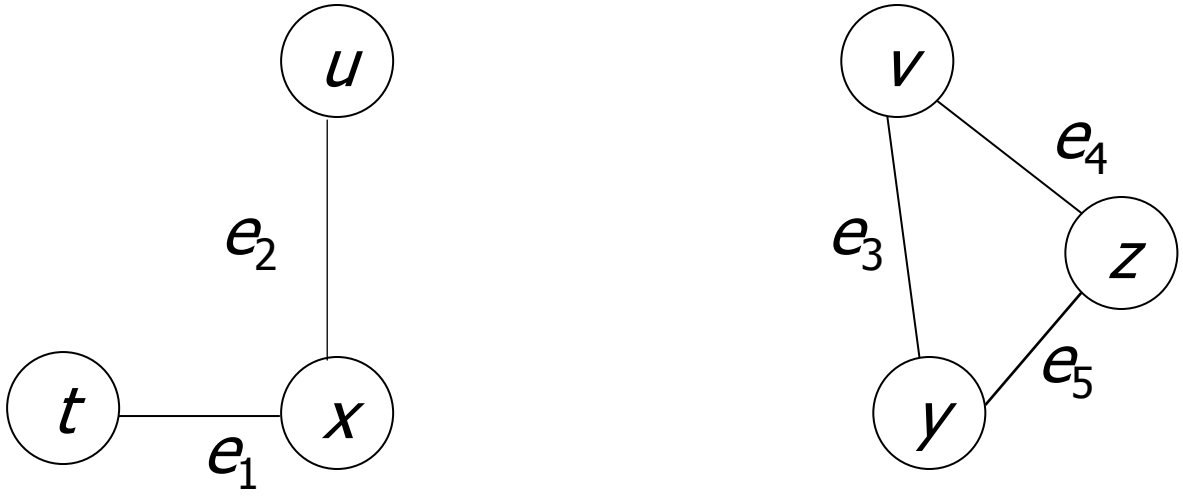
KHÁI NIỆM VÀ THUẬT NGỮ

Ví dụ 7: Đồ thị là liên thông



KHÁI NIỆM VÀ THUẬT NGỮ

Ví dụ 8: Đồ thị không liên thông



BIỂU DIỄN ĐỒ THI

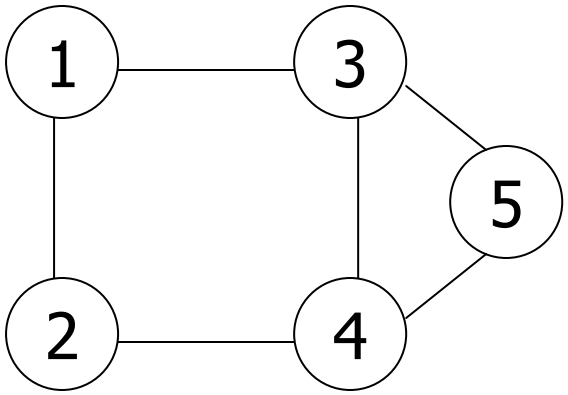
- Biểu diễn bằng danh sách kề (adjacency list)
- Biểu diễn bằng ma trận kề (adjacency matrix)
- So sánh các phương pháp biểu diễn đồ thị

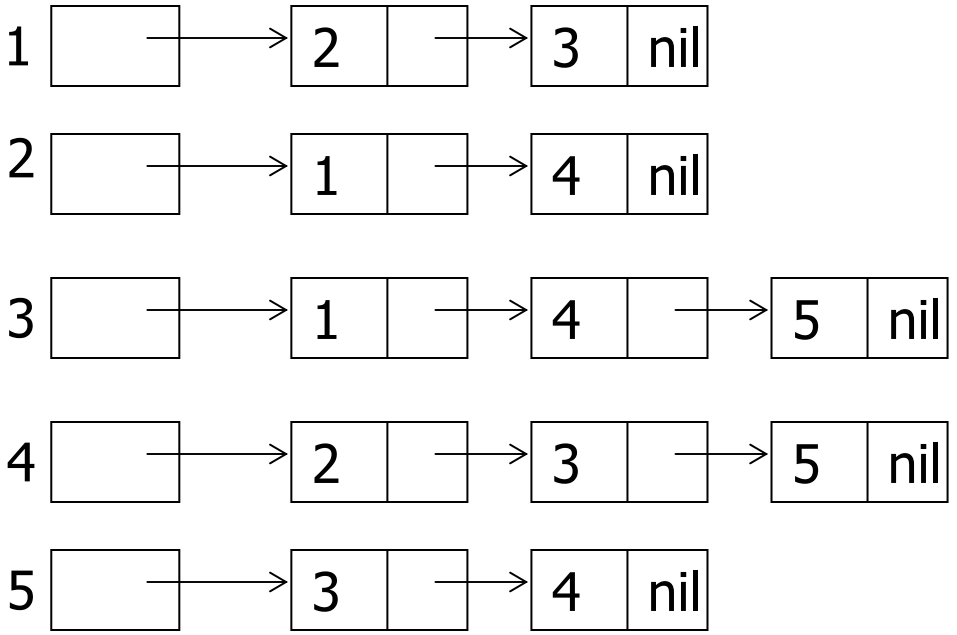
DANH SÁCH KỀ

- Danh sách kề của đỉnh u: adj(u) = $\{v \in V \mid (u, v) \in E\}$
- Có thể biểu diễn đồ thị $G = (V, E)$ như một tập các danh sách kề bằng cách lưu trữ mỗi đỉnh $u \in V$ cùng với danh sách các đỉnh kề với u

DANH SÁCH KỀ

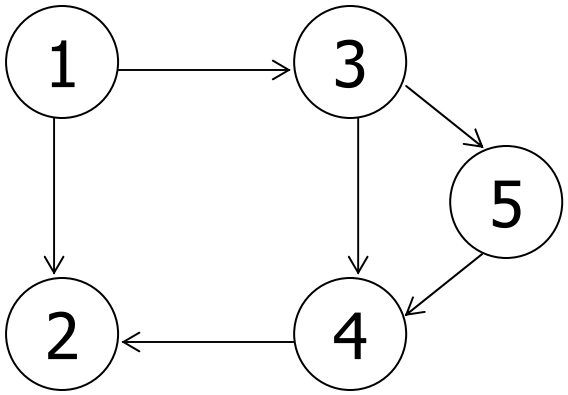
Ví dụ 1: Đồ thị vô hướng

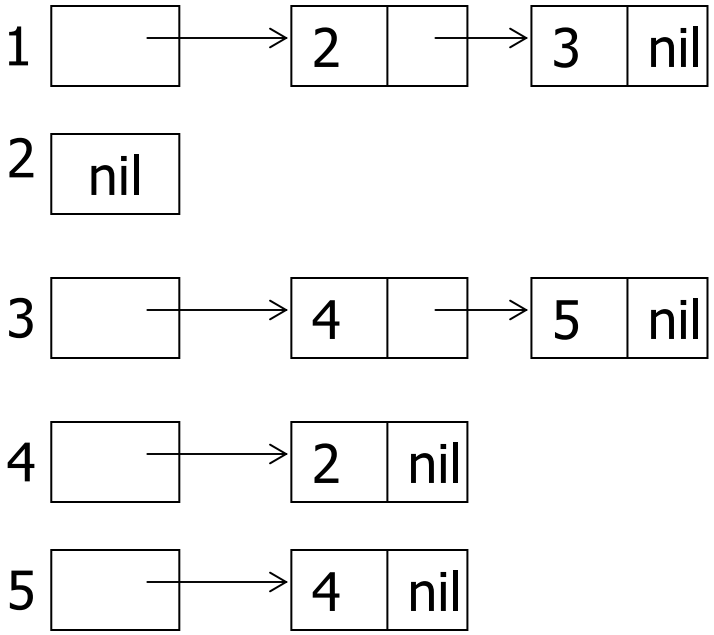




DANH SÁCH KỀ

Ví dụ 2: Đồ thị có hướng





MA TRẬN KỀ

- Cho đơn đồ thị $G = (V, E)$, với tập đỉnh $V = \{1, 2, ..., n\}$, ma trận kề của G là

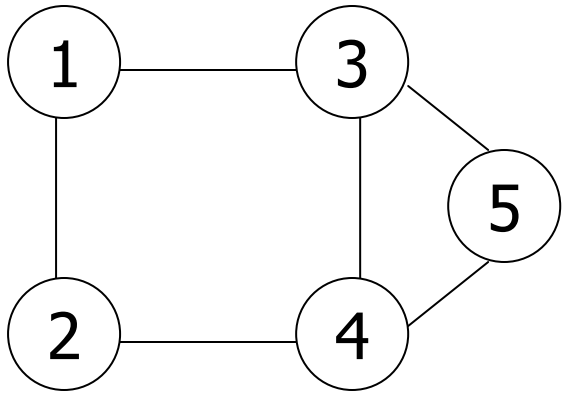
  $A = \{a_{ij} \mid i, j = 1, 2, ..., n\}$, $a_{ij} = 0$ nếu $(i, j) \notin E$ và $a_{ij} = 1$ nếu $(i, j) \in E$

- Nếu G là đa đồ thị thì

  $aij = 0$ nếu $(i, j) \notin E$ và $a_{ij} = k$ nếu có k cạnh nối hai đỉnh i và j

MA TRẬN KỀ

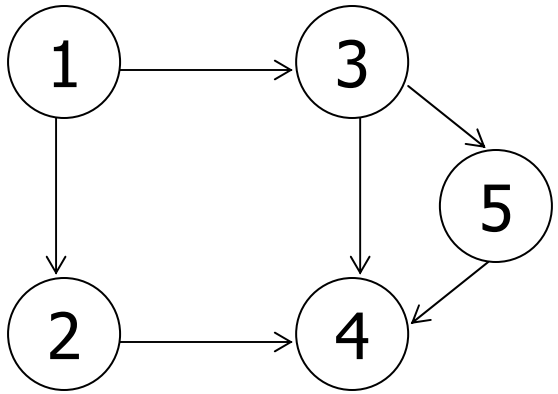
Ví dụ 1: Đồ thị vô hướng



$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

MA TRẬN KỀ

Ví dụ 2: Đồ thị có hướng



$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

MA TRẬN KỀ

- Ma trận kề của đồ thị vô hướng đối xứng

  $a_{ij} = a_{ji}$, $i, j = 1, 2, 3, \ldots, n$

- Tổng các phần tử trên dòng i (cột j) của ma trận kề là bậc của đỉnh i (đỉnh j)

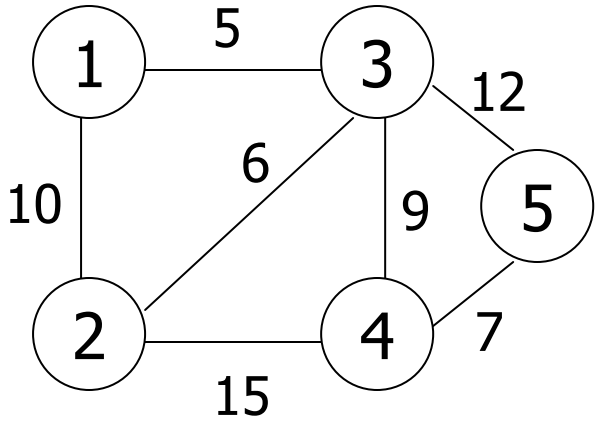
MA TRẬN KỀ

- Đồ thị có trọng số (weighted graph) là đồ thị mà mỗi cạnh (i, j) được gán một số thực $w(i, j)$
- Một đồ thị có trọng số với n đỉnh có thể được biểu diễn bởi ma trận trọng số

  $C = \{c_{ij}: i, j = 1,2,...,n\}$, trong đó $c_{ij} = w(i, j)$ nếu có cạnh (i, j) và $c_{ij} = 0, \infty$, hoặc $-\infty$ nếu không có cạnh (i, j)

MA TRẬN KỀ

Ví dụ 3: Ma trận trọng số của đồ thị vô hướng



$$\begin{bmatrix} 0 & 10 & 5 & 0 & 0 \\ 10 & 0 & 6 & 15 & 0 \\ 5 & 6 & 0 & 9 & 12 \\ 0 & 15 & 9 & 0 & 7 \\ 0 & 0 & 12 & 7 & 0 \end{bmatrix}$$

SO SÁNH CÁC CÁCH BIỂU DIỄN

Biểu diễn đồ thị vô hướng bằng danh sách và ma trận

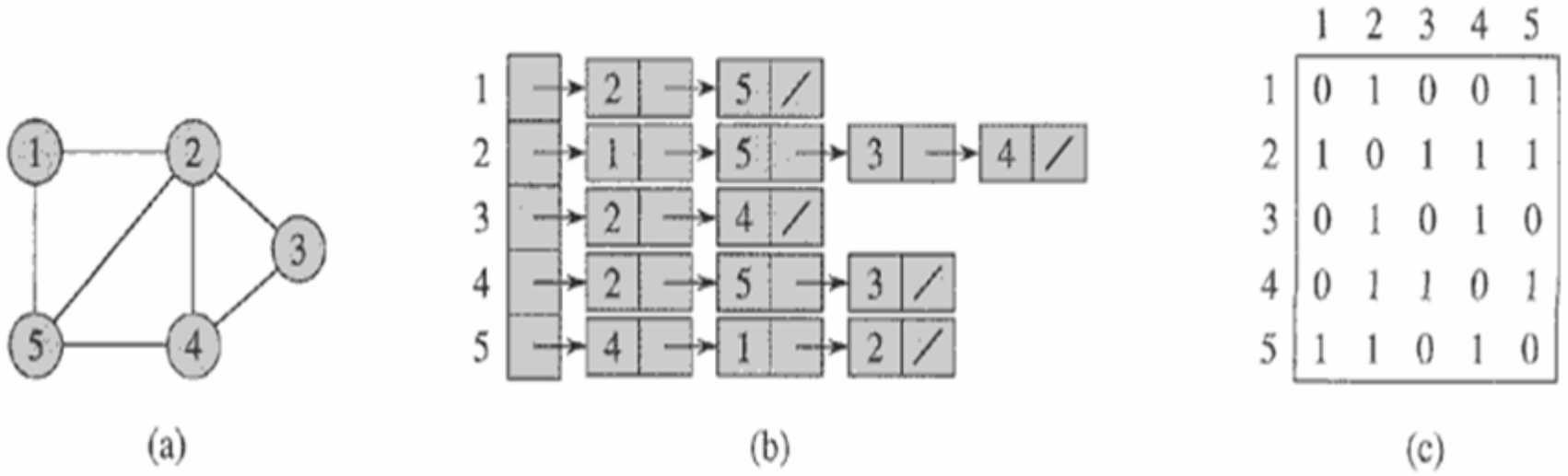
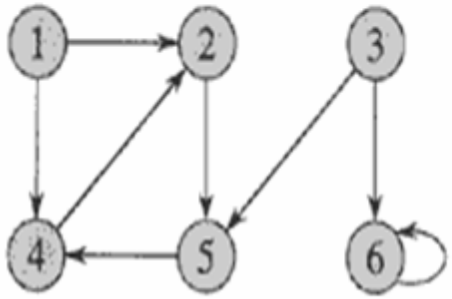


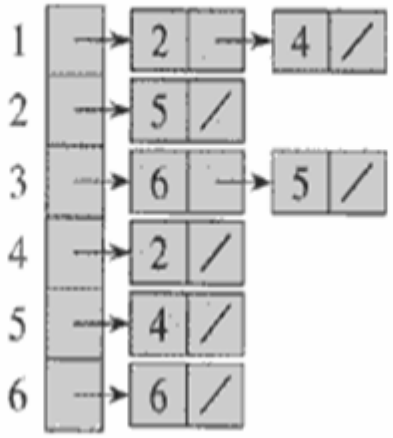
Figure 22.1 Two representations of an undirected graph. **(a)** An undirected graph G having five vertices and seven edges. **(b)** An adjacency-list representation of G. **(c)** The adjacency-matrix representation of G.

SO SÁNH CÁC CÁCH BIỂU DIỄN

Biểu diễn đồ thị có hướng bằng danh sách và ma trận



(a)



(b)

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

(c)

Figure 22.2 Two representations of a directed graph. **(a)** A directed graph G having six vertices and eight edges. **(b)** An adjacency-list representation of G. **(c)** The adjacency-matrix representation of G.

SO SÁNH CÁC CÁCH BIỂU DIỄN

- Chi phí bộ nhớ cho ma trận là $O(|V|^2)$ và cho danh sách là $O(|V| + 2|E|)$
- Chi phí xử lý khi dùng ma trận là O(1) và khi dùng danh sách là $O|V|$

TÌM KIẾM THEO CHIỀU RỘNG
(Breadth-First Search-BFS)

- Thuật toán BFS
- Phân tích BFS
- Đường đi ngắn nhất

THUẬT TOÁN BFS

Ý tưởng thuật toán

- Bắt đầu tìm kiếm từ đỉnh *s* cho trước tuỳ ý
- Tại thời điểm đã tìm thấy *u,* thuật toán tiếp tục tìm kiếm tập tất cả các đỉnh kề với *u*
- Thực hiện quá trình này cho các đỉnh còn lại

THUẬT TOÁN BFS

Ý tưởng thuật toán

- Dùng một hàng đợi để duy trì trật tự tìm kiếm theo chiều rộng
- Dùng các màu để không lặp lại các đỉnh tìm kiếm
- Dùng một mảng để xác định đường đi ngắn nhất từ *s* đến các đỉnh đã được tìm kiếm
- Dùng một mảng để lưu trữ đỉnh đi trước của đỉnh được tìm kiếm

THUẬT TOÁN BFS

```
BFS(G, s)
 1   for each vertex u ∈ V[G] − {s}
 2         do color[u] ← WHITE
 3            d[u] ← ∞
 4            π[u] ← NIL
 5   color[s] ← GRAY
 6   d[s] ← 0
 7   π[s] ← NIL
 8   Q ← ∅
 9   ENQUEUE(Q, s)
10   while Q ≠ ∅
11         do u ← DEQUEUE(Q)
12            for each v ∈ Adj[u]
13                do if color[v] = WHITE
14                      then color[v] ← GRAY
15                           d[v] ← d[u] + 1
16                           π[v] ← u
17                           ENQUEUE(Q, v)
18            color[u] ← BLACK
```

THUẬT TOÁN BFS

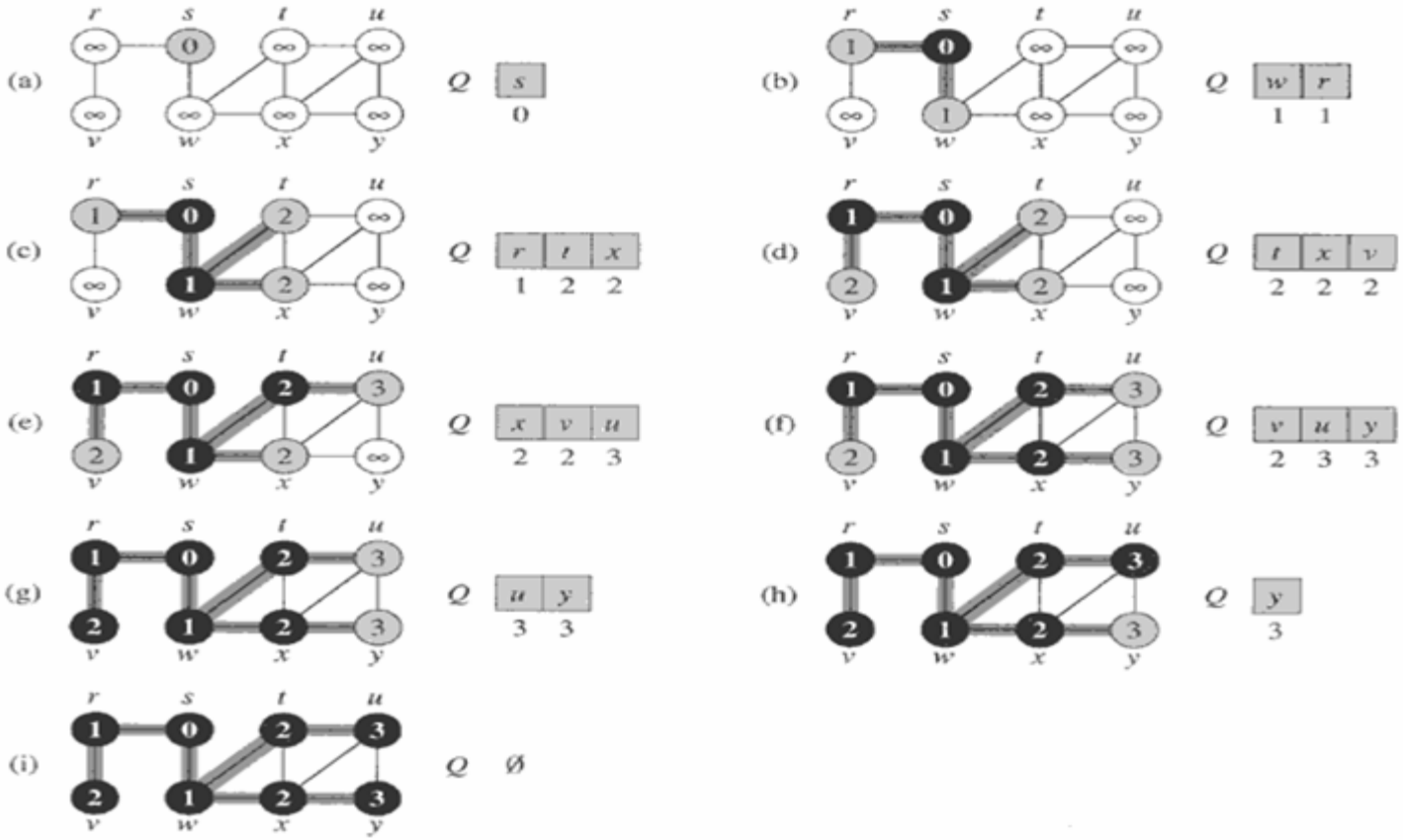


Figure 22.3 The operation of BFS on an undirected graph. Tree edges are shown shaded as they are produced by BFS. Within each vertex u is shown $d[u]$. The queue Q is shown at the beginning of each iteration of the **while** loop of lines 10–18. Vertex distances are shown next to vertices in the queue.

PHÂN TÍCH BFS

- Tổng phí khởi tạo là O(*V*)
- Mỗi thao tác trên hàng đợi là O(1), vì vậy tổng thời gian cho thao tác trên hàng đợi là O(*V*)
- Tổng thời gian chi phí cho quét các danh sách kề là O(*E*)
- Tổng thời gian chạy của BFS là O(*V*+*E*)

ĐƯỜNG ĐI NGẮN NHẤT

- Khoảng cách đường đi ngắn nhất (shortest-path distance) từ s đến v là số cạnh ít nhất trong các đường đi từ s đến v, ký hiệu $\delta(s, v)$
- Qui ước $\delta(s, v) = \infty$ nếu không có đường đi từ s đến v
- Một đường đi độ dài bằng $\delta(s, v)$ từ s đến v được gọi là đường đi ngắn nhất từ s đến v

ĐƯỜNG ĐI NGẮN NHẤT

- **Định lý**: Cho BFS chạy trên một đồ thị từ đỉnh *s,* thì thuật toán tìm kiếm được mọi đỉnh v mà có thể đạt được từ s, khi kết thúc, BFS xác định các đường đi ngắn nhất từ s đến v sao cho $d[v] = \delta(s, v)$ với mọi $v \in V$

ĐƯỜNG ĐI NGẮN NHẤT

```
PRINT-PATH(G, s, v)
1  if v = s
2      then print s
3      else if π[v] = NIL
4              then print "no path from" s "to" v "exists"
5              else PRINT-PATH(G, s, π[v])
6                   print v
```

TÌM KIẾM THEO CHIỀU SÂU
(Depth-First Search-DFS)

- Thuật toán DFS
- Phân tích DFS

THUẬT TOÁN DFS

Ý tưởng thuật toán

- Bắt đầu tìm kiếm từ một đỉnh u nào đó
- Chọn đỉnh kề v tùy ý của u để tiếp tục quá trình tìm kiếm và lặp lại quá trình tìm kiếm này đối với v

THUẬT TOÁN DFS

Ý tưởng thuật toán

- Dùng các màu để không lặp lại các đỉnh tìm kiếm
- Dùng các biến thời gian để xác định các thời điểm phát hiện và hoàn thành tìm kiếm của một đỉnh
- Dùng một mảng để lưu trữ đỉnh đi trước của đỉnh được tìm kiếm

THUẬT TOÁN DFS

```
DFS(G)
1   for each vertex u ∈ V[G]
2       do color[u] ← WHITE
3          π[u] ← NIL
4   time ← 0
5   for each vertex u ∈ V[G]
6       do if color[u] = WHITE
7          then DFS-VISIT(u)
```

THUẬT TOÁN DFS

```
DFS-VISIT(u)
1  color[u] ← GRAY          ▷ White vertex u has just been discovered.
2  time ← time + 1
3  d[u] ← time
4  for each v ∈ Adj[u]      ▷ Explore edge (u, v).
5       do if color[v] = WHITE
6            then π[v] ← u
7                 DFS-VISIT(v)
8  color[u] ← BLACK         ▷ Blacken u; it is finished.
9  f[u] ← time ← time + 1
```

THUẬT TOÁN DFS

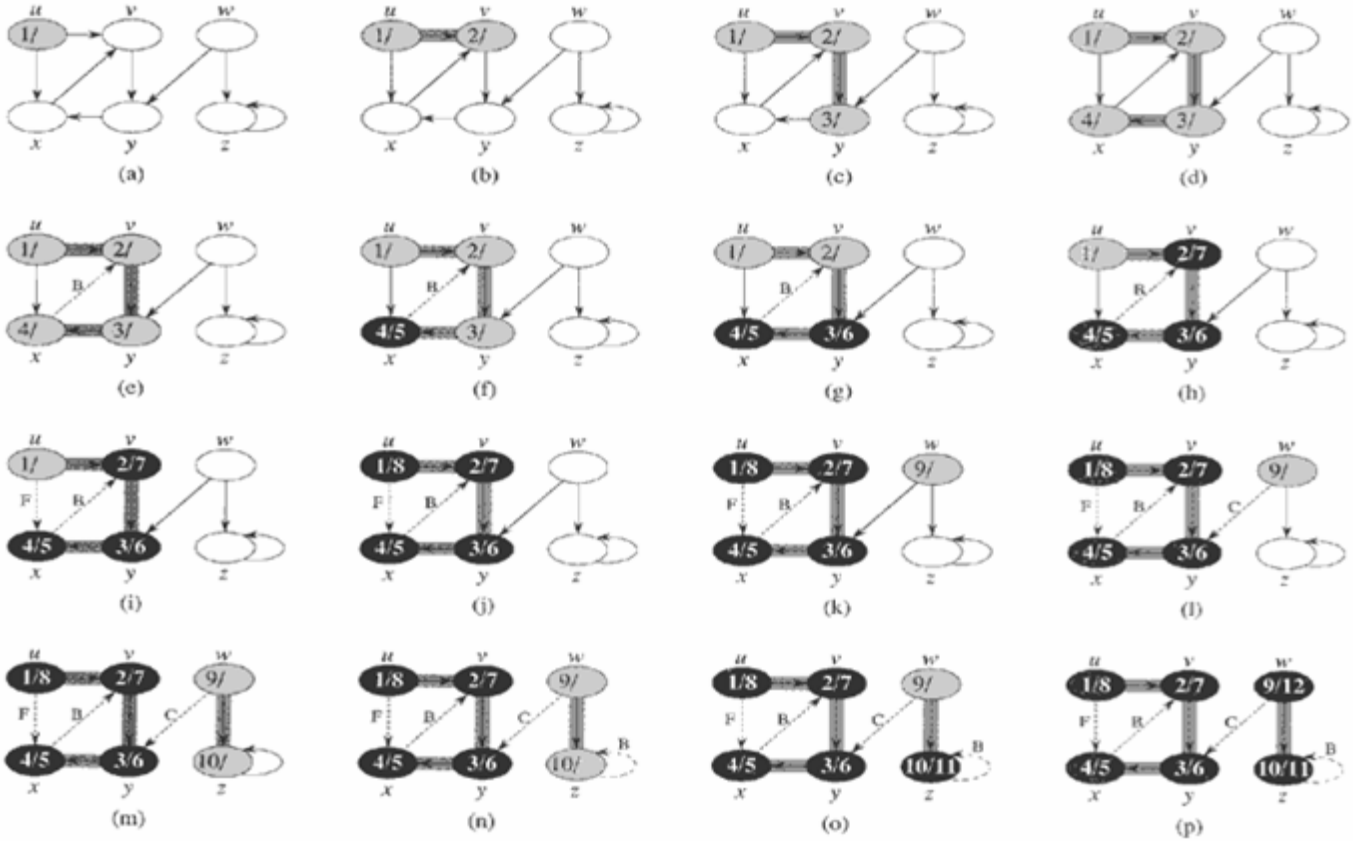


Figure 22.4 The progress of the depth-first-search algorithm DFS on a directed graph. As edges are explored by the algorithm, they are shown as either shaded (if they are tree edges) or dashed (otherwise). Nontree edges are labeled B, C, or F according to whether they are back, cross, or forward edges. Vertices are timestamped by discovery time/finishing time.

PHÂN TÍCH DFS

- Nếu chưa tính thời gian thực thi DFS-VISIT, vòng lặp 1-3 và 5-7 có chi phí là $O(V)$
- Trong một lần thực thi DFS-VISIT(u), vòng lặp 4-7 thực thi trong $|Adj[u]|$ lần
- Vì $\Sigma_{u \in V} |Adj[u]| = O(E)$, nên tổng chi phí thực thi dòng 4-7 của DFS-VISIT là $O(E)$.
- Vậy thời gian chạy của DFS là $O(V+E)$

CÂY BAO TRÙM NHỎ NHẤT

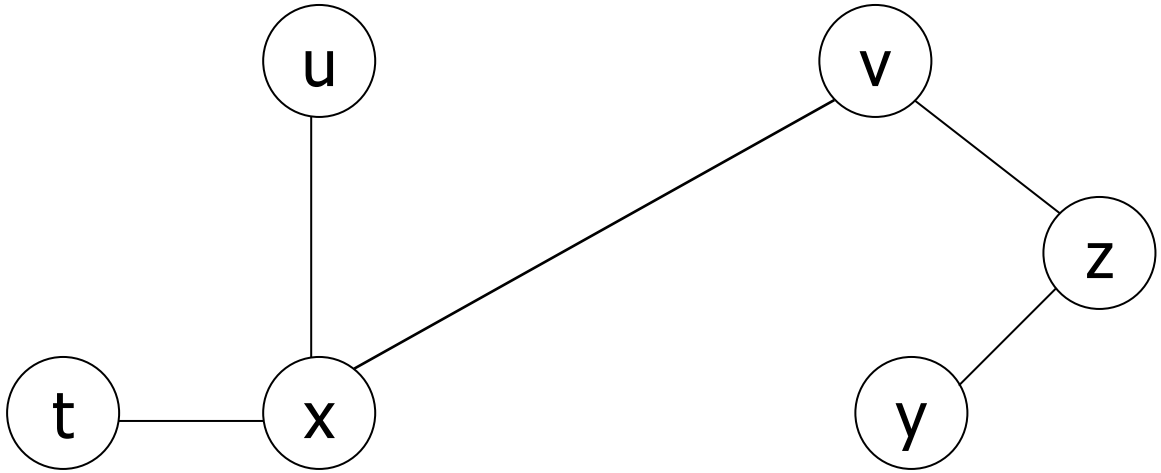
- Cây và cây bao trùm
- Cây bao trùm nhỏ nhất

CÂY VÀ CÂY BAO TRÙM

- Định nghĩa cây
- Các tính chất của cây
- Cây bao trùm

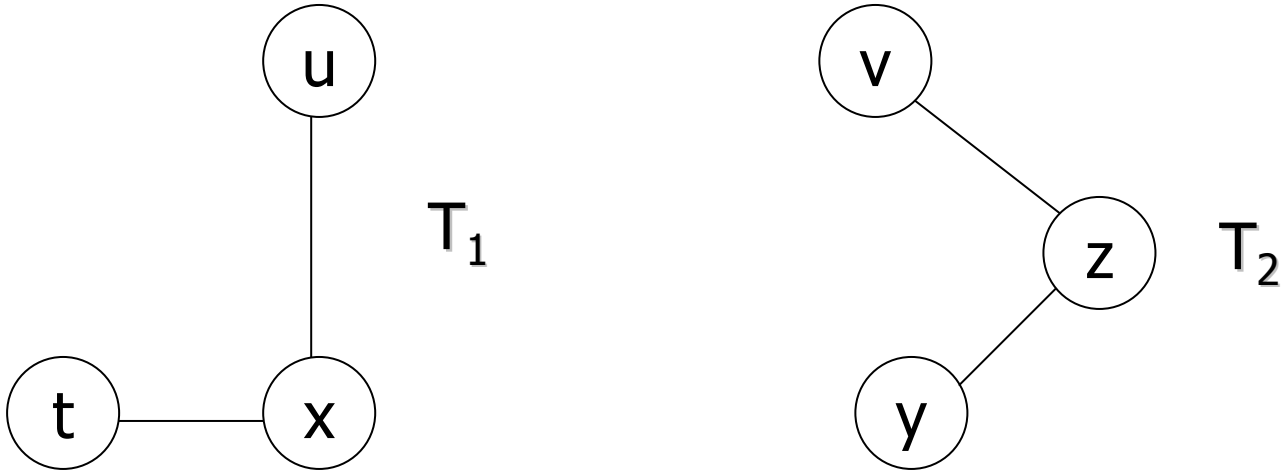
ĐINH NGHĨA CÂY

- Cây tự do (free tree) là một đồ thị vô hướng liên thông không có chu trình (rừng là tập nhiều cây)



ĐINH NGHĨA CÂY

- Một rừng gồm hai cây

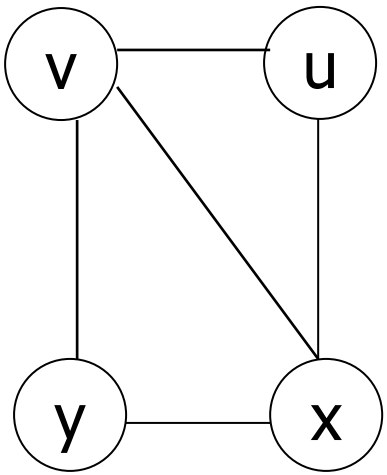


CÁC TÍNH CHẤT CỦA CÂY

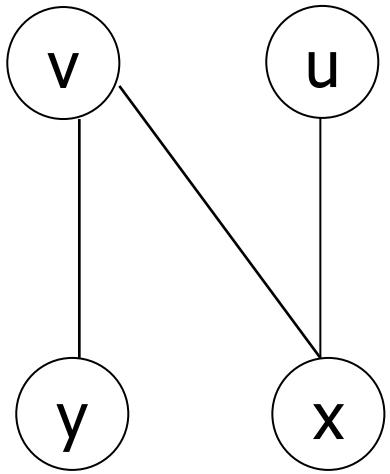
- **Định lý**: Đồ thị T vô hướng n đỉnh là một cây nếu thỏa một trong các điều kiện sau
  - T không chứa chu trình và có n -1 cạnh
  - T liên thông và có n -1 cạnh
  - T liên thông và mỗi cạnh của nó đều là cầu
  - Hai đỉnh bất kỳ được nối với nhau bằng một đường đi duy nhất
  - T không chứa chu trình nhưng nếu thêm vào một cạnh thì có một chu trình duy nhất

CÂY BAO TRÙM

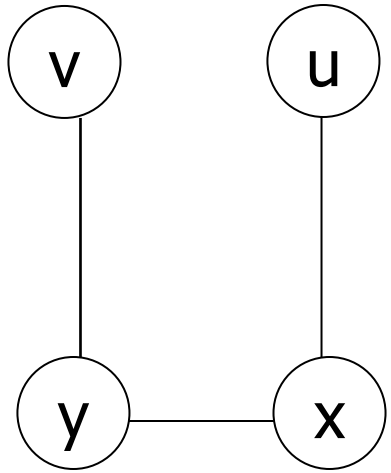
- Cây T= (V, F) được gọi là một cây bao trùm (spanning tree) của đồ thị vô hướng liên thông G = (V, E) nếu $F \subseteq E$



G



Cây BT T_1



Cây BT T_2

CÂY BAO TRÙM

- Nhận xét
  - Một đồ thị có thể có nhiều cây bao trùm
  - Ví dụ đồ thị K_n (gồm n đỉnh và mỗi đỉnh đều có cạnh nối với n-1 đỉnh còn lại) có n^{n-2} cây bao trùm
  - Cây bao trùm của G = (V, E) là đồ thị V đỉnh liên thông ít cạnh nhất

CÂY BAO TRÙM NHỎ NHẤT

- Khái niệm
- Thuật giải Kruskal
- Thuật giải Prim

KHÁI NIỆM

- Cho G là một đồ thị vô hướng, liên thông có trọng số và T là một cây bao trùm của G
  - Trọng số của T, ký hiệu w(T), là tổng trọng số của tất cả các cạnh của nó: $w(T) = \Sigma_{e \in T} w(e)$
  - Bài toán: Tìm một cây bao trùm T có trọng số nhỏ nhất (minimum spanning tree-MST) của G

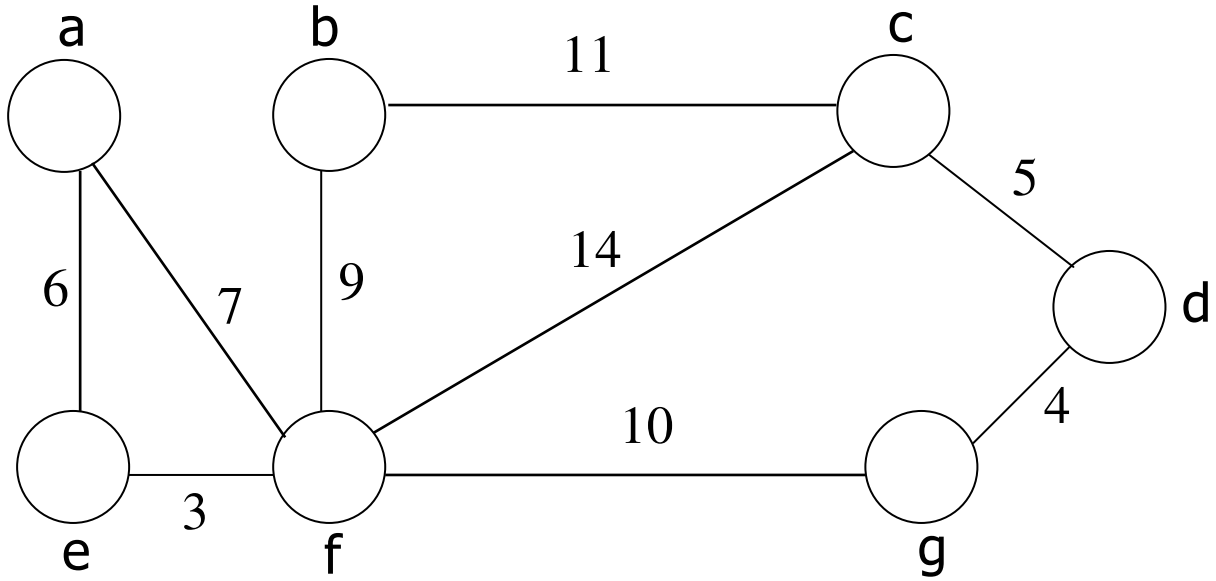
THUẬT GIẢI KRUSKAL

Ý tưởng

- Tại mỗi bước, thuật giải tìm một cạnh có trọng số nhỏ nhất thêm vào tập cạnh của cây bao trùm sao cho không gây ra chu trình
- Thuật giải dừng khi số cạnh của cây bằng số đỉnh của đồ thị trừ 1

THUẬT GIẢI KRUSKAL

- Đồ thị G có trọng số và các cạnh được sắp

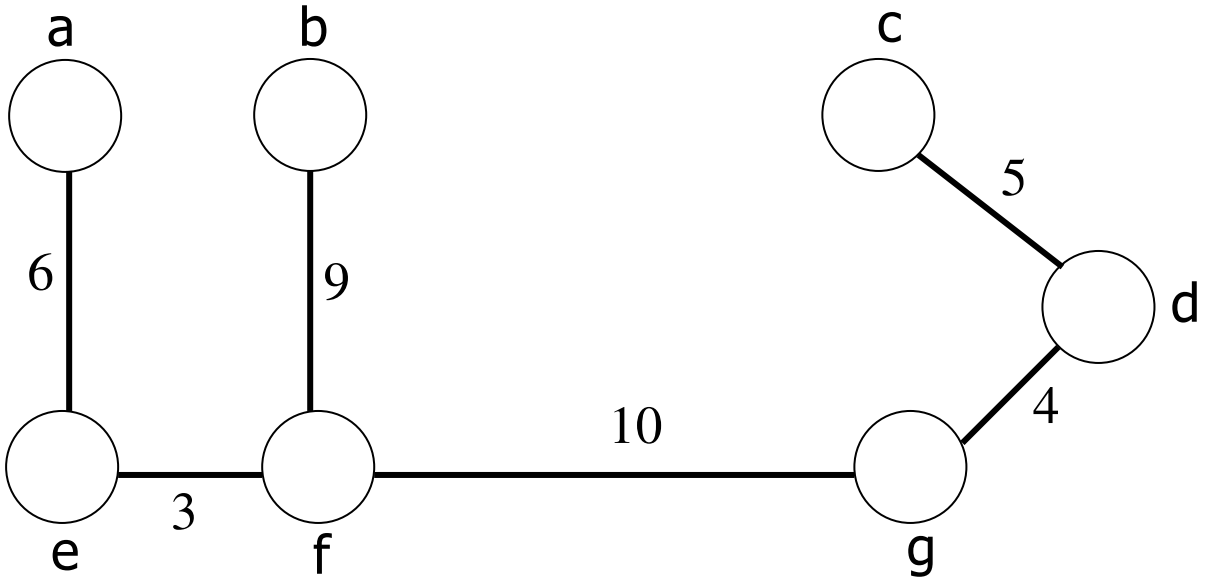


(e, f), (d, g), (c, d), (a, e), (a, f), (b, f), (f, g), (b, c), (c, f)

3, 4, 5, 6, 7, 9, 10, 11, 14

THUẬT GIẢI KRUSKAL

- Cây bao trùm nhỏ nhất của G



(e, f), (d, g), (c, d), (a, e), (b, f), (f, g)

3, 4, 5, 6, 9, 10

THUẬT GIẢI KRUSKAL

KRUSKAL(G, w) // G =(V, E) có n đỉnh

1. F $\leftarrow \varnothing$ // F là số cạnh của cây MST
2. Sort the edges of E into nondecreasing order by weight w
3. **while** $|F| < n-1$ and $E \neq \varnothing$ // thực hiện cho đến khi $|F| = n-1$ hoặc $E = \varnothing$
4. **do** $e \leftarrow x \mid w(x) = \min\{w(y), y \in E)\}$ ▷ e có trọng số bé nhất
5. $E \leftarrow E-\{e\}$
6. **if** $F \cup \{e\}$ not contain cycle **then** $F \leftarrow F \cup \{e\}$
7. **if** $|F| < n-1$
8. **then** G is not connected
9. **else return** T =(V, F)

THUẬT GIẢI KRUSKAL

- Thời gian sắp xếp là O(E lgE)
- Chi phí cho tất cả các lần lặp trong vòng lặp **while** 3-6 không quá $O(V^2)$
- Do đó, tổng chi phí là $O(E \lg E) + O(V^2)$

THUẬT GIẢI PRIM

Ý tưởng

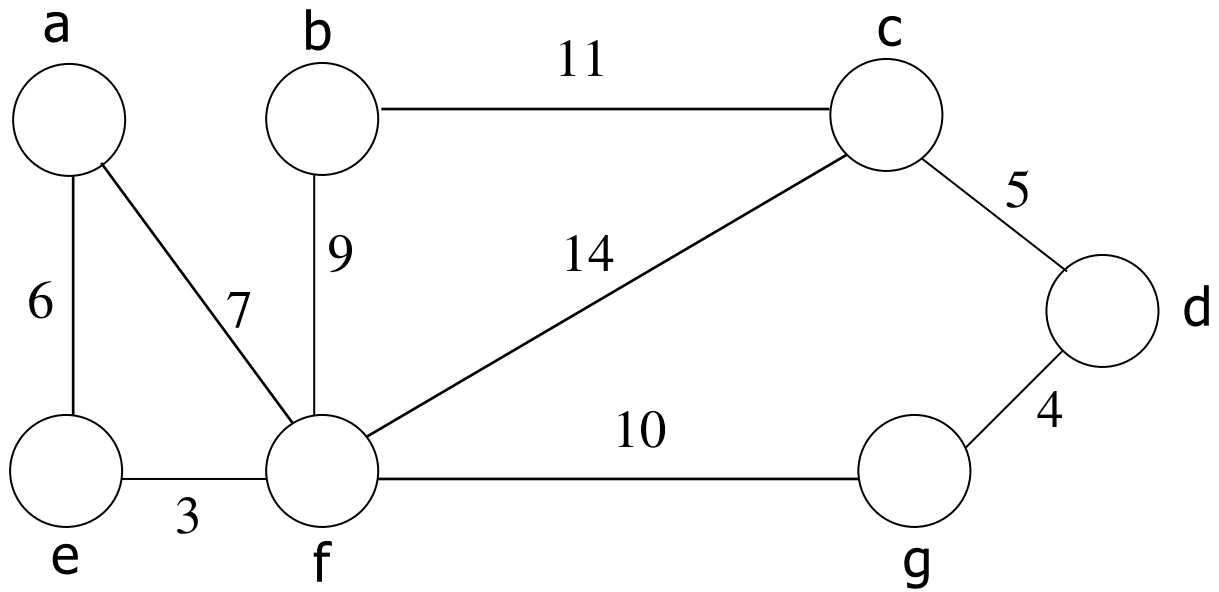
- Khởi đầu, thuật giải chọn một đỉnh bất kỳ của đồ thị làm đỉnh gốc của cây bao trùm bé nhất
- Tại mỗi bước chọn thêm một đỉnh của đồ thị mà trọng số cạnh nối nó với một đỉnh của cây là nhỏ nhất
- Thuật giải kết thúc khi tất cả các đỉnh của đồ thị đã được chọn

THUẬT GIẢI PRIM

```
MST-PRIM(G, w, s)
1.  for each u ∈ V[G]
2.        do  key[u] ← ∞ // key[u] là trọng số nhỏ nhất của cạnh nối u
3.            π[u] ← NIL  // với một đỉnh trong cây MSTđang xây dựng
4.  key[s] ←0
5.  Q ←V[G]
6.  while Q ≠ ∅
7.        do   u ←Extract-min(Q)
8.             for each v ∈ Adj[u]
9.                 do  if  v ∈ Q and w(u,v)<key[v]
10.                       then    π[v] ← u
11.                               key[v] ← w(u,v)
```

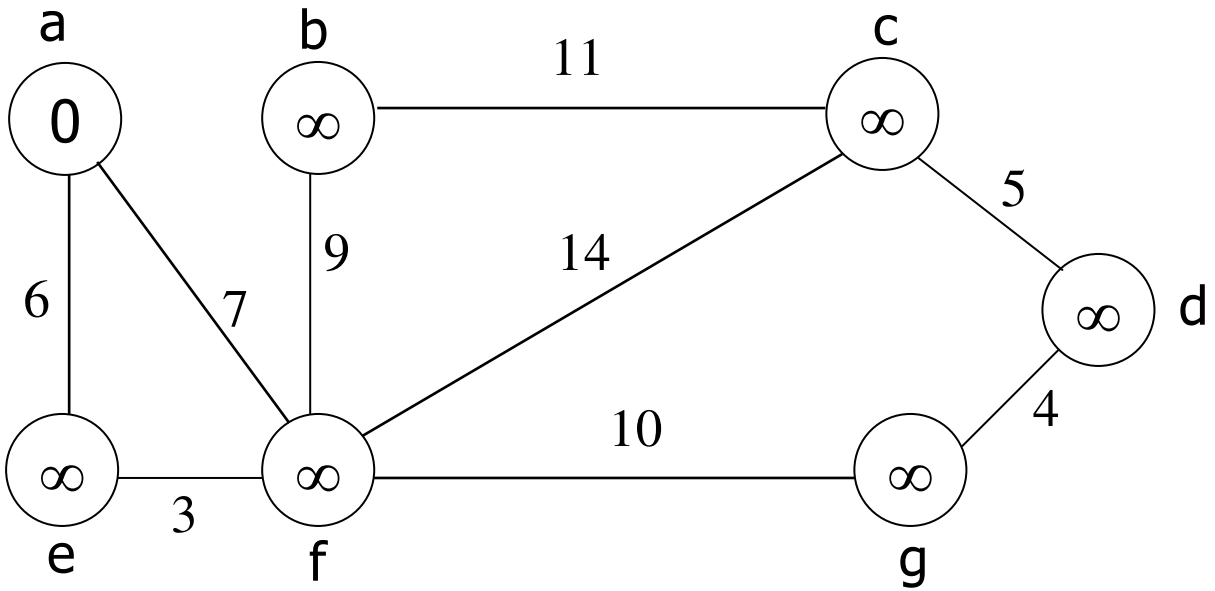
THUẬT GIẢI PRIM

- Đồ thị G có trọng số, lấy a làm đỉnh xuất phát



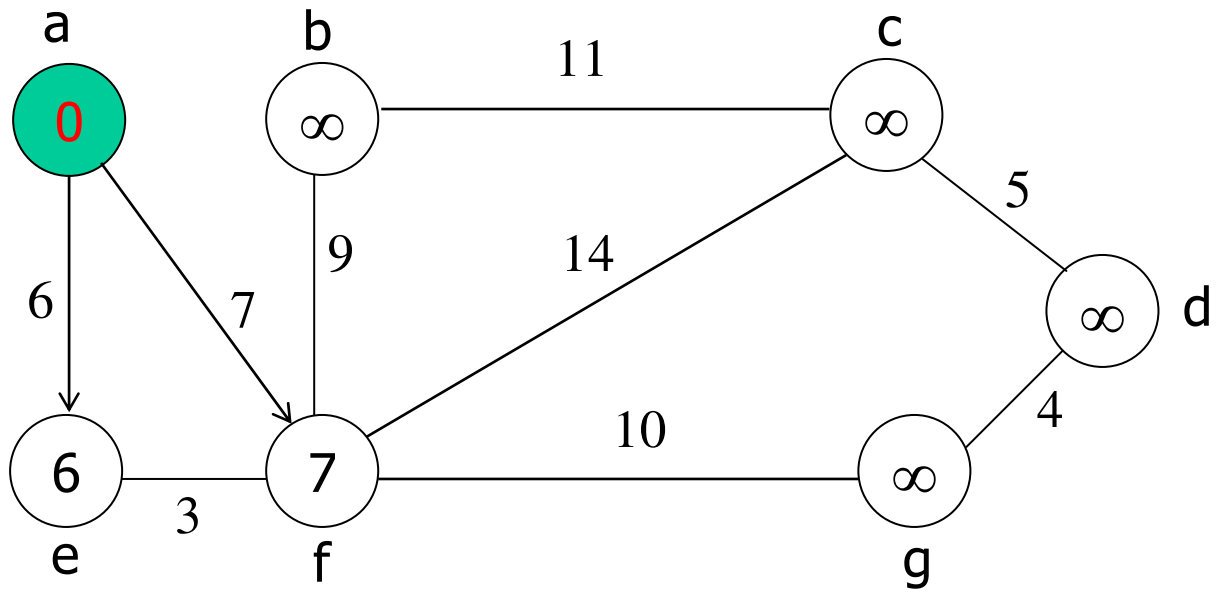
THUẬT GIẢI PRIM

- Key[a]=0, key[u]= ∞ với mọi u thuộc V



THUẬT GIẢI PRIM

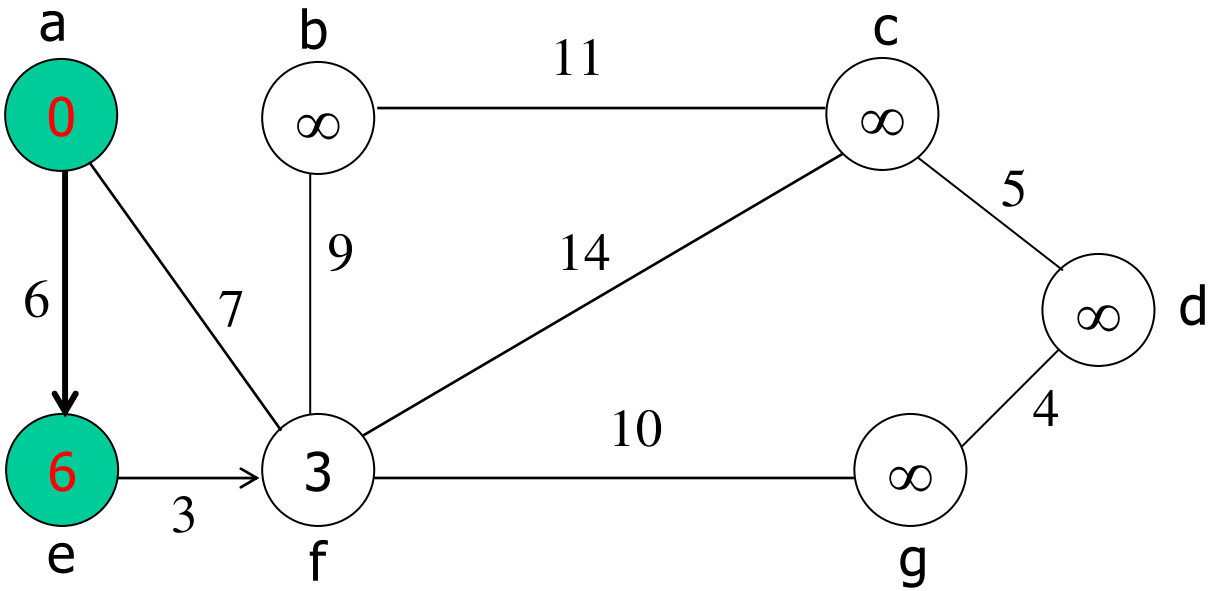
- Chọn a là đỉnh đầu tiên của MST(do key[a] =0 nhỏ nhất)



Cập nhật key[e]=6, key[f]=7

THUẬT GIẢI PRIM

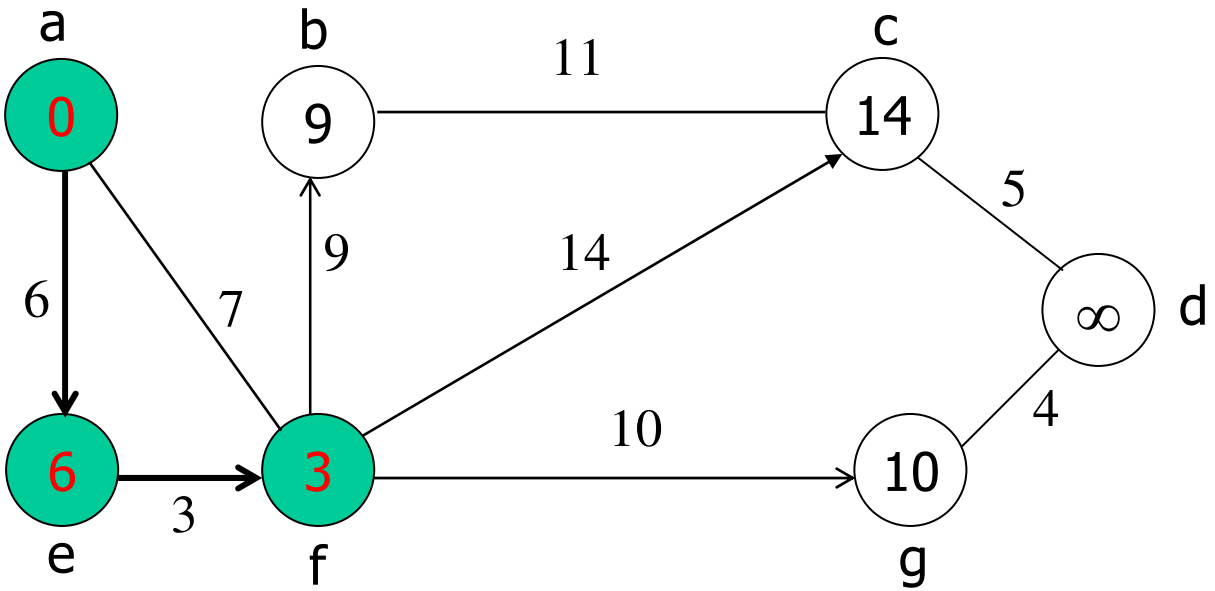
- Chọn e là đỉnh kế, key[e] =6



Cập nhật key[f]=3

THUẬT GIẢI PRIM

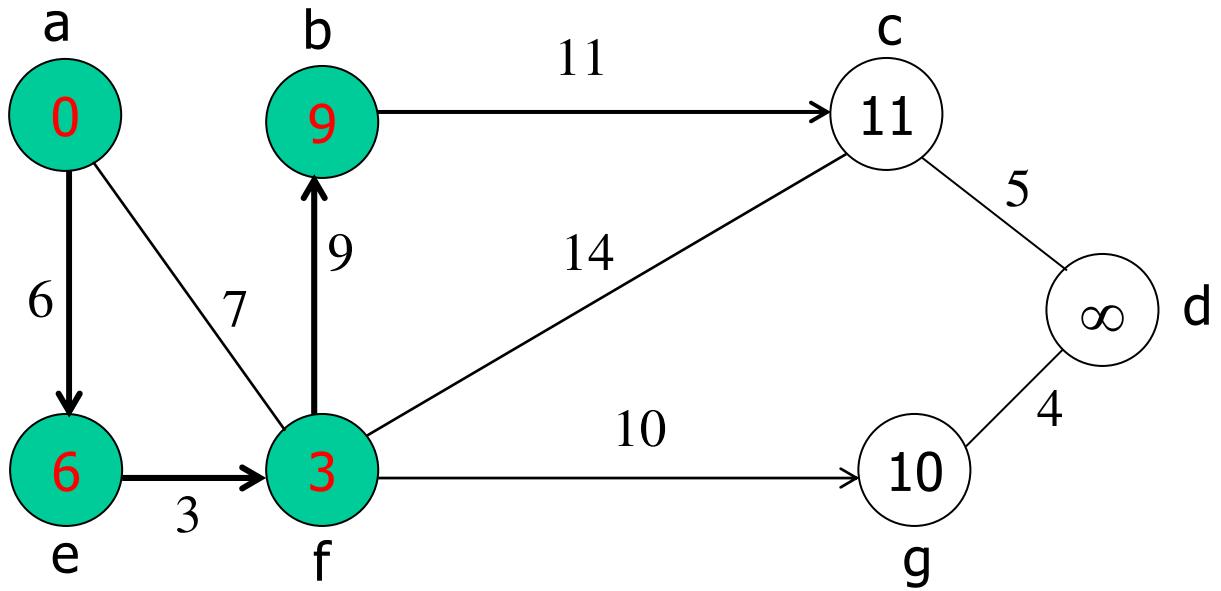
- Chọn f là đỉnh kế của MST, key[f] =3



Cập nhật key[b]=9, key[c]=14, key[g]=10

THUẬT GIẢI PRIM

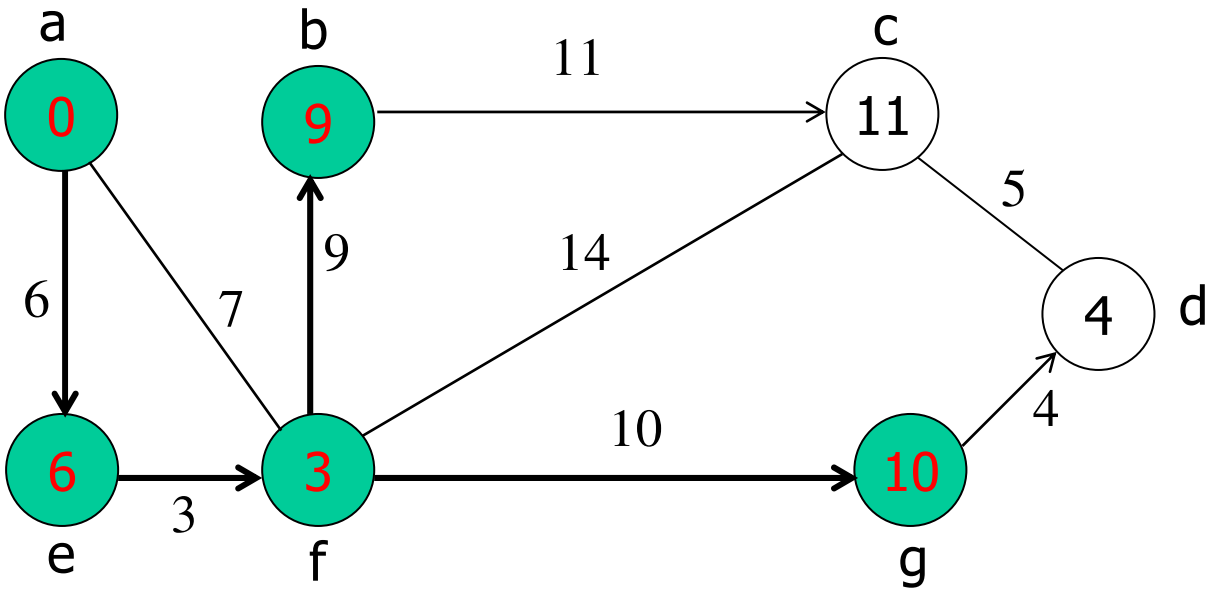
- Chọn b là đỉnh kế của MST, key[b] =9



Cập nhật key[c]=11

THUẬT GIẢI PRIM

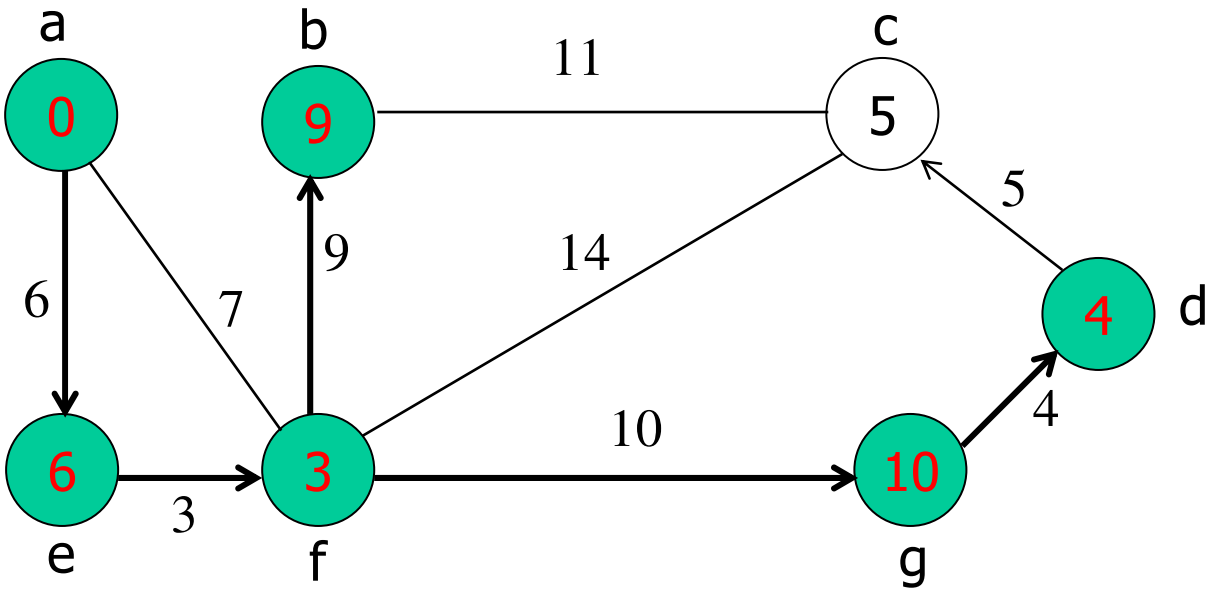
- Chọn g là đỉnh kế của MST, key[g] =10



Cập nhật key[d]=4

THUẬT GIẢI PRIM

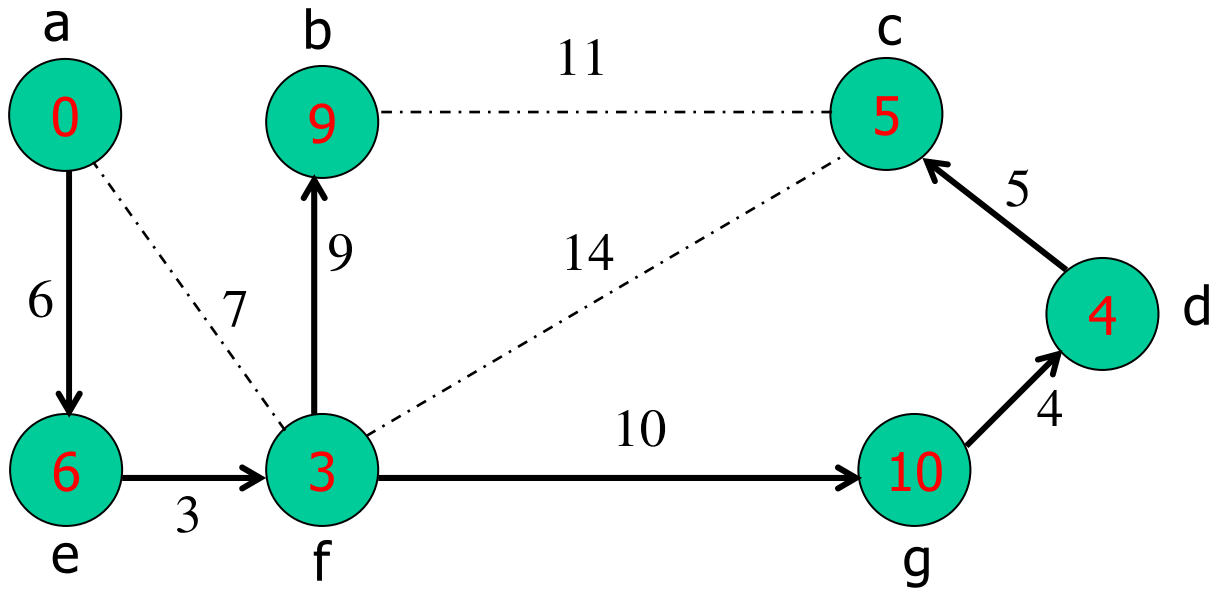
- Chọn d là đỉnh kế của MST, key[d] =4



Cập nhật key[c]=5

THUẬT GIẢI PRIM

- Chọn c là đỉnh kế của MST, key[c] =5, kết thúc thuật giải



THUẬT GIẢI PRIM

- Chi phí khởi tạo dòng 1-3 là O(V)
- Tổng thời gian cho tất cả các lần gọi EXTRACT-MIN trong vòng lặp **while** là O(V lg V)
- Tổng thời gian cho tất cả các lần lặp của vòng lặp **for** 8-11 là O(E lg V)
- Do đó, tổng chi phí là O(V lg V + E lg V) = O(E lg V)