Heap (Algorithms)

procedure used in heapsort :-

→ Max-Heapify → O(log n) → maintain max-heap property

→ Build-max-Heap → produce max head from unordered list.

→ Heapsort → O(n log n)

1. MAX-HEAPIFY(A, i)

```
l = Left(i)  ---> 2i+1   } (index in array)
r = Right(i) ---> 2i+2   }
if l ≤ A.heap-size and A[l] > A[i]
    largest = l
    else largest = i
if r ≤ A.heap-size and A[r] > A[largst]
        largest = r
if largest ≠ i                  → Index
        exchange A[i] with A[largest]
        MAX-HEAPIFY (A, largest)
```

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

Time complexity
O(log n)
↓
See the tree structure.

2. BUILD-MAX-HEAP (A) → A[]

```
A.heap-size = A.length (length of array)
for i from ⌊A.length/2⌋ to 1
        MAX-HEAPIFY (A, i)
```

(Important)

Important

3. HEAPSORT(A)

```
BUILD-MAX-HEAP (A)
for i = A.length downto 2
    Exchange A[1] with A[i]
    A.heap-size = A.heap-size - 1
    max-heapify (A, 1)
end for
```

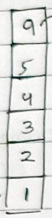
Anyone, anywhere, can make a positive difference. -Mark Sanborn

priority Queue implementation in Heap data structure.

↓

[The highest priority is always at the root of the head]

Let's see for Max heap



→ Element highest priority → Dequeue

priority queue operations:-

I. Inserting into priority Queue:-

MaxHeapInsert (A, element, priority):
(A, el, p)

A.append(element, priority)

i = A.length − 1 (length of element = position)

Now length = A.length.

BUILD-MAX-HEAP (A);

II. Remove (i)

III. extractMAX ()

IV. getMax ()

V. Changepriority (i, p)

A man who wants to lead the orchestra must turn his back on the crowd. -Max Lucado

(color representation)

Breadth-first search (Important)

pseudocode:-

BFS (G, s) [G → graph, s → queue → s is the starting element/node]

For each vertex u ∈ G.V − {s}

u.color = white

u.d = ∞

u.π = NIL (predecessor of u)

s.color = Gray

s.d = 0

s.π = NIL

Q = ∅

Enqueue (Q, s) [Q = queue]

While Q ≠ ∅

u = Dequeue (Q)

For each v ∈ G.Adjacence[u]

if v.color == white

v.color = Gray

v.d = u.d + 1

v.π = u

EnQueue (Q, v)

end for.

u.color = Black

end while.

Sunday 27

The strength of a nation derives from the integrity of the home. -Confucius

DFS pseudocode -1

```
DFS(G)
  For each vertex u ∈ G.V
      u.color = White
      u.π = NIL
  time = 0
  for each vertex u ∈ G.V
      if u.color == White
          DFS-VISIT(G,u)

DFS-VISIT(G,u)
  time = time+1          // White vertex u just been discovered.
  u.d = time
  u.color = Gray
  For each v ∈ G.adjacent[u]      // explore edge(u,v)
      if v.color == White
          v.π = u
          DFS-visit(G,v)
  u.color = BLACK        // Blacken u; it is done.
  time = time+1
  u.f = time.
```


Never let the fear of striking out keep you from playing the game. -Babe Ruth

Bipartite using BFS:-

Find set pseudocode:-

```
Find(x)
  if x.parent ≠ x
      x.parent := Find(x.parent)
      return x.parent
  else
      return x
end function.
```

makeset:-

```
Find(x)
  if x.parent = x
      return x
  else
      x.parent = Find(x.parent)
      return x.parent
end function
```

MST-Kruskal(G,w)

A = ϕ

Unionset pseudocode:-

```
function union(x,y)
    x := Find(x)
    y := Find(y)        } x and y are replaced by root.
  if x = y then
      return
  end if
  if x.size < y.size
      (x,y) := (y,x)
  end if
  y.parent := x
  // update the size of x
  x.size := x.size + y.size
end function
```

A happy family is but an earlier heaven. -George Bernard Shaw

MST_Kruskal(G,w)
A = ϕ
for each vertex v ∈ G.V
makeset(v)
Create a simple list of the edges G.E sord the edges (monotonically by w)
for each edge (u,v) form the sorted list.
if finset(u) ≠ finset(v)
A = A ∪ {(u,v)}
union(u,v)
return A.

09
10
11
12
01
02
03
04
05
06
07
08

The family is one of nature's masterpieces. George Santayana

DFS pseudocode-II (Important)

→ S& Initiative stack S with one element s
while S is not empty
09 Take a node u from s (pop)
if visited[u] = false
10 set visited[u] = ~~false~~ true
for each edge (u,v) from u
11 add v to S (push)
end for
12 end if
end while

→ DFS-VISIT(s)
↓
using stack

01 DFS(u)
set visited[u] = true
02 add u to tree T
for edge (u,v) from u
03 if visited[v] = false
~~DFS-VISIT~~ DFS(v)
04 end if
end for

05
06
07
08

The measure of intelligence is the ability to change.- Albert Einstein

Prim's algorithm:-

```
MST-prim's-alg (G, w, r)
for each vertex u ∈ G.V
    u.key = ∞
    u.π = Nil
r.key = 0
Q = ∅  // primary queue
for each vertex u ∈ G.V
    Insert (Q, u)
while Q ≠ ∅
    u = Extract-min(Q)
    for each vertex v in adjacence[u]
    if v ∈ Q and w(u,v) < v.key
        v.π = u
        v.key = w(u,v)
```

Dijkstra's Algorithm:-

~~DIJKSTRA (G,w,s)~~

Before that,

→ INITIALIZE-SINGLE-SOURCE (G,s)

1. for each vertex $v \in G.V$
2. $v.d = \infty$
3. $v.\pi = NIL$
4. $S.d = 0$

It is not that I'm so smart. But I stay with the questions much longer.- Albert Einstein

Dijkstra's Algorithm

```
DIJKSTRA (G, w, s)
for each vertex v ∈ G.V - {s}
    v.d = ∞
    v.π = NIL
end for
s.d = 0
S = ∅
Q = S.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    S = S ∪ {u}
    for each vertex v ∈ G.Adj[u]
        if v.d > u.d + w(u,v)
            v.d = u.d + w(u,v)      → RELAX(u,v,w)
            v.π = u
    end for
end while
```

Sunday 06

Start each day with a positive thought and a grateful heart.- Roy T. Bennett,

MARCH '22

S	M	T	W	T	F	S
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

APRIL '22

S	M	T	W	T	F	S
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

Ad-1

⇒ [Priority Queue = Q]

→ Max-Heapify (A, i)
l = left(i)
r = right(i)
largest = 0
if l ≤ A.heap-size and A[l] > A[i]
largest = l
else
largest = i
if r ≤ A.heap-size and A[r] > A[largest]
else
largest returned
if largest ≠ i
swap ~~largest~~ A[i] and A[largest]
max-Heapify (A, largest)

→ min-heapify (A, i)
l = left(i)
r = right(i)
minimum = 0
if l ≤ A.heap-size and A[l] < A[i]
Minimum = l
else minimum = i
if r ≤ A.heap-size and A[r] < A[minimum]
minimum = r
if minimum ≠ i
swap A[i] and A[minimum]
min-heapify (A, minimum)

What I'm looking for is not out there, it is in me.- Helen Keller

→ BUILD-MAX-HEAP (A, n)
A.heapsize = A.length
09 For i from ⌊A.length / 2⌋ to 1
10 MAX-Heapify (A, i)
end for
11 end function

→ BUILD-min-Heap (A, n)
A.heapsize = A.length
for i from ⌊A.length / 2⌋ to 1
min-Heapify (A, i)
end for
end function

→ Extract-min(A)
01 BUILD-min-Heap(A)
//A.heapsize = A.length
02 swap A[1] and A[A.length] and i = A.length
// Now, Remove the element A[A.length] from array
03 A.heapsize = A.length - 1 or Delete (A, i) → (deletes it's element)
BUILD-min-Heap (A)
04 end process.

05 Priority Queue using heap
Create Heap (H) → empty heap H
06 Insert (H, v) → Insert element v in H
Min-Heapify (H, i) → O(log n)
07 Find in (H) → (find the highest element or min key element)
Delete (H, i) → Delete ith element
08 [Extract-min] → To extract min key element.

Never bend your head. Hold it high. Look the world straight in the eye.- Helen Keller

BFS :- (using Queue)

BFS (G, V)
for all vertices $u \in G.V - \{r\}$
    $u.$colour = white or visited() = none
    $u.d = \infty$ → (distance)
    $u.\pi$ = nil → (predecessors)
$r.$colour = gray or $r.$visited() = True
$r.d = 0$
$r.\pi$ = nil
$Q = \phi$
enqueue (Q, r)
  while $Q \neq \phi$
      dequeue (u)
      for all vertices v adjacent [u]
          if $v.$colour = white or ~~v~~ [visited [v] = False]
              $v.$colour = gray
              $v.d = u.d + 1$
              $v.\pi = u$
              Enqueue (Q, v)
          end if
      end for
    $u.$colour = (black)
  end while
end process

DFS :- (using stack)

DFS (G, s)
Let Q be a stack and $S = \phi$
  Q.push(s)
  visited [s] = True
  while $s \neq \phi$
    $v = s.$top()
    s.pop()
  for all neighbour 'u' of vertice v
    if u is not visited
      s.push (u)
      visited [u] = True
  end for
end while

OR

DFS (G, s)
Let $S = \phi$ s.length = G.size
for all vertices $u \in G.V - \{s\}$
  visited [u] = false and $u.\pi$ = NIL
  time = 0
S.push(s)
while S is not equals to ϕ
  ~~while~~ if for $v \in G.$adjacent [S.top()]
    if visited [v] = false and v = left (s)
      S.push (v)
    if visited [v] = true and v = ~~right~~ left (s)
      S.push (S.~~left~~ right)
    if both left and right child visited
      S.pop()
  end for
end while
end process

Gravitation is not responsible for people falling in love.- Albert Einstein

Topological ordering →

```
Topo(G)
  find a node u with no incoming edges
  Orden it as first
  Delete u from G
    Topo(G - {u})
    append this after u
```

Bipartidness

```
Bipartite(G,s)
  Let Q be and empty Q
  Q = ϕ
  Suppose for all u ∈ G.V - {s}
    u.color = red
    u.p = 0
    u.π = nil
  s.color = blue
  enqueue(Q,s)
  While Q ≠ ϕ
      dequeue(Q,s) and s.color = blue
      for each v adjacent [s]
          if (v.color = red)
          q.enqueue(v)
      else if (v.color = s.color)
          return not bipartite
  return bipartite
```


The most pathetic person in the world is some one who has sight but no vision.-Helen Keller

Hullman coding

Message :- BCCABBDDAECCBBAEDDCC

length = 20

[ASCII - 8 bit] → we know that

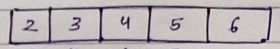
Character	count/frequency		code(3 bits)
A	3	3/20	000
B	5	5/20	001
C	6	6/20	010
D	4	4/20	011
E	2	2/20	100
	20	1	

here [total bits for msg] 20×3 = 60 bits

Use hullman coding :-

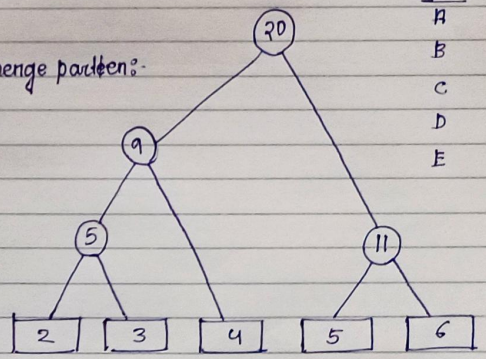
↳ optimal merge partten

here



Now, see.

optimum merge partten :-



Char	count	code
A	3	001
B	5	10
C	6	11
D	4	01
E	2	000

Sunday 13

Show me a family of readers, and I will show you the people who move the world.- Napoléon Bonaparte