INSERT(x, H) (Cont.) in min heap

Begin
  $i \leftarrow size(H)$;
  $H[size] \leftarrow x$;
  $size(H) \leftarrow size(H) + 1$;

  while $(i > 0$ and $H[i] < H[\lfloor (i-1)/2 \rfloor])$
    $swap(H[i], H[\lfloor (i-1)/2 \rfloor])$;
    $i \leftarrow \lfloor (i-1)/2 \rfloor$;
End

Complexity: $\mathcal{O}(\log n)$.

HEAPIFY(i, H)

Begin
  $n \leftarrow size(H) - 1$;
  Flag $\leftarrow$ true;

  while $(i \le \lfloor (n-1)/2 \rfloor$ and Flag $=$ true$)$
    min $\leftarrow i$;
    if $(H[i] > H[2i+1])$
      min $\leftarrow 2i+1$;

    if $(2i+2 \le n$ and $H[min] > H[2i+2])$
      min $\leftarrow 2i+2$;

    if (min $\neq i$)
      $swap(H[i], H[\text{min}])$;
      $i \leftarrow min$;
    else
      Flag $\leftarrow$ false;
End

Heapsort

- Build heap H on the given n elements.
- **While** (H is not empty)
  $x \leftarrow$ EXTRACT-MIN(H);
  print x;
- **Complexity:** $\mathcal{O}(n \log n)$.

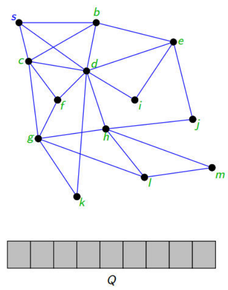
Algorithm: UNIVERSALSINK(M)

I/P: A $|V| \times |V|$ adjacency-matrix of a graph $G = (V, E)$.

Begin
  $i = 1$;
  $j = 1$;

  while $(i \le |V|$ and $j \le |V|)$ {
    if $(M[i,j] == 1)$
      $i = i + 1$;
    else
      $j = j + 1$;
  }
  if $(i > |V|)$
    print "there is no universal sink"
  else
    if (ISSINK(M, i) == False)
      print "there is no universal sink"
    else
      print "i is the universal sink"
End

BFS(G, s)



V_0: $\{s\}$
V_1: $\{b,\ c,\ d\}$
V_2: $\{e,\ g,\ f,\ k,\ h,\ i\}$
V_3: $\{j,\ l,\ m\}$

I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow$ WHITE;
    $d[u] \leftarrow \infty$;   // $d[u] = \delta(s, u)$
    $\pi[u] \leftarrow$ NIL; }   // $\pi[u]$ = predecessor of u

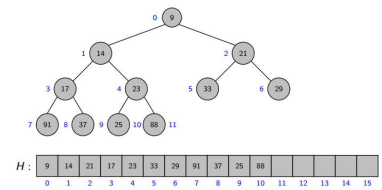
  $color[s] \leftarrow$ GRAY;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow$ NIL;
  $Q \leftarrow \emptyset$;   // Q denotes a queue
  ENQUEUE(Q, s);

  while $(Q \neq \emptyset)$ {
    $u \leftarrow$ DEQUEUE(Q);
    for each $v \in Adj[u]$ {
      if $(color[v] = white)$ {
        $color[v] \leftarrow$ GRAY;
        $d[v] \leftarrow d[u] + 1$;
        $\pi[v] \leftarrow u$;
        ENQUEUE(Q, v); } }
    $color[u] \leftarrow$ BLACK; }
End

EXTRACT-MIN(H)

Goal: Deletes the smallest key from H.

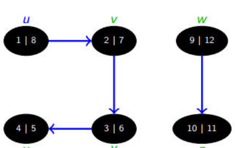
Challenge: Preserve the complete binary tree structure as well as the heap property!



- $swap(H[0], H[size-1])$.
- $size = size - 1$.
- **While** $x > key[left[x]]$ or $x > key[right[x]]$, **then**
  - $swap(x, \min\{left[x], right[x]\})$.
- **Complexity:** # swaps $= \mathcal{O}$(# levels in binary heap) $= \mathcal{O}(\log n)$ (show it!).

DFS(G)

Begin
  for each vertex $u \in V$
    $color[u] \leftarrow$ WHITE;
    $\pi[u] \leftarrow$ NIL;
  $time \leftarrow 0$;

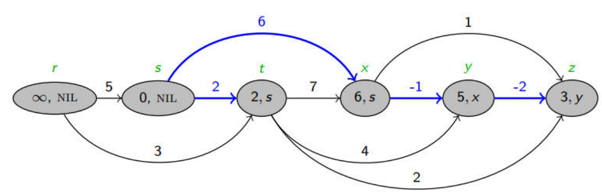


DFS Forest

DFS(G)
I/P: $G = (V, E)$ in adjacency-list representation.
Begin
  ...
  for each vertex $u \in V$
    if $(color[u] =$ WHITE$)$
      DFS-VISIT(u);
End

DFS-VISIT(u)
Begin
  $color[u] \leftarrow$ GRAY;   // u discovered.
  $time \leftarrow time + 1$;
  $d[u] \leftarrow time$;
  for each $v \in Adj[u]$ // Explore (u, v).
    if $(color[v] =$ WHITE$)$
      $\pi[v] \leftarrow u$;
      DFS-VISIT(v);
  $color[u] \leftarrow$ BLACK; // Blacken u (finished).
  $f[m] \leftarrow time \leftarrow time + 1$;
End

DAG-SHORTEST-PATHS(G, w, s) Topological Sort



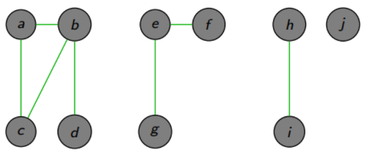
TOPOLOGICAL-SORT(G)
for each vertex $v \in V$
  $d[v] \leftarrow \infty$
  $\pi[v] \leftarrow$ NIL
$d[s] \leftarrow 0$
for each vertex $u \in V$, taken in topologically sorted order
  for each vertex $v \in Adj[u]$
    if $(d[v] > d[u] + w(u, v))$
      $d[v] \leftarrow d[u] + w(u, v)$
        $\pi[v] \leftarrow u$

Algorithm 1: IsBipartite.

Input: Undirected graph $g = (V, E)$.
Output: *True* if g is bipartite, and *False* otherwise.

foreach *Non-empty subset* $V_1 \subset V$ **do**
  $V_2 \leftarrow V \setminus V_1$;
  $bipartite \leftarrow True$;   // $bipartite = False$ if an edge's endpoints are both in the same set
  foreach *Edge* $\{u, v\} \in E$ **do**
    if $\{u, v\} \subseteq V_1$ *or* $\{u, v\} \subseteq V_2$ **then**
      $bipartite \leftarrow False$;   // $\langle V_1, V_2 \rangle$ is not a "good" bipartition
      Break;
  if $bipartite = True$ **then**
    return *True*;   // $\langle V_1, V_2 \rangle$ is a "good" bipartition; g is bipartite
return *False*;   // g is not bipartite

Connected Components: An Example



CONNECTED-COMPONENTS(G)
for each vertex $v \in V$
MAKE-SET(v);
for each edge $(u, v) \in E$
if (FIND-SET(u) $\neq$ FIND-SET(v))
UNION(u, v);

Edge processed	Collection of disjoint sets			
(b, c)	$\{a, b, c, d\}$	$\{e, f, g\}$	$\{h, i\}$	$\{j\}$

SAME-COMPONENT

SAME-COMPONENT(u, v)
if (FIND-SET(u) = FIND-SET(v))
return TRUE;
else
return FALSE;

Find SCC

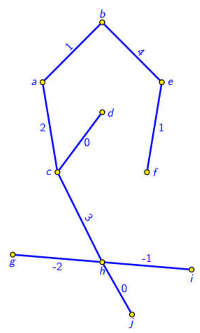
MAKE-SET, UNION and LINK

MAKE-SET(x)
$p[x] \leftarrow x$; // $p[x]$ denotes the parent of x
$rank[x] \leftarrow 0$;

UNION(x, y)
LINK(FIND-SET(x), FIND-SET(y));

LINK(x, y) // Takes pointers to the roots of x and y as inputs
if ($rank[x] > rank[y]$)
$p[y] \leftarrow x$;
else
$p[x] \leftarrow y$;
if ($rank[x] = rank[y]$)
$rank[y] = rank[y] + 1$;

KRUSKALALGORITHM(G)



I/P: A weighted undirected $G = (V, E, w)$.
O/P: An MST $T = (V, E')$ of G.
$\mathcal{L} \leftarrow E$
Sort $\mathcal{L}$ in ascending order.
(Grow a subgraph $T = (V, E')$ into a tree)
Initially $E' = \emptyset$
while $(|E'| < n - 1)$
Pick the next edge in $\mathcal{L}$.
if $T \cup \{e\}$ *creates a cycle*
Reject e.
else
$E' \leftarrow E' \cup \{e\}$.

$$w(G) = \sum_{e \in E'} w(e) = 8.$$

Question: How to check whether $T \cup \{e\}$ creates a cycle or not?

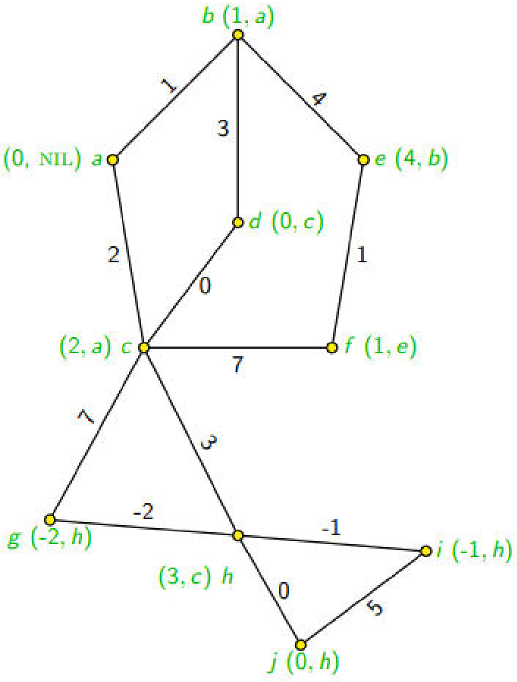
An Implementation Using Disjoint-set Data Structure

MST-KRUSKAL(G, w)
I/P: A connected, weighted undirected graph $G = (V, E)$ and the corresponding weight function w.
O/P: A list of edges of the MST.
$A \leftarrow \emptyset$
for each vertex $v \in V$
MAKE-SET(v)
sort the edges of E into non-decreasing order of weight w
for each edge $(u, v) \in E$, taken in non-decreasing order of weight
if (FIND-SET(u) $\neq$ FIND-SET(v))
$A \leftarrow A \cup \{(u, v)\}$
UNION(u, v)
return A

FIND-SET

FIND-SET(x)
if ($x \neq p[x]$)
$p[x] \leftarrow$ FIND-SET($p[x]$);
return $p[x]$;

MST-PRIM(G, w, r)



$$w(G) = \sum_{e \in A} w(e) = 8.$$

I/P: A weighted connected undirected graph $G = (V, E, w)$ and a root vertex r.

O/P: A MST $T = (V, A)$ of G.

```
for each u ∈ V
    key[u] ← ∞
    π[u] ← NIL
key[r] ← 0
Q ← V
while (Q ≠ ∅)
    u ← EXTRACT-MIN(Q)
    for each v ∈ Adj[u]
        if (v ∈ Q) and (w(u, v) < key[v])
            π[v] ← u
            key[v] ← w(u, v)
```

I/P: A weighted connected undirected graph $G = (V, E, w)$ and a root vertex r.

O/P: A MST $T = (V, A)$ of G.

```
   // Constructing the MST T
T ← ∅
for each v ∈ V                {r = root}
  if (v != r)
    T ← T ∪ {(π[v], v)}
return T
```

The solution of the recurrence relation $T(n) = aT(n/b) + cn^k$, where a and b are integer constants, $a \geq 1, b \geq 2$, and c and k are positive constants, is

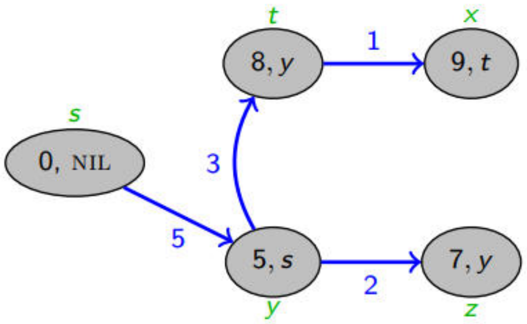
$$T(n) = \begin{cases} \mathcal{O}(n^{\log_b a}) & \text{if } a > b^k \\ \mathcal{O}(n^k \log n) & \text{if } a = b^k \\ \mathcal{O}(n^k) & \text{if } a < b^k \end{cases}$$

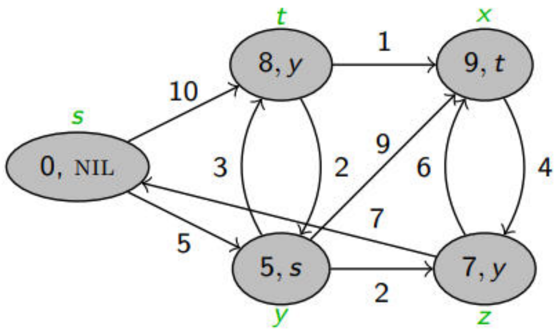
BFS

```
BFS_Cycle(G, s)

// G is a graph (V, E)
// s is the source vertex
// each vertex contains three
fields, color, d, π

// Output: return 1 if the part of
graph reachable via s has a cycle;
otherwise, return 0
```

```
BFS_cycle(G, s)

for each vertex u ∈ G.V - {s}
    u.color = WHITE
    u.π = NIL

s.color = GRAY
s.π = NIL
Q = ϕ
ENQUEUE(Q, s)
while Q ≠ ϕ
    u = DEQUEUE(Q)
    for each vertex v in G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.π = u
            ENQUEUE(Q, v)
        else if u.π != v
            return 1
    u.color = BLACK
return 0
```

DIJKSTRA(G, w, s)





I/P: A directed graph $G = (V, E)$, with a weight function $w : E \rightarrow \mathbb{R}_{\geq 0}$ and a source vertex s.

O/P: Shortest-path tree S with root vertex s.

Initialization Step:

```
for each v ∈ V
    d[v] ← ∞
    π[v] ← NIL
d[s] ← 0
S ← ∅
Q ← V   // Q(d): MIN-PRIORITY queue
```

Updation Step:

```
while (Q ≠ ∅)
    u ← EXTRACT-MIN(Q)
    S ← S ∪ {u}
    for each vertex v ∈ Adj[u]
        if (v ∈ Q) and (d[v] > d[u] + w(u, v))
            d[v] ← d[u] + w(u, v)
            π[v] ← u
```

I/P: A directed graph $G = (V, E)$, with a weight function $w : E \rightarrow \mathbb{R}_{\geq 0}$ and a source vertex s.

O/P: Shortest-path tree S with root vertex s.

...

Constructing the Shortest-path Tree S:

```
S ← ∅
for each v ∈ V
    if (v ≠ s)
        S ← S ∪ {(π[v], v)}
return S
```

HASH-INSERT(T, k)

I/P: A hash table T and a key k.

```
repeat j ← h(k, i)
    if (T[j] = NIL)
        T[j] ← k
        return j
    else
        i ← i + 1
until i = m    {m = Size of hash-table}
error "hash table overflow"
```

HASH-SEARCH(T, k)

I/P: A hash table T and a key k.

O/P: j if slot j is found to contain key k, else NIL.

```
i ← 0
repeat j ← h(k, i)
    if (T[j] = k)
        return j
    i ← i + 1
until T[j] = NIL or i = m
return NIL
```