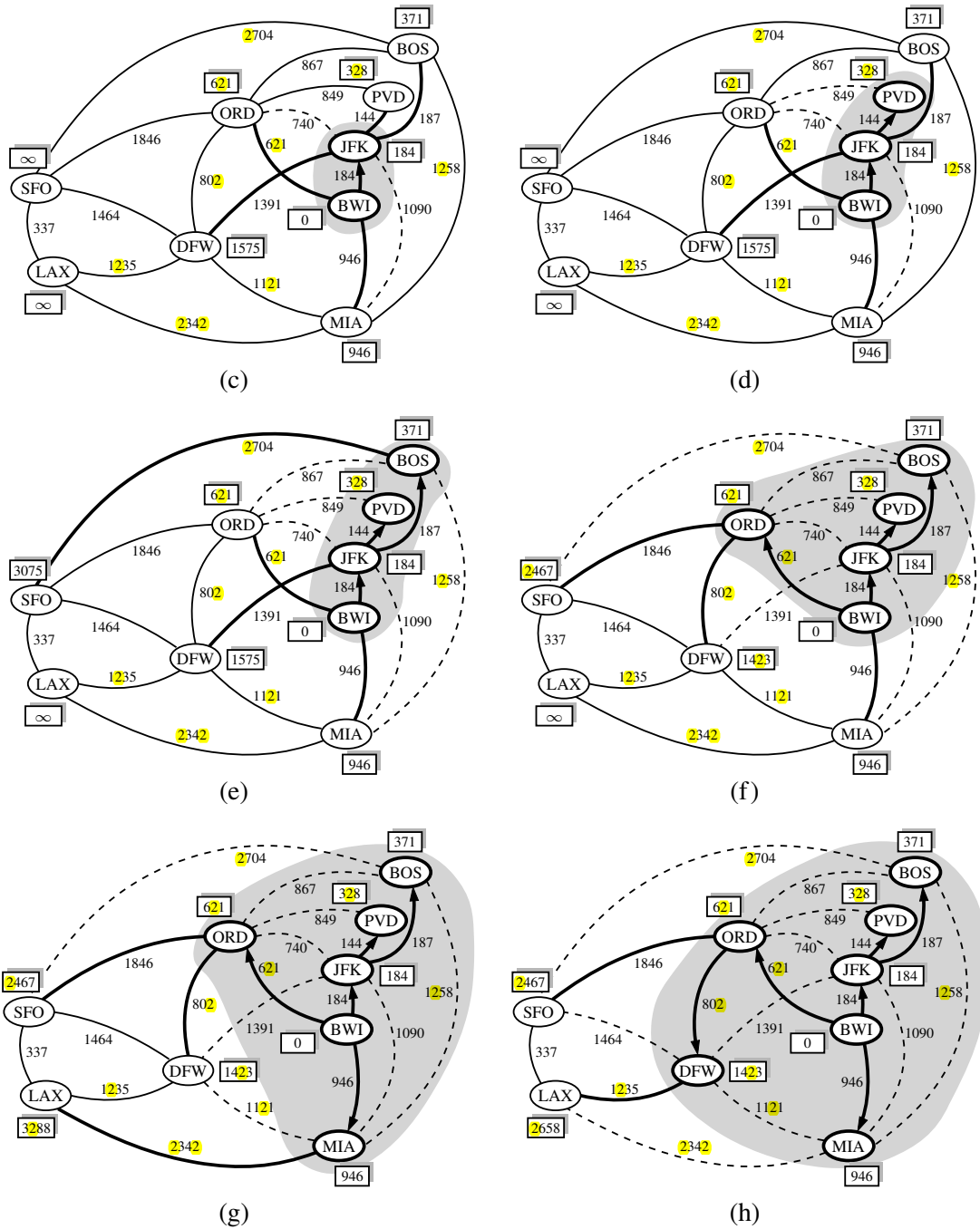


- abc module, 60, 93, 306
- Abelson, Hal, 182
- abs function, 29, 75
- abstract base class, 60, 93–95, 306, 317, 406
- abstract data type, v, 59
  - deque, 247–248
  - graph, 620–626
  - map, 402–408
  - partition, 681–684
  - positional list, 279–281
  - priority queue, 364
  - queue, 240
  - sorted map, 427
  - stack, 230–231
  - tree, 305–306
- abstraction, 58–60
- $(a, b)$  tree, 712–714
- access frequency, 286
- accessors, 6
- activation record, 23, 151, 703
- actual parameter, 24
- acyclic graph, 623
- adaptability, 57, 58
- adaptable priority queue, 390–395, 666, 667
- AdaptableHeapPriorityQueue class, 392–394, 667
- adapter design pattern, 231
- Adel'son-Vel'skii, Georgii, 481, 535
- adjacency list, 627, 630–631
- adjacency map, 627, 632, 634
- adjacency matrix, 627, 633
- ADT, *see* abstract data type
- Aggarwal, Alok, 719
- Aho, Alfred, 254, 298, 535, 618
- Ahuja, Ravindra, 696
- algorithm, 110
- algorithm analysis, 123–136
  - average-case, 114
  - worst-case, 114
- alias, 5, 12, 101, 189
- all function, 29
- alphabet, 583
- amortization, 164, 197–200, 234, 237, 246, 376, 681–684
- ancestor, 302
- and operator, 12
- any function, 29
- arc, 620
- arithmetic operators, 13
- arithmetic progression, 89, 199–200
- ArithmeticError, 83, 303
- array, 9, 183–222, 223, 227
  - compact, 190, 711
  - dynamic, 192–201, 246
- array module, 191
- ArrayQueue class, 242–246, 248, 292, 306
- ASCII, 721
- assignment statement, 4, 24
  - chained, 17
  - extended, 16
  - simultaneous, 45, 91
- asymptotic notation, 123–127, 136
  - big-Oh, 123–127
  - big-Omega, 127, 197
  - big-Theta, 127
- AttributeError, 33, 100
- AVL tree, 481–488
  - balance factor, 531
  - height-balance property, 481
- back edge, 647, 689
- backslash character, 3
- Baeza-Yates, Ricardo, 535, 580, 618, 719
- Barůvka, Otakar, 693, 696
- base class, 82
- BaseException, 83, 303
- Bayer, Rudolf, 535, 719
- Beazley, David, 55
- Bellman, Richard, 618
- Bentley, Jon, 182, 400, 580
- best-fit algorithm, 699
- BFS, *see* breadth-first search
- biconnected graph, 690
- big-Oh notation, 123–127
- big-Omega notation, 127, 197
- big-Theta notation, 127
- binary heap, 370–384
- binary recursion, 174
- binary search, 155–156, 162–163, 428–433, 571
- binary search tree, 332, 460–479



**Figure 14.16:** An example execution of Dijkstra's algorithm. (Continued from Figure 14.15; continued in Figure 14.17.)

- Cormen, Thomas, 535, 696
- Counter class, 450
- CPU, 111
- CRC cards, 63
- CreditCard class, 63, **69–73**, 73, 83–86
- Crochemore, Maxime, 618
- cryptography, **216–218**
- ctypes module, 191, 195
- cubic function, 119
- cyber-dollar, 197–199, 497–500, **682**
- cycle, **623**
  - directed, **623**
- cyclic-shift hash code, 413–414
- DAG, *see* directed acyclic graph
- data packets, **227**
- data structure, 110
- Dawson, Michael, 55
- debugging, **62**
- decision tree, 311, 463, **562**
- decorate-sort-undecorate design pattern, 570
- decrease-and-conquer, 571
- decryption, **216**
- deep copy, **102**, 188
- deepcopy function, **102**, 188
- def keyword, **23**
- degree of a vertex, **621**
- del operator, 15, 75
- DeMorgan's Law, 137
- Demurjian, Steven, 108, **254**
- depth of a tree, 308–310
- depth-first search (DFS), 639–647
- deque, **247–249**
  - abstract data type, **247–248**
  - linked-list implementation, **249**, **275**
- deque class, **249**, **251**
- descendant, **302**
- design patterns, v, 61
  - adapter, **231**
  - amortization, 197–**200**
  - brute force, 584
  - composition, **287**, 365, 407
  - divide-and-conquer, 538–**542**, 550–551
  - dynamic programming, 594–600
  - factory method, 479
  - greedy method, 603
  - position, **279–281**
  - prune-and-search, 571–573
  - template method, 93, **342**, 406, 448, 478
- DFS, *see* depth-first search
- Di Battista, Giuseppe, 361, 696
- diameter, 358
- dict class, 7, 11, **402**
- dictionary, 11, 16, **402–408**, *see also* map
- dictionary comprehension, 43
- Dijkstra's algorithm, 661–669
- Dijkstra, Edsger, **182**, 696
- dir function, 46
- directed acyclic graph, 655–657
- disk usage, 157–160, 163–164, 340
- divide-and-conquer, 538–**542**, 550–551
- division method for hash codes, 416
- documentation, 66
- double hashing, 419
- double-ended queue, *see* deque
- doubly linked list, **260**, **270–276**
  - \_DoublyLinkedBase class, **273–275**, **278**
- down-heap bubbling, 374
- duck typing, 60, 306
- dynamic array, 192–**201**, **246**
  - shrinking, **200**, **246**
- DynamicArray class, **195–196**, **204**, **206**, **224**, **225**, **245**
- dynamic binding, 100
- dynamic dispatch, 100
- dynamic programming, 594–600
- dynamically typed, 5
- Eades, Peter, 361, 696
- edge, **302**, **620**
  - destination, **621**
  - endpoint, **621**
  - incident, **621**
  - multiple, **622**
  - origin, **621**
  - outgoing, **621**
  - parallel, **622**
  - self-loop, **622**
- edge list, **627–629**
- edge relaxation, 661
- edit distance, 616

- Karger, David, 696
- Karp, Richard, 361
- KeyboardInterrupt, 33, 83, 303
- KeyError, 33, 34, 83, 303, 403, 404, 422, 460
- keyword parameter, 27
- Klein, Philip, 696
- Kleinberg, Jon, 580
- Knuth, Donald, 147, 227, 298, 361, 400, 458, 535, 580, 618, 696, 719
- Knuth-Morris-Pratt algorithm, 590–593
- Kosaraju, S. Rao, 696
- Kruskal’s algorithm, 676–684
- Kruskal, Joseph, 696
  
- L’Hôpital’s rule, 731
- Landis, Evgenii, 481, 535
- Langston, Michael, 580
- last-in, first-out (LIFO), 229
- lazy evaluation, 39, 80
- LCS, *see* longest common subsequence
- leaves, 302
- Lecroq, Thierry, 618
- Leiserson, Charles, 535, 696
- len function, 29
- Lesuisse, R., 182
- Letscher, David, 55, 108
- level in a tree, 315
- level numbering, 325, 371
- lexicographic order, 15, 203, 385, 565
- LIFO, 229
- linear exponential, 728
- linear function, 117
- linear probing, 418
- linearity of expectation, 573, 730
- linked list, 256–293
  - doubly linked, 260, 270–276, 281
  - singly linked, 256–260
- linked structure, 317
- LinkedBinaryTree class, 303, 318–324, 335, 348
- LinkedDeque class, 275–276
- LinkedQueue class, 264–265, 271, 306, 335
- LinkedStack class, 261–263
- Lins, Rafael, 719
- Liotta, Giuseppe, 361, 696
  
- Liskov, Barbara, 108, 254, 298
- list
  - of favorites, 286–291
  - positional, 277–285
- list class, 7, 9, 202–207
  - sort method, 23, 569
- list comprehension, 43, 207, 209, 221
- literal, 6
- Littman, Michael, 580
- live objects, 700
- load factor, 417, 420–421
- local scope, 23–25, 46, 96
- locality of reference, 289, 707
- locator, 390
- log-star function, 684
- logarithm function, 115–116, 725
- logical operators, 12
- longest common subsequence, 597–600
- looking-glass heuristic, 586
- lookup table, 410
- LookupError, 83, 303
- loop invariant, 140
- lowest common ancestor, 358
- Lutz, Mark, 55
  
- Magnanti, Thomas, 696
- main memory, 705
- map
  - abstract data type, 402–408
  - AVL tree, 481–488
  - binary search tree, 460–479
  - hash table, 410–426
  - red-black tree, 512–525
  - skip list, 437–445
  - sorted, 460
  - (2,4) tree, 502–511
  - update operations, 442, 465, 466, 483, 486
- MapBase class, 407–408
- Mapping abstract base class, 406
- mark-sweep algorithm, 701, 702
- math module, 28, 49
- matrix, 219
- matrix chain-product, 594–596
- max function, 27–29
- maximal independent set, 692
- McCreight, Edward, 618, 719

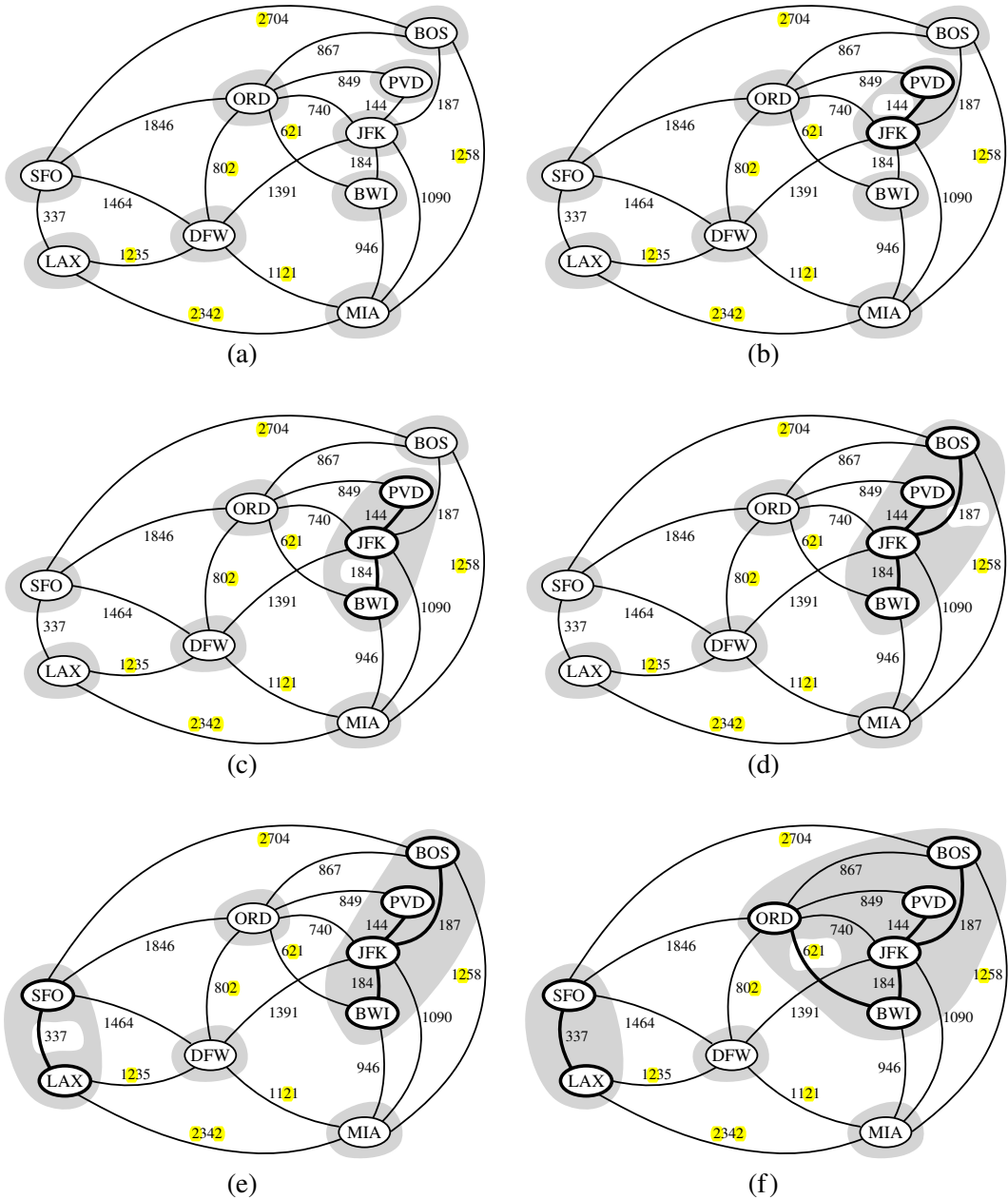
- element uniqueness problem, 135–136, 165
- elif keyword, 18
- Empty exception class, 232, 242, 303
- encapsulation, 58, 60
- encryption, 216
- endpoints, 621
- EOFError, 33, 37, 38
- escape character, 10
- Euclidean norm, 53
- Euler tour of a graph, 686, 691
- Euler tour tree traversal, 341–347, 361
- EulerTour class, 342–345
- event, 729
- except statement, 36–38
- exception, 33–38, 83
  - catching, 36–38
  - raising, 34–35
- Exception class, 33, 83, 232, 303
- expected value, 729
- exponential function, 120–121, 172–173
- expression tree, 312, 348–351
- expressions, 12–17
- ExpressionTree class, 348–351
- external memory, 705–716, 719
- external-memory algorithm, 705–716
- external-memory sorting, 715–716
  
- factorial function, 150–151, 161, 166–167, 726
- factoring a number, 40–41
- factory method pattern, 479
- False, 7
- favorites list, 286–291
- FavoritesList class, 287–288
- FavoritesListMTF class, 290, 399
- Fibonacci heap, 667
- Fibonacci series, 41, 45, 90–91, 727
- FIFO, 239, 363
- file proxy, 31–32
- file system, 157–160, 302, 340
- finally, 38
- first-class object, 47
- first-fit algorithm, 699
- first-in, first-out (FIFO), 239, 363
- Flajolet, Philippe, 147
- float class, 7, 8
- floor function, 122, 172, 726
  
- flowchart, 19
- Floyd, Robert, 400, 696
- Floyd-Warshall algorithm, 652–654, 696
- for loop, 21
- forest, 623
- formal parameter, 24
- fractal, 152
- fragmentation of memory, 699
- free list, 699
- frozenset class, 7, 11, 446
- full binary tree, 311
- function, 23–28
  - body, 23
  - built-in, 28
  - signature, 23
  
- game tree, 330, 361
- GameEntry class, 210
- Gamma, Erich, 108
- garbage collection, 209, 245, 275, 700–702
  - mark-sweep, 701, 702
- Gauss, Carl, 118
- gc module, 701
- generator, 40–41, 79
- generator comprehension, 43, 209
- geometric progression, 90, 199
- geometric sum, 121, 728
- getsizeof function, 190, 192–194
- global scope, 46, 96
- Goldberg, Adele, 298
- Goldwasser, Michael, 55, 108
- Gonnet, Gaston, 400, 535, 580, 719
- Goodrich, Michael, 719
- grade-point average (GPA), 3, 26
- Graham, Ronald, 696
- graph, 620–696
  - abstract data type, 620–626
  - acyclic, 623
  - breadth-first search, 648–650
  - connected, 623, 638
  - data structures, 627–634
    - adjacency list, 627, 630–631
    - adjacency map, 627, 632, 634
    - adjacency matrix, 627, 633
    - edge list, 627–629
  - depth-first search, 639–647

- tree, 669
- shuffle function, 50, 225
- sieve algorithm, 454
- signature, 23
- simultaneous assignment, 45, 91
- singly linked list, 256–260
- skip list, 437–445
  - analysis, 443–445
  - insertion, 440
  - removal, 442–443
  - searching, 439–440
  - update operations, 440–443
- Sleator, Daniel, 535
- slicing notation, 14–15, 188, 203, 583
- sorted function, 6, 23, 28, 29, 136, 537, 569
- sorted map, 427–436
  - abstract data type, 427
  - search table, 428–433
- SortedPriorityQueue class, 368–369
- SortedTableMap class, 429–433
- sorting, 214, 385–386, 537–566
  - bubble-sort, 297
  - bucket-sort, 564–565
  - external-memory, 715–716
  - heap-sort, 384, 388–389
  - in-place, 389, 559
  - insertion-sort, 214–215, 285, 387
  - key, 385
  - lower bound, 562–563
  - merge-sort, 538–550
  - priority-queue, 385–386
  - quick-sort, 550–561
  - radix-sort, 565–566
  - selection-sort, 386–387
  - stable, 565
  - Tim-sort, 568
- source code, 2
- space usage, 110
- spanning tree, 623, 638, 642, 643, 670
- sparse array, 298
- splay tree, 478, 490–501
- split function of str class, 724
- stable sorting, 565
- stack, 229–238
  - abstract data type, 230–231
  - array implementation, 231–234
  - linked-list implementation, 261–263
- Stein, Clifford, 535, 696
- Stephen, Graham, 618
- Stirling’s approximation, 727
- stop words, 606, 617
- StopIteration, 33, 39, 41, 79
- str class, 7, 9, 10, 208–210, 721–724
- strict weak order, 385
- string, *see also* str class
  - null, 583
  - prefix, 583
  - suffix, 583
- strongly connected components, 646
- strongly connected graph, 623
- subclass, 82
- subgraph, 623
- subproblem overlap, 597
- subsequence, 597
- subtree, 302
- suffix, 583
- sum function, 29, 35
- summation, 120
  - geometric, 121, 728
  - telescoping, 727
- Summerfield, Mark, 55
- super function, 84
- superclass, 82
- Sussman, Gerald, 182
- Sussman, Julie, 182
- sys module, 49, 190, 192, 701
- SystemExit, 83, 303
  
- Tamassia, Roberto, 361, 696
- Tardos, Éva, 580
- Tarjan, Robert, 361, 535, 696
- telescoping sum, 727
- template method pattern, 93, 342, 406, 448, 478
- testing, 62
  - unit, 49
- text compression, 601–602
- three-way set disjointness, 134–135
- Tic-Tac-Toe, 221–223, 330, 361
- Tim-sort, 568
- time module, 49, 111
- Tollis, Ioannis, 361, 696
- topological ordering, 655–657

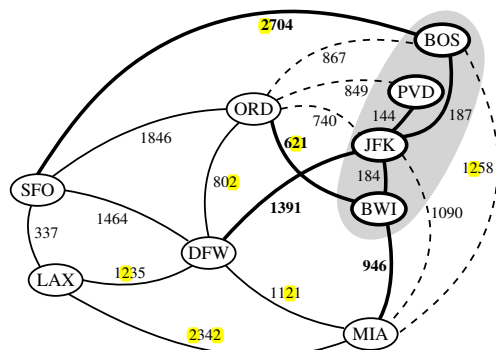
- directed, 620, 621, 657
  - acyclic, 655–657
  - strongly connected, 623
- mixed, 621
- reachability, 651–654
- shortest paths, 654
- simple, 622
- traversal, 638–650
- undirected, 620, 621
- weighted, 659–696
- greedy method, 603, 660, 661
- Guibas, Leonidas, 535
- Guttag, John, 108, 254, 298
- Harmonic number, 131, 180, 728
- hash code, 411–415
  - cyclic-shift, 413–414
  - polynomial, 413
- hash function, 415
- hash table, 410–426
  - clustering, 419
  - collision, 411
  - collision resolution, 417–419
  - double hashing, 419
  - linear probing, 418
  - quadratic probing, 419
- HashMapBase class, 422–423
- header sentinel, 270
- heap, 370–384
  - bottom-up construction, 380–384
- heap-sort, 384, 388–389
- HeapPriorityQueue class, 377–378, 382
- heapq module, 384
- height of a tree, 309–310, 474
- height-balance property, 481, 483
- Hell, Pavol, 696
- Hennessy, John, 719
- heuristic, 289
- hierarchy, 82
- Hoare, C. A. R., 580
- hook, 342, 468, 478
- Hopcroft, John, 254, 298, 535, 696
- Hopper, Grace, 36
- Horner’s method, 146
- HTML, 236–238, 251, 582
- Huang, Bing-Chao, 580
- Huffman coding, 601–602
- I/O complexity, 711
- id function, 29
- identifier, 4
- IDLE, 2
- immutable type, 7, 11, 415
- implied method, 76
- import statement, 48
- in operator, 14–15, 75
- in-degree, 621
- in-place algorithm, 389, 396, 559, 702
- incidence collection, 630
- incident, 621
- incoming edges, 621
- independent, 729, 730
- index, 186
  - negative, 14
  - zero-indexing, 9, 14
- IndexError, 20, 33, 34, 83, 232, 303
- induction, 138–139, 162
- infix notation, 359
- inheritance, 82–95
  - multiple, 468
- inorder tree traversal, 331, 335–336, 461, 476
- input function, 29, 30–31
- insertion-sort, 214–215, 285–286, 387
- instance, 57
- instantiation, 6
- int class, 7, 8
- integrated development environment, 2
- internal memory, 705
- Internet, 227
- interpreter, 2
- inversion, 567, 578
- inverted file, 456
- IOError, 33, 37
- is not operator, 12
- is operator, 12, 76
- isinstance function, 29, 34
- isomorphism, 355
- iterable type, 9, 21, 35, 39
- iterator, 39–40, 76, 79, 87
- JáJá, Joseph, 361
- Jarník, Vojtěch, 696
- join function of str class, 723
- Jones, Richard, 719

6.2	<b>Queues</b>	239
6.2.1	The Queue Abstract Data Type	240
6.2.2	Array-Based Queue Implementation	241
6.3	<b>Double-Ended Queues</b>	247
6.3.1	The Deque Abstract Data Type	247
6.3.2	Implementing a Deque with a Circular Array	248
6.3.3	Dequeues in the Python Collections Module	249
6.4	<b>Exercises</b>	250
7	<b>Linked Lists</b>	255
7.1	<b>Singly Linked Lists</b>	256
7.1.1	Implementing a Stack with a Singly Linked List	261
7.1.2	Implementing a Queue with a Singly Linked List	264
7.2	<b>Circularly Linked Lists</b>	266
7.2.1	Round-Robin Schedulers	267
7.2.2	Implementing a Queue with a Circularly Linked List	268
7.3	<b>Doubly Linked Lists</b>	270
7.3.1	Basic Implementation of a Doubly Linked List	273
7.3.2	Implementing a Deque with a Doubly Linked List	275
7.4	<b>The Positional List ADT</b>	277
7.4.1	The Positional List Abstract Data Type	279
7.4.2	Doubly Linked List Implementation	281
7.5	<b>Sorting a Positional List</b>	285
7.6	<b>Case Study: Maintaining Access Frequencies</b>	286
7.6.1	Using a Sorted List	286
7.6.2	Using a List with the Move-to-Front Heuristic	289
7.7	<b>Link-Based vs. Array-Based Sequences</b>	292
7.8	<b>Exercises</b>	294
8	<b>Trees</b>	299
8.1	<b>General Trees</b>	300
8.1.1	Tree Definitions and Properties	301
8.1.2	The Tree Abstract Data Type	305
8.1.3	Computing Depth and Height	308
8.2	<b>Binary Trees</b>	311
8.2.1	The Binary Tree Abstract Data Type	313
8.2.2	Properties of Binary Trees	315
8.3	<b>Implementing Trees</b>	317
8.3.1	Linked Structure for Binary Trees	317
8.3.2	Array-Based Representation of a Binary Tree	325
8.3.3	Linked Structure for General Trees	327
8.4	<b>Tree Traversal Algorithms</b>	328

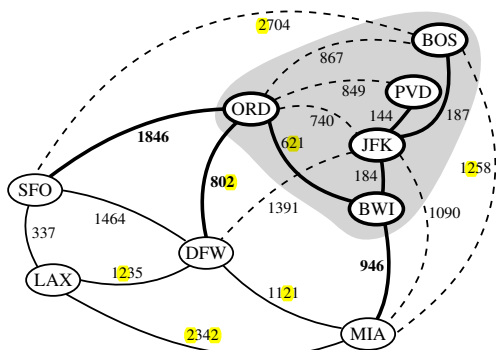




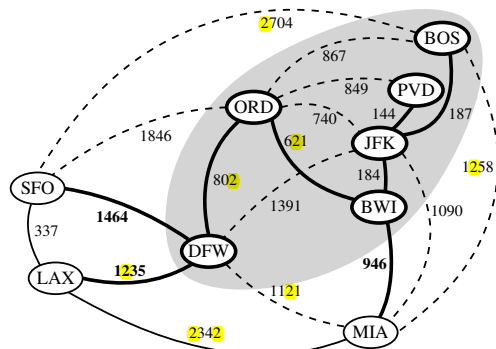
**Figure 14.22:** Example of an execution of Kruskal's MST algorithm on a graph with integer weights. We show the clusters as shaded regions and we highlight the edge being considered in each iteration. (Continues in Figure 14.23.)



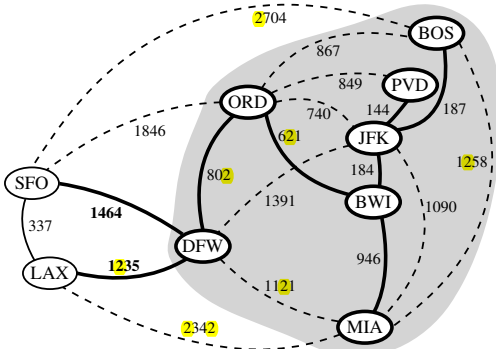
(e)



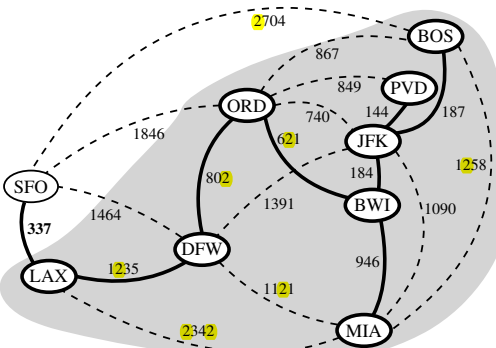
(f)



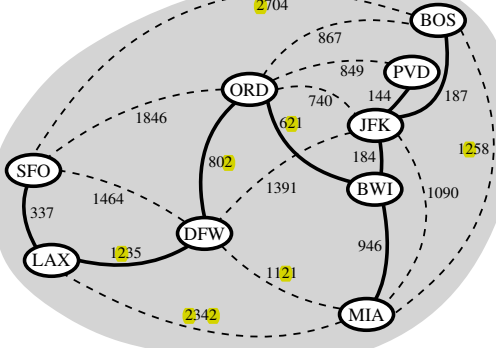
(g)



(h)



(i)



(j)

**Figure 14.21:** An illustration of the Prim-Jarník MST algorithm. (Continued from Figure 14.20.)