Data structures

Stacks

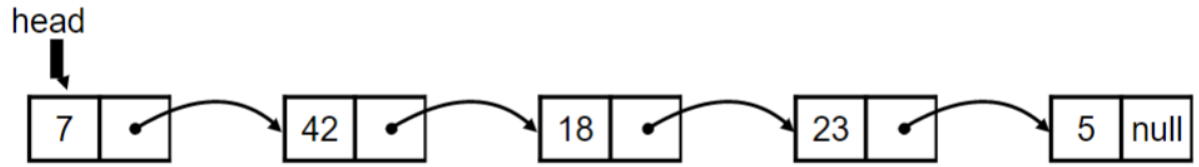
- PUSH(x): tilføj et nyt element x til S.
- POP(): fjern og returner det seneste tilføjede element i S.
- ISEMPTY(): returner sand hvis S ikke indeholder nogle elementer.

Queues

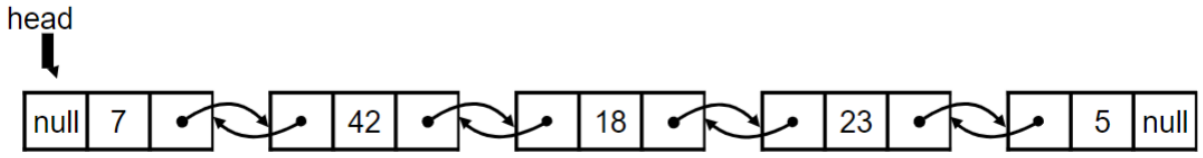
- ENQUEUE(x): tilføj et nyt element x til K
- DEQUEUE(): fjern og returner det tidligst tilføjede element i K.
- ISEMPTY(): returner sand hvis K ikke indeholder nogle elementer.

Lists

Single linked list



Double linked list

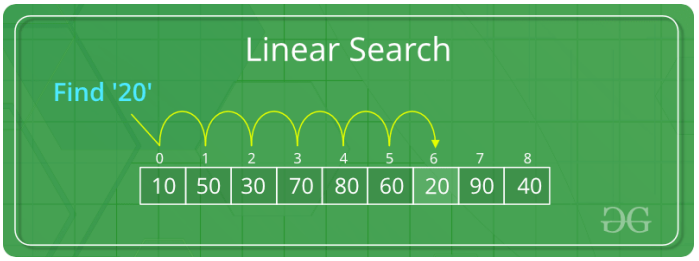


Searching

Linear search

```
LIST-SEARCH(L, k)
1  x = L.head
2  while x ≠ NIL and x.key ≠ k
3      x = x.next
4  return x
```

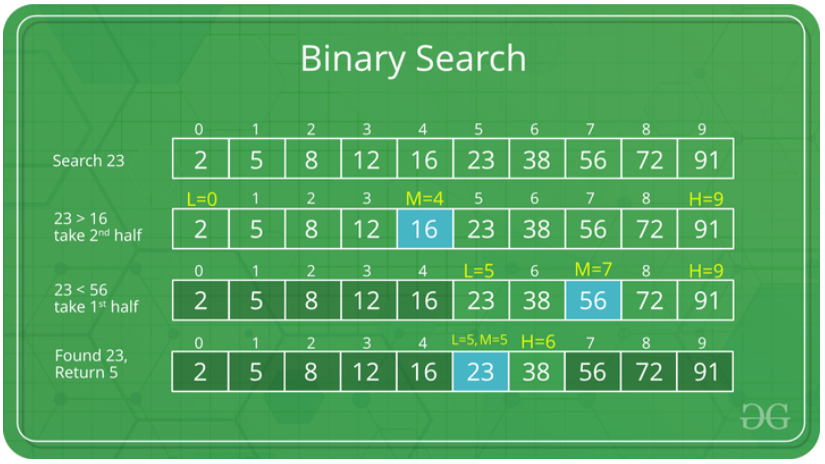
Time complexity: O(n)



Binary search

```
BINARY-SEARCH ( A, lo, hi, x )
    if ( lo > hi )
        return -1
    else
        mid := ⌊(lo + hi) / 2⌋
        if ( A[mid] == x )
            return mid
        else if ( A[mid] < x )
            return BINARY-SEARCH ( A, mid+1, hi, x)
        else // A[mid] > x
            return BINARY-SEARCH (A, lo, mid-1, x)
```

Time complexity: O(log n)



Sorting

Running Time

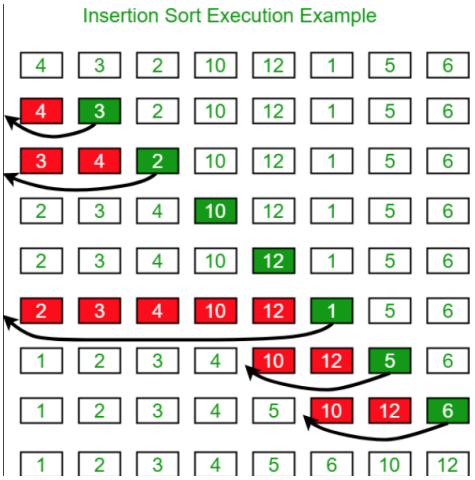
Algorithm	Worst-case running time	Average-case/expected running time
Insertion sort	$\Theta(n^2)$	$\Theta(n^2)$
Merge sort	$\Theta(n \lg n)$	$\Theta(n \lg n)$
Heapsort	$O(n \lg n)$	—
Quicksort	$\Theta(n^2)$	$\Theta(n \lg n)$ (expected)
Counting sort	$\Theta(k + n)$	$\Theta(k + n)$
Radix sort	$\Theta(d(n + k))$	$\Theta(d(n + k))$
Bucket sort	$\Theta(n^2)$	$\Theta(n)$ (average-case)

Array Sorting Algorithms

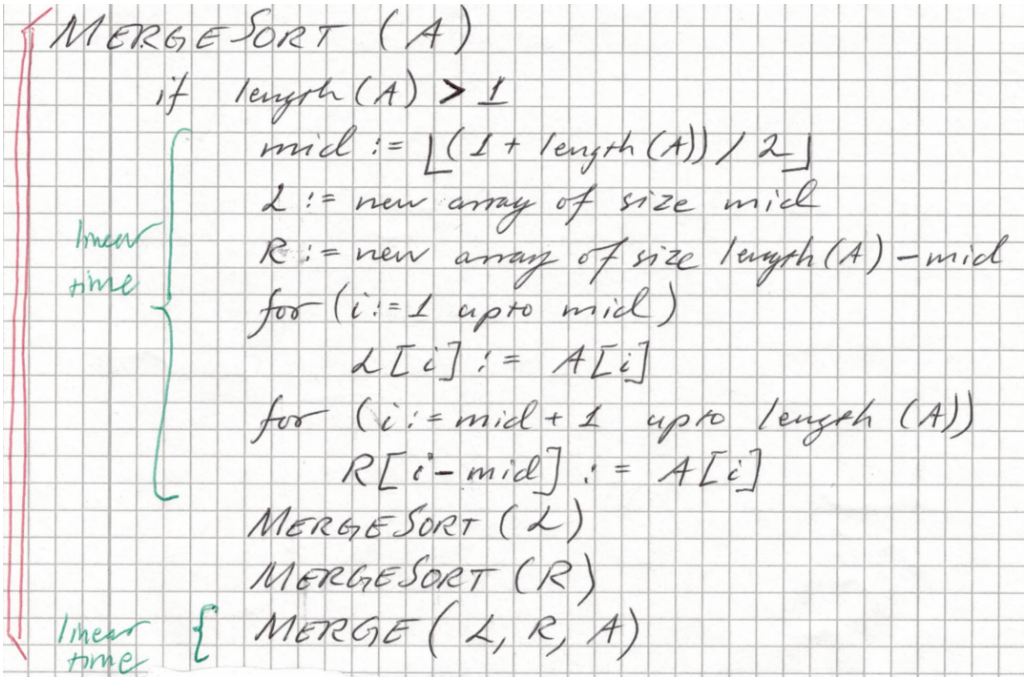
Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	θ(n log(n))	O(n log(n))	O(n)

Insertion sort

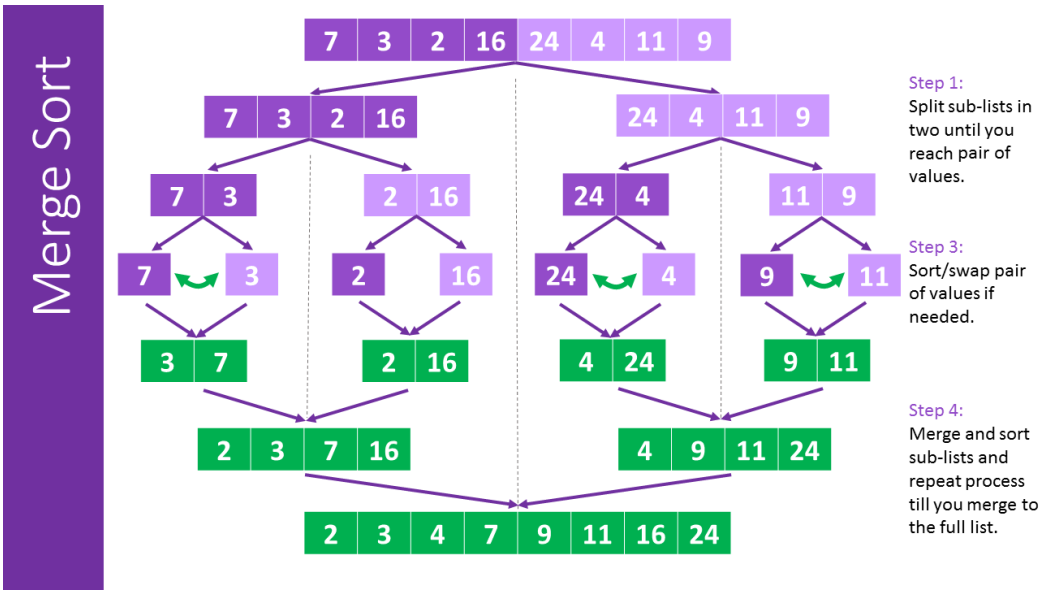
```
INSERTION-SORT(A)
1  for j = 2 to A.length
2      key = A[j]
3      // Insert A[j] into the sorted sequence A[1 .. j − 1].
4      i = j − 1
5      while i > 0 and A[i] > key
6          A[i + 1] = A[i]
7          i = i − 1
8      A[i + 1] = key
```



Merge sort

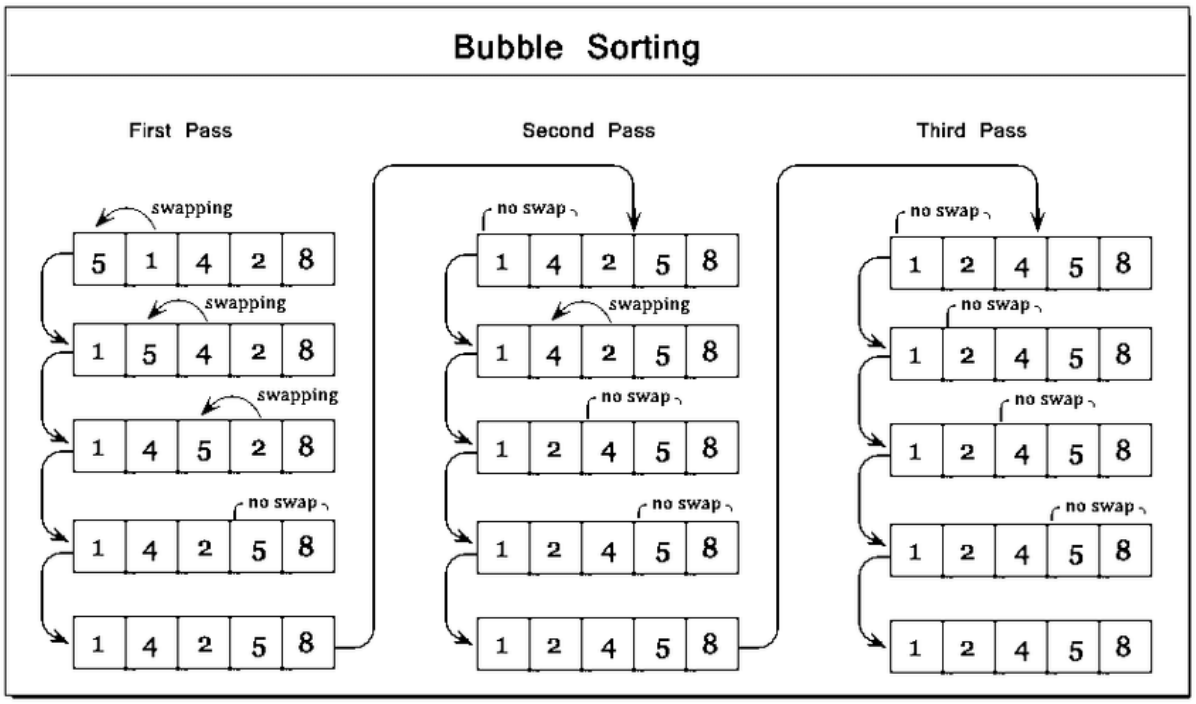


```
MERGE(A, p, q, r)
 1  n1 = q − p + 1
 2  n2 = r − q
 3  let L[1..n1 + 1] and R[1..n2 + 1] be new arrays
 4  for i = 1 to n1
 5      L[i] = A[p + i − 1]
 6  for j = 1 to n2
 7      R[j] = A[q + j]
 8  L[n1 + 1] = ∞
 9  R[n2 + 1] = ∞
10  i = 1
11  j = 1
12  for k = p to r
13      if L[i] ≤ R[j]
14          A[k] = L[i]
15          i = i + 1
16      else A[k] = R[j]
17          j = j + 1
```



Bubble sort

```
BUBBLESORT(A)
1  for i = 1 to A.length − 1
2      for j = A.length downto i + 1
3          if A[j] < A[j − 1]
4              exchange A[j] with A[j − 1]
```

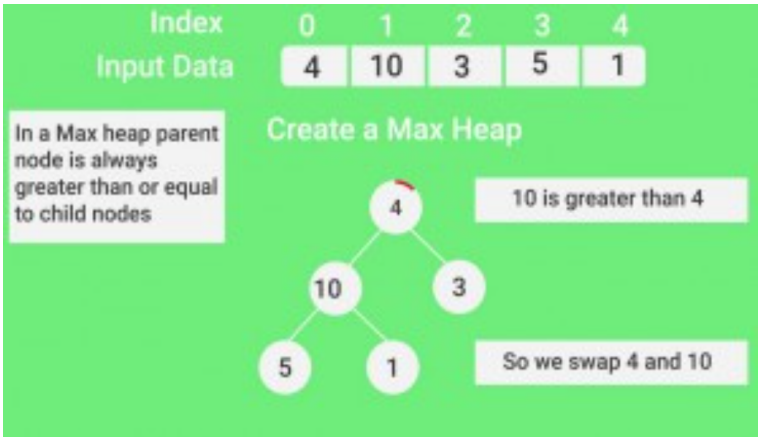


Heap sort

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

```
BUILD-MAX-HEAP(A)
1  A.heap-size = A.length
2  for i = ⌊A.length/2⌋ downto 1
3      MAX-HEAPIFY(A, i)
```

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



Divisors

Euclidean for GCD (Greatest common divisor)

```
function gcd(a, b)
    while b ≠ 0
        t := b
        b := a mod b
        a := t
    return a
```

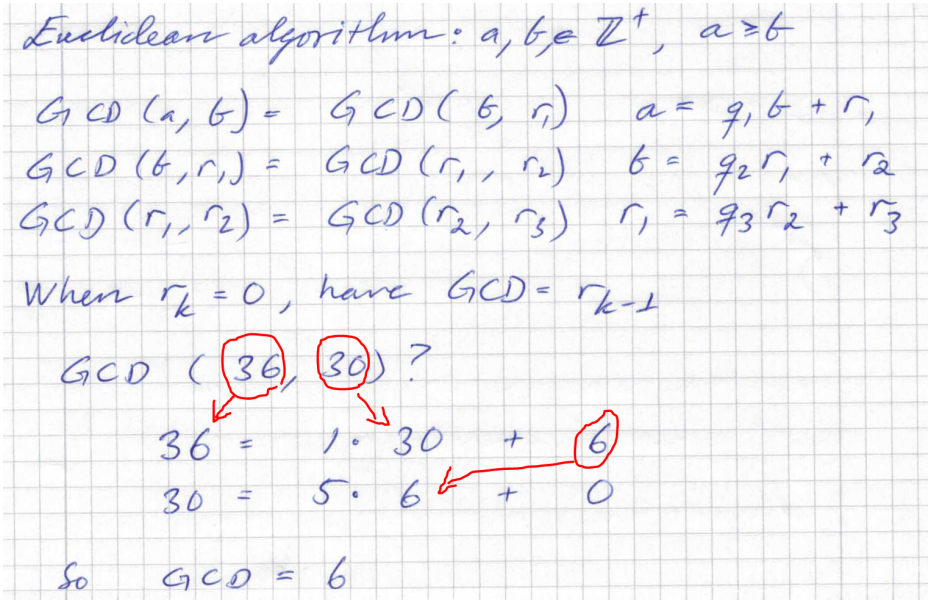
Example:

$GCD(36,30)$:

1. $30 = 1 \cdot 30 + 6$
   a. $30 mod 6 = 5$
2. $30 = 5 \cdot 6 + 0$
3. $GCD(36,30) = 6$

$GCD(90,28)$:

1. $90 = 3 \cdot 28 + 6$
2. $28 = 4 \cdot 6 + 4$
3. $6 = 1 \cdot 4 + 2$
4. $4 = 2 \cdot 2 = 0$
5. $GCD(90,28) = 2$



LCM (Least common multiple)

$$LCM(a,b) = \frac{a \cdot b}{GCD(a,b)}$$

Example:

$$LCM(8,10) = \frac{8 \cdot 10}{GCD(8,10)} = \frac{80}{2} = 40$$

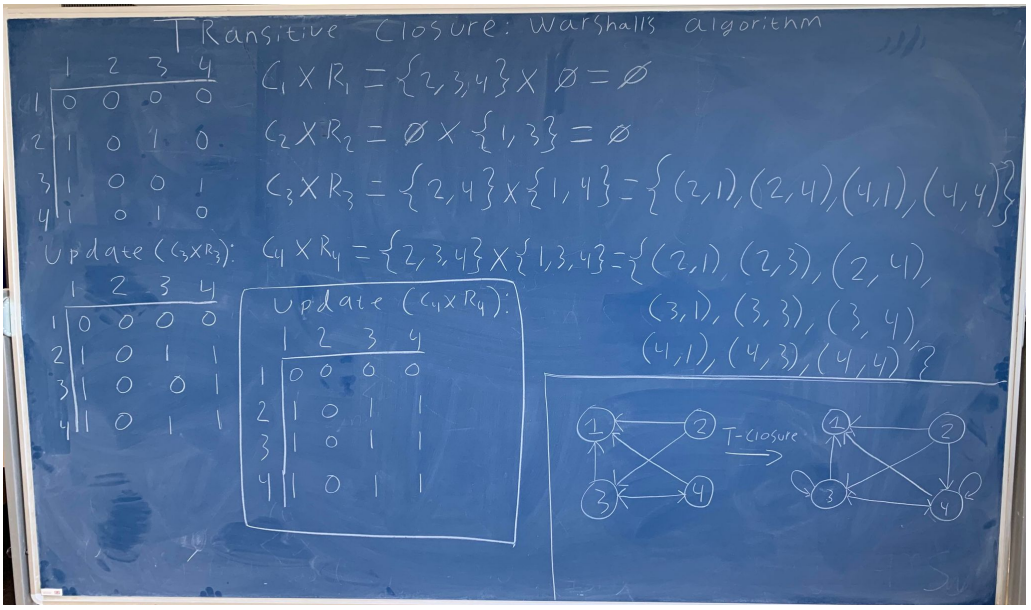
Relations

Warshall's (transitive closure)

```
FLOYD-WARSHALL(W)
1  n = W.rows
2  D^(0) = W
3  for k = 1 to n
4      let D^(k) = (d_ij^(k)) be a new n × n matrix
5      for i = 1 to n
6          for j = 1 to n
7              d_ij^(k) = min(d_ij^(k−1), d_ik^(k−1) + d_kj^(k−1))
8  return D^(n)
```

Transitive closure

```
TRANSITIVE-CLOSURE(G)
 1  n = |G.V|
 2  let T^(0) = (t_ij^(0)) be a new n × n matrix
 3  for i = 1 to n
 4      for j = 1 to n
 5          if i == j or (i, j) ∈ G.E
 6              t_ij^(0) = 1
 7          else t_ij^(0) = 0
 8  for k = 1 to n
 9      let T^(k) = (t_ij^(k)) be a new n × n matrix
10      for i = 1 to n
11          for j = 1 to n
12              t_ij^(k) = t_ij^(k−1) ∨ (t_ik^(k−1) ∧ t_kj^(k−1))
13  return T^(n)
```



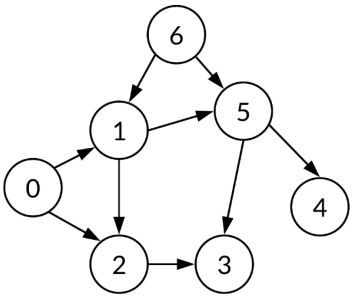
Graphs

Topological sort (DAG)

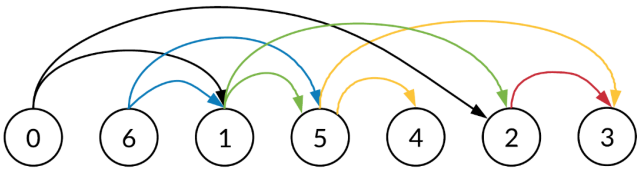
TOPOLOGICAL-SORT(G)

1 call DFS(G) to compute finishing times $v.f$ for each vertex v
2 as each vertex is finished, insert it onto the front of a linked list
3 **return** the linked list of vertices

Unsorted graph



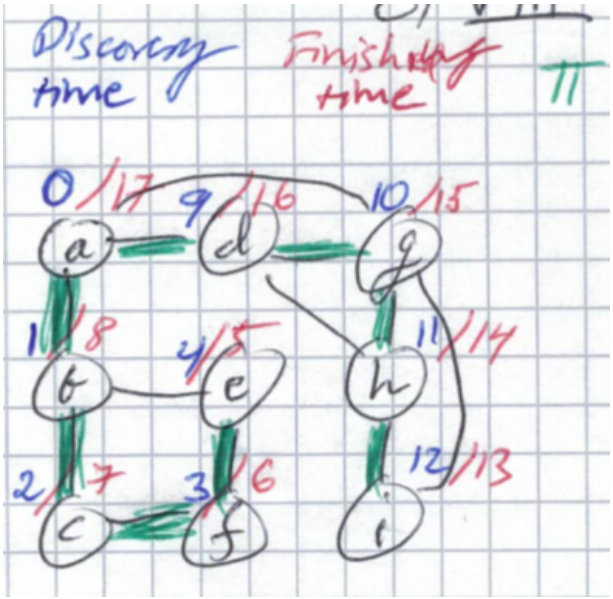
Topologically sorted graph



DFS

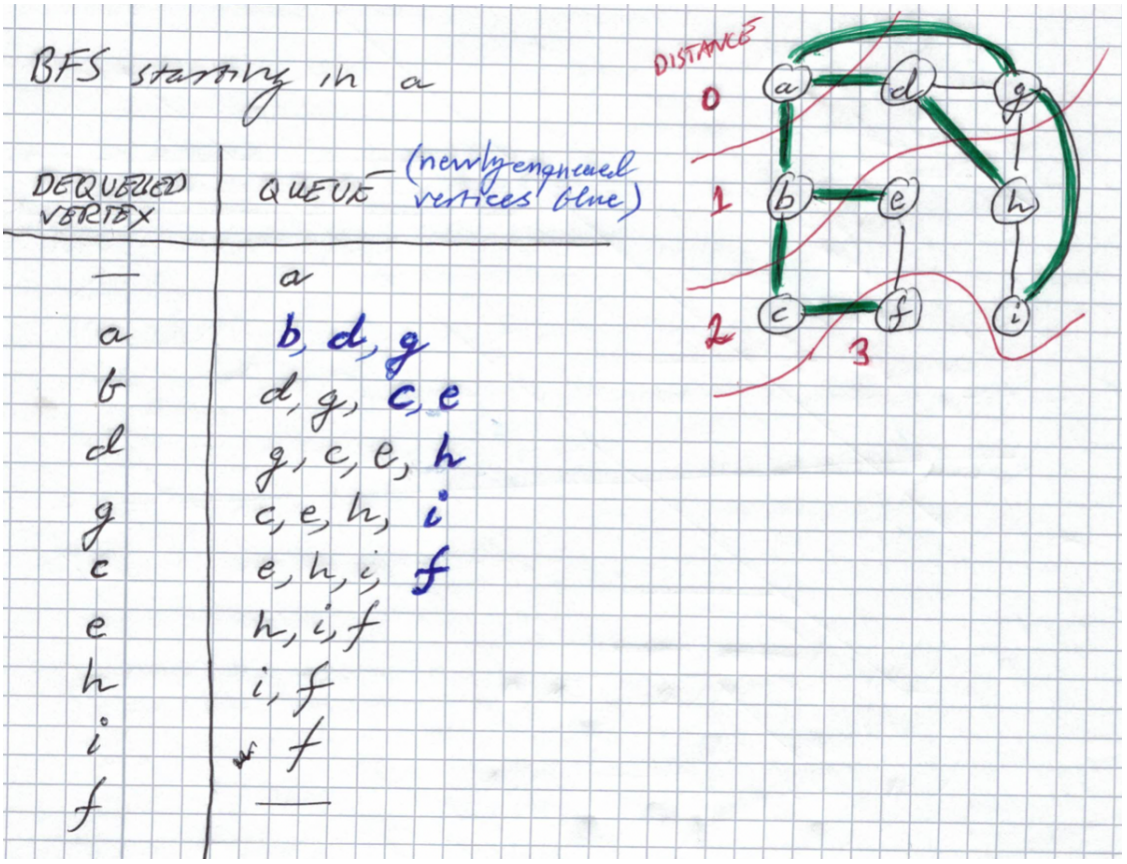
```
DFS(G)
1  for each vertex u ∈ G.V
2      u.color = WHITE
3      u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6      if u.color == WHITE
7          DFS-VISIT(G, u)
```

```
DFS-VISIT(G, u)
 1  time = time + 1            // white vertex u has just been discovered
 2  u.d = time
 3  u.color = GRAY
 4  for each v ∈ G.Adj[u]      // explore edge (u, v)
 5      if v.color == WHITE
 6          v.π = u
 7          DFS-VISIT(G, v)
 8  u.color = BLACK            // blacken u; it is finished
 9  time = time + 1
10  u.f = time
```



BFS

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



SCC (Strongly connected components)

STRONGLY-CONNECTED-COMPONENTS(G)

1 call DFS(G) to compute finishing times $u.f$ for each vertex u
2 compute G^T
3 call DFS(G^T), but in the main loop of DFS, consider the vertices in order of decreasing $u.f$ (as computed in line 1)
4 output the vertices of each tree in the depth-first forest formed in line 3 as a separate strongly connected component

Example:

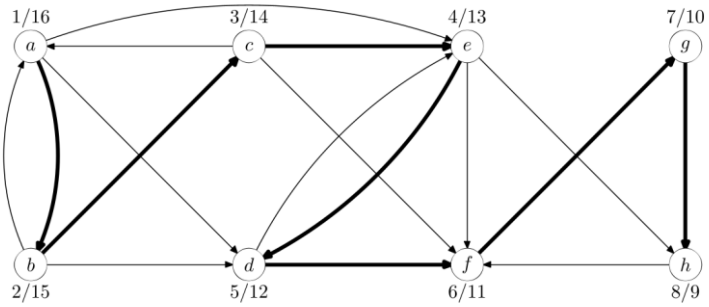


Figure 3: Directed graph G with depth-first search tree and start and finishing times.

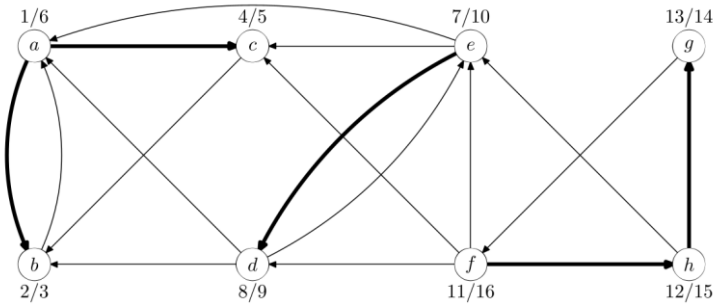


Figure 4: Directed graph G^T with DFS forest identifying strongly connected components.

Kruskal's (Minimum spanning tree/lowest weighted edge)

Find the smallest edge in entire graph and add it to the tree if it will not make a cycle

KRUSKAL(V, E, w)

1. A ← ∅
2. **for** each vertex v ∈ V
3. **do** MAKE-SET(v)

} O(V)

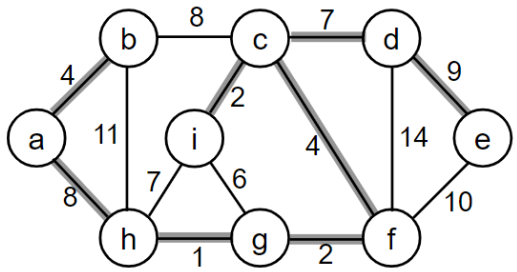
4. sort E into non-decreasing order by w } O(ElgE)
5. **for** each (u, v) taken from the sorted list ← O(E)
6. **do if** FIND-SET(u) ≠ FIND-SET(v)
7. **then** A ← A ∪ {(u, v)}
8. UNION(u, v)

} ← O(lgV)

9. **return** A

Running time: O(V+ElgE+ElgV)=O(ElgE) – dependent on the implementation of the disjoint-set data structure

Example



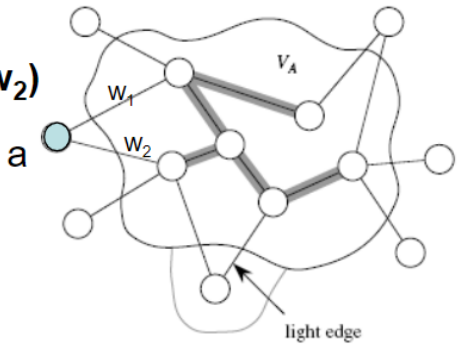
1: (h, g)
2: (c, i), (g, f)
4: (a, b), (c, f)
6: (i, g)
7: (c, d), (i, h)
8: (a, h), (b, c)
9: (d, e)
10: (e, f)
11: (b, h)
14: (d, f)

{a}, {b}, {c}, {d}, {e}, {f}, {g}, {h}, {i}

1.	Add (h, g)	{g, h}, {a}, {b}, {c}, {d}, {e}, {f}, {i}
2.	Add (c, i)	{g, h}, {c, i}, {a}, {b}, {d}, {e}, {f}
3.	Add (g, f)	{g, h, f}, {c, i}, {a}, {b}, {d}, {e}
4.	Add (a, b)	{g, h, f}, {c, i}, {a, b}, {d}, {e}
5.	Add (c, f)	{g, h, f, c, i}, {a, b}, {d}, {e}
6.	Ignore (i, g)	{g, h, f, c, i}, {a, b}, {d}, {e}
7.	Add (c, d)	{g, h, f, c, i, d}, {a, b}, {e}
8.	Ignore (i, h)	{g, h, f, c, i, d}, {a, b}, {e}
9.	Add (a, h)	{g, h, f, c, i, d, a, b}, {e}
10.	Ignore (b, c)	{g, h, f, c, i, d, a, b}, {e}
11.	Add (d, e)	{g, h, f, c, i, d, a, b, e}
12.	Ignore (e, f)	{g, h, f, c, i, d, a, b, e}
13.	Ignore (b, h)	{g, h, f, c, i, d, a, b, e}
14.	Ignore (d, f)	{g, h, f, c, i, d, a, b, e}

Prim (MSP)

Key[a]=min(w_1,w_2)



PRIM(V, E, w, r)

```
1.  Q ← ∅
2.  for each u ∈ V
3.      do key[u] ← ∞
4.         π[u] ← NIL
5.         INSERT(Q, u)
6.  DECREASE-KEY(Q, r, 0)        ▶ key[r] ← 0
7.  while Q ≠ ∅
8.        do u ← EXTRACT-MIN(Q)
9.           for each v ∈ Adj[u]
10.             do if v ∈ Q and w(u, v) < key[v]
11.                   then π[v] ← u
12.                        DECREASE-KEY(Q, v, w(u, v))
```

Total time: $O(V \lg V + E \lg V) = O(E \lg V)$

$O(V)$ if Q is implemented as a min-heap

$O(\lg V)$

Executed |V| times

Takes $O(\lg V)$

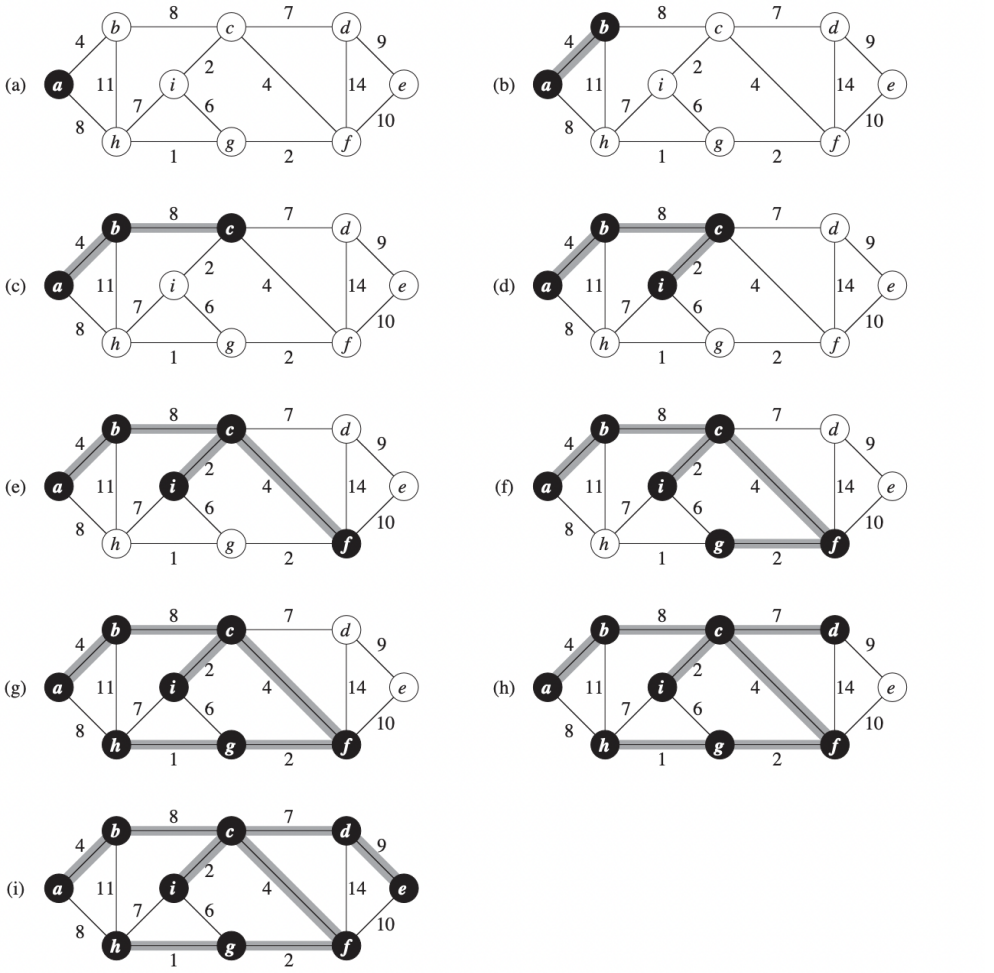
Min-heap operations: $O(V \lg V)$

Executed O(E) times total

Constant

Takes $O(\lg V)$

$O(E \lg V)$

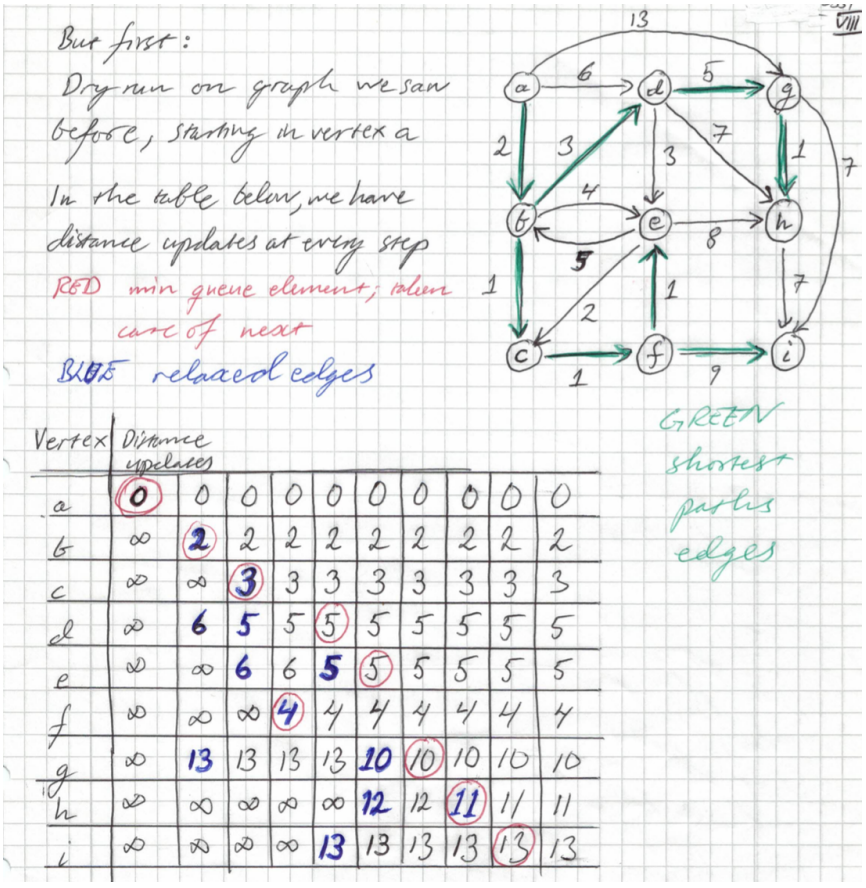


Dijkstra

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S = ∅
3  Q = G.V
4  while Q ≠ ∅
5      u = EXTRACT-MIN(Q)
6      S = S ∪ {u}
7      for each vertex v ∈ G.Adj[u]
8          RELAX(u, v, w)
```

```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      v.π = u
```

Example:

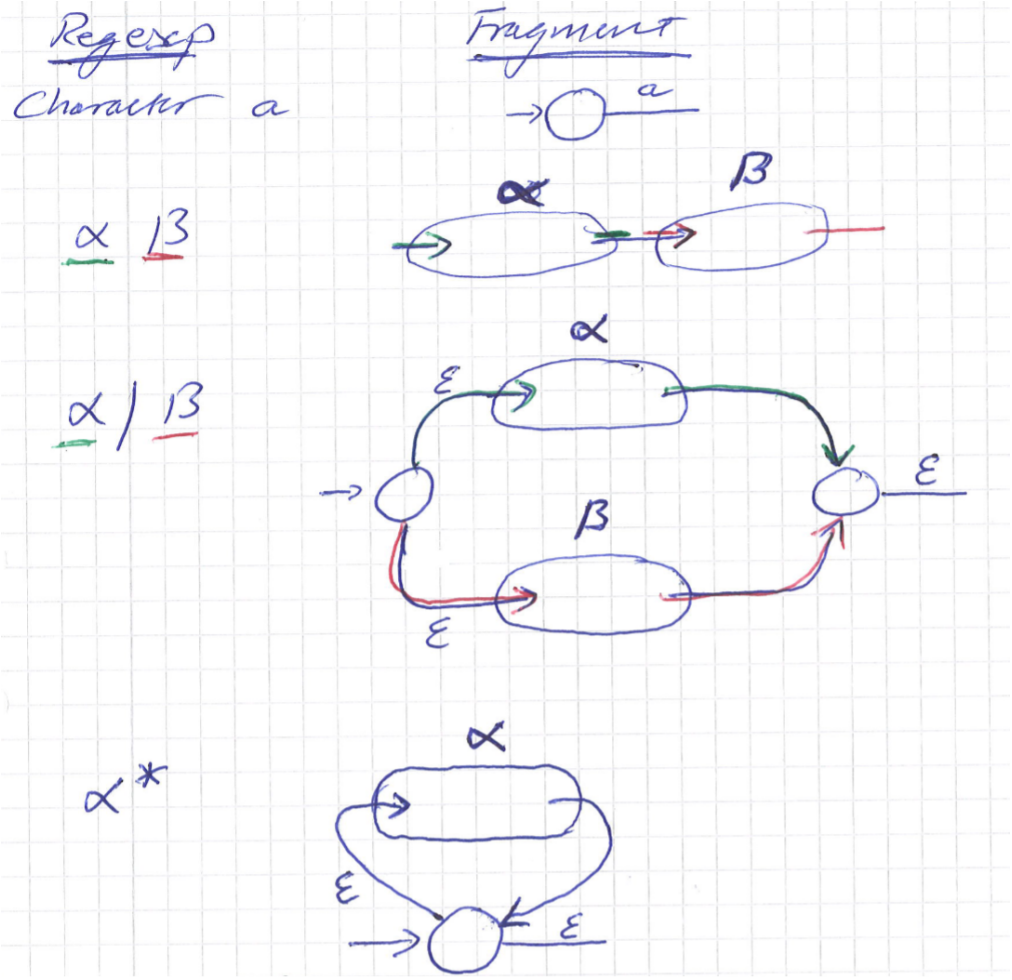


Vertex	Distance updates									
a	0	0	0	0	0	0	0	0	0	0
b	∞	2	2	2	2	2	2	2	2	2
c	∞	∞	3	3	3	3	3	3	3	3
d	∞	6	5	5	5	5	5	5	5	5
e	∞	∞	6	6	5	5	5	5	5	5
f	∞	∞	∞	4	4	4	4	4	4	4
g	∞	13	13	13	13	10	10	10	10	10
h	∞	∞	∞	∞	∞	12	12	11	11	11
i	∞	∞	∞	∞	13	13	13	13	13	13

Syntax/grammar/lexer

Regular expression to NFA

REGEXP	Derives
ϵ	Empty string
α	α
$\alpha\|\beta$	α or β
$\alpha \cdot \beta$	*a then b*
α^*	ϵ *or* $\alpha\alpha^*$



NFA to DFA

Solution: We follow Algorithm 1.3 in Section 1.5.2 of Mogesen's notes, except that we use calligraphic letters $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, ... to refer to the constructed states in the deterministic finite automaton (to distinguish them more clearly from the NFA states). Referring in what follows to the numbered states in the NFA in Figure 3, the starting state is

$$\varepsilon\text{-closure}(\{1\}) = \{1, 2, 3, 6, 10\} = \mathcal{A}$$

(where we recall that the *ε-closure* of a set of states S is any state that can be reached from some state in S via zero or more ε-transitions). The possible transitions from $\mathcal{A}$ on characters

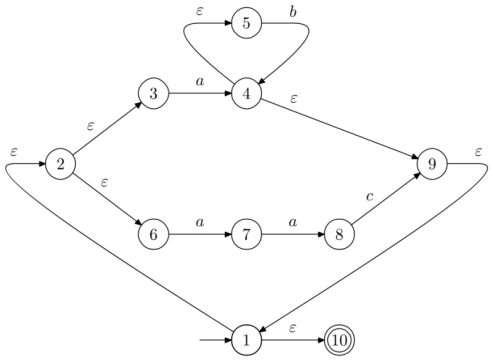


Figure 3: Full translation to NFA of regular expression in Problem 4.

a, b, and c are

$\text{move}(\mathcal{A}, a) = \varepsilon\text{-closure}(\{4, 7\}) = \{1, 2, 3, 4, 5, 6, 7, 9, 10\} = \mathcal{B}$ [**new state**]
$\text{move}(\mathcal{A}, b) = \emptyset$ [no transition possible]
$\text{move}(\mathcal{A}, c) = \emptyset$ [no transition possible]

We consider next the new state $\mathcal{B}$, for which we get the transitions

$\text{move}(\mathcal{B}, a) = \varepsilon\text{-closure}(\{4, 7, 8\}) = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\} = \mathcal{C}$ [**new state**]
$\text{move}(\mathcal{B}, b) = \varepsilon\text{-closure}(\{4\}) = \{1, 2, 3, 4, 5, 6, 9, 10\} = \mathcal{D}$ [**new state**]
$\text{move}(\mathcal{B}, c) = \emptyset$ [no transition possible]

Moving on to state $\mathcal{C}$ we obtain

$\text{move}(\mathcal{C}, a) = \varepsilon\text{-closure}(\{4, 7, 8\}) = \mathcal{C}$
$\text{move}(\mathcal{C}, b) = \varepsilon\text{-closure}(\{4\}) = \mathcal{D}$
$\text{move}(\mathcal{C}, c) = \varepsilon\text{-closure}(\{9\}) = \{1, 2, 3, 6, 9, 10\} = \mathcal{E}$ [**new state**]

and for state $\mathcal{D}$ we derive

$\text{move}(\mathcal{D}, a) = \varepsilon\text{-closure}(\{4, 7\}) = \mathcal{B}$
$\text{move}(\mathcal{D}, b) = \varepsilon\text{-closure}(\{4\}) = \mathcal{D}$
$\text{move}(\mathcal{D}, c) = \emptyset$ [no transition possible]

Finally, for state $\mathcal{E}$ we can observe that it is identical to $\mathcal{A}$ except for the added NFA state 9, the only transition from which is an ε-transition to 1. We therefore should get the same transitions as for state $\mathcal{A}$, and indeed we can compute that

$$\begin{aligned}\text{move}(\mathcal{E}, a) &= \mathcal{B} \\ \text{move}(\mathcal{E}, b) &= \emptyset \\ \text{move}(\mathcal{E}, c) &= \emptyset\end{aligned}$$

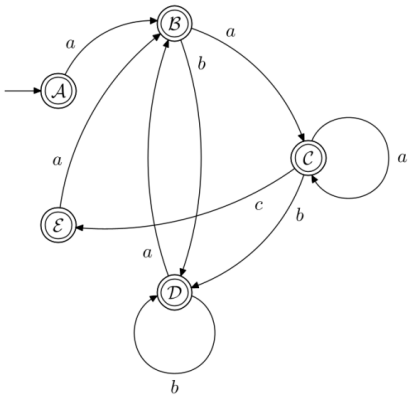


Figure 4: Conversion of NFA in Figure 3 to DFA.

First, follow, nullable

LL(1)

Miscellaneous

Logic

p	q	q ∧ q(and)	p ∨ q(or)	~p(not)	p→q (implies)	p ⇔ q (equivalence)
F	F	F	F	T	T	T
F	T	F	T	T	T	F
T	F	F	T	F	F	F
T	T	T	T	F	T	T

P	Q	R
T	T	T
T	T	F
T	F	T
T	F	F
F	T	T
F	T	F
F	F	T
F	F	F

A	B	C	D
T	T	T	T
T	T	T	F
T	T	F	T
T	T	F	F
T	F	T	T
T	F	T	F
T	F	F	T
T	F	F	F
F	T	T	T
F	T	T	F
F	T	F	T
F	T	F	F
F	F	T	T
F	F	T	F
F	F	F	T
F	F	F	F

Combinatorics

	With repetition	Without repetition
Ordered	Sequence	Permutation
Unordered	Multiset	Set

	With rep	Without rep
Ordered	n^r	$\frac{n!}{(n-r)!}$
Unordered	$\binom{n+r-1}{r}$	$\frac{n!}{r!\,(n-r)!}$

Poker hands: (Remember to divide by 52 choose 5 for probability)

Hand	Distinct hands	Frequency	Probability	Cumulative probability	Odds against	Mathematical expression of absolute frequency
Royal flush 	1	4	0.000154%	0.000154%	649,739 : 1	$\binom{4}{1}$
Straight flush (excluding royal flush) 	9	36	0.00139%	0.0015%	72,192.33 : 1	$\binom{10}{1}\binom{4}{1} - \binom{4}{1}$
Four of a kind 	156	624	0.02401%	0.0256%	4,164 : 1	$\binom{13}{1}\binom{4}{4}\binom{12}{1}\binom{4}{1}$
Full house 	156	3,744	0.1441%	0.17%	693.1667 : 1	$\binom{13}{1}\binom{4}{3}\binom{12}{1}\binom{4}{2}$
Flush (excluding royal flush and straight flush) 	1,277	5,108	0.1965%	0.367%	508.8019 : 1	$\binom{13}{5}\binom{4}{1} - \binom{10}{1}\binom{4}{1}$
Straight (excluding royal flush and straight flush) 	10	10,200	0.3925%	0.76%	253.8 : 1	$\binom{10}{1}\binom{4}{1}^5 - \binom{10}{1}\binom{4}{1}$
Three of a kind 	858	54,912	2.1128%	2.87%	46.32955 : 1	$\binom{13}{1}\binom{4}{3}\binom{12}{2}\binom{4}{1}^2$
Two pair 	858	123,552	4.7539%	7.62%	20.03535 : 1	$\binom{13}{2}\binom{4}{2}^2\binom{11}{1}\binom{4}{1}$
One pair 	2,860	1,098,240	42.2569%	49.9%	2.366477 : 1	$\binom{13}{1}\binom{4}{2}\binom{12}{3}\binom{4}{1}^3$
No pair / High card 	1,277	1,302,540	50.1177%	100%	0.9953015 : 1	$\left[\binom{13}{5} - \binom{10}{1}\right]\left[\binom{4}{1}^5 - \binom{4}{1}\right]$
Total	7,462	2,598,960	100%	---	0 : 1	$\binom{52}{5}$

Induction

Method:

1. Base case
    a. n=1
2. Hypothesis
3. Induction step (n+1)
    a. tip: try and insert hypothesis

proof of correctness

- Find invariant (the part of the list which is always correct)