线性查找

Linear Search (3)

```
i = 0
while i < n do
begin
   if X == a[i] then
        report "Found!" and stop
   else
        i = i+1
end
report "Not Found!"
```

二分查找

Binary Search (4)

```
first=0, last=n-1
while (first <= last) do
begin
     mid = ⌊(first+last)/2⌋
     if (X == a[mid])
         report "Found!" & stop
     else
         if (X < a[mid])
             last = mid-1
         else
             first = mid+1
end
report "Not Found!"
```

⌊ ⌋ is the floor function, truncate the decimal part

选择排序：

Selectio

```
for i = 0 to n-2 do
begin
  min = i
  for j = i+1 to n-1 do
    if a[j] < a[min] then
      min = j
  swap a[i] and a[min]
end
```

冒泡排序：

Bubble Sort Algorithm

```
for i = 0 to n-2 do
  for j = n-1 downto i+1 do
    if (a[j] < a[j-1])
      swap a[j] & a[j-1]
```

the smallest will be moved to a[i]

start from a[n-1],
check up to a[i+1]

插入排序：

Insertion Sort Algorithm

```
for i = 1 to n-1 do
begin
  key = a[i]
  pos = 0
  while (a[pos] < key) && (pos < i) do
    pos = pos + 1
  shift a[pos], …, a[i-1] to the right
  a[pos] = key
end
```

using linear search to find the correct position for key

finally, place key (the original a[i]) in a[pos]

i.e., move a[i-1] to a[i], a[i-2] to a[i-1], …, a[pos] to a[pos+1]

二分查找（递归版）：

Recursive Binary Search

```
RecurBinarySearch(A, first, last, X)
begin
  if (first > last) then
     return false
  mid = ⌊(first + last)/2⌋
  if (X == A[mid]) then
     return true
  if (X < A[mid]) then
     return RecurBinarySearch(A, first, mid-1, X)
  else
     return RecurBinarySearch(A, mid+1, last, X)
end
```

invoke by calling RecurBinarySearch(A, 0, n-1, X) return true if X is found, false otherwise

归并排序：

```
Algorithm Mergesort(A[0..n-1])
  if n > 1 then begin
    copy A[0..⌊n/2⌋-1] to B[0..⌊n/2⌋-1]
    copy A[⌊n/2⌋..n-1] to C[0..⌈n/2⌉-1]
    Mergesort(B[0..⌊n/2⌋-1])
    Mergesort(C[0..⌈n/2⌉-1])
    Merge(B, C, A)
  end
```

```
Algorithm Merge(B[0..p-1], C[0..q-1], A[0..p+q-1])
    Set i=0, j=0, k=0
    while i<p and j<q do
    begin
        if B[i]≤C[j] then set A[k]=B[i] and increase i
        else set A[k] = C[j] and increase j
        k = k+1
    end
    if i==p then copy C[j..q-1] to A[k..p+q-1]
    else copy B[i..p-1] to A[k..p+q-1]
```

37

DFS:

DFS – pseudo code (recursive)

```
Algorithm DFS(G)          //G=(V,E)
  for each v in V
      mark v with 0       //means v is not visited yet
  count = 0
  for each vertex in V do
      if v is marked with 0
          dfs(v)
```

```
dfs(v)
count = count +1
Mark v with count
  for each vertex w in Adj(v)
   do
    if w is marked with 0
       dfs(w)
```

61

(Graph)

Bfs:

BFS – Pseudo Code (with data structure)

```
for each vertex u in V[G]-{s}
   do color[u] = white
Q=empty                          //Q is a queue
enqueue(Q, s)
while Q is not empty
   do u = dequeue(Q)
        for each v in Adj(u)     //adjacency list of u
            do if color[v]=white then
                 color[v]=gray
                 enqueue(Q, v)
         color[u]=black
```


Prim 算法：

Pseudo code

// Given a weighted connected graph G=(V,E)

pick a vertex v_0 in V

$V_T = \{ v_0 \}$

$E_T = \varnothing$

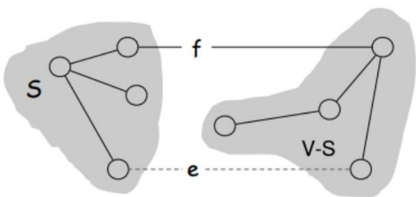
For i=1 to |V|-1 do

pick an edge e =(v*, u*) with minimum weight among all the edges (v, u) such that v is in V_T and u is in $V-V_T$

$V_T = V_T \cup \{ u^* \}$

$E_T = E_T \cup \{ e^* \}$

Return E_T



Kruskal 算法：

Pseudo code

// Given an undirected connected graph $G=(V,E)$

pick an edge e in E with minimum weight

$T = \{ e \}$ and $E' = E - \{ e \}$

while $E' \neq \varnothing$ **do**

begin

    pick an edge e in E' with minimum weight

    if adding e to T does not form cycle **then**

        $T = T \cup \{ e \}$

    $E' = E' - \{ e \}$

end

Time complexity?

O(nm)

Can be tested by marking vertices

31

Dijkstra's 算法

Pseudo code

// Given a graph $G=(V,E)$ and a source vertex s

for every vertex v in the graph **do**

    set $d(v) = \infty$ and $p(v) =$ null

set $d(s) = 0$ and $V_T = \varnothing$

this should be $\varnothing$

while $V - V_T \neq \varnothing$ **do**   // there is still some vertex left

begin

    choose the vertex u in $V - V_T$ with minimum $d(u)$

    set $V_T = V_T \cup \{ u \}$

    for every vertex v in $V - V_T$ that is a neighbour of u **do**

        if $d(u) + w(u,v) < d(v)$ **then**   *// a shorter path is found*

            set $d(v) = d(u) + w(u,v)$ and $p(v) = u$

end

https://www.youtube.com/watch?v=EFg3u_E6eHU&ab_channel=SpanningTree

Example: Question 3 in Week6 Tutorial

20

流水线调度：

Pseudo code

Time complexity is O(n)

set $f_1[1] = a_{1,1}$

set $f_2[1] = a_{2,1}$

for j = 2 to n **do**

begin

  set $f_1[j] = \min(\, f_1[j-1]+a_{1,j} \;,\; f_2[j-1]+t_{2,j-1}+a_{1,j} \,)$

  set $f_2[j] = \min(\, f_2[j-1]+a_{2,j} \;,\; f_1[j-1]+t_{1,j-1}+a_{2,j} \,)$

end

set $f^* = \min(f_1[n] \,,\, f_2[n])$

Floyd 算法：

Floyd's Algorithm (pseudocode)

```
let V = number of vertices in graph
let dist = V × V array of minimum distances initialized to ∞
for each vertex v
  dist [v][v] ← 0
for each edge (u,v)
  dist [u][v] ← weight(u,v)
for k from 1 to V
  for i from 1 to V
    for j from 1 to V
      if dist [i][j] > dist [i][k] + dist [k][j]
        dist [i][j] ← dist [i][k] + dist [k][j]
      end if
```